Probabilistic Model Checking:
Advances and Applications

Marta Kwiatkowska, Gethin Norman and David Parker

Abstract Probabilistic model checking is a powerful technique for formally verifying quantitative properties of systems that exhibit stochastic behaviour. Such systems are found in many application domains: for example, probabilistic behaviour may arise due to the presence of failures in unreliable hardware, message loss in wireless communication channels, or the use of randomisation in distributed protocols. This chapter starts with an introduction to the technique of probabilistic model checking. We then survey some recent advances in the area, including controller synthesis, compositional verification, probabilistic real-time systems and parametric model checking. We illustrate the application of the various techniques with a combination of toy examples and descriptions of larger case studies. The chapter concludes with a discussion of some of the key challenges in the field.

Marta Kwiatkowska
Department of Computer Science, University of Oxford, Oxford, UK,
e-mail: marta.kwiatkowska@cs.ox.ac.uk

Gethin Norman
School of Computing Science, University of Glasgow, UK,
e-mail: gethin.norman@glasgow.ac.uk

David Parker
School of Computer Science, University of Birmingham, UK,
e-mail: d.a.parker@cs.bham.ac.uk

1 Introduction

Computer systems play an important role in almost all aspects of everyday life, including many examples where safety and reliability is critical, from control systems for autonomous vehicles to embedded software in medical devices such as cardiac pacemakers. There is therefore a demand for rigorous, formal techniques which can verify that these systems function correctly and safely. Often, this requires an analysis of quantitative aspects such as reliability, responsiveness and resource usage. Furthermore, since such devices often operate in unpredictable and unknown environments, it is essential to consider the inherently probabilistic nature of real systems, such as the random timing of events, failures of embedded components and the loss of packets when using wireless communication networks.

Probabilistic model checking is an automated technique for formally verifying quantitative properties of stochastic systems. This involves the construction of a mathematical model that represents the behaviour of a system over time, i.e., the possible states that it can be in, the transitions that can occur between states, and information about the likelihood or timing of these transitions. Properties specifying the required behaviour of these systems are then formally specified in temporal logic and a systematic exploration and analysis of the system model is then performed to ascertain whether the properties are satisfied.

This approach allows a wide variety of quantitative properties to be specified, regarding, for example, 'the probability of a system failure occurring', 'the probability of a packet being successfully delivered within 5ms' or 'the expected power consumption of a sensor network during 1 hour of operation'. The basic theory and algorithms for probabilistic model checking were first put forward in the 1980s but, since then, substantial progress has been made in the development of theory, algorithms and tools for many different types of probabilistic models and a wide range of property specifications. This has resulted in the successful usage of probabilistic model checking on a huge range of computerised systems, from airbag controllers to cardiac pacemakers, and in a diverse range of applications domains, from computer security to robotics to quantum computing.

This chapter aims to provide both an introduction to the basics of probabilistic model checking and a survey of some of the key advances that have been made in recent years. In both cases, we illustrate the ideas using a variety of toy examples and real-life case studies, and provide pointers to further work and resources. We also make available electronic copies of the files needed to study these examples and case studies using the PRISM model checker [115].

In the first section of the chapter, we give an introduction to probabilistic model checking applied to several different types of models: discrete-time Markov chains, Markov decision processes and stochastic multi-player games. We then move on to cover a section of more advanced topics. This includes: (i) controller synthesis, which can be used to generate correct-by-construction controllers, e.g., for robots or vehicles, along with quantitative guarantees on their behaviour; (ii) modelling and verification techniques designed for large complex systems, including compositional (divide and conquer) approaches and the use of abstraction; (iii) verification

techniques for real-time probabilistic models, i.e., those that capture more realistic information about the timing and duration of system events; and (iv) parametric model checking methods, which provide more powerful ways to analyse models whose parameters (e.g., probabilities) may vary or be difficult to quantify accurately. We conclude the chapter with a discussion of the limitations of probabilistic model checking and some of the key current challenges and research directions.

2 Probabilistic Model Checking

In this section, we give an overview of the basics of probabilistic model checking. We focus on *discrete-time* models: discrete-time Markov chains (DTMCs), Markov decision processes (MDPs) and stochastic multi-player games (SMGs). These all model the behaviour of a probabilistic system as a sequence of discrete time-steps. We introduce the key definitions and concepts, and illustrate them with some examples. For more in-depth tutorial material on probabilistic model checking, see for example [77] (for DTMCs), [44] (for MDPs) and [104] (for SMGs).

Preliminaries. Before we start, we first introduce some definitions and notation used in the following sections. A (discrete) *probability distribution* over a countable set S is a function $\mu : S \to [0,1]$ such that $\sum_{s \in S} \mu(s) = 1$. For an arbitrary set S, we let $Dist(S)$ be the set of functions $\mu : S \to [0,1]$ such that $\{s \in S \mid \mu(s) > 0\}$ is a countable set and μ restricted to $\{s \in S \mid \mu(s) > 0\}$ is a probability distribution. The *point distribution* at $s \in S$, denoted η_s, is the distribution that assigns probability 1 to s (and 0 to everything else). Given two sets S_1 and S_2 and distributions $\mu_1 \in Dist(S_1)$ and $\mu_2 \in Dist(S_2)$, the *product distribution* $\mu_1 \times \mu_2 \in Dist(S_1 \times S_2)$ is given by $\mu_1 \times \mu_2((s_1, s_2)) = \mu_1(s_1) \cdot \mu_2(s_2)$. We will also often use the more general notion of a *probability measure*. We omit a complete definition here and instead refer the reader to, for example, [23] for introductory material on this topic.

2.1 Discrete-time Markov Chains

We now give an overview of probabilistic model checking for discrete-time Markov chains, the simplest class of models that we consider in this chapter.

Definition 1 (Discrete-time Markov chain). A *discrete-time Markov chain* (DTMC) is a tuple $D = (S, \bar{s}, \mathbf{P}, L)$ where:

- S is a set of states;
- $\bar{s} \in S$ is an initial state;
- $\mathbf{P} : S \times S \to [0,1]$ is a probabilistic transition matrix such that $\sum_{s' \in S} \mathbf{P}(s, s') = 1$ for all $s \in S$;
- $L : S \to 2^{AP}$ is a labelling function assigning to each state a set of atomic propositions from a set AP.

The state space S of a DTMC $D=(S,\bar{s},\mathbf{P},L)$ represents the set of all possible configurations of the system being modelled. The system's initial configuration is given by $\bar{s}$ and its subsequent evolution is represented by the probabilistic transition matrix $\mathbf{P}$: for states $s,s' \in S$, the entry $\mathbf{P}(s,s')$ is the probability of making a transition from state s to s'. By definition, for any state of D, the probabilities of all outgoing transitions from that state sum to 1.

A possible execution of D is represented by a *path*, which is a (finite or infinite) sequence of states $\pi = s_0 s_1 s_2 \ldots$ such that $\mathbf{P}(s_i, s_{i+1}) > 0$ for all $i \geqslant 0$. For a path π, we let $\pi(i)$ denote the $(i+1)$th state s_i of the path, and $\pi[i\ldots]$ be the suffix of π starting in state s_i. We also let $|\pi|$ be its length and, if π is finite, $last(\pi)$ be its last state. We let $IPaths_D(s)$ and $FPaths_D(s)$ denote the sets of finite and infinite paths of D starting in state s, respectively, and we write $IPaths_D$ and $FPaths_D$ for the sets of *all* finite and infinite paths, respectively.

To reason quantitatively about the behaviour of DTMC D we must determine the probability that certain paths are executed. To do so, we define, for each state s of D, a probability measure $Pr_{D,s}$ over the set of infinite paths of D starting in s. We present just the basic idea here; for the complete construction, see [75].

For any finite path $\pi = s s_1 s_2 \ldots s_n \in FPaths_D(s)$, the probability of the path occurring is given by $\mathbf{P}(\pi) = \mathbf{P}(s,s_1) \cdot \mathbf{P}(s_1,s_2) \cdots \mathbf{P}(s_{n-1},s_n)$. The *cylinder set* of π, denoted $C(\pi)$, is the set of all infinite paths which have π as a prefix, and the probability assigned to this set of paths is $Pr_{D,s}(C(\pi)) = \mathbf{P}(\pi)$. This can be extended uniquely to define the probability measure $Pr_{D,s}$ over $IPaths_D(s)$.

Using this probability measure, we can quantify the probability that, starting from a state s, the behaviour of D satisfies a particular specification (assuming that the behaviour of interest is represented by a measurable set of paths). For example, we can consider the probability of reaching a particular class of states, or of visiting some set of states infinitely often. Furthermore, given a random variable f over the infinite paths $IPaths_D$ (i.e., a real-valued function $f : IPaths_D \to \mathbb{R}_{\geqslant 0}$), we can define, using the probability measure $Pr_{D,s}$, the *expected value* of the variable f when starting in s, denoted $\mathbb{E}_{D,s}(f)$. More formally, we have:

$$\mathbb{E}_{D,s}(f) \overset{\text{def}}{=} \int_{\pi \in IPaths_D(s)} f(\pi) \mathrm{d}Pr_{D,s}.$$

We use random variables to formalise a variety of other quantitative properties of DTMCs. We do so by annotating the model with *rewards* (sometimes, these in fact represent *costs*, but we will consistently refer to these as rewards). Rewards can be used to model, for example, the energy consumption of a device, or the number of packets lost by a communication protocol. Formally, these are defined as follows.

Definition 2 (DTMC reward structure). A *reward structure* for a DTMC $D = (S,\bar{s},\mathbf{P},L)$ is a tuple $r=(r_S,r_T)$ where $r_S : S \to \mathbb{R}_{\geqslant 0}$ is a *state reward function* and $r_T : S \times S \to \mathbb{R}_{\geqslant 0}$ is a *transition reward function*.

State rewards are also called cumulative rewards and transition rewards are sometimes known as instantaneous or impulse rewards. We use random variables to measure, for example, the expected total amount of reward cumulated (over some num-
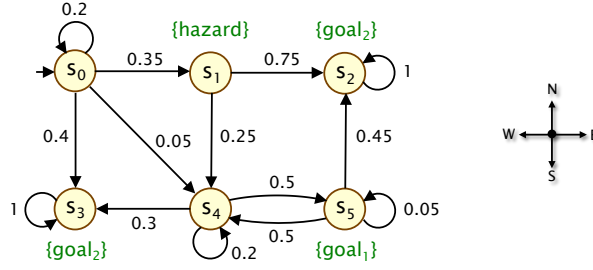
Fig. 1 Running example: a DTMC D representing a robot moving about a 3×2 grid.

ber of steps, until a set of states is reached, or indefinitely) or the expected value of a reward structure at a particular instant.

Example 1. We now introduce a running example, which we will develop throughout the chapter. It concerns a robot moving through terrain that is divided up into a 3×2 grid, with each grid section represented as a state. Fig. 1 shows a DTMC model of the robot. In each of the 6 states, the robot selects, at random, a direction to move. Due to the presence of obstacles, certain directions are unavailable in some states. For example, in state s_0, the robot will either remain in its current location (with probability 0.2), move east (with probability 0.35), move south (with probability 0.4) or move south-east (with probability 0.05). We also show labels for the states, taken from the set of atomic propositions $AP = \{\mathsf{hazard}, \mathsf{goal}_1, \mathsf{goal}_2\}$.   ■

2.1.1 Property Specifications

In order to formally specify properties of interest of a DTMC, we use quantitative extensions of *temporal logic*. For the purposes of this presentation, we introduce a rather general logic that essentially coincides with the property specification language of the PRISM model checker [79]. We refer to it here as *the PRISM logic*. This extends the probabilistic temporal logic PCTL* with operators to specify expected reward properties. PCTL*, in turn, subsumes the logic PCTL (probabilistic computation tree logic) [60] and probabilistic LTL (linear time logic) [95].

Definition 3 (PRISM logic syntax). The syntax of our logic is given by:

$$\phi ::= \mathtt{true} \mid \mathtt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \mathtt{P}_{\bowtie p}[\,\psi\,] \mid \mathtt{R}^r_{\bowtie q}[\,\rho\,]$$
$$\psi ::= \phi \mid \neg\psi \mid \psi \wedge \psi \mid \mathtt{X}\,\psi \mid \psi\,\mathtt{U}^{\leqslant k}\,\psi \mid \psi\,\mathtt{U}\,\psi$$
$$\rho ::= \mathtt{I}^{=k} \mid \mathtt{C}^{\leqslant k} \mid \mathtt{C} \mid \mathtt{F}\,\phi$$

where $\mathtt{a} \in AP$ is an atomic proposition, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, $p \in [0,1]$, r is a reward structure, $q \in \mathbb{R}_{\geqslant 0}$ and $k \in \mathbb{N}$.

The syntax in Defn. 3 distinguishes between state formulae (ϕ), path formulae (ψ) and reward formulae (ρ). State formulae are evaluated over the states of a DTMC,

while path and reward formulae are both evaluated over paths. A property of a DTMC is specified as a state formula; path and reward formulae appear only as subformulae, within the P and R operators, respectively.

For a state s of a DTMC D, we say that s *satisfies* ψ (or ψ *holds* in s), written $D, s \models \psi$, if ψ evaluates to true in s. If the model D is clear from the context, we simply write $s \models \psi$. In addition to the standard operators of propositional logic, state formulae ϕ can include the probabilistic operator P and reward operator R, which have the following meanings:

- s satisfies $P_{\bowtie p}[\psi]$ if the probability of taking a path from s satisfying ψ is in the interval specified by $\bowtie p$;
- s satisfies $R^r_{\bowtie q}[\rho]$ if the expected value of reward operator ρ from state s, using reward structure r, is in the interval specified by $\bowtie q$.

The core temporal operators used to construct path formulae ψ are:

- $X \psi$ ('next') – ψ holds in the next state;
- $\psi_1 U^{\leqslant k} \psi_2$ ('bounded until') – ψ_2 becomes true within k steps, and ψ_1 holds up until that point;
- $\psi_1 U \psi_2$ ('until') – ψ_2 eventually becomes true, and ψ_1 holds until then.

We often use the equivalences $F \psi \equiv \text{true } U \psi$ ('eventually' ψ) and $G \psi \equiv \neg F \neg \psi$ ('always' ψ), as well as the bounded variants $F^{\leqslant k} \psi$ and $G^{\leqslant k} \psi$. When restricting ψ to be an atomic proposition, we get the following common classes of property:

- $F a$ ('reachability') – eventually a holds;
- $G a$ ('invariance') – a remains true forever;
- $F^{\leqslant k} a$ ('step-bounded reachability') – a becomes true within k steps;
- $G^{\leqslant k} a$ ('step-bounded invariance') – a remains true for k steps.

More generally, path formulae allow temporal operators to be be combined. In fact the syntax of path formulae ψ given in Defn. 3 is that of linear temporal logic (LTL) [95].[1] This logic can express a large class of useful properties, core examples of which include:

- $G F \psi$ ('recurrence') – ψ holds infinitely often;
- $F G \psi$ ('persistence') – eventually ψ always holds;
- $G(\psi_1 \rightarrow X \psi_2)$ - whenever ψ_1 holds, ψ_2 holds in the next state;
- $G(\psi_1 \rightarrow F \psi_2)$ - whenever ψ_1 holds, ψ_2 holds at some point in the future.

For reward formulae ρ, we allow four operators:

- $I^{=k}$ ('instantaneous reward') – the state reward at time step k;
- $C^{\leqslant k}$ ('bounded cumulative reward') – the reward accumulated over k steps;
- C ('total reward') – the total reward accumulated (indefinitely);
- $F \phi$ ('reachability reward') – the reward accumulated up until the first time a state satisfying ϕ is reached.

---

[1] The bounded until operator $\psi_1 U^{\leqslant k} \psi_2$ is not usually included in the syntax of LTL, but it can be derived from other operators so its inclusion is not problematic.

Numerical queries. It is often of more interest to know the actual probability with which a path formula is satisfied or the expected value of a reward formula, than whether or not the probability or expected value meets a particular bound. To allow such queries, we extend the logic of Defn. 3 to include *numerical* queries of the form $P_{=?}[\psi]$ or $R^r_{=?}[\rho]$, which yield the probability that ψ holds and the expected value of reward operator ρ using reward structure r, respectively.

Example 2. We now return to our running example of a robot navigating a grid (see Example 1 and Fig. 1) and illustrate some properties specified in the PRISM logic.

- $P_{\geqslant 1}[F\ goal_2]$ – the probability the robot reaches a $goal_2$ state is 1.
- $P_{\geqslant 0.9}[G\ \neg hazard]$ – the probability it never visits a hazard state is at least 0.9.
- $P_{=?}[\neg hazard\ U^{\leqslant k}\ (goal_1 \vee goal_2)]$ – what is the probability that the robot reaches a state labelled with either $goal_1$ or $goal_2$, while avoiding hazard-labelled states, during the first k steps of operation?
- $R^{r^1}_{\leqslant 4.5}[C^{\leqslant k}]$ where $r^1=(r^1_S, r^1_T)$, $r^1_S(s)=1$ if s is labelled hazard and 0 otherwise and $r_T(s,s')=0$ for all $s,s' \in S$ – the expected number of times the robot visits a hazard labelled state during the first k steps is at most 4.5.
- $R^{r^2}_{=?}[F\ (goal_1 \vee goal_2)]$ where $r^2=(r^2_S, r^2_T)$, $r^2_S(s)=0$ for all $s \in S$ and $r_T(s,s')=1$ for all $s,s' \in S$ – what is the expected number of steps required for the robot to reach a state labelled $goal_1$ or $goal_2$? ∎

The formal semantics of the PRISM logic, for DTMCs, is defined as follows.

Definition 4 (PRISM logic semantics for DTMCs). For a DTMC $D=(S,\bar{s},\mathbf{P},L)$, reward structure $r=(r_S,r_T)$ for D and state $s \in S$, the satisfaction relation $\models$ is defined as follows:

$$
\begin{aligned}
D,s &\models \texttt{true} &&\text{always} \\
D,s &\models \texttt{a} &\Leftrightarrow\ &\texttt{a} \in L(s) \\
D,s &\models \neg\phi &\Leftrightarrow\ &D,s \not\models \phi \\
D,s &\models \phi_1 \wedge \phi_2 &\Leftrightarrow\ &D,s \models \phi_1 \wedge D,s \models \phi_2 \\
D,s &\models P_{\bowtie p}[\psi] &\Leftrightarrow\ &Pr_{D,s}\{\pi \in IPaths_D(s) \mid D,\pi \models \psi\} \bowtie p \\
D,s &\models R^r_{\bowtie q}[\rho] &\Leftrightarrow\ &\mathbb{E}_{D,s}(rew^r(\rho)) \bowtie q
\end{aligned}
$$

where for any path $\pi = s_0 s_1 s_2 \ldots \in IPaths_D$:

$$
\begin{aligned}
\mathsf{D}, \pi \models \phi &\Leftrightarrow \mathsf{D}, s_0 \models \phi \\
\mathsf{D}, \pi \models \neg \psi &\Leftrightarrow \mathsf{D}, \pi \not\models \psi \\
\mathsf{D}, \pi \models \psi_1 \wedge \psi_2 &\Leftrightarrow \mathsf{D}, \pi \models \psi_1 \wedge \mathsf{D}, \pi \models \psi_2 \\
\mathsf{D}, \pi \models \mathsf{X}\ \psi &\Leftrightarrow \mathsf{D}, \psi[1\dots] \models \psi \\
\mathsf{D}, \pi \models \psi_1\ \mathsf{U}^{\leqslant k}\ \psi_2 &\Leftrightarrow \exists i \in \mathbb{N}.\,(\,i \leqslant k \wedge \mathsf{D}, \pi[i\dots] \models \psi_2 \wedge \forall j {<} i.\,(\mathsf{D}, \pi[j\dots] \models \psi_1)\,) \\
\mathsf{D}, \pi \models \psi_1\ \mathsf{U}\ \psi_2 &\Leftrightarrow \exists i \in \mathbb{N}.\,(\,\mathsf{D}, \pi[i\dots] \models \psi_2 \wedge \forall j {<} i.\,(\mathsf{D}, \pi[j\dots] \models \psi_1)\,)
\end{aligned}
$$

$$
\begin{aligned}
\mathit{rew}^r(\mathtt{I}^{=k})(\pi) &= r_S(s_k) \\
\mathit{rew}^r(\mathtt{C}^{\leqslant k})(\pi) &= \sum_{j=0}^{k-1}\big(r_S(s_j) + r_T(s_j, s_{j+1})\big) \\
\mathit{rew}^r(\mathtt{C})(\pi) &= \sum_{j=0}^{\infty}\big(r_S(s_j) + r_T(s_j, s_{j+1})\big) \\
\mathit{rew}^r(\mathtt{F}\ \phi)(\pi) &= \begin{cases} \infty & \text{if } \forall j \in \mathbb{N}.\,\mathsf{D}, s_j \not\models \phi \\ \sum_{j=0}^{m_\phi - 1}\big(r_S(s_j) + r_T(s_j, s_{j+1})\big) & \text{otherwise} \end{cases}
\end{aligned}
$$

and $m_\phi = \min\{j \mid \mathsf{D}, s_j \models \phi\}$.

2.1.2 Model Checking

Verifying formulae in this logic against a DTMC requires a combination of graph-based algorithms, automata-based methods using deterministic Rabin automata (DRAs) and solving systems of linear equations. The main components of the model checking procedure are computing the probability that a path formula is satisfied and the expected value of a reward formula. Computing the probability that a path formula is satisfied requires first translating the formula into a DRA, finding the *bottom strongly connected components* on the product of the DTMC (informally, these are the sets of states of a DTMC which once entered are never left) and the constructed automaton and finally solving a linear equation system [17]. Computing the expected value of a reward formula, for unbounded cumulative and reachability reward formulae, also involves graph based analysis (either finding the bottom strongly connected components for unbounded cumulative reward properties or finding the states that reach a target with probability 1 for reachability reward properties) and solving a system of linear equations [77]. For the remaining reward formulae, computation of expected values involves iteratively solving a set of linear equations.

   The overall complexity of model checking is doubly exponential in the formula and polynomial in the size of the DTMC, but can be reduced to a single exponential. For scalability reasons, when implementing model checking of DTMCs, iterative numerical methods such as Jacobi and Gauss-Seidel, as opposed to direct methods such as Gaussian elimination, are often employed when solving systems of linear equations.
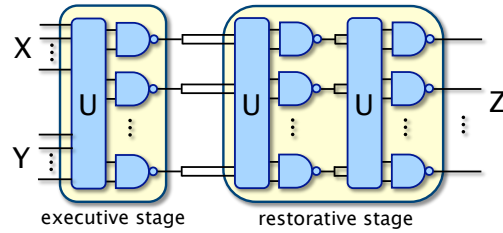
Fig. 2 An example of a NAND multiplexing unit with one restorative stage (M=1).

2.1.3 Case Study: NAND Multiplexing

We now describe a case study in which the system is modelled as a DTMC. This is taken from [91] and concerns the analysis of defect-tolerant systems used in computer-aided design. The system under study uses *multiplexing*, a technique introduced by von Neumann [89] which enables reliable computations when using unreliable devices. The approach was developed due to the unreliability of the valves (also known as vacuum tubes) that were used in computers, and these techniques are becoming relevant again for systems developed using nanotechnology where, due to their small-scale, components are again unreliable.

Multiplexing involves replacing a single processing unit by a multiplexing unit which has N copies of the inputs and outputs of the original processing unit. In the multiplexing unit, the N inputs are processed in parallel, giving N outputs. If the inputs and devices are reliable, then each of the N outputs would equal the output of the single processing unit. However, if there are errors in the inputs or the processing is unreliable, then there will also be errors in the outputs. To give a value to the output of the multiplexing unit, we define a critical level $\Delta \in [0, 0.5)$ and, if at least $(1-\Delta) \cdot N$ of the outputs take a certain value (i.e., either true or false), this is taken as the output value. If this criteria is not met by either true or false, the output value of the multiplexing unit is unknown and an error occurs.

The design of a multiplexing unit comprises an *executive stage*, which carries out the basic function of the unit to be replaced, and *M restorative stages*, which reduce the degradation of the output from the executive stage caused by errors in the inputs and unreliable processing. For the case of NAND multiplexing, the focus of this case study, a design with a single restorative stage is shown in Fig. 2.

Fig. 3 presents results obtained with the probabilistic model checker PRISM [79, 114] when analysing: (i) the probability of errors being less than than 10 percent; and (ii) the expected percentage of incorrect outputs of the system. The values are plotted as the number of restorative units (M) and the probability that a NAND gate is unreliable (err) vary. The first property can be expressed as the numerical query $\mathrm{P}_{=?}[\mathrm{F}\ below]$, where below is an atomic proposition labelling states of the DTMC where the computation has finished and the number of errors is below 10%. The second property can be expressed as the query $\mathrm{R}^r_{=?}[\mathrm{F}\ done]$, where done labels states of the DTMC where the computation has completed, and the reward structure
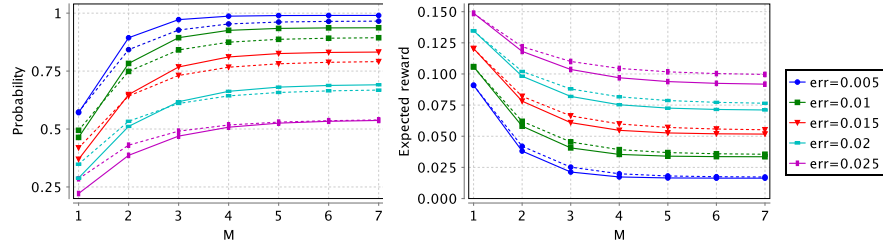
Fig. 3 Probabilistic model checking results for the NAND case study.

r labels the transitions entering this state with a reward equal to the percentage of incorrect outputs. When studying this model with PRISM [91], an error was found in the analytical analysis of [57]. The dashed lines in Fig. 3 show the results obtained in this case and demonstrate that this error can cause both under- and over-approximations of the reliability of a NAND multiplexing unit.

2.2 Markov Decision Processes

The second discrete-time model we consider is *Markov decision processes* (MDPs). These extend DTMCs by allowing nondeterministic as well as (discrete) probabilistic behaviour. Nondeterminism is a valuable tool for a modeller and can be used to represent a variety of unknown aspects of a system's environment or execution. For example, it can model the scheduling between a set of components running concurrently, the instructions and inputs provided to a robot to control its execution, or the unknown behaviour of an adversary trying to attack a security system. More generally, nondeterminism can also be used to abstract parts of a system that are unknown, under-specified or unimportant.

Definition 5 (Markov decision process). A *Markov decision process* (MDP) is a tuple $M = (S, \bar{s}, A, \delta, L)$ where:

- S is a finite set of states;
- $\bar{s} \in S$ is an initial state;
- A is a finite set of *actions*;
- $\delta : S \times A \to Dist(S)$ is a (partial) probabilistic transition function, mapping state-action pairs to probability distributions over S;
- $L : S \to 2^{AP}$ is a state labelling function.

In a state s of an MDP $M = (S, \bar{s}, A, \delta, L)$, there is first a nondeterministic choice between a set of actions that are *available* in the state. This set, denoted $A(s)$, includes the actions for which a probabilistic transition is defined: $A(s) = \{a \mid \delta(s, a) \text{ is defined}\}$. We assume that the set $A(s)$ is non-empty for all states $s \in S$. Once an available action $a \in A(s)$ has been chosen in s, the action is performed and
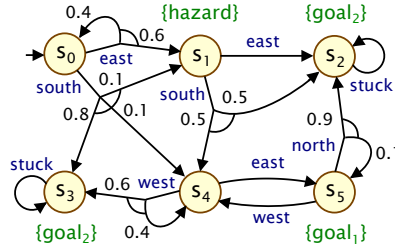
Fig. 4 Running example: an MDP M representing a robot moving about a 3×2 grid.

the successor state s' is chosen probabilistically, where the probability of moving to state s' is $\delta(s,a)(s')$.

Like for DTMCs, a path is a sequence of states corrected by transitions, but now also incorporates the action choice made. A (finite or infinite) path of M is of the form $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, where $a_i \in A(s_i)$ and $\delta(s_i,a_i)(s_{i+1})>0$ for all $i \geqslant 0$. The sets of all finite and infinite paths from state s of M are denoted $FPaths_M(s)$ and $IPaths_M(s)$, respectively, and the sets of all such paths are $FPaths_M$ and $IPaths_M$.

As for DTMCs we can define a reward structure over an MDP. State rewards remain unchanged, however for MDPs, instead of rewards beging associated with individual transitions, rewards are associated with performing actions in states.

Definition 6 (MDP reward structure). A *reward structure* for an MDP M $=$ $(S,\bar{s},A,\delta,L)$ is a tuple $r=(r_S,r_A)$ where $r_S : S \to \mathbb{R}_{\geqslant 0}$ is a *state reward function* and $r_A : S \times A \to \mathbb{R}_{\geqslant 0}$ is an *action reward function*.

Example 3. We now return to our running example of a robot moving through terrain that is divided up into a 3×2 grid (see Example 1 and Fig. 1). We extend our earlier DTMC model so that, instead of the robot choosing a direction to move at random, the choice is modelled using nondeterminism in an MDP. The model is shown in Fig. 4. The probabilistic transition function is drawn as grouped, labelled arrows and, when the probability is 1, it is omitted. In each state, one or more actions from the set $A=\{north,east,south,west,stuck\}$ are available, which move the robot between grid sections. As for the DTMC model, due to the presence of obstacles, certain directions are unavailable and in this case the obstacles can also cause the robot to probabilistically move to an alternative grid section. We use the action *stuck* to indicate that the robot cannot move in any direction in the states s_2 and s_3. ∎

To reason about the behaviour of an MDP, we need the notion of *strategies* (also called policies, adversaries and schedulers in other contexts). A strategy resolves the nondeterminism in the model, that is, the choices of which action to perform in each state. This choice can depend on the history of the MDP's execution and can be made either deterministically or randomly.

Definition 7 (MDP strategy). A *strategy* of an MDP M$=(S,\bar{s},A,\delta,L)$ is a function $\sigma : FPaths_M \to Dist(A)$ such that $\sigma(\pi)(a)>0$ only if $a \in A(last(\pi))$. The set of all strategies of M is denoted Σ_M.

We classify strategies in terms of both their use of *randomisation* and of *memory*.

- **Randomisation**: we say that strategy σ is *deterministic* (or *pure*) if $\sigma(\pi)$ is a point distribution for all finite paths π, and *randomised* otherwise.

- **Memory**: a strategy σ is *memoryless* if $\sigma(\pi)$ depends only on $last(\pi)$ for all finite paths π, and *finite-memory* if there are finitely many *modes* such that, for any π, $\sigma(\pi)$ depends only on $last(\pi)$ and the current mode, which is updated each time an action is performed; otherwise, it is *infinite-memory*.

Under a strategy $\sigma \in \Sigma_M$ of MDP M, all nondeterminism of M is resolved, and hence the behaviour is fully probabilistic. We can represent this using an (infinite) *induced discrete-time Markov chain*, whose states are the finite paths of M. For a given state s of M, we can then use this DTMC (see Sect. 2.1) to construct a probability measure $Pr_{M,s}^{\sigma}$ over the infinite paths $IPaths_M(s)$, capturing the behaviour of M when starting from state s under strategy σ. Furthermore, for a random variable $f : IPaths_M \to \mathbb{R}_{\geqslant 0}$, we can define the expected value $\mathbb{E}_{M,s}^{\sigma}(f)$ of f when starting from state s under strategy σ. Formally, the induced DTMC can be defined as follows.

Definition 8 (Induced DTMC). For an MDP $M=(S,\bar{s},A,\delta,L)$ and strategy $\sigma \in \Sigma_M$ for M, the *induced DTMC* is the DTMC $M_\sigma=(FPaths_M,\bar{s},\mathbf{P},L')$ where, for any $\pi,\pi' \in FPaths_M$:

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

and $L'(\pi)=L(last(\pi))$ for all $\pi \in FPaths_M$. Furthermore, a reward structure $r=(r_S,r_A)$ over M induces the reward structure $r^\sigma=(r_S^\sigma,r_T^\sigma)$ over M_σ where for any $\pi,\pi' \in FPaths_M$:

$$r_S^\sigma(\pi) = r_S(last(\pi))$$
$$r_T^\sigma(\pi,\pi') = \begin{cases} r_A(last(\pi),a) & \text{if } \pi' = \pi \xrightarrow{a} s \text{ for some } a \in A \text{ and } s \in S; \\ 0 & \text{otherwise.} \end{cases}$$

An induced DTMC has an infinite number of states. However, in the case of finite-memory strategies (and hence also the subclass of memoryless strategies), we can construct a finite-state *quotient DTMC* [44].

To specify properties of MDPs, we again use the PRISM logic defined for DTMCs in the previous section. The syntax (see Defn. 3) is identical, and the semantics (see Defn. 4) is very similar, the key difference being that, for the $P_{\bowtie p}[\psi]$ and $R_{\bowtie q}^r[\rho]$ operators, we quantify over all possible strategies of the MDP. The treatment of reward operators is also adapted slightly to consider action, as opposed to transition, reward functions.

Definition 9 (PRISM logic semantics for MDPs). For an MDP $M=(S,\bar{s},A,\delta,L)$ and reward structure $r=(r_S,r_A)$ for M, the satisfaction relation $\models$ is defined as for DTMCs in Defn. 4, except that, for any state $s \in S$:

$$\mathsf{M}, s \models \mathsf{P}_{\bowtie p}[\psi] \quad \Leftrightarrow \quad Pr^{\sigma}_{\mathsf{M},s}\{\pi \in IPaths_{\mathsf{M}}(s) \mid \mathsf{M}, \pi \models \psi\} \bowtie p \ \text{ for all } \sigma \in \Sigma_{\mathsf{M}}$$

$$\mathsf{M}, s \models \mathsf{R}^r_{\bowtie q}[\rho] \quad \Leftrightarrow \quad \mathbb{E}^{\sigma}_{\mathsf{M},s}(rew^r(\rho)) \bowtie q \ \text{ for all } \sigma \in \Sigma_{\mathsf{M}}$$

and, for any path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \in IPaths_{\mathsf{M}}$:

$$
\begin{aligned}
rew^r(\mathtt{I}^{=k})(\pi) &= r_S(s_k) \\
rew^r(\mathtt{C}^{\leqslant k})(\pi) &= \sum_{j=0}^{k-1} \big(r_S(s_j) + r_A(s_j, a_j)\big) \\
rew^r(\mathtt{C})(\pi) &= \sum_{j=0}^{\infty} \big(r_S(s_j) + r_A(s_j, a_j)\big) \\
rew^r(\mathtt{F}\ \phi)(\pi) &= \begin{cases} \infty & \text{if } \forall j \in \mathbb{N}.\, \mathsf{M}, s_j \not\models \phi \\ \sum_{j=0}^{m_\phi - 1} \big(r_S(s_j) + r_A(s_j, a_j)\big) & \text{otherwise} \end{cases}
\end{aligned}
$$

where $m_\phi = \min\{j \mid \mathsf{M}, s_j \models \phi\}$.

The main components of the model checking procedure for this logic against an MDP are computing the optimal probabilities that a path formula is satisfied and the optimal expected values for a reward formula. More precisely we are concerned with the following optimal values for an MDP M and state s:

$$Pr^{\min}_{\mathsf{M},s}(\psi) \quad \overset{\text{def}}{=} \quad \inf_{\sigma \in \Sigma_{\mathsf{M}}} Pr^{\sigma}_{\mathsf{M},s}\{\pi \in IPaths_{\mathsf{M}}(s) \mid \mathsf{M}, \pi \models \psi\} \tag{1}$$

$$Pr^{\max}_{\mathsf{M},s}(\psi) \quad \overset{\text{def}}{=} \quad \sup_{\sigma \in \Sigma_{\mathsf{M}}} Pr^{\sigma}_{\mathsf{M},s}\{\pi \in IPaths_{\mathsf{M}}(s) \mid \mathsf{M}, \pi \models \psi\} \tag{2}$$

$$\mathbb{E}^{\min}_{\mathsf{M},s}(r, \rho) \quad \overset{\text{def}}{=} \quad \inf_{\sigma \in \Sigma_{\mathsf{M}}} \mathbb{E}^{\sigma}_{\mathsf{M},s}(rew^r(\rho)) \tag{3}$$

$$\mathbb{E}^{\max}_{\mathsf{M},s}(r, \rho) \quad \overset{\text{def}}{=} \quad \sup_{\sigma \in \Sigma_{\mathsf{M}}} \mathbb{E}^{\sigma}_{\mathsf{M},s}(rew^r(\rho)) \tag{4}$$

where ψ is a path formula, r is a reward structure of M and ρ is a reward formula.

For example, verifying the property $\phi = \mathsf{P}_{<p}[\psi]$ in state s of M can be achieved by computing the optimal probability $Pr^{\max}_{\mathsf{M},s}(\psi)$ since the state s satisfies ϕ if and only if $Pr^{\max}_{\mathsf{M},s}(\psi) < p$. Similarly to DTMCs, rather than fixing a specific bound, we can query the (optimal) values directly. In the case of MDPs, the syntax of the PRISM logic is extended to include numerical queries of the form $\mathsf{P}_{\min=?}[\psi]$, $\mathsf{P}_{\max=?}[\psi]$, $\mathsf{R}^r_{\min=?}[\rho]$ and $\mathsf{R}^r_{\max=?}[\rho]$.

Model checking for an MDP reduces to building DRAs, performing graph analysis and numerical computation. As for DTMC model checking, DRAs are built to represent path formulae. The graph analysis involves identifying states of the MDP for which the probability is 0 or 1 and finding *maximal end components* of the MDP (or of the product of the MDP and a DRA). Informally, end components of an MDP are sets of states for which it possible (i.e., assuming certain nondeterministic choices are made) to remain in forever once entered.

The numerical computation can be achieved using various methods including: solving a linear programming problem; policy iteration (which builds a sequence of strategies until an optimal one is reached); and value iteration, which computes increasingly precise approximations to the optimal probability or expected value. The overall complexity for model checking is doubly exponential in the formula and polynomial in the size of the MDP.

Further details on the techniques needed to analyse MDPs can be found in, for example, [44, 17, 3] and in standard texts on MDPs [22, 64, 97].

2.3 Stochastic Multi-Player Games

The final model we consider in this introductory section is *stochastic multi-player games* (SMGs). These extend MDPs by allowing different *players* to resolve the nondeterminism (MDPs can thus be considered as 1-player stochastic games). SMGs allow us to reason about the strategic decisions of several agents either competing or collaborating to achieve some objective. We restrict our attention to *turn-based* stochastic games, in which a single player is responsible for the nondeterministic choices available in each state. We have the following formal definition.

Definition 10 (Stochastic multi-player game). A (turn-based) *stochastic multi-player game* (SMG) is a tuple $\mathsf{G}{=}(\Pi, S, (S_i)_{i\in\Pi}, \bar{s}, A, \delta, L)$, where:

- $(S, \bar{s}, A, \delta, L)$ represents an MDP (see Defn. 5);
- Π is a finite set of *players*;
- $(S_i)_{i\in\Pi}$ is a partition of S.

In a state s of an SMG G, the evolution is similar to an MDP: first an available action is nondeterministically chosen and then the successor state is chosen according to the distribution $\delta(s,a)$. The difference is that the nondeterministic choice is resolved by the *player* that controls the state s, that is, the player $i \in \Pi$ for which $s \in S_i$. As for MDPs, we can define the set of finite and infinite paths $FPaths_\mathsf{G}$ ($FPaths_\mathsf{G}(s)$) and $IPaths_\mathsf{G}$ ($IPaths_\mathsf{G}(s)$) of G. Furthermore, we can define reward structures for SMGs in the same way as for MDPs (see Defn. 6).

To resolve the nondeterminism in an SMG, we again use strategies, however we now define a separate strategy for each player of the game.

Definition 11 (SMG strategy). For an SMG $\mathsf{G}{=}(\Pi, S, (S_i)_{i\in\Pi}, \bar{s}, A, \delta, L)$, a strategy σ_i for player i of G is a function $\sigma_i : \{\pi \mid \pi \in FPaths_\mathsf{G} \wedge last(\pi) \in S_i\} \to Dist(A)$ such that, if $\sigma_i(\pi)(a){>}0$, then $a \in A(last(\pi))$. The set of all strategies for player $i \in \Pi$ in SMG G is denoted by Σ_G^i.

For an SMG $\mathsf{G}{=}(\Pi, S, (S_i)_{i\in\Pi}, \bar{s}, A, \delta, L)$ and strategies $\sigma_1, \ldots, \sigma_k$ for multiple players $1, \ldots, k$, we can combine them into a single strategy $\sigma = \sigma_1, \ldots, \sigma_k$ which controls the nondeterminism when the game is in the states $S_1 \cup \cdots \cup S_k$. If a combined strategy σ is constructed from all the players Π of G (sometimes called a *strategy profile*), then the nondeterminism is resolved in all the states of the game and, as for MDPs, we can construct probability measures $Pr_{\mathsf{G},s}^\sigma$ over the infinite paths of G.

To specify properties of SMGs, we consider an extension of the PRISM logic used earlier for DTMCs and MDPs, adding the *coalition* operator $\langle\langle C \rangle\rangle$ from al-

ternating temporal logic (ATL) [8]. The result is (essentially) the logic RPATL*
proposed in [29].[2]

Definition 12 (RPATL* syntax). The syntax of RPATL* is given by:

$$\phi ::= \texttt{true} \mid \texttt{a} \mid \neg\phi \mid \phi \wedge \phi \mid \langle\langle C \rangle\rangle \, \texttt{P}_{\bowtie p}[\psi] \mid \langle\langle C \rangle\rangle \, \texttt{R}^r_{\bowtie q}[\rho]$$

where path formulae ψ and reward formulae ρ are defined in identical fashion to the
PRISM logic in Defn. 3, $C \subseteq \Pi$ is a coalition of players, $\texttt{a} \in AP$, $\bowtie \in \{<, \leqslant, \geqslant, >\}$,
$p \in [0,1]$, r is a reward structure and $q \in \mathbb{R}_{\geqslant 0}$.

Intuitively, the formulae $\langle\langle C \rangle\rangle \, \texttt{P}_{\bowtie p}[\psi]$ and $\langle\langle C \rangle\rangle \, \texttt{R}^r_{\bowtie q}[\rho]$ mean that it is possible
for the players in C to collectively ensure that $\texttt{P}_{\bowtie p}[\psi]$ or $\texttt{R}^r_{\bowtie q}[\rho]$, respectively, is
satisfied, no matter what the other players of the game decide to do. We can also
adapt these to numerical queries, writing for example $\langle\langle C \rangle\rangle \, \texttt{P}_{\max=?}[\psi]$ to represent
the maximum probability of ψ that the players in C can ensure, regardless of the
choices of the other players of the game.

In order to formalise the semantics of RPATL*, we define *coalition games*.

Definition 13 (Coalition game). Given an SMG $\mathsf{G}=(\Pi, S, (S_i)_{i \in \Pi}, \bar{s}, A, \delta, L)$ and
coalition of players $C \subseteq \Pi$, the *coalition game* of G induced by C is the stochas-
tic two-player game $\mathsf{G}_C=(\{1,2\}, S, (S'_1, S'_2), \bar{s}, A, \delta, L)$ where $S'_1 = \cup_{i \in C} S_i$ and $S'_2 = \cup_{i \in \Pi \setminus C} S_i$.

Definition 14 (RPATL* semantics). For an SMG $\mathsf{G}=(\Pi, S, (S_i)_{i \in \Pi}, \bar{s}, A, \delta, L)$ and
reward structure $r = (r_S, r_A)$ for G, the satisfaction relation $\models$ is defined as in Defn. 9
except that, for any state $s \in S$:

$$
\begin{aligned}
\mathsf{G}, s \models \langle\langle C \rangle\rangle \, \texttt{P}_{\bowtie p}[\psi] \quad &\Leftrightarrow \quad \text{there exists } \sigma_1 \in \Sigma^1_{\mathsf{G}_C} \text{ such that, for any } \sigma_2 \in \Sigma^2_{\mathsf{G}_C}, \\
&\qquad \text{we have } Pr^{\sigma_1, \sigma_2}_{\mathsf{G}_C, s}\{\pi \in IPaths_{\mathsf{G}_C}(s) \mid \mathsf{G}, \pi \models \psi\} \bowtie p \\
\mathsf{G}, s \models \langle\langle C \rangle\rangle \, \texttt{R}^r_{\bowtie q}[\rho] \quad &\Leftrightarrow \quad \text{there exists } \sigma_1 \in \Sigma^1_{\mathsf{G}_C} \text{ such that, for any } \sigma_2 \in \Sigma^2_{\mathsf{G}_C}, \\
&\qquad \text{we have } \mathbb{E}^{\sigma_1, \sigma_2}_{\mathsf{G}_C, s}(rew^r(\rho)) \bowtie q
\end{aligned}
$$

where $\mathsf{G}_C=(S, (S'_1, S'_2), \bar{s}, A, \delta, L)$ is the coalition game of G induced by C.

As can be seen in Defn. 14, model checking of RPATL* reduces to the analysis of
stochastic two-player games. The exact complexity of analysing such games is a
long-standing open problem [32], but key properties such as reachability probabili-
ties and expected cumulated rewards can be efficiently approximated using methods
such as value iteration [33]. The overall model checking problem can be performed
in a similar manner to the algorithms described for model checking MDPs de-
scribed in Sect. 2.2. For further details, see [29]. SMG model checking has been ap-
plied to case studies across a number of application domains, including autonomous
transport, security protocols and energy management systems. See, for example,
[29, 111, 116] for details.

---

[2] Strictly speaking, the definition of reward operators differs in [29].

2.4 Tool Support

There are several software tools available for probabilistic model checking. One of the most widely used of these is PRISM [79], which incorporates the majority of the techniques covered in this chapter. In particular, it supports model checking of DTMCs and MDPs, as described above, as well as probabilistic automata, continuous-time Markov chains and probabilistic timed automata, which are discussed in later sections. PRISM-games [85] is an extension of PRISM for the verification of SMGs. Another widely used tool is MRMC [72], which can be used to analyse Markov chains and Markov reward models, and also has support for continuous-time MDPs (a model combining nondeterministic, probabilistic and real-time features, see Sect. 5). Other general purpose probabilistic model checking tools include the Modest Toolset [61], iscasMc [55] and PAT [103]. More specialised tools, focusing on techniques such as parametric model checking or abstraction refinement, are mentioned in the corresponding sections of this chapter. A more extensive list of available tools is maintained at [117].

3 Controller Synthesis

In this section, we describe a technique that is closely related to probabilistic model checking: *controller synthesis*. For probabilistic models that include nondeterminism, such as MDPs and SMGs, there are two, dual ways that we can reason about them. First, as done in the earlier sections of this chapter, we can *verify* that the model satisfies some formally specified property *for all* possible resolutions of nondeterminism. Secondly, we can *synthesise* a controller (i.e., a means of resolving the nondeterminism) under which a formally specified property is guaranteed to hold.

In this section, we describe controller synthesis techniques applied to MDPs. For SMGs, model checking of the logic RPATL*, discussed earlier, provides a good basis for controller synthesis in the context of multiple agents. Later, in Sect. 5, we will illustrate controller synthesis for real-time probabilistic systems using probabilistic timed automata.

3.1 Controller Synthesis for MDPs

To apply controller synthesis to a system modelled as an MDP, we use *strategy synthesis*, which generates a strategy under which a particular formally-specified property is guaranteed to be true. We focus on a subset of the PRISM logic from Defn. 3 comprising a single instance of a $P_{\bowtie p}[\cdot]$ or $R^r_{\bowtie q}[\cdot]$ operator. In particular, further instances of these operators are not allowed to be nested within path formulae or reward formulae (these cases are known to be more challenging [15, 25]).

A formal definition of strategy synthesis is given below. For this, we use a slightly different form of the satisfaction relation $\models$, where we write $\mathsf{M}, \sigma, s \models \phi$ to state that property ϕ is satisfied by MDP M under the strategy σ (which is essentially the same as the satisfaction of ϕ under the induced DTMC M_{σ}).

Definition 15 (Strategy synthesis). The *strategy synthesis* problem is: given an MDP M with initial state $\bar{s}$ and a formula ϕ of the form $\mathsf{P}_{\bowtie p}[\psi]$ or $\mathsf{R}^r_{\bowtie q}[\rho]$ (see Defn. 3), find, if it exists, a strategy $\sigma^* \in \Sigma_{\mathsf{M}}$ such that $\mathsf{M}, \sigma^*, \bar{s} \models \phi$.

Like for probabilistic model checking of MDPs, as discussed in Sect. 2.2, the problem of strategy synthesis for a $\mathsf{P}_{\bowtie p}[\psi]$ or $\mathsf{R}^r_{\bowtie q}[\rho]$ operator can be solved by computing an *optimal value* (i.e., minimum or maximum value) for ψ or ρ. For example, when attempting to synthesise a strategy for $\phi = \mathsf{P}_{\leqslant p}[\psi]$, we can compute $Pr^{\min}_{\mathsf{M},\bar{s}}(\psi)$. Then, there exists a strategy σ^* satisfying ϕ if and only if $Pr^{\min}_{\mathsf{M},\bar{s}}(\psi) \leqslant p$, in which case we can take σ^* to be a corresponding *optimal strategy*, i.e., one that achieves the optimal value $Pr^{\min}_{\mathsf{M},\bar{s}}(\psi)$. So, in general, rather than fixing a specific bound p, we can just use a numerical query such as $\mathsf{P}_{\min=?}[\psi]$ to specify a strategy synthesis problem, and directly compute an optimal value and strategy for it.

We already sketched the techniques required to compute optimal values for such properties of MDPs in Sect. 2.2. In the sections below, we recap the required computations, additionally discussing which classes of strategies need to be considered for optimality (i.e., the smallest class of strategies guaranteed to contain an optimal one) and the methods required to generate them.

3.1.1 Probabilistic Reachability

For probabilistic reachability queries $\mathsf{P}_{\min=?}[\mathsf{F}\,a]$ or $\mathsf{P}_{\max=?}[\mathsf{F}\,a]$, *memoryless deterministic* strategies achieve optimal values, and so this class of strategy suffices for strategy synthesis. Determining optimal probability values requires an analysis of the underlying graph structure of the MDP, followed by a numerical computation phase using, for example, linear programming, policy iteration or value iteration.

The construction of an optimal strategy σ^* then depends on the method used in the numerical computation phase. Policy iteration is the most direct as an optimal strategy is constructed as part of the algorithm. For the remaining methods, the optimal strategy corresponds to selecting locally optimal actions in each state, although maximum probabilities require care.

In the case of a bounded reachability query $\mathsf{P}_{\min=?}[\mathsf{F}^{\leqslant k}\,a]$ or $\mathsf{P}_{\max=?}[\mathsf{F}^{\leqslant k}\,a]$, memoryless strategies do not achieve optimal values and instead we need to consider the larger class of *finite-memory deterministic* strategies. Strategy synthesis and the computation of optimal reachability probabilities for step-bounded reachability corresponds to working backwards through the MDP and determining, at each step, the actions that yields optimal probabilities in each state.

Example 4. We now return to the MDP M from Fig. 4 and synthesise a strategy for the numerical probabilistic reachability query $\mathsf{P}_{\max=?}[\mathsf{F}\,\mathsf{goal}_1]$. Therefore, we first compute the optimal value $Pr^{\max}_{\mathsf{M},s_0}(\mathsf{F}\,\mathsf{goal}_1)$, which we find equals 0.5. Synthesising
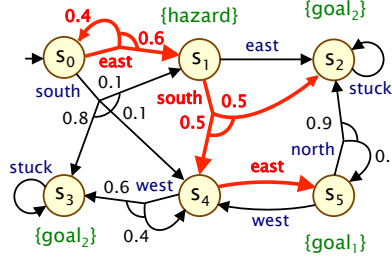
Fig. 5 Running example: an MDP M representing a robot moving about a 3×2 grid.

an optimal strategy, we find the memoryless deterministic strategy (see Fig. 5) that selects *east* in s_0, *south* in s_1 and *east* in s_4 (there is no choice needed in s_2 or s_3, and the choice in s_5 is not relevant as the target $goal_1$ has been reached).

Next, consider the bounded probabilistic reachability query $P_{max=?}[F^{\leq k} goal_2]$. We find that the maximum probability equals 0.8, 0.96 and 0.99 for $k = 1, 2$ and 3, respectively. In the case where $k=3$, the synthesised strategy is deterministic and finite-memory. In particular the strategy, when arriving in state s_4, after 1 step, selects *east* (since $goal_2$ is reached with probability 0.9). On the other hand, arriving in state s_4 after 2 steps, the strategy selects *west* (since otherwise $goal_2$ cannot be reached within $k-2=1$ steps). ∎

3.1.2 Reward Properties

Strategy synthesis for a numerical reward query $R^r_{min=?}[\rho]$ or $R^r_{max=?}[\rho]$ is similar to probabilistic reachability queries. In the case of reachability rewards, i.e. when ρ is of the form $F\ a$, as for unbounded probabilistic reachability, first there is a pre-computation phase (identifying the states for which the expected reward is infinite), and then a numerical computation phase using methods such as value iteration, policy iteration or linear programming. As for unbounded probabilistic reachability, it is sufficient to consider the class of memoryless and deterministic strategies. For unbounded cumulative rewards, i.e. when ρ is of the form C, one must additionally identify the maximal end components containing non-zero rewards.

For bounded cumulative rewards ($\rho = C^{\leq k}$) and instantaneous rewards ($\rho = I^{=k}$) the situation is the same as for bounded probabilistic reachability: the class of deterministic finite-memory strategies are required and a strategy can be synthesised by stepping backwards through the MDP.

Example 5. Returning to our running example we consider strategy synthesis for the query $R^{moves}_{min=?}[F\ goal_2]$ where the reward structure *moves* returns 1 for all state-action pairs and all state rewards are zero. This will therefore return a strategy that minimises the expected number of *moves* that the robot needs to make to reach a state satisfying $goal_2$. We find that the minimum expected number of steps equals $\frac{19}{15}$

and the synthesised memoryless deterministic strategy (not represented in the figure) chooses the actions *south*, *east*, *west* and *north* in s_0, s_1, s_4 and s_5 respectively. ■

3.1.3 LTL Properties

We now consider strategy synthesis for a numerical query of the form $P_{min=?}[\psi]$ or $P_{max=?}[\psi]$ where ψ is an LTL formula. For a given MDP M, the problem can be reduced to the strategy synthesis of a reachability query (see Sect. 3.1.1) on the product of M and a deterministic Rabin automaton (DRA) representing ψ [3]. Since for any strategy σ we have:

$$Pr^{\sigma}_{M,\bar{s}}(\psi) \ = \ 1 - Pr^{\sigma}_{M,\bar{s}}(\neg\psi)$$

the problem of finding a minimum probability and strategy for achieving this value can be reduced to finding the maximum probability and corresponding strategy by considering the negated LTL formula. Hence, for the remainder of this section we restrict our attention to the case of maximum numerical queries.

For any LTL formula ψ using atomic propositions from *AP*, we can construct a DRA A_ψ with alphabet 2^{AP} that represents it [108, 34]. More precisely, we have that an infinite path $\omega = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \cdots$ of M satisfies ψ if and only if the infinite word $L(s_0)L(s_1)L(s_2)\dots$ is in the language of A_ψ. To perform strategy synthesis, we proceed by constructing the product MDP, denoted $M \otimes A_\psi$, of M and A_ψ. Next we find the maximal end components of this MDP which meet the acceptance conditions of A_ψ and label the states of these components with the atomic proposition acc. This then reduces the problem to a maximum probabilistic reachability query since:

$$Pr^{max}_{M,\bar{s}}(\psi) = Pr^{max}_{M \otimes A_\psi,(\bar{s},\bar{q})}(F \text{ acc}).$$

We can now follow the approach described in Sect. 3.1.1 to synthesise a *memoryless* deterministic strategy for $M \otimes A_\psi$ which maximises the probability of reaching accepting end components (and then stays in those end components, visiting each state infinitely often). This strategy can then be used to construct an optimal *finite-memory deterministic* strategy of M for the query $P_{max=?}[\psi]$.

Example 6. Returning again to the running example of a robot (Fig. 4), we consider synthesising a strategy for the query $P_{max=?}[(G \neg hazard) \wedge (G F goal_1)]$. This corresponds to finding a strategy which maximises the probability of avoiding a hazard-labelled state *and* visiting a $goal_1$ state infinitely often. We find that the maximum probability equals 0.1 and that, in this case, a memoryless strategy suffices for achieving optimality. The synthesised strategy selects *south* in state s_0, which leads to state s_4 with probability 0.1. We then remain in states s_4 and s_5 indefinitely by choosing actions *east* and *west*, respectively. ■

3.2 Multi-objective Controller Synthesis

We now extend the synthesis problem to *multi-objective* queries, this concerns find-
ing a strategy σ that simultaneously satisfies multiple quantitative properties. We
first describe the case for LTL properties and then outline how this can be extended.

Definition 16 (Multi-objective probabilistic LTL). A *multi-objective probabilistic
LTL property* is a conjunction $\phi = P_{\bowtie_1 p_1}[\psi_1] \wedge \ldots \wedge P_{\bowtie_n p_n}[\psi_n]$ where $\psi_1, \ldots \psi_n$ are
LTL formulae and, for $1 \leqslant i \leqslant n$, $\bowtie_i \in \{<, \leqslant, \geqslant, >\}$ and $p_i \in [0, 1]$. For MDP M and
strategy σ, we have $M, \sigma, \bar{s} \models \phi$ if $M, \sigma, \bar{s} \models P_{\bowtie_i p_i}[\psi_i]$ for all $1 \leqslant i \leqslant n$.

The first algorithm for multi-objective probabilistic LTL strategy synthesis was pre-
sented in [42]. Here we outline an adapted version of this, based on [45], which
uses DRAs. The overall approach is similar to standard (single-objective) LTL strat-
egy synthesis in that it constructs a product automaton and reduces the problem to
(multi-objective) reachability.

First, for each $1 \leqslant i \leqslant n$, we can ensure that $\bowtie_i$ is a lower bound ($\geqslant$ or $>$) in each
formula $P_{\bowtie_i p_i}[\psi_i]$ by negating the formulae ψ_i where necessary. The next step is to
build a DRA A_{ψ_i} to represent each LTL formula. Using these automata we then build
the product MDP $M' = M \otimes A_{\psi_1} \otimes \cdots \otimes A_{\psi_n}$. For each combination $X \subseteq \{1, \ldots, n\}$ of
objectives we find the end components of M' that are accepting for each of the DRAs
in the set $\{A_i \mid i \in X\}$. A special sink state for X is then added to the product MDP
M' for X where for $1 \leqslant i \leqslant n$ we label this sink with $\mathsf{acc_i}$ if and only if $i \in X$ and we
add transitions from states in the end components found to this sink state. After we
have added these components, the problem on M reduces to a multi-objective prob-
abilistic reachability problem on M' of the form $P_{\bowtie_1 p_1}[F\ \mathsf{acc}_1] \wedge \ldots \wedge P_{\bowtie_n p_n}[F\ \mathsf{acc}_n]$
which can be solved through a linear programming (LP) problem [42], or a value
iteration based solution method [46].

The class of strategies required for multi-objective probabilistic LTL is *finite-
memory* and *randomised*. A memoryless randomised strategy for the product au-
tomaton M' can be obtained, for example, directly from the solution of the LP prob-
lem and then, similarly to LTL objectives (in Sect. 3.1.3), we can convert this to a
finite-memory, randomised strategy for M.

We now summarise several useful extensions and improvements. For details of the
algorithms and any restrictions or assumptions that are required see the relevant
references.

Boolean combinations of LTL objectives. The approach can be extended to gen-
eral Boolean combinations of formulae, rather than just conjunctions as presented
in Defn. 16. This is achieved by translating into disjunctive normal form [42, 45].

Expected reward objectives. One can allow unbounded cumulative reward for-
mulae in addition to LTL formulae. The method outlined above has been extended
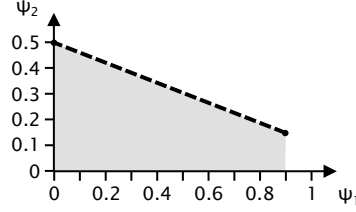in [45] to include such reward formulae. In addition, an alternative approach, using

Fig. 6 Pareto curve (dashed line) for maximisation of the probabilities of LTL formulae $\psi_1 = \text{G} \neg\text{hazard}$ and $\psi_2 = \text{G F goal}_1$ (see Example 7).

value iteration, presented in [46], allows bounded cumulative reward formulae. This approach has also been shown to provide significant efficiency gains in practice.

Numerical multi-objective queries. One can again consider numerical queries which return optimal values rather than `true` or `false`. For example, rather than synthesising a strategy satisfying $P_{\bowtie_1 p_1}[\psi_1] \wedge P_{\bowtie_2 p_2}[\psi_2]$, we can instead find a strategy that maximises the probability of satisfying the path formula ψ_1, whilst simultaneously satisfying $P_{\bowtie_2 p_2}[\psi_2]$. The method outlined above using linear programming is easily extended to handle such numerical queries through the addition of an objective function.

Pareto queries. To analyse the trade-off between multiple objectives we can construct the corresponding *Pareto curve* or an approximation of it [46]. For example, suppose we are interested in maximising the probabilities of two LTL formulae ψ_1 and ψ_2 for the MDP M, then the Pareto curve consists of the bounds $(p_1, p_2) \in [0,1]^2$ such that:

- there exists a strategy σ such that $Pr^\sigma_{M,\bar{s}}(\psi_1) \geqslant p_1$ and $Pr^\sigma_{M,\bar{s}}(\psi_2) \geqslant p_2$;
- if either bound p_1 or p_2 is increased, no strategy σ exists satisfying $Pr^\sigma_{M,\bar{s}}(\psi_1) \geqslant p_1$ and $Pr^\sigma_{M,\bar{s}}(\psi_2) \geqslant p_2$ without decreasing the other bound.

Example 7. We return again to the robot example presented in Fig. 4. Recall that, in Example 6, we considered the numerical query $P_{\max=?}[(\text{G} \neg\text{hazard}) \wedge (\text{G F goal}_1)]$ and found that the optimal probability was 0.1. Instead here we consider each conjunct of the LTL formula as a separate objective and, to ease notation, let $\psi_1 = \text{G} \neg\text{hazard}$ and $\psi_2 = \text{G F goal}_1$.

Consider the numerical multi-objective query that maximises the probability of satisfying ψ_2 whilst satisfying $P_{\geqslant 0.7}[\psi_1]$. We find that the optimal value, i.e. the maximum probability for satisfying ψ_2, equals $\frac{41}{180} \approx 0.2278$. The corresponding strategy is randomised and, in state s_0, chooses *east* with probability approximately 0.3226 and *south* with probability approximately 0.6774.

Finally, the Pareto curve for maximising the probabilities of the LTL formulae ψ_1 and ψ_2 is presented in Fig. 6. The dashed line in the figure form the Pareto curve, while the grey shaded area below shows all points (x, y) for which there is a strategy satisfying $P_{\geqslant x}[\psi_1] \wedge P_{\geqslant y}[\psi_2]$. ∎

4 Modelling and Verification of Large Probabilistic Systems

In practice, models of real-life systems are often large and complex, and their state space has a tendency to grow exponentially with the size of the system itself, a phenomenon known as the state space explosion problem.

In this section, we discuss some approaches that facilitate both the modelling and verification of large complex probabilistic systems. We first describe higher level modelling of systems comprising multiple components using the notion of parallel composition. Then we describe verification techniques designed to scale up to large, complex systems. Many such approaches exist; examples include symbolic model checking [14, 94], partial order reduction [50], symmetry reduction [76, 39], and bisimulation minimisation [71]. In this presentation, we focus on two particular methods: *compositional verification*, using an assume-guarantee framework, and *abstraction refinement*. We conclude the section with a case study that illustrates both of these techniques.

4.1 Compositional Modelling of Probabilistic Systems

Complex system designs usually comprise multiple components operating in parallel. For such a system, if there is probabilistic behaviour present in the system, then an MDP is the natural mathematical model for the system as nondeterminism can be used to represent the concurrency between the components. However, for compositional modelling and analysis, probabilistic automata (PAs) [100, 101], a minor generalisation of MDPs, are a more suitable formalism. The key difference is that states of a PA can have more than one transition labelled by the same action.

Definition 17 (Probabilistic automaton). A *probabilistic automaton* (PA) is a tuple $M=(S,\bar{s},A,\delta,L)$, where:

- S is a finite set of states;
- $\bar{s} \in S$ is an initial state;
- A is a finite alphabet;
- $\delta \subseteq S \times (A \cup \{\tau\}) \times Dist(S)$ is a finite probabilistic transition relation;
- $L : S \to 2^{AP}$ is a state labelling function.

The difference from Defn. 5 is that we now have a transition relation as opposed to a transition function and allow transitions to be labelled with the silent action τ, representing transitions that are internal to the component being represented.

The notions basic for MDPs presented in Sect. 2.2, such as paths and strategies, carry over straightforwardly to PAs. Reward structures for PAs can be defined in exactly the same way as for MDPs (see Defn. 6), although here a strategy chooses a particular transition (element of δ) as opposed to an action in a state. The semantics of the PRISM logic (see Defn. 3) and corresponding model checking algorithm presented in Sect. 2.2 for MDPs also carry over to PAs.

We next describe parallel composition of PAs, first introduced in [100, 101].

Definition 18 (Parallel composition of PAs). If $M_i = (S_i, \bar{s}_i, A_i, \delta_i, L_i)$ are PAs for $i=1,2$, then their *parallel composition* $M_1 \| M_2 = (S_1 \times S_2, (\bar{s}_1, \bar{s}_2), A_1 \cup A_2, \delta, L)$ is the PA where $((s_1, s_2), a, \mu) \in \delta$ if and only if one of the following holds:

- $a \in A_1 \cap A_2$, $\mu = \mu_1 \times \mu_2$, $(s_1, a, \mu_1) \in \delta_1$ and $(s_2, a, \mu_2) \in \delta_2$;
- $a \in A_1 \setminus A_2$, $\mu = \mu_1 \times \eta_{s_2}$ and $(s_1, a, \mu_1) \in \delta_1$;
- $a \in A_2 \setminus A_1$, $\mu = \eta_{s_1} \times \mu_2$ and $(s_2, a, \mu_2) \in \delta_2$;

and $L((s_1, s_2)) = L_1(s_1) \cup L_2(s_2)$.

The above definition allows several components to synchronise over the same action, so called multi-way synchronisation, as used by the process algebra CSP [99]. It also assumes that the components M_1 and M_2 synchronise over their common actions $A_1 \cap A_2$. Defn. 18 can easily be adapted to use other definitions of synchronisation, such as the two-way synchronisation used by the process algebra CCS [88], or to incorporate additional process algebraic operators for hiding or renaming actions.

Below, we demonstrate how a reward structure for a system can be constructed from the reward structures of its components. In this definition we have used addition as this is used in later case studies, however we can easily use other arithmetic operations depending on the quantities that the reward structure represents.

Definition 19. If $M_i = (S_i, \bar{s}_i, A_i, \delta_i, L_i)$ are PAs with reward structures $r_i = (r_{S_i}, r_{A_i})$ for $i=1,2$, then the composed reward structure $r = (r_S, r_A)$ for $M_1 \| M_2$ is such that for any $(s_1, s_2) \in S_1 \times S_2$ and $a \in A_1 \cup A_2$:

$$
\begin{aligned}
r_S((s_1, s_2)) &= r_{S_1}(s_1) + r_{S_2}(s_2) \\
r_A((s_1, s_2), a) &= \begin{cases} r_{A_1}(s_1, a) + r_{A_2}(s_2, a) & \text{if } a \in A_1 \cap A_2 \\ r_{A_1}(s_1, a) & \text{if } a \in A_1 \setminus A_2 \\ r_{A_2}(s_2, a) & \text{if } a \in A_2 \setminus A_1. \end{cases}
\end{aligned}
$$

4.2 Compositional Probabilistic Model Checking

We now describe an approach for *compositional* verification of probabilistic automata presented in [80], based on the popular *assume-guarantee* paradigm. This allows the verification of complex system to be performed through the analysis of individual components of the system in isolation, rather than verifying the much larger complete system. We begin by defining the underlying concepts and then illustrate two of the assume-guarantee proof rules.

The approach is based on the use of linear-time, *action-based* properties Ψ, which are defined in terms of the actions that label the transitions of a probabilistic automaton (or MDP). This is in contrast to the properties discussed elsewhere in this chapter, which are defined in terms of the atomic propositions that label states.[3] More precisely, a property Ψ represents a set of infinite words over the set A of

---

[3] In fact, state and action-labelled variants of temporal logics are equally expressive [90].

action labels of a probabilistic automaton M. An infinite path ω of M satisfies Ψ, written $M, \omega \models \Psi$, if the *trace* of π (the sequence of actions labelling its transitions, ignoring silent τ actions) is in the set of infinite words defining Ψ. Then, following the same style as the other property specifications introduced earlier, the property $P_{\bowtie p}[\Psi]$ states that, for all strategies of the probabilistic automaton M, the probability of a path satisfying Ψ is within the interval given by $\bowtie p$.

We focus our attention here on compositional verification of a class of linear-time action-based properties called *regular safety properties*.

Definition 20 (Regular safety property). A *safety property* Ψ_P represents a set of infinite words over an alphabet α which is characterised by a set of 'bad prefixes': finite words of which any extension is *not* in the set. A *regular safety property* is a safety property whose set of bad prefixes can be represented as a regular language.

Probabilistic safety properties are of the form $P_{\geqslant p}[\Psi_P]$, where Ψ_P is a regular safety property. These can be used to capture a variety of useful properties of probabilistic models, including:

- the probability of no failures occurring is at least 0.99;
- event a always occurs before event b with probability at least 0.75;
- the probability of completing a task within k steps is at least 0.9.

A technical requirement of the compositional verification approach described here is the use of *partial strategies*, which can opt to (with some probability) take none of the available actions and remain in the current state. In [80] it is shown that by considering only *fair* strategies, that is the strategies that choose an action from each component of the system infinitely often, this requirement can be removed. We first define the alphabet extension of a PA.

Definition 21 (Alphabet extension of PA). For a PA $M=(S, \bar{s}, A, \delta, L)$ and set of actions α, we *extend* M's alphabet to α, denoted by the PA $M[\alpha]$, as follows: $M[\alpha]=(S, \bar{s}, A \cup \alpha, \delta', L)$ where $\delta'=\delta \cup \{(s, a, \eta_s) \mid s \in S \wedge a \in \alpha \setminus A\}$.

The approach uses *probabilistic assume-guarantee triples*. These take the form $\langle P_{\geqslant p_A}[\Psi_A] \rangle M \langle P_{\geqslant p_G}[\Psi_G] \rangle$ where Ψ_A, Ψ_G are regular safety properties (see Defn. 20) and M is a PA. Informally, the triple means: 'whenever M is part of a system satisfying Ψ_A with probability at least p_A, the system satisfies Ψ_G with probability at least p_G'. Formally, we have the following definition.

Definition 22 (Probabilistic assume-guarantee triple). If Ψ_A, Ψ_G are regular safety properties, $p_A, p_G \in [0, 1]$ bounds, $M=(S, \bar{s}, A, \delta, L)$ is a PA and $\alpha_G \subseteq \alpha_A \cup A$, then $\langle P_{\geqslant p_A}[\Psi_A] \rangle M \langle P_{\geqslant p_G}[\Psi_G] \rangle$ is a *probabilistic assume-guarantee triple*, meaning:

$$\forall \sigma \in \Sigma_{M[\alpha_A]} \cdot \left(M[\alpha_A], \sigma, \bar{s} \models P_{\geqslant p_A}[\Psi_A] \rightarrow M[\alpha_A], \sigma, \bar{s} \models P_{\geqslant p_G}[\Psi_G] \right).$$

The use of $M[\alpha_A]$, i.e. M extended to the alphabet of Ψ_A, in the above is needed to allow the assumption to refer to actions not used in M. Verifying that a triple $\langle P_{\geqslant p_A}[\Psi_A] \rangle M \langle P_{\geqslant p_G}[\Psi_G] \rangle$ holds requires the use of multi-objective model checking, as discussed in Sect. 3.2, as the following proposition demonstrates.

Proposition 1 ([80]). *If* Ψ_A, Ψ_G *are regular safety properties,* $p_A, p_G \in [0,1]$ *and* M *is a PA, then* $\langle P_{\geqslant p_A}[\Psi_A] \rangle M \langle P_{\geqslant p_G}[\Psi_G] \rangle$ *if and only if*

$$\neg \exists \sigma \in M[\alpha_A] \,.\, \left(M[\alpha_A], \sigma, \bar{s} \models P_{\geqslant p_A}[\Psi_A] \wedge M[\alpha_A], \sigma, \bar{s} \not\models P_{\geqslant p_G}[\Psi_G] \right).$$

Based on the definitions given above, [80] presents the following *asymmetric* assume-guarantee proof rule for a two component system $M_1 \| M_2$.

Proposition 2 ([80]). *If* Ψ_A, Ψ_G *are regular safety properties,* $p_A, p_G \in [0,1]$ *and* M_1, M_2 *are PAs such that* $\alpha_A \subseteq A_1$ *and* $\alpha_G \subseteq A_2 \cup \alpha_A$, *then the following proof rule holds:*

$$\frac{\begin{array}{c} M_1, \bar{s}_1 \models P_{\geqslant p_A}[\Psi_A] \\ \langle P_{\geqslant p_A}[\Psi_A] \rangle M_2 \langle P_{\geqslant p_G}[\Psi_G] \rangle \end{array}}{M_1 \| M_2, (\bar{s}_1, \bar{s}_2) \models P_{\geqslant p_G}[\Psi_G]} \quad (\text{ASYM})$$

Proposition 2 presents a method to verify the property $P_{\geqslant p_G}[\Psi_G]$ on $M_1 \| M_2$ in a *compositional* fashion. More precisely, verification reduces to two sub-problems, one for each premise of the rule:

1. computing the optimal probability of a regular safety property on M_1;
2. performing multi-objective model checking on $M_2[\alpha_A]$.

A limitation of the above rule is the fact it is asymmetric: we analyse the component M_2 using an assumption about the component M_1, but when verifying M_1 we cannot make any assumptions about M_2. Below, we give a proof rule which does allow the use of additional assumptions in this way.

Proposition 3 ([80]). *If* $\Psi_{A_1}, \Psi_{A_2}, \Psi_G$ *are regular safety properties,* $p_{A_1}, p_{A_2}, p_G \in [0,1]$ *and* M_1, M_2 *are PAs such that* $\alpha_{A_2} \subseteq A_2$, $\alpha_{A_1} \subseteq A_2 \cup \alpha_{A_2}$ *and* $\alpha_G \subseteq A_1 \cup \alpha_{A_1}$, *then the following proof rule holds:*

$$\frac{\begin{array}{c} M_2, \bar{s}_2 \models P_{\geqslant p_{A_2}}[\Psi_{A_2}] \\ \langle P_{\geqslant p_{A_2}}[\Psi_{A_2}] \rangle M_1 \langle P_{\geqslant p_{A_1}}[\Psi_{A_1}] \rangle \\ \langle P_{\geqslant p_{A_1}}[\Psi_{A_1}] \rangle M_2 \langle P_{\geqslant p_G}[\Psi_G] \rangle \end{array}}{M_1 \| M_2, (\bar{s}_1, \bar{s}_2) \models P_{\geqslant p_G}[\Psi_G]} \quad (\text{CIRC})$$

For further details of the assume-guarantee proof rules, including extensions to allow both ω-regular properties and reward-based properties, see [80].

4.3 Quantitative Abstraction Refinement

An alternative way to verify large, complex systems is using *abstraction-refinement* techniques, which have been established as one of the most effective ways of performing *non-probabilistic* model checking on complex systems [31]. The basic idea

is to build a small abstract model, by removing details of the complex concrete system which are not relevant to the property of interest, which is consequently easier to analyse. The abstract model is constructed in such a way that, when the property of interest is verified `true` for the abstraction, the property also holds for the concrete system. On the other hand, if the property does not hold for the abstraction, then information from the model checking process (typically a counterexample) is used either to show that the property is false in the concrete system or to refine the abstraction. This process forms the basis of a loop which refines the abstraction until the property is shown either to be `true` or `false` in the concrete system.

In the case of probabilistic model checking a number of abstraction-refinement approaches exist. D'Argenio et al. [35] introduce an approach for verifying reachability properties of PAs based on probabilistic simulation [100] and implement a corresponding tool RAPTURE [66]. Properties are analysed on abstractions obtained through successive refinements, starting from a coarse partition derived from the property under study. This approach only produces lower (upper) bounds for the minimum (maximum) reachability probabilities. Based on [35] and using predicate abstraction [48], an abstraction-refinement framework for PAs is developed in [110, 63] and implemented in the PASS tool [53]. The framework is used for verifying or refuting properties of the form 'the maximum probability of error is at most p' for a given probability threshold p. Since abstractions produce only upper bounds on maximum probabilities, to refute a property, probabilistic counterexamples [58] (comprising multiple paths whose combined probability exceeds p) are generated. If these paths are spurious, then they are used to generate further predicates using interpolation.

An alternative framework is presented in [74] where the key idea is to maintain a distinction between the nondeterminism from the original PA and the nondeterminism introduced during the abstraction process. To achieve this, abstractions of PAs are modelled as stochastic two player games (see Sect. 2.3), where the two players correspond to the two different forms of nondeterminism. Analysis of these abstract models results in a separate lower and upper bound for the property of interest (e.g. an optimal reachability probability or expected reward value). These bounds both provide quantitative measure of the quality (or preciseness) of the abstraction and an indication of how to improve it. The abstraction-refinement framework is presented in Fig. 7. The framework starts with a simple, coarse abstraction (i.e. partition of the state space) and then refines the abstraction until the difference between the bounds is below some threshold value ε. Two methods for automatically refining abstractions are considered. The first is based on the difference between specific strategies that achieve the lower and upper bounds. The second method differs by considering all the strategies that achieve the bounds. In [73] and [78], this game-based abstraction and refinement framework has been used to develop verification techniques for probabilistic software and probabilistic timed automata (see Sect. 5.1), respectively.
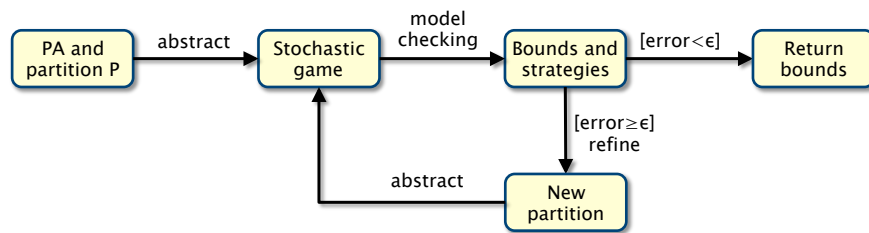
Fig. 7 Quantitative abstraction-refinement framework for PAs [74].

4.4 Case Study: The Zeroconf Protocol

This case study concerns the ZeroConf dynamic configuration protocol for IPv4 link-local addresses [30] used to enable devices to connect to a local network. The protocol is a distributed 'plug-and-play' solution to IP address configuration. This case study was originally introduced and analysed using probabilistic model checking in [81]. The compositional approach present in Sect. 4.2 and abstraction-refinement framework present in Sect. 4.3 have since been used to analyse this model in [80] and [74], respectively.

The protocol is used to configure an IP address for a device when it first joins a local network. This address is then used for the communication between the device and others within the network. When connecting to the network, a device first randomly selects an address from the 65,024 possible local IP addresses. The device then waits a random time of between 0 and 2 seconds before sending a number of probes including the chosen address over 4 second intervals. These probes are sent all of the other hosts of the network and are used to check whether any other device is using the chosen address. If the original device gets a message back saying the address is already in use it will restart the protocol by reconfiguring. If the host repeats this process 10 times, it 'backs off' and remains idle for at least one minute. If the host does not get a reply to any of the probes it commences to use the chosen IP address. We assume that messages can also get lost with a fixed probability.

The model of this system studied in [80] consists of the parallel composition of two PAs: one automaton representing a new device joining the local network and the other representing the environment, i.e. the existing network including the devices present in the network. Using the composition approach, [80] analysed the following properties:

- the minimum probability that the new host employs a fresh IP address;
- the minimum probability that the new host is configured by time T;
- the minimum probability that the protocol terminates;
- the minimum and maximum expected time for the protocol to terminate.

The first two can be expressed using regular safety properties and were verified compositionally by applying the rules presented in Proposition 2 and Proposition 3. In the case of the final two properties extensions of the rules to LTL and reward properties were used.
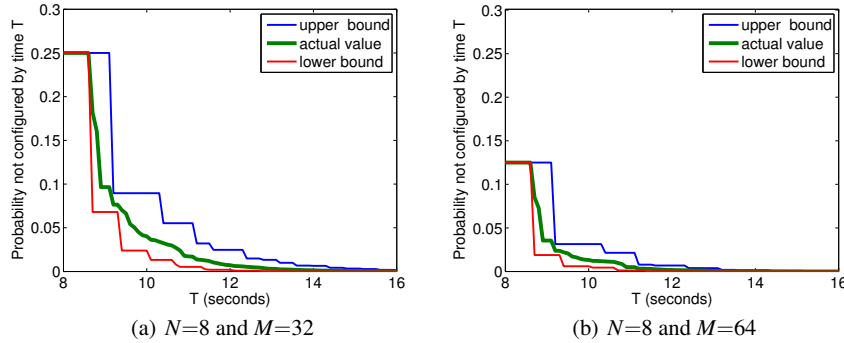
(a) N=8 and M=32   (b) N=8 and M=64

Fig. 8 Zeroconf case study: maximum probability device not configured successfully by T [74]

The case study developed in [74] using the game based abstraction-refinement described in Sect. 4.3 also considers a new device joining a network. The network consists of N devices and there are M available addresses. The model is the parallel composition of $2 \cdot N$+1 component PAs: the device joining the network and N pairs of channels for the two-way communication between the new device and each of the configured devices. Fig. 8 presents, for a fixed abstraction, the upper and lower bounds obtained when calculating the maximum probability that the new device has not configured successfully by time T. The figure also includes the results obtained when model checking the concrete system. The graphs demonstrate how the differences between the lower and upper bounds can be used to quantify the utility of the abstraction. For the fixed abstraction, the number of states in the abstraction is independent of M and equals 881, on the other hand, the concrete system has 432,185 states when M=32 and 838,905 states when M=64.

5 Real-Time Probabilistic Model Checking

So far, we have seen *discrete-time* models which exhibit just probabilistic behaviour (DTMCs) or both probabilistic and nondeterministic behaviour (MDPs and SMGs). However it is also often important to model the *real-time* characteristics and the interplay between the different types of behaviour. Relevant application areas range from wireless communication, automotive networks to security protocols. We will first give an overview of *probabilistic timed automata*, a formalism that allows for the modelling of systems exhibiting nondeterministic, probabilistic and real-time behaviour and a case study using this formalism. The final part of this section concerns *continuous-time Markov chains*, which are also suitable for modelling systems with probabilistic and real-time characteristics.

5.1 Probabilistic Timed Automata

Probabilistic timed automata (PTAs) [68, 83, 19] extend classical timed automata [7] with discrete probabilistic choice. Real-time behaviour is modelled through *clocks* which are variables whose values range over the non-negative reals and increase at the same rate as time. For the remainder of this section we assume a finite set of clocks $\mathscr{X}$. Before we give the formal definition of PTAs we require the following notation and definitions relating to clocks.

A function $v : \mathscr{X} \to \mathbb{R}_{\geqslant 0}$ is called a *clock valuation* and we denote the set of all clock valuations by $\mathbb{R}_{\geqslant 0}^{\mathscr{X}}$. For a clock valuation $v \in \mathbb{R}_{\geqslant 0}^{\mathscr{X}}$, real-time delay $t \in \mathbb{R}_{\geqslant 0}$ and set of clocks $X \subseteq \mathscr{X}$, we use $v+t$ to denote the clock valuation obtained from v by incrementing all clock values by t and $v[X{:=}0]$ for the clock valuation obtained from v by resetting the clocks in X to 0. We let $\mathbf{0}$ denote the clock valuation that assigns 0 to all clocks in $\mathscr{X}$. We define the set of *clock constraints* over $\mathscr{X}$, denoted $CC(\mathscr{X})$, by the syntax:

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

where $x, y \in \mathscr{X}$ and $c, d \in \mathbb{N}$. A clock valuation v satisfies a clock constraint ζ, denoted $v \models \zeta$, if the constraint ζ resolves to $\texttt{true}$ after substituting the occurrences of each clock x with $v(x)$.

Definition 23 (PTA syntax). A *probabilistic timed automaton* (PTA) is a tuple of the form $\mathsf{P}=(L, \bar{l}, \mathscr{X}, Act, inv, enab, prob, L_{\mathsf{P}})$ where:

- L is a finite set of *locations* and $\bar{l} \in L$ is an *initial location*;
- $\mathscr{X}$ is a finite set of *clocks*;
- Act is a finite set of *actions*;
- $inv : L \to CC(\mathscr{X})$ is an *invariant condition*;
- $enab : L{\times}Act \to CC(\mathscr{X})$ is an *enabling condition*;
- $prob : L{\times}Act \to Dist(2^{\mathscr{X}}{\times}L)$ is a (partial) *probabilistic transition function*;
- $L_{\mathsf{P}} : L \to 2^{AP}$ is a labelling function.

A *state* of a PTA is a pair $(l, v) \in L{\times}\mathbb{R}_{\geqslant 0}^{\mathscr{X}}$ such that $v \models inv(l)$. In any state (l, v), there is a nondeterministic choice between either a certain amount of time elapsing, or an action being performed. If time elapses, then the choice of time $t \in \mathbb{R}_{\geqslant 0}$ requires that the invariant $inv(l)$ remains continuously satisfied while time passes. The resulting state after this transition is $(l, v+t)$. In the case where an action is performed, an action a can only be chosen if it is *enabled*, i.e., if the clock constraint $enab(l, a)$ is satisfied by v. Once an enabled action a is chosen, a set of clocks to reset and a successor location are selected at random, according to the distribution $prob(l, a)$.

Definition 24 (PTA semantics). Let $\mathsf{P}=(L, \bar{l}, \mathscr{X}, Act, inv, enab, prob, L_{\mathsf{P}})$ be a PTA. The semantics of P is defined as the (infinite-state) MDP $[\![\mathsf{P}]\!] = (S, \bar{s}, A, \delta, L)$ where:

- $S = \{(l, v) \in L \times \mathbb{R}_{\geqslant 0}^{\mathscr{X}} \mid v \models inv(l)\}$;
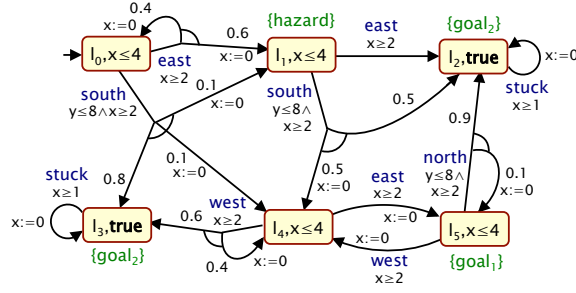- $\bar{s} = (\bar{l}, \mathbf{0})$;

Fig. 9 Running example: a PTA P representing a robot moving about a 3×2 grid.

- $A = \mathbb{R}_{\geqslant 0} \cup Act$;
- for any $(l,v) \in S$ and $a \in Act \cup \mathbb{R}_{\geqslant 0}$, we have $\delta((l,v),a) = \lambda$ if and only if either:

  – *(time transitions)* $a \in \mathbb{R}_{\geqslant 0}$, $v+t' \models inv(l)$ for all $0 \leqslant t' \leqslant a$, and $\lambda = \eta_{(l,v+a)}$;
  – *(action transitions)* $a \in Act$, $v \models enab(l,a)$ and for each $(l',v') \in S$:

$$\lambda(l',v') = \sum \{\mid prob(l,a)(X,l') \mid X \in 2^{\mathscr{X}} \wedge v' = v[X:=0] \mid\},$$

- for any $(l,v) \in S$ we have $L(l,v) = L_\mathsf{P}(l) \cup \{\zeta \mid \zeta \in CC(\mathscr{X}) \wedge v \models \zeta\}$.

The set of atomic propositions of $[\![\mathsf{P}]\!]$ is the union of the atomic propositions used for labelling the locations L and the clock constraints obtained from the clocks $\mathscr{X}$. We now return to our running example of a robot and extend our MDP model to exhibit real-time behaviour.

Example 8. Fig. 9 shows a PTA model of our robot moving through terrain that is divided up into a 3×2 grid, extending the MDP model described in Example 3. The PTA has two clocks x and y and each grid section is represented as a location (with initial location l_0). In each location, one or more actions from the set $Act=\{north, east, south, west, stuck\}$ are again available. As before, due to the presence of obstacles, certain directions are unavailable in some states or probabilistically move the robot to an alternative state.

The invariant $x\leqslant4$ in the locations l_0, l_1, l_4 and l_5 and the fact that the clock x is reset on all transitions entering these locations implies that at most 4 time units can be spent in these locations. While the inclusion of the constraint $x\geqslant2$ in all guards of the transitions leaving these locations, implies that the robot must remain in these location for at least 2 time units. In addition, the inclusion of $y\leqslant8$ in the guards on the transitions labelled *north* and *south* and the fact the clock y is never reset, means that the robot can only move 'north' and 'south' during the first 8 time units of operation.   ∎

As for DTMCs and MDPs (see Sect. 2), we can define *reward structures* for PTAs.

Definition 25 (PTA reward structure). For PTA P with locations L and actions Act, a *reward structure* is given by a pair $r=(r_L, r_{Act})$ where:

- $r_L : L \rightarrow \mathbb{R}_{\geqslant 0}$ is a function assigning to each location the rate at which rewards are accumulated as time passes in that location;
- $r_{Act} : L \times Act \rightarrow \mathbb{R}_{\geqslant 0}$ is a function assigning the reward of executing each action in each location.

The location rewards of a PTA assign the rate at which rewards are accumulated as time passes in a state, and therefore the corresponding reward structure of the MDP $[\![P]\!]$ consists of only action rewards. More precisely, in the corresponding reward structure of $[\![P]\!]$ we have $r_A((l,v),t)=r_L(l)\cdot t$ and $r_A((l,v),a)=r_{Act}(l,a)$ for all $(l,v) \in L \times \mathbb{R}_{\geqslant 0}^{\mathscr{X}}$, $t \in \mathbb{R}_{\geqslant 0}$ and $a \in Act$. PTAs equipped with reward structures are a probabilistic extension of linearly-priced timed automata (also known as weighted timed automata) [21, 10]. Parallel composition (see Sect. 4) can also be extended to PTAs [92] under the restriction that the sets of clocks of the component PTAs are disjoint.

An important issue with regard to the analysis of models exhibiting real-time behaviour is that of *time divergence*. More precisely, we do not consider executions in which time does not advance beyond a certain point. These can be ignored on the grounds that they do not correspond to actual, realisable behaviour of the system being modelled [5, 9]. For a PTA P this corresponds to restricting the analysis of the MDP $[\![P]\!]$ to the class of time-divergent (or non-Zeno) strategies (those strategies for which the probability of time passing beyond any bound is 1). This clearly has an impact on the complexity of any analysis. However, there are syntactic conditions, derived from analogous results on timed automata [106, 107], which guarantee that all strategies will be time-divergent, see [92] for further details.

The PRISM logic (see Defn. 3) previously used for specifying properties for DTMCs and MDPs can also be applied to PTAs. There is one key difference since time is now dense as opposed to discrete: the bounds appearing in formulae correspond to bounds on the elapsed time as opposed to bounds on the number of discrete steps. More precisely, path formula $\psi_1 \; U^{\leqslant k} \; \psi_2$ holds if a state satisfying ψ_2 is reached before k time units have elapsed and, up until that point in time, ψ_1 is continuously satisfied, and the reward formulae $I^{=k}$ and $C^{\leqslant k}$ represent the reward at time instant k and the reward accumulated up until k time units have elapsed. As the underlying semantics of a PTA is an MDP the semantics of the logic for PTAs is as given in Defn. 9, modified for bounded properties due to the different interpretation for PTAs given above [92]. The logic can also be extended to allow more general timing properties through *formula clocks* and *freeze quantifiers* [83].

There are a number of different model checking approaches for PTAs which support different classes of properties. Each is based on first constructing a finite state MDP and then analysing this MDP (either computing the optimal probability of a path formula or the expected value of a reward formula). Approaches for model checking PTAs include:

- the region graph construction [83];
- the boundary region graph [69];
- the digital clocks method [81];
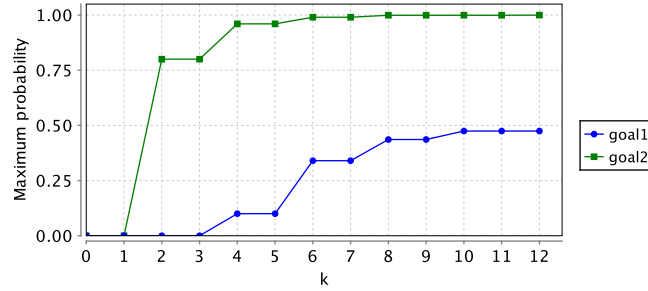- forwards reachability [83];

Fig. 10 Running example: Time-bounded probabilistic reachability for the PTA model of the robot.

- backwards reachability [84];
- abstraction refinement with stochastic games [78].

For a discussion of the advantages and disadvantages of these approaches, see [92].

Example 9. For the PTA model of the robot given in Example 8 and Fig. 9 the maximum probability of reaching a goal_1 labelled state ($\text{P}_{\max=?}[\text{F goal}_1]$) is now 0.4744 as opposed to 0.5 for the MDP model (see Example 3). This is due to the fact that the *north* and *south* actions are only available during the first 8 time units of operation and the robot must remain in the locations l_0, l_1, l_4 and l_5 for between 2 and 4 time units. The minimum expected time to reach a goal_2 state equals 2.5333 and is obtained through the query $\text{R}^r_{\min=?}[\text{F goal}_2]$ where the reward structure $r=(r_L, r_{Act})$ is such that $r(l)=1$ and $r(l,a)=0$ for for all locations l and actions a. Finally, Fig. 10 plots results for the time-bounded maximum reachability properties $\text{P}_{\max=?}[\text{F}^{\leqslant k}\text{ goal}_1]$ and $\text{P}_{\max=?}[\text{F}^{\leqslant k}\text{ goal}_2]$ as the time bound k varies.   ■

Extensions to PTAs. One way of extending PTAs is to allow more general continuous dynamics to model hybrid systems (see Sect. 7). We also mention the introduction of continuously-distributed time delays, see for example [82, 4, 87] and probabilistic timed games (see for example [69, 11]), which can build on the success of (non-probabilistic) timed games for the analysis of synthesis problems [20].

5.1.1 Case Study: Processor Task Scheduling

This PTA case study is taken from [92] and is based on the *task-graph scheduling* problem described in [24] using (non-probabilistic) timed automata. The case study concerns determining optimal schedulers for either the (expected) time or energy consumption required to compute the arithmetic expression $D \times (C \times (A+B)) + ((A+B)+(C \times D))$ using two processors (P_1 and P_2) that have different speed and energy requirements. Fig. 11 presents a task graph for computing this expression and shows both the tasks that need to be performed (the subterms of the expression) and the dependencies between the tasks (the order the tasks must be evaluated in).
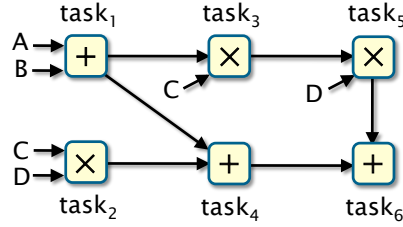
Fig. 11 Processor task scheduling problem: computing $D \times (C \times (A+B)) + ((A+B) + (C \times D))$.

The specification of the processors, as given in [24], is as follows:

- *time for addition:* 2 and 5 picoseconds for processors P_1 and P_2;
- *time for multiplication:* 3 and 7 picoseconds for processors P_1 and P_2;
- *idle energy usage:* 10 and 20 Watts for processors P_1 and P_2;
- *active energy usage:* 90 and 30 Watts for processors P_1 and P_2.

The system is formed as the parallel composition of three PTAs - one for each processor and one for the scheduler. In Fig. 12(a) we give a timed automaton representing P_1. The labels *p1_add* and *p1_mult* on the transitions represent an addition and multiplication task being scheduled on P_1 respectively, while the label *p1_done* indicates that the current task has been completed. The PTA includes a clock x which is used to keep track of the time that a task has been running and is therefore reset when a task starts and the invariants and guards correspond to the time required to complete the tasks of addition and multiplication for P_1. The reward structure for computing the expected energy consumption associates a reward of 10 with the *stdby* location and reward 90 with the locations *add* and *mult* (corresponding to the energy usage of process P_1 when idle and active respectively) and all action rewards are 0. The PTA and reward structure for processor P_2 are similar except for the names of the labels, invariants, guards and reward values correspond to the specification of P_2. After forming the parallel composition, the reward structure for the expected energy consumption then includes the addition of the reward structures for energy consumption of P_1 and P_2. The reward structure for computing the expected time associates a reward of 1 with all locations of the composed system.

In [92] the model of [24] is extended in the following ways.

- A third processor P_3 that has faulty behaviour is added to the sytem. We assume the faulty processor consumes the same energy consumption as P_2, but is faster (addition takes 3 picoseconds and multiplication 5 picoseconds) and has probability p of failing to successfully complete a task. The PTA model of the P_3 is given in Fig. 12(b).
- The processors P_1 and P_2 are changed to have random execution times. We assume that, if the original time to perform a task was t, then the time taken is now uniformly distributed between the delays $t-1$, t and $t+1$. Fig. 12(c) presents the resulting PTA model of P_1.

(a) TA model of processor P_1 (b) Faulty PTA model of processor P_3

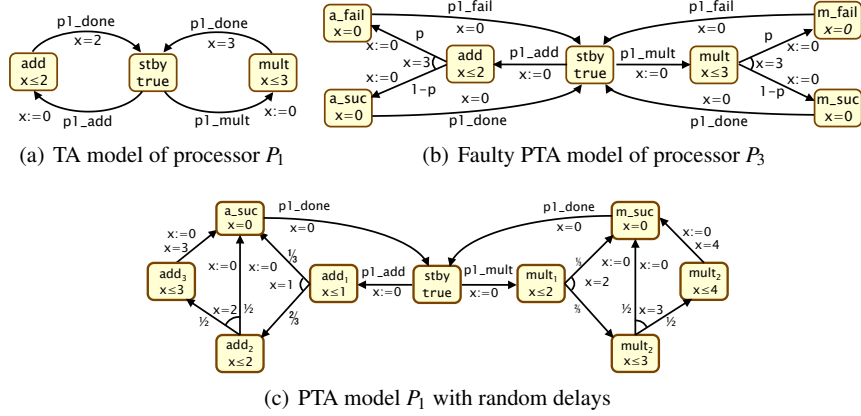(c) PTA model P_1 with random delays

Fig. 12 PTAs for the task-graph scheduling case study

For each model we synthesise the optimal schedulers for both the expected time and energy usage to complete all tasks. To achieve this we used the numerical reward queries $\mathrm{R}^{time}_{\min=?}[\mathrm{F}\ \textsf{complete}]$ and $\mathrm{R}^{energy}_{\min=?}[\mathrm{F}\ \textsf{complete}]$ with the reward structures described above.

Basic model. For the basic (non-probabilistic) model, as proposed in [24], an optimal scheduler for minimising the elapsed time to complete all tasks, takes 12 picoseconds to complete all tasks and schedules the tasks as follows:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_5$			$task_4$		$task_6$									
P_2			$task_2$																	

When considering the energy consumption to complete all tasks, an optimal scheduler makes the following choices:

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_4$														
P_2			$task_2$					$task_5$							$task_6$					

The above scheduler requires 1.3200 nanojoules and 19 picoseconds to complete all tasks. Since processor P_1 consumes additional energy when active, the first scheduler described, optimising the time to complete all tasks, requires 1.3900 nanojoules.

Faulty processor. When adding the faulty processor P_3 we find that for small values of p (the probability of P_3 failing to successfully complete a task), as P_3 has better performance than P_2, both the optimal expected time and energy consumption can be improved using P_3. However, as the probability of failure increases, P_3's better performance is outweighed by the chance of its failure and using it no longer yields optimal values. For example, below, we give an optimal scheduler for minimising the expected time when $p=0.25$ which takes 11.0625 picoseconds (the optimal expected time is 12 picoseconds when P_3 is not used). The dark boxes are used to

denote the cases when P_3 is scheduled to complete a task, but experiences a fault and does not complete the scheduled task correctly.

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_5$			$task_6$											
P_2																				
P_3		$task_2$				$task_4$														

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_2$			$task_4$			$task_6$								
P_2																				
P_3		$task_2$					$task_5$													

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_2$			$task_4$			$task_5$				$task_6$				
P_2																				
P_3		$task_2$					$task_5$													

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1	$task_1$		$task_3$			$task_5$			$task_4$			$task_6$								
P_2																				
P_3		$task_2$				$task_4$														

This optimal scheduler uses the processor P_3 for $task_2$ and, if this task is completed successfully, it then uses P_3 for $task_4$. However, if the processor fails to complete $task_2$, P_3 is instead then used for $task_5$ with $task_4$ being rescheduled on P_1.

Random execution times. For this model the optimal expected time and energy consumption are 12.226 picoseconds and 1.3201 nanojoules respectively. The optimal schedulers change their decision based upon the delays of previously completed tasks. For example, a scheduler that optimises the elapsed time starts by following the choices for the optimal scheduler described for the basic model: first scheduling $task_1$ followed by $task_3$ on P_1 and $task_2$ on P_2. Due to the random execution times it is now possible for $task_2$ to complete before $task_3$ (if the execution times for $task_1$, $task_2$ and $task_3$ are 3, 6 and 4 respectively) and in this case the optimal decision differ from those made for the basic model. To illustrate this we give one possible set of execution times for the tasks and a corresponding optimal scheduling.

time	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
P_1		$task_1$			$task_3$				$task_5$			$task_6$								
P_2			$task_2$					$task_4$												

5.2 Continuous-time Markov Chains

Continuous-time Markov chains (CTMCs) are an alternative way to model systems exhibiting probabilistic and real-time behaviour. This model type is very frequently used in performance analysis and can be considered as a real-time extension of DTMCs. While each transition between states in a DTMC corresponds to a discrete time-step, in a CTMC transitions occur in real time.

Definition 26 (Continuous-time Markov chain). A *continuous-time Markov chain* (CTMC) is a tuple $\mathsf{C}=(S,\bar{s},\mathbf{R},L)$ where:

- S is a *finite* set of states;
- $\bar{s} \in S$ is an initial state;
- $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ is a transition rate matrix;
- $L : S \to 2^{AP}$ is a state labelling function.

For a CTMC $\mathsf{C}=(S,\bar{s},\mathbf{R},L)$ and states $s,s' \in S$, a transition can occur from s to s' if and only if $\mathbf{R}(s,s')>0$ and, when a transition can occur, the time until the transition is triggered is exponentially distributed with parameter $\mathbf{R}(s,s')$, i.e. the probability the transition is triggered within $t \in \mathbb{R}_{\geqslant 0}$ time-units equals $1 - e^{-\mathbf{R}(s,s')\cdot t}$. If more than one transition can occur from a state, then the first transition triggered determines the next state. This is known as a *race condition*.

Using properties of the exponential distribution, we can alternatively consider the behaviour of the CTMC as follows: for any state s the time spent in the state is exponentially distributed with rate $E(s) \overset{\text{def}}{=} \sum_{s' \in S} \mathbf{R}(s,s')$ and the probability that a transition to state s' is then taken equals $\mathbf{R}(s,s')/E(s)$.

As for DTMCs and MDPs, an execution of a CTMC is represented as a path. However, here, we must also consider the time at which a transition is taken. Formally, a path of a CTMC is a (finite or infinite) sequence $\pi = s_0 t_0 s_1 t_1 s_2 t_2 \ldots$ such that $\mathbf{R}(s_i, s_{i+1})>0$ and $t_i \in \mathbb{R}_{>0}$ for all $i \geqslant 0$. Furthermore, let $time(\pi,i)$ denote the time spent in the $(i+1)$th state, that is t_i.

To define a probability measure over infinite paths of a CTMC, we need to extend the cylinder sets used in the probability measure construction for DTMC (see Sect. 2.1) to include time intervals. More precisely, if $s_0,\ldots,s_n$ is a sequence of states such that $\mathbf{R}(s_i,s_{i+1})>0$ for all $0 \leqslant i < n$ and $I_0,\ldots,I_{n-1}$ are non-empty intervals in $\mathbb{R}_{\geqslant 0}$, then the cylinder set $C(s_0,I_0,\ldots,I_{n-1},s_n)$ is the set of infinite paths such that $\pi \in C(s_0,I_0,\ldots,I_{n-1},s_n)$ if and only if $\pi(i) = s_i$ and $time(\pi,i) \in I_j$ for all $0 \leqslant i \leqslant n$ and $0 \leqslant j < n$. We can then construct a probability measure Pr_{C,s_0} over the infinite paths of the CTMC. For further details on this construction see [16].

Reward structures can be defined for a CTMC and, as for PTAs, state rewards assign the rate at which rewards are accumulated as time passes in a state. Also, as for PTAs, when applying the PRISM logic to CTMCs, the bounds appearing in path and reward formulae correspond to the elapsed time as opposed to the number of steps performed. It follows that the only difference between model checking DTMCs and CTMCs concerns the analysis of bounded properties. The standard approach for verifying such time-bounded properties is to use uniformisation [67, 49]. For more details on the model checking algorithms for CTMCs see, for example, [16, 77].

To express nondeterministic behaviour, CTMCs can be extended to *continuous-time Markov decision processes* and related models such as *interactive Markov chains* [62] and *Markov automata* [41]. For such models the main difference from model checking MDPs is again when verifying bounded properties which is considerable more complex in the continuous-time setting where the bounds correspond to elapsed time as opposed to the number of discrete steps. Model checking algorithms for such models have been developed, see for example [27], as well as

temporal logics which allow specification of more expressive timing requirements; see for example [40].

6 Parametric Probabilistic Model Checking

In this section, we consider another extension to the basic technique of probabilistic model checking which provides *parametric* techniques for analysing models. One or more values in definition of the model (for example, a transition probability) or in the property to be verified (for example, a time bound) are provided as a parameter to the verification problem, rather than being instantiated to a specific value. For a numerical query, *parametric model checking* can compute a symbolic expression for the result, as a function of the parameters, rather than a concrete value. For Boolean-valued queries, *parameter synthesis* can be applied to determine the set of all parameter values for which the model is true.

We first consider the parametric model checking of DTMC models and, following this, consider approaches for other probabilistic models.

6.1 Parametric Model Checking for DTMCs

Parametric model checking of DTMCs was first proposed by Daws [36] for the logic PCTL. The basic idea is to represent transition probabilities as rational functions and then use a language-theoretic approach to compute the probability of reaching a set of target states. This is done by treating the transition probabilities as letters of an alphabet, converting the DTMC to a finite automaton over this alphabet and then using the state elimination method to determine a rational function representing the probability of reaching the target.

Since the approach of [36] was first presented, a variety of extensions and implementations have been developed. For example, [54] builds on the basic ideas of Daws, incorporating various optimisations and integrating bisimulation minimisation to improve efficiency. This was implemented in the tool PARAM [52] and later also added to PRISM [79]. Since then, further improvements to parametric model checking of DTMCs have been proposed [65], including the use of strongly connected component decompositions and optimised approaches to the generation of rational functions; these have been implemented in the PROPhESY tool [37].

Below, we explain the key definitions and illustrate the approach on some examples. We refer the reader to the references above for more details.

Definition 27 (Rational function). Let $V = \{x_1, \ldots, x_n\}$ be a set of real-valued variables. A *rational function* f over V is a function of the form $f(x_1, \ldots, x_n) = g_1(x_1, \ldots, x_n)/g_2(x_1, \ldots, x_n)$ where g_1 and g_2 are polynomials each taking the form $\sum_{i=1}^{m} a_i x_1^{k_{i,1}} \cdots x_n^{k_{i,n}}$ for $m \in \mathbb{N}$, $a_i \in \mathbb{R}$ for $1 \leqslant i \leqslant m$ and $k_{i,j} \in \mathbb{N}$ for $1 \leqslant i \leqslant m$ and $1 \leqslant j \leqslant n$. The set of all rational functions over variables V is denoted $\mathscr{F}_V$.

Given a rational function f over the variables V, a subset $V' \subseteq V$ of the variables and an evaluation $u : V' \to \mathbb{R}$ of V', we let $f[V'/u]$ denote the rational function obtained from f by substituting any occurence of a variable $v' \in V'$ with the value $u(v')$. If $V' = V$, then u is a *total evaluation* and $f[V'/u]$ is a rational constant.

Definition 28 (Parametric DTMC). A *parametric DTMC* (PDTMC) is a tuple $D = (S, \bar{s}, \mathbf{P}, L, V)$ where the set of states S, initial state $\bar{s}$ and labelling L are as for a DTMC (see Defn. 1) and:

- $V = \{x_1, \ldots, x_n\}$ is a set of real-valued variables called *parameters*;
- $\mathbf{P} : S \times S \to \mathscr{F}_V$ is a probabilistic transition matrix mapping each pair of states to a rational function over the parameters.

A PDTMC retains the same basic structure as a DTMC, but transition probabilities are expressed as functions of its parameters. Evaluations for this set of parameters (satisfying certain conditions) then induce normal DTMCs.

Definition 29 (Induced DTMC). Let $D = (S, \bar{s}, \mathbf{P}, L, V)$ be a PDTMC and $u : V \to \mathbb{R}$ be a total evaluation of its parameters. Let $\mathbf{P}_u(s, s) : S \times S \to \mathbb{R}$ be the matrix defined by $\mathbf{P}_u(s, s) = \mathbf{P}(s, s)[V/u]$. We say that the evaluation u is *well defined* for D if $\mathbf{P}_u(s, s') \in [0, 1]$ and $\sum_{s' \in S} \mathbf{P}_u(s, s') = 1$ for all $s, s' \in S$. In this case, the *induced DTMC* of the evaluation u is the DTMC $D_u = (S, \bar{s}, \mathbf{P}_u, L)$.

Since the behaviour of a DTMC can be qualitatively different if its underlying transition graph changes, we assume that parameter evaluations u are *graph preserving*, meaning that, $\mathbf{P}(s, s') \neq 0$ implies $\mathbf{P}_u(s, s') > 0$ for all $s, s' \in S$. The basic property of interest for parametric DTMCs can then be defined as follows.

Definition 30 (Probabilistic reachability for PDTMCs). Let $D = (S, \bar{s}, \mathbf{P}, L, V)$ be a PDTMC and $a \in AP$ be an atomic proposition. The probabilistic reachability problem is to find a rational function $f \in \mathscr{F}_V$ such that, for any well-defined and graph preserving evaluation $u : V \to \mathbb{R}$ for D, we have:

$$f[V/u] = Pr_{D_u, \bar{s}} \{ \pi \in IPaths_{D_u}(\bar{s}) \mid D_u, \pi \models \mathsf{F}\, \mathsf{a} \} .$$

Parametric probabilistic model checking of DTMCs has been applied to various problems, including model repair [18] and sensitivity analysis [43]. Below, we illustrate its usage on a simple example.

Example 10. We return to our running example, and adapt the DTMC version of the robot navigation model presented in Example 1 (see Fig. 1). Fig. 13 (left) shows a modified version of this model, to which we have added to parameters p and q which occur in some of the transition probabilities. The original DTMC results from the parameter evaluation u that chooses $u(p) = 0.05$ and $u(q) = 0.75$.

We consider the property $\mathsf{P}_{=?}[\neg \mathsf{goal}_1 \ \mathsf{U} \ \mathsf{goal}_2]$, i.e., the probability of reaching goal 2 before goal 1. Applying parametric probabilistic model checking yields the rational function $(25 \cdot p \cdot q + 40 \cdot p - 10 \cdot q - 24)/(40 \cdot p - 34)$ as a result, which is plotted for the valid ranges of p and q in Fig. 13 (right). ∎
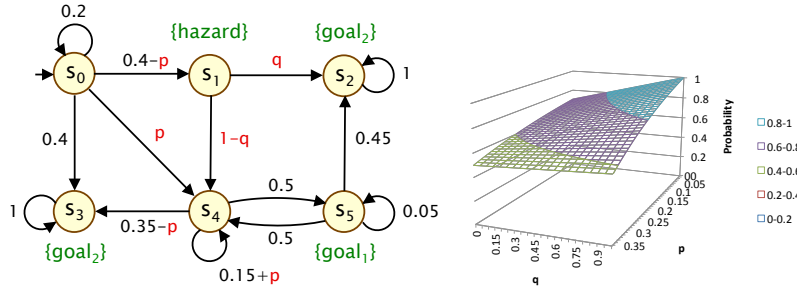
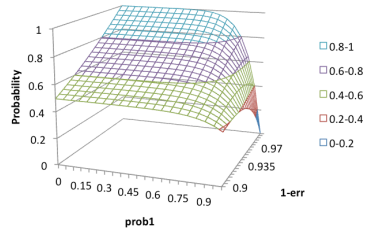**Fig. 13** Parametric model checking applied to an adapted version of the DTMC from Fig. 1.

Fig. 14 Parametric model checking results for the NAND case study.

Example 11. As a second example, we revisit the NAND multiplexing case study described in Sect. 2.1.3. There are two parameters we consider for this case study: *err* representing the probability that a NAND gate is unreliable and *prob1* the probability the initial input is correct (takes the value true). In Fig. 14, for the case when there are five copies of the inputs and outputs (N=5) and one restorative stage (M=1), we have plotted the probability that the error is less than 10 percent (the first property considered in Sect. 2.1.3) as the parameters *err* and *prob1* vary. ∎

6.2 Parametric Model Checking for Other Probabilistic Models

Parametric model checking techniques have also been developed for several of the other probabilistic models described in this chapter. For example, Hahn et al. [54] extend the approach described above to the analysis of MDPs, where the nondeterministic choices are encode as additional (binary) parameters. They found, however, that this method was limited by the number of nondeterministic choices available in a state and the fact that it could not be extended to nested properties.

They have since proposed an alternative method for parametric model checking of MDPs [51]. Instead of finding a rational function corresponding to an optimal probability or expected reward value, this approach finds parameter values for which a given property holds (or does not hold), i.e., it solves the parameter synthesis problem. This is achieved by repeatedly dividing the parameter space into regions

(hyper-rectangles) until regions are found over which the property of interest holds or does not hold. Checking this requirement over a region is performed by first finding an optimal strategy for the 'middle' point of the region, using standard (non-parametric) MDP model checking of MDPs, and then performing parametric model checking on the induced (parametric) DTMC of this strategy over the region.

Techniques also exist for CTMCs. Parametric model checking of unbounded properties for CTMCs can use the same methods as those developed for DTMCs. For time-bounded properties, [59] proposes an approach which approximates the set of parameter values for which a time-bounded probabilistic reachability property holds, based on a discretisation of the parameter space. We also mention [26, 109], which allows for precise parametric model checking of time bounded properties of CTMCs. This works by iteratively dividing the parameter space into regions through the computation of upper and lower bound approximations for the time-bounded reachability probability of interest over the regions.

Finally, concerning PTAs, both [13] and [2] study the problem of synthesising timing constraints of a PTA to ensure the satisfaction of a given property. The approach of [13] is based on the inverse method for parametric (non-probabilistic) timed automata [12], while [2] extends the forwards reachability [83] and game-based [78] approaches for model checking PTAs.

7 Future Challenges and Directions

This chapter has provided an overview of probabilistic model checking and surveyed some of the significant advances that have been made in the area in recent years. Probabilistic model checking has shown itself to be a powerful, flexible and broadly applicable verification technique, but a number of key challenges remain and work continues on many fronts to improve the state of the art.

As with most areas of formal verification, a recurring limitation of probabilistic model checking is its scalability to large, complex systems. We have discussed various efforts to tackle this problem in earlier sections. Another related and fundamental issue, which is true of any model-based analysis technique, is that the results of verification are only as reliable as the model itself. For models with quantitative aspects such as probability and time, which may be difficult to measure accurately, this is particularly pertinent.

We conclude this chapter by highlighting some of the key challenges and research directions in the area of probabilistic model checking, many of which aim to tackle these issues.

Hybrid systems. Probabilistic model checking has many applications in the domain of embedded and cyber-physical systems, for example in the verification of sensor networks or robotic applications. In this setting, the interaction of (discrete) computerised systems with their (continuous) environment becomes a crucial issue.

Such hybrid systems (or cyber-physical systems) raise new challenges because they require more powerful models such as stochastic hybrid automata.

Hybrid automata allow both discrete behaviour and continuous flows defined through differential equations, for example to model thermodynamics. The verification of hybrid automata is in general undecidable, therefore the analysis is restricted to certain subclasses and considering only approximate results. Early work on probabilistic hybrid automata concerned decidability results for different subclasses [102]. Recent work [113, 56] combines abstraction approaches for non-probabilistic hybrid automata [6, 98] with the abstraction-refinement approaches for MDPs [35, 74] discussed in Sect. 4. We also mention [38], where two approximation techniques for classical hybrid automata are extended to the probabilistic case and [47] which, using stochastic satisfiability modulo theories, presents a decision procedure for verifying time-bounded properties.

Probabilistic software and programs. Although the modelling languages of tools such as PRISM are sufficiently expressive for many purposes, direct support for the probabilistic model checking of mainstream programming languages such as C or Java or of system-level modelling languages such as SystemC will be required for the verification of real applications. Programs in these languages yield extremely large, or infinite state, models, which need dedicated techniques to tackle. A related area, which has attracted interest in recent years, is the verification of probabilistic programming languages [70], which have applications both for the specification of randomised or probabilistic software and for the development of probabilistic models used for inference and machine learning.

Ubiquitous computing. The vision of ubiquitous or pervasive computing sees thousands of computerised devices integrating seamlessly in daily life. This emphasises the need for techniques to ensure their correctness, but also demands the development of new modelling formalisms and analysis techniques that can handle both the dynamic nature and the enormous scale of these systems. One key aspect to modelling ubiquitous computing devices is *autonomous behaviour*, as can be seen in for example driverless cars and drone missions. In addition, we need to model the *constrained resources* (often devices have limited memory and CPU processing and are battery powered) and the fact that devices need to be *adaptive* as requirements and the environment evolve.

Partial observability. In this chapter we have assumed that the state of the system and history are fully visible to a strategy when making decisions. However, in many situations, this is unrealistic, for example to verify that a security protocol is functioning correctly, it may be essential to model the fact that some data held by a participant is not externally visible, or, when synthesising a controller for a robot, the controller may not be implementable in practice if it bases its decisions on information that cannot be physically observed.

Partially observable MDPs (POMDPs) are a natural extension of MDPs for modelling such strategies and they are widely used in areas such as planning and artificial intelligence, but verification of POMDPs is considerably more difficult than

MDPs since key problems are undecidable [86]. Work in this area towards practical verification of POMDPs includes [28, 105, 93].

Robustness and uncertainty. In many potential applications, such as the generation of controllers in embedded systems, it may be difficult to formulate a precise model of the stochastic behaviour of the system's environment. Thus, developing appropriate models of uncertainty, and corresponding methods to synthesise strategies that are robust in these environments, is important. Developing more sophisticated approaches is an active area of research [112, 96].

Counterexamples. One final challenge is to improve the quality and usefulness of the results that are generated by probabilistic model checking. One of the main reasons for the success of non-probabilistic modeling checking is the generation of counterexamples which provide, when the property being verified does not hold, evidence of this violation. This evidence is usually in the form of a path demonstrating the violation. In the probabilistic case, there is the complication that, to refute a property, a single path is in general not sufficient as more than one path can contribute to the probability of the property not holding. Initial research [58], focused on DTMCs and reachability properties and generating a finite set of paths. Recent research has focused on generating a more useful representation for counterexamples, including regular expressions, hierarchical representations and critical sub-systems, for further information see, for example, the survey [1].

References

1. Ábrahám, E., Becker, B., Dehnert, C., Jansen, N., Katoen, J.P., Wimmer, R.: Counterexample generation for discrete-time Markov models: An introductory survey. In: M. Bernardo, F. Damiani, R. Haehnle, E. Johnsen, I. Schaefer (eds.) Formal Methods for the Design of Computer, Communication, and Software Systems (SFM'14), *LNCS*, vol. 8483, pp. 65–121. Springer (2014)
2. A.Jovanovic, M.Kwiatkowska: Parameter synthesis for probabilistic timed automata using stochastic games. In: J. Ouaknine, I. Potapov, J. Worrell (eds.) Proc. 8th Int. Workshop Reachability Problems (RP'14), *LNCS*, vol. 8762, pp. 176–189. Springer (2014)
3. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University (1997)
4. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for probabilistic real-time systems. In: Proc. 19th Int. Colloq. Automata, Languages and Programming (ICALP'91), *LNCS*, vol. 510, pp. 115–136. Springer (1991)
5. Alur, R., Courcoubetis, C., Dill, D.: Model checking in dense real time. Information and Computation **104**(1), 2–34 (1993)
6. Alur, R., Dang, T., Ivancic, F.: Predicate abstraction for reachability analysis of hybrid systems. ACM Transactions on Embedded Computing Systems **5**(1), 152–199 (2006)
7. Alur, R., Dill, D.: A theory of timed automata. Theoretical Computer Science **126**, 183–235 (1994)

8. Alur, R., Henzinger, T., Kupferman, O.: Alternating-time temporal logic. Journal of the ACM **49**(5), 672–713 (2002)

9. Alur, R., Henzinger, T., Rajamani, S.: Symbolic exploration of transition hierarchies. In: Proc. 4th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'98), *LNCS*, vol. 1384, pp. 330–344. Springer (1998)

10. Alur, R., La Torre, S., Pappas, G.: Optimal paths in weighted timed automata. Theoretical Computer Science **318**(3), 297–322 (2004)

11. Alur, R., Trivedi, A.: Relating average and discounted costs for quantitative analysis of timed systems. In: Proc. 11th Int. Conf. Embedded Software (EMSOFT'11), pp. 165–174. ACM (2011)

12. André, E., Chatain, T., Encrenaz, E., Fribourg, L.: An inverse method for parametric timed automata. International Journal of Foundations of Computing Science **20**(5), 819–836 (2009)

13. André, E., Fribourg, L., Sproston, J.: An extension of the inverse method to probabilistic timed automata. Formal Methods in System Design **42**(2), 119–145 (2013)

14. Baier, C., Clarke, E., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: P. Degano, R. Gorrieri, A. Marchetti-Spaccamela (eds.) Proc. 24th Int. Colloq. Automata, Languages and Programming (ICALP'97), *LNCS*, vol. 1256, pp. 430–440. Springer (1997)

15. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: J.J. Lévy, E. Mayr, J. Mitchell (eds.) Proc. 3rd IFIP Int. Conf. Theoretical Computer Science (TCS'06), pp. 493–5062. Kluwer (2004)

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

17. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)

18. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C., Smolka, S.: Model repair for probabilistic systems. In: P. Abdulla, K. Leino (eds.) Proc. 17th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11), *LNCS*, vol. 6605, pp. 326–340. Springer (2011)

19. Beauquier, D.: Probabilistic timed automata. Theoretical Computer Science **292**(1), 65–84 (2003)

20. Behrmann, G., Cougnard, A., David, A., Fleury, E., Larsen, K., Lime, D.: UPPAAL-Tiga: Time for playing games! In: Proc. 19th Int. Conf. Computer Aided Verification (CAV'07), *LNCS*, vol. 4590, pp. 121–125. Springer (2007)

21. Behrmann, G., Fehnker, A., Hune, T., Larsen, K., Pettersson, P., Romijn, J.: Efficient guiding towards cost-optimality in UPPAAL. In: T. Margaria, W. Yi (eds.) Proc. 7th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'01), *LNCS*, vol. 2031, pp. 174–188. Springer (2001)

22. Bellman, R.: Dynamic Programming. Princeton University Press (1957)

23. Billingsley, P.: Probability and Measure. Wiley (1995)

24. Bouyer, P., Fahrenberg, U., Larsen, K., Markey, N.: Quantitative analysis of real-time systems using priced timed automata. Communications of the ACM **54**(9), 78–87 (2011)

25. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Stochastic games with branching-time winning objectives. In: Proc. 21th IEEE Symp. Logic in Computer Science (LICS'06), pp. 349–358. IEEE Computer Society (2006)

26. Brim, L., Češka, M., S. Dražan, D.v.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: Proc. 25th Int. Conf. Computer Aided Verification (CAV'13), *LNCS*, vol. 8044, pp. 107–123. Springer (2013)

27. Buchholz, P., Hahn, E.M., Hermanns, H., Zhang, L.: Model checking algorithms for CT-MDPs. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd Int. Conf. Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 225–242. Springer (2011)

28. Chatterjee, K., Chmelík, M., Gupta, R., Kanodia, A.: Qualitative analysis of POMDPs with temporal logic specifications for robotics applications. In: Proc. IEEE Int. Conf. Robotics and Automation, (ICRA'15), pp. 325–330. IEEE Computer Society (2015)

29. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Formal Methods in System Design **43**(1), 61–92 (2013)
30. Cheshire, S., Adoba, B., Gutterman, E.: Dynamic configuration of IPv4 link local addresses. Available from www.ietf.org/rfc/rfc3927.txt
31. Clarke, E., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: A. Emerson, A. Sistla (eds.) Proc. 12th Int. Conf. Computer Aided Verification (CAV'00), *LNCS*, vol. 1855, pp. 154–169. Springer (2000)
32. Condon, A.: The complexity of stochastic games. Information and Computation **96**(2), 203–224 (1992)
33. Condon, A.: On algorithms for simple stochastic games. Advances in computational complexity theory, DIMACS Series in Discrete Mathematics and Theoretical Computer Science **13**, 51–73 (1993)
34. Daniele, M., Giunchiglia, F., Vardi, M.: Improved automata generation for linear temporal logic. In: N. Halbwachs, D. Peled (eds.) Proc. 11th Int. Conf. Computer Aided Verification (CAV'99), *LNCS*, vol. 1633, pp. 249–260. Springer (1999)
35. D'Argenio, P., Jeannet, B., Jensen, H., Larsen, K.: Reachability analysis of probabilistic systems by successive refinements. In: L. de Alfaro, S. Gilmore (eds.) Proc. 1st Joint Int. Workshop Process Algebra and Probabilistic Methods, Performance Modelling and Verification (PAPM/PROBMIV'01), *LNCS*, vol. 2165, pp. 39–56. Springer (2001)
36. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Z. Liu, K. Araki (eds.) Proc. 1st Int Coll. Theoretical Aspects of Computing (ICTAC'04), *LNCS*, vol. 3407, pp. 280–294. Springer (2004)
37. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.P., Ábrahám, E.: PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In: Proc. 27th Int. Conf. Computer Aided Verification (CAV'15), *LNCS*, vol. 9206, pp. 214–231. Springer (2015)
38. Desharnais, J., Assouramou, J.: Analysis of non-linear probabilistic hybrid systems. In: Proc. 9th Workshop Quantitative Aspects of Programming Languages (QAPL'11), *EPTCS*, vol. 57, pp. 104–119 (2011)
39. Donaldson, A., Miller, A.: Symmetry reduction for probabilistic model checking using generic representatives. In: S. Graf, W. Zhang (eds.) Proc. 4th Int. Symp. Automated Technology for Verification and Analysis (ATVA'06), *LNCS*, vol. 4218, pp. 9–23. Springer (2006)
40. Donatelli, S., Haddad, S., Sproston, J.: Model checking timed and stochastic properties with CSL^{TA}. IEEE Transactions on Software Engineering **35**(2), 224–240 (2008)
41. Eisentraut, C., Hermanns, H., Zhang, L.: On probabilistic automata in continuous time. In: Proc. 25th Annual IEEE Symp. Logic in Computer Science (LICS'10), pp. 342–351. IEEE Computer Society (2010)
42. Etessami, K., Kwiatkowska, M., Vardi, M., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Logical Methods in Computer Science **4**(4), 1–21 (2008)
43. Filieri, A., Tamburrelli, G., Ghezzi, C.: Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Transactions on Software Engineering **42**(1), 75–99 (2016)
44. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: M. Bernardo, V. Issarny (eds.) Formal Methods for Eternal Networked Software Systems (SFM'11), *LNCS*, vol. 6659, pp. 53–113. Springer (2011)
45. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: P. Abdulla, K. Leino (eds.) Proc. 17th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11), *LNCS*, vol. 6605, pp. 112–127. Springer (2011)
46. Forejt, V., Kwiatkowska, M., Parker, D.: Pareto curves for probabilistic model checking. In: S. Chakraborty, M. Mukund (eds.) Proc. 10th Int. Symp. Automated Technology for Verification and Analysis (ATVA'12), *LNCS*, vol. 7561, pp. 317–332. Springer (2012)
47. Fränzle, M., Teige, T., Eggers, A.: Engineering constraint solvers for automatic analysis of probabilistic hybrid automata. Journal of Logic and Algebraic Programming **79**(7), 436–466 (2010)

48. Graf, S., Saidi, H.: Construction of abstract state graphs with PVS. In: O. Grumberg (ed.) Proc. 9th Int. Conf. Computer Aided Verification (CAV'97), *LNCS*, vol. 1254, pp. 72–83. Springer-Verlag (1997)

49. Gross, D., Miller, D.: The randomization technique as a modeling tool and solution procedure for transient Markov processes. Operations Research **32**(2), 343–361 (1984)

50. Größer, M., Baier, C.: Partial order reduction for Markov decision processes: A survey. In: F. de Boer, M. Bonsangue, S. Graf, W.P. de Roever (eds.) Proc. 4th Int. Symp. Formal Methods for Component and Objects (FMCO '05), *LNCS*, vol. 4111, pp. 408–427. Springer (2006)

51. Hahn, E.M., Han, T., Zhang, L.: Synthesis for PCTL in parametric Markov decision processes. In: Proc. 3rd NASA Formal Methods Symp. (NFM'11), *LNCS*, vol. 6617. Springer (2011)

52. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: A model checker for parametric Markov models. In: Proc. 22nd Int. Conf. Computer Aided Verification (CAV'10), *LNCS*, vol. 6174, pp. 660–664. Springer (2010)

53. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: Abstraction refinement for infinite probabilistic models. In: J. Esparza, R. Majumdar (eds.) Proc. 16th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10), *LNCS*, vol. 6105, pp. 353–357. Springer (2010)

54. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. International Journal on Software Tools for Technology Transfer (STTT) **13**(1), 3–19 (2011)

55. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: Proc. 19th International Symposium on Formal Methods (FM'14), pp. 312–317 (2014)

56. Hahn, E.M., Norman, G., Parker, D., Wachter, B., Zhang, L.: Game-based abstraction and controller synthesis for probabilistic hybrid systems. In: Proc. 8th Int. Conf. Quantitative Evaluation of SysTems (QEST'11), pp. 69–78. IEEE Computer Society Press (2011)

57. Han, J., Jonker, P.: A system architecture solution for unreliable nanoelectronic devices. IEEE Transactions on Nanotechnology **1**, 201–208 (2002)

58. Han, T., Katoen, J.P., Damman, B.: Counterexample generation in probabilistic model checking. IEEE Transactions on Software Engineering **35**(2), 241–257 (209)

59. Han, T., Katoen, J.P., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: Proc. IEEE Real-Time Systems Symp. (RTSS 08), pp. 173–182. IEEE Computer Society Press (2008)

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

61. Hartmanns, A., Hermanns, H.: A modest approach to checking probabilistic timed automata. In: Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09) (2009). To appear

62. Hermanns, H.: Interactive Markov Chains and the Quest for Quantified Quality, *LNCS*, vol. 2428. Springer Verlag (2002)

63. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: A. Gupta, S. Malik (eds.) Proc. 20th Int. Conf. Computer Aided Verification (CAV'08), *LNCS*, vol. 5123, pp. 162–175. Springer (2008)

64. Howard, R.: Dynamic Programming and Markov Processes. The MIT Press (1960)

65. Jansen, N., Corzilius, F., Volk, M., Wimmer, R., brahm, E., Katoen, J.P., Becker, B.: Accelerating parametric probabilistic verification. In: Proc. 11th Int. Conf. Quantitative Evaluation of Systems (QEST'14), pp. 404–420 (2014)

66. Jeannet, B., D'Argenio, P., Larsen, K.: Rapture: A tool for verifying Markov decision processes. In: I. Cerna (ed.) Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02), Technical Report FIMU-RS-2002-05, Faculty of Informatics, Masaryk University, pp. 84–98 (2002)

67. Jensen, A.: Markoff chains as an aid in the study of Markoff processes. Skandinavisk Aktuarietidskrift **36**, 87–91 (1953)

68. Jensen, H.: Model checking probabilistic real time systems. In: Proc. 7th Nordic Workshop Programming Theory, pp. 247–261 (1996)

69. Jurdziński, M., Kwiatkowska, M., Norman, G., Trivedi, A.: Concavely-priced probabilistic timed automata. In: M. Bravetti, G. Zavattaro (eds.) Proc. 20th Int. Conf. Concurrency Theory (CONCUR'09), *LNCS*, vol. 5710, pp. 415–430. Springer (2009)

70. Katoen, J.P.: Probabilistic programming: A true challenge in verification. In: Proc. 13th International Symposium on Automated Technology for Verification and Analysis (ATVA'15), LNCS, pp. 1–3. Springer (2015)

71. Katoen, J.P., Kemna, T., Zapreev, I., Jansen, D.: Bisimulation minimisation mostly speeds up probabilistic model checking. In: O. Grumberg, M. Huth (eds.) Proc. 13th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07), *LNCS*, vol. 4424, pp. 87–101. Springer (2007)

72. Katoen, J.P., Zapreev, I., Hahn, E.M., Hermanns, H., Jansen, D.: The ins and outs of the probabilistic model checker MRMC. In: Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09), pp. 167–176. IEEE Computer Society Press (2009)

73. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: N. Jones, M. Muller-Olm (eds.) Proc. 10th Int. Conf. Verification, Model Checking, and Abstract Interpretation (VMCAI'09), *LNCS*, vol. 5403, pp. 182–197. Springer (2009)

74. 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)

75. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains, 2nd edn. Springer-Verlag (1976)

76. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: T. Ball, R. Jones (eds.) Proc. 18th Int. Conf. Computer Aided Verification (CAV'06), *LNCS*, vol. 4114, pp. 234–248. Springer (2006)

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

78. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: J. Ouaknine, F. Vaandrager (eds.) Proc. 7th Int. Conf. Formal Modelling and Analysis of Timed Systems (FORMATS'09), *LNCS*, vol. 5813, pp. 212–227. Springer (2009)

79. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: Verification of probabilistic real-time systems. In: G. Gopalakrishnan, S. Qadeer (eds.) Proc. 23rd Int. Conf. Computer Aided Verification (CAV'11), *LNCS*, vol. 6806, pp. 585–591. Springer (2011)

80. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Compositional probabilistic verification through multi-objective model checking. Information and Computation **232**, 38–65 (2013)

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

82. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Verifying quantitative properties of continuous probabilistic timed automata. In: C. Palamidessi (ed.) In Proc. 11th Int. Conf. Concurrency Theory (CONCUR'00), *LNCS*, vol. 1877, pp. 123–137. Springer (2000)

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

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

85. Kwiatkowska, M., Parker, D., Wiltsche, C.: PRISM-games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In: Proc. 22nd Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'16), LNCS. Springer (2016)

86. Madani, O., Hanks, S., Condon, A.: On the undecidability of probabilistic planning and related stochastic optimization problems. Artificial Intelligence **147**(1–2), 5–34 (2003)

87. Maler, O., Larsen, K., Krogh, B.: On zone-based analysis of duration probabilistic automata. In: Proc. 12th Int. Workshop Verification of Infinite-State Systems (INFINITY'10), *EPTCS*, vol. 39, pp. 33–46 (2010)

88. Milner, R.: Calculi for synchrony and asynchrony. Theor. Comput. Sci. **25**(3), 267–310 (1993)

89. von Neumann, J.: Probabilistic logics and synthesis of reliable organisms from unreliable components. In: C. Shannon, J. McCarthy (eds.) Automata Studies, pp. 43–98. Princeton University Press (1956)

90. Nicola, R., Vaandrager, F.: Action versus state based logics for transition systems. In: I. Guessarian (ed.) Proc. LITP Spring School on Theoretical Computer Science: Semantics of Systems of Concurrent Processes, pp. 407–419. Springer (1990)

91. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S.: Evaluating the reliability of NAND multiplexing with PRISM. IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems **24**(10), 1629–1637 (2005)

92. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Formal Methods in System Design **43**(2), 164–190 (2013)

93. Norman, G., Parker, D., Zou, X.: Verification and control of partially observable probabilistic real-time systems. In: S. Sankaranarayanan, E. Vicario (eds.) Proc. 13th Int. Conf. Formal Modelling and Analysis of Timed Systems (FORMATS'15), *LNCS*, vol. 9268, pp. 240–255. Springer (2015)

94. Parker, D.: Implementation of symbolic model checking for probabilistic systems. Ph.D. thesis, University of Birmingham (2002)

95. Pnueli, A.: The temporal semantics of concurrent programs. Theoretical Computer Science **13**, 45–60 (1981)

96. Puggelli, A., Li, W., Sangiovanni-Vincentelli, A., Seshia, S.: Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In: Proc. 25th Int. Conf. Computer Aided Verification (CAV'13), *LNCS*, vol. 8044, pp. 527–542. Springer (2013)

97. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons (1994)

98. Ratschan, S., She, Z.: Safety verification of hybrid systems by constraint propagation-based abstraction refinement. ACM Transactions on Embedded Computing Systems **6**(1) (2007)

99. Roscoe, A.W.: The theory and practice of concurrency. Prentice-Hall (1997)

100. Segala, R.: Modelling and verification of randomized distributed real time systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)

101. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing **2**(2), 250–273 (1995)

102. Sproston, J.: Decidable model checking of probabilistic hybrid automata. In: M. Joseph (ed.) Proc. Int. Symp. on Formal Techniques in Real-Time and Fault Tolerant Systems (FTRTFT'00), *LNCS*, vol. 1926, pp. 31–45. Springer (2000)

103. Sun, J., Liu, Y., Dong, J.S., Pang, J.: Pat: Towards flexible verification under fairness. In: Proc. 21st Int. Conf. Computer Aided Verification (CAV'09), *LNCS*, vol. 5643, pp. 709–714. Springer (2009)

104. Svorenova, M., Kwiatkowska, M.: Quantitative verification and strategy synthesis for stochastic games. European Journal of Control **30**, 1530 (2016)

105. Svoreňová, M., Chmelík, M., Leahy, K., Eniser, H., Chatterjee, K., Černá, I., Belta, C.: Temporal logic motion planning using POMDPs with parity objectives: Case study paper. In: Proc. 18th Int. Conf. Hybrid Systems: Computation and Control (HSCC'15), pp. 233–238. ACM (2015)

106. Tripakis, S.: The analysis of timed systems in practice. Ph.D. thesis, Université Joseph Fourier, Grenoble (1998)

107. Tripakis, S., Yovine, S., Bouajjan, A.: Checking timed Buchi automata emptiness efficiently. Formal Methods in System Design **26**(3), 267–292 (2005)

108. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation **115**(1), 1–37 (1994)

109. Češka, M., Dannenberg, F., Kwiatkowska, M., Paoletti, N.: Precise parameter synthesis for stochastic biochemical systems. In: P. Mendes, J. Dada, K. Smallbone (eds.) Proc. 12th Int. Conf. Computational Methods in Systems Biology (CMSB'14), *LNCS/LNBI*, vol. 8859, pp. 86–98. Springer (2014)
110. Wachter, B., Zhang, L., Hermanns, H.: Probabilistic model checking modulo theories. In: Proc. 4th Int. Conf. Quantitative Evaluation of Systems (QEST'07), pp. 129–140. IEEE Computer Society Press (2007)
111. Wiltsche, C.: Assume-guarantee strategy synthesis for stochastic games. Phd thesis, University of Oxford (2015)
112. Wolff, E., Topcu, U., Murray, R.: Robust control of uncertain Markov decision processes with temporal logic specifications. In: Proc. IEEE 51st Annual Conf. Decision and Control (CDC'12), pp. 3372–3379. Computer Society Press (2012)
113. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.M.: Safety verification for probabilistic hybrid systems. European Journal of Control **18**(6), 572–587 (2012)
114. http://www.prismmodelchecker.org
115. http://www.prismmodelchecker.org/files/fsv-pmc/
116. http://www.prismmodelchecker.org/games
117. http://www.prismmodelchecker.org/other-tools.php