Advances in Probabilistic Model Checking

Marta KWIATKOWSKA [a], David PARKER [a]
[a] *Department of Computer Science, University of Oxford, Oxford, UK*

Abstract. Probabilistic model checking is an automated verification method that aims to establish the correctness of probabilistic systems. Probability may arise, for example, due to failures of unreliable components, communication across lossy media, or through the use of randomisation in distributed protocols. Probabilistic model checking enables a range of exhaustive, quantitative analyses of properties such as "the probability of a message being delivered within 5ms is at least 0.89".

In the last ten years, probabilistic model checking has been successfully applied to numerous real-world case studies, and is now a highly active field of research. This tutorial gives an introduction to probabilistic model checking, as well as presenting material on selected recent advances. The first half of the tutorial concerns two classical probabilistic models, discrete-time Markov chains and Markov decision processes, explaining the underlying theory and model checking algorithms for the temporal logic PCTL. The second half discusses two advanced topics: quantitative abstraction refinement and model checking for probabilistic timed automata. We also briefly summarise the functionality of the probabilistic model checker PRISM, the leading tool in the area.

Keywords. Markov models; Probabilistic temporal logics; Probabilistic model checking; Quantitative model checking

1. Introduction

Probabilistic modelling is widely used for the design and analysis of computer systems. Probability is typically employed to quantify unreliable or unpredictable behaviour, for example in fault-tolerant systems and communication protocols, where properties such as component failure and message loss can be expressed probabilistically. In distributed co-ordination algorithms, randomness serves as a symmetry breaker in order to derive efficient algorithms, see e.g. random back-off schemes in IEEE 802.11 or Bluetooth device discovery, and population protocols [4]. Traditionally, probability has also been used as a tool to analyse system performance and Quality of Service.

Probabilistic model checking is an automatic procedure for establishing if a desired property holds in a probabilistic system model. Conventional model checkers input a description of a model, representing a state-transition system, and a specification, typically a formula in some temporal logic, and return "yes" or "no", indicating whether or not the model satisfies the specification. In the latter case, a diagnostic trace, referred to as a counterexample, is returned. In probabilistic model checking, the models are probabilistic (typically variants of Markov

chains), in the sense that they encode the *probability* of making a transition between states instead of simply the existence of such a transition. A probability space induced on the executions of the system enables calculation of the likelihood of the occurrence of certain events of interest. This in turn allows *quantitative* statements to be made about the system's behaviour, expressed as probabilities or expectations, in addition to the qualitative statements made by conventional model checking. Probabilities are captured via *probabilistic operators* that extend conventional (timed or untimed) temporal logics. Models can be additionally annotated with costs and rewards, and the *reward operator* enables the computation of expectations with respect to the underlying probability space. The extended logics can express the following *probabilistic* and *reward specifications*:

- for a randomised leader election algorithm: "leader election is eventually resolved *with probability* 1";
- for a security protocol: "the chance of intrusion is *at most* 0.0001%";
- for a web service: "what is the probability of a response *within* 5ms?";
- for a wireless communication protocol: "what is the worst-case *expected time* for delivering a data packet?";
- for a battery-powered device: "the maximum *expected energy consumption* in 24hrs is at most 190J".

Note that answers to the above queries can be truth values, when the specification simply asks for a comparison to a probability threshold, or quantitative, returning the actual probability or expectation.

The first algorithms for probabilistic model checking were proposed in the 1980s [36,63,21], originally focussing on *qualitative* probabilistic temporal properties (i.e. those satisfied with probability 1 or 0) but later also introducing *quantitative* properties. These were followed by various extensions [35,14,8] and first implementations [34,5]. However, the first industrial strength probabilistic model checkers were developed only in the 2000s [24,40], when the field matured. Probabilistic model checking draws on conventional model checking, since it relies on reachability analysis of the underlying transition system, and to this end techniques such as symbolic model checking, symmetry reduction, counterexamples and abstraction refinement have been usefully adapted. This is combined with appropriate numerical methods, such as linear algebra or linear programming, in order to provide the calculation of the actual likelihoods and expectations. The main advantage is that the analysis is exhaustive, resulting in numerically exact answers to the temporal logic queries (in contrast to approximate analysis methods such as simulation), and is able to express detailed temporal constraints on the system's executions (in contrast to analytical methods).

Probabilistic model checking has been successfully applied in a multitude of domains: distributed coordination algorithms, wireless communication protocols, security, anonymity and quantum cryptographic protocols, nanotechnology designs, power management and modelling of biological processes. Several flaws and unusual features have been discovered using the techniques; for more information, see e.g. [50,67]. As more real-world case studies are being analysed, user expectations are growing with regards to the efficiency and accuracy of the results, the degree of automation of the methods, and the range of systems that can be modelled

and analysed. Probabilistic model checking has developed into an exciting and highly active field of research that covers the full spectrum, from theory, through implementation techniques, to applications, and we are pleased to introduce the reader to the main concepts, as well describe some research highlights.

Outline. This tutorial begins with an introduction to probabilistic model checking based on two classical probabilistic models with discrete states and discrete probability distributions – discrete time Markov chains and Markov decision processes – which respectively model fully probabilistic systems and concurrent probabilistic systems. We introduce the probabilistic temporal logic PCTL and its reward extension, which is interpreted over states of the two classical models, and explain the corresponding model checking methods. The second half of the tutorial discusses two advanced topics: quantitative abstraction refinement for Markov decision processes and model checking for probabilistic timed automata. The latter can be viewed as Markov decision processes extended with real-valued clocks, and the former method underpins their model checking, in addition to model checking for real probabilistic software. The models, specification formalisms and techniques introduced here are supported by the probabilistic model checker PRISM [55,67], which is also briefly described.

2. Preliminaries

Let Ω be a *sample set*, the set of possible outcomes of an experiment. A pair $(\Omega, \mathcal{F})$ is said to be a *sample space* if $\mathcal{F}$ is a σ-field of subsets of Ω, often built from *basic cylinders/cones* by closing w.r.t. countable unions and complement. The elements of $\mathcal{F}$ are called *events*. A triple $(\Omega, \mathcal{F}, \mu)$ is a *probability space* if μ is a probability measure over $\mathcal{F}$, i.e. $0 \leqslant \mu(A) \leqslant 1$ for all $A \in \mathcal{F}$; $\mu(\varnothing) = 0$, $\mu(\Omega) = 1$ and $\mu(\bigcup_{k=1}^{\infty} A_k) = \sum_{k=1}^{\infty} \mu(A_k)$ for disjoint A_k.

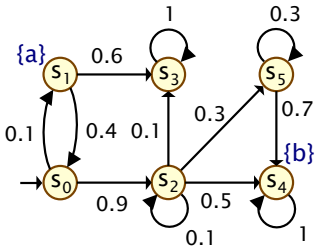
Let $(\Omega, \mathcal{F}, \mu)$ be a probability space. A measurable function $X : \Omega \to \mathbb{R}_{\geqslant 0}$ is said to be a *random variable*. The *expectation* (or average value) of X with respect to the measure μ is given by the following integral:

$$\mathbb{E}[X] \stackrel{\text{def}}{=} \int_{\omega \in \Omega} X(\omega) \, \mathrm{d}\mu \, .$$

For a countable set S, a *discrete probability distribution* on S is a function $\mu : S \to [0, 1]$ such that $\sum_{s \in S} \mu(s) = 1$. The set of all probability distributions over the S is denoted $\mathit{Dist}(S)$.

3. Model Checking for Discrete-time Markov Chains

We begin with the simplest probabilistic model, that of *discrete-time Markov chains (DTMCs)*, which model systems whose behaviour at each point in time can be described by a discrete probabilistic choice over several possible outcomes. This might represent, for example, an electronic coin toss, as used to implement a randomised algorithm, or transmission of a message over an unreliable channel,



$$\mathbf{P} = \begin{pmatrix} 0 & 0.1 & 0.9 & 0 & 0 & 0 \\ 0.4 & 0 & 0 & 0.6 & 0 & 0 \\ 0 & 0 & 0.1 & 0.1 & 0.5 & 0.3 \\ 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0.7 & 0.3 \end{pmatrix}$$

Figure 1. An example DTMC and its transition probability matrix $\mathbf{P}$.

which is known to fail with a certain probability. Essentially, a DTMC can be thought of as a labelled state-transition system in which each transition is annotated with a probability value indicating the likelihood of its occurrence. For this model, we introduce PCTL, a probabilistic and reward extension of the branching time logic CTL, and briefly describe the underlying algorithms.

3.1. Discrete-time Markov Chains

A discrete time Markov chain consists of discrete states, representing the configurations of the system, and has transitions governed by (discrete) probability distributions over the target states.

Definition 1 (Discrete-time Markov chain) *A* discrete-time Markov chain *(DTMC) is a tuple* $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ *where* S *is a (countable) set of states,* $\overline{s} \in S$ *is an initial state,* $\mathbf{P} : S \times S \to [0,1]$ *is a transition probability matrix such that* $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ *for all* $s \in S$*, and* $L : S \to 2^{AP}$ *is a labelling function mapping each state to a set of atomic propositions taken from a set* AP*.*

Each element $\mathbf{P}(s, s')$ of the matrix $\mathbf{P}$ gives the probability of taking a transition from s to s', where the transition is assumed to take a discrete time-step. This means that there is no notion of real time, though reasoning about discrete time is possible through state variables keeping track of time and 'counting' transition steps. Note that there are no deadlocks, and all terminating states are modelled with a self-loop.

We can unfold a DTMC model into a set of paths. A *path* through a DTMC is a non-empty (finite or infinite) sequence of states $\omega = s_0\, s_1\, s_2 \ldots$ with $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geqslant 0$. The probability matrix $\mathbf{P}$ induces a probability space on the set of infinite paths $Path_s$, which start in the state s, using the cylinder construction [48] as follows. An observation of a finite path determines a basic event (cylinder). Let $s = s_0$. For $\pi = s_0 s_1 \ldots s_n$, we define the probability measure Pr_s^{fin} for the π-cylinder by putting $Pr_s^{\text{fin}} = 1$ if π consists of a single state, and $Pr_s^{\text{fin}} = \mathbf{P}(s_0, s_1) \cdot \mathbf{P}(s_1, s_2) \cdot \ldots \cdot \mathbf{P}(s_{n-1}, s_n)$ otherwise. This extends to a unique measure Pr_s on the set of infinite paths $Path_s$ [48].

Example 1 *Figure 1 shows an example of a DTMC* $\mathsf{D} = (S, \overline{s}, \mathbf{P}, L)$ *with 6 states* $S = \{s_0, \ldots, s_5\}$ *and initial state* $\overline{s} = s_0$*. The transition probability matrix* $\mathbf{P}$ *is also shown in Figure 1. The set of atomic propositions* AP *is* $\{a, b\}$ *and function* L *labels* s_1 *with* a *and* s_4 *with* b*.*

3.2. The Logic PCTL

Specifications for DTMC models can be written in PCTL (Probabilistic Computation Tree Logic) [35], a probabilistic extension of the temporal logic CTL.

Definition 2 (PCTL syntax) *The syntax of PCTL is as follows:*

$$\phi ::= \texttt{true} \mid a \mid \phi \wedge \phi \mid \neg\phi \mid \texttt{P}_{\bowtie p}[\psi]$$
$$\psi ::= \texttt{X}\,\phi \mid \phi\ \texttt{U}\ \phi$$

where a is an atomic proposition, $\bowtie \in \{\leqslant, <, \geqslant, >\}$ and $p \in [0,1]$.

PCTL formulas are interpreted over the states of a DTMC. PCTL replaces the CTL existential and universal quantification over paths with the *probabilistic operator* $\texttt{P}_{\bowtie p}[\cdot]$, where $p \in [0,1]$ is a *probability bound* or *threshold*. Path formulas can occur only within the scope of the probabilistic operator. Intuitively, a state s satisfies $\texttt{P}_{\bowtie p}[\psi]$ if the probability of taking a path from s that satisfies ψ is in the interval specified by $\bowtie p$. Formally, the meaning of the $\texttt{P}_{\bowtie p}[\cdot]$ operator is:

$$s \models \texttt{P}_{\bowtie p}[\psi] \Leftrightarrow Pr_s(\psi) \bowtie p \quad \text{where} \quad Pr_s(\psi) \stackrel{\text{def}}{=} Pr_s\{\omega \in Path_s \mid \omega \models \psi\}.$$

We allow standard CTL path formulas ψ, namely *next state* ($\texttt{X}\ \phi$) and *unbounded until* ($\phi_1\ \texttt{U}\ \phi_2$), as well as the usual abbreviations, e.g. $\texttt{F}\ \phi$ is equivalent to $\texttt{true}\ \texttt{U}\ \phi$. For simplicity, we omit *bounded until* ($\phi_1\ \texttt{U}^{\leqslant k}\ \phi_2$). The formula $\texttt{X}\ \phi$ is true for a path $\omega \in Path_s$ if ϕ is satisfied in the next state, and $\phi_1\ \texttt{U}\ \phi_2$ is true if ϕ_2 is satisfied at some state along the path and ϕ_1 is true up until that point. The intuition is that the probability measure of the set of ψ-paths is calculated and compared to the probability bound, yielding true or false respectively (that this set is measurable was shown in [63]). The *qualitative* fragment of PCTL comprises the formulas where p is equal to 0 or 1. Note that $\texttt{P}_{>0}[\phi_1\ \texttt{U}\ \phi_2]$ is equivalent to CTL $\exists[\phi_1\ \texttt{U}\ \phi_2]$, whereas $\texttt{P}_{\geqslant 1}[\phi_1\ \texttt{U}\ \phi_2]$ is weaker than $\forall[\phi_1\ \texttt{U}\ \phi_2]$.

We often use the *quantitative* (numerical) form of PCTL formulas, $\texttt{P}_{=?}[\cdot]$, omitting the probability bound for the outermost probabilistic operator. Such a formula evaluates to the probability value computed by the PCTL model checking algorithm. It is often useful to study and plot a range of such probability values by varying one or more parameters, either of the model or of the property.

Example 2 *Below are examples of PCTL formulas for the DTMC in Figure 1:*

- $\texttt{P}_{\leqslant 0.1}[\texttt{X}\ a]$ - *"the probability that a is true in the next state is at most* 0.1*";*
- $\texttt{P}_{=?}[\neg a\ \texttt{U}\ b]$ - *"what is the probability of reaching a b-labelled state passing only through states that do not satisfy a?"*

3.3. Model Checking for PCTL over DTMCs

The PCTL model checking algorithm [35] takes as inputs a labelled DTMC $\mathsf{D} = (S, \overline{s}, \mathbf{P}, L)$ and a PCTL formula ϕ. The algorithm proceeds, as for CTL [19],

by bottom-up traversal of the parse tree for ϕ, recursively computing the set $Sat(\phi') = \{s \in S \mid s \models \phi'\}$ of states satisfying each subformula ϕ'. Therefore, the algorithm will eventually compute the set of *all* states satisfying ϕ. To establish if a given state s satisfies ϕ, we simply check if $s \in Sat(\phi)$. For the non-probabilistic operators, the algorithm computes as for CTL:

$$\begin{aligned} Sat(\texttt{true}) &= S \\ Sat(a) &= \{s \in S \mid a \in L(s) \\ Sat(\neg\phi) &= S \setminus Sat(\phi) \\ Sat(\phi_1 \wedge \phi_2) &= Sat(\phi_1) \cap Sat(\phi_2)\,. \end{aligned}$$

For the probabilistic operator $\mathtt{P}_{\bowtie p}[\psi]$, we have:

$$Sat(\mathtt{P}_{\bowtie p}[\psi]) = \{s \in S \mid Pr_s(\psi) \bowtie p\}\,.$$

It is convenient to view the DTMC as the matrix $\mathbf{P}$ and $Sat(\phi)$ as a *column vector* $\underline{\phi} : S \longrightarrow \{0,1\}$ given by $\underline{\phi}(s) = 1$ if $s \models \phi$ and 0 otherwise. Consider the next state operator. The probabilities for all states can then be computed by a single matrix-by-vector multiplication, written in vector notation $\underline{Pr}(\mathtt{X}\,\phi) = \mathbf{P} \cdot \underline{\phi}$.

For the path formula $\phi_1\ \mathtt{U}\ \phi_2$, the probabilities $Pr_s(\phi_1\ \mathtt{U}\ \phi_2)$ are obtained as the unique solution of the *linear equation system* in variables $\{x_s \mid s \in S\}$:

$$x_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \sum_{s' \in S} \mathbf{P}(s,s') \cdot x_{s'} & \text{if } s \in S^{?} \end{cases}$$

where $S^{no} \stackrel{\text{def}}{=} Sat(\mathtt{P}_{\leqslant 0}[\phi_1\ \mathtt{U}\ \phi_2])$ and $S^{yes} \stackrel{\text{def}}{=} Sat(\mathtt{P}_{\geqslant 1}[\phi_1\ \mathtt{U}\ \phi_2])$ denote the sets of *all* states that satisfy $\phi_1\ \mathtt{U}\ \phi_2$ with probability exactly 0 and 1, respectively, and $S^{?} = S \setminus (S^{no} \cup S^{yes})$.

The sets S^{no} and S^{yes} are precomputed using conventional fixed point computations. For example, for S^{no}, we first compute the set of states from which we can reach, with positive probability, a ϕ_2-state passing only through states satisfying ϕ_1, and then subtract this set from S. Since the values for the precomputed states are known (0 or 1), the solution of the resulting linear equation system in $|S^{?}|$ variables can be obtained by any direct method (e.g. Gaussian elimination) or iterative method (e.g. Jacobi, Gauss-Seidel). For qualitative PCTL properties, it suffices to use these precomputation algorithms alone. Note that the precomputation algorithms determine the exact probability in case it is 0 or 1, thus avoiding the problem of round-off errors that are typical for numerical computation.

Example 3 *Consider the DTMC* D *from Example 1 (see Figure 1) and the PCTL formula* $\mathtt{P}_{>0.8}[\neg a\ \mathtt{U}\ b]$. *We have* $S^{no} = \{s_1, s_3\}$ *and* $S^{yes} = \{s_4, s_5\}$. *These sets, and the resulting linear equation system, are shown in Figure 2. This yields the solution* $(0.8, 0, \frac{8}{9}, 0, 1, 1)$, *and we see that* $Sat(\mathtt{P}_{>0.8}[\neg a\ \mathtt{U}\ b]) = \{s_2, s_4, s_5\}$.

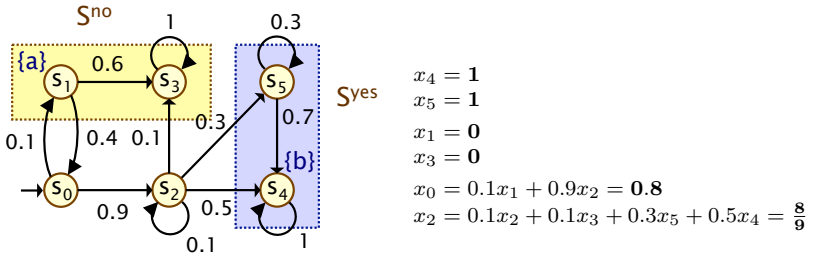


Figure 2. Determining the probabilities $Pr_s(\neg a \,\mathtt{U}\, b)$ for the DTMC D from Example 1.
Left: the DTMC, labelled with the sets S^{no} and S^{yes} from precomputation;
Right: linear equation system to determine probabilities for the remaining states.

3.4. Extending DTMCs and PCTL with Rewards

In this section we introduce *rewards* which can be used to annotate DTMCs with information about resources and their usage, for example the energy consumption or the number of lost messages. We consistently use the terminology *rewards* but, often, these will be used to model *costs*.

Let $\mathsf{D} = (S, \overline{s}, \mathbf{P}, L)$ be a DTMC. A *reward structure* is a pair (r_s, r_t) of functions: a *state reward* function $r_s : S \to \mathbb{R}_{\geqslant 0}$ mapping a state to the reward acquired per time-step, and a *transition reward* function $r_t : S \times S \to \mathbb{R}_{\geqslant 0}$, mapping each transition to the reward acquired as the transition is taken. The rewards can be interpreted in two ways: *instantaneous* (omitted for simplicity) or *cumulated* over system execution.

Example 4 *The reward structure "number of messages lost" is defined by r_s equal to zero and r_t mapping each transition that loses a message to 1. Reward structure "power consumption" is given by r_s equal to per-time-step energy consumption in each state and r_t equal to the energy cost of each transition.*

The logic PCTL is extended [53] to allow for the reward properties by the addition of the *reward operator* $\mathtt{R}_{\bowtie r}[\cdot]$ and the following state formulas:

$$\mathtt{R}_{\bowtie r}[\mathtt{C}^{\leqslant k}] \;\big|\; \mathtt{R}_{\bowtie r}[\mathtt{F}\,\phi]$$

where $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $r \in \mathbb{R}_{\geqslant 0}$, $k \in \mathbb{N}$ and ϕ is a PCTL state formula.

Intuitively, a state s satisfies $\mathtt{R}_{\bowtie r}[\mathtt{C}^{\leqslant k}]$ (*cumulative* reward) if, from state s, the expected reward cumulated over k time-steps satisfies $\bowtie r$; and $\mathtt{R}_{\bowtie r}[\mathtt{F}\,\phi]$ (*reachability* reward) is true if, from state s, the expected reward cumulated before a state satisfying ϕ is reached meets $\bowtie r$. Formally, the semantics of the reward operator is defined using the expectation of an appropriate random variable:

$$\begin{array}{lcl} s \models \mathtt{R}_{\bowtie r}[\mathtt{C}^{\leqslant k}] & \Leftrightarrow & \mathbb{E}_s(X_{\mathtt{C}^{\leqslant k}}) \bowtie r \\ s \models \mathtt{R}_{\bowtie r}[\mathtt{F}\,\phi] & \Leftrightarrow & \mathbb{E}_s(X_{\mathtt{F}\,\phi}) \bowtie r \end{array}$$

for any $s \in S$, $k \in \mathbb{N}$, $r \in \mathbb{R}_{\geqslant 0}$ and PCTL formula ϕ, and where $\mathbb{E}_s$ denotes expectation with respect to the probability measure Pr_s. The random variables

$X_{\mathtt{C}^{\leqslant k}}, X_{\mathtt{F}\,\phi} : Path_s \to \mathbb{R}_{\geqslant 0}$ corresponding to the two forms of the reward operator are defined, for any path $\omega = s_0 s_1 s_2 \cdots \in Path_s$, as follows:

$$X_{\mathtt{C}^{\leqslant k}}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } k = 0 \\ \sum_{i=0}^{k-1} r_s(s_i) + r_t(s_i, s_{i+1}) & \text{otherwise} \end{cases}$$

$$X_{\mathtt{F}\,\phi}(\omega) \stackrel{\text{def}}{=} \begin{cases} 0 & \text{if } s_0 \models \phi \\ \infty & \text{if } \forall i \in \mathbb{N}.\, s_i \not\models \phi \\ \sum_{i=0}^{\min\{j | s_j \models \phi\}-1} r_s(s_i) + r_t(s_i, s_{i+1}) & \text{otherwise.} \end{cases}$$

As for the probabilistic operator, if the outermost operator of a PCTL formula is $\mathtt{R}_{\bowtie r}[\cdot]$, we can omit the bound $\bowtie r$ and compute the expected value instead. This also enables a range of such values to be obtained by varying one or more parameters, either of the model or of the property.

Example 5 *Below are some examples of reward based specifications:*

- $\mathtt{R}_{=?}[\mathtt{C}^{\leqslant 10}]$ - *what is the expected number of messages lost within the first 10 time-steps?;*
- $\mathtt{R}_{\leqslant 5}[\mathtt{F}\ \mathit{end}]$ - *the expected time to termination is at most 5.*

Model checking of the reward operator is similar to computing probabilities for the probabilistic operator, and follows through the solution of recursive equations (for $\mathtt{R}_{\bowtie r}[\mathtt{C}^{\leqslant k}]$) or a system of linear equations (for $\mathtt{R}_{\bowtie r}[\mathtt{F}\,\phi]$). For more details on this, and other aspects of probabilistic model checking for DTMCs, see e.g. [53,9].

3.5. More on Model Checking for DTMCs

The time complexity for PCTL model checking over DTMCs, including the reward operator, is linear in the size of the formula $|\phi|$ (number of logical connectives and temporal operators) and polynomial in the size of the state space $|S|$ [35].

DTMCs can also be verified against LTL properties, but the verification method is quite different. The LTL formula is first translated into a Rabin automaton and then model checking reduces to the computation of probabilistic reachability of bottom strongly connected components on the product of the DTMC and the formula automaton [9]. The overall complexity for LTL is doubly exponential in $|\phi|$ and polynomial in in $|S|$, but can be reduced to a single exponential.

4. Model Checking for Markov Decision Processes

We now explain how to model check *Markov decision processes* (MDPs), which generalise discrete-time Markov chains with the addition of *nondeterminism*. Probability is employed to quantify aspects of system behaviour where probability distributions are known. In contrast, nondeterminism is used to model *unknown environments*, where such distributions are not known. It is also used to model *concurrency*, where it represents the different possible interleavings of multiple components operating in parallel. Alternatively, we can use nondeterminism to capture the possible ways that a *controller* can influence the behaviour of the system, e.g. in planning and robotics.

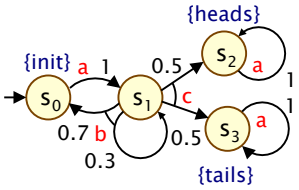


Figure 3. An example MDP.

4.1. Markov Decision Processes

Formally, we define a Markov decision process as follows.

Definition 3 (Markov decision process) *A* Markov decision process *(MDP) is a tuple* $\mathsf{M}=(S,\overline{s},Act,Steps,L)$ *where* S *is a set of states,* $\overline{s}\in S$ *is an initial state,* Act *is an alphabet of actions,* $Steps : S\times Act \to Dist(S)$ *is a (partial) probabilistic transition function and* $L : S \to 2^{AP}$ *is a labelling function mapping each state to a set of atomic propositions taken from a set* AP*.*

In an MDP, several *actions* may be available in a given state s, each corresponding to a probability distribution. We denote this set by $A(s) = \{a \in Act \mid Steps(s,a) \text{ is defined}\}$. As for DTMCs, we disallow deadlocks, and hence assume that $A(s)$ is non-empty for all $s \in S$. The behaviour of an MDP M is as follows. Firstly, a choice between one or more actions from the alphabet Act is made *nondeterministically.* Secondly, for the chosen action a, a successor state s' is chosen randomly, according to the probability distribution $Steps(s,a)$, i.e. the probability that a transition to s' occurs is $Steps(s,a)(s')$.

An *infinite path* through an MDP is a sequence $\omega=s_0a_0s_1a_1\ldots$ where $s_i \in S$, $a_i \in A(s_i)$ and $Steps(s_i,a_i)(s_{i+1})>0$ for all $i \in \mathbb{N}$. A *finite path* $\pi=s_0a_0s_1\ldots s_n$ is a prefix of an infinite path ending in a state. We denote by $Path_s$ and $Path_s^{\text{fin}}$ the sets of all infinite and finite paths from state s, respectively, and by $Path$ and $Path^{\text{fin}}$ the corresponding sets of paths from any state. For a finite path π, the last state of π is denoted $last(\pi)$.

Example 6 *Figure 3 shows an example MDP* $\mathsf{M}=(S,\overline{s},Act,Steps,L)$ *with states* $S=\{s_0,\ldots,s_3\}$*, initial state* s_0 *and alphabet of actions* $\{a,b,c\}$*. State* s_1*, for example, has a nondeterministic choice between two actions,* b *and* c*.*

To reason formally about MDPs, we need a probability space over infinite paths. However, a probability space can only be constructed once all the nondeterminism has been resolved. Each possible resolution of nondeterminism is represented by an *adversary*, also called a *policy*, which is responsible for choosing an action in each state of the MDP, based on the history of its execution so far.

Definition 4 (Adversary) *An* adversary *of an MDP* $\mathsf{M}=(S,\overline{s},Act,Steps,L)$ *is a function* $\sigma : Path^{\text{fin}} \to Dist(Act)$ *such that* $\sigma(\pi)(a)>0$ *only if* $a \in A(last(\pi))$*. An adversary* σ *is* memoryless *if* $\sigma(\pi)$ *depends only on* $last(\pi)$ *and* deterministic *if the distribution* $\sigma(\pi)$ *always selects a single action with probability 1.*

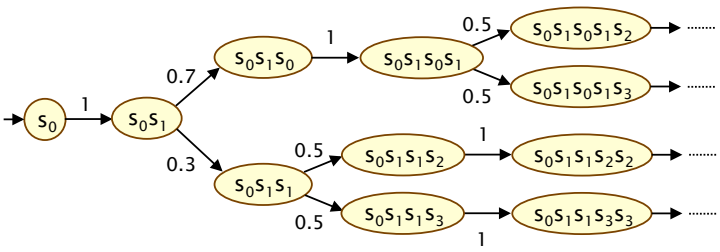


Figure 4. The induced DTMC for an adversary of the MDP in Figure 3.

The set of *all* adversaries of an MDP M is denoted Adv. Under a particular adversary $\sigma \in Adv$, the behaviour of M is fully probabilistic and can be captured by an *induced DTMC*, denoted M^σ, each state of which is a finite path of M.

Definition 5 (Induced DTMC) *For an MDP* $\mathsf{M} = (S, \overline{s}, Act, Steps, L)$ *and adversary* σ*, the* induced DTMC *is* $\mathsf{M}^\sigma = (Path^{\text{fin}}, \overline{s}, \mathbf{P}, L')$ *where:*

- *for any* $\pi, \pi' \in Path^{\text{fin}}$*:*

$$\mathbf{P}(\pi, \pi') = \begin{cases} \sigma(\pi)(a) \cdot Steps(last(\pi), a)(s) & \text{if } \pi' = \pi a s,\ a \in A(last(\pi)) \\ 0 & \text{otherwise;} \end{cases}$$

- $L'(\pi) = L(last(\pi))$ *for all* $\pi \in Path^{\text{fin}}$.

Notice that there is a one-to-one mapping between the infinite paths of the DTMC M^σ and the infinite paths of MDP M when under the control of adversary σ. This means that the DTMC yields, for a start state s, a probability space, denoted Pr^σ_s over these infinite paths. The induced DTMC M^σ has a (countably) infinite number of states. However, in the case of memoryless adversaries, its state space is isomorphic to S and M^σ can be reduced to an $|S|$-state DTMC.

Example 7 *Consider the MDP* M *from Example 6 (shown in Figure 3) and the (deterministic, but non-memoryless) adversary* σ*, which picks action* b *the first time that state* s_1 *is reached, and then action* c *the second time. The induced DTMC* M^σ *is shown in Figure 4.*

4.2. PCTL Model Checking over MDPs

Probabilistic statements about MDPs typically involve quantification over adversaries, so as to establish that some specified event is observed *for all* possible adversaries. The logic PCTL, for example, is defined for MDPs as for DTMCs [14], the key difference being that the semantics of the probabilistic operator contains explicit universal quantification:

$$s \models \mathsf{P}_{\bowtie p}[\psi] \quad \Leftrightarrow \quad Pr^\sigma_s\{\omega \in Path_s \mid \omega \models \psi\} \bowtie p \ \text{ for all } \sigma \in Adv.$$

The algorithm for PCTL model checking proceeds as for DTMCs, except for the probabilistic operator. For $\mathsf{P}_{\triangleright p}[\psi]$ where $\triangleright \in \{\geqslant, >\}$, this reduces to the

calculation of the *minimum probability* $Pr^{\min}_s(\psi)$. The case of $\mathtt{P}_{\lhd p}[\psi]$ for $\lhd \in \{\leqslant, <\}$ is dual, via the *maximum probability* $Pr^{\max}_s(\psi)$:

$$Sat(\mathtt{P}_{\rhd p}[\psi]) = \{s \in S \mid Pr^{\min}_s(\psi) \rhd p\}$$
$$Sat(\mathtt{P}_{\lhd p}[\psi]) = \{s \in S \mid Pr^{\max}_s(\psi) \lhd p\}$$

where $Pr^{\min}_s(\psi) = \inf_{\sigma \in Adv}\{Pr^{\sigma}_s(\psi)\}$ and $Pr^{\max}_s(\psi) = \sup_{\sigma \in Adv}\{Pr^{\sigma}_s(\psi)\}$.

To describe the computation of these values, we restrict our attention to the case of minimum probabilities. If $\psi = \mathtt{X}\,\phi$, we have:

$$Pr^{\min}_s(\mathtt{X}\,\phi) = \min_{a \in A(s)}\big[\textstyle\sum_{s' \in Sat(\phi)} Steps(s,a)(s')\big]\,.$$

For $\psi = \phi_1\ \mathtt{U}\ \phi_2$, the minimum probabilities are the unique solution of:

$$Pr^{\min}_s(\psi) = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ \min_{a \in A(s)}\big[\sum_{s' \in S} Steps(s,a)(s') \cdot Pr^{\min}_{s'}(\psi)\big] & \text{if } s \in S^{?} \end{cases}$$

with S^{no} and S^{yes} denoting the sets of states where the minimum probability is respectively 0 and 1, precomputed in a similar fashion to the DTMC case via a fixpoint (see e.g. [28]). The computation of these probabilities can be performed in several different ways; we detail three such methods below.

Linear programming. The minimum probabilities $Pr^{\min}_s(\phi_1\ \mathtt{U}\ \phi_2)$ for $s \in S$ can be obtained as the unique solution of the following linear program (LP):

$$\begin{array}{ll} \text{maximise } \textstyle\sum_{s \in S} x_s \text{ subject to the constraints:} & \\ x_s = 0 & \text{for all } s \in S^{no} \\ x_s = 1 & \text{for all } s \in S^{yes} \\ x_s \leqslant \textstyle\sum_{s' \in S} Steps(s,a)(s') \cdot x_{s'} & \text{for all } s \in S^{?} \text{ and } a \in A(s)\,. \end{array}$$

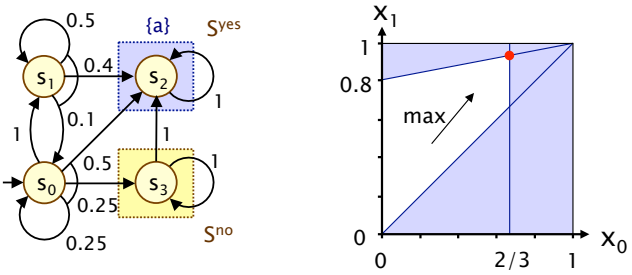
This can be solved using standard techniques such as the simplex, ellipsoid or branch-and-cut methods.

Value iteration. The minimum probabilities can also be approximated iteratively, since $Pr^{\min}_s(\phi_1\ \mathtt{U}\ \phi_2) = \lim_{n\to\infty} x^{(n)}_s$, as follows:

$$x^{(n)}_s = \begin{cases} 0 & \text{if } s \in S^{no} \\ 1 & \text{if } s \in S^{yes} \\ 0 & \text{if } s \in S^{?} \text{ and } n = 0 \\ \min_{a \in A(s)}\left\{\sum_{s' \in S} Steps(s,a)(s') \cdot x^{(n-1)}_{s'}\right\} & \text{if } s \in S^{?} \text{ and } n > 0\,. \end{cases}$$

In practice, the iterative computation is terminated when the values $x^{(n)}_s$ have converged sufficiently with respect to a given level of precision $\varepsilon > 0$.

Policy iteration. This method iterates over adversaries (i.e. policies), as opposed to probability values. It starts with an arbitrary, memoryless adversary σ and computes the probability p^{σ}_s of satisfying $\phi_1\ \mathtt{U}\ \phi_2$ from each state s under σ. This



(a) Example MDP, labelled with the sets S^{no} and S^{yes} from precomputation.

(b) Linear programming problem yielding minimum probabilities of reaching a-labelled states.

Figure 5. Determining the probabilities $Pr_s^{\min}(\mathtt{F}\ a)$ for an example MDP.

is done on the induced DTMC, e.g., by solving a linear equation system. The adversary σ can be improved to adversary σ' as follows:

$$\sigma'(s) = \arg\min_{a \in A(s)} \sum_{s' \in S} Steps(s,a)(s') \cdot p^{\sigma}_{s'}.$$

The process is repeated until there is no change, and hence the optimal adversary is found. To ensure termination, adversary choices should only be changed when there is a strict improvement in the probability for s.

Example 8 *Consider the MDP in Figure 5(a) and the probabilities* $Pr_s^{\min}(\mathtt{F}\, a) = Pr_s^{\min}(\mathtt{true\ U}\ a)$. *We have* $S^{no} = \{s_3\}$ *and* $S^{yes} = \{s_2\}$, *which are also marked in Figure 5(a). We need to compute the minimum probabilities for the remaining two states in* $S^{?} = \{s_0, s_1\}$. *First, we solve a linear program. We have:*

maximise $x_0 + x_1$ *subject to the constraints:*

$$\begin{aligned} &x_0 \leqslant x_1 \\ &x_0 \leqslant 0.25 \cdot x_0 + 0.5 \\ &x_1 \leqslant 0.1 \cdot x_0 + 0.5 \cdot x_1 + 0.4 \\ &x_2 = 1 \\ &x_3 = 0 \end{aligned}$$

The optimisation problem (restricted to variables x_0 *and* x_1*) is illustrated graphically in Figure 5(b), with the central white region representing the intersection of the constraints. The required solution is shown by the dot in the top right corner, which is* $(x_0, x_1) = (\frac{2}{3}, \frac{14}{15})$, *so we have* $\underline{Pr}^{\min} = (\frac{2}{3}, \frac{14}{15}, 1, 0)$.

Alternatively, we can compute the probabilities using value iteration. Figure 6 (left) shows the values computed for $x_s^{(n)}$, *converging after 21 steps when the maximum difference between values in successive iterations drops below* $\varepsilon = 10^{-6}$. *The right-hand side of Figure 6 shows the same values (for states* s_0 *and* s_1*) plotted over the linear program from above.*

Finally, we observe that the points $(\frac{2}{3}, \frac{14}{15})$ *and* $(1, 1)$ *in the linear program correspond to the two memoryless adversaries obtained by taking the two different choices in state* s_0. *If policy iteration were applied instead, starting with the second*

$$
\begin{array}{rl}
 & [\ x_0^{(n)},\ x_1^{(n)},\ x_2^{(n)},\ x_3^{(n)}\ ] \\
n=0: & [\ 0.000000,\ 0.000000,\ 1,\ 0\ ] \\
n=1: & [\ 0.000000,\ 0.400000,\ 1,\ 0\ ] \\
n=2: & [\ 0.400000,\ 0.600000,\ 1,\ 0\ ] \\
n=3: & [\ 0.600000,\ 0.740000,\ 1,\ 0\ ] \\
n=4: & [\ 0.650000,\ 0.830000,\ 1,\ 0\ ] \\
n=5: & [\ 0.662500,\ 0.880000,\ 1,\ 0\ ] \\
n=6: & [\ 0.665625,\ 0.906250,\ 1,\ 0\ ] \\
n=7: & [\ 0.666406,\ 0.919688,\ 1,\ 0\ ] \\
n=8: & [\ 0.666602,\ 0.926484,\ 1,\ 0\ ] \\
n=9: & [\ 0.666650,\ 0.929902,\ 1,\ 0\ ] \\
 & \ldots \\
n=20: & [\ 0.666667,\ 0.933332,\ 1,\ 0\ ] \\
n=21: & [\ 0.666667,\ 0.933332,\ 1,\ 0\ ] \\
\approx & [\ 2/3,\ 14/15,\ 1,\ 0\ ]
\end{array}
$$

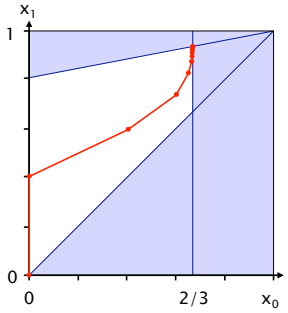


Figure 6. Left: Vectors from value iteration on the reachability problem from Figure 5; Right: Value iteration points plotted against the linear program

one (which gives $x_0 = x_1 = 1$), then the first round of improvement of policy iteration would yield the first, optimal adversary.

4.3. Extending MDPs with Rewards

As for DTMCs, we can augment an MDP with rewards, useful for representing quantitative system information such as the energy consumption or the number of packets sent. A *reward structure* for an MDP $\mathsf{M} = (S, \overline{s}, Act, Steps, L)$ is a pair (r_s, r_a) comprising a *state reward* function $r_s : S \to \mathbb{R}_{\geqslant 0}$ and an *action reward* function $r_a : S \times Act \to \mathbb{R}_{\geqslant 0}$.

The logic PCTL is extended, as for DTMCs, with the reward operator $\mathtt{R}_{\bowtie r}[\cdot]$ (and its quantitative form $\mathtt{R}_{\min=?}[\cdot]$ or $\mathtt{R}_{\max=?}[\cdot]$). The difference is that now we compute minimum (or maximum) reward values as expectations.

Example 9 *Below are some examples of reward based specifications:*

- $\mathtt{R}_{\min=?}[\mathtt{C}^{\leqslant 3600}]$ - *the minimum expected energy consumption over one hour;*
- $\mathtt{R}_{\max=?}[\mathtt{F}\ end]$ - *the maximum expected time to termination.*

4.4. More on Model Checking for MDPs

The time complexity for PCTL model checking over an MDP is (again) linear in the size of the formula $|\phi|$ and polynomial in the size of the state space $|S|$. Like for DTMCs, MDPs can also be verified against LTL properties via the construction of the product with a deterministic Rabin automaton for the LTL formula. Model checking reduces to the computation of maximum reachability probabilities of a set of end components of the product MDP. See e.g. [9,28] for details. The overall complexity for LTL is doubly exponential in $|\phi|$ and polynomial in $|S|$; unlike for DTMCs, this *cannot* be reduced to a single exponential.

MDPs can also be verified under *fairness* conditions [10,7]. Fairness is necessary, for example, in the context of parallel composition to ensure progress for

each concurrent component whenever possible. For modelling and verification of probabilistic systems comprising multiple components, the closely related model of *probabilistic automata* has been developed, along with rich theories for compositional modelling and analysis [61].

5. Quantitative Abstraction Refinement

The techniques described in the preceding two sections can be used to establish a wide range of useful properties of systems modelled as DTMCs and MDPs. Furthermore, these methods are efficient: typically the time complexity is polynomial in the size of the state space of the model. In practice, though, one of the principal challenges in applying probabilistic model checking to real-life systems is *scalability*: the models that need to be constructed and analysed are often simply too big for the process to be feasible. This phenomenon, commonly called the *state-space explosion problem*, affects all verification approaches that rely on an exhaustive analysis of a model's state space.

In this section, we describe the use of *abstraction* as a mechanism for overcoming the state-space explosion problem. Abstraction techniques work by hiding aspects of the system being modelled that are not relevant to the property currently being verified, resulting in a smaller *abstract model*. Let us refer to the states $s \in S$ of the model that is to be abstracted as *concrete states*. We will define an abstraction of the model based on a partition $\mathcal{A}$ of these concrete states, with each subset in the partition referred to as an *abstract state* $a \in \mathcal{A}$. We then build an abstract model, whose state space is the set of abstract states $\mathcal{A}$.

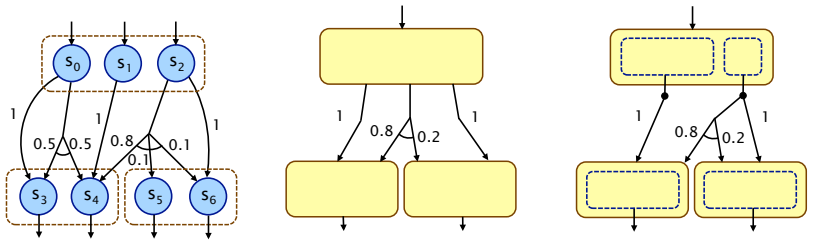
5.1. Abstracting MDPs as Stochastic Games

In this tutorial, we will focus on the problem of building abstractions for Markov decisions processes (MDPs), since this is the more general of the two models that we have considered. One approach to defining an abstraction of an MDP M, based on a partition $\mathcal{A}$ of its states into abstract states, is to use an *existential abstraction* [22], which takes the form of another MDP $\mathsf{M}_{\mathcal{A}}$ with state space $\mathcal{A}$.

The existential abstraction $\mathsf{M}_{\mathcal{A}}$ is built by *lifting* each probability distribution μ (over S) in MDP M to a distribution $\mu_{\mathcal{A}}$ over $\mathcal{A}$, i.e. for each $a \in \mathcal{A}$, $\mu_{\mathcal{A}}(a) = \sum_{s \in a} \mu(a)$. For each abstract state a in $\mathsf{M}_{\mathcal{A}}$, $\mathit{Steps}(a)$ then contains the distributions $\mu_{\mathcal{A}}$ for all distributions μ from any state s such that $s \in a$.

Example 10 *Figure 7(a) shows a fragment of an MDP* M*, annotated with a partition* $\mathcal{A}$ *of its states. Figure 7(b) shows the corresponding fragment of the existential abstraction* $\mathsf{M}_{\mathcal{A}}$*.*

Formally, the MDP M and its abstraction $\mathsf{M}_{\mathcal{A}}$ are related through *strong simulation* [62], which means that the abstraction preserves the "safe" fragment of PCTL. Furthermore, $\mathsf{M}_{\mathcal{A}}$ yields bounds on the reachability probabilities for M. More precisely, letting t be an atomic proposition labelling a *target* abstract state in $\mathsf{M}_{\mathcal{A}}$, we can obtain a lower bound on the minimum probability of reaching t and an upper bound on the maximum probability of reaching it:



(a) MDP M labelled with partition $\mathcal{A}$ of its states. (b) MDP representing the existential abstraction $\mathsf{M}_{\mathcal{A}}$. (c) The stochastic game abstraction $\mathsf{G}_{\mathcal{A}}$.

Figure 7. Fragments of an MDP and corresponding abstractions.

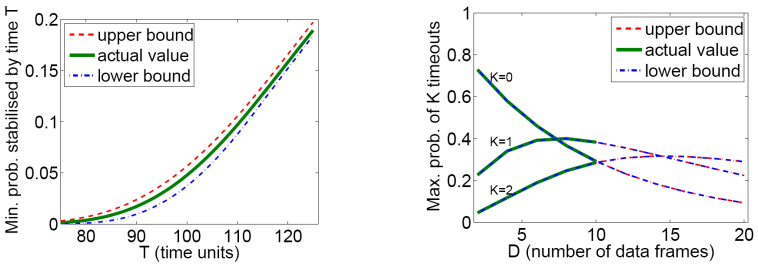
$$Pr^{\min}_{\mathsf{M}_{\mathcal{A}}}(\mathsf{F}\, t) \leqslant Pr^{\min}_{\mathsf{M}}(\mathsf{F}\, t) \quad \text{and} \quad Pr^{\max}_{\mathsf{M}}(\mathsf{F}\, t) \leqslant Pr^{\max}_{\mathsf{M}_{\mathcal{A}}}(\mathsf{F}\, t)$$

An alternative approach to abstraction was presented in [51], based on the construction of a *stochastic two-player game* $\mathsf{G}_{\mathcal{A}}$ to abstract the MDP M using state space partition $\mathcal{A}$. Stochastic two-player games are a generalisation of MDPs that include two distinct forms of nondeterminism, each controlled by a separate player. The key idea behind the stochastic game abstraction is to use player 1 to represent the nondeterminism introduced through the process of abstraction and player 2 to represent nondeterminism from the original MDP M.

Example 11 *Figure 7(c) illustrates the construction of the stochastic game abstraction $\mathsf{G}_{\mathcal{A}}$ for the same MDP M used in Example 10 (and shown in Figure 7(a)). Like for existential abstraction, distributions from M are lifted from S to $\mathcal{A}$, except that they are now grouped according to the concrete states of M from which they emanate. For example, the right-hand black dot in Figure 7(c) shows the (lifted) distributions from state s_2 of M. The left-hand dot shows the distributions for states s_0 and s_1, which become identical when lifted to $\mathcal{A}$. The corresponding groups of states for each dot are indicated by the rounded boxes with dashed edges.*

The game is played as follows. The abstract states, shown as rounded boxes with solid edges, are controlled by player 1. He chooses one of the successor player 2 states, represented by the black dots. Player 2 then chooses one of the distributions emanating from the player 2 state. The chosen distribution is used to randomly select the next abstract state.

An important advantage of the stochastic game abstraction over the existential abstraction is that it yields both lower and upper bounds for either minimum or maximum reachability probabilities. From the stochastic game, we can determine the *optimal probabilities* for each player, e.g. the maximum probability of reaching a t-labelled state that player 1 can achieve, assuming that player 2 is trying to minimise it, denoted $Pr^{\max,\min}_{\mathsf{G}_{\mathcal{A}}}(\mathsf{F}\, t)$. The dual probability $Pr^{\min,\max}_{\mathsf{G}_{\mathcal{A}}}(\mathsf{F}\, t)$ is defined similarly. We can also determine either the minimum or maximum probability of reaching t, assuming that the two players collaborate, denoted $Pr^{\min,\min}_{\mathsf{G}_{\mathcal{A}}}(\mathsf{F}\, t)$ and $Pr^{\max,\max}_{\mathsf{G}_{\mathcal{A}}}(\mathsf{F}\, t)$, respectively. All four values can be computed (approximately) with an adapted version of the value iteration method described



(a) Self-stabilisation protocol. (b) Sliding window protocol.

Figure 8. Graphs illustrating the lower and upper bounds obtained from the stochastic game abstractions of MDPs from two large case studies.

in Section 4.2 for MDPs. The values, when computed for the stochastic game abstraction $\mathsf{G}_{\mathcal{A}}$ of an MDP M, yield lower and upper bounds as follows:

$$Pr^{\min,\min}_{\mathsf{G}_{\mathcal{A}}}(\mathtt{F}\, t) \leqslant Pr^{\min}_{\mathsf{M}}(\mathtt{F}\, t) \leqslant Pr^{\max,\min}_{\mathsf{G}_{\mathcal{A}}}(\mathtt{F}\, t)$$
$$Pr^{\min,\max}_{\mathsf{G}_{\mathcal{A}}}(\mathtt{F}\, t) \leqslant Pr^{\max}_{\mathsf{M}}(\mathtt{F}\, t) \leqslant Pr^{\max,\max}_{\mathsf{G}_{\mathcal{A}}}(\mathtt{F}\, t)$$

Figure 8 shows some typical examples of these bounds, obtained from real case studies [45]. Part (a) plots results for the "minimum probability of termination within time T" on a model of Israeli & Jalfon's self-stabilisation protocol, for a range of values of T. In this case, the MDP has 1,048,575 states, whereas the stochastic game abstraction has only 627. Part (b) shows results for "the maximum probability of K time-outs" on a model of a sliding window protocol, varying both K and the number frames sent D. In this case, we see that the lower and upper bounds are tight, yielding the exact answers. Also, in this example, the use of *predicate abstraction* allows us to build and analyse abstractions for MDPs that are too large to verify directly. Thus, we see results on the right-hand side of the graph that could not have been obtained without the use of abstraction.

5.2. Quantitative Abstraction Refinement

Although the stochastic game abstractions presented above were constructed and analysed in a fully automated fashion, one crucial part of the process was still performed manually: the specification of the abstract state space A. The practical applicability and usefulness of the abstraction techniques are completely reliant on being able to determine an appropriate partition A of the concrete state space, i.e. one that is coarse enough to yield a small abstract model, but which gives lower and upper bounds that are close enough to provide useful information.

Here, we discuss a way to construct such a partition A in a fully automatic fashion, using *abstraction refinement*. This is inspired by the successful *counterexample-guided abstraction refinement* (CEGAR) approach [20], developed to build abstractions for non-probabilistic model checking. The idea is to start with a simple, coarse partition, build the corresponding abstraction, and then iteratively refine the partition, resulting in increasingly precise abstractions. In CEGAR, the refinement step is driven by counterexamples obtained by model

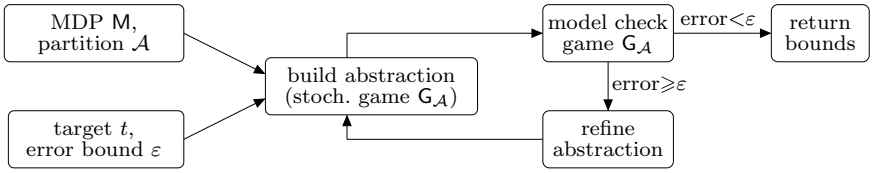


Figure 9. Quantitative abstraction-refinement loop.

checking the abstractions produced at each iteration, and the iterations of refinement are terminated when the current abstraction can be used either to verify or refute the property being verified.

Here, we describe *quantitative abstraction refinement* [47], which can be seen as a quantitative analogue of CEGAR, applied to the stochastic game abstraction approach of [51]. The idea is that the difference between the lower and upper bounds obtained from an abstraction (which we call the "error") *quantifies* the precision of the abstraction. If the error is too high, the abstraction should be refined (i.e. a finer partition A used).

Figure 9 shows a quantitative abstraction-refinement loop used to automatically construct stochastic game abstractions of an MDP M. Starting with an initial (coarse) partition, abstractions are repeatedly constructed, analysed and refined until the difference between the lower and upper bounds obtained for the property of interest (say, the minimum or maximum probability of reaching t-labelled states) falls below some threshold ε.

The applicability of quantitative abstraction refinement was initially demonstrated in [47], on a simple explicit-state implementation, applied to a set of benchmark models. Subsequently, the approach has been successfully applied to the verification of:

- probabilistic real-time systems, modelled using probabilistic timed automata [54] (see the next section for a more in-depth discussion);
- probabilistic software, based on the use of ANSI-C code, augmented with probabilistic commands [46];
- concurrent probabilistic programs, as analysed by the probabilistic abstraction tool PASS [29];
- probabilistic programs over arbitrary abstract domains [27];
- probabilistic hybrid automata [31].

Several other methods for abstraction refinement of probabilistic systems have also been proposed [22,41,25,16]; see e.g. [47] for a discussion of the differences between these approaches.

6. Probabilistic Timed Automata

The two models that we presented in the first half of this tutorial, DTMCs and MDPs, both use a *discrete* model of time, i.e. a system execution is modelled as a sequence of discrete time-steps. In this section, we discuss modelling and verification of probabilistic *real-time* systems, which use a continuous (dense) model of time. These systems will be modelled by *probabilistic timed automata*.

Another popular model incorporating a continuous notion of time is *continuous-time Markov chains* (CTMCs), an extension of DTMCs where the delays between state transitions are represented by exponential distributions. See, for example, [8,53] for overviews of probabilistic model checking techniques for CTMCs.

6.1. Probabilistic Timed Automata

Probabilistic timed automata (PTAs) [42,59] model systems that exhibit probabilistic, nondeterministic and real-time characteristics. They can be seen as an extension of MDPs with *clocks*, real-valued variables whose values increase simultaneously over time. Like in the classic model of timed automata [2], states and transitions of PTAs can be labelled with *invariants* and *guards*, that is, predicates on clocks respectively dictating how long to remain in a state and when transitions can occur. Transitions between states (represented, as in MDPs, by probability distributions) can also reset the values of one or more clocks.

To specify invariants and guards, we use *clock constraints*. Assuming a set of clocks $\mathcal{X}$, the set of allowable clock constraints, denoted $CC(\mathcal{X})$, is defined by the following grammar:

$$\chi ::= \texttt{true} \mid x \leqslant d \mid c \leqslant x \mid x{+}c \leqslant y{+}d \mid \neg\chi \mid \chi \wedge \chi$$

where $x, y \in \mathcal{X}$ and $c, d \in \mathbb{N}$. A PTA can then be formally defined as follows.

Definition 6 (Probabilistic timed automaton) *A* probabilistic timed automaton *(PTA) is defined by a tuple* $\mathsf{P}{=}(Loc, \bar{l}, \mathcal{X}, Act, inv, enab, prob, L)$ *where:*

- Loc *is a finite set of* locations*;*
- $\bar{l} \in Loc$ *is an* initial location*;*
- $\mathcal{X}$ *is a finite set of* clocks*;*
- Act *is a finite set of actions;*
- $inv : Loc \to CC(\mathcal{X})$ *is the* invariant condition*;*
- $enab : Loc{\times}Act \to CC(\mathcal{X})$ *is the* enabling condition*;*
- $prob : Loc{\times}Act \to Dist(2^{\mathcal{X}}{\times}Loc)$ *is a (partial)* probabilistic transition function*;*
- $L : Loc \to 2^{AP}$ *is a* labelling function *mapping each location to a set of atomic propositions taken from a set* AP*.*

The semantics of a PTA P is given by an infinite-state MDP whose states are of the form $(l, v) \in Loc \times (\mathbb{R}_{\geqslant 0})^{\mathcal{X}}$, where l is a location and v gives a value for each clock in $\mathcal{X}$. For simplicity, we omit a full descriptions of the semantics (see e.g. [58] for details). Intuitively, the behaviour of a PTA is as follows. The initial state is $(l, \mathbf{0})$, where $\mathbf{0}$ denotes a value of 0 for all clocks. For a general state $s = (l, v)$, there is a nondeterministic choice between either: (i) time elapsing, i.e. all clocks increasing in value, subject to the invariant $inv(l)$ staying true; (ii) an action a being taken, such that $prob(l, a)$ is defined and the guard $enab(l, a)$ is satisfied. If the latter, $prob(l, a)$ is a distribution over pairs $(X, l') \in 2^{\mathcal{X}} \times Loc$, giving the probability of moving to location l' and resetting the clocks in X to zero.

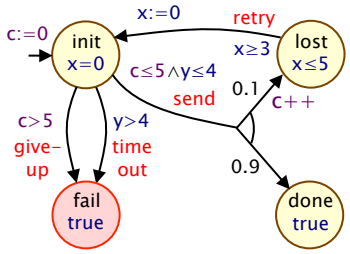


Figure 10. Example PTA.

Example 12 *Figure 10 shows an example of a PTA with locations* $Loc = \{\text{init}, \text{lost}, \text{done}, \text{fail}\}$ *and two clocks* $\{x, y\}$*. Locations are labelled with their invariants, under the location name, and transitions are annotated with actions (from the set* $Act = \{send, retry, giveup, timeout\}$*), guards (e.g.* $y > 4$*), clock resets (e.g.* $x := 0$*) and probabilities. For modelling convenience, we also use an integer variable* c*, whose value can feature in guards or be set on a transition.*

The PTA models repeated attempts to transmit a message over a faulty communication channel. Each send happens instantaneously (thanks to the invariant $x = 0$ *in location* init*), after which the transmission succeeds with probability 0.9 and fails with probability 0.1. In the case of failure, a delay of between 3 and 5 time units occurs before transmission is re-attempted. The variable* c *counts the number of attempts. If either* c *exceeds 5, or the total time elapsed (stored by clock* y*) exceeds 4, then transmission is aborted and the PTA moves to location* fail*.*

6.2. Model Checking for PTAs

A variety of model checking approaches have been proposed for PTAs. Properties to be verified are typically expressed in probabilistic temporal logic. One possibility is the logic PTCTL [59], a probabilistic extension of the timed temporal logic TCTL. In many cases, though, the simpler logic PCTL, discussed earlier in this tutorial for DTMCs and MDPs, suffices.

The principle hurdle to overcome when developing model checking algorithms for PTAs is the fact that the models are inherently infinite-state. Fundamental results about the decidability and complexity of model checking for PTAs can be obtained through the construction of a *region graph* [59]. This is based on a division of the PTA's state space into a finite number of *regions*, sets of states which satisfy exactly the same temporal logic formulas. This reduces the problem of model checking a PTA to the problem of analysing a finite-state MDP, whose states are regions. In practice, however the region graph is prohibitively large, and thus several more efficient methods have been developed for model checking:

- The *digital clocks* method [57] translates *closed* PTAs (those where clock constraints contain no strict comparisons) into a finite-state MDP, based on the *digitisation* of real-valued to integer-valued clocks. The MDP is then analysed in standard fashion. For PTAs whose clock values vary across large ranges this approach can become expensive, but it has been successfully applied to several large case studies.

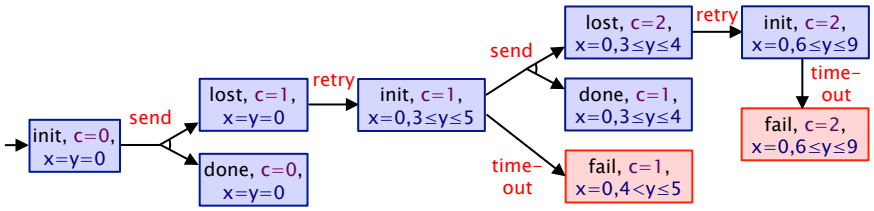


Figure 11. Reachability graph.

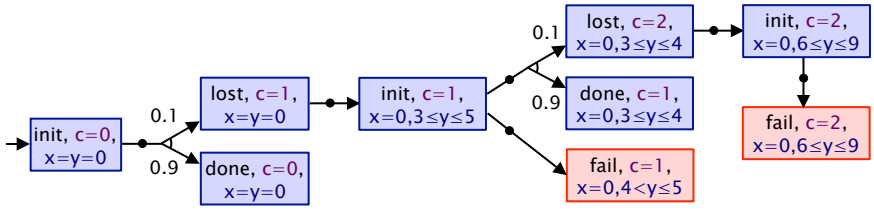


Figure 12. Stochastic game abstraction.

- *Backwards reachability* [60] performs a backwards traversal of the state space of a PTA using efficient data structures to represent and manipulate clock constraints for PTA state sets. As for digital clocks, a finite-state MDP is then built and model checked to obtain results. Although the operations required to implement this approach can be expensive, with appropriate optimisations [13] it has been shown to perform well.
- *Abstraction refinement with stochastic games* [54] uses the quantitative abstraction refinement approach [47] described in Section 5 to verify PTAs. In this instance, abstract states are modelled with clock constraints (zones), represented by difference-bound matrices (DBMs). A sequence of stochastic games are built through successive refinements, finally yielding an exact result for the PTA model checking problem.

Example 13 *We illustrate the third of these approaches, abstraction refinement, using the PTA given previously in Example 12 (see Figure 10). Our aim is to determine the maximum probability of reaching location* fail. *The abstraction refinement approach of [54] works by constructing a* reachability graph, *a connected set of abstract states of the form* (l, ζ) *where* l *is a location and* ζ *a clock constraint. This is constructed based on a forwards traversal of the reachable fragment of the PTA. The reachability graph for the example is shown in Figure 11.*

From the graph, we can construct a stochastic game abstraction, which is then analysed to obtain lower and upper bounds on the required reachability probability, in the manner described in Section 5.1. Figure 12 shows the stochastic game for the reachability graph of Figure 11. In this case, the obtained lower and upper bounds reveal that the maximum probability of reaching location fail *is in the interval* $[0.01, 0.1]$. *From the results of the analysis of the game, we are able to refine the reachability graph, splitting one or more of its abstract states into parts.*

For this example, a single refinement (which splits the clock constraint $x{=}0 \wedge 3{\leqslant}y{\leqslant}5$ *in abstract state* $((\text{init}, c{=}1), x{=}0 \wedge 3{\leqslant}y{\leqslant}5)$ *into two,* $x{=}0 \wedge 3{\leqslant}y{\leqslant}4$ *and* $x{=}0 \wedge 4{<}y{\leqslant}5$*) suffices to produce an abstraction which yields the bounds* $[0.1, 0.1]$*. Thus, the maximum probability of failure is 0.1.*

6.3. More on Model Checking for PTAs

As discussed previously for DTMCs and MDPs, we can augment PTAs with *rewards* to capture additional quantitative properties of the models. In fact, these are often used to model *costs* or *prices*. Computation of the (minimum or maximum) expected reward cumulated until a target is reached can be performed using the digital clocks method [57]. Another useful class of reward properties for PTAs is the probability that the cumulative reward exceeds a certain bound. A semi-algorithm for model checking such properties can be found in [12].

7. Probabilistic Model Checking in Practice

We conclude this tutorial with a discussion of some of the practical aspects of probabilistic model checking. As usage of these techniques has become more widespread, a variety of supporting software tools have been developed. The two most widely used are PRISM [55] and MRMC [43]. MRMC's core functionality is model checking of (discrete- and continuous-time) Markov chains, with a particular focus on reward properties. PRISM supports Markov chains, as well as MDPs and PTAs, and is described in more detail below. Other tools providing model checking of MDPs include LiQuor [18] and ProbDiVinE [11]. Software supporting verification of PTAs includes UPPAAL-PRO [66], Fortuna [13] and `mcpta` [37]. A more detailed list of probabilistic model checking tools can be found at [68].

7.1. PRISM

PRISM [55] is a probabilistic model checker developed at the Universities of Birmingham and Oxford. It is an open-source software tool which accepts probabilistic models written in a textual modelling language based on Reactive Modules, a state-based language using guarded commands. The three main model types discussed in this tutorial, DTMCs, MDPs and PTAs, and their reward (price) extensions are all directly supported by PRISM, as are continuous-time Markov chains (CTMCs). A wide range of properties can be specified and model checked: PRISM admits the logics PCTL, LTL and PCTL*, and additionally CSL (for CTMCs). It also includes the reward operators and quantitative (numerical) properties discussed earlier in this tutorial.

PRISM includes multiple model checking engines, notably several based on *symbolic* implementations (using binary decision diagrams and their extensions). These can enable the probabilistic verification of models of up to 10^{10} states. On average, the tool usually handles models with up to $10^7 - 10^8$ states. PRISM also features explicit-state model checking functionality, as well as a variety of advanced techniques such as abstraction refinement and symmetry reduction. There

is also support for approximate/statistical model checking through a discrete-event simulation engine.

PRISM has been applied to numerous case studies, from communication, security and population protocols, power management, embedded control, nanotechnology designs, to biological systems; for more information consult [67].

7.2. Research Directions

Finally, we briefly mention some other directions for the development of probabilistic model checking that have shown promising results.

- *Symbolic model checking* techniques, using data structures based on binary decision diagrams (BDDs) [5,49] offer scalability to large models by exploiting high-level structure and regularities.
- *Model reduction* techniques such as bisimulation minimisation [44], symmetry reduction [52,26] and partial order reduction [23,6] provide ways to reduce the size of probabilistic models, whilst preserving exact model checking results.
- *Compositional probabilistic verification* [61] decomposes the effort required for model checking into separate sub-tasks for each system component; in particular, *assume-guarantee* verification techniques for MDPs [56] have recently proved to provide real practical gains in scalability.
- *Approximate (or statistical) probabilistic model checking* improves scalability by combining Monte Carlo simulation with statistical methods to either generate approximate results to quantitative model checking queries [39] or efficiently check properties featuring probability thresholds [65].
- *Systems biology* is proving to be an exciting and challenging application of probabilistic model checking, for which various novel techniques have been specifically developed [64,38].
- *Probabilistic counterexamples* [32,1,3] provide diagnostic information to the user of a probabilistic model checker if a temporal logic query is found to be false. Typically, this takes the form of a *set* of violating system executions, whose combined probability exceeds some desired threshold.
- *Synthesis* techniques aim to generate correct-by-construction model components or controllers based on quantitative objectives [17,15].
- *Parametric* approaches to probabilistic model checking can be used to synthesise the range of model parameter values that satisfy some specification [33] or to produce model checking results that are symbolic expressions in terms of the parameters [30].

8. Conclusions

This tutorial presented an introduction to probabilistic model checking, covering details of two basic models, Markov chains and Markov decision processes, and two

more advanced topics, quantitative abstraction refinement and model checking of probabilistic timed automata. The focus has been on models which feature discrete states, discrete probability distributions and possibly nondeterminism, in conjunction with (in the case of PTAs) dense real-time.

The following tutorial papers provide more comprehensive introductory material for the topics covered here:

- [53] (sections 1–3), for discrete-time Markov chains (DTMCs);
- [28] (sections 1–7), for Markov decision processes (MDPs);
- [58], for probabilistic timed automata (PTAs).

Chapter 10 of [9] is also highly recommended for additional and deeper coverage of model checking for DTMCs and MDPs.

Acknowledgments. The authors are part supported by ERC Advanced Grant VERIWARE, EPSRC grant EP/F001096/1 and EU-FP7 project CONNECT.

References

[1] H. Aljazzar and S. Leue. Directed explicit state-space search in the generation of counterexamples for stochastic model checking. *IEEE Transactions on Software Engineering*, 36(1):37–60, 2010.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] M. Andrés, P. D'Argenio, and P. van Rossum. Significant diagnostic counterexamples in probabilistic model checking. In H. Chockler and A. Hu, editors, *Proc. 4th Int. Haifa Verification Conf. Hardware and Software: Verification and Testing (HVC'08)*, volume 5394 of *LNCS*, pages 129–148. Springer, 2008.

[4] D. Angluin, J. Aspnes, and D. Eisenstat. A simple population protocol for fast robust approximate majority. *Distributed Computing*, 21(2):87–102, 2008.

[5] C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In P. Degano, R. Gorrieri, and A. Marchetti-Spaccamela, editors, *Proc. 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

[6] C. Baier, M. Groesser, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*, pages 230–239. IEEE CS Press, 2004.

[7] C. Baier, M. Größer, and F. Ciesinski. Quantitative analysis under fairness constraints. In *Proc. 7th International Symposium on Automated Technology for Verification and Analysis (ATVA'09)*, volume 5799 of *LNCS*, pages 135–150. Springer, 2009.

[8] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Transactions on Software Engineering*, 29(6):524–541, 2003.

[9] C. Baier and J.-P. Katoen. *Principles of Model Checking*. MIT Press, 2008.

[10] C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.

[11] J. Barnat, L. Brim, I. Cerna, M. Ceska, and J. Tumova. ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In *Proc. 5rd International Conference on Quantitative Evaluation of Systems (QEST'08)*, pages 77–78. IEEE CS Press, 2008.

[12] J. Berendsen, D. Jansen, and J.-P. Katoen. Probably on time and within budget – on reachability in priced probabilistic timed automata. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 311–322. IEEE CS Press, 2006.

[13] J. Berendsen, D. Jansen, and F. Vaandrager. Fortuna: Model checking priced probabilistic timed automata. In *Proc. 7th International Conference on Quantitative Evaluation of SysTems (QEST'10)*, pages 273–281, 2010.

[14] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In P. Thiagarajan, editor, *Proc. 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *LNCS*, pages 499–513. Springer, 1995.

[15] P. Cerný, K. Chatterjee, T. Henzinger, A. Radhakrishna, and R. Singh. Quantitative synthesis for concurrent programs. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 243–259. Springer, 2011.

[16] R. Chadha and M. Viswanathan. A counterexample guided abstraction-refinement framework for Markov decision processes. *ACM Transactions on Computational Logic*, 12(1):1–49, 2010.

[17] K. Chatterjee, T. Henzinger, B. Jobstmann, and R. Singh. Measuring and synthesizing systems in probabilistic environments. In *Proc. 22nd International Conference on Computer Aided Verification (CAV'10)*, LNCS. Springer, 2010.

[18] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 131–132. IEEE CS Press, 2006.

[19] E. Clarke and A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proc. Workshop on Logic of Programs*, volume 131 of *LNCS*. Springer, 1981.

[20] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-guided abstraction refinement. In A. Emerson and A. Sistla, editors, *Proc. 12th Int. Conf. Computer Aided Verification (CAV'00)*, volume 1855 of *Lecture Notes in Computer Science*, pages 154–169. Springer, 2000.

[21] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite state probabilistic programs. In *Proc. 29th Annual Symposium on Foundations of Computer Science (FOCS'88)*, pages 338–345. IEEE Computer Society Press, 1988.

[22] P. D'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In L. de Alfaro and S. Gilmore, editors, *Proc. 1st Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01)*, volume 2165 of *LNCS*, pages 39–56. Springer, 2001.

[23] P. D'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In *Proc. 1st International Conference on Quantitative Evaluation of Systems (QEST'04)*. IEEE CS Press, 2004.

[24] L. de Alfaro, M. Kwiatkowska, G. Norman, D. Parker, and R. Segala. Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 395–410. Springer, 2000.

[25] L. de Alfaro and P. Roy. Magnifying-lens abstraction for Markov decision processes. In *Proc. 19th International Conference on Computer Aided Verification (CAV'07)*, volume 4590 of *LNCS*, pages 325–338. Springer, 2007.

[26] A. Donaldson and A. Miller. Symmetry reduction for probabilistic model checking using generic representatives. In S. Graf and W. Zhang, editors, *Proc. 4th Int. Symp. Automated Technology for Verification and Analysis (ATVA'06)*, volume 4218 of *Lecture Notes in Computer Science*, pages 9–23. Springer, 2006.

[27] J. Esparza and A. Gaiser. Probabilistic abstractions with arbitrary domains. In *Proc. 18th International Symposium on Static Analysis (SAS'11)*, pages 334–350, 2011.

[28] V. Forejt, M. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In M. Bernardo and V. Issarny, editors, *Formal Methods for Eternal Networked Software Systems (SFM'11)*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[29] E. M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. PASS: Abstraction refinement for infinite probabilistic models. In J. Esparza and R. Majumdar, editors, *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of*

Systems (TACAS'10), volume 6105 of *LNCS*, pages 353–357. Springer, 2010.

[30] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. In C. Pasareanu, editor, *Proc. 16th International SPIN Workshop*, volume 5578 of *LNCS*, pages 88–106. Springer, 2009.

[31] E. M. Hahn, G. Norman, D. Parker, B. Wachter, and L. Zhang. Game-based abstraction and controller synthesis for probabilistic hybrid systems. In *Proc. 8th International Conference on Quantitative Evaluation of SysTems (QEST'11)*, pages 69–78. IEEE CS Press, September 2011.

[32] T. Han, J.-P. Katoen, and B. Damman. Counterexample generation in probabilistic model checking. *IEEE Transactions on Software Engineering*, 35(2):241–257, 2009.

[33] T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Proc. IEEE Real-Time Systems Symposium (RTSS 08)*, pages 173–182. IEEE CS Press, 2008.

[34] H. Hansson. *Time and Probability in Formal Design of Distributed Systems*. Elsevier, 1994.

[35] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.

[36] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent programs. *ACM Transactions on Programming Languages and Systems*, 5(3):356–380, 1983.

[37] A. Hartmanns and H. Hermanns. A modest approach to checking probabilistic timed automata. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*, 2009. To appear.

[38] T. Henzinger, M. Mateescu, L. Mikeev, and V. Wolf. Hybrid numerical solution of the chemical master equation. In *Proc. 8th International Conference on Computational Methods in Systems Biology (CMSB'10)*, pages 55–65. ACM, 2010.

[39] T. Hérault, R. Lassaigne, F. Magniette, and S. Peyronnet. Approximate probabilistic model checking. In *Proc. 5th International Conference on Verification, Model Checking and Abstract Interpretation (VMCAI'04)*, volume 2937 of *LNCS*. Springer, 2004.

[40] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In S. Graf and M. Schwartzbach, editors, *Proc. 6th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'00)*, volume 1785 of *LNCS*, pages 347–362. Springer, 2000.

[41] H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In A. Gupta and S. Malik, editors, *Proc. 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *LNCS*, pages 162–175. Springer, 2008.

[42] H. Jensen. Model checking probabilistic real time systems. In *Proc. 7th Nordic Workshop on Programming Theory*, pages 247–261, 1996.

[43] J.-P. Katoen, E. M. Hahn, H. Hermanns, D. Jansen, and I. Zapreev. The ins and outs of the probabilistic model checker MRMC. In *Proc. 6th International Conference on Quantitative Evaluation of Systems (QEST'09)*, pages 167–176. IEEE CS Press, 2009.

[44] J.-P. Katoen, T. Kemna, I. Zapreev, and D. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In O. Grumberg and M. Huth, editors, *Proc. 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *LNCS*, pages 87–101. Springer, 2007.

[45] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Game-based probabilistic predicate abstraction in PRISM. In *Proc. 6th Workshop on Quantitative Aspects of Programming Languages (QAPL'08)*, 2008.

[46] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. Abstraction refinement for probabilistic software. In N. Jones and M. Muller-Olm, editors, *Proc. 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09)*, volume 5403 of *LNCS*, pages 182–197. Springer, 2009.

[47] M. Kattenbelt, M. Kwiatkowska, G. Norman, and D. Parker. A game-based abstraction-refinement framework for Markov decision processes. *Formal Methods in System Design*, 36(3):246–280, 2010.

[48] J. Kemeny, J. Snell, and A. Knapp. *Denumerable Markov Chains*. Springer-Verlag, 2nd edition, 1976.

[49] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.

[50] M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic model checking in practice: Case studies with PRISM. *ACM SIGMETRICS Performance Evaluation Review*, 32(4):16–21, 2005.

[51] M. Kwiatkowska, G. Norman, and D. Parker. Game-based abstraction for Markov decision processes. In *Proc. 3rd International Conference on Quantitative Evaluation of Systems (QEST'06)*, pages 157–166. IEEE CS Press, 2006.

[52] M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In T. Ball and R. Jones, editors, *Proc. 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4114 of *LNCS*, pages 234–248. Springer, 2006.

[53] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In M. Bernardo and J. Hillston, editors, *Formal Methods for the Design of Computer, Communication and Software Systems: Performance Evaluation (SFM'07)*, volume 4486 of *LNCS (Tutorial Volume)*, pages 220–270. Springer, 2007.

[54] M. Kwiatkowska, G. Norman, and D. Parker. Stochastic games for verification of probabilistic timed automata. In J. Ouaknine and F. Vaandrager, editors, *Proc. 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09)*, volume 5813 of *LNCS*, pages 212–227. Springer, 2009.

[55] M. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In G. Gopalakrishnan and S. Qadeer, editors, *Proc. 23rd International Conference on Computer Aided Verification (CAV'11)*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[56] M. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Assume-guarantee verification for probabilistic systems. In J. Esparza and R. Majumdar, editors, *Proc. 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10)*, volume 6105 of *LNCS*, pages 23–37. Springer, 2010.

[57] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Performance analysis of probabilistic timed automata using digital clocks. *Formal Methods in System Design*, 29:33–78, 2006.

[58] M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, chapter Verification of Real-Time Probabilistic Systems, pages 249–288. John Wiley & Sons, 2008.

[59] M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.

[60] M. Kwiatkowska, G. Norman, J. Sproston, and F. Wang. Symbolic model checking for probabilistic timed automata. *Information and Computation*, 205(7):1027–1077, 2007.

[61] R. Segala. *Modelling and Verification of Randomized Distributed Real Time Systems*. PhD thesis, Massachusetts Institute of Technology, 1995.

[62] R. Segala and N. Lynch. Probabilistic simulations for probabilistic processes. In B. Jonsson and J. Parrow, editors, *Proc. 5th International Conference on Concurrency Theory (CONCUR'94)*, volume 836 of *LNCS*, pages 481–496. Springer, 1994.

[63] M. Vardi. Automatic verification of probabilistic concurrent finite state programs. In *Proc. 26th Annual Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, 1985.

[64] V. Wolf, R. Goel, M. Mateescu, and T. Henzinger. Solving the chemical master equation using sliding windows. *BMC Systems Biology Journal*, 4(42), 2010.

[65] H. Younes and R. Simmons. Probabilistic verification of discrete event systems using acceptance sampling. In E. Brinksma and K. Larsen, editors, *Proc. 14th International Conference on Computer Aided Verification (CAV'02)*, volume 2404 of *LNCS*, pages 223–235. Springer, 2002.

[66] `http://www.cs.aau.dk/~arild/uppaal-probabilistic/`.

[67] `http://www.prismmodelchecker.org/`.

[68] `http://www.prismmodelchecker.org/other-tools.php`.