

On Abstraction of Probabilistic Systems

Christian Dehnert¹, Daniel Gebler², Michele Volpato³, and David N. Jansen³

¹ Software Modeling and Verification Group,
RWTH Aachen University, Ahornstraße 55, 52056 Aachen, Germany

² Department of Computer Science, VU University Amsterdam,
De Boelelaan 1081a, 1081 HV Amsterdam, The Netherlands

³ Institute for Computing and Information Sciences,
Radboud University, Heyendaalseweg 135, 6500 GL Nijmegen, The Netherlands

Abstract. Probabilistic model checking extends traditional model checking by incorporating quantitative information about the probability of system transitions. However, probabilistic models that describe interesting behavior are often too complex for straightforward analysis. Abstraction is one way to deal with this complexity: instead of analyzing the (“concrete”) model, a simpler (“abstract”) model that preserves the relevant properties is built and analyzed. This paper surveys various abstraction techniques proposed in the past decade. For each abstraction technique we identify in what sense properties are preserved or provide alternatively suitable boundaries.

1 Introduction

The advent of large-scale, distributed, dependable systems requires formal specification and verification methods which capture both *qualitative* and *quantitative* properties of systems. Performance and dependability evaluation of distributed systems therefore demands to use formal models and methods where both aspects are represented. Labeled transition systems (LTS) allow to capture qualitative (functional) aspects of software and hardware systems. To model the mentioned quantitative phenomena, one uses a probabilistic formalism, typically some extension of Markov chains. For instance, exchanging messages between distributed systems typically suffers from a failure probability. Hence, interesting properties of real systems often express that some functional behavior can be guaranteed to happen with at least some given probability or, dually, some bad behavior appears with at most some given probability. Probabilistic models, such as Markov chains and Markov decision processes, allow to model and reason over both qualitative (functional) and quantitative (non-functional) aspects.

The properties of probabilistic systems are typically specified in temporal logics such as the probabilistic computation tree logic (PCTL) [25, 6]. Model checking is a method to verify those properties. However, it suffers from the state space explosion problem, which means that the number of reachable states

of the model under investigation is too large. In models specified compositionally, it is often exponential in the number of components of the model. Additionally, probabilistic model checking relies on expensive numerical methods, making the problem even more pressing. Consequently, it is generally of crucial importance to simplify the model prior to verification. However, given the complexity of typical models, this procedure needs to be both automated and efficient.

In this paper, we present selected *abstraction* techniques for probabilistic systems. Intuitively, abstraction removes details from concrete models that are not relevant to the property of interest. In many cases, only abstraction makes the analysis of the model feasible or at least speeds up verification considerably. Verification of realistic models requires the application of aggressive abstraction techniques (c.f. [10]). We present the following abstraction techniques:

Multi-valued abstraction allows to partition the state space and to abstract transition probabilities by intervals. Both positive and negative verification results in the abstract model carry over to the concrete model. However, in the abstract model some properties may evaluate to “unknown” if the abstract model does not allow a conclusive evaluation of the property.

Counterexample-guided abstraction refinement (CEGAR) uses counterexamples for the abstract model obtained from a model checker to refine the abstraction. By (dis)proving realizability of the abstract counterexample in the concrete model, this allows for an automatic abstraction-refinement technique that proves or refutes properties. We survey the probabilistic CEGAR algorithm introduced by [27] and [45, Section 7].

Game-based abstraction provides the means to abstraction while maintaining the separation between nondeterminism present in the concrete model and nondeterminism introduced by the abstraction. We present probabilistic game-based and menu-based abstraction, which employ two-player games where opponent and defender take different roles in resolving the nondeterminism. The resulting game allows to give distinct upper and lower bounds on reachability properties. This interval can also be understood as a measure of the quality of the abstraction.

Organization of the paper. Section 2 provides a survey on the available literature and the tools for the discussed subject. Section 3 introduces the formal framework, i.e. probabilistic models, probabilistic temporal logics and probabilistic games. Multi-valued abstraction is covered in Section 4. Probabilistic CEGAR is surveyed in Section 5 and, finally, game-based abstraction techniques are presented in Section 6. We summarize and conclude in Section 7.

2 Related Work

2.1 Literature

Abstraction is of immense importance for the analysis of large probabilistic systems. Consequently, the field has been studied extensively. One of the most popular techniques is *bisimulation minimization* [4]. Here, the states of the abstract

system represent equivalence classes of an equivalence relation on the states, a bisimulation, such that the abstract system is guaranteed to preserve certain properties. [18] investigates several kinds of strong and weak bisimulations regarding the minimality of the quotient system with respect to the number of states, the number of transitions and transition fan-out. In [32], the authors show that (strong) bisimulation can, in practice, lead to significant savings in memory and runtime of explicit state probabilistic model checking. Approaches that compute the bisimulation quotients on a symbolic representation of the abstract state space are proposed in both [47] and [14], where the former uses multi-terminal binary decision diagrams and the latter focuses on a representation of the state space in terms of predicates. [37, 17] compute a bisimulation based on symmetry in the models that is easier to compute than strong bisimulation, but may produce larger quotients. [15] proposes an abstraction technique for probabilistic automata based on may and must modalities inspired by modal transition systems [41]. [12, 13] pioneered the use of an abstraction-refinement approach for probabilistic systems that tries to prove a reachability property on a very coarse abstraction of the system. If the verification fails, the system is successively refined until a conclusive answer can be given. While probabilistic *CEGAR* [27, 8] uses counterexamples obtained from a (probabilistic) model checker to refine the abstract system, the game-based techniques [35, 45] typically rely on disagreeing strategies for the individual players to make the abstraction more precise when required. *Magnifying-lens abstraction* [1] uses a similar scheme, but rather considers the concrete states contained in an abstract state in each step and thus “magnifies” the state.

In infinite state probabilistic models, typically almost the whole probability mass is concentrated in a finite subset of the states. *Sliding-window abstraction* [26] is a technique to abstract from an infinite state space by “hiding” irrelevant states (in the sense that they possess a negligible amount of probability mass) in a way similar to the view through a window. Over time, as different states become relevant, the window slides to different areas of the state space.

Often, probabilistic models arise from the parallel composition of several components. *Assume-guarantee* verification [36, 21, 39] aims at proving a property of the composed model without actually building a representation of the full model by verifying the components in isolation. As the interaction between the components is typically essential to prove a given property, these techniques try to create small assumptions that can be proven on one component and suffice to establish the property on the other components. Note that some of the aforementioned techniques, for example bisimulation minimization, are also compositional in the sense that they can be applied to the individual components that are further subject to parallel composition.

Abstraction and refinement are closely related to *simulation* relations. A simulation relation is a relation between two models that shows a form of weak preservation: all properties expressible as positive formulas are preserved. In a probabilistic context, one usually chooses a *liveness* view on simulation: a probabilistic liveness property is a lower bound on the probability of some (good)

behavior. One wants the concrete model to be at least as good as the abstract one, so every liveness property ensured by the abstract model should also hold in the concrete one. A good simulation relation shows a form of weak preservation, i. e., all liveness properties in some suitable logic are preserved. Simulation relations for probabilistic systems have been studied for systems without [5, 29] and with nondeterminism [43, 50]. Work in this area also explores systems with continuous state spaces and how such state spaces can be approximated by a finite Markov model [16].

2.2 Tools

Tools implementing one or several of the aforementioned abstraction techniques have been developed. Such tools not only served as prototypical implementations for evaluations in the literature, but are still available and in use. SIGREF [47] is a tool implementing bisimulation minimization for systems represented as (variants of) binary decision diagrams that is, for example, applied in performance analysis of AADL models [7]. Another tool, PASS [23], employs a mixture of probabilistic CEGAR and the game-based approaches to provide lower and upper bounds for both minimal and maximal reachability probabilities. Finally, PRISM-games [9], extends the well-known probabilistic model checker PRISM [38] by an engine for probabilistic games.

3 Preliminaries

This section briefly introduces the basic notions and definitions. All material is standard and the interested reader is pointed to the original literature [4] and the referred material therein.

3.1 Markov models

Markov models are similar to transition systems in that they comprise states and transitions between these states. In discrete-time Markov chains, each state is associated with a discrete probability distribution over successor states according to which the next state is chosen. Let $Dist(S)$ be the set of *discrete probability distributions* over a set S , i. e., the set of functions $\mu: S \rightarrow [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$.

Definition 1 (Discrete-time Markov chain (DTMC)). A discrete-time Markov chain is a tuple $\mathcal{D} = (S, \mathbf{P}, s_{init}, AP, L)$ where

- S is a countable, non-empty set of states,
- $\mathbf{P}: S \times S \rightarrow [0, 1]$ is the transition probability function that assigns to each pair (s, s') of states the probability $\mathbf{P}(s, s')$ of moving from state s to s' in one step such that $\mathbf{P}(s, \cdot) \in Dist(S)$,
- $s_{init} \in S$ is the initial state,
- AP is a set of atomic propositions, and

- $L: S \rightarrow 2^{AP}$ is the labeling function that assigns a (possibly empty) set of atomic propositions $L(s) \subseteq AP$ to a state $s \in S$.

Let $\mathbf{P}(s, S) = \sum_{s' \in S} \mathbf{P}(s, s')$ be the probability to move from state s to some state $s' \in S$. A path in a DTMC is an infinite sequence of states of the form $\omega = s_1 s_2 \dots$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geq 1$. Let $Path^{\mathcal{D}}$ denote the set of paths in the DTMC \mathcal{D} and $Path_{\text{fin}}^{\mathcal{D}}$ denote the set of finite prefixes of all paths. For $\omega \in Path_{\text{fin}}^{\mathcal{D}}$, by $last(\omega)$ we refer to the last state of the finite path. A probability measure $Pr^{\mathcal{D}}$ on the set $Path^{\mathcal{D}}$ can be defined as unique extension of the measure on the respective cones [25].

As the behavior of a DTMC is purely probabilistic, it is not well suited to model concurrent systems. *Markov decision processes* add nondeterministic behavior to DTMCs by allowing (external) nondeterministic choice over probability distributions in each state.

Definition 2 (Markov decision process (MDP)). A Markov decision process is a tuple $\mathcal{M} = (S, Act, \mathbf{P}, s_{\text{init}}, AP, L)$ where

- S, s_{init}, AP and L are as for DTMCs (see Definition 1),
- Act is a finite set of actions,
- $\mathbf{P}: S \times Act \times S \rightarrow [0, 1]$ is the transition probability function that specifies the probability to move from s to s' with action $\alpha \in Act$ such that $\mathbf{P}(s, \alpha, \cdot) \in Dist(S)$ or $\mathbf{P}(s, \alpha, \cdot)$ is the constant zero function.

Let $Act(s)$ denote the set of enabled actions in state s , i.e., the actions α that satisfy $\sum_{s' \in S} \mathbf{P}(s, \alpha, s') = 1$. For simplicity, we require that the MDP has no deadlock states, i.e., $Act(s) \neq \emptyset$ for all states $s \in S$. We denote the set of distributions available at state s by $Steps(s) = \{\mathbf{P}(s, \alpha, \cdot) \in Dist(S) \mid \alpha \in Act(s)\}$. In each state s , first some enabled action $\alpha \in Act(s)$ is chosen nondeterministically. Then, the probabilistic choices given by $\mathbf{P}(s, \alpha, \cdot)$ yield the successor state s' . Thus, the set $Path^{\mathcal{M}}$ is given by all sequences of the form $\omega = s_1 \alpha_1 s_2 \alpha_2 \dots$, where $s_i \in S$ and $\alpha_i \in Act$, such that $\mathbf{P}(s_i, \alpha_i, s_{i+1}) > 0$ for all $i \geq 1$. Similarly to DTMCs, we define $Path_{\text{fin}}^{\mathcal{M}}$ as the set of finite prefixes, ending with a state, of all paths and use $last(\cdot)$ accordingly.

Schedulers provide means to resolve the nondeterministic choices of MDPs. The most general class of schedulers uses the complete trajectory up to the current state and resolves the nondeterminism to a probabilistic choice (history-dependent randomized schedulers). An important subclass are schedulers which depend only on the current state (not on the history) and which resolve the nondeterministic choice to a deterministic choice (memoryless deterministic schedulers). Formally they are given by a function $\sigma: S \rightarrow Act$ such that $\sigma(s) \in Act(s)$. They are powerful enough to reason over probabilistic reachability properties. The resolution of the nondeterministic choices in an MDP by a scheduler σ leads to an (infinite) DTMC $\mathcal{D}_{\sigma}^{\mathcal{M}} = (S^+, \mathbf{P}', s_{\text{init}}, AP, L')$ where $\mathbf{P}'(\omega, \omega s') = \mathbf{P}(last(\omega), \sigma(\omega), \omega s')$ and $L'(\omega) = L(last(\omega))$. Intuitively, the behavior of this DTMC corresponds to the behavior of the MDP under the scheduler σ . Not surprisingly, the probability measure over the infinite paths of \mathcal{M} under σ , denoted $Pr_{\sigma}^{\mathcal{M}}$, is thus given by the measure over the DTMC $\mathcal{D}_{\sigma}^{\mathcal{M}}$.

In order to express properties over these models, probabilistic extensions of common logics are used. While *probabilistic computation tree logic* (PCTL) [3] is the most prominent one, a probabilistic interpretation of linear temporal logic [44] that can enforce probability bounds on the set of paths satisfying a regular LTL formula is also popular. We will, however, focus our attention to PCTL in the further course of the paper. Formulae in this logic are given by the following grammar.

Definition 3 (Probabilistic computation tree logic (PCTL)). *The syntax of PCTL state formulae over a set of atomic propositions AP is given by the following rules:*

$$\Phi ::= true \mid a \mid \Phi \wedge \Phi \mid \neg\Phi \mid P_{\bowtie p}(\varphi)$$

where $a \in AP$, φ is a PCTL path formula, $\bowtie \in \{<, \leq, >, \geq\}$ and $p \in [0, 1]$.

PCTL path formulae are defined by the grammar:

$$\varphi ::= X\Phi \mid \Phi U \Phi \mid \Phi U^{\leq k} \Phi$$

where Φ is a state formula and $k \in \mathbb{N}$.

PCTL keeps the basic structure of CTL [19] and replaces the path quantifiers by the single new operator $P_{\bowtie p}(\varphi)$. Intuitively, this formula holds, if the probability mass of all paths in the model that satisfy φ conforms to $\bowtie p$. To improve readability, we will use the abbreviations *false* for $\neg true$ and $\Phi_1 \vee \Phi_2$ for $\neg(\neg\Phi_1 \wedge \neg\Phi_2)$. For DTMCs the interpretation of a PCTL formula is straightforward. However, for MDPs the probability mass of all paths that satisfy a given path formula depends on the resolution of nondeterminism. A PCTL formula holds in a state of an MDP if it holds for all possible schedulers. Fortunately, there is no algorithmic need to optimize over all (infinitely many) schedulers, because it can be proven that only finitely many schedulers have to be considered [6]. More specifically, memoryless schedulers attain minimal and maximal probabilities for next-step and unbounded-until path formulae whereas k step-bounded schedulers are sufficient for bounded-until formulae with time-bound k . The minimal and maximal probabilities of an MDP \mathcal{M} satisfying a given PCTL path formula φ are denoted $Pr_{min}^{\mathcal{M}}(\varphi)$ and $Pr_{max}^{\mathcal{M}}(\varphi)$, respectively:

$$Pr_{min}^{\mathcal{M}}(\varphi) = \inf_{\sigma} Pr_{\sigma}^{\mathcal{M}}(\varphi) \quad Pr_{max}^{\mathcal{M}}(\varphi) = \sup_{\sigma} Pr_{\sigma}^{\mathcal{M}}(\varphi).$$

Sometimes, we need to restrict ourselves to a fragment $PCTL_{reach}$ of PCTL that only expresses *probabilistic reachability* properties. A probabilistic reachability property P is a PCTL formula of the form $\Phi = P_{\bowtie p}(true U \Phi_F)$, which we abbreviate as $P_{\bowtie p}(\diamond\Phi_F)$, where $p \in [0, 1]$, $\bowtie \in \{<, \leq, >, \geq\}$ and Φ_F is a propositional logic formula over atomic propositions. Note that the truth value of Φ_F can be determined for each state in isolation, which is why we will treat F like an atomic proposition and assume that states are labeled accordingly.

Yet, sometimes it is necessary to further restrict this subset of PCTL to the set $PCTL_{safe}$ of *probabilistic safety* properties, which are PCTL formulae that

apply negation only to literals and only use comparison operators from $\{<, \leq\}$. While these restrictions seem very severe, many problems can be reduced to reachability problems, and safety properties allow for expressing very interesting properties of a system in practice, e. g., “the probability to reach a set of states F is less than 0.5”.

3.2 Probabilistic two-player games

While MDPs extend DTMCs with nondeterministic choices, some abstractions rely on the separation of the nondeterminism introduced in the abstraction and the nondeterminism present in the original model. Probabilistic, or stochastic, two-player games are a natural formalism for this.

Definition 4 (Probabilistic game). *A probabilistic two-player game is a tuple $\mathcal{G} = ((V, E), v_{init}, (V_1, V_2, V_p), \delta)$ where*

- (V, E) is a directed graph with edge set $E \subseteq V_1 \times V_2 \cup V_2 \times V_p \cup V_p \times V_1$,
- $v_{init} \in V_1$ is the initial vertex,
- (V_1, V_2, V_p) is a partition of V where V_1 is the set of vertices of player 1, V_2 is the set of vertices of player 2, and elements of V_p are probabilistic vertices,
- $\delta: V_p \rightarrow \text{Dist}(V_1)$ is a function that maps each probabilistic vertex to a probability distribution specifying its successor vertices such that $\delta(v_p)(v_1) > 0$ implies $(v_p, v_1) \in E$.

A play in this game is an infinite sequence $\omega = v_{1,1}v_{2,1}v_{p,1}v_{1,2}v_{2,2}v_{p,2} \dots$, where $v_{1,i} \in V_1, v_{2,i} \in V_2$ and $v_{p,i} \in V_p$, such that $(v_{1,i}, v_{2,i}), (v_{2,i}, v_{p,i}) \in E$ and $\delta(v_{p,i})(v_{1,i+1}) > 0$ for all $i \geq 1$. Let $\text{Play}^{\mathcal{G}}$ denote the set of all plays in \mathcal{G} , $\text{Play}_{\text{fin}}^{\mathcal{G}}$ the finite prefixes thereof and $\text{last}(\omega_{\text{fin}})$ refers to the last vertex of the finite prefix ω_{fin} of a play. Furthermore, $\omega(i)$ denotes the i th vertex of ω .

Intuitively, the behavior of a probabilistic game is as follows. Initially, starting in $v_{1,1} \in V_1$, player 1 nondeterministically chooses a successor vertex $v_{2,1} \in V_2$ belonging to player 2. Player 2 reacts by choosing a successor state $v_{p,1} \in V_p$. Then, the next vertex is chosen according to the probability distribution in v_p and it is again player 1’s turn. Thus, in order to resolve the nondeterminism, a scheduler is needed for each of the players. In the context of games, these are called strategies for player 1 and 2, respectively. Just like for MDPs, these strategies are functions that map a finite prefix of a play to a possible choice. Formally, strategies for the players are given by functions $\sigma_1: (V_1V_2V_p)^*V_1 \rightarrow V_2$ and $\sigma_2: (V_1V_2V_p)^*V_1V_2 \rightarrow V_p$, respectively, such that $\sigma_1(\omega v_1) = v_2$ and $\sigma_2(\omega v_1 v_2) = v_p$ implies $(v_1, v_2) \in E$ and $(v_2, v_p) \in E$, respectively. If two strategies, σ_1 for player 1 and σ_2 for player 2, are fixed, then, given a vertex v , the sets of finite and infinite plays starting in v which follow those strategies are denoted as $\text{Play}_{\text{fin}}^{\sigma_1, \sigma_2}(v)$ and $\text{Play}^{\sigma_1, \sigma_2}(v)$ respectively. In such a play, all nondeterministic choices are resolved by the strategies and the remaining behavior is purely probabilistic. Hence, a probability measure, denoted $\text{Prob}_v^{\sigma_1, \sigma_2}$, can be defined over the resulting model as in [11].

Given a set $F \subseteq V_1$ of target vertices, let

$$p_v^{\sigma_1, \sigma_2}(F) = \text{Prob}_v^{\sigma_1, \sigma_2}(\{\omega \in \text{Play}^{\sigma_1, \sigma_2}(v) \mid \exists i \in \mathbb{N}, \omega(i) \in F\})$$

be the probability for reaching a vertex in F if the game is played according to the strategies σ_1 and σ_2 . For the case where the two players play adversarially, we define the optimal reachability probabilities as

$$\begin{aligned} p_v^{+-}(F) &= \sup_{\sigma_1} \inf_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F) \\ p_v^{-+}(F) &= \inf_{\sigma_1} \sup_{\sigma_2} p_v^{\sigma_1, \sigma_2}(F). \end{aligned}$$

Accordingly, for the opposite case in which the two players cooperate, we have the optimal reachability probabilities

$$\begin{aligned} p_v^{--}(F) &= \inf_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F) \\ p_v^{++}(F) &= \sup_{\sigma_1, \sigma_2} p_v^{\sigma_1, \sigma_2}(F). \end{aligned}$$

These probabilities can be computed, e. g., using value iteration [11].

3.3 Probabilistic programs

The main motivation for abstracting models prior to verification is the hope that this will reduce the time and memory needed for verification. As building the full model is often the a time and memory consuming step in the verification procedure, applying the abstraction after that step has little potential to improve the overall performance. Hence, many successful abstraction techniques avoid building the concrete model by employing a symbolic representation. A common model for succinctly representing Markov models are probabilistic programs, which are also able to finitely represent possibly infinite Markov models. Let $BExpr_{Var}$ denote the set of boolean expressions over a set of variables Var and $b \in BExpr_{Var}$ be an element of such a set. We denote by $\llbracket b \rrbracket$ the set of all valuations of variables in Var under which b evaluates to true.

Definition 5 (Probabilistic program). A probabilistic program is a tuple $\mathcal{P} = (Var, \Sigma, s_{init}, C)$ where

- $Var = \{v_1, \dots, v_n\}$ is a finite set of variables,
- $\Sigma = \Sigma(v_1) \times \dots \times \Sigma(v_n)$ is the state space of the program, where $\Sigma(v)$ denotes the (possibly infinite) domain of the variable $v \in Var$,
- $s_{init} \in \Sigma$ is the initial state,
- C is a finite set of guarded commands of the form $c = g \rightarrow p_1 : u_1 \oplus \dots \oplus p_m : u_m$ where
 - $g \in BExpr_{Var}$ is the guard of the command,
 - probabilities $p_i \in [0, 1]$, such that $\sum_{1 \leq i \leq m} p_i = 1$,
 - update functions $u_i : \Sigma \rightarrow \Sigma$ such that $u_i \neq u_j$ for $i \neq j$.

$$\begin{aligned}
[a] \quad & x + y \leq 1 \longrightarrow 0.5 : x' = x + 1 \oplus 0.5 : x' = x + 1 \wedge y' = y + 1; \\
[b] \quad & 1 \leq x + y \leq 2 \longrightarrow 0.8 : x' = 2 \wedge y' = x - 1 \oplus 0.2 : x' = 2; \\
[c] \quad & x = 2 \longrightarrow 1 : x' = x \wedge y' = y;
\end{aligned}$$

Figure 1: A probabilistic program \mathcal{P} .

Additionally, without loss of generality, we assume that for every state $s \in \Sigma$ there is a command $c \in C$ such that $s \in \llbracket g \rrbracket$ where g is the guard of c .

Intuitively, the state space of a probabilistic program is the set of all valuations of its variables. The commands then define the probabilistic transitions between these states. A guarded command is enabled in all states satisfying its guard g , i. e., the states $s \in \llbracket g \rrbracket$, which we write as $s \models g$. If a command is enabled in some state, that state possesses an outgoing probability distribution that is given by the probabilities and the update functions. Given a command $c = g \rightarrow p_1 : u_1 \oplus \dots \oplus p_m : u_m$ and a variable valuation $s \in \Sigma$ such that $s \models g$, s has a transition with probability p_i to the state $s_i = u_i(s)$ for all $i \in \{1, \dots, m\}$.

The semantics of a probabilistic program is a DTMC or an MDP, depending on whether there exists a state that satisfies multiple guards. If there is no such state, each state has exactly one command that is enabled and the resulting model is a DTMC. If a state satisfies multiple guards, this corresponds to a nondeterministic choice between multiple commands in that particular state and, hence, the model is an MDP.

Example 1. Consider the probabilistic program \mathcal{P} depicted in Figure 1 with three commands a , b and c over two integer variables x and y with range $[0, 2]$. The semantics of this program is the MDP \mathcal{M} shown in Figure 2 where we assume that only state $\langle 2, 1 \rangle$ is of interest and thus labeled with the special atomic proposition F indicated by the double circle around the state. ■

Note that a set of predicates $\Pi = \{b_1, \dots, b_k\} \subseteq BExpr_{Var}$ induces a (finite) partitioning Q of the state space, where for $q \in Q$, $s_i \in q$ and $s_j \in q$ if and only if $s_i \models p \Leftrightarrow s_j \models p$ for all $p \in \Pi$. Stated differently, the partition is given by the sets of states that satisfy exactly the same predicates of Π . Satisfiability solvers that support richer theories, such as linear integer arithmetic, can be used to reason over probabilistic programs and build abstractions w.r.t. a set of predicates directly from such a representation [45]. That is, abstractions may be built without building the full model first. In the further course of this paper, we will only consider partitions that respect the labeling of states in the model. That is, a partition Q is viable if for each $a \in AP$ and $q \in Q$ we have that either all or no $s \in q$ are labeled with a . For a given partition Q of a set S and a probability distribution $\mu \in Dist(S)$, we denote by $\bar{\mu}$ the lifted distribution $\bar{\mu} \in Dist(Q)$ defined by $\bar{\mu}(q) = \sum_{s \in q} \mu(s)$ for all $q \in Q$.

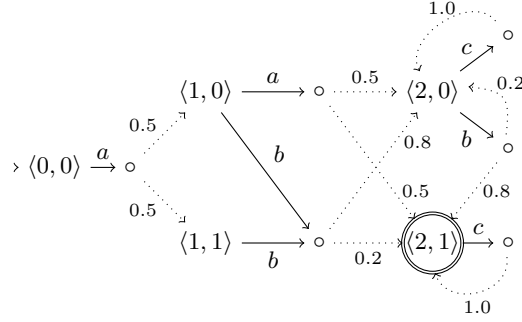


Figure 2: The MDP \mathcal{M} induced by the program \mathcal{P} .

Example 2. Reconsider the probabilistic program from Figure 1 and let the set of predicates Π be given as $\Pi = \{x < 2, x \geq 2 \wedge y \geq 1\}$. Π induces the partition

$$Q = \underbrace{\{\langle 0, 0 \rangle, \langle 1, 0 \rangle, \langle 1, 1 \rangle\}}_A \underbrace{\{\langle 2, 0 \rangle\}}_B \underbrace{\{\langle 2, 1 \rangle\}}_C$$

of the state space of \mathcal{M} , because, e. g., for all $s \in A$ we have that $s \models x < 2$ and $s \not\models x \geq 2 \wedge y \geq 1$. ■

3.4 MDP quotienting

Given an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ and a partition Q of its state space S that respects its labeling, a first idea is to construct a simple abstract system by merging the states of \mathcal{M} according to Q . A state of the abstract system, thus, corresponds to a block of Q . In order to still keep the behavior of the original system, the transition probability function must over-approximate the transition probability function of the concrete model. This can be achieved by giving an abstract state q the joint behavior of all contained states $s \in q$. Put differently, if there is a state $s \in q$ and an action $\alpha \in Act$ such that α is enabled in s and associated with the distribution $\mu \in Dist(S)$, then there must be an action α' that is enabled in q and associated with $\bar{\mu} \in Dist(Q)$. Note that this possibly involves a renaming of the action name. This is necessary, because the action α may be enabled in several states in q , but can only be associated with a single probability distribution in the quotient system. Formally, the resulting MDP is given by $\mathcal{M}/Q = (Q, Act', \mathbf{P}/Q, AP, L/Q)$ where \mathbf{P}/Q is such that $(\mathbf{P}/Q)(q, \alpha', \cdot) = \bar{\mu}$ if and only if there exists an $s \in q$ such that $\mathbf{P}(s, \alpha, \cdot) = \mu$ and $(L/Q)(q) = L(s)$ for an $s \in q$. Note that the labeling is well-defined, because of our requirement for partitions to only group states with the same labeling.

Example 3. Consider the MDP \mathcal{M} depicted in Figure 2 and the partition Q from Example 2. The quotient MDP \mathcal{M}/Q is shown in Figure 3. Note that

from block A of the partition we now have the union of all (lifted) distributions available in the states contained in A and that we preserved the labeling of the states contained in each block. Furthermore, we needed to rename the two distributions labeled with a to a_1 and a_2 , respectively.

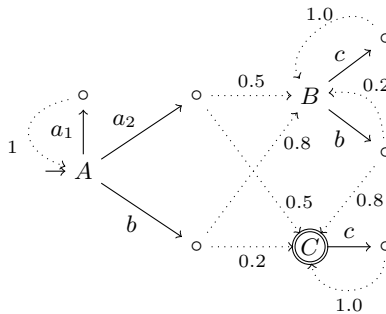


Figure 3: The MDP \mathcal{M}/Q .

As can be seen from the example, this abstraction mixes the nondeterminism present in the original model and the nondeterminism introduced by the abstraction. Consequently, the minimal probability for satisfying any given PCTL path formula in the abstract MDP \mathcal{M}/Q is a lower bound for the corresponding probability in \mathcal{M} . A similar result holds for the maximal probability. Formally, we have the following theorem.

Theorem 1. *Let \mathcal{M} be an MDP, Q be a partition of its state space and φ be a PCTL path formula, then*

$$Pr_{min}^{\mathcal{M}/Q}(\varphi) \leq Pr_{min}^{\mathcal{M}}(\varphi) \leq Pr_{max}^{\mathcal{M}}(\varphi) \leq Pr_{max}^{\mathcal{M}/Q}(\varphi)$$

Stated differently, this means that the abstraction only guarantees that the extremal probabilities for satisfying φ are in between the extremal probabilities obtained from the abstract MDP. This can lead to very coarse results as illustrated by the next example.

Example 4. For \mathcal{M}/Q of Figure 3 and $\varphi = \diamond F$, we have $Pr_{min}^{\mathcal{M}/Q}(\varphi) = 0$ and $Pr_{max}^{\mathcal{M}/Q}(\varphi) = 1$, which gives no information about the real values of $Pr_{min}^{\mathcal{M}}(\varphi)$ and $Pr_{max}^{\mathcal{M}}(\varphi)$ which can lie anywhere in between. In fact, the correct values are 0.2 and 1, respectively.

4 Multi-valued abstraction

Multi-valued abstraction aims to partition the state space combined with an abstraction of transition probabilities to sets of transition probabilities such that

both positive and negative verification results in the abstract model carry over to the concrete model [20]. Only assertions that evaluate to other values than true or false in the abstract model, typically called indefinite values, are non-conclusive for the concrete model. Hence, it is a safe (also called conservative) abstraction in the sense that if some property can be proven or disproven in the abstract model, then it carries over to the concrete model. In contrast, for MDP quotients (Section 3.4) only positive verification results carry over to the concrete model while negative verification results may occur due to over-approximation in the quotient abstraction and are not conclusive for the concrete model.

In this section we provide a survey of the three-valued abstraction technique presented in [33]. We consider abstractions of DTMCs and properties expressed by PCTL. Three-valued abstraction of models without probabilistic choice [28, 40] yields abstract models that over- and under-approximate transitions in the concrete model by *may* and *must* transitions. This concept generalizes naturally for DTMCs to transitions equipped with intervals in the abstract model where upper and lower bounds of the intervals represent accordingly the over- and under-approximation of the abstract probabilistic transitions in the concrete model. We will show that abstract states simulate the concrete states by an adapted notion of probabilistic simulation [31]. Furthermore, we demonstrate that an appropriate three-valued semantics of PCTL provides that affirmative and negative verification results on abstract DTMCs carry over to the concrete model.

As running example we will consider the DTMC presented in Figure 4a. This DTMC corresponds to the MDP of Figure 2 after abstracting from transition labels. The colour of states represents the state labelling L .

4.1 Three-valued abstraction

We start by introducing abstract DTMCs. A state in the abstract DTMC represents a set of concrete states. Transitions between abstract states are equipped with an interval of probabilities instead of a concrete probability. The lower and upper bound of the intervals represent the lowest and highest probability in the concrete model. Let \mathbb{B}_3 denote the three-valued domain with carrier $\{\perp, ?, \top\}$ and order $\perp < ? < \top$.

Definition 6 (Abstract DTMC (ADTMC)). *An abstract DTMC is a tuple $\mathcal{M} = (S, \mathbf{P}^l, \mathbf{P}^u, L, \mu_0)$ where*

- S is a countable set of states,
- $\mathbf{P}^l, \mathbf{P}^u : S \times S \rightarrow [0, 1]$ are probabilistic transition functions with
 - $\mathbf{P}^l(s, s') \leq \mathbf{P}^u(s, s')$ for all $s, s' \in S$ and
 - $\mathbf{P}^l(s, S) \leq 1 \leq \mathbf{P}^u(s, S)$,
- $L : S \times AP \rightarrow \mathbb{B}_3$ evaluates an atomic proposition for a given state and
- $\mu_0 \in \text{Dist}(S)$ is the initial distribution.

We write $s \xrightarrow{[a,b]} s'$ for $\mathbf{P}^l(s, s') = a$ and $\mathbf{P}^u(s, s') = b$. This transition may happen with any (nondeterministically chosen) probability in the interval $[a, b]$.

Furthermore, the validity of atomic propositions, denoted by L , may now evaluate to the indefinite value $?$ in \mathbb{B}_3 . Thus, ADTMCs can be described by MDPs, where the distributions reachable from s are given by $\{\mu \in \text{Dist}(S) \mid \mu(s') \in [\mathbf{P}^l(s, s'), \mathbf{P}^u(s, s')]\}$. It is clear that every DTMC is also an ADTMC if $\mathbf{P}^l(s, s') = \mathbf{P}^u(s, s')$ for all $s, s' \in S$ and $L(s, p) \in \{\top, \perp\}$ for all $s \in S, p \in AP$.

We proceed by defining the abstraction of an ADTMC based on some partitioning of its state space. Because every DTMC is also an ADTMC this directly gives us a notion to abstract a DTMC to an ADTMC.

Definition 7 (Abstraction of ADTMC). *Let $\mathcal{M} = (S, \mathbf{P}^l, \mathbf{P}^u, L, \mu_0)$ be an ADTMC and Q be a finite partitioning of S . The abstraction of \mathcal{M} with respect to Q is an ADTMC $(Q, \tilde{\mathbf{P}}^l, \tilde{\mathbf{P}}^u, \tilde{L}, \mu_0)$ such that for any $q, q' \in Q$ we have*

$$\begin{aligned} - \tilde{\mathbf{P}}^l(q, q') &= \inf_{s \in q} \mathbf{P}^l(s, q'), \\ - \tilde{\mathbf{P}}^u(q, q') &= \min(\sup_{s \in q} \mathbf{P}^u(s, q'), 1) \\ - \tilde{L}(q, a) &= \begin{cases} \top & \text{if } L(s, a) = \top \text{ for all } s \in q \\ \perp & \text{if } L(s, a) = \perp \text{ for all } s \in q \\ ? & \text{otherwise} \end{cases} \end{aligned}$$

We denote by \mathcal{M}/Q the ADTMC that arises from abstracting \mathcal{M} by Q . The definition of the upper bound $\tilde{\mathbf{P}}^u(q, q')$ of the probabilistic transition between q and q' needs to be bounded by 1 because $\mathbf{P}^u(s, q') = \sum_{s' \in q'} \mathbf{P}^u(s, s')$ may exceed 1. Every abstraction leads again to an ADTMC [33, Lemma 1].

Example 5. Figure 4b represents the ADTMC after grouping states $\langle 1, 0 \rangle$ and $\langle 1, 1 \rangle$ of the DTMC in Figure 4a into a single abstract state. The probabilistic transition between state $(\langle 1, 0 \rangle, \langle 1, 1 \rangle)$ and $\langle 2, 0 \rangle$ is equipped with the interval $[0.5, 0.8]$ which represents exactly the minimal and maximal probabilities of reaching state $\langle 2, 0 \rangle$ from some of the states $\{\langle 1, 0 \rangle, \langle 1, 1 \rangle\}$. ■

The notion of abstraction on ADTMCs is closely related to forward simulation [31]. In detail, for any ADTMC \mathcal{M} and partition Q we have that \mathcal{M} is simulated by \mathcal{M}/Q [33, Theorem 1]. The three-valued abstraction technique can be adapted to CTMCs without technical difficulties when applying prior uniformization (i. e. all states have equal residence time).

4.2 Reachability analysis and model checking

In the following section we investigate how logical properties, in detail reachability analysis, can be verified on abstract models. The nondeterminism introduced by intervals is resolved using schedulers which lead also to a natural notion of induced DTMC from an ADTMC by a specific scheduler. Interestingly, extreme schedulers, which are schedulers that resolve the probabilities in the intervals to one of the boundaries, suffice to compute maximal/minimal reachability properties ([33, Theorem 2]).

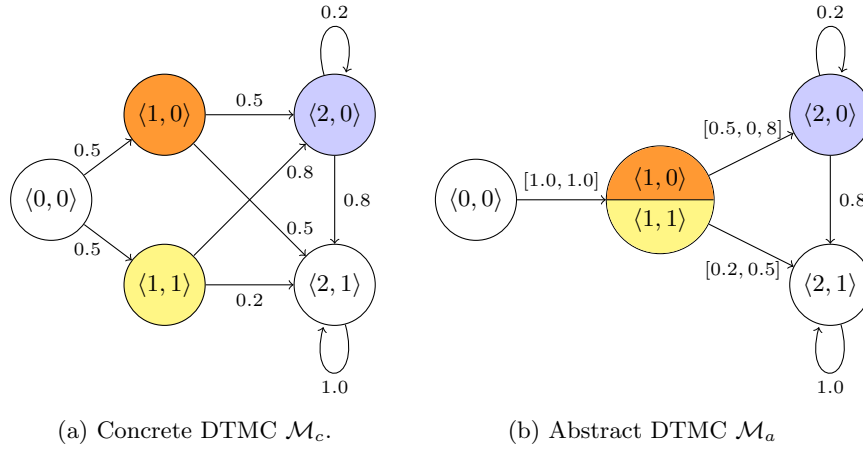


Figure 4: Example for three-valued abstraction

The classical interpretation of PCTL is over a two-valued truth domain $\{\perp, \top\}$. ADTMCs group states together such that some PCTL properties are no longer strictly true or false. For instance, consider the reachability property $\Phi = P_{\geq 0.7}(\diamond \bullet)$ which evaluates to true if the probability to reach a state with label \bullet is at least 0.7. Let us consider the state $(\langle 1, 0 \rangle, \langle 1, 1 \rangle)$ of the ADTMC \mathcal{M}_a . The interval of probabilities $[0.5, 0.8]$ to reach state $\langle 2, 0 \rangle$ allows for probabilities that are greater than 0.7 but also for probabilities that are less than 0.7. Hence, the property Φ evaluates to the indefinite value $?$ in state $(\langle 1, 0 \rangle, \langle 1, 1 \rangle)$. On the other hand, $P_{\geq 0.9}(\diamond \bullet)$ evaluates to \perp in $(\langle 1, 0 \rangle, \langle 1, 1 \rangle)$ because for none of the realizable probabilities the property can become true. Similarly, $P_{\geq 0.3}(\diamond \bullet)$ evaluates in $(\langle 1, 0 \rangle, \langle 1, 1 \rangle)$ to \top because for all realizable probabilities the property becomes true.

To summarize, PCTL properties evaluate in the abstract states of ADTMCs to \mathbb{B}_3 . The semantics differs from the two-valued semantics mainly by the fact that the evaluation of $P_{\bowtie p}(\diamond \varphi)$ is split up in the case $P_{< p}(\diamond \varphi), P_{\leq p}(\diamond \varphi)$ and case $P_{> p}(\diamond \varphi), P_{\geq p}(\diamond \varphi)$. For the first case, $P_{< p}(\diamond \varphi)$ (resp. $P_{\leq p}(\diamond \varphi)$) evaluates to \top if in all realizable probabilistic choices φ is reachable by strictly less than p (resp. at most p). $P_{< p}(\diamond \varphi)$ (resp. $P_{\leq p}(\diamond \varphi)$) evaluates to \perp if in all realizable probabilistic choices φ is reachable by at least p (resp. strictly more than p). The reasoning for the second case is analogous. In all other cases the property evaluates to $?$. It was shown that abstraction preserves validity of PCTL formulae [33, Thm. 3,5 & Cor. 1]. This paves the way for three-valued abstraction-based model checking.

5 Counterexample-guided abstraction refinement

Proposed in 2000 [10], *counterexample-guided abstraction refinement* (CEGAR) quickly became a very successful technique for qualitative verification of safety

properties that proceeds in an iterative manner: starting with an initially coarse overapproximation of the concrete model, it tries to add precision to the parts of the model where required. It does so by analyzing information obtained from the model checking process on the abstract model. The key idea is the following: if the abstract model violates the safety property Φ , it must be possible for a model checker to extract a reason for this, a so-called *counterexample*. This is then analyzed with respect to its realizability in the original model. If it is in fact realizable, we can conclude that the original model also violates Φ . On the other hand, if the counterexample is not realizable, the verification result on the abstract model does not carry over to the concrete model. In this case, the abstraction introduced the *spurious* behavior and the abstract models needs to be refined. As it is known that the counterexample was indeed spurious, it also carries information about the reason why the abstraction introduced this behavior, which can be exploited to refine the current partition. The overall approach is sketched in Figure 5. It is easy to see that for finite models the

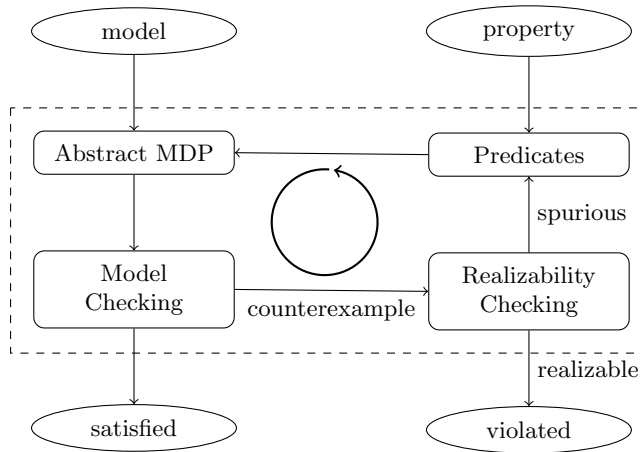


Figure 5: A schema of the CEGAR loop.

CEGAR loop will only be traversed a finite number of times until a decision can be made, assuming that the predicate synthesis always splits at least one abstract state. For infinite models, however, the procedure may not terminate.

We start by introducing the notion of a counterexample for safety properties. Then, we present the core procedure of the CEGAR loop [27]; in particular we discuss the counterexample analysis using satisfiability solvers, and the synthesis of predicates. Note that while [27] also treats systems with arbitrary (even infinitely) many initial states, we restrict our attention to systems with one initial state for the sake of simplicity.

5.1 Counterexamples for safety properties in MDPs

In the traditional qualitative model checking setting, safety properties express that a certain set of “bad” states F must not be reachable. A probabilistic safety property $\mathcal{P}_{\leq p}(\diamond F)$ with $\leq \in \{<, \leq\}$ establishes an upper bound p on the probability to reach the bad states F .

Example 6. Consider for example the MDP in Figure 2 and the probabilistic safety property $\Phi = \mathcal{P}_{<1}(\diamond F)$. Obviously, there are schedulers that violate Φ , namely all schedulers σ that pick action b in $\langle 2, 0 \rangle$ and some available action in the other states. ■

Given a (deterministic memoryless) scheduler σ that violates the safety property Φ , σ can be called a counterexample for Φ . Likewise, the DTMC \mathcal{M}_σ resulting from the application of σ on \mathcal{M} can be called a counterexample. A lot of work has been conducted on finding more succinct representations of counterexamples to make them effectively useful in the context of debugging the system. Several approaches [48, 2, 30] revolve around the identification of a small subsystem of the concrete model that already violates the property. Recently, [49] showed how to characterize and compute counterexamples in terms of the commands of a probabilistic program. Despite this progress, it is yet unclear how to use these sophisticated counterexample representations for the purpose of CEGAR. Hence, we stick to the presentation in [27] and consider deterministic memoryless schedulers $\sigma^{\mathcal{M}/Q}$ and the corresponding DTMC $(\mathcal{M}/Q)_{\sigma^{\mathcal{M}/Q}}$ as counterexamples. As we will see in the next section, in order to avoid building a possibly huge concrete model for checking realizability of a counterexample, the authors of [27] resort to the notion of realizability of an abstract path and view the abstract DTMC $(\mathcal{M}/Q)_{\sigma^{\mathcal{M}/Q}}$ as a set of paths. In general, however, $(\mathcal{M}/Q)_{\sigma^{\mathcal{M}/Q}}$ is cyclic and thus possesses infinitely many paths contributing to the probability mass reaching the set of “bad” states. Unfortunately, sometimes infinitely many paths are in fact needed to prove violation of a safety property.

Example 7. Reconsider the setting of Example 6. It is easy to verify that no finite number of paths suffices to prove that the maximal reachability probability is 1. In this example all paths, i. e., infinitely many, are needed to witness the violation of the safety property. ■

Luckily, it can be shown that this phenomenon only occurs when the comparison operator is strict [24]. In other words, if a property that uses only smaller-or-equal comparisons is violated, then there is always a finite set of paths whose probability mass exceeds the bound. In this case, following the ideas of Han and Katoen [24], a minimal set of paths that exceeds the probability bound p can be efficiently obtained by a reduction to a graph problem. Hence, from now on we assume that we can obtain, one by one, a finite set of paths in decreasing probability order whose accumulated probability mass exceeds the given bound.

5.2 Realizability of a counterexample

Figure 5 illustrates that counterexample analysis is at the heart of the CEGAR approach. It must be possible to determine whether a counterexample in the

abstract system carries over to the concrete model without building it. In order to do this, we first need to define what it means for a counterexample to be realizable. For the remainder of this section, we will assume that the concrete MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ is given as a probabilistic program such that no two commands have the same label. Furthermore, for the current partition Q of the state space the abstract MDP $\mathcal{M}^\# := \mathcal{M}/Q = (Q, Act', \mathbf{P}/Q, AP, L/Q)$ has been built and proven to violate a reachability property $\Phi = \mathcal{P}_{\leq p}(\diamond F)$ by a model checker. In addition, a counterexample $\sigma^\#$ is provided as a witness for the violation by the model checker.

For checking realizability of the abstract scheduler $\sigma^\#$, the idea is to check whether a similarly behaving scheduler σ on \mathcal{M} will exhibit the same violating behavior. Formally, the concretization of a counterexample is defined as follows.

Definition 8 (Concretization and realizability of a counterexample).

- (i) The concretization $\gamma(\sigma^\#)$ of a counterexample $\sigma^\#$ is defined as the scheduler σ for \mathcal{M} such that for all $s \in S$

$$\sigma(s) = \begin{cases} \sigma^\#(q) & s \in q \wedge s \models g \\ \perp & \text{otherwise} \end{cases}$$

where $q \in Q$ and g is the guard of the command associated with the command $\sigma^\#(q)$, which is given by the probabilistic program for \mathcal{M} .

- (ii) A counterexample $\sigma^\#$ is called *realizable* if the probability of reaching a state in F in $\mathcal{M}_{\gamma(\sigma^\#)}$ exceeds the given bound, i. e., $\text{Prob}_{\gamma(\sigma^\#)}^{\mathcal{M}}(\diamond F) > p$ and spurious otherwise.

Intuitively, the concretisation $\gamma(\sigma^\#)$ of a scheduler $\sigma^\#$ is a scheduler for the concrete MDP \mathcal{M} that chooses an action a in a concrete state s iff $\sigma^\#$ chooses a in the abstract state $q \in Q$ containing s and a is available in s . If a is not available in s , because s fails to satisfy the guard of the command, the concretization will just stop in s , which is indicated by \perp . Now, a counterexample is realizable if the concretization induces enough probability mass on the concrete model to violate the bound p of the probabilistic safety property under the concrete scheduler.

Example 8. Reconsider the MDP from Figure 2 and let the safety property be given as $\Phi = \mathcal{P}_{\leq 0.6}(\diamond F)$. Obviously, $\mathcal{M}^\#$ (Figure 3) does not satisfy Φ , because the scheduler $\sigma^\#$ that picks a_2 in A , b in B and c in C achieves a probability of 1. The concretization $\gamma(\sigma^\#)$, thus, has also to choose a in all states of A and so on. However, $\langle 1, 1 \rangle$ fails to satisfy the guard of b , so $\gamma(\sigma^\#)(\langle 1, 1 \rangle) = \perp$. ■

However, given the definition of realizability of a counterexample, it remains to show how to actually perform the realizability check without having the concrete model at hand. As previously mentioned, the authors of [27] resort to viewing the abstract DTMC induced by the abstract counterexample as a set of paths reaching a state in F . This enables to check the realizability of paths in isolation rather than a “full” counterexample at once.

Realizability of a Path. Intuitively, a path in the abstract counterexample DTMC $\mathcal{M}_{\sigma\#}^{\#}$ is *realizable*, if there exists a path in \mathcal{M} that (i) starts in the initial state and ends in F , (ii) does not visit F before the last state (iii) chooses the same actions and updates as the abstract path and (iv) is in a concrete state $s_i \in q_i$ whenever the abstract path is in $q_i \in Q$. Note that this implies that all states along the path satisfy the appropriate guards. The following definition captures this formally:

Definition 9 (Concretization of a path).

Let $\omega^{\#} = q_0\alpha_1q_1\alpha_2 \dots q_n \in \text{Path}_{\text{fin}}^{\mathcal{M}_{\sigma\#}^{\#}}$ be a finite path prefix.

(i) The concretization $\gamma(\omega^{\#})$ of $\omega^{\#}$ is given by

$$\{\omega \in \text{Path}_{\text{fin}}^{\mathcal{M}_{\gamma(\omega^{\#})}} \mid \omega = s_0\alpha_1s_1\alpha_2 \dots s_n \text{ with } s_i \in q_i \text{ for } 0 \leq i \leq n\}$$

(ii) $\omega^{\#}$ is called *realizable* if $\gamma(\omega^{\#}) \neq \emptyset$.

Example 9. Let the setting be the same as the one in Example 8. Furthermore, consider the path prefix $\omega^{\#} = Aa_2C$ in $\mathcal{M}_{\sigma\#}^{\#}$. By inspection, we can see that $\gamma(\omega^{\#}) = \emptyset$, because it is impossible to reach a state in F from the initial state of \mathcal{M} within one step. In fact, no finite path prefix in $\mathcal{M}_{\sigma\#}^{\#}$ possesses a non-empty concretization. ■

The existence of an element in the concretization of the abstract path is formulated as a query to an SMT solver. The actual formula $\varphi_{\gamma}(\omega^{\#})$ for the path $\omega^{\#}$ can be constructed using repeated applications of the weakest precondition operator and additional information obtainable from the path. This formula may then be dispatched to a standard SMT solver supporting linear integer arithmetic, as, for example, Z3 or MathSat. For details, we refer to [27].

Algorithmically checking realizability of a counterexample. Now that we have presented a method for determining whether a path is realizable, we can check the realizability of the counterexample as follows. Given $\mathcal{M}_{\sigma\#}^{\#}$, we start extracting finite paths $\omega^{\#} = q_0\alpha_1q_1\alpha_2 \dots q_n$ with $q_i \notin F$ for $0 \leq i < n$ and $q_n \in F$ in decreasing probability order. Each of these paths is individually checked for realizability. All realizable paths are added to a set Ω^+ whereas unrealizable paths are added to Ω^- and we denote by p_{Ω^+} and p_{Ω^-} , respectively, the sum of the probabilities of the paths in these sets. If we get to a point where $p_{\Omega^+} > p$, we know that enough probability mass of the abstract counterexample is also present in the concrete model to exceed the bound p . This directly implies that the counterexample is in fact realizable and we can conclude that \mathcal{M} violates the given safety property as well. Conversely, we need a criterion to stop looking for new paths in $\mathcal{M}_{\sigma\#}^{\#}$ if there is no hope of ever exceeding p . As the model checker that provided the abstract counterexample computed the probability $p_{\sigma\#}(\diamond F)$ of reaching F in the abstract model, we can at any point determine whether there is enough probability mass left in the abstract counterexample that is potentially realizable and suffices to exceed p . If $p_{\Omega^+} + (p_{\sigma\#}(\diamond F) - p_{\Omega^-}) \leq p$,

we can conclude that even if all remaining paths in $\mathcal{M}_{\sigma^\#}^\#$ were realizable, they could still not exceed the bound p and thus, the abstract counterexample was determined to be spurious. Formally, we get the following result:

Lemma 1 (Termination criterion). *Let $\epsilon = p_{\sigma^\#}(\diamond F) - p_{\Omega^-}$ be the probability of paths reaching F in the abstract model that were not yet proven to be (un)realizable.*

- (i) $p_{\Omega^+} > p$ implies that $\sigma^\#$ is realizable.
- (ii) $p_{\Omega^+} + \epsilon \leq p$ implies that $\sigma^\#$ is spurious.

Stated differently, the result is only inconclusive if $p_{\Omega^+} \in (p - \epsilon, p]$. In this case, the next most likely abstract path is considered until the result is in fact conclusive.

Example 10. Reconsider Example 8. The model checker initially returns the counterexample $\sigma^\#$ along with the reachability probability $p_{\sigma^\#}(\diamond F) = 1$. Let the first path prefix that is found in $\mathcal{M}_{\sigma^\#}^\#$ by the procedure be $\omega^\# = Aa_2C$. As shown in Example 9, this path prefix is found to be unrealizable. It is then added to Ω^- and its probability is added to p_{Ω^-} , which then becomes 0.5. Now, however, ϵ becomes 0.5, which means that there is at most a probability mass of 0.5 left in $\mathcal{M}_{\sigma^\#}^\#$ that might be realizable. As this does not suffice to exceed the bound $p = 0.6$ of Φ , the counterexample is known to be spurious. If this was not directly the case, the procedure would check the next path prefix for realizability and carry on. ■

5.3 Predicate synthesis

Suppose the decision procedure previously described determines that a given counterexample in the abstract model is spurious. This means that the abstraction falsely introduced behavior that was not present in the original model. In other words, we need to refine the abstract model to rule out this spurious behavior. As the abstract model is built using predicate abstraction, this corresponds to introducing additional predicates. Since the formulae $\varphi_\gamma(\omega^\#)$ do not involve quantitative aspects, but are similar to the non-probabilistic case, standard techniques, such as predicate interpolation [42, 45], may be used to obtain predicates to rule out the source of spuriousness.

Example 11. After the counterexample from Example 10 was found to be spurious, the predicate $x = 0 \wedge y = 0$ could be added to rule out the possibility to reach F within one step in the quotient model $\mathcal{M}^\#$. ■

6 Game-based abstraction

The success of *counterexample-guided abstraction refinement* inspired another abstraction-refinement framework for MDPs. First, observe that probabilistic

CEGAR (as presented in Section 5) can only (dis)prove probabilistic safety properties. Intuitively, this is because the concrete MDP is again abstracted to an MDP. Doing so, however, merges the nondeterminism of the concrete model with the nondeterminism introduced by the abstraction. Effectively this means that the minimal and maximal reachability probabilities in the abstract MDP are lower and upper bounds, respectively, for the corresponding reachability probabilities in the concrete model (see Theorem 1). Rather than merging the two sources of nondeterminism, the two abstraction techniques presented in this section keep them separated by using probabilistic games [11] as their underlying abstract model. This way, they are able to provide lower and upper bounds on both minimal and maximal reachability probabilities. Hence, they are applicable to the broader class of probabilistic reachability properties.

6.1 Idea

Reconsider Example 4. In the abstract MDP, the minimal and maximal probability to reach a state in F are 0 and 1, respectively. This, however, means that the reachability probabilities in the concrete model may lie anywhere in between those values providing no information at all. This phenomenon stems from the fact that, in the abstract state A , both the nondeterministic choices of state $\langle 0, 1 \rangle$ and the transitions emanating from $\langle 0, 0 \rangle$ are enabled.

The key idea of game-based abstraction [34] is the following. Instead of using an MDP as the “target” of the abstraction, the concrete MDP is mapped to an abstract probabilistic game. This way, the two sources of nondeterminism can be assigned to the different players and, thus, kept separate. In other words, one player is responsible for resolving the nondeterminism of the abstraction while the other governs the nondeterminism of the original model. Depending on whether the two players both try to maximize or minimize the probability to reach the target states or they take an adversarial role, the resulting value of the game is a lower or upper bound on the minimal or maximal reachability probability, respectively. If these bounds are precise enough for proving or refuting a given property, a conclusive answer for satisfaction of the property on the concrete model can be given. In the other case, at least one block of the abstraction can be refined based on the strategies of the players. The resulting game then yields more precise results and, similarly to CEGAR, the procedure may be iterated until the obtained bounds are precise enough. The approach is sketched in Figure 6. Note that in practice this approach can be implemented in a fully symbolic way (just like CEGAR) by using SMT solvers that avoids building the concrete model \mathcal{M} altogether. However, for the sake of simplicity, our presentation will abstract from this.

6.2 Simple game-based abstraction

We will now present how to obtain an appropriate (abstract) probabilistic game from an MDP. Just like for MDP quotienting, a set of predicates is used to partition the state space S of the concrete MDP into blocks of states $Q =$

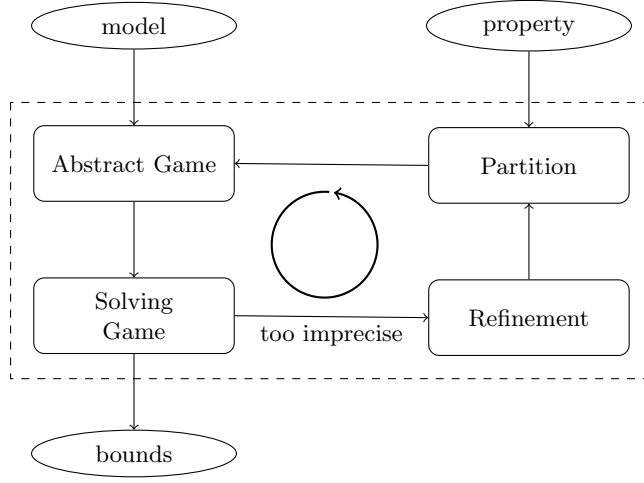


Figure 6: A schema of refinement loop of game-based abstraction.

$\{S_1, \dots, S_n\}$. Recall that $\bar{\mu} \in \text{Dist}(Q)$ denotes the lifting of $\mu \in \text{Dist}(S)$ to $\text{Dist}(Q)$.

Definition 10 (Simple probabilistic game-based abstraction). *Given an MDP $\mathcal{M} = (S, \text{Act}, \mathbf{P}, s_{\text{init}}, AP, L)$ and a partition Q of S , the simple probabilistic game-based abstraction of \mathcal{M} over Q is the probabilistic game*

$$\mathcal{G}_{\mathcal{M}}^Q = ((V = V_1 \dot{\cup} V_2 \dot{\cup} V_p, E), v_{\text{init}}, (V_1, V_2, V_p), \delta)$$

where

- $V_1 = Q$ are player 1's vertices,
- $V_2 = \{v_2 \subseteq \text{Dist}(Q) \mid v_2 = \{\bar{\mu} \mid \mu \in \text{Steps}(s)\} \text{ for some } s \in S\}$ are player 2's vertices,
- $V_p = \{v_p \in \text{Dist}(Q) \mid v_p \in \{\bar{\mu} \mid \mu \in \text{Steps}(s)\} \text{ for some } s \in S\}$ are the probabilistic vertices,
- $v_{\text{init}} = q \in Q$ such that $s_{\text{init}} \in q$ is the initial vertex,
- $\delta : V_p \rightarrow \text{Dist}(V)$ is the identity function,

and the set of edges E is given by

$$\begin{aligned} E = & \{(v_1, v_2) \mid v_1 \in V_1 \text{ and } v_2 = \{\bar{\mu} \mid \mu \in \text{Steps}(s) \text{ for some } s \in v_1\} \\ & \cup \{(v_2, v_p) \mid v_2 \in V_2 \text{ and } v_p \in v_2\} \\ & \cup \{(v_p, v_1) \mid v_p = \bar{\mu} \text{ with } \bar{\mu}(v_1) > 0\}. \end{aligned}$$

Intuitively, the game proceeds as follows. In each $v_1 \in V_1$, player 1 picks a concrete state $s \in v_1$ and moves to the corresponding player 2 vertex. Then, player 2 chooses a (lifted) probability distribution $\bar{\mu}$ available in s . Finally, the

S. Then for all $v \in V_1$ and $s \in v$

$$p_v^{--}(F) \leq p_s^-(F) \leq p_v^{+-}(F), \quad (1)$$

$$p_v^{-+}(F) \leq p_s^+(F) \leq p_v^{++}(F). \quad (2)$$

Example 13. For the abstraction in Example 12 we obtain $p_{v_{init}}^{--}(\{C\}) = 0$. Unlike the MDP abstraction \mathcal{M}/Q , we can, however, obtain a better upper bound on the minimal reachability probability than 1. Observe that, if only player 1 tries to maximize the probability value, he does not choose the state $\langle 0, 0 \rangle$ (the bottommost choice emanating from A) but any of the other states. Then, player 2 can not completely avoid reaching C any more, but has to go to C with a probability of at least 0.2. Indeed, solving the game yields $p_{v_{init}}^{+-}(\{C\}) = 0.2$. Hence, the minimal reachability probability in \mathcal{M} is determined to lie in the interval $[0, 0.2]$ providing more precise information than the MDP quotient over the same partition Q . ■

As indicated in Figure 6, after obtaining bounds by solving a game, the partition Q may need to be refined in order to obtain more precise results. We will show how the refinement may be done in a way that guarantees termination of the procedure for finite models \mathcal{M} .

Refinement. Recall that, given the goals of players 1 and 2, solving a game not only comprises computing the extremal reachability probability for the game, but also produces memoryless deterministic strategies for the two players that together achieve the computed probability. Suppose we obtained the bounds $[l, u]$ for the minimal reachability probability by solving the game $\mathcal{G}_{\mathcal{M}}^Q$ (twice). Further assume that the bounds were imprecise, i.e., $l < u$. Then, two pairs of memoryless deterministic strategies (σ_1^l, σ_2^l) and (σ_1^u, σ_2^u) are generated such that:

$$p_{v_{init}}^{\sigma_1^l \sigma_2^l}(F) = l \text{ and } p_{v_{init}}^{\sigma_1^u \sigma_2^u}(F) = u.$$

Since $l \neq u$, there is at least one $v_1 \in V_1$ where the two player 1 strategies disagree, i.e., $\sigma_1^l(v_1) \neq \sigma_1^u(v_1)$. Intuitively, this means that player 1 chose different concrete states contained in v_1 depending on whether he wanted to minimize or maximize the reachability probability. Consequently, v_1 can be split to narrow down the choices of player 1 in the resulting vertices. A possible way to achieve this, is to split v_1 into blocks v_1^l, v_1^u and v_1^r where

$$\begin{aligned} v_1^l &= \{s \in v_1 \mid \sigma_1^l(v_1) = \{\bar{\mu} \mid \mu \in \text{Steps}(s)\}\} \\ v_1^u &= \{s \in v_1 \mid \sigma_1^u(v_1) = \{\bar{\mu} \mid \mu \in \text{Steps}(s)\}\} \\ v_1^r &= v_1 \setminus (v_1^l \cup v_1^u). \end{aligned}$$

Of course, there may be several vertices that can be split according to this criterion and it is not clear which or how many blocks should be refined in order to get more precise bounds that are able to prove or disprove the property at hand. This refinement method is called *strategy-based*. There exist other refinement techniques, for example *value-based* refinement, which are not covered here. For details, we refer to [34].

Example 14. As pointed out in Example 13, player 1 chooses state $\langle 0, 0 \rangle \in A$ or either of the states $\langle 1, 0 \rangle, \langle 1, 1 \rangle \in A$ if he wants to minimize or maximize, respectively, the reachability probability in $\mathcal{G}_{\mathcal{M}}^Q$. Consequently, A is split into blocks $A_1 = \{\langle 0, 0 \rangle\}$ and $A_2 = \{\langle 1, 0 \rangle, \langle 1, 1 \rangle\}$. The resulting game over the partition $Q' = (Q \setminus A) \cup \{A_1, A_2\}$ is depicted in Figure 8. Solving the refined game determines the minimal reachability probability to be in the interval $[0.2, 0.2]$. ■

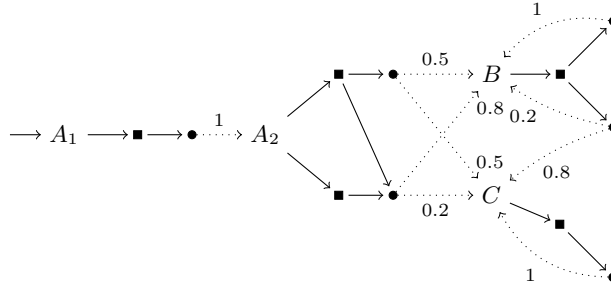


Figure 8: Game-based abstraction after the refinement of block A .

6.3 Menu-based abstraction

While game-based abstraction yields good results for many examples, the constructed game can become very large. The reason is that the game representation may need one player 2 vertex for each combination of (lifted) probability distributions available in some block $q \in Q$. The worst-case appears if all the states contained in a particular block happen to have different (combinations of) lifted probability distributions. Roughly speaking, in the context of MDPs given by a probabilistic program, the game may become large if there exist many states in which different combinations of guarded commands are enabled. In this case, constructing and solving the game might be very expensive.

Example 15. The game $\mathcal{G}_{\mathcal{M}}^Q$ in Example 12 has three player 2 vertices reachable in one step from player 1 vertex A , even though block A contained only three states of the concrete model. ■

Menu-based abstraction [46, 45] aims to overcome this by considering commands of the probabilistic program in isolation. That is, it builds a possibly smaller game than game-based abstraction that might produce coarser probability approximations in the hope that it can be constructed and solved more easily. Instead of letting player 1 pick a concrete state out of a given block and move to the vertex representing the enabled commands at this particular state, it lets player 1 choose a command. This means that the choice of a concrete state is still

open (among all states that have the chosen command enabled). Consequently, in the successor vertex, player 2 has the choice between all possible realizations of the chosen command in all states of the block.

Definition 11 (Menu game). *Given an MDP $\mathcal{M} = (S, Act, \mathbf{P}, s_{init}, AP, L)$ and a partition Q of S , the menu-based abstraction of \mathcal{M} over Q is the probabilistic game*

$$\widehat{\mathcal{G}}_{\mathcal{M}}^Q = ((V = V_1 \dot{\cup} V_2 \dot{\cup} V_p, E), v_{init}, (V_1, V_2, V_p), \delta)$$

where

- $V_1 = Q \cup \{\perp\}$ are player 1's vertices,
- $V_2 = \{(v_1, a) \mid v_1 \in V_1, a \in Act(v_1)\}$ are player 2's vertices,
- $V_p = \{\mathbf{P}(s, a, \cdot) \mid s \in S, a \in Act(s)\} \cup \{v_p^\perp\}$ are the probabilistic vertices,
- $v_{init} = B \in P$ such that $s_{init} \in B$ is the initial vertex, and
- $\delta : V_p \rightarrow Dist(V)$ is the identity function,

and the set of edges E is given by

$$\begin{aligned} E = & \{(v_1, v_2) \mid v_1 \in V_1, v_2 = (v_1, a) \in V_2, a \in Act(v_1)\} \\ & \cup \{(v_2, v_p) \mid v_2 = (v_1, a) \in V_2, \exists s \in v_1 : v_p = \mathbf{P}(s, a, \cdot)\} \\ & \cup \{(v_2, v_p^\perp), (v_p^\perp, \perp) \mid v_2 = (v_1, a) \in V_2, \exists s \in V_1 : a \notin Act(s)\} \\ & \cup \{(v_p, v') \mid v_p \in V_p, v' \in V_1 : v_p(v') > 0\} \end{aligned}$$

where $v_p^\perp \in Dist(S \cup \{\perp\})$ is defined by $v_p^\perp(v) = 1$ iff $v = \perp$.

To distinguish the probabilities obtained in the game-based abstraction $\mathcal{G}_{\mathcal{M}}^Q$ from the ones obtained in the menu-based abstraction $\widehat{\mathcal{G}}_{\mathcal{M}}^Q$, we will denote the latter by $\widehat{p}_v^{\circ_1 \circ_2}(F)$ with $\circ_1, \circ_2 \in \{-, +\}$ and $\widehat{p}_v^{\sigma_1 \sigma_2}(F)$.

Starting in the initial vertex, player 1 chooses one of the commands that are enabled in at least one state in the current block. Then, player 2 implicitly chooses a state from the current block by choosing a probability distribution that is (i) created by the chosen command and (ii) is available at some state in the block. Finally, the successor vertices are given by that distribution and the play is once again in a vertex owned by player 1. ■

Example 16. Reconsider the MDP \mathcal{M} in Figure 2 and the partition Q from Example 2. The resulting menu game is shown in Figure 9. Note that the labeling of player 1's choices with commands is added to illustrate the correspondence, but is not actually part of the game itself. ■

As for game-based abstraction, we can state the correctness of the abstraction in the sense that the reachability probabilities obtained from the game are lower and upper bounds for the reachability probabilities in the original MDP.

will result in lower and upper bounds for both the minimal and the maximal reachability probability with respect to the given set of target states.

Compared to game-based abstraction, it should be noted that it is no longer the case that player 1 resolves the nondeterminism introduced by the abstraction and player 2 the nondeterminism of the original model, but that the two have swapped roles. In the game-based abstraction setting, player 1 determined whether the resulting probability was a lower or upper bound while player 2 could control whether the result was an approximation of the minimal or maximal reachability probability in the original MDP. This is exactly reversed in the context of menu games, which is reflected in the previous correctness theorem by swapping the goals of the two players (compared to game-based abstraction).

Also, there are examples for which game-based abstraction produces tighter bounds than menu-based abstraction if the same partition Q of the state space is used for building the games. Technically, this happens because game-based abstraction constructs a game representation of the best transformer on the partition induced by the predicates whereas menu-based abstraction represents an abstract transformer that does not necessarily coincide with the best transformer [45].

Example 18. Reconsider the menu game $\widehat{\mathcal{G}}_{\mathcal{M}}^Q$ from Example 16. For the menu-based abstraction, the lower bounds for minimum and maximum reachability are both 0 and, likewise, the upper bounds are both 1, effectively yielding no information. As shown in Example 13, this is coarser than the bounds obtained via game-based abstraction (using the same partition Q). ■

This immediately raises the issue of termination if menu-based abstraction is to be used in a refinement loop. Fortunately, the following result can be established.

Theorem 4 (Refinability). *Any finite partition Q can be refined to a finite partition Q' on which the reachability probabilities in the menu game approximate the reachability properties in the original MDP at least as precisely as the game-based abstraction over Q . Formally, for every $s \in S$, let $v \in Q$, $v' \in Q'$ such that $s \in v$ and $s \in v'$, then:*

$$p_{v,Q}^{\bar{-}}(F) \leq \widehat{p}_{v',Q'}^{\bar{-}}(F \cup \{\perp\}) \leq p_s^{\bar{-}}(F) \leq \widehat{p}_{v',Q'}^{\bar{+}}(F \cup \{\perp\}) \leq p_{v,Q}^{\bar{+}}(F) \quad (5)$$

$$p_{v,Q}^{\bar{+}}(F) \leq \widehat{p}_{v',Q'}^{\bar{+}}(F) \leq p_s^{\bar{+}}(F) \leq \widehat{p}_{v',Q'}^{\bar{+}}(F) \leq p_{v,Q}^{\bar{+}}(F). \quad (6)$$

Note that the players swapped roles, so the order of the superscripts of the reachability probabilities is important.

Extensions. For models that involve parametric transition probabilities depending on state variables, the usual game construction possibly produces games of infinite size, because infinitely many probability distributions might be available in a block. Recently, [22] proposed to solve this problem by constructing *constraint Markov games*, an extension of probabilistic games that is able to deal

with *variable probabilities*, instead. Intuitively, the idea is to avoid introducing a game vertex for every available distribution by shifting the selection of the distribution into a different level of non-determinism in the game.

7 Conclusion

We have described three successful techniques for the abstraction of probabilistic systems. Which one is most useful in a concrete situation? Multi-valued abstraction seems to be the simplest method: one stays within the model of MDPs, so analysis of the abstract model can use mainstream model checkers. However, the disadvantage is that some questions cannot be answered. In those cases the abstraction-refinement frameworks demonstrate their strengths. CEGAR overcomes a part of the weakness of multi-valued abstraction, by providing a direction in which to refine a model if model checking on the abstract model has led to a spurious counterexample. Game-based techniques do not rely on MDPs as their underlying abstract model but rather use probabilistic games. This way, they can provide lower bounds on both minimal and maximal reachability probabilities.

References

- [1] de Alfaro, L., Pritam, R.: Magnifying-lens abstraction for Markov decision processes. In: Proc. CAV. LNCS, vol. 4590, pp. 325–338. Springer (2007)
- [2] Aljazzar, H., Leue, S.: Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Trans. Software Eng.* 36(1), 37–60 (2010)
- [3] Aziz, A., Singhal, V., Balarin, F., Brayton, R. K., Sangiovanni-Vincentelli, A. L.: It usually works: The temporal logic of stochastic systems. In: Wolper, P. (ed.) Proc. CAV. LNCS, vol. 939, pp. 155–165. Springer, Berlin (1995)
- [4] Baier, C., Katoen, J.-P.: Principles of model checking. MIT Press, Cambridge, Mass. (2008)
- [5] Baier, C., Katoen, J.-P., Hermanns, H., Wolf, V.: Comparative branching-time semantics for Markov chains. *Information and Computation* 200(2), 149–214 (2005)
- [6] Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P. S. (ed.) Proc. FSTTCS. LNCS, vol. 1026, pp. 499–513. Springer (1995)
- [7] Bozzano, M., Cimatti, A., Katoen, J.-P., Nguyen, V. Y., Noll, T., Roveri, M.: Safety, dependability and performance analysis of extended aadl models. *Comput. J.* 54(5), 754–775 (2011)
- [8] Chadha, R., Viswanathan, M.: A counterexample-guided abstraction-refinement framework for markov decision processes. *ACM Trans. Comput. Log.* 12(1), 1 (2010)
- [9] Chen, T., Forejt, V., Kwiatkowska, M. Z., Parker, D., Simaitis, A.: Prism-games: A model checker for stochastic multi-player games. In: Proc. TACAS’13. pp. 185–191 (2013)
- [10] Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. CAV’00. LNCS, vol. 1855, pp. 154–169. Springer (2000)

- [11] Condon, A.: The complexity of stochastic games. *Information and Computation* 96(2), 203–224 (1992)
- [12] D’Argenio, P. R., Jeannet, B., Jensen, H. E., Larsen, K. G.: Reachability analysis of probabilistic systems by successive refinements. In: de Alfaro, L., Gilmore, S. (eds.) PAPM-PROBMIV. LNCS, vol. 2165, pp. 39–56. Springer (2001)
- [13] 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.) PAPM-PROBMIV. LNCS, vol. 2399, pp. 57–76. Springer (2002)
- [14] Dehnert, C., Katoen, J.-P., Parker, D.: Smt-based bisimulation minimisation of markov models. In: Giacobazzi, R., Berdine, J., Mastroeni, I. (eds.) VMCAI. LNCS, vol. 7737, pp. 28–47. Springer (2013)
- [15] Delahaye, B., Katoen, J.-P., Larsen, K. G., Legay, A., Pedersen, M. L., Sher, F., Wasowski, A.: Abstract probabilistic automata. In: Jhala, R., Schmidt, D. A. (eds.) VMCAI. LNCS, vol. 6538, pp. 324–339. Springer (2011)
- [16] Desharnais, J., Gupta, V., Jagadeesan, R., Panangaden, P.: Approximating labelled Markov processes. *Information and computation* 184(1), 160–200 (2003)
- [17] Donaldson, A. F., Miller, A.: Symmetry reduction for probabilistic model checking using generic representatives. In: Proc. ATVA’06. LNCS, vol. 4218, pp. 9–23. Springer (2006)
- [18] Eisentraut, C., Hermanns, H., Schuster, J., Turrini, A., Zhang, L.: The quest for minimal quotients for probabilistic automata. In: Proc. TACAS’13. LNCS, vol. 7795, pp. 16–31. Springer (2013)
- [19] Emerson, E. A., Clarke, E. M.: Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming* 2(3), 241–266 (1982)
- [20] Fecher, H., Leucker, M., Wolf, V.: *Don’t know* in probabilistic systems. In: Valmari, A. (ed.) Proc. SPIN’06. LNCS, vol. 3925, pp. 71–88. Springer, Berlin (2006)
- [21] Feng, L., Kwiatkowska, M. Z., Parker, D.: Compositional verification of probabilistic systems using learning. In: QEST. pp. 133–142. IEEE Computer Society (2010)
- [22] Ferrer Fioriti, L. M., Hahn, E. M., Hermanns, H., Wachter, B.: Variable probabilistic abstraction refinement. In: Chakraborty, S., Mukund, M. (eds.) Proc. ATVA’12. LNCS, vol. 7561, pp. 300–316. Springer, Berlin (2012)
- [23] Hahn, E. M., Hermanns, H., Wachter, B., Zhang, L.: Pass: Abstraction refinement for infinite probabilistic models. In: Proc. TACAS’10. LNCS, vol. 6015, pp. 353–357. Springer (2010)
- [24] Han, T., Katoen, J.-P.: Counterexamples in probabilistic model checking. In: Grumberg, O., Huth, M. (eds.) TACAS’07. LNCS, vol. 4424, pp. 72–86. Springer, Berlin (2007)
- [25] Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6(5), 512–535 (1994)
- [26] Henzinger, T. A., Mateescu, M., Wolf, V.: Sliding window abstraction for infinite Markov chains. In: Bouajjani, A., Maler, O. (eds.) Proc. CAV’09. LNCS, vol. 5643, pp. 337–352. Springer, Berlin (2009)
- [27] Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) Proc. CAV’08. LNCS, vol. 5123, pp. 162–175. Springer, Berlin (2008)
- [28] Huth, M., Jagadeesan, R., Schmidt, D.: Modal transition systems: A foundation for three-valued program analysis. In: Sands, D. (ed.) Proc. ESOP’01. LNCS, vol. 2028, pp. 155–169. Springer, Berlin (2001)

- [29] Jansen, D. N., Song, L., Zhang, L.: Revisiting weak simulation for substochastic Markov chains. In: Joshi, K., Siegle, M., Stoelinga, M., D’Argenio, P. (eds.) Proc. QEST’13. LNCS, vol. 8054, pp. 209–224. Springer (2013)
- [30] Jansen, N., Ábrahám, E., Katelaan, J., Wimmer, R., Katoen, J.-P., Becker, B.: Hierarchical counterexamples for discrete-time markov chains. In: Bultan, T., Hsiung, P.-A. (eds.) ATVA. LNCS, vol. 6996, pp. 443–452. Springer (2011)
- [31] Jonsson, B., Larsen, K. G.: Specification and refinement of probabilistic processes. In: Proc. LICS’91. pp. 266–277. IEEE Comp. Soc. Pr. (1991)
- [32] Katoen, J.-P., Kemna, T., Zapreev, I. S., Jansen, D. N.: Bisimulation minimisation mostly speeds up probabilistic model checking. In: Grumberg, O., Huth, M. (eds.) TACAS. LNCS, vol. 4424, pp. 87–101. Springer (2007)
- [33] Katoen, J.-P., Klink, D., Leucker, M., Wolf, V.: Three-valued abstraction for probabilistic systems. *JLAP* 81(4), 356–389 (2012)
- [34] Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design* 36(3), 246–280 (2010)
- [35] Kattenbelt, M., Kwiatkowska, M. Z., Norman, G., Parker, D.: A game-based abstraction-refinement framework for markov decision processes. *Formal Methods in System Design* 36(3), 246–280 (2010)
- [36] Komuravelli, A., Pasareanu, C. S., Clarke, E. M.: Assume-guarantee abstraction refinement for probabilistic systems. In: Madhusudan, P., Seshia, S. A. (eds.) CAV. LNCS, vol. 7358, pp. 310–326. Springer (2012)
- [37] Kwiatkowska, M. Z., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Ball, T., Jones, R. B. (eds.) CAV. LNCS, vol. 4144, pp. 234–248. Springer (2006)
- [38] Kwiatkowska, M. Z., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV. LNCS, vol. 6806, pp. 585–591. Springer (2011)
- [39] Kwiatkowska, M. Z., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Proc. TACAS’10. LNCS, vol. 6015, pp. 23–37. Springer (2010)
- [40] Larsen, K. G., Thomsen, B.: A modal process logic. In: Proc. LICS’88. pp. 203–210. Los Alamitos, Calif. (1988)
- [41] Larsen, K. G., Thomsen, B.: A modal process logic. In: LICS. pp. 203–210 (1988)
- [42] McMillan, K. L.: Lazy abstraction with interpolants. In: Proc. CAV’06. pp. 123–136. Springer (2006)
- [43] Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. *Nordic Journal of Computing* 2, 250–273 (1995)
- [44] Vardi, M. Y.: Automatic verification of probabilistic concurrent finite-state programs. In: FOCS. pp. 327–338. IEEE Comp. Soc. Pr., Washington DC (1985)
- [45] Wachter, B.: Refined probabilistic abstraction. Ph.D. thesis, Universität des Saarlandes, Saarbrücken (2011)
- [46] Wachter, B., Zhang, L.: Best probabilistic transformers. In: Barthe, G., Hermenegildo, M. (eds.) Proc. VMCAI’10. LNCS, vol. 5944, pp. 362–379. Springer, Berlin (2010)
- [47] Wimmer, R., Herbstritt, M., Hermanns, H., Strampp, K., Becker, B.: Sigref- a symbolic bisimulation tool box. In: Proc. ATVA’06. LNCS, vol. 4218, pp. 477–492. Springer (2006)
- [48] Wimmer, R., Jansen, N., Ábrahám, E., Becker, B., Katoen, J.-P.: Minimal critical subsystems for discrete-time markov models. In: Flanagan, C., König, B. (eds.) TACAS. LNCS, vol. 7214, pp. 299–314. Springer (2012)

- [49] Wimmer, R., Jansen, N., Vorpahl, A., Ábrahám, E., Katoen, J.-P., Becker, B.: High-level counterexamples for probabilistic automata. In: Joshi, K. R., Siegle, M., Stoelinga, M., D'Argenio, P. R. (eds.) QEST. LNCS, vol. 8054, pp. 39–54. Springer (2013)
- [50] Zhang, L.: Decision algorithms for probabilistic simulations. Ph.D. thesis, Universität des Saarlandes, Saarbrücken (2009)