Automated Verification Techniques for Probabilistic Systems

Vojtech Forejt, Marta Kwiatkowska, Gethin Norman, David Parker

Automated Verification Techniques for Probabilistic Systems

Vojtěch Forejt[1], Marta Kwiatkowska[1], Gethin Norman[2], and David Parker[1]

[1] Department of Computer Science, University of Oxford, Oxford, OX1 3QD, UK
[2] School of Computing Science, University of Glasgow, Glasgow, G12 8RZ, UK

Abstract. This tutorial provides an introduction to probabilistic model checking, a technique for automatically verifying quantitative properties of probabilistic systems. We focus on Markov decision processes (MDPs), which model both stochastic and nondeterministic behaviour. We describe methods to analyse a wide range of their properties, including specifications in the temporal logics PCTL and LTL, probabilistic safety properties and cost- or reward-based measures. We also discuss multi-objective probabilistic model checking, used to analyse trade-offs between several different quantitative properties. Applications of the techniques in this tutorial include performance and dependability analysis of networked systems, communication protocols and randomised distributed algorithms. Since such systems often comprise several components operating in parallel, we also cover techniques for compositional modelling and verification of multi-component probabilistic systems. Finally, we describe three large case studies which illustrate practical applications of the various methods discussed in the tutorial.

1 Introduction

Many computerised systems exhibit probabilistic behaviour. Messages transmitted across wireless networks, for example, may be susceptible to losses and delays, or system components may be prone to failure. In both cases, probability is a valuable tool for the modelling and analysis of such systems. Another source of stochastic behaviour is the use of randomisation, for example to break symmetry or prevent flooding in communication networks. This is an integral part of wireless communication protocols such as Bluetooth or Zigbee. Randomisation is also a useful tool in security protocols, for example to guarantee anonymity, and in the construction of dynamic power management schemes.

Formal verification is a systematic approach that applies mathematical reasoning to obtain guarantees about the correctness of a system. One successful method in this domain is *model checking*. This is based on the construction and analysis of a system model, usually in the form of a finite state automaton, in which states represent the possible configurations of the system and transitions between states capture the ways that the system can evolve over time. Desired properties such as "no two threads obtain a lock simultaneously" or "the system always eventually delivers an acknowledgement to a request" are then expressed in temporal logic and the model is analysed in an automatic fashion to determine whether or not the model satisfies the properties.

There is increasing interest in the development of *quantitative* verification techniques, which take into account probabilistic and timed aspects of a system. *Probabilistic model checking*, for example, is a generalisation of model checking that builds and analyses probabilistic models such as Markov chains and Markov decision processes. A wide range of quantitative properties of these systems can be expressed in probabilistic extensions of temporal logic and systematically analysed against the constructed model. These properties can capture many different aspects of system behaviour, from reliability, e.g. "the probability of an airbag failing to deploy on demand", to performance, e.g. "the expected time for a network protocol to successfully send a message packet".

This tutorial gives an introduction to probabilistic model checking for *Markov decision processes*, a commonly used formalism for modelling systems that exhibit a combination of probabilistic and nondeterministic behaviour. It covers the underlying theory, discusses probabilistic model checking algorithms and their implementation, and gives an illustration of the application of these techniques to some large case studies. The tutorial is intended to complement [67], which focuses on probabilistic model checking for discrete- and continuous-time Markov chains, rather than Markov decision processes. There is also an accompanying website [91], providing models for the PRISM probabilistic model checker [56] that correspond to the various running examples used throughout and to the case studies in Section 10.

There are many other good references relating to the material covered in this tutorial and we provide pointers throughout. In particular, Chapter 10 of [11] covers some of the MDP model checking techniques presented here, but in greater depth, with additional focus on the underlying theory and proofs. We also recommend the theses by Segala [80], de Alfaro [1] and Baier [7], which provide a wealth of in-depth material on a variety of topics relating to MDPs, along with detailed pointers to other relevant literature. Finally, although not focusing on verification, [76] is an excellent general reference on MDPs.

Outline. The tutorial is structured as follows. We begin, in Section 2, with background material on probability theory and discrete-time Markov chains. In Section 3, we introduce the model of Markov decision processes. Section 4 describes the computation of a key property of MDPs, probabilistic reachability, and Section 5 covers reward-based properties. Section 6 concerns how to formally specify properties of MDPs using the probabilistic temporal logic PCTL, and shows how the techniques introduced in the previous sections can be used to perform model checking. Section 7 describes how to verify MDPs against safety properties and the logic LTL using automata-based techniques. Two advanced topics, namely multi-objective probabilistic model checking and compositional probabilistic verification, are the focus of Sections 8 and 9. In Section 10, we list some of the software tools available for model checking MDPs and describe three illustrative case studies. Section 11 concludes by discussing active research areas and suggesting further reading.

2 Background Material

2.1 Probability Distributions & Measures

We begin by briefly summarising some definitions and notations relating to probability distributions and measures. We assume that the reader has some familiarity with basic probability theory. Good introductory texts include [19,42].

Definition 1 (Probability distribution). *A (discrete)* probability distribution *over a countable set S is a function $\mu : S \to [0,1]$ satisfying $\sum_{s \in S} \mu(s){=}1$.*

We use $[s_0{\mapsto}x_0, \ldots, s_n{\mapsto}x_n]$ to denote the distribution that chooses s_i with probability x_i for all $0{\leqslant}i{\leqslant}n$ and $\mathit{Dist}(S)$ for the set of distributions over S. The *point distribution* on $s \in S$, denoted $[s{\mapsto}1]$, is the distribution that assigns probability 1 to s. Given two distributions $\mu_1 \in \mathit{Dist}(S_1)$ and $\mu_2 \in \mathit{Dist}(S_2)$, the *product distribution* $\mu_1{\times}\mu_2 \in \mathit{Dist}(S_1{\times}S_2)$ is defined by $\mu_1{\times}\mu_2((s_1,s_2)) = \mu_1(s_1){\cdot}\mu_2(s_2)$.

Definition 2 (Probability space). *A* probability space *over a sample space Ω is a triple $(\Omega, \mathcal{F}, Pr)$, where $\mathcal{F} \subseteq 2^{\Omega}$ is a* σ-algebra *over Ω, i.e.*

- *$\varnothing, \Omega \in \mathcal{F}$;*
- *if $A \in \mathcal{F}$, then $\Omega \setminus A \in \mathcal{F}$;*
- *if $A_i \in \mathcal{F}$ for all $i \in \mathbb{N}$, then $\cup_{i \in \mathbb{N}} A_i \in \mathcal{F}$*

and $Pr : \mathcal{F} \to [0,1]$ is a probability measure *over $(\Omega, \mathcal{F})$, i.e.*

- *$Pr(\varnothing) = 0$ and $Pr(\Omega) = 1$;*
- *$Pr(\cup_{i \in \mathbb{N}} A_i) = \sum_{i \in \mathbb{N}} Pr(A_i)$ for all countable pairwise disjoint sequences $A_1, A_2, \ldots$ of $\mathcal{F}$.*

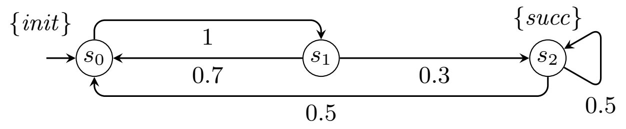
Sets contained in the σ-algebra $\mathcal{F}$ are said to be *measurable*. A (non-negative) *random variable* over a probability space $(\Omega, \mathcal{F}, Pr)$ is a *measurable function* $X : \Omega \to \mathbb{R}_{\geqslant 0}$, i.e. a function such that $X^{-1}([0,r]) \in \mathcal{F}$ for all $r \in \mathbb{R}_{\geqslant 0}$. The *expected value* of X with respect to Pr is given by the following integral:

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

2.2 Discrete-time Markov Chains

Next, we introduce the model of *discrete-time Markov chains* (DTMCs). We provide just a brief overview of DTMCs, as required for the remainder of this tutorial. For more in-depth coverage of the topic, we recommend the textbooks by Stewart [83] and Kulkarni [65]. For a tutorial on probabilistic model checking for Markov chains, see for example [67].

Definition 3 (Discrete-time Markov chain). *A* discrete-time Markov chain *(DTMC) is a tuple $\mathcal{D} = (S, \overline{s}, \mathbf{P}, L)$ where S is a (countable) set of states, $\overline{s} \in S$ is an initial state, $\mathbf{P} : S{\times}S \to [0,1]$ is a transition probability matrix such that $\sum_{s' \in S} \mathbf{P}(s,s') = 1$ for all $s \in S$, and $L : S \to 2^{AP}$ is a labelling function mapping each state to a set of atomic propositions taken from a set AP.*



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

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

One way to view a DTMC $\mathcal{D}{=}(S, \overline{s}, \mathbf{P}, L)$ is as a state-transition system in which transitions are augmented with probabilities indicating their likelihood. From each state $s \in S$, the probability of a transition to s' occurring is $\mathbf{P}(s, s')$.

A *path* represents one possible execution of $\mathcal{D}$. Formally, a (finite or infinite) path of $\mathcal{D}$ is a sequence of states $s_0 s_1 s_2 \ldots$ such that $\mathbf{P}(s_i, s_{i+1}){>}0$ for all $i{\geqslant}0$. We use $\mathit{FPath}_{\mathcal{D},s}$ and $\mathit{IPath}_{\mathcal{D},s}$, respectively, to denote the set of all finite and infinite paths starting from state s of $\mathcal{D}$.

In order to reason formally about the behaviour of $\mathcal{D}$, we need to determine the probability that certain paths are taken. We proceed by constructing, for each state $s \in S$, a *probability space* over the set of infinite paths $\mathit{IPath}_{\mathcal{D},s}$. This is outlined below and for further details see [64]. The basis of the construction is the probability of individual finite paths induced by the transition probability matrix $\mathbf{P}$. More precisely, the probability of the path $\rho{=}s_0 \ldots s_n$ is given by $\mathbf{P}(\rho) \overset{\text{def}}{=} \prod_{i=0}^{n-1} \mathbf{P}(s_i, s_{i+1})$. We begin by defining, for each finite path $\rho \in \mathit{FPath}_{\mathcal{D},s}$, the *basic cylinder* C_ρ that consists of all infinite paths starting with ρ. Using properties of cylinders, we can then construct the probability space $(\mathit{IPath}_{\mathcal{D},s}, \mathcal{F}_{\mathcal{D},s}, \mathit{Pr}_{\mathcal{D},s})$ where $\mathcal{F}_{\mathcal{D},s}$ is the smallest σ-algebra generated by the basic cylinders $\{C_\rho \mid \rho \in \mathit{FPath}_{\mathcal{D},s}\}$ and $\mathit{Pr}_{\mathcal{D},s}$ is the unique measure such that $\mathit{Pr}_{\mathcal{D},s}(C_\rho) = \mathbf{P}(\rho)$ for all $\rho \in \mathit{FPath}_{\mathcal{D},s}$.

Example 1. Consider the 3-state DTMC $\mathcal{D}{=}(S, \overline{s}, \mathbf{P}, L)$ of Figure 1. Here, $S{=}\{s_0, s_1, s_2\}$, $\overline{s}{=}s_0$, the transition probability matrix $\mathbf{P}$ is shown in Figure 1, $L(s_0){=}\{\mathit{init}\}$, $L(s_1){=}\varnothing$ and $L(s_2){=}\{\mathit{succ}\}$. We have, for example:

– $\mathit{Pr}_{\mathcal{D},s_0}(\{\pi \text{ starts } s_0 s_1 s_2 s_0\}) = 1{\cdot}0.3{\cdot}0.5 = 0.15$;
– $\mathit{Pr}_{\mathcal{D},s_0}(\{(s_0 s_1 s_2)^\omega\}) = \lim_{n\to\infty} \mathit{Pr}_{\mathcal{D},s_0}(\{\pi \text{ starts } (s_0 s_1 s_2)^n\})$
$= \lim_{n\to\infty} 1{\cdot}0.3{\cdot}(0.5{\cdot}1{\cdot}0.3)^{n-1} = 0$;
– $\mathit{Pr}_{\mathcal{D},s_0}(\{\pi \text{ contains } s_2\}) = \sum_{n=1}^{\infty} \mathit{Pr}_{\mathcal{D},s_0}(\{\pi \text{ starts } (s_0 s_1)^n s_2\})$
$= \sum_{n=1}^{\infty} 1{\cdot}(0.7{\cdot}1)^{n-1}{\cdot}0.3 = 1.$ ■

3 Markov Decision Processes

This tutorial focuses on the model of *Markov decision processes* (MDPs), which are a widely used formalism for modelling systems that exhibit both *probabilistic* and *nondeterministic* behaviour. From the point of view of applying *quantitative verification*, nondeterminism is an essential tool to capture several different aspects of system behaviour:

– *unknown environment*: if the system interacts with other components whose behaviour is unknown, this can be modelled with nondeterminism;

– *concurrency*: in a distributed system comprising multiple components operating in parallel, nondeterminism is used to represent the different possible interleavings of the executions of the components;
– *underspecification*: if certain parts of a system are either unknown or too complex to be modelled efficiently, these can be abstracted away using nondeterminism.

Alternatively, we can use nondeterminism to capture the possible ways that a *controller* can influence the behaviour of the system. The multi-objective techniques that we describe in Section 8 can be seen as a way of performing *controller synthesis*. In a similar vein, MDPs are also widely used in domains such as planning and robotics. Formally, we define an MDP as follows.

Definition 4 (Markov decision process). *A* Markov decision process *(MDP) is a tuple* $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ *where* S *is a* finite *set of states,* $\overline{s} \in S$ *is an initial state,* $\alpha_{\mathcal{M}}$ *is a* finite *alphabet,* $\delta_{\mathcal{M}} : S{\times}\alpha_{\mathcal{M}} \to \mathit{Dist}(S)$ *is a (partial) probabilistic transition function and* $L : S \to 2^{AP}$ *is a labelling function mapping each state to a set of atomic propositions taken from a set* AP*.*

Transitions between states in an MDP $\mathcal{M}$ occur in two steps. First, a choice between one or more *actions* from the alphabet $\alpha_{\mathcal{M}}$ is made. The set of available actions in a state s is given by $A(s) \stackrel{\text{def}}{=} \{a \in \alpha_{\mathcal{M}} \mid \delta_{\mathcal{M}}(s, a) \text{ is defined}\}$. To prevent deadlocks, we assume that $A(s)$ is non-empty for all $s \in S$. The selection of an action $a \in A(s)$ is *nondeterministic*. Secondly, a successor state s' is chosen randomly, according to the probability distribution $\delta_{\mathcal{M}}(s, a)$, i.e. the probability that a transition to s' occurs equals $\delta_{\mathcal{M}}(s, a)(s')$.

An *infinite path* through an MDP is a sequence $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$ (occasionally written as $s_0 a_0 s_1 a_1 \ldots$) where $s_i \in S$, $a_i \in A(s_i)$ and $\delta_{\mathcal{M}}(s_i, a_i)(s_{i+1}){>}0$ for all $i \in \mathbb{N}$. A *finite path* $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n$ is a prefix of an infinite path ending in a state. We denote by $\mathit{FPath}_{\mathcal{M},s}$ and $\mathit{IPath}_{\mathcal{M},s}$, respectively, the set of all finite and infinite paths starting from state s of $\mathcal{M}$. We use $\mathit{FPath}_{\mathcal{M}}$ and $\mathit{IPath}_{\mathcal{M}}$ for the sets of *all* such paths in the MDP. Where the context is clear, we will drop the subscript $\mathcal{M}$. For a finite path $\rho = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \xrightarrow{a_{n-1}} s_n$, $|\rho| = n$ denotes its length and $\mathit{last}(\rho) = s_n$ its last state. For a (finite or infinite) path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots$, its $(i{+}1)$th state s_i is denoted $\pi(i)$ and its *trace*, $\mathit{tr}(\pi)$, is the sequence of actions $a_0 a_1 \ldots$ When, in later parts of this tutorial, we formalise ways to define properties of MDPs, we will use both *action-based* properties, based on path traces, and *state-based* properties, using the atomic propositions assigned to each state by the labelling function L.

A *reward structure* on an MDP is useful for representing quantitative information about the system the MDP represents, for example, the power consumption, number of packets sent, size of a queue or the number of lost requests. Formally, we define rewards on both the states and actions of an MDP as follows.

Definition 5 (Reward structure). *A* reward structure *for an MDP* $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ *is a tuple* $r{=}(r_{state}, r_{action})$ *comprising a* state reward function $r_{state} : S \to \mathbb{R}_{\geqslant 0}$ *and an* action reward function $r_{action} : S{\times}\alpha_{\mathcal{M}} \to \mathbb{R}_{\geqslant 0}$.

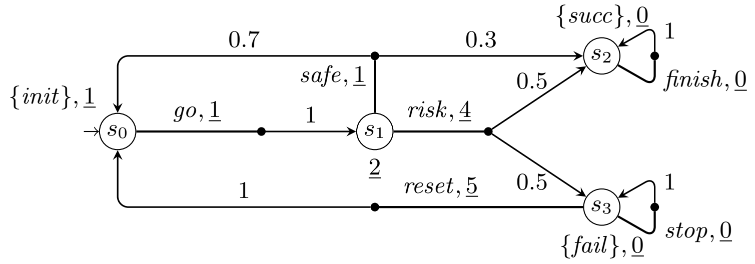


Fig. 2. A running example: an MDP, annotated with a reward structure

We consistently use the terminology *rewards* but, often, these will be used to model *costs*. The *action rewards* in a reward structure are also referred to elsewhere as *transition rewards*, *impulse rewards* or *state-action* rewards.

We next introduce the notion of *end components* which, informally, are parts of the MDP in which it possible to remain forever once entered.

Definition 6 (End component). *Let $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ be an MDP. An* end component *(EC) of $\mathcal{M}$ is a pair (S', δ') comprising a subset $S' \subseteq S$ of states and partial probabilistic transition function $\delta' : S' \times \alpha_{\mathcal{M}} \to Dist(S)$ satisfying the following conditions:*

- *(S', δ') defines a sub-MDP of $\mathcal{M}$, i.e. for all $s' \in S'$ and $a \in \alpha_M$, if $\delta'(s', a)$ is defined, then $\delta'(s', a) = \delta(s', a)$ and $\delta'(s', a)(s'') > 0$ only for states $s'' \in S'$;*
- *the underlying graph of (S', δ') is strongly connected.*

An EC (S', δ') is maximal *if there is no distinct EC (S'', δ'') such that for any $s \in S$ and $a \in \alpha_{\mathcal{M}}$, if $\delta'(s, a)$ is defined, then so is $\delta''(s, a)$.*

Algorithms to detect end components can be found in [1,11].

Example 2. Consider the MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ from Figure 2. Here, $S = \{s_0, s_1, s_2, s_3\}$, $\overline{s} = s_0$ and $\alpha_{\mathcal{M}} = \{go, risk, safe, finish, stop, reset\}$. Considering the probabilistic transition function, for example, from state s_1 we have:

$$\delta_{\mathcal{M}}(s_1, risk) = [\, s_2 \mapsto 0.5,\ s_3 \mapsto 0.5 \,]$$
$$\delta_{\mathcal{M}}(s_1, safe) = [\, s_0 \mapsto 0.7,\ s_2 \mapsto 0.3 \,]$$

and $\delta(s_1, a)$ is undefined if $a \in \{go, finish, stop, reset\}$, i.e. $A(s_1) = \{risk, safe\}$. The labelling of e.g. states s_1 and s_2 is given by $L(s_1) = \varnothing$ and $L(s_2) = \{succ\}$.

The MDP models a system that aims to execute a task. After some routine initialisation (captured by the action *go*), there are two possibilities: it can either perform the task using a *safe* procedure, in which case the probability of finishing the task successfully is 0.3, but with probability 0.7 the system restarts; or, it can perform the task by a *risk*y procedure, in which case the probability of finishing the task is higher (0.5), but there is a 50% chance of complete failure, after which the system can only be restarted using the action *reset*.

State and action reward functions r_{state} and r_{action} are also depicted in the figure where the rewards are the underlined numbers next to states or action labels, e.g. $r_{state}(s_1)$=2 and $r_{action}(s_1, \mathit{risk})$=4. An example of a finite path is $\rho = s_0 \xrightarrow{go} s_1 \xrightarrow{risk} s_3$ and an example of an infinite path is:

$$\pi = s_0 \xrightarrow{go} s_1 \xrightarrow{safe} s_0 \xrightarrow{go} s_1 \xrightarrow{safe} s_0 \xrightarrow{go} \cdots$$

The MDP contains two end components, namely $(\{s_2\}, \{(s_2, \mathit{stop}) \mapsto [s_2 \mapsto 1]\})$ and $(\{s_3\}, \{(s_3, \mathit{stop}) \mapsto [s_3 \mapsto 1]\})$. Both are maximal. ■

Adversaries. To reason formally about MDPs, in the way described for DTMCs in Section 2.2, we need a probability space over infinite paths. However, a probability space can only be constructed once all the nondeterminism present has been resolved. Each possible resolution of nondeterminism is represented by an *adversary*, which is responsible for choosing an action in each state of the MDP, based on the history of its execution so far. Adversaries are, depending on the context, often referred to by a variety of other names, including *strategies*, *schedulers* and *policies*.

Definition 7 (Adversary). *An* adversary *of an MDP* $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ *is a function* $\sigma : \mathit{FPath}_{\mathcal{M}} \to \mathit{Dist}(\alpha_{\mathcal{M}})$ *such that* $\sigma(\rho)(a) > 0$ *only if* $a \in A(\mathit{last}(\rho))$.

In general, the choice of an action can be made randomly and depend on the full history of the MDP, but is limited to the actions available in the current state. The set of *all* adversaries of $\mathcal{M}$ is $\mathit{Adv}_{\mathcal{M}}$. There are several important classes of adversaries that we will now summarise. An adversary σ is *deterministic* if $\sigma(\rho)$ is a point distribution for all $\rho \in \mathit{FPath}_{\mathcal{M}}$, and *randomised* otherwise.

Definition 8 (Memoryless adversary). *An adversary* σ *is* memoryless *if* $\sigma(\rho)$ *depends only on* $\mathit{last}(\rho)$*, that is, for any* $\rho, \rho' \in \mathit{FPath}_{\mathcal{M}}$ *such that* $\mathit{last}(\rho) = \mathit{last}(\rho')$*, we have* $\sigma(\rho) = \sigma(\rho')$.

Definition 9 (Finite-memory adversary). *An adversary* σ *is* finite-memory *if there exists a tuple* $(Q, \overline{q}, \sigma_u, \sigma_s)$ *comprising:*

– *a finite set of* modes Q;
– *an initial mode* $\overline{q} \in Q$;
– *a mode update function* $\sigma_u : Q \times \alpha_{\mathcal{M}} \times S \to Q$;
– *an action selection function* $\sigma_s : Q \times S \to \mathit{Dist}(\alpha_{\mathcal{M}})$

such that $\sigma(\rho) = \sigma_s(\hat{\sigma}_u(\rho), \mathit{last}(\rho))$ *for all* $\rho \in \mathit{FPath}_{\mathcal{M}}$*, where* $\hat{\sigma}_u : \mathit{FPath}_{\mathcal{M}} \to Q$ *is the function determined by* $\hat{\sigma}_u(s) = \overline{q}$ *and* $\hat{\sigma}_u(\rho a s) = \sigma_u(\hat{\sigma}_u(\rho), a, s)$ *for all* $\rho \in \mathit{FPath}_{\mathcal{M}}$*,* $a \in \alpha_{\mathcal{M}}$*, and* $s \in S$.

Notice that a memoryless adversary is a finite-memory adversary with one mode.

Under a particular adversary σ, the behaviour of an MDP $\mathcal{M}$ is fully probabilistic and can be captured by a (countably infinite-state) discrete-time Markov chain, denoted $\mathcal{M}_\sigma$, each state of which is a finite path of $\mathcal{M}$.

Definition 10 (Induced DTMC). *For an MDP* $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ *and adversary* $\sigma \in \mathit{Adv}_{\mathcal{M}}$*, the* induced DTMC *is* $\mathcal{M}_\sigma = (T, \overline{s}, \mathbf{P}, L')$ *where:*

- *$T = FPath_{\mathcal{M}}$;*
- *for any $\rho, \rho' \in FPath_{\mathcal{M}}$:*

$$\mathbf{P}(\rho, \rho') = \begin{cases} \sigma(\rho)(a) \cdot \delta_{\mathcal{M}}(last(\rho), a)(s) & \text{if } \rho' = \rho a s,\ a \in A(last(\rho)) \text{ and } s \in S \\ 0 & \text{otherwise;} \end{cases}$$

- *$L'(\rho) = L(last(\rho))$ for all $\rho \in FPath_{\mathcal{M}}$.*

The induced DTMC $\mathcal{M}_\sigma$ has an infinite number of states. However, in the case of finite-memory adversaries (and hence also the subclass of memoryless adversaries), we can also construct a finite-state *quotient DTMC*. More precisely, if the finite-memory adversary is defined by the tuple $(Q, \overline{q}, \sigma_u, \sigma_s)$, then we stipulate two paths ρ and ρ' to be equivalent whenever they get mapped to the same mode, i.e. $\hat{\sigma}_u(\rho)=\hat{\sigma}_u(\rho')$, and the last action and state of ρ and ρ' are the same. It follows that we can classify equivalence classes of paths by tuples of the form (q, a, s), where q is the mode that paths of the equivalence class get mapped to and a and s are the last action and state of these paths. Formally, the finite-state quotient can be defined as follows (since the path consisting of just the initial state does not have a “last” action, we introduce a new symbol $\perp$).

Definition 11 (Quotient DTMC). *The* quotient DTMC *for an MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ and finite-memory adversary σ defined by the tuple $(Q, \overline{q}, \sigma_u, \sigma_s)$ is the finite-state DTMC $\mathcal{M}^q_\sigma = (T, \overline{s}', \mathbf{P}, L')$ where:*

- *$T = (Q \times \alpha_{\mathcal{M}} \times S) \cup \{(\overline{q}, \perp, \overline{s})\}$;*
- *$\overline{s}' = (\overline{q}, \perp, \overline{s})$;*
- *$\mathbf{P}((q, a, s), (q', a', s')) = \sigma_s(q, s)(a') \cdot \delta_{\mathcal{M}}(s, a')(s')$ whenever $q' = \sigma_u(q, a', s')$ and equals 0 otherwise;*
- *$L'((q, a, s)) = L(s)$.*

Probability Spaces. For a given (general) adversary σ, we associate the infinite paths in $\mathcal{M}$ and $\mathcal{M}^\sigma$ by defining the following bijection f:

$$f(s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots) \stackrel{\text{def}}{=} (s_0)\,(s_0 a_0 s_1)\,(s_0 a_0 s_1 a_1 s_2)\,\ldots$$

for all infinite paths $s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} s_2 \xrightarrow{a_2} \cdots \in IPath_{\mathcal{M}}$. Now, for any state s of $\mathcal{M}$, using this function and the probability measure $Pr_{\mathcal{M}_\sigma,s}$ given in Section 2.2 for the DTMC $\mathcal{M}_\sigma$, we can define a probability measure $Pr^\sigma_{\mathcal{M},s}$ over $IPath_{\mathcal{M},s}$ capturing the behaviour of $\mathcal{M}$ from state s under adversary σ. Furthermore, for a random variable $X : IPath_{\mathcal{M},s} \to \mathbb{R}_{\geqslant 0}$, we can compute the expected value of X from state s in $\mathcal{M}$ under adversary σ, which we denote by $\mathbb{E}^\sigma_{\mathcal{M},s}(X)$.

In practice, we are mainly interested in *minimising* or *maximising* either the probability of a certain set of paths, or the expected value of some random variable. Therefore, for any measurable set of paths $\Omega \subseteq IPath_{\mathcal{M},s}$ and random variable $X : IPath_{\mathcal{M},s} \to \mathbb{R}_{\geqslant 0}$, we define:

$$\begin{aligned} Pr^{\min}_{\mathcal{M},s}(\Omega) &\stackrel{\text{def}}{=} \inf\nolimits_{\sigma \in Adv_{\mathcal{M}}} Pr^\sigma_{\mathcal{M},s}(\Omega) \\ Pr^{\max}_{\mathcal{M},s}(\Omega) &\stackrel{\text{def}}{=} \sup\nolimits_{\sigma \in Adv_{\mathcal{M}}} Pr^\sigma_{\mathcal{M},s}(\Omega) \\ \mathbb{E}^{\min}_{\mathcal{M},s}(X) &\stackrel{\text{def}}{=} \inf\nolimits_{\sigma \in Adv_{\mathcal{M}}} \mathbb{E}^\sigma_{\mathcal{M},s}(X) \\ \mathbb{E}^{\max}_{\mathcal{M},s}(X) &\stackrel{\text{def}}{=} \sup\nolimits_{\sigma \in Adv_{\mathcal{M}}} \mathbb{E}^\sigma_{\mathcal{M},s}(X) \end{aligned}$$

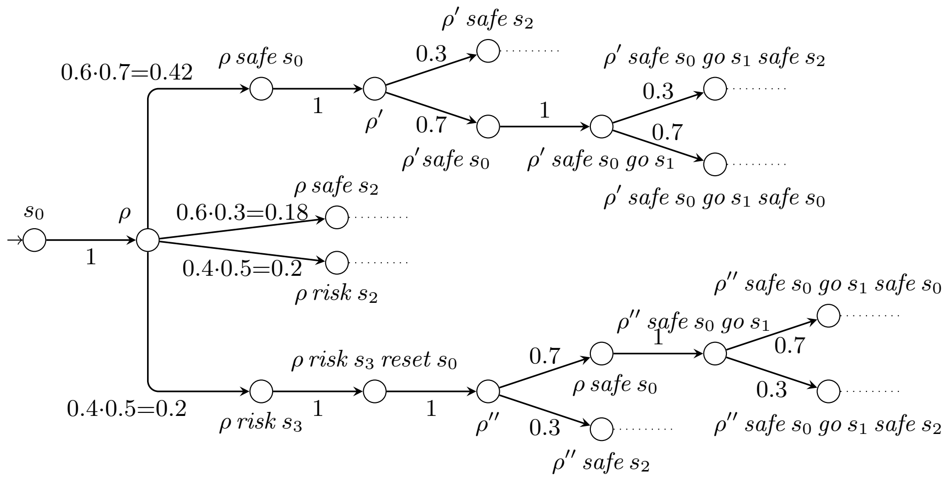


Fig. 3. Fragment of the induced DTMC $\mathcal{M}_\sigma$ (where $\rho = s_0\,go\,s_1$, $\rho' = \rho\,safe\,s_0\,go\,s_1$ and $\rho'' = \rho\,risk\,s_3\,reset\,s_0\,go\,s_1$)

When using an MDP to model and verify quantitative properties of a system, this equates to evaluating the *best-* or *worst-case* behaviour that can arise; for example, we may be interested in "the minimum probability of a message being delivered" or "the maximum expected time for a task to be completed".

Although not every subset of $IPath_{\mathcal{M}}$ is measurable, all the sets we will consider in this tutorial are measurable, so we can use the above notation freely without stressing the need of the measurability every time.

Example 3. Consider again the example MDP from Figure 2. Let σ be the adversary such that, for any finite path ρ:

$$\sigma(\rho) = \begin{cases} [go{\mapsto}1] & \text{if } last(\rho){=}s_0 \\ [risk{\mapsto}0.4, safe{\mapsto}0.6] & \text{if } \rho = s_0\xrightarrow{go}s_1 \\ [safe{\mapsto}1] & \text{if } last(\rho){=}s_1 \text{ and } \rho \neq s_0\xrightarrow{go}s_1 \\ [finish{\mapsto}1] & \text{if } last(\rho){=}s_2 \\ [reset{\mapsto}1] & \text{if } last(\rho){=}s_3\,. \end{cases}$$

Part of the induced DTMC $\mathcal{M}_\sigma$ is depicted in Figure 3. The adversary σ is neither memoryless, nor deterministic, but is finite-memory. More precisely, σ is defined by the tuple $(Q, \overline{q}, \sigma_u, \sigma_s)$ where $Q = \{q_0, q_1\}$, $\overline{q} = q_0$ and, for any $q \in Q$, $a \in \alpha_{\mathcal{M}}$ and $s \in S$:

$$\sigma_u(q,a,s) \;=\; \begin{cases} q_1 & \text{if } q{=}q_0 \text{ and } a \in \{risk, safe\} \\ q_0 & \text{if } q{=}q_0 \text{ and } a \notin \{risk, safe\} \\ q_1 & \text{otherwise} \end{cases}$$

and:

$$\sigma_s(q,s) \;=\; \begin{cases} [go{\mapsto}1] & \text{if } s{=}s_0 \\ [risk{\mapsto}0.4, safe{\mapsto}0.6] & \text{if } q{=}q_0 \text{ and } s{=}s_1 \\ [safe{\mapsto}1] & \text{if } q{=}q_1 \text{ and } s{=}s_1 \\ [finish{\mapsto}1] & \text{if } s{=}s_2 \\ [reset{\mapsto}1] & \text{if } s{=}s_3\,. \end{cases}$$

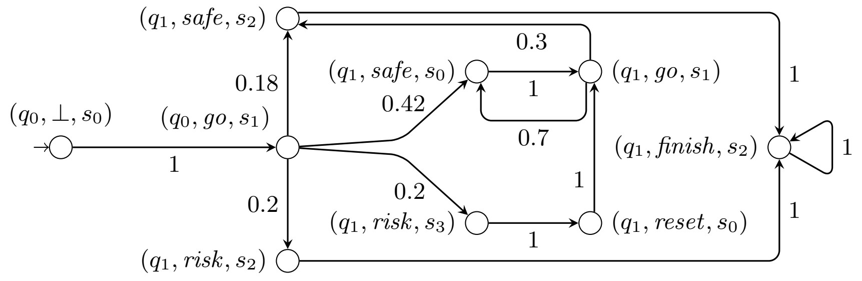


Fig. 4. The quotient DTMC $\mathcal{M}^q_\sigma$

The quotient DTMC $\mathcal{M}^q_\sigma$ is depicted in Figure 4.

Returning to the finite path $\rho = s_0 \xrightarrow{go} s_1 \xrightarrow{risk} s_3$ from Example 2, we have $Pr^\sigma_{\mathcal{M},s_0}(\{\pi \mid \rho \text{ is a prefix of } \pi\}) = (1{\cdot}1){\cdot}(0.4{\cdot}0.5) = 0.2$. ■

Related Models. We conclude this section by briefly surveying models that are closely related to MDPs and clarifying certain differences in terminology used elsewhere. Our definition of MDPs in this tutorial essentially coincides with the classical definitions (see e.g. [14,57,76]), although there are notational differences. Also commonly used in probabilistic verification is the model of (simple) *probabilistic automata* (PAs), due to Segala [80,81]. These permit multiple distributions labelled with the same action to be enabled from a state (i.e. $\delta_\mathcal{M}$ is a relation $\delta_\mathcal{M} \subseteq S{\times}\alpha_\mathcal{M}{\times}Dist(S)$), thus strictly generalising MDPs. This model is particularly well suited to compositional modelling and analysis of probabilistic systems, a topic that we will discuss further in Section 9. The names are sometimes used interchangeably, for example, the PRISM model checker [56] supports both PAs and MDPs, but refers to them simply as MDPs.

Confusingly, there is an alternative model called *probabilistic automata*, due to Rabin [78], which is also well known. From a syntactic point of view, these are essentially the same as MDPs, but are typically used in a language-theoretic setting, rather than for modelling and verification. An exception is [72], which uses Rabin's probabilistic automata to build a game-theoretic framework for verifying probabilistic programs. Another approach is to use "alternating" models, which distinguish between states that offer a probabilistic choice and those that offer a nondeterministic choice. Examples include the model of Hansson [53] and the concurrent Markov chains of [34,84]. We do not attempt a complete survey of MDP-like models here. See [48] for a classification scheme of such models, [82] for a thorough comparison and [80,1,7] for further references and discussion.

4 Probabilistic Reachability

In the remainder of this tutorial, we will introduce a variety of properties of MDPs and describe the corresponding methods to perform probabilistic model checking. We begin with the simple, yet fundamental, property of *probabilistic* *reachability*. This refers to the minimum or maximum probability, when starting from a given state s, of reaching a set of target states $T \subseteq S$. To formalise this, let $reach_s(T)$ be the set of all paths that start from state s and contain a state from T. More precisely, we have:

$$reach_s(T) \overset{\text{def}}{=} \{\pi \in IPath_{\mathcal{M},s} \mid \pi(i) \in T \text{ for some } i \in \mathbb{N}\}$$

and, when s is clear from the context, we will write $reach(T)$ instead of $reach_s(T)$. The measurability of $reach_s(T)$ follows from the fact that $reach_s(T) = \cup_{\rho \in I}\{\pi \in IPath_{\mathcal{M},s} \mid \pi \text{ has prefix } \rho\}$, where I is the (countable) set of all finite paths from s ending in T, and each element of this union is measurable. We then aim to compute one or both of the following probability bounds:

$$\begin{aligned} Pr^{\min}_{\mathcal{M},s}(reach_s(T)) &\overset{\text{def}}{=} \inf_{\sigma \in Adv_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(reach_s(T)) \\ Pr^{\max}_{\mathcal{M},s}(reach_s(T)) &\overset{\text{def}}{=} \sup_{\sigma \in Adv_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(reach_s(T))\,. \end{aligned}$$

In the remainder of this section, we will first consider the special case of *qualitative* reachability, that is, finding those states for which the probability is either 0 or 1. Next, we consider the general *quantitative* case and discuss several different approaches that can be used to either compute or approximate minimum and maximum reachability probabilities. Finally, we describe how to generate adversaries which achieve the reachability probability of interest. Further details about many of the methods described in this section can be found in [76]. One important fact that we use is that there always exist *deterministic* and *memoryless* adversaries that achieve the minimum and maximum probabilities of reaching a target T.

4.1 Qualitative Reachability

In this section, we will present methods for finding the sets of states for which the minimum or maximum reachability probability is either 0 or 1. More precisely, we will be concerned with constructing the following sets of states:

$$\begin{aligned} S^0_{\min} &\overset{\text{def}}{=} \{s \in S \mid Pr^{\min}_s(reach_s(T)){=}0\} \\ S^1_{\min} &\overset{\text{def}}{=} \{s \in S \mid Pr^{\min}_s(reach_s(T)){=}1\} \\ S^0_{\max} &\overset{\text{def}}{=} \{s \in S \mid Pr^{\max}_s(reach_s(T)){=}0\} \\ S^1_{\max} &\overset{\text{def}}{=} \{s \in S \mid Pr^{\max}_s(reach_s(T)){=}1\}\,. \end{aligned}$$

The probability 0 cases are often *required* as input to the algorithms for quantitative reachability, while using both can reduce round-off errors and yield a speed-up in verification time. The gains are attributed to the fact that, to perform qualitative analysis, it is sufficient to know whether transitions are possible, not their precise probabilities. Hence, the analysis can be performed using *graph-based*, rather than numerical, computation.

Algorithms 1–4 give a formal description of how to compute the above sets; examples of executing these algorithms are presented in Section 4.2. For further details, see [18,1].

```
  Input: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, target set $T \subseteq S$
  Output: the set $S^0_{\min} = \{s \in S \mid Pr^{\min}_s(\mathit{reach}(T))=0\}$
1 $R := T$;
2 do
3     $R' := R$;
4     $R := R' \cup \{\, s \in S \mid \forall a \in A(s).\, (\, \exists s' \in R'.\, \delta_{\mathcal{M}}(s,a)(s')>0\,)\,\}$;
5 while $R \neq R'$;
6 return $S \setminus R$;
```

Algorithm 1: Computation of $S^0_{\min}$

```
  Input: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, set $S^0_{\min}$
  Output: the set $S^1_{\min} = \{s \in S \mid Pr^{\min}_s(\mathit{reach}(T))=1\}$
1 $R := S \setminus S^0_{\min}$;
2 do
3     $R' := R$;
4     $R := R' \setminus \{\, s \in R' \mid \exists a \in A(s).\, \big(\exists s' \in S.\, (\, \delta_{\mathcal{M}}(s,a)(s')>0 \wedge s' \notin R'\,)\,\big)\,\}$;
5 while $R \neq R'$;
6 return $R$;
```

Algorithm 2: Computation of $S^1_{\min}$

4.2 Quantitative Reachability

Before we introduce the actual algorithms for computing minimum and maximum reachability probabilities, we present the *Bellman equations* that describe these probabilities. It should be apparent that, if $x_s = Pr^{\min}_s(\mathit{reach}(T))$ for all $s \in S$, then the following equations are satisfied:

$$\begin{array}{ll} x_s = 1 & \text{if } s \in S^1_{\min} \\ x_s = 0 & \text{if } s \in S^0_{\min} \\ x_s = \min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'} & \text{otherwise.} \end{array}$$

When solving these equations, we in fact find the probability $Pr^{\min}_s(\mathit{reach}(T))$ *for all* states s of the MDP, rather than just a specific state of interest. From the results presented in [17,16] (since the problem of finding minimum reachability probabilities is a special case of the *stochastic shortest path problem*), the equations above have a *unique* solution. Furthermore, it is actually sufficient to just compute $S^0_{\min}$ and replace $S^1_{\min}$ with T in the above. Below, we will discuss various methods to solve these equations.

Similarly, if $x_s = Pr^{\max}_s(\mathit{reach}(T))$, then the following equations are satisfied:

$$\begin{array}{ll} x_s = 1 & \text{if } s \in S^1_{\max} \\ x_s = 0 & \text{if } s \in S^0_{\max} \\ x_s = \max_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'} & \text{otherwise.} \end{array}$$

In this case, from the results of [17,1], it follows that the maximum reachability probabilities are the *least* solution to these equations. Again, like for the minimum case, it suffices to just compute $S^0_{\max}$ and replace $S^1_{\max}$ with T.

```
**Input**: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, target set $T \subseteq S$
**Output**: the set $S^0_{\max} = \{s \in S \mid Pr^{\max}_s(\mathit{reach}(T)){=}0\}$
1  $R := T$;
2  do
3      $R' := R$;
4      $R := R' \cup \{\, s \in S \mid \exists a \in A(s).\, (\, \exists s' \in R'.\, \delta_{\mathcal{M}}(s,a)(s'){>}0\,)\}$;
5  while $R \neq R'$;
6  return $S \setminus R$;
```

Algorithm 3: Computation of $S^0_{\max}$

```
**Input**: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, target set $T \subseteq S$
**Output**: $S^1_{\max} = \{s \in S \mid Pr^{\max}_s(\mathit{reach}(T)){=}1\}$
1  $R := S$;
2  do
3      $R' := R$;
4      $R := T$;
5      do
6          $R'' := R$;
7          $R := R'' \cup \{s \in S \mid \exists a \in A(s).$
8           $(\forall s' \in S.\, (\, \delta_{\mathcal{M}}(s,a)(s'){>}0 \to s' \in R'\,)\,) \wedge (\,\exists s' \in R''.\, \delta_{\mathcal{M}}(s,a)(s'){>}0\,)\}$;
9      while $R \neq R''$;
10 while $R \neq R'$;
11 return $R$;
```

Algorithm 4: Computation of $S^1_{\max}$

Linear Programming. One approach to computing the minimum and maximum reachability probabilities is to construct and solve a *linear programming* (LP) problem. In the case of minimum probabilities $Pr^{\min}_s(\mathit{reach}(T))$, it has been demonstrated [17,35,1] that the following linear program:

maximise $\sum_{s \in S} x_s$ subject to the constraints:

$x_s = 1$	for all $s \in S^1_{\min}$
$x_s = 0$	for all $s \in S^0_{\min}$
$x_s \leqslant \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}$	for all $s \notin S^1_{\min} \cup S^0_{\min}$ and $a \in A(s)$

has a unique solution satisfying $x_s = Pr^{\min}_s(\mathit{reach}(T))$.

Example 4. Consider again the example from Figure 2 and let us compute $Pr^{\min}_s(\mathit{reach}(\{s_2\}))$ for all $s \in S$. We first need to execute Algorithm 1, starting with $R = \{s_2\}$, and changing R to $\{s_1, s_2\}$ and to $\{s_0, s_1, s_2\}$ in two consecutive iterations of the do-while loop. This is a fixed point, so the returned set $S^0_{\min}$ is $S \backslash \{s_0, s_1, s_2\} = \{s_3\}$. Next, we execute Algorithm 2, starting with $R{=}\{s_0, s_1, s_2\}$ and then consecutively change R to $\{s_0, s_2\}$ and $\{s_2\}$, which is the fixed point, so $S^1_{\min} = \{s_2\}$. Using these sets, we obtain the linear program:

$$\begin{aligned}
&\text{maximise } x_{s_0}{+}x_{s_1}{+}x_{s_2}{+}x_{s_3} \text{ subject to:}\\
&x_{s_2} = 1\\
&x_{s_3} = 0\\
&x_{s_0} \leqslant x_{s_1}\\
&x_{s_1} \leqslant 0.5{\cdot}x_{s_2} + 0.5{\cdot}x_{s_3}\\
&x_{s_1} \leqslant 0.7{\cdot}x_{s_0} + 0.3{\cdot}x_{s_2}
\end{aligned}$$

which has the unique solution x_{s_0}=0.5, x_{s_1}=0.5, x_{s_2}=1 and x_{s_3}=0. Hence, the vector of values for $Pr^{\min}_s(\mathit{reach}(\{s_2\}))$ is $(0.5, 0.5, 1, 0)$. ■

In the case of maximum probabilities, the situation is similar [35,1], and we have the following the linear program:

$$\boxed{\begin{array}{ll}
\text{minimise } \sum_{s\in S} x_s \text{ subject to the constraints:} & \\
x_s = 1 & \text{for all } s \in S^1_{\max}\\
x_s = 0 & \text{for all } s \in S^0_{\max}\\
x_s \geqslant \sum_{s'\in S} \delta(s,a)(s')\cdot x_{s'} & \text{for all } s \notin S^1_{\max} \cup S^0_{\max} \text{ and } a \in A(s)
\end{array}}$$

which yields a unique solution satisfying $x_s = Pr^{\max}_s(\mathit{reach}(T))$.

Example 5. We will illustrate the computation of maximum reachability probabilities using the MDP from Figure 2 and the target set $\{s_3\}$. We first run Algorithm 3, initialising the set R to $\{s_3\}$. After the first iteration of the do-while loop, we get R=$\{s_1, s_3\}$, and after the second we get R=$\{s_0, s_1, s_3\}$, which is the fixed point, so the returned set is $S^0_{\max}$=$\{s_2\}$. Then, we execute Algorithm 4. In the outer do-while loop (lines 2–10), we start with R'=S and R=$\{s_3\}$. The first execution of the inner loop (lines 5–9) yields R=$\{s_0, s_1, s_3\}$ and the second yields R=$\{s_3\}$. The latter is the fixed point, so we return $S^1_{\max}$=$\{s_3\}$. Setting up the linear program, we get:

$$\begin{aligned}
&\text{minimise } x_{s_0}{+}x_{s_1}{+}x_{s_2}{+}x_{s_3} \text{ subject to:}\\
&x_{s_3} = 1\\
&x_{s_2} = 0\\
&x_{s_0} \geqslant x_{s_1}\\
&x_{s_1} \geqslant 0.5{\cdot}x_{s_2} + 0.5{\cdot}x_{s_3}\\
&x_{s_1} \geqslant 0.7{\cdot}x_{s_0} + 0.3{\cdot}x_{s_2}
\end{aligned}$$

which has the unique solution x_{s_0}=0.5, x_{s_1}=0.5, x_{s_2}=0 and x_{s_3}=1, giving the vector of values $(0.5, 0.5, 0, 1)$ for $Pr^{\max}_s(\mathit{reach}(\{s_3\}))$. ■

The benefit of the linear programming approach is that it can be used to compute *exact* answers. The drawback, however, is that its scalability to large models is limited. Despite a wide range of LP solution methods being available, in practice models used in probabilistic verification become too large to solve in this way.

```
Input: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, sets $S^0_{\min}, S^1_{\min}$ and convergence criterion $\varepsilon$
Output: (approximation of) $Pr^{\min}_s(reach(T))$ for all $s \in S$
1 foreach $s \in S$ do $x_s := \begin{cases} 1 \text{ if } s \in S^1_{\min} \\ 0 \text{ otherwise} \end{cases}$;
2 do
3     foreach $s \in S \backslash (S^0_{\min} \cup S^1_{\min})$ do
4         $x'_s := \min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}$;
5     end
6     $\delta := \max_{s \in S} (x'_s - x_s)$;
7     foreach $s \in S \backslash (S^0_{\min} \cup S^1_{\min})$ do $x_s := x'_s$;
8 while $\delta > \varepsilon$;
9 return $(x'_s)_{s \in S}$
```

Algorithm 5: Value iteration for $Pr^{\min}_s(reach(T))$

Value Iteration. An alternative method is *value iteration*, which offers better scalability than linear programming, but at the expense of accuracy. Instead of computing a precise solution to the set of linear equations for $Pr^{\min}_s(reach(T))$, it computes the probability of reaching T within n steps. For large enough n, this yields a good enough approximation in practice.

Formally, we introduce variables x^n_s for $s \in S$ and $n \in \mathbb{N}$ and equations:

$$x^n_s = \begin{cases} 1 & \text{if } s \in S^1_{\min} \\ 0 & \text{if } s \in S^0_{\min} \\ 0 & \text{if } s \notin (S^1_{\min} \cup S^0_{\min}) \text{ and } n{=}0 \\ \min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x^{n-1}_{s'} & \text{otherwise.} \end{cases}$$

It can be shown [76,1,7] that $\lim_{n\to\infty} x^n_s = Pr^{\min}_s(reach(T))$. We can thus approximate $Pr^{\min}_s(reach(T))$ by computing x^n_s for sufficiently large n. Furthermore, we can compute the maximum probabilities $Pr^{\max}_s(reach(T))$ in near-identical fashion, by replacing "min" with "max" in the above.

Typically, a suitable value of n is not decided in advance, but rather determined on-the-fly, based on the convergence of the values x^n_s. A simple but effective scheme is to terminate the computation when $\max_{s \in S}(x^n_s - x^{n-1}_s)$ drops below a specified threshold ε. In cases where the probability values are very small, the maximum *relative* difference, i.e. $\max_{s \in S}((x^n_s - x^{n-1}_s)/x^{n-1}_s)$ may be a more reliable criterion. It is important to stress, however, that these tests do *not* guarantee that the resulting values are within ε of the true answer. In theory, it is possible to make certain guarantees on the precision obtained, based on the denominators of the (rational) transition probabilities [27]. However, it is unclear whether these are practically applicable.

An illustration of how value iteration can be implemented is given in Algorithm 5. In practice, there is no need to store all vectors $\mathbf{x}^n$; just two ($\mathbf{x}$ and $\mathbf{x}'$ in the algorithm) are required.

Example 6. To illustrate value iteration, we will slightly modify the running example from Figure 2: let us suppose that the action *reset* is not available in s_3,

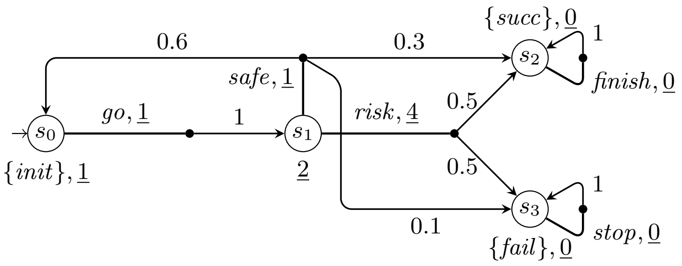


Fig. 5. A modified version of the running example from Figure 2

and that the action *safe* is not completely reliable, but results in a failure with probability 0.1 and leads to s_0 only with probability 0.6. The modified MDP is depicted in Figure 5. Suppose we want to compute $Pr^{\max}_{s_0}(reach(\{s_2\}))$. By executing Algorithms 3 and 4, we obtain $S^0_{\max}=\{s_3\}$ and $S^1_{\max}=\{s_2\}$, yielding the following equations for value iteration:

$$\begin{array}{lll}
x^n_{s_2} & = 1 & \text{for } n\geqslant 0 \\
x^n_{s_3} & = 0 & \text{for } n\geqslant 0 \\
x^0_{s_i} & = 0 & \text{for } i\in\{0,1\} \\
x^n_{s_0} & = x^{n-1}_{s_1} & \text{for } n>0 \\
x^n_{s_1} & = \max\{0.6{\cdot}x^{n-1}_{s_0}+0.1{\cdot}x^{n-1}_{s_3}+0.3{\cdot}x^{n-1}_{s_2}, 0.5{\cdot}x^{n-1}_{s_2}+0.5{\cdot}x^{n-1}_{s_3}\} & \text{for } n>0
\end{array}$$

Below, are the vectors $\mathbf{x}^n = (x^n_{s_0}, x^n_{s_1}, x^n_{s_2}, x^n_{s_3})$, shown up to a precision of 5 decimal places, for increasing n, terminating with ε=0.001.

$$\begin{array}{ll}
\mathbf{x}^0 = (0.0,\ 0.0,\ 1.0, 0.0) & \mathbf{x}^7 = (0.66,\ 0.696,\ 1.0, 0.0) \\
\mathbf{x}^1 = (0.0,\ 0.5,\ 1.0, 0.0) & \mathbf{x}^8 = (0.696,\ 0.696,\ 1.0, 0.0) \\
\mathbf{x}^2 = (0.5,\ 0.5,\ 1.0, 0.0) & \mathbf{x}^9 = (0.696,\ 0.7176,\ 1.0, 0.0) \\
\mathbf{x}^3 = (0.5,\ 0.6,\ 1.0, 0.0) & \mathbf{x}^{10} = (0.7176,\ 0.7176,\ 1.0, 0.0) \\
\mathbf{x}^4 = (0.6,\ 0.6,\ 1.0, 0.0) & \cdots \\
\mathbf{x}^5 = (0.6,\ 0.66, 1.0, 0.0) & \mathbf{x}^{22} = (0.74849, 0.74849, 1.0, 0.0) \\
\mathbf{x}^6 = (0.66, 0.66, 1.0, 0.0) & \mathbf{x}^{23} = (0.74849, 0.74909, 1.0, 0.0)
\end{array}$$

The exact values for $Pr^{\max}_{s}(reach(\{s_2\}))$ are $(0.75, 0.75, 1, 0)$, which differ from $\mathbf{x}^{23}$ by up to 0.00151 (for state s_0). ■

Gauss-Seidel Value Iteration. Several variants of value iteration exist that improve its efficiency. One such variant is *Gauss-Seidel* value iteration. Intuitively, this method exhibits faster convergence by using the most up-to-date probability values available for each state within each iteration of the computation. This has the added benefit that only a single vector of probabilities needs to be stored, since new values are written directly to the vector. Algorithm 6 shows the algorithm for the case of minimum reachability probabilities. Notice that it uses only a single vector $\mathbf{x}$.

Policy Iteration. An alternative class of algorithms for computing reachability probabilities is *policy iteration* (recall that "policy" is alternative terminology

```
   Input: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, sets $S^0_{\min}, S^1_{\min}$ and convergence criterion $\varepsilon$
   Output: (approximation of) $Pr^{\min}_s(reach(T))$ for all $s \in S$
 1 foreach $s \in S$ do $x_s := \begin{cases} 1 \text{ if } s \in S^1_{\min} \\ 0 \text{ otherwise} \end{cases}$;
 2 do
 3     $\delta := 0$;
 4     foreach $s \in S \backslash (S^0_{\min} \cup S^1_{\min})$ do
 5         $x_{new} := \min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}$;
 6         $\delta := \max(\delta, x_{new} - x_s)$;
 7         $x_s := x_{new}$;
 8     end
 9 while $\delta > \varepsilon$;
10 return $(x_s)_{s \in S}$
```

Algorithm 6: Gauss-Seidel value iteration for $Pr^{\min}_s(reach(T))$

for "adversary"). Whereas value iteration steps through vectors of values, policy iteration generates a sequence of adversaries. We start with an arbitrary, deterministic and memoryless adversary, and then repeatedly construct an improved (deterministic and memoryless) adversary by computing the corresponding probabilities and changing the actions taken so that the probability of reaching T is decreased or increased (depending on whether minimum or maximum probabilities are being computed). The existence of deterministic and memoryless adversaries exhibiting minimum and maximum reachability probabilities, together with properties of the reachability problem, implies that this method will return the correct result (assuming it terminates). Termination is guaranteed by the fact that there are only finitely many such adversaries.

Algorithm 7 describes policy iteration for computing $Pr^{\min}_s(reach(T))$; the case for maximum values is similar. Notice that, since the adversary is both deterministic and memoryless, we can describe it simply as a mapping σ from states to actions. Computing the probabilities $Pr^{\sigma}_s(reach(T))$ for each adversary σ is relatively straightforward and is done by computing reachability probabilities for the corresponding quotient DTMC $\mathcal{M}^q_\sigma$. Since $\mathcal{M}^q_\sigma$ is finite-state, this can be done either by treating it as a (finite-state) MDP and using the other methods described in this section, or by solving a linear equation system [67]. We remark also that, to ensure termination of the algorithm, the action assigned to $\sigma(s)$ when improving the policy should only be changed when there is a strict improvement in the probability for s.

Example 7. To demonstrate the policy iteration method, let us modify the MDP from Figure 2 by adding a self-loop on s_0 labelled with a new action *wait* (i.e. $\delta_{\mathcal{M}}(s_0, wait) = [s_0{\mapsto}1]$). Note that there are $2^3{=}8$ different deterministic and memoryless adversaries in the new MDP. Let us maximise the probability of reaching $\{s_2\}$. We start with the adversary σ that picks *wait*, *safe* and *stop* in s_0, s_1 and s_3, respectively. The vector of probabilities of reaching the state s_2 under σ is $(0, 0.3, 1, 0)$. We then change the decision of σ in s_0 to *go*, and the decision

```
Input: MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, target set $T \subseteq S$
Output: $Pr_s^{\min}(reach(T))$ for all $s \in S$
1 Pick arbitrary adversary $\sigma$;
2 do
3     Compute $p_s := Pr_s^{\sigma}(reach(T))$ for all $s \in S$;
4     foreach $s \in S$ do $\sigma(s) := \arg\min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot p_{s'}$
5 while $\sigma$ has changed;
6 return $(p_s)_{s \in S}$
```

Algorithm 7: Policy iteration for $Pr_s^{\min}(reach(T))$

in s_1 to *risk*. Recomputing the values p_s, we get $(0.5, 0.5, 1, 0)$ and subsequently change the decision of σ in s_1 back to *safe* and in s_3 to *reset*. Computing the values of p_s then yields $(1, 1, 1, 1)$, which cannot be further improved, so these probabilities are returned. ■

Method Comparison. To give an idea of the relative performance of the computation methods described in this section, we provide a small set of results, using models from the PRISM benchmark suite [89]. Table 1 shows results for 8 model checking problems on 6 different models. The models, and associated parameters, are: *consensus* (N=4, K=16), *firewire_dl* (*delay*=36, *deadline*=800), *csma* (N=3, K=4), *wlan* (*BOFF*=5, *COL*=6), *zeroconf* (N=1000, K=8, *reset*=f), *zeroconf_dl* (N=1000, K=2, *reset*=f, *bound*=30); see [89] for further details.

We show the model sizes (number of states) and, for each of three methods (value iteration, Gauss-Seidel, policy iteration), the number of iterations needed and the total solution time (in seconds), running on a 2.53 GHz machine with 8 GB RAM. For policy iteration, we use Gauss-Seidel to analyse each adversary and, for all iterative methods, we terminate when the maximum absolute difference is below ε=10^{-6}. Results are omitted for linear programming since this approach does not scale to these model sizes. We see that Gauss-Seidel is always faster than standard value iteration and often gives a significant speed-up, thanks to its faster convergence. It is likely that further gains could be made by re-ordering the state space. Policy iteration, in most cases, needs to examine a relatively small number of adversaries. However, it does not (on this benchmark set, at least) offer any improvement over Gauss-Seidel value iteration and on some models can be considerably slower than standard value iteration.

4.3 Adversary Generation

As stated earlier in this section, for MDP $\mathcal{M}$ and target set T, there are always *deterministic* and *memoryless* adversaries, say $\sigma^{\min}$ and $\sigma^{\max}$, exhibiting the minimum and maximum probabilities of reaching T, respectively. So far, we have described how to compute the values of these probability bounds. Here we discuss how to generate adversaries $\sigma^{\min}$ and $\sigma^{\max}$ that achieve these bounds.

Model	Property	Size	Value iter.		Gauss-Seidel		Policy iteration		
		(states)	Iter.s	Time	Iter.s	Time	Adv.s	Iter.s	Time
consensus	*c2*	166,016	70,681	368.8	35,403	**190.1**	6	35,687	196.1
consensus	*disagree*	166,016	90,861	637.6	45,432	**336.5**	52	83,775	533.3
firewire_dl	*deadline*	530,965	621	5.74	614	**5.67**	1	614	5.69
csma	*all_before*	1,460,287	118	3.53	87	**2.44**	2	161	5.68
csma	*some_before*	1,460,287	59	0.25	45	**0.14**	2	76	0.58
wlan	*collisions*	1,591,710	825	2.34	323	**0.97**	4	788	2.58
zeroconf	*correct*	1,870,338	345	16.6	259	**14.4**	4	581	24.8
zeroconf_dl	*deadline*	666,378	122	1.09	81	**0.76**	18	758	6.86

Table 1. Comparison of the methods for computing reachability probabilities

As for policy iteration discussed earlier, since the adversaries are deterministic and memoryless, we can describe them as a mapping from states to actions.

For the case of minimum probabilities, this is straightforward. Regardless of the method used for computation of the minimum probabilities $Pr_s^{\min}(reach(T))$, we define, for each state $s \in S$:

$$\sigma^{\min}(s) \overset{\text{def}}{=} \arg\min_{a \in A(s)} \left(\sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot Pr_{s'}^{\min}(reach(T)) \right).$$

For the case of maximum probabilities, more care is required [1]. There are several solutions, depending on the probability computation method used. If policy iteration was applied, for example, the process is trivial since adversaries are explicitly constructed and solved during the algorithm's execution. We will now demonstrate how to adapt the value iteration algorithm to compute an optimal adversary as it proceeds. The idea is essentially the same as for minimum probabilities above, but we perform this at every step of the computation and, crucially, only update the adversary for a state s if the probability is strictly better in that state. The adapted algorithm is shown in Algorithm 8. An additional caveat of this version of the algorithm is that we skip the (optional) computation of $S^1_{\max}$ in order to correctly determine optimal choices for those states. For the states in $S^0_{\max}$, by definition the reachability probability is 0 for all adversaries, and hence we can choose arbitrary actions in these states.

5 Reward-based Properties

Reward structures are a powerful mechanism for reasoning about various quantitative properties of models. As mentioned earlier, reward structures can be used to model system characteristics such as power consumption, the cost or time to execute a task and the size of a queue. In this tutorial, we discuss *expected reward* properties, focussing on two particular classes: *instantaneous reward* and *cumulative reward*. Instantaneous reward allows reasoning about the expected reward associated with the state of an MDP after a particular number of steps. Cumulative reward is useful in situations where we are interested in the sum of rewards

```
   Input: MDP $\mathcal{M}$, target $T$, set $S^0_{\max}$ and convergence criterion $\varepsilon$
   Output: (approximation of) $Pr^{\max}_s(reach(T))$ for all $s \in S$, optimal adversary
 1 foreach $s \in S\backslash(S^0_{\max} \cup T)$ do
 2     $x_s := \begin{cases} 1 \text{ if } s \in T \\ 0 \text{ otherwise} \end{cases}$;
 3     $\sigma^{\max}(s) := \perp$;
 4 end
 5 do
 6     foreach $s \in S\backslash(S^0_{\max} \cup T)$ do
 7         $x'_s := \max_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}$;
 8         if $\sigma^{\max}(s) = \perp$ or $x'_s > x_s$ then
 9             $\sigma^{\max}(s) := \arg\max_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}$';
10         end
11         $\delta := \max_{s \in S} (x'_s - x_s)$;
12         foreach $s \in S\backslash(S^0_{\max} \cup T)$ do  $x_s := x'_s$;
13     end
14 while $\delta > \varepsilon$;
15 return $(x'_s)_{s \in S}, \sigma^{\max}$
```

Algorithm 8: Value iteration/adversary generation for $Pr^{\max}_s(reach(T))$

accumulated up to a certain point. There are also various other reward-based properties for MDPs, of which two of the most prominent are:

– *Discounted reward*, in which reward is accumulated step by step, but the reward gained in the nth step is multiplied by a factor λ^n for some $\lambda < 1$, thus giving preference to rewards gained earlier in time;
– *Expected long-run average reward*, in which the average reward gained per state or transition is considered.

The reader is refered to [76] for a more comprehensive review of these and other such properties.

5.1 Instantaneous Rewards

One of the simplest MDP reward properties is *instantaneous reward*, which is defined as the expected reward of the state entered after k steps, for a given reward structure and step $k \in \mathbb{N}$. For example, if the MDP models a system equipped with a queue and the reward structure assigns the current queue size to each state, then instantaneous reward properties allow us to formalise questions such as "what is the maximum expected size of the queue after 200 steps?".

Formally, given an MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, state reward function r_{state} for $\mathcal{M}$, state $s \in S$ and step k, we define a random variable $I^{=k}_{r_{state}} : IPath_s \to \mathbb{R}_{\geqslant 0}$ such that $I^{=k}_{r_{state}}(\pi) \overset{\text{def}}{=} r_{state}(\pi(k))$ for all infinite paths $\pi \in IPath_s$. The value of interest is then either $\mathbb{E}^{\min}_s(I^{=k}_{r_{state}})$ or $\mathbb{E}^{\max}_s(I^{=k}_{r_{state}})$, i.e. either the minimum or maximum expected reward at step k when starting from state s.

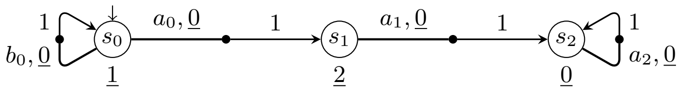


Fig. 6. An example MDP for which optimal memoryless adversaries do not exist

It is easy to see that memoryless adversaries do not suffice for minimising or maximising instantaneous reward. For example, consider the MDP from Figure 6 and the value of $\mathbb{E}^{\max}_{s_0}(I^{=3}_{r_{state}})$. The optimal behaviour is to take self-loop b_0 twice and then the action a_0, which yields expected instantaneous reward of 2, which no memoryless adversary can achieve. Intuitively, the adversary may need to "wait" in some states until the time comes to take a step towards states in which the reward is large (or small if we consider the minimising case).

The values $\mathbb{E}^{\min}_{s}(I^{=k}_{r_{state}})$ are computed through the following equations, which exploit the relation between instantaneous reward in the ℓth and $(\ell{-}1)$th steps:

$$x^\ell_s = \begin{cases} r_{state}(s) & \text{if } \ell{=}0 \\ \min_{a\in A(s)} \big(\sum_{s'\in S} \delta_{\mathcal{M}}(s,a)(s')\cdot x^{\ell-1}_{s'} \big) & \text{otherwise} \end{cases}$$

We set $\mathbb{E}^{\min}_{s}(I^{=k}_{r_{state}})$ equal to x^k_s. It is also straightforward to extend this to compute an optimal adversary $\sigma^{\min}$ on-the-fly by remembering the action:

$$a^\ell_s = \arg\min_{a\in A(s)} \big(\textstyle\sum_{s'\in S} \delta_{\mathcal{M}}(s,a)(s')\cdot x^{\ell-1}_{s'} \big)$$

for all $1{\leqslant}\ell{\leqslant}k$ and $s \in S$, and setting $\sigma^{\min}(\rho) = [a_\rho{\mapsto}1]$ where $a_\rho = a^{k-|\rho|}_{last(\rho)}$ for all $\rho \in \mathit{FPath}$ such that $|\rho|{\leqslant}k{-}1$. The choices for paths longer than $k{-}1$ can be arbitrary as these do not influence the expected reward.

The equations for computing $\mathbb{E}^{\max}_{s}(I^{=k}_{r_{state}})$ and $\sigma^{\max}$ can be obtained by replacing "min" with "max" in those above for $\mathbb{E}^{\min}_{s}(I^{=k}_{r_{state}})$.

Example 8. Let us compute the maximum instantaneous reward after 4 steps in the MDP from Figure 2. This amounts to finding the solution to the equations:

$$\begin{aligned}
x^0_{s_0} &= 1 \\
x^0_{s_1} &= 2 \\
x^0_{s_2} &= 0 \\
x^0_{s_3} &= 0 \\
x^\ell_{s_0} &= x^{\ell-1}_{s_1} \\
x^\ell_{s_1} &= \max\{0.7\cdot x^{\ell-1}_{s_0} + 0.3\cdot x^{\ell-1}_{s_2}, 0.5\cdot x^{\ell-1}_{s_2} + 0.5\cdot x^{\ell-1}_{s_3}\} \\
x^\ell_{s_2} &= x^{\ell-1}_{s_2} \\
x^\ell_{s_3} &= \max\{x^{\ell-1}_{s_3}, x^{\ell-1}_{s_0}\}
\end{aligned}$$

for $1{\leqslant}\ell{\leqslant}4$. The following are the values $\mathbf{x}^i = (x^i_{s_0}, x^i_{s_1}, x^i_{s_2}, x^i_{s_3})$:

$$\begin{aligned}
\mathbf{x}^1 &= (\ \ 2, 0.7, 0, 1) \\
\mathbf{x}^2 &= (\ 0.7, 1.4, 0, 2) \\
\mathbf{x}^3 &= (\ 1.4, \ \ 1, 0, 2) \\
\mathbf{x}^4 &= (\ \ 1, \ \ 1, 0, 2)
\end{aligned}$$

So, e.g., $\mathbb{E}^{\max}_{s_0}(I^{=4}_{r_{state}}) = 1$. The associated optimal adversary $\sigma^{\max}$ satisfies:

- if $last(\rho){=}s_1$, then $\sigma^{\max}(\rho){=}[risk{\mapsto}1]$ when $|\rho|{\leqslant}1$ and $[safe{\mapsto}1]$ otherwise;
- if $last(\rho){=}s_3$, then $\sigma^{\max}(\rho){=}[stop{\mapsto}1]$ when $|\rho|{\leqslant}1$ and $[reset{\mapsto}1]$ otherwise. ■

5.2 Step-bounded Cumulative Reward

Rather than computing the expected reward gained *at* the kth step, it may be useful to compute the expected reward *accumulated up to* the kth step. Formally, given an MDP $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, reward structure $r{=}(r_{state}, r_{action})$, state s and step bound k, we define the random variable $C_r^{\leqslant k} : IPath_s \to \mathbb{R}_{\geqslant 0}$ where:

$$C_r^{\leqslant k}(\pi) \stackrel{\text{def}}{=} \textstyle\sum_{i=0}^{k-1} \big(r_{state}(s_i) + r_{action}(s_i, a_i) \big)$$

for all infinite paths $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \in IPath_s$ and consider either $\mathbb{E}_s^{\min}(C_r^{\leqslant k})$ or $\mathbb{E}_s^{\max}(C_r^{\leqslant k})$. For example, if the MDP models a system in which energy is consumed at each step, and the reward structure assigns energy values to actions, then step-bounded cumulative reward can be used to reason about properties such as "the expected energy consumption of the system within the first 1000 time-units of operation".

As for the previous case, there may not exist a memoryless optimal adversary. For example, consider $\mathbb{E}_{s_0}^{\min}(C_r^{\leqslant 3})$ for the MDP from Figure 6. The optimal behaviour is to take the self-loop b_0 once and then the actions a_0 and a_1, which yields cumulated reward 5. This is not achievable by any memoryless adversary.

The value of $\mathbb{E}_s^{\min}(C_r^{\leqslant k})$ is computed in a similar way to instantaneous rewards through the following equations:

$$x_s^\ell = \begin{cases} 0 & \text{if } \ell{=}0 \\ r_{state}(s) + \min_{a \in A(s)} \big(r_{action}(s,a) + \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}^{\ell-1} \big) & \text{otherwise} \end{cases}$$

Like for the instantaneous case, an optimal adversary can be constructed on-the-fly by remembering the action:

$$a_s^\ell = \arg\min_{a \in A(s)} \big(r_{action}(s,a) + \textstyle\sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}^{\ell-1} \big)$$

for all $1{\leqslant}\ell{\leqslant}k$ and $s \in S$, and setting $\sigma(\rho) = [a_\rho{\mapsto}1]$ where $a_\rho = a_{last(\rho)}^{k-|\rho|}$ for all $\rho \in FPath$ such that $|\rho|{\leqslant}k{-}1$. The choices for paths longer than $k{-}1$ can be arbitrary as these do not influence the expected reward.

When considering maximum rewards, the corresponding equations can be obtained by replacing "min" with "max" in those above.

Example 9. Let us compute the maximum expected reward accumulated within 4 steps in the MDP from Figure 2. In order to do this, we need to solve the following equations (cf. Example 8):

$$\begin{aligned}
x_{s_0}^0 &= 0 \\
x_{s_1}^0 &= 0 \\
x_{s_2}^0 &= 0 \\
x_{s_3}^0 &= 0 \\
x_{s_0}^\ell &= 1 + 1 + x_{s_1}^{\ell-1} \\
x_{s_1}^\ell &= 2 + \max\{1 + 0.7 \cdot x_{s_0}^{\ell-1} + 0.3 \cdot x_{s_2}^{\ell-1}, 4 + 0.5 \cdot x_{s_2}^{\ell-1} + 0.5 \cdot x_{s_3}^{\ell-1}\} \\
x_{s_2}^\ell &= x_{s_2}^{\ell-1} \\
x_{s_3}^\ell &= \max\{x_{s_3}^{\ell-1}, 5 + x_{s_0}^{\ell-1}\}
\end{aligned}$$

for $1 \leqslant \ell \leqslant 4$. The following are the values $\mathbf{x}^i = (x^i_{s_0}, x^i_{s_1}, x^i_{s_2}, x^i_{s_3})$:

$$\begin{aligned}
\mathbf{x}^1 &= (2, \; 6, 0, \; 5)\\
\mathbf{x}^2 &= (8, \; 8.5, 0, \; 7)\\
\mathbf{x}^3 &= (10.5, \; 9.5, 0, \; 13)\\
\mathbf{x}^4 &= (11.5, 12.5, 0, 15.5)
\end{aligned}$$

So, e.g., $\mathbb{E}^{\max}_{s_0}(C^{\leqslant 4}_r) = 11.5$. The computed optimal adversary $\sigma^{\max}$ is in fact memoryless and satisfies $\sigma^{\max}(s_1) = [\mathit{risk} \mapsto 1]$ and $\sigma^{\max}(s_3) = [\mathit{reset} \mapsto 1]$. ■

5.3 Cumulative Reward to Reach a Target

Sometimes it is more useful to consider the cumulative reward gained before some set of target states is reached, rather than within a time bound. This could be used, for example, to compute "the expected cost of completing a task" or "the total expected energy consumption during a system's lifetime".

Let $\mathcal{M}=(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ be an MDP, $r=(r_{state}, r_{action})$ a reward structure, $s \in S$ and $T \subseteq S$ a set of target states. We aim to compute $\mathbb{E}^{\min}_s(F^T_r)$ or $\mathbb{E}^{\max}_s(F^T_r)$ where $F^T_r : \mathit{IPath}_s \to \mathbb{R}_{\geqslant 0}$ is the random variable such that, for any path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \in \mathit{IPath}_s$:

$$F^T_r(\pi) \stackrel{\text{def}}{=} \begin{cases} \infty & \text{if } s_i \notin T \text{ for all } i \in \mathbb{N} \\ \sum_{i=0}^{k^T_\pi - 1} \left(r_{state}(s_i) + r_{action}(s_i, a_i) \right) & \text{otherwise} \end{cases}$$

where $k^T_\pi = \min\{k \mid s_k \in T\}$. As for probabilistic reachability, there are always *deterministic* and *memoryless* adversaries exhibiting the minimum and maximum expected cumulative reward of reaching a target T.

Let us first consider the case $\mathbb{E}^{\min}_s(F^T_r)$. By definition, the value of $\mathbb{E}^{\min}_s(F^T_r)$ is infinite if and only if, for all adversaries, when starting in state s, the probability of reaching T is strictly less than 1. Using the methods presented in Section 4, we can compute $S^1_{\max}$, i.e. the set of states for which the probability of reaching T equals 1 for some adversary. Hence, $\mathbb{E}^{\min}_s(F^T_r)$ is infinite if and only if $s \notin S^1_{\max}$, and it remains to compute the values for the states in $s \in S^1_{\max}$. Let $C^T_r : \mathit{IPath}_s \to \mathbb{R}_{\geqslant 0}$ be the random variable such that for any path $\pi = s_0 \xrightarrow{a_0} s_1 \xrightarrow{a_1} \cdots \in \mathit{IPath}_s$:

$$C^T_r(\pi) \stackrel{\text{def}}{=} \sum_{i=0}^{l^T_\pi - 1} \left(r_{state}(s_i) + r_{action}(s_i, a_i) \right)$$

where $l^T_\pi = \infty$ if $s_i \notin T$ for all $i \in \mathbb{N}$ and equals $\min\{k \mid s_k \in T\}$ otherwise. Now, using [1,17], if there exists a *proper* adversary that achieves the minimum expected value of C^T_r, i.e. an adversary σ such that:

$$\mathit{Pr}^\sigma_s(\mathit{reach}(T)) = 1 \text{ and } \mathbb{E}^\sigma_s(C^T_r) = \mathbb{E}^{\min}_s(C^T_r) \quad \text{for all } s \in S^1_{\max},$$

then the values $\mathbb{E}^{\min}_s(F^T_r)$ are the unique solution to the following equations:

$$x_s = \begin{cases} 0 & \text{if } s \in T \\ r_{state}(s) + \min_{a \in A(s)} \left(r_{action}(s, a) + \sum_{s' \in S} \delta_{\mathcal{M}}(s, a)(s') \cdot x_{s'} \right) & \text{otherwise.} \end{cases}$$

As for probabilistic reachability in Section 4, techniques such as linear programming, value iteration or policy iteration can be used to compute the value. However, there still remains the problem of checking for the existence of such a proper adversary or dealing with the case when no such proper adversary exists. The solution is to use the techniques presented in [1], and perform a transformation of the MDP, by essentially removing end components with only zero rewards, to guarantee that such a proper adversary exists.

For the maximum case, by definition, the value $\mathbb{E}_s^{\max}(F_r^T)$ is infinite if and only if there is an adversary under which T is reached from s with probability strictly less than 1. Again, using the methods presented in Section 4, we can compute $S^1_{\min}$, i.e. the set of states for which no such adversary exists. Hence, $S^1_{\min}$ identifies precisely those states for which the maximum expected reward is finite. For such states s, the values $\mathbb{E}_s^{\max}(F_r^T)$ satisfy the follow equations:

$$x_s = \begin{cases} 0 & \text{if } s \in T \\ r_{state}(s) + \max_{a \in A(s)} \left(r_{action}(s,a) + \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'} \right) & \text{otherwise} \end{cases}$$

and are in this case the least solution [1]. We can again use techniques such as those described in Section 4 to find this solution.

Example 10. Let us consider the MDP from Figure 5, and compute the minimum expected reward to reach $\{s_2, s_3\}$. Since $S^1_{\max}{=}S$, we obtain the equations:

$$\begin{aligned} x_{s_2} &= 0 \\ x_{s_3} &= 0 \\ x_{s_0} &= 1 + 1 + x_{s_1} \\ x_{s_1} &= 2 + \min\{4 + 0.5{\cdot}x_{s_2} + 0.5{\cdot}x_{s_3}, 1 + 0.3{\cdot}x_{s_2} + 0.1{\cdot}x_{s_3} + 0.6{\cdot}x_{s_0}\} \end{aligned}$$

The unique solution is the vector $(8, 6, 0, 0)$ and, e.g., $\mathbb{E}_{s_0}^{\min}(F_r^{\{s_2,s_3\}}) = 8$. ■

A careful reader may have noticed that both properties considered earlier in this section, i.e. instantaneous reward at the kth step and cumulative reward up to the kth step, can be reduced to the cumulative reward to reach a target set. More precisely, for an MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ and reward structure $r{=}(r_{state}, r_{action})$, we can construct an MDP $\mathcal{M}' = (S', \overline{s}', \delta_{\mathcal{M}'}, L')$, reward structure $r'{=}(r'_{state}, r'_{action})$ and target T' where in both cases:

- $S' = S {\times} \{0, \dots, k{+}1\}$ and $\overline{s}' = (\overline{s}, 0)$;
- for any $s, s' \in S$, $a \in \alpha_{\mathcal{M}}$ and $i {\leqslant} k$ we have $\delta_{\mathcal{M}'}((s,i),a)((s',i{+}1)) = \delta_{\mathcal{M}}(s,a)(s')$ and $\delta_{\mathcal{M}'}((s,k{+}1),a)((s',k{+}1)) = \delta_{\mathcal{M}}(s,a)(s')$.

In the instantaneous case, $T' = S {\times} \{k{+}1\}$ and, for any $s \in S$, $0 {\leqslant} i {\leqslant} k{+}1$ and $a \in \alpha_{\mathcal{M}}$, we set $r'_{state}(s,i)$ to $r_{state}(s)$ if $i{=}k$ and to 0 otherwise, and $r'_{action}((s,i),a){=}0$. We then have that $\mathbb{E}_{(s,0)}^{\min}(F_{r'}^{T'})$ (respectively $\mathbb{E}_{(s,0)}^{\max}(F_{r'}^{T'})$) in $\mathcal{M}'$ equals $\mathbb{E}_s^{\min}(I_{r_{state}}^{=k})$ (respectively $\mathbb{E}_s^{\max}(I_{r_{state}}^{=k})$) in $\mathcal{M}$.

For cumulative rewards, $T' = S {\times} \{k\}$, $r'_{state}(s,i){=}r_{state}(s)$ for all $s \in S$ and $0 {\leqslant} i {\leqslant} k{+}1$, and $r'_{action}((s,i),a){=}r_{action}(s,a)$ for all $s \in S$, $1 {\leqslant} i {\leqslant} k{+}1$, and $a \in \alpha_{\mathcal{M}}$. In this case, we have that $\mathbb{E}_{(s,0)}^{\min}(F_{r'}^{T'})$ (respectively $\mathbb{E}_{(s,0)}^{\max}(F_{r'}^{T'})$) in $\mathcal{M}'$ equals $\mathbb{E}_s^{\min}(C_r^{\leqslant k})$ (respectively $\mathbb{E}_s^{\max}(C_r^{\leqslant k})$) in $\mathcal{M}$.

6 PCTL Model Checking

In this section, we discuss the use of temporal logic to express and verify more complex properties of systems than just reachability or reward-based properties. We then show how the techniques introduced in Sections 4 and 5 can be used to perform model checking of these properties.

6.1 The Logic PCTL

PCTL (Probabilistic Computational Tree Logic) [54,18] is a probabilistic extension of the temporal logic CTL [33]. PCTL is used to express properties of both DTMCs [54] and MDPs [18]. Here, we focus on MDPs. In the last part of this section, we will extend PCTL to reward-based properties and, in Section 7, we will discuss the alternative temporal logic LTL (linear temporal logic).

Definition 12 (PCTL syntax). *The syntax of PCTL is as follows:*

$$\begin{aligned}\phi &::= \texttt{true} \;\big|\; c \;\big|\; \phi \wedge \phi \;\big|\; \neg\phi \;\big|\; \texttt{P}_{\bowtie p}[\psi] \\ \psi &::= \texttt{X}\,\phi \;\big|\; \phi\ \texttt{U}^{\leqslant k}\ \phi \;\big|\; \phi\ \texttt{U}\ \phi\end{aligned}$$

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

PCTL formulas are interpreted over an MDP and we assume that the atomic propositions c are taken from the set AP used to label its states.

In the syntax above, we distinguish between state formulas ϕ and path formulas ψ, which are evaluated over states and paths, respectively. A property of a model will always be expressed as a state formula; path formulas only occur as the parameter of the *probabilistic path operator* $\texttt{P}_{\bowtie p}[\psi]$. Intuitively, a state s satisfies $\texttt{P}_{\bowtie p}[\psi]$ if, under any adversary, the probability of taking a path from s satisfying path formula ψ is in the interval specified by $\bowtie p$.

As path formulas, we allow the $\texttt{X}$ (*next*), $\texttt{U}^{\leqslant k}$ (*bounded until*) and $\texttt{U}$ (*until*) operators, which are standard in temporal logics. Intuitively: $\texttt{X}\,\phi$ is true if ϕ is satisfied in the next state; $\phi_1\ \texttt{U}^{\leqslant k}\ \phi_2$ is true if ϕ_2 is satisfied within k time-steps and ϕ_1 holds up until that point; and $\phi_1\ \texttt{U}\ \phi_2$ is true if ϕ_2 is satisfied at some point in the future and ϕ_1 holds up until then.

Semantics. To give the semantics of PCTL, we must first specify a class of adversaries Adv. More precisely, the satisfaction relation is parameterised by Adv and a PCTL formula is satisfied in a state s if it is satisfied under *all* adversaries $\sigma \in Adv$. In practice, Adv is usually taken to be the set $Adv_{\mathcal{M}}$ of all adversaries. The formal semantics of PCTL is as follows.

Definition 13 (PCTL semantics). *Let $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ be an MDP, Adv a class of adversaries of $\mathcal{M}$ and $s \in S$. The satisfaction relation $\models_{Adv}$ of*

PCTL is defined inductively by:

$$\begin{array}{lcl} s \models_{Adv} \texttt{true} & & \textit{always} \\ s \models_{Adv} c & \iff & c \in L(s) \\ s \models_{Adv} \phi_1 \wedge \phi_2 & \iff & s \models_{Adv} \phi_1 \wedge s \models_{Adv} \phi_2 \\ s \models_{Adv} \neg\phi & \iff & s \not\models_{Adv} \phi \\ s \models_{Adv} \mathtt{P}_{\bowtie p}[\psi] & \iff & Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in IPath_{\mathcal{M},s} \mid \pi \models_{Adv} \psi\}) \bowtie p \textit{ for all } \sigma \in Adv \end{array}$$

where, for any $\pi \in IPath_{\mathcal{M}}$*:*

$$\begin{array}{lcl} \pi \models_{Adv} \mathtt{X}\,\phi & \iff & \pi(1) \models_{Adv} \phi \\ \pi \models_{Adv} \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2 & \iff & \exists i \leqslant k\ .\ \big(\pi(i) \models_{Adv} \phi_2 \wedge \pi(j) \models_{Adv} \phi_1\ \forall j < i\big) \\ \pi \models_{Adv} \phi_1\ \mathtt{U}\ \phi_2 & \iff & \exists k \geqslant 0\ .\ \pi \models_{Adv} \phi_1\ \mathtt{U}^{\leqslant k}\ \phi_2\,. \end{array}$$

As for probabilistic reachability in Section 4, it is straightforward to show that the set of paths satisfying any PCTL path formula ψ is measurable [84,11].

With slight abuse of notation, we will use $Pr^{\sigma}_{\mathcal{M},Adv,s}(\psi)$ to denote the probability that a path from s satisfies path formula ψ under adversary σ:

$$Pr^{\sigma}_{\mathcal{M},Adv,s}(\psi) \stackrel{\text{def}}{=} Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in IPath_{\mathcal{M},s} \mid \pi \models_{Adv} \psi\})$$

and define the minimum and maximum probabilities of satisfying the formula under the adversaries Adv for a starting state s:

$$\begin{array}{l} Pr^{\min}_{\mathcal{M},Adv,s}(\psi) \stackrel{\text{def}}{=} \inf_{\sigma \in Adv} Pr^{\sigma}_{\mathcal{M},Adv,s}(\psi) \\ Pr^{\max}_{\mathcal{M},Adv,s}(\psi) \stackrel{\text{def}}{=} \sup_{\sigma \in Adv} Pr^{\sigma}_{\mathcal{M},Adv,s}(\psi). \end{array}$$

Where clear from the context, we will omit the subscripts $\mathcal{M}$ and/or Adv.

Additional Operators. From the basic PCTL syntax, we can derive several other useful operators. Among these are the well known logical equivalences:

$$\begin{array}{rl} \texttt{false} & \equiv \neg\texttt{true} \\ \phi_1 \vee \phi_2 & \equiv \neg(\neg\phi_1 \wedge \neg\phi_2) \\ \phi_1 \rightarrow \phi_2 & \equiv \neg\phi_1 \vee \phi_2 \end{array}$$

We also allow path formulas to contain the $\mathtt{F}$ (*future*) operator (often written as $\Diamond$), which is common in temporal logics. Intuitively, $\mathtt{F}\ \phi$ means that ϕ is eventually satisfied, and its bounded variant $\mathtt{F}^{\leqslant k}\ \phi$ means that ϕ is satisfied within k steps. These can be expressed in terms of the PCTL until and bounded until operators as follows:

$$\mathtt{F}\ \phi \equiv \texttt{true}\ \mathtt{U}\ \phi \quad \text{and} \quad \mathtt{F}^{\leqslant k}\ \phi \equiv \texttt{true}\ \mathtt{U}^{\leqslant k}\ \phi\,.$$

Similarly, we add a $\mathtt{G}$ (*globally*) operator (often written as $\Box$), where $\mathtt{G}\ \phi$ intuitively means that ϕ is always satisfied. It too has a bounded variant $\mathtt{G}^{\leqslant k}\ \phi$, which means that ϕ is continuously satisfied for k steps. These operators can be expressed using the equivalences:

$$\mathtt{G}\ \phi \equiv \neg(\mathtt{F}\ \neg\phi) \quad \text{and} \quad \mathtt{G}^{\leqslant k}\ \phi \equiv \neg(\mathtt{F}^{\leqslant k}\ \neg\phi)\,.$$

Strictly speaking, the $\mathtt{G}$ and $\mathtt{G}^{\leqslant k}$ operators cannot be derived from the basic syntax of PCTL since we do not allow negation in path formulas. However, it can be shown that, for example:

$$\mathtt{P}_{\geqslant p}[\mathtt{G}\ \phi] \equiv \mathtt{P}_{\leqslant 1-p}[\mathtt{F}\ \neg\phi]$$

See Section 7.2 (page 35) for an explanation of the above equivalence.

Examples. Some examples of PCTL formulas, taken from case studies, are:

– $\mathtt{P}_{<0.05}[\mathtt{F}\ (\mathit{sensor\_fail}_1 \wedge \mathit{sensor\_fail}_2)]$ – "the probability of simultaneous failures occurring in both sensors is less than 0.05";
– $\mathtt{P}_{\geqslant 0.8}[\mathtt{F}^{\leqslant k}\ \mathit{ack}_n]$ – "the probability that the sender has received n acknowledgements within k clock-ticks is at least 0.8";
– $\mathtt{P}_{<0.4}[\neg \mathit{fail}_A\ \mathtt{U}\ \mathit{fail}_B]$ – "the probability that component B fails before component A is less than 0.4";
– $\neg\mathit{oper} \rightarrow \mathtt{P}_{\geqslant 1}[\mathtt{F}\ (\mathtt{P}_{>0.99}[\mathtt{G}^{\leqslant 100}\ \mathit{oper}])]$ – "if the system is not operational, it almost surely reaches a state from which it has a greater than 0.99 chance of staying operational for 100 time units".

Extensions of PCTL. A commonly used extension of PCTL, which derives from the PRISM model checker [56], is the addition of *quantitative* versions of the $\mathtt{P}$ operator. Rather than stating that the probability of some path formula ψ being satisfied is always above or below a threshold, *quantitative* properties simply ask: "what is the minimum/maximum probability of ψ holding?". For this, we add the operators $\mathtt{P}_{\mathtt{min}=?}[\psi]$ and $\mathtt{P}_{\mathtt{max}=?}[\psi]$ and, adapting the examples from above, we can express:

– $\mathtt{P}_{\mathtt{min}=?}[\mathtt{F}^{\leqslant k}\ \mathit{ack}_n]$ - "what is the minimum probability that the sender has received n acknowledgements within k clock-ticks?";
– $\mathtt{P}_{\mathtt{max}=?}[\neg \mathit{fail}_A\ \mathtt{U}\ \mathit{fail}_B]$ - "what is the maximum probability that component B fails before component A?".

Of course, these operators cannot be nested within PCTL formulas, like in the fourth example from the earlier list. Thus, in the two examples above, $\mathtt{P}_{\mathtt{min}=?}[\psi]$ or $\mathtt{P}_{\mathtt{max}=?}[\psi]$ is the outermost operator of the formula. As will be seen in the next section, the process of model checking a PCTL formula $\mathtt{P}_{\bowtie p}[\psi]$ requires computation of the probabilities $Pr_s^{\min}(\psi)$ or $Pr_s^{\max}(\psi)$ anyway, so these quantitative properties are no more expensive to analyse.

In addition, when writing specifications for MDPs in PCTL, it may sometimes be useful to consider the *existence* of an adversary that satisfies a particular property, rather than stating that *all* adversaries satisfy it. For simple formulas, this can be done via translation to a dual property. For example, verifying that "there exists an adversary σ for which, from state s, the probability of satisfying ψ is at least p" is equivalent to model checking the PCTL formula $\neg\mathtt{P}_{<p}[\psi]$, which states "it is not the case that under all adversaries the probability of satisfying ψ from state s is less than p". Later, in Section 8, we will discuss the feasibility of checking the existence of adversaries satisfying more complex formulas.

$$\begin{tikzcd}
 & \mathtt{P}_{<0.5}[\mathtt{X}\neg\mathtt{P}_{\geqslant 1}[a\ \mathtt{U}\ (b \wedge c)]] \arrow[d] & & \\
 & \neg\mathtt{P}_{\geqslant 1}[a\ \mathtt{U}\ (b \wedge c)] \arrow[d] & & \\
 & \mathtt{P}_{\geqslant 1}[a\ \mathtt{U}\ (b \wedge c)] \arrow[dl] \arrow[dr] & & \\
a & & b \wedge c \arrow[dl] \arrow[dr] & \\
 & b & & c
\end{tikzcd}$$

Fig. 7. The parse tree for a formula $\mathtt{P}_{<0.5}[\mathtt{X}\,\neg\mathtt{P}_{\geqslant 1}[a\ \mathtt{U}\ (b \wedge c)]]$

6.2 PCTL Model Checking

Model checking a PCTL formula ϕ on an MDP $\mathcal{M}$ amounts to checking which states of $\mathcal{M}$ satisfy ϕ. The basic structure of the algorithm for PCTL model checking [54,18] is similar to the model checking algorithm for the temporal logic CTL [33]. First, we construct a parse tree of the formula ϕ. Each node of the tree is labelled with a subformula of ϕ, the root is labelled with ϕ itself and the leaves are labelled with either `true` or atomic propositions (an example parse tree is depicted in Figure 7). Working upwards towards the root, we recursively compute the set of states satisfying each subformula, and at the end we have determined the set of states satisfying ϕ. For a given class of adversaries *Adv*, let $\mathit{Sat}_{\mathit{Adv}}(\phi)$ denote the set $\{s \in S \mid s \models_{\mathit{Adv}} \phi\}$ of all states satisfying the formula ϕ under the class of adversaries *Adv*. Letting $\triangleright \in \{\geqslant, >\}$ and $\triangleleft \in \{\leqslant, <\}$, the algorithm for PCTL formulas can be summarised as follows:

$$\begin{aligned}
\mathit{Sat}_{\mathit{Adv}}(\mathtt{true}) &= S \\
\mathit{Sat}_{\mathit{Adv}}(c) &= \{s \mid c \in L(s)\} \\
\mathit{Sat}_{\mathit{Adv}}(\neg\, \phi) &= S \setminus \mathit{Sat}_{\mathit{Adv}}(\Phi) \\
\mathit{Sat}_{\mathit{Adv}}(\phi_1 \wedge \phi_2) &= \mathit{Sat}_{\mathit{Adv}}(\phi_1) \cap \mathit{Sat}_{\mathit{Adv}}(\phi_2) \\
\mathit{Sat}_{\mathit{Adv}}(\mathtt{P}_{\triangleright p}[\psi]) &= \{s \in S \mid \mathit{Pr}^{\min}_{\mathit{Adv},s}(\psi) \triangleright p\} \\
\mathit{Sat}_{\mathit{Adv}}(\mathtt{P}_{\triangleleft p}[\psi]) &= \{s \in S \mid \mathit{Pr}^{\max}_{\mathit{Adv},s}(\psi) \triangleleft p\}\,.
\end{aligned}$$

Obviously, the most difficult part in the above algorithm is the computation of the probability bounds $\mathit{Pr}^{\min}_{\mathit{Adv},s}(\psi)$ and $\mathit{Pr}^{\max}_{\mathit{Adv},s}(\psi)$. In what follows, we describe how to compute these probability bounds for the different possible path formulas ψ when *Adv* is the set $\mathit{Adv}_{\mathcal{M}}$ of *all* adversaries of the MDP, and therefore omit *Adv* from the subscript. An alternative would be to consider the class of *fair* adversaries. We do not discuss the issue of *fairness* when model checking MDPs in this tutorial; for details, see e.g. [2,8,12].

The "Next" Operator. If $\psi = \mathtt{X}\,\phi$, then it follows that:

$$\begin{aligned}
\mathit{Pr}^{\min}_{s}(\mathtt{X}\,\phi) &= \min\nolimits_{a \in A(s)} \textstyle\sum_{s' \in \mathit{Sat}(\phi)} \delta_{\mathcal{M}}(s,a)(s') \\
\mathit{Pr}^{\max}_{s}(\mathtt{X}\,\phi) &= \max\nolimits_{a \in A(s)} \textstyle\sum_{s' \in \mathit{Sat}(\phi)} \delta_{\mathcal{M}}(s,a)(s')
\end{aligned}$$

both of which can be computed easily. An optimal memoryless adversary σ can be constructed by putting $\sigma(s) = [a{\mapsto}1]$ where a is an action that minimises (or maximises) $\sum_{s' \in Sat(\phi)} \delta_{\mathcal{M}}(s,a)(s')$.

The "Bounded Until" Operator. Consider the case $\psi = \phi_1 \; \mathtt{U}^{\leqslant k} \; \phi_2$ for minimum probabilities, i.e. computation of $Pr_s^{\min}(\phi_1 \; \mathtt{U}^{\leqslant k} \; \phi_2)$ for all states s. These can be computed by solving the following equations:

$$x_s^\ell = \begin{cases} 1 & \text{if } s \in Sat(\phi_2) \\ 0 & \text{if } s \notin (Sat(\phi_1) \cup Sat(\phi_2)) \\ 0 & \text{if } s \in (Sat(\phi_1) \backslash Sat(\phi_2)) \text{ and } \ell{=}0 \\ \min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}^{\ell-1} & \text{otherwise} \end{cases}$$

and setting $Pr_s^{\min}(\phi_1 \; \mathtt{U}^{\leqslant k} \; \phi_2) = x_s^k$. Like for reward-based properties, an optimal adversary can be constructed on-the-fly by remembering the action:

$$a_s^\ell = \arg\min_{a \in A(s)} \sum_{s' \in S} \delta_{\mathcal{M}}(s,a)(s') \cdot x_{s'}^{\ell-1}$$

for all $0{\leqslant}\ell{\leqslant}k$ and $s \in S$, and setting $\sigma(\rho) = [a_\rho{\mapsto}1]$ where $a_\rho = a_{last(\rho)}^{k-|\rho|}$ for all $\rho \in FPath$ such that $|\rho|{\leqslant}k{-}1$. For maximum probabilities, we simply replace "min" with "max" in the equations above.

The "Unbounded Until" Operator. It remains to give a procedure that computes the minimum and maximum probabilities when $\psi = \phi_1 \; \mathtt{U} \; \phi_2$. The approach is based on the probabilistic reachability computation given in Section 4 for the target $T{=}Sat(\phi_2)$. However, here, we also need to ensure that ϕ_1 holds before satisfying ϕ_2. Thus, we restrict attention to paths that remain in the set of states $Sat(\phi_1)$ before reaching the target. Formally, this can be done by preprocessing the MDP in the following way. For each state $s \in S \backslash Sat(\phi_1)$ and action $a \in A(s)$ we change $\delta_{\mathcal{M}}(s,a)$ to $[s{\mapsto}1]$. It then follows that $Pr_s^{\min}(\phi_1 \; \mathtt{U} \; \phi_2)$ (respectively, $Pr_s^{\max}(\phi_1 \; \mathtt{U} \; \phi_2)$) in the original MDP equals $Pr_s^{\min}(reach(Sat(\phi_2)))$ (respectively, $Pr_s^{\max}(reach(Sat(\phi_2)))$) in the preprocessed MDP. The solution methods such as value iteration or policy iteration presented in Section 4 can therefore be applied, as well as the adversary generation.

Example 11. Consider again the example MDP from Figure 2, together with the formula $\mathtt{P}_{<1}[\mathtt{X}\,(\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init])]$. We start with the analysis of the innermost subformulas and obtain $Sat(fail){=}\{s_3\}$ and $Sat(init){=}\{s_0\}$. Then, we proceed with $\neg fail$ and find $Sat(\neg fail){=}\{s_0, s_1, s_2\}$. Next, considering the formula $\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init]$, we modify the MDP so that in s_3 we loop on action *reset*, and then compute the minimum probability of reaching $\{s_0\}$. Computing $S^0_{\min}$ and $S^1_{\min}$ yields $S^0_{\min}{=}\{s_1, s_2, s_3\}$ and $S^1_{\min}{=}\{s_0\}$, which gives values for all states, and hence $Sat(\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init]) = \{s_0\}$. Finally, for $\mathtt{P}_{<1}[\mathtt{X}\,(\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init])]$, we compute $Pr_s^{\max}(\mathtt{X}\,(\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init]))$ for $s \in \{s_0, s_1, s_2, s_3\}$, and obtain the values $(0, \max\{0.7{\cdot}1 + 0.3{\cdot}0, 0.5{\cdot}0 + 0.5{\cdot}0\}, 0, \max\{0, 1\}) = (0, 0.7, 0, 1)$, yielding $Sat(\mathtt{P}_{<1}[\mathtt{X}\,(\mathtt{P}_{\geqslant 0.5}[\neg fail \; \mathtt{U} \; init])]) = \{s_0, s_1, s_2\}$. ■

6.3 Extending PCTL with Rewards

We can extend the definition of PCTL to include the reward-related properties introduced in Section 5. Here, we present the extension of PCTL used in PRISM. More expressive logics for reward-based properties of MDPs can be found in [1]. The syntax for state formulas of PCTL becomes:

$$\phi ::= \texttt{true} \;\big|\; c \;\big|\; \phi \wedge \phi \;\big|\; \neg\phi \;\big|\; \texttt{P}_{\bowtie p}[\psi] \;\big|\; \texttt{R}^{r}_{\bowtie x}[\texttt{I}^{=k}] \;\big|\; \texttt{R}^{r}_{\bowtie x}[\texttt{C}^{\leqslant k}] \;\big|\; \texttt{R}^{r}_{\bowtie x}[\texttt{F}\ \phi]$$

where c is an atomic proposition, $\bowtie \in \{\leqslant, <, \geqslant, >\}$, $p \in [0,1]$, r is a reward structure, $x \in \mathbb{R}_{\geqslant 0}$ and $k \in \mathbb{N}$. The semantics of the previously introduced operators remains unchanged (see Definition 13), while the semantics of the new operators is defined as follows:

$$\begin{aligned}
s \models_{Adv} \texttt{R}^{r}_{\bowtie x}[\texttt{I}^{=k}] &\iff \mathbb{E}^{\sigma}_{\mathcal{M},s}(I^{=k}_{r_{state}}) \bowtie x \text{ for all } \sigma \in Adv, \text{ where } r{=}(r_{state}, r_{action}) \\
s \models_{Adv} \texttt{R}^{r}_{\bowtie x}[\texttt{C}^{\leqslant k}] &\iff \mathbb{E}^{\sigma}_{\mathcal{M},s}(C^{\leqslant k}_{r}) \bowtie x \text{ for all } \sigma \in Adv \\
s \models_{Adv} \texttt{R}^{r}_{\bowtie x}[\texttt{F}\ \phi] &\iff \mathbb{E}^{\sigma}_{\mathcal{M},s}(F^{Sat_{Adv}(\phi)}_{r}) \bowtie x \text{ for all } \sigma \in Adv\,.
\end{aligned}$$

We can reuse the basic model checking algorithm from above, but we need to extend it to deal with the new operators. Letting $\rhd \in \{\geqslant, >\}$ and $\lhd \in \{\leqslant, <\}$, we have to compute the following sets:

$$\begin{aligned}
Sat_{Adv}(\texttt{R}^{r}_{\rhd x}[\texttt{I}^{=k}]) &= \{s \in S \mid \inf_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(I^{=k}_{r_{state}}) \rhd x\} \\
Sat_{Adv}(\texttt{R}^{r}_{\lhd x}[\texttt{I}^{=k}]) &= \{s \in S \mid \sup_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(I^{=k}_{r_{state}}) \lhd x\} \\
Sat_{Adv}(\texttt{R}^{r}_{\rhd x}[\texttt{C}^{\leqslant k}]) &= \{s \in S \mid \inf_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(C^{\leqslant k}_{r}) \rhd x\} \\
Sat_{Adv}(\texttt{R}^{r}_{\lhd x}[\texttt{C}^{\leqslant k}]) &= \{s \in S \mid \sup_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(C^{\leqslant k}_{r}) \lhd x\} \\
Sat_{Adv}(\texttt{R}^{r}_{\rhd x}[\texttt{F}\ \phi]) &= \{s \in S \mid \inf_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(F^{Sat_{Adv}(\phi)}_{r}) \rhd x\} \\
Sat_{Adv}(\texttt{R}^{r}_{\lhd x}[\texttt{F}\ \phi]) &= \{s \in S \mid \sup_{\sigma \in Adv} \mathbb{E}^{\sigma}_{s}(F^{Sat_{Adv}(\phi)}_{r}) \lhd x\}\,.
\end{aligned}$$

Hence, if Adv is the set of all adversaries of $\mathcal{M}$, then we need to compute the minimum and maximum expected values of the random variables $I^{=k}_{r_{state}}$, $C^{\leqslant k}_{r}$ and $F^{Sat_{Adv}(\phi)}_{r}$, which can be achieved using the algorithms of Section 5. Like for the P operator, we can also consider *quantitative* versions $\texttt{R}^{r}_{\texttt{min=?}}[\cdot]$ and $\texttt{R}^{r}_{\texttt{max=?}}[\cdot]$ of R which ask: "what is the minimum/maximum expected reward?".

6.4 Complexity

The model checking algorithms for PCTL on MDPs [35,18] are polynomial in the size of the model and linear in the size of the formula, where the sizes of the parameters are defined as follows. The size of an MDP $\mathcal{M}$, denoted $|\mathcal{M}|$, equals the total number of nondeterministic choices (since we require $A(s)$ to be non-empty for all states s, it is always the case that this is greater than the number of states). The size of a formula ϕ, denoted $|\phi|$, equals the number of logical connectives and temporal operators in the formula plus the sum of the sizes of the temporal operators.

Due to the recursive nature of the model checking algorithm, we perform model checking for each of the $|\phi|$ operators of ϕ individually. The most expensive cases concern computation of minimum and maximum reachability probabilities or expected cumulative reward to reach a target, for which we must solve a linear optimisation problem of size $|\mathcal{M}|$. Using, for example, the ellipsoid method, this can be performed in polynomial time.

Generally, to simplify the complexity analysis, we will ignore issues regarding number representations, e.g. of probabilities in the MDP or constants in a PCTL formula. If, for example, we took the size of the formula to also include the binary encoding of the numbers k in the bounded until operators, the complexity of model checking algorithm would be exponential in the size of the formula.

7 Linear-time Probabilistic Model Checking

The logic PCTL described in the previous section is a *branching-time* logic. Notice that, when referring to the probability of an event occurring (using the $\mathtt{P}$ operator), only a single temporal operator (such as $\mathtt{U}$, $\mathtt{F}$ or $\mathtt{G}$) can be used. The only way to combine temporal operators is to use a nested formula, such as $\mathtt{P}_{<0.2}[\mathtt{F}\ \mathtt{P}_{\geqslant 0.9}[\mathtt{G}\ a]]$. However, the meaning of such formulas can be subtle as each appearance of the $\mathtt{P}$ operator has a separate quantification over adversaries.

In this section, we consider *linear-time* properties for MDPs, in which the probability of more complex events can be expressed. We will discuss two distinct classes: *probabilistic safety properties* and properties expressed in *linear temporal logic* (LTL), which are in fact a special case of *ω-regular properties*. In each case, computing the required probabilities can be acheived through the use of *automata*: either finite automata, for probabilistic safety properties, or ω-automata such as Rabin automata, for LTL and ω-regular properties.

Another important point to make is that, in this section, we will consider linear-time properties expressed in terms of the *actions* that label the transitions of an MDP, rather than the *atomic propositions* labelling its states, as was done for PCTL in the previous section. In fact, either approach can be taken and the model checking process is very similar. In this presentation, we opt for action-based properties since these are required for the compositional probabilistic model checking techniques that we discuss in Section 9. See, for example, [35,11,32] for details of the state-based approach.

7.1 Probabilistic Safety Properties

To define probabilistic safety properties, we first recall the definitions of *deterministic finite automata* and *regular safety properties.*

Definition 14 (Deterministic finite automaton). *A* deterministic finite automaton *(DFA) is a tuple $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{A}}, \delta_{\mathcal{A}}, F)$, comprising a finite set of states Q, initial state $\overline{q} \in Q$, finite alphabet $\alpha_{\mathcal{A}}$, transition function $\delta_{\mathcal{A}} : Q \times \alpha_{\mathcal{A}} \to Q$ and accepting states $F \subseteq Q$.*

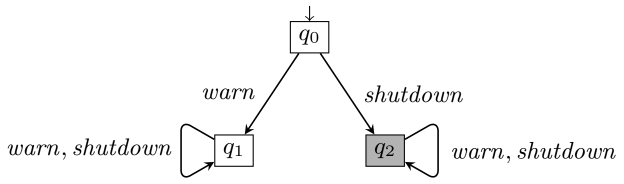


Fig. 8. DFA $\mathcal{A}_A^{err}$ for the regular safety property Φ_A in Example 12

We say that $\mathcal{A}$ is *complete* (or *total*) if the transition function $\delta_{\mathcal{A}}$ is total. A DFA $\mathcal{A}$ defines the regular language $\mathcal{L}(\mathcal{A}) \subseteq (\alpha_{\mathcal{A}})^*$ where $a_0 \dots a_k \in \mathcal{L}(\mathcal{A})$ if and only if $\overline{q} \xrightarrow{a_0} q_1 \xrightarrow{a_1} \cdots \xrightarrow{a_k} q_{k+1}$ is a path of $\mathcal{A}$ (i.e. $\delta_{\mathcal{A}}(\overline{q}, a_0)=q_1$ and $\delta_{\mathcal{A}}(q_i, a_i)=q_{i+1}$ for all $1\leqslant i\leqslant k$) and $q_{k+1} \in F$.

Definition 15 (Regular safety property). *A* regular safety property Φ_P *represents a set of infinite words $\mathcal{L}(\Phi_P) \subseteq (\alpha_P)^\omega$ over an alphabet α_P, which is characterised by a regular language of "bad prefixes", i.e. finite words of which any extension is* not *in $\mathcal{L}(\Phi_P)$. We represent Φ_P by an* error automaton $\mathcal{A}_P^{err}$*: a (complete) DFA over α_P that stores these bad prefixes. Formally:*

$$\mathcal{L}(\Phi_P) \stackrel{\text{def}}{=} \{w \in (\alpha_P)^\omega \mid \textit{no prefix of } w \textit{ is in } \mathcal{L}(\mathcal{A}_P^{err})\} .$$

Regular safety properties can capture properties such as:

- "event A always occurs before event B";
- "a system failure never occurs";
- "termination occurs within at most k steps".

Example 12. Consider the regular safety property Φ_A with the meaning "*warn* occurs before *shutdown*" for a model whose alphabet includes *warn* and *shutdown*. Figure 8 shows the corresponding DFA $\mathcal{A}_A^{err}$ where accepting states are shaded. The "bad prefixes" are the finite words that do not begin with *warn*. ■

To determine the probability that a safety property Φ_P is satisfied, we look at the set of paths whose traces, when restricted to α_P, are in $\mathcal{L}(\Phi_P)$. Consider an MDP $\mathcal{M}$ and regular safety property Φ_P with $\alpha_P \subseteq \alpha_{\mathcal{M}}$. For any adversary σ and state s of $\mathcal{M}$, we define the probability, under σ, of Φ_P being satisfied when starting from s as:

$$Pr^{\sigma}_{\mathcal{M},s}(\Phi_P) \stackrel{\text{def}}{=} Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in \mathit{IPath}_{\mathcal{M},s} \mid tr(\pi)\restriction_{\alpha_P} \in \mathcal{L}(\Phi_P) \cup \mathcal{L}^*(\Phi_P)\})$$

where $w\restriction_\alpha$ is the projection of word w onto the alphabet α and $\mathcal{L}^*(\Phi_P) = \{w \in (\alpha_P)^* \mid \text{no prefix of } w \text{ is in } \mathcal{L}(\mathcal{A}_P^{err})\}$. The inclusion of $\mathcal{L}^*(\Phi_P)$ is required because the projection can return finite words. As stated earlier, the set of paths that satisfy Φ_P is always measurable, so the notation is well-defined. As for other classes of property, we also define:

$$Pr^{\min}_{\mathcal{M},s}(\Phi_P) \stackrel{\text{def}}{=} \inf_{\sigma\in Adv_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\Phi_P)$$
$$Pr^{\max}_{\mathcal{M},s}(\Phi_P) \stackrel{\text{def}}{=} \sup_{\sigma\in Adv_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\Phi_P) .$$

We can now introduce, using the probability bound operator $\mathtt{P}_{\geqslant p}[\cdot]$, the class of *probabilistic safety properties.*

Definition 16 (Probabilistic safety property). *A* probabilistic safety property $\mathtt{P}_{\geqslant p}[\Phi_P]$ *comprises a regular safety property Φ_P and a (lower) probability bound $p \in (0,1]$. A state s of an MDP $\mathcal{M}$ satisfies the property, denoted $s \models \mathtt{P}_{\geqslant p}[\Phi_P]$, if the probability of satisfying Φ_P is at least p for all adversaries:*

$$\begin{aligned} s \models \mathtt{P}_{\geqslant p}[\Phi_P] &\iff \forall \sigma {\in} Adv_{\mathcal{M}} \ . \ Pr^{\sigma}_{\mathcal{M},s}(\Phi_P) \geqslant p \\ &\iff Pr^{\min}_{\mathcal{M},s}(\Phi_P) \geqslant p \, . \end{aligned}$$

Probabilistic safety properties can be used to capture a variety of useful properties of MDPs; for example:

– "event A always occurs before event B with probability at least 0.9";
– "the probability of a system failure occurring is at most 0.02";
– "the probability of terminating within k time-units is at least 0.75".

Notice that, in the second example above, we express the property in terms of the *maximum* probability of the safety property *not* holding, rather than the (equivalent) *minimum* probability that it *does* hold. This equivalence is also used when model checking a probabilistic safety property $\mathtt{P}_{\geqslant p}[\Phi_P]$. We reduce the problem of computing the probability $Pr^{\min}_{\mathcal{M},s}(\Phi_P)$ for each state s of an MDP to the problem of computing *maximum* reachability probabilities in the *product* $\mathcal{M} {\otimes} \mathcal{A}^{err}_P$ of the MDP $\mathcal{M}$ and an error automaton $\mathcal{A}^{err}_P$ for Φ_P.

Definition 17 (MDP-DFA product). *If $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L_{\mathcal{M}})$ is an MDP, Φ_P is a regular safety property with $\alpha_P \subseteq \alpha_{\mathcal{M}}$ and $\mathcal{A}^{err}_P{=}(Q, \overline{q}, \alpha_P, \delta_P, F)$ is an error automaton for Φ_P, then the* product MDP*, denoted $\mathcal{M}{\otimes}\mathcal{A}^{err}_P$, is given by $(S{\times}Q, (\overline{s}, \overline{q}), \alpha_{\mathcal{M}}, \delta_{\mathcal{M}\otimes\mathcal{A}^{err}_P}, L_{\mathcal{M}\otimes\mathcal{A}^{err}_P})$ where, for each $(s,q) \in S{\times}Q$ and $a \in \alpha_{\mathcal{M}}$:*

– $\delta_{\mathcal{M}\otimes\mathcal{A}^{err}_P}((s,q),a) = \begin{cases} \delta_{\mathcal{M}}(s,a){\times}[\delta_P(q,a){\mapsto}1] & \text{if } a \in A(s) \cap \alpha_P \\ \delta_{\mathcal{M}}(s,a){\times}[q{\mapsto}1] & \text{if } a \in A(s) \backslash \alpha_P \\ \text{undefined} & \text{otherwise} \end{cases}$

– $L_{\mathcal{M}\otimes\mathcal{A}^{err}_P}(s,q) = \begin{cases} L_{\mathcal{M}}(s) \cup \{err_P\} & \text{if } q \in F \\ L_{\mathcal{M}}(s) & \text{otherwise.} \end{cases}$

Intuitively, the product records both the state of the MDP $\mathcal{M}$ and the state of the DFA $\mathcal{A}^{err}_P$, based on the actions seen so far in the history of $\mathcal{M}$. The labelling is also modified by marking states corresponding to accepting states of $\mathcal{A}^{err}_P$ with the new atomic proposition err_P (we will use this later in Section 9). Crucially, because the automaton is deterministic, each path through $\mathcal{M}{\otimes}\mathcal{A}^{err}_P$ corresponds to a unique path in each of $\mathcal{M}$ and $\mathcal{A}^{err}_P$. Consequently: (i) the probability of events in $\mathcal{M}$ is preserved in $\mathcal{M}{\otimes}\mathcal{A}^{err}_P$; and (ii) each path of $\mathcal{M}{\otimes}\mathcal{A}^{err}_P$ that corresponds to a path of $\mathcal{M}$ that violates Φ_P contains a state in $S{\times}F$.

Proposition 1 ([69]). *If $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L_{\mathcal{M}})$ is an MDP, $s \in S$, Φ_P is a safety property such that $\alpha_P \subseteq \alpha_{\mathcal{M}}$, and $\mathcal{A}^{err}_P$ is an error automaton for Φ_P with accepting states F, then:*

$$Pr^{\min}_{\mathcal{M},s}(\Phi_P) = 1 - Pr^{\max}_{\mathcal{M}\otimes\mathcal{A}^{err}_P,(s,\overline{q})}(reach(S{\times}F)) \, .$$

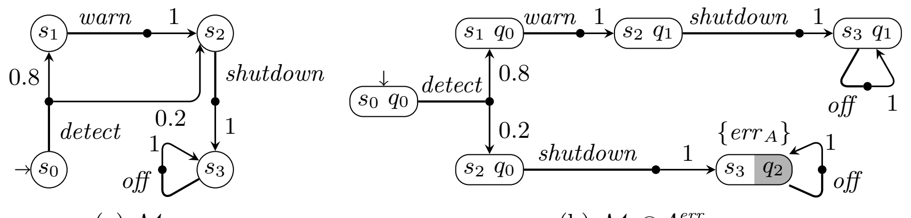


Fig. 9. An MDP $\mathcal{M}_1$ and the product $\mathcal{M}_1 \otimes \mathcal{A}_A^{err}$ with the DFA $\mathcal{A}_A^{err}$ from Figure 8

Thus, checking satisfaction of $\mathtt{P}_{\geqslant p}[\Phi_P]$ on MDP $\mathcal{M}$ can be achieved by applying the techniques of Section 4 to the product MDP $\mathcal{M} \otimes \mathcal{A}_P^{err}$.

Example 13. Consider the MDP $\mathcal{M}_1$ in Figure 9(a) and probabilistic safety property $\mathtt{P}_{\geqslant 0.8}[\Phi_A]$ where Φ_A is the regular safety property "*warn* occurs before *shutdown*" from Example 12, represented by the DFA $\mathcal{A}_A^{err}$ of Figure 8. The product $\mathcal{M}_1 \otimes \mathcal{A}_A^{err}$ is shown in Figure 9(b) and, using the techniques of Section 4, we find that $Pr^{\max}_{\mathcal{M}_1 \otimes \mathcal{A}_A^{err},(s_0,q_0)}(reach(S \times \{q_2\}))=0.2$. Therefore, using Proposition 1, we have $Pr^{\min}_{\mathcal{M}_1,s_0}(\Phi_A)=1-0.2=0.8$, yielding $s_0 \models \mathtt{P}_{\geqslant 0.8}[\Phi_A]$. ■

7.2 LTL and ω-regular Properties

LTL (linear temporal logic) [75] is a widely used temporal logic that is particularly well suited for expressing long-run properties of systems. As discussed above, in this presentation, we use LTL to define properties of MDPs in terms of their action labels. For the remainder of this section, we assume an MDP $\mathcal{M}$ with the alphabet of action labels $\alpha_{\mathcal{M}}$.

Definition 18 (LTL syntax). *The syntax of LTL is defined by the grammar:*

$$\psi ::= \mathtt{true} \mid a \mid \psi \wedge \psi \mid \neg\psi \mid \mathtt{X}\,\psi \mid \psi\ \mathtt{U}\ \psi$$

where $a \in \alpha_{\mathcal{M}}$.

As for PCTL (see Section 6), we can derive additional operators $\vee$, $\rightarrow$, $\mathtt{F}$ and $\mathtt{G}$.

The satisfaction of an LTL formula ψ is given in terms of infinite words over the alphabet $\alpha_{\mathcal{M}}$. To give the semantics, we require some additional notation regarding words. For any infinite word $w = a_0a_1a_2\ldots$, let $w[i]$ denote the $(i+1)$th element a_i of w and let $w[i\ldots]$ denote the suffix $a_ia_{i+1}\ldots$ of w.

Definition 19 (LTL semantics). *Let w be an infinite word over the alphabet $\alpha_{\mathcal{M}}$. The satisfaction relation $\models$ for LTL is defined inductively by:*

$$\begin{array}{lll}
w \models \mathtt{true} & & \textit{always} \\
w \models a & \iff & w[0] = a \\
w \models \psi_1 \wedge \psi_2 & \iff & w \models \psi_1 \wedge w \models \psi_2 \\
w \models \neg\psi & \iff & w \not\models \psi \\
w \models \mathtt{X}\,\psi & \iff & w[1\ldots] \models \psi \\
w \models \psi_1\ \mathtt{U}\ \psi_2 & \iff & \exists i \in \mathbb{N}\ .\ \big(w[i\ldots] \models \psi_2 \wedge (\forall j<i\ .\ w[j\ldots] \models \psi_1) \big)\,.
\end{array}$$

Using this definition, the satisfaction of an LTL formula ψ by an infinite path π of the MDP $\mathcal{M}$ can be defined in terms of the trace of the path as follows:

$$\pi \models \psi \iff tr(\pi) \models \psi .$$

Example 14. Let us consider the following paths from the MDP in Figure 2:

$$\pi_1 = s_0 \left(\xrightarrow{go} s_1 \xrightarrow{risk} s_3 \xrightarrow{reset} s_0 \right)^{\omega}$$
$$\pi_2 = s_0 \xrightarrow{go} s_1 \xrightarrow{risk} s_3 \xrightarrow{reset} s_0 \xrightarrow{go} s_1 \xrightarrow{safe} s_2 \left(\xrightarrow{finish} s_2 \right)^{\omega}$$

The formula $\mathtt{F}\ (\mathit{reset} \wedge (\mathtt{X}\,\mathtt{F}\ \mathit{reset}))$, which intuitively means that *reset* occurs at least twice in a path, is true in π_1, but not true in π_2. On the other hand, the formula $\mathtt{F}\ \mathtt{G}\ \mathit{finish}$, which says that, from some point on, we will only take the action *finish*, is true in π_2, but not in π_1. Other examples are the formula $\mathtt{X}\,\mathtt{X}\,(\neg \mathit{risk}\ \mathtt{U}\ \mathit{finish})$, which is satisfied in π_2, but not in π_1, and the formula $\mathtt{G}\ (\mathit{go} \rightarrow \mathtt{X}\,\mathit{risk})$, which is satisfied in π_1 but not in π_2. ■

It is now straightforward to define the probability of satisfying an LTL formula ψ when starting in a state s under an adversary σ of the MDP $\mathcal{M}$:

$$Pr^{\sigma}_{\mathcal{M},s}(\psi) \overset{\text{def}}{=} Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in \mathit{IPath}_{\mathcal{M},s} \mid \pi \models \psi\}) .$$

The set of paths satisfying an LTL formula ψ is always measurable [84,11]. As usual, we define the minimum and maximum probability of satisfaction over all adversaries of the MDP:

$$Pr^{\min}_{\mathcal{M},s}(\psi) \overset{\text{def}}{=} \inf_{\sigma \in \mathit{Adv}_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\psi)$$
$$Pr^{\max}_{\mathcal{M},s}(\psi) \overset{\text{def}}{=} \sup_{\sigma \in \mathit{Adv}_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\psi).$$

Definition 20 (Probabilistic LTL specification). *A* probabilistic LTL specification *is a formula* $\mathtt{P}_{\bowtie p}[\psi]$ *where* $\bowtie \in \{\leqslant, <, \geqslant, >\}$, $p \in [0,1]$ *and* ψ *is an LTL formula. We say that the probabilistic LTL specification is satisfied in a state* s *of an MDP* $\mathcal{M}$ *if and only if* $Pr^{\sigma}_{\mathcal{M},s}(\psi) \bowtie p$ *for all adversaries* $\sigma \in \mathit{Adv}_{\mathcal{M}}$.

In order to model check an MDP $\mathcal{M}$ and probabilistic LTL specification $\mathtt{P}_{\bowtie p}[\psi]$, we need to compute $Pr^{\min}_{\mathcal{M},s}(\psi)$ if $\bowtie \in \{\geqslant, >\}$ or $Pr^{\max}_{\mathcal{M},s}(\psi)$ if $\bowtie \in \{\leqslant, <\}$. Because every path either satisfies ψ or $\neg\psi$, the problem of computing the minimum probability of satisfying ψ is easily reducible to the computation of the maximum probability of satisfying $\neg\psi$. More precisely, by definition:

$$\begin{aligned}
Pr^{\min}_{\mathcal{M},s}(\psi) &= \inf_{\sigma \in \mathit{Adv}_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\{\pi \mid \pi \models \psi\}) \\
&= \inf_{\sigma \in \mathit{Adv}_{\mathcal{M}}} \left(1 - Pr^{\sigma}_{\mathcal{M},s}(\{\pi \mid \pi \not\models \psi\})\right) && \text{since } Pr^{\sigma}_{\mathcal{M},s} \text{ is a probability measure} \\
&= \inf_{\sigma \in \mathit{Adv}_{\mathcal{M}}} \left(1 - Pr^{\sigma}_{\mathcal{M},s}(\{\pi \mid \pi \models \neg\psi\})\right) && \text{by definition of } \models \\
&= 1 - \sup_{\sigma \in \mathit{Adv}_{\mathcal{M}}} Pr^{\sigma}_{\mathcal{M},s}(\{\pi \mid \pi \models \neg\psi\}) && \text{rearranging} \\
&= 1 - Pr^{\max}_{\mathcal{M},s}(\neg\psi) && \text{by definition.}
\end{aligned}$$

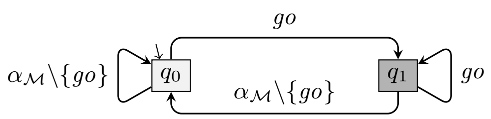

go
$\alpha_{\mathcal{M}}\backslash\{go\}$
q_0
q_1
$\alpha_{\mathcal{M}}\backslash\{go\}$
go

Fig. 10. A DRA for F G go with $Acc = \{(\{q_0\},\{q_1\})\}$ (see Example 15)

Like for probabilistic safety properties in the previous section, LTL model checking is based on the use of automata. However, since the satisfaction of LTL formulas is defined over infinite words, we use ω-automata, rather than finite automata. In particular, as proposed in [1], we use *deterministic Rabin automata*. An alternative, which we do not discuss here, is to use partially determinised Büchi automata [34,35].

Definition 21 (Deterministic Rabin automaton). *A* deterministic Rabin automaton *(DRA) is a tuple $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{A}}, \delta_{\mathcal{A}}, Acc)$ of finitely many states Q, initial state $\overline{q} \in Q$, finite alphabet $\alpha_{\mathcal{A}}$, transition function $\delta_{\mathcal{A}} : Q \times \alpha_{\mathcal{A}} \to Q$ and acceptance condition $Acc = \{(L_i, K_i)\}_{i=1}^{k}$ where $k \in \mathbb{N}$ and $L_i, K_i \subseteq Q$ for $1 \leqslant i \leqslant k$.*

For a DRA $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{A}}, \delta_{\mathcal{A}}, Acc)$, since the transition function is deterministic and total, for any infinite word $w = a_0a_1a_2\dots$ over $\alpha_{\mathcal{A}}$ there is a corresponding unique path $\overline{q} \xrightarrow{a_0} q_1 \xrightarrow{a_1} q_2 \xrightarrow{a_2} \cdots$ of $\mathcal{A}$ (where the path of $\mathcal{A}$ is defined as for DFAs). Using this fact, we say that $\mathcal{A}$ *accepts* an infinite word w if the corresponding path contains *finitely* many states from L_i and *infinitely* many from K_i for some $1 \leqslant i \leqslant k$. The language $\mathcal{L}(\mathcal{A})$ of the DRA $\mathcal{A}$ is given by the set of infinite words that the automaton accepts.

Now, for any LTL formula ψ, we can construct a DRA, say $\mathcal{A}_\psi$, over $\alpha_{\mathcal{M}}$ that accepts precisely the words satisfying ψ, i.e. for any infinite word w:

$$w \models \psi \quad \Longleftrightarrow \quad w \in \mathcal{L}(\mathcal{A}_\psi)$$

The construction of the DRA $\mathcal{A}_\psi$ from the formula ψ is beyond the scope of this tutorial; for details see e.g. [85,36,11].

In fact, the set of properties that can be captured by a DRA is a strict superset of those expressible as LTL formulas, known as *ω-regular properties*. The same class of properties can also be captured with nondeterministic Büchi automata. However, the model checking process for MDPs requires *deterministic* automata and deterministic Büchi automata are not sufficiently expressive (e.g. the LTL formula F G a cannot be represented by a deterministic Büchi automaton).

Example 15. Consider again the running example of Figure 2, together with the DRA $\mathcal{A} = (Q, q_0, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, Acc)$ given in Figure 10, i.e. $Q = \{q_0, q_1\}$, $\delta_{\mathcal{A}}(q, go) = q_1$ and $\delta_{\mathcal{A}}(q, a) = q_0$ for all $q \in Q$ and $a \in \alpha_{\mathcal{M}}\backslash\{go\}$, and Acc is a singleton set containing $(\{q_0\},\{q_1\})$. The property specified by the automaton is "eventually we will only take the action go", which is equivalent to the LTL formula F G go. This holds with probability 0 under all adversaries, since, for any adversary, almost all paths eventually reach and stay in either s_2 or s_3. ■

7.3 Model Checking LTL and ω-regular Properties

We now describe the process of computing the maximum probabilities, in an MDP $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, for an ω-regular property given in the form of a DRA $\mathcal{A}$. As mentioned earlier, this subsumes the problem of computing the probabilities for an LTL formula. This is done by constructing the *product* of $\mathcal{M}$ and $\mathcal{A}$, and then identifying *accepting end components*. To ease presentation, for the remainder of the section we omit the labelling function from MDPs.

Definition 22 (MDP-DRA product). *The* product $\mathcal{M}{\otimes}\mathcal{A}$ *of an MDP* $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$ *and DRA* $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{M}}, \delta_{\mathcal{A}}, \{(L_i, K_i)\}_{i=1}^{k})$ *is given by the MDP* $(S{\times}Q, (\overline{s}, \overline{q}), \alpha_{\mathcal{M}}, \delta_{\mathcal{M}\otimes\mathcal{A}})$ *where, for any* $(s,q) \in S{\times}Q$ *and* $a \in \alpha_{\mathcal{M}}$*:*

– $\delta_{\mathcal{M}\otimes\mathcal{A}}((s,q),a) = \begin{cases} \delta_{\mathcal{M}}(s,a){\times}[\delta_{A}(q,a){\mapsto}1] & \textit{if } a \in A(s) \\ \textit{undefined} & \textit{otherwise.} \end{cases}$

For an MDP $\mathcal{M}$ and DRA $\mathcal{A}$ with acceptance condition $\{(L_i, K_i)\}_{i=1}^{k}$, an *accepting path* of $\mathcal{M}{\otimes}\mathcal{A}$ is an infinite path such that, when projecting its states onto Q, there are *finitely* many states from L_i and *infinitely* many from K_i for some $1{\leqslant}i{\leqslant}k$. Furthermore, an *accepting EC* (recall the definition of end components from Definition 6) of $\mathcal{M}{\otimes}\mathcal{A}$ is an EC (S', δ') for which there exists an $1{\leqslant}i{\leqslant}k$ such that the set of states S', when projected onto Q, contains some state from K_i, but no states from L_i. A key property of the product $\mathcal{M}{\otimes}\mathcal{A}$ is that, for every state s and adversary σ in $\mathcal{M}$, there is an adversary σ' in $\mathcal{M}{\otimes}\mathcal{A}$ such that:

$$\begin{aligned} & Pr^{\sigma}_{\mathcal{M},s}(\{\pi \in \mathit{IPath}_{\mathcal{M},s} \mid tr(\pi) \in \mathcal{L}(\mathcal{A})\}) \\ & \quad = Pr^{\sigma'}_{\mathcal{M}\otimes\mathcal{A},(s,\overline{q})}(\{\pi' \in \mathit{IPath}_{M\otimes\mathcal{A},(s,\overline{q})} \mid \pi' \text{ is an accepting path}\}) \end{aligned}$$

and vice versa. Using this property, together with properties of end components [1], we can further show that the following proposition holds.

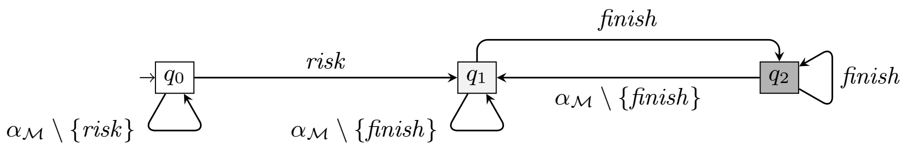
Proposition 2. *For any MDP* $\mathcal{M} = (S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$*, state* $s \in S$ *and DRA* $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{M}}, \delta_{\mathcal{A}}, \{(L_i, K_i)\}_{i=1}^{k})$*, we have:*

$$Pr^{\max}_{\mathcal{M},s}(\{\pi \in \mathit{IPath}_{\mathcal{M},s} \mid tr(\pi) \in \mathcal{L}(\mathcal{A})\}) = Pr^{\max}_{\mathcal{M}\otimes\mathcal{A},(s,\overline{q})}(\mathit{reach}(T))$$

where $(s', q') \in T$ *if and only if* (s', q') *appears in some accepting EC of* $\mathcal{M}{\otimes}\mathcal{A}$*.*

We have therefore reduced the problem of model checking LTL and ω-regular properties to the detection of end components in a product MDP and the computation of maximum reachability probabilities. The latter is covered in Section 4, while [1,11] describes computation of end components. Furthermore, [11] shows how to optimise model checking by considering only *maximal* ECs.

For a given MDP $\mathcal{M}$ and DRA $\mathcal{A} = (Q, \overline{q}, \alpha_{\mathcal{M}}, \delta_{\mathcal{A}}, \{(L_i, K_i)\}_{i=1}^{k})$, suppose that we have obtained (for example through the techniques of Section 4), a memoryless adversary $\sigma^{\max}$ that maximises the probability of reaching a state in an accepting EC of the product. We can then construct a finite-memory adversary for $\mathcal{M}$ that maximises the probability of satisfying the corresponding



q_0
q_1
q_2
risk
finish
finish
$\alpha_{\mathcal{M}} \setminus \{finish\}$
$\alpha_{\mathcal{M}} \setminus \{risk\}$
$\alpha_{\mathcal{M}} \setminus \{finish\}$

Fig. 11. A DRA for (F G *finish*) ∧ (F *risk*) with $Acc = \{(\{q_1\},\{q_2\})\}$ (see Example 16)

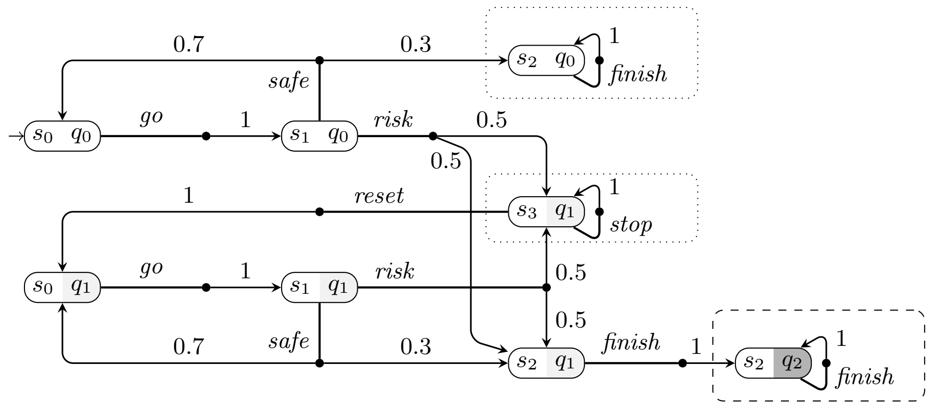


Fig. 12. The product MDP $\mathcal{M}\otimes\mathcal{A}$ for Example 16

ω-regular property. This adversary is specified by the tuple $(Q, \overline{q}, \sigma_u, \sigma_s)$ (see Definition 9) where $\sigma_u(q,a,s)=\delta_{\mathcal{A}}(q,a)$ and $\sigma_s(q,s)=\sigma^{\max}((s,q))$ for all $q \in Q$, $s \in S$ and $a \in \alpha_M$.

Example 16. Let us return to the running example of Figure 2 and compute the maximum probability that $\psi = ($F G *finish*$) \wedge ($F *risk*$)$ is satisfied. Intuitively, this means maximising the probability of eventually reaching and staying in the "finished" state, while at the same time taking a risky decision at least once. A DRA $\mathcal{A}$ that accepts precisely the words satisfying ψ is depicted in Figure 11.

To compute the maximum probability, we first construct the product $\mathcal{M}\otimes\mathcal{A}$. Restricting the MDP to states that are reachable from (s_0, q_0) yields the MDP in Figure 12. It has three end components, which are denoted in the figure as follows: the (single) accepting end component is represented by a dashed box, the remaining two by dotted boxes. From Proposition 2, $Pr^{\max}_{\mathcal{M},s_0}(\psi)$ equals $Pr^{\max}_{\mathcal{M}\otimes\mathcal{A},(s_0,q_0)}(reach(\{(s_2,q_2)\}))$, which we now proceed to compute using the methods from Section 4. First, using Algorithm 3 and Algorithm 4, we find the sets $S^0_{\max}=\{(s_2,q_0)\}$ and $S^1_{\max}=S\backslash S^0_{\max}$. This gives us the probabilities for all states and, since $(s_0,q_0) \in S^1_{\max}$, we have $Pr^{\max}_{\mathcal{M},s_0}(\psi)=1$.

An example optimal adversary σ' in $\mathcal{M}\otimes\mathcal{A}$ is the one that satisfies $\sigma'(\rho) = [reset{\mapsto}1]$ for all ρ ending in (s_3,q_1), and $\sigma'(\rho) = [risk{\mapsto}1]$ for all ρ ending in (s_1,q) for any q. This in fact transfers directly to a memoryless optimal adversary σ in $\mathcal{M}$ satisfying $\sigma(s_3) = [reset{\mapsto}1]$ and $\sigma(s_1) = [risk{\mapsto}1]$. ■

Complexity. We now discuss the complexity of the algorithm above. Starting with LTL formula ψ and MDP $\mathcal{M}$, we first construct a DRA $\mathcal{A}_\psi$. In the worst case, the size $|\mathcal{A}_\psi|$ (number of states) of the smallest DRA $\mathcal{A}_\psi$ can be doubly exponential in $|\psi|$, and the time complexity of the computation is doubly exponential as well. In practice, though, both ψ and $\mathcal{A}_\psi$ are much smaller than $\mathcal{M}$.

After computing $\mathcal{A}_\psi$, we construct the product $\mathcal{M}\otimes\mathcal{A}_\psi$, which can be done in time polynomial in $|\mathcal{M}|$ and $|\mathcal{A}_\psi|$. Then, we identify the end components and compute reachability probabilities, both in time polynomial in the size of the product. Overall, we get that the algorithm runs in time polynomial in $|\mathcal{M}|$ and doubly exponential in $|\psi|$.

8 Multi-objective Probabilistic Model Checking

In this section, we consider *multi-objective* verification techniques for MDPs [40]. These permit the analysis of trade-offs between several linear-time objectives, for example "the probability of reaching a good state is at least 0.98 *and*, with probability at most 0.3, it will be reached in more than 10 steps". These techniques will also be used, in Section 9, to perform compositional verification of MDPs. We begin with the problem of multiple probabilistic reachability objectives, and then describe how to generalise this to multiple ω-regular objectives.

Definition 23 (Multi-objective reachability query). *For an MDP $\mathcal{M}$, target sets $T_1,\dots,T_n \subseteq S$, relational operators $\rhd_1,\dots,\rhd_n \in \{>,\geqslant\}$ and bounds $p_1 \dots, p_n \in [0,1]$, a* multi-objective reachability query *asks whether there exists an adversary $\sigma \in Adv_\mathcal{M}$ such that $Pr^\sigma_{\mathcal{M},\overline{s}}(reach(T_i)) \rhd_i p_i$ for all $1\leqslant i\leqslant n$.*

Observe two key differences between this type of query and the verification problems we have considered so far in this tutorial: firstly, the quantification over adversaries is *existential*, not universal; and secondly, we are concerned only with the probability from the initial state $\overline{s}$ of $\mathcal{M}$.

For technical reasons, we assume that all states in the target sets T_i are absorbing, i.e. the only available transitions are self-loops. This restriction is lifted in the generalised version of the problem that we discuss subsequently. Letting $T = \cup_{i=1}^n T_i$, we also compute the states from which no $s' \in T$ is reachable. These states can be identified by computing the set $S^0_{\max}$ from Section 4. Supposing that $\overline{s} \notin S^0_{\max} \cup T$ (as otherwise the solution is trivial), it can be shown [40] that there exists an adversary σ such that $Pr^\sigma_{\mathcal{M},\overline{s}}(reach(T_i)) \rhd_i p_i$ for all $1\leqslant i\leqslant n$ if and only if the set of inequalities $L(\mathcal{M})$ in Figure 13 has a solution.

Furthermore, it turns out that a solution to the inequalities $L(\mathcal{M})$ yields a *memoryless randomised* adversary σ under which $Pr^\sigma_{\mathcal{M},\overline{s}}(reach(T_i)) \rhd_i p_i$ for all $1\leqslant i\leqslant n$. The adversary σ is defined by $\sigma(s)(a) = y_{(s,a)}/(\sum_{a'\in A(s)} y_{(s,a')})$ whenever the denominator is non-zero, and arbitrarily for all other states s (such states are not reachable from $\overline{s}$ under σ). The intuition behind the solution to $L(\mathcal{M})$ is that, the variables $y_{(s,a)}$ represent the expected number of times, under adversary σ, that a is taken in s, when starting in $\overline{s}$. This is ensured by the equations on the first and last lines of $L(\mathcal{M})$, which intuitively state that the

$$\begin{array}{rcll} in(s) + \sum_{s' \in U} \sum_{a \in A(s')} \delta(s',a)(s) \cdot y_{(s',a)} & = & \sum_{a \in A(s)} y_{(s,a)} & \text{for all } s \in U \\ \sum_{s \in T_i} \sum_{s' \in U} \sum_{a \in A(s')} \delta(s',a)(s) \cdot y_{(s',a)} & \rhd_i & p_i & \text{for all } 1 \leqslant i \leqslant n \\ y_{(s,a)} & \geqslant & 0 & \text{for all } s \in U \text{ and } a \in A(s) \end{array}$$

where $U = S \backslash (T \cup S^0_{\max})$ and $in(s) = 1$ if $s = \overline{s}$ and equals 0 otherwise.

Fig. 13. The linear inequalities $L(\mathcal{M})$ for multi-objective reachability

number of times we enter a state must be equal to the number of times we leave it. The equations on the second line capture the conditions imposed on the probabilities of reaching each target set T_i. Since the target states are absorbing and their outgoing self-loops are omitted from $L(\mathcal{M})$, the expected number of times to enter a target state equals the probability of reaching it.

Multi-objective LTL Queries. We now generalise these multi-objective techniques to the case of ω-regular properties. For simplicity, our presentation will be in terms of LTL formulas.

Definition 24 (Multi-objective LTL query). *A* multi-objective LTL query *is a formula generated by the following grammar:*

$$\theta \ ::= \ \mathtt{P}_{\bowtie p}[\psi] \mid \neg\theta \mid \theta \wedge \theta$$

where ψ is an LTL formula, $\bowtie \in \{<, \leqslant, \geqslant, >\}$, and $p \in [0,1]$.

As before, we allow the use of connectives $\theta_1 \vee \theta_2$ and $\theta_1 \rightarrow \theta_2$ as abbreviations for $\neg(\neg\theta_1 \wedge \neg\theta_2)$ and $\neg\theta_1 \vee \theta_2$, respectively. The semantics of multi-objective LTL queries is defined with respect to both a state s and a specific adversary σ. As above, the verification problem is *existential*: we need to decide, given an MDP $\mathcal{M}$ and multi-objective LTL query θ, whether θ is satisfied in the initial state $\overline{s}$ *for some* adversary σ of $\mathcal{M}$. Formally, the semantics is defined as follows.

Definition 25 (Multi-objective LTL semantics). *Let $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}})$ be an MDP, $s \in S$ a state and σ an adversary of $\mathcal{M}$. The satisfaction of a multi-objective LTL query θ is defined inductively by:*

$$\begin{array}{lcl} \sigma, s \models \mathtt{P}_{\bowtie p}[\psi] & \iff & Pr^{\sigma}_{\mathcal{M},s}(\psi) \bowtie p \\ \sigma, s \models \neg\theta & \iff & \sigma, s \not\models \theta \\ \sigma, s \models \theta_1 \wedge \theta_2 & \iff & \sigma, s \models \theta_1 \text{ and } \sigma, s \models \theta_2 \end{array}$$

The result of the query θ is `true` *in $\mathcal{M}$ if and only if $\sigma, \overline{s} \models \theta$ for some $\sigma \in Adv_{\mathcal{M}}$.*

Note the following difference in the semantics from PCTL (see Section 6) in the way it quantifies over adversaries. In the PCTL semantics, we quantify over adversaries every time we evaluate a formula $\mathtt{P}_{\bowtie p}[\psi]$, while in the case of multi-objective queries we evaluate the whole formula under one fixed adversary.

As the first step towards the solution of multi-objective problems, we employ well-established results of propositional logic and transform a given multi-objective query θ to an equivalent query θ' in disjunctive normal form, i.e. θ' is a disjunction of conjunctions of literals of the form $\mathtt{P}_{\bowtie p}[\psi]$ or $\neg\mathtt{P}_{\bowtie p}[\psi]$. We can further remove negations by replacing $\mathtt{P}_{\bowtie p}[\psi]$ with $\mathtt{P}_{\bar{\bowtie} p}[\psi]$ where $\bar{\bowtie}$ is chosen appropriately, e.g. $\bar{\leqslant}{=}{>}$. Finally, we can ensure that no comparison operators $\leqslant$ or $<$ occur in the formula, by replacing $\mathtt{P}_{<p}[\psi]$ with $\mathtt{P}_{>1-p}[\neg\psi]$, and $\mathtt{P}_{\leqslant p}[\psi]$ with $\mathtt{P}_{\geqslant 1-p}[\neg\psi]$. We obtain a formula which is a disjunction of clauses of the form:

$$\mathtt{P}_{\rhd_1 p_1}[\psi_1] \wedge \ldots \wedge \mathtt{P}_{\rhd_n p_n}[\psi_n]$$

where each $\rhd_i$ is $\geqslant$ or $>$. We can then analyse each clause separately, concluding that the formula is true if and only if at least one clause is true. For this reason, for the remainder of this section, we can assume that the multi-objective query θ is a conjunction of propositions $\mathtt{P}_{\rhd_i p_i}[\psi_i]$ for $\rhd_i \in \{\geqslant, >\}$.

The next step of the solution is similar to standard (single-objective) LTL model checking (see Section 7): we convert each LTL formula ψ_i to a DRA $\mathcal{A}_i$ such that $w \models \psi_i \iff w \in \mathcal{L}(\mathcal{A}_i)$ and then construct the product MDP $\mathcal{M}' = (\cdots((\mathcal{M}\otimes\mathcal{A}_1)\otimes\mathcal{A}_2)\cdots)\otimes\mathcal{A}_n$. Next, for each subset $X \subseteq \{1,\ldots,n\}$ we identify the end components of $\mathcal{M}'$ that are accepting for all $\mathcal{A} \in \{\mathcal{A}_i \,|\, i \in X\}$. As in Section 7, an end component is accepting for $\mathcal{A}$ if its acceptance condition contains some (L_j, K_j) such that the states of the EC, when projected onto the states of $\mathcal{A}$, contain some state from K_j, but no states from L_j.

It can then be shown that the multi-objective LTL query θ is satisfied for some adversary of $\mathcal{M}$ if and only if there exists an adversary of $\mathcal{M}'$ which ensures that, from initial state $(\overline{s}, \overline{q}_1, \ldots, \overline{q}_n)$ of $\mathcal{M}'$, the probability of eventually reaching and staying in end components that are accepting for $\mathcal{A}_i$ satisfies $\rhd_i\ p_i$ for $1{\leqslant}i{\leqslant}n$. To determine whether such an adversary exists, we reduce the problem to a multi-objective reachability problem, which can then be solved via the inequalities presented earlier in Figure 13. The reduction involves the construction, based on $\mathcal{M}' = (S', \overline{s}', \alpha_{\mathcal{M}}, \delta'_{\mathcal{M}})$, of another MDP $\mathcal{M}'' = (S'', \overline{s}'', \alpha''_{\mathcal{M}}, \delta''_{\mathcal{M}})$ where:

- $S'' = S' \cup \{s_X \mid X{\subseteq}\{1,\ldots,n\}\}$ and $\overline{s}''{=}\overline{s}'$;
- $\alpha''_{\mathcal{M}} = \alpha_{\mathcal{M}} \cup \{a_X \mid X \subseteq \{1,\ldots,n\}\}$;
- $\delta''_{\mathcal{M}}$ is created from the probabilistic transition function of $\mathcal{M}'$ by adding transitions $\delta''_{\mathcal{M}}(s, a_X) = [s_X{\mapsto}1]$ for every s and $X \subseteq \{1,\ldots,n\}$ such that s is in an end component that is accepting for all $\mathcal{A} \in \{\mathcal{A}_i \,|\, i \in X\}$.

The following statement captures the correspondence between $\mathcal{M}'$ and $\mathcal{M}''$.

Proposition 3. *For any $1{\leqslant}i{\leqslant}n$, there is an adversary of $\mathcal{M}'$ under which, from $\overline{s}'$, the probability of reaching and staying in end components that are accepting for $\mathcal{A}_i$ satisfies $\rhd_i\ p_i$, if and only if there is an adversary of $\mathcal{M}''$ under which, from $\overline{s}''$, set $T_i{=}\{s_X \mid X \subseteq \{1,\ldots,n\} \wedge i \in X\}$ is reached with probability $\rhd_i\ p_i$.*

Thus, we have a method to check a multi-objective LTL query on an MDP $\mathcal{M}$, via multi-objective reachability on $\mathcal{M}''$. We conclude by describing how, when a satisfying adversary of $\mathcal{M}$ is shown to exist, it can be constructed from the adversary σ'' of $\mathcal{M}''$ obtained via multi-objective reachability. First, we construct the adversary σ' of $\mathcal{M}'$ which follows the decisions of σ'' except that, instead of taking actions a_X, it "switches" its mode and starts mimicking an adversary that stays in end components that are accepting for all $\mathcal{A} \in \{\mathcal{A}_i \mid i \in X\}$ and visits all its states infinitely often. The adversary σ can then be constructed from σ' using the techniques presented in Section 7.

Example 17. Consider the MDP $\mathcal{M}$ of Figure 14(a) and the multi-objective query $\mathtt{P}_{\geqslant 0.6}[\mathtt{F}\ b_1] \wedge \mathtt{P}_{\geqslant 0.3}[\mathtt{G}\ b_2]$. The formula is already a conjunction of propositions and contains only the comparison operator $\geqslant$, so we proceed with construction of the equivalent DRAs $\mathcal{A}_1$ and $\mathcal{A}_2$ for the formulas $\mathtt{F}\ b_1$ and $\mathtt{G}\ b_2$, respectively. The automata are depicted in Figures 14(b)–(c) and the accepting tuples are $Acc_1 = \{(\varnothing, \{q_1\})\}$ and $Acc_2 = \{(\varnothing, \{q_2\})\}$.

We next construct the product MDP $\mathcal{M}'{=}\mathcal{M}{\otimes}\mathcal{A}_1{\otimes}\mathcal{A}_2 = (S', \overline{s}', \alpha_{\mathcal{M}}, \delta'_{\mathcal{M}})$, where $S'{=}S{\times}\{q_0,q_1\}{\times}\{q_2,q_3\}$, $\overline{s}'{=}(t_0,q_0,q_2)$, and the structure of the part of $\mathcal{M}'$ reachable from $\overline{s}'$ is exactly the same as the structure of $\mathcal{M}$, except that t_0, t_1 and t_2 are replaced with (t_0,q_0,q_2), (t_1,q_1,q_3) and (t_2,q_0,q_2), respectively. The MDP has 2 end components $C_1 = (\{(t_1,q_1,q_3)\}, \delta_1)$ and $C_2 = (\{(t_0,q_0,q_2)(t_2,q_0,q_2)\}, \delta_2)$ where δ_1 and δ_2 are uniquely determined by the set of states contained in the end component. We next construct the MDP $\mathcal{M}''$ according to the definition introduced earlier on page 41. The part of $\mathcal{M}''$ reachable from (t_0,q_0,q_2) is depicted in Figure 15. Our target sets for multi-objective reachability in $\mathcal{M}''$ are $T_1 = \{s_{\{1\}}\}$ and $T_2 = \{s_{\{2\}}\}$. We then get the following set of inequalities $L(\mathcal{M}'')$:

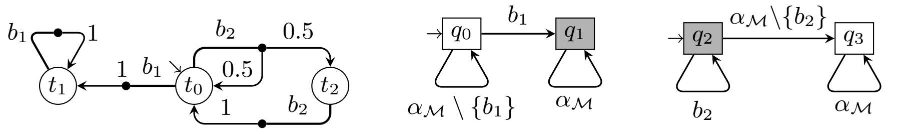
$$\begin{aligned}
1 + y_{((t_2,q_0,q_2),b_2)} + 0.5{\cdot}y_{((t_0,q_0,q_2),b_2)} &= y_{((t_0,q_0,q_2),b_1)} + y_{((t_0,q_0,q_2),b_2)} + y_{((t_0,q_0,q_2),a_{\{2\}})}\\
y_{((t_0,q_0,q_2),b_1)} + y_{((t_1,q_1,q_3),b_1)} &= y_{((t_1,q_1,q_3),b_1)} + y_{((t_1,q_1,q_3),a_{\{1\}})}\\
0.5{\cdot}y_{((t_0,q_0,q_2),b_2)} &= y_{((t_2,q_0,q_2),b_2)} + y_{((t_2,q_0,q_2),a_{\{2\}})}\\
y_{((t_1,q_1,q_3),a_{\{1\}})} &\geqslant 0.6\\
y_{((t_0,q_0,q_2),a_{\{2\}})} + y_{((t_2,q_0,q_2),a_{\{2\}})} &\geqslant 0.3
\end{aligned}$$

These equations *do* have (infinitely many) solutions and we can pick an arbitrary one, e.g. the solution with non-zero values:

$$\begin{aligned}
y_{((t_0,q_0,q_2),b_1)} &= 0.6\\
y_{((t_0,q_0,q_2),b_2)} &= 0.8\\
y_{((t_1,q_1,q_3),a_{\{1\}})} &= 0.6\\
y_{((t_2,q_0,q_2),a_{\{2\}})} &= 0.4\,.
\end{aligned}$$

This gives adversary σ'' which, in (t_0,q_0,q_2), picks b_1 or b_2 with probability $0.6/(0.6{+}0.8){=}\frac{3}{7}$ or $0.8/(0.6{+}0.8){=}\frac{4}{7}$, respectively, and in (t_1,q_1,q_3) and (t_2,q_0,q_2) chooses $a_{\{1\}}$ and $a_{\{2\}}$ deterministically. The induced adversary σ' of $\mathcal{M}'$ is both randomised and history dependent: if only (t_0,q_0,q_2) is in the history, it selects $[b_1{\mapsto}\frac{3}{7}, b_2{\mapsto}\frac{4}{7}]$; for all other paths ending in (t_0,q_0,q_2), it selects $[b_2{\mapsto}1]$. The adversary σ' maps naturally to an adversary σ of $\mathcal{M}$ that satisfies the query $\mathtt{P}_{\geqslant 0.6}[\mathtt{F}\ b_1] \wedge \mathtt{P}_{\geqslant 0.3}[\mathtt{G}\ b_2]$. ■

Quantitative Approaches. The techniques presented in this section can also be generalised in several other ways. Firstly, like for the other verification prob-



(a) MDP $\mathcal{M}$ (b) DRA $\mathcal{A}_1$ for $\mathtt{F}\ b_1$ (c) DRA $\mathcal{A}_2$ for $\mathtt{G}\ b_2$

Fig. 14. The MDP and deterministic Rabin automata for Example 17

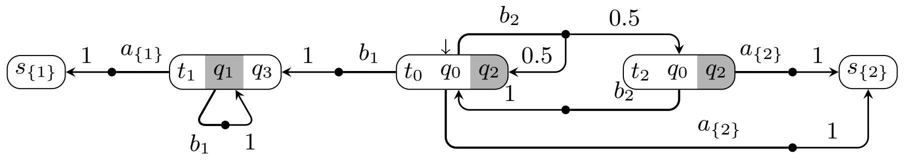


Fig. 15. The MDP $\mathcal{M}''$, built from $\mathcal{M}' = \mathcal{M}\otimes\mathcal{A}_1\otimes\mathcal{A}_2$, for Example 17

lems in this chapter, we can consider *quantitative* approaches to multi-objective model checking [69,45]. We can define *numerical* queries, which may be more useful than existential ones in practice. These optimise one objective, subject to constraints on several others. Formally we have the following definition.

Definition 26 (Numerical multi-objective query). *For a PA $\mathcal{M}$ with initial state $\overline{s}$, multi-objective LTL query θ and LTL formula ψ, a* (maximising) numerical multi-objective query *is to find the following value:*

$$Pr^{\max}_{\mathcal{M},\overline{s}}(\psi \,|\, \theta) \overset{\text{def}}{=} \sup\{Pr^{\sigma}_{\mathcal{M},\overline{s}}(\psi) \mid \sigma \in Adv_{\mathcal{M}} \wedge \sigma, \overline{s} \models \theta\}\,.$$

If the property θ is not satisfied by any adversary of $\mathcal{M}$, the query returns $\perp$. A minimising *numerical multi-objective query is defined similarly.*

Numerical queries can solved at essentially the same cost as existential ones. This is done by adding an objective function to the set of linear inequalities and solving a linear program instead. Multi-objective queries can be further extended by integrating reward-based properties similar to those in Section 5; see [45].

Complexity. Consider a multi-objective LTL query θ for an MDP $\mathcal{M}$. Converting the query to disjunctive normal form can be done in time exponential in the number of connectives of θ, yielding at most exponentially many clauses, each containing at most polynomially many literals. The formula in disjunctive normal form, with negations and operators $<$ and $\leqslant$ removed, contains at most k different LTL formulas, where k is polynomial in the size of θ. We convert each LTL formula to a DRA in time doubly exponential in its size.

For a single clause containing n LTL formulas, we then build the product $\mathcal{M}'$ of $\mathcal{M}$ and the constructed DRAs. This product is polynomial in the size of $\mathcal{M}$, n and the size of the DRAs. Identifying accepting end components and then constructing the MDP $\mathcal{M}''$ both require time polynomial in the size of $\mathcal{M}'$ and exponential in n. The set of equations is constructed in time polynomial in $\mathcal{M}''$, and solved in polynomial time using the techniques for linear programming. The whole procedure thus runs in the time doubly exponential in the size of θ, and polynomial in the size of $\mathcal{M}$.

Controller Synthesis. The problems discussed in this chapter concern the generation of MDP adversaries that satisfy certain formally-specified properties. This is often referred to as *controller synthesis*, and can be generalised in several ways. We can, for example, return to the temporal logic PCTL defined in Section 6 and consider an alternative semantics for the logic. Let us define validity of a PCTL formula under a fixed adversary σ by stipulating $Adv=\{\sigma\}$ in the semantics presented in Definition 13. The problem is to determine whether there exists a σ under which the given formula is true.

This problem has also been studied [9,21,20], and—perhaps surprisingly—it is fundamentally different from the problem in which Adv is the set of all adversaries. In particular, answering the question whether there is a satisfying adversary is undecidable. It becomes decidable when we restrict to qualitative case (i.e. when the probability bounds in the formula are taken from the set $\{0, 1\}$), but even then a satisfying adversary may require infinite memory.

9 Compositional Probabilistic Model Checking

In this section, we discuss *compositional* approaches to probabilistic model checking of MDPs, in particular illustrating an *assume-guarantee* framework [69].

9.1 Probabilistic Automata and Parallel Composition

System designs often comprise multiple components operating in parallel. When these components exhibit stochastic behaviour, MDPs are a natural formalism to model the system since nondeterminism can be used to capture concurrency between the components. In fact, for the purposes of compositional modelling and analysis, it is preferable to use probabilistic automata (PAs) [80,81], which are a (slight) generalisation of MDPs. The essential difference is that a state of a PA can have more than one outgoing transition with the same action label.

Definition 27 (Probabilistic automaton). *A* probabilistic automaton *(PA) is a tuple $\mathcal{M}=(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$, where S is a finite set of states, $\overline{s} \in S$ is an initial state, $\alpha_{\mathcal{M}}$ is a finite alphabet, $\delta_{\mathcal{M}} \subseteq S{\times}(\alpha_{\mathcal{M}}{\cup}\{\tau\}){\times}Dist(S)$ is a finite probabilistic transition relation and $L : S \to 2^{AP}$ is a labelling function mapping each state to a set of atomic propositions taken from a set AP.*

Notice that $\delta_{\mathcal{M}}$ is now a relation, unlike in Definition 4 (see page 5) where it is a function. Observe also that we allow transitions to be labelled with a special τ action, representing "silent" transitions that are internal to some component. We introduce additional notation and use $s \xrightarrow{a} \mu$ to denote that $(s, a, \mu) \in \delta_{\mathcal{M}}$.

Basic notions such as paths, traces and adversaries of MDPs (see Section 3) need slight modifications for PAs, but this is straightforward. In the case of traces, for example, we omit τ actions, following the intuition that these actions are "silent". A technical detail required by the compositional approach of [69] is the use of *partial adversaries*, which can opt to (with some probability) take none of the available choices and remain in the current state. However, model checking of *probabilistic safety properties*, on which the compositional techniques described here are based, is unaffected by this distinction (intuitively, this is because remaining in a state can only decrease the probability of satisfying a safety property). The definition of a PA-DFA product and the process for model checking probabilistic safety properties are also essentially the same as the MDP case; see [69] for details.

We now introduce a few additional concepts required for *compositional* modelling and analysis of PAs.

Definition 28 (Parallel composition of PAs). *If* $\mathcal{M}_i = (S_i, \overline{s}_i, \alpha_{\mathcal{M}_i}, \delta_{\mathcal{M}_i}, L_i)$ *are PAs for* $i{=}1,2$*, then their* parallel composition*, denoted* $\mathcal{M}_1 \| \mathcal{M}_2$*, is given by the PA* $(S_1{\times}S_2, (\overline{s}_1, \overline{s}_2), \alpha_{\mathcal{M}_1} \cup \alpha_{\mathcal{M}_2}, \delta_{\mathcal{M}_1\|\mathcal{M}_2}, L)$ *where* $\delta_{\mathcal{M}_1\|\mathcal{M}_2}$ *is defined such that* $(s_1, s_2) \xrightarrow{a} \mu_1{\times}\mu_2$ *if and only if one of the following holds:*

– $s_1 \xrightarrow{a} \mu_1$*,* $s_2 \xrightarrow{a} \mu_2$ *and* $a \in \alpha_{\mathcal{M}_1} \cap \alpha_{\mathcal{M}_2}$
– $s_1 \xrightarrow{a} \mu_1$*,* $\mu_2 = [s_2{\mapsto}1]$ *and* $a \in \alpha_{\mathcal{M}_1} \backslash \alpha_{\mathcal{M}_2}$
– $\mu_1 = [s_1{\mapsto}1]$*,* $s_2 \xrightarrow{a} \mu_2$ *and* $a \in \alpha_{\mathcal{M}_2} \backslash \alpha_{\mathcal{M}_1}$

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

This form of parallel composition [80,81], which allows multi-way synchronisation over the same action by several components, is in a similar style to the scheme used in the process algebra CSP [79] and is also used in PRISM [56]. By default, we assume that $\mathcal{M}_1$ and $\mathcal{M}_2$ synchronise over all common actions. However, this can easily be generalised to incorporate more flexible definitions of synchronisation, as well as operators to hide and rename action labels.

Definition 29 (Alphabet extension of PA). *For a PA* $\mathcal{M}{=}(S, \overline{s}, \alpha_{\mathcal{M}}, \delta_{\mathcal{M}}, L)$ *and set of actions* α*, we* extend $\mathcal{M}$*'s alphabet to* α*, denoted* $\mathcal{M}[\alpha]$*, as follows:* $\mathcal{M}[\alpha]{=}(S, \overline{s}, \alpha_{\mathcal{M}}{\cup}\alpha, \delta_{\mathcal{M}[\alpha]}, L)$ *where* $\delta_{\mathcal{M}[\alpha]}{=}\delta_{\mathcal{M}}{\cup}\{(s, a, [s{\mapsto}1]) \mid s{\in}S{\wedge}a{\in}\alpha{\backslash}\alpha_{\mathcal{M}}\}$*.*

Example 18. Figure 16 shows two PAs used to model a system comprising a machine ($\mathcal{M}_2$) and a controller ($\mathcal{M}_1$) that is responsible for powering it down. When $\mathcal{M}_1$ *detect*s a problem, it should send two messages: *warn* and then *shutdown* to $\mathcal{M}_2$. However, with probability 0.2, it fails to transmit *warn*. If $\mathcal{M}_2$ does not receive the *warn* message before the *shutdown* message, there is a 10% chance it will *fail* to shut down correctly. Figure 16(c) also shows the DFA $\mathcal{A}_G^{err}$ for a safety property Φ_G "action *fail* never occurs". On the parallel composition $\mathcal{M}_1\|\mathcal{M}_2$, we can compute $Pr^{\min}_{\mathcal{M}_1\|\mathcal{M}_2,(s_0,t_0)}(\Phi_G) = 1 - 0.2{\cdot}0.1 = 0.98$. Thus, the initial state of $\mathcal{M}_1\|\mathcal{M}_2$ satisfies the probabilistic safety property $\mathtt{P}_{\geqslant 0.98}[\Phi_G]$. ■

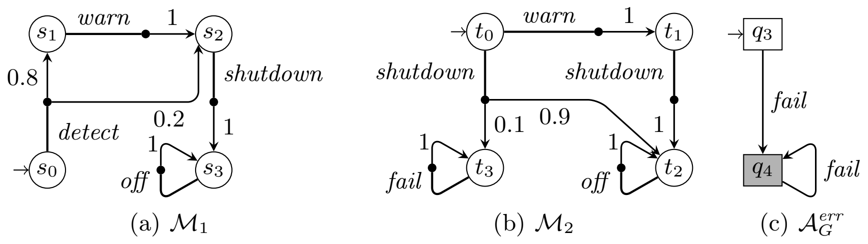


Fig. 16. Two PAs $\mathcal{M}_1, \mathcal{M}_2$ and a DFA $\mathcal{A}_G^{err}$ for a safety property Φ_G

9.2 Assume-Guarantee Verification

We now describe the approach for *compositional* verification of probabilistic automata presented in [69]. This is based on the popular *assume-guarantee* paradigm, in which components of a system are verified separately, under *assumptions* about their environment. After verifying that the other system components satisfy these assumptions, proof rules are used to establish properties about the combined system. We will first define the basic underlying ideas and then illustrate one of the assume-guarantee proof rules from [69].

The approach uses *probabilistic assume-guarantee triples*. These take the form $\langle\Phi_A\rangle_{\geqslant p_A}\,\mathcal{M}\,\langle\Phi_G\rangle_{\geqslant p_G}$, where $\mathtt{P}_{\geqslant p_A}[\Phi_A]$ and $\mathtt{P}_{\geqslant p_G}[\Phi_G]$ are probabilistic safety properties and $\mathcal{M}$ is a PA. Informally, the triple means: "whenever $\mathcal{M}$ is part of a system satisfying Φ_A with probability at least p_A, the system satisfies Φ_G with probability at least p_G". Formally, we have the following definition.

Definition 30 (Probabilistic assume-guarantee triple). *If* $\mathtt{P}_{\geqslant p_A}[\Phi_A]$ *and* $\mathtt{P}_{\geqslant p_G}[\Phi_G]$ *are probabilistic safety properties,* $\mathcal{M}$ *is a PA and* $\alpha_G \subseteq \alpha_A \cup \alpha_{\mathcal{M}}$*, then* $\langle\Phi_A\rangle_{\geqslant p_A}\,\mathcal{M}\,\langle\Phi_G\rangle_{\geqslant p_G}$ *is a* probabilistic assume-guarantee triple*, meaning:*

$$\forall\sigma\in Adv_{\mathcal{M}[\alpha_A]}\ .\ \big(\,Pr^{\sigma}_{\mathcal{M}[\alpha_A],\overline{s}}(\Phi_A)\geqslant p_A \rightarrow Pr^{\sigma}_{\mathcal{M}[\alpha_A],\overline{s}}(\Phi_G)\geqslant p_G\,\big)\,.$$

The use of $\mathcal{M}[\alpha_A]$, i.e. $\mathcal{M}$ extended to the alphabet of Φ_A, in the above is needed to allow the assumption to refer to actions not used in $\mathcal{M}$. We use $\langle\mathtt{true}\rangle\,\mathcal{M}\,\langle\Phi_G\rangle_{\geqslant p_G}$ to indicate the case where there is no assumption. This is therefore equivalent to $\overline{s}\models\mathtt{P}_{\geqslant p_G}[\Phi_G]$ and can be verified using the techniques described in Section 7.1. Checking that a triple $\langle\Phi_A\rangle_{\geqslant p_A}\,\mathcal{M}\,\langle\Phi_G\rangle_{\geqslant p_G}$ holds in the general case, however, requires the use of multi-objective (LTL) probabilistic model checking, as discussed in Section 8.

Proposition 4 ([69]). *If* $\mathcal{M}$ *is a PA,* $\mathtt{P}_{\geqslant p_A}[\Phi_A]$ *and* $\mathtt{P}_{\geqslant p_G}[\Phi_G]$ *are probabilistic safety properties and* $\mathcal{M}' = \mathcal{M}[\alpha_A]\otimes\mathcal{A}_A^{err}\otimes\mathcal{A}_G^{err}$ *with initial state* $\overline{s}'$*, then:*

$$\begin{aligned}&\langle\Phi_A\rangle_{\geqslant p_A}\,\mathcal{M}\,\langle\Phi_G\rangle_{\geqslant p_G}\\ &\iff \neg\exists\sigma'\in Adv_{\mathcal{M}'}\ .\ \big(\,Pr^{\sigma'}_{\mathcal{M}',\overline{s}'}(\mathtt{G}\ \neg err_A)\geqslant p_A \wedge Pr^{\sigma'}_{\mathcal{M}',\overline{s}'}(\mathtt{F}\ err_G)>1-p_G\,\big)\,.\end{aligned}$$

Based on the definitions given above, [69] presents the following *asymmetric* assume-guarantee proof rule for a two component system $\mathcal{M}_1\|\mathcal{M}_2$.

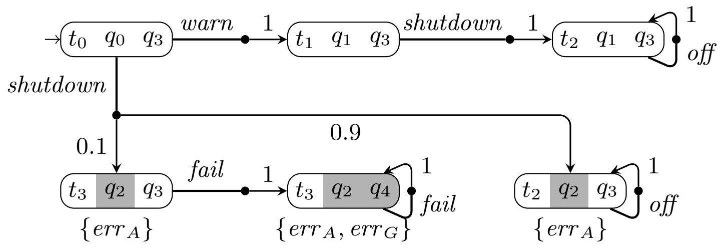


Fig. 17. The product PA $\mathcal{M}_2 \otimes \mathcal{A}_A^{err} \otimes \mathcal{A}_G^{err}$ for the PA $\mathcal{M}_2$ and error automata $\mathcal{A}_A^{err}$ and $\mathcal{A}_G^{err}$ from Figures 8 and 16 (see Example 19)

Proposition 5 ([69]). *If $\mathcal{M}_1, \mathcal{M}_2$ are PAs and $\mathtt{P}_{\geqslant p_A}[\Phi_A], \mathtt{P}_{\geqslant p_G}[\Phi_G]$ probabilistic safety properties such that $\alpha_A \subseteq \alpha_{\mathcal{M}_1}$ and $\alpha_G \subseteq \alpha_{\mathcal{M}_2} \cup \alpha_A$, then the following proof rule holds:*

$$\frac{\begin{array}{c}\langle \mathtt{true} \rangle\, \mathcal{M}_1 \,\langle \Phi_A \rangle_{\geqslant p_A} \\ \langle \Phi_A \rangle_{\geqslant p_A} \mathcal{M}_2 \,\langle \Phi_G \rangle_{\geqslant p_G}\end{array}}{\langle \mathtt{true} \rangle\, \mathcal{M}_1 \parallel \mathcal{M}_2 \,\langle \Phi_G \rangle_{\geqslant p_G}} \qquad (\textsc{ASym})$$

This rule is asymmetric in the sense that it only uses one assumption ($\mathtt{P}_{\geqslant p_A}[\Phi_A]$) about one of the components ($\mathcal{M}_1$). Given such an assumption, we can now model check the probabilistic safety property $\mathtt{P}_{\geqslant p_G}[\Phi_G]$ on $\mathcal{M}_1 \| \mathcal{M}_2$ in a *compositional* fashion. More precisely, verification reduces to two sub-problems, one for each premise of the rule: (i) checking a probabilistic safety property on $\mathcal{M}_1$; (ii) performing multi-objective model checking on $\mathcal{M}_2[\alpha_A] \otimes \mathcal{A}_A^{err} \otimes \mathcal{A}_G^{err}$. If $\mathcal{A}_A^{err}$ is much smaller than $\mathcal{M}_1$, significant gains in performance and/or scalability can be made.

Example 19. We illustrate the rule (ASym) on the PAs $\mathcal{M}_1, \mathcal{M}_2$ and property $\mathtt{P}_{\geqslant 0.98}[\Phi_G]$ from Example 18 (see Figure 16). We also reuse the probabilistic safety property $\mathtt{P}_{\geqslant 0.8}[\Phi_A]$ from Example 13, where Φ_A means "*warn* occurs before *shutdown*" and its error automaton is the DFA $\mathcal{A}_A^{err}$ in Figure 8. Our goal is to verify that $\mathtt{P}_{\geqslant 0.98}[\Phi_G]$ holds in $\mathcal{M}_1 \| \mathcal{M}_2$ using the rule (ASym) and with assumption $\mathtt{P}_{\geqslant 0.8}[\Phi_A]$. To check the first premise of (ASym), we need to verify $s_0 \models \mathtt{P}_{\geqslant 0.8}[\Phi_A]$ in $\mathcal{M}_1$, which has already been shown to be true in Example 13.

Next, we check the second premise, i.e. $\langle \Phi_A \rangle_{\geqslant 0.8}\, \mathcal{M}_2 \,\langle \Phi_G \rangle_{\geqslant 0.98}$, using Proposition 4 above. The product $\mathcal{M}_2 \otimes \mathcal{A}_A^{err} \otimes \mathcal{A}_G^{err}$ is shown in Figure 17 (in this case, $\mathcal{M}_2[\alpha_A] = \mathcal{M}_2$). Recall that we label product states corresponding to accepting states of $\mathcal{A}_A^{err}$ and $\mathcal{A}_G^{err}$ with atomic propositions err_A and err_G. We also shade these grey in Figure 17 for clarity. We need to establish that there is *no* adversary under which the probability of remaining within states not satisfying err_A is at least 0.8 *and* the probability of reaching an err_G state is above $1-0.98 = 0.02$. Through a manual inspection, we see that no such adversary exists. This check can be automated using the multi-objective probabilistic model checking techniques from Section 8. In conclusion, combining the two results, we can state that $(s_0, t_0) \models \mathtt{P}_{\geqslant 0.98}[\Phi_G]$ *does* hold in $\mathcal{M}_1 \| \mathcal{M}_1$, as required.

Consider, however, the adversary σ of $\mathcal{M}_2 \otimes \mathcal{A}_A^{err} \otimes \mathcal{A}_G^{err}$ which, in the initial state, chooses *warn* with probability 0.8 and *shutdown* with probability 0.2. This satisfies $\mathtt{G}\ \neg err_A$ with probability 0.8 and $\mathtt{F}\ err_G$ with probability 0.02. Hence, $\langle \Phi_A \rangle_{\geqslant 0.8}\, \mathcal{M}_2\, \langle \Phi_G \rangle_{\geqslant p_G}$ does *not* hold for any value of $p_G > 1 - 0.02 = 0.98$. ■

For details of additional probabilistic assume-guarantee proof rules, as well as *quantitative* approaches to the problem, see [69]; for further extensions, including the use of ω-regular and reward-based properties, see [45].

10 Tools & Case Studies

There are several software tools available for probabilistic verification of Markov decision processes (or probabilistic automata). One of the most widely used of these is PRISM [56], which incorporates the majority of the techniques described in this tutorial. It also supports discrete- and continuous-time Markov chains, and probabilistic timed automata.

Two other tools for probabilistic model checking of MDPs are LiQuor [30], which has an expressive modelling language extending Promela with probabilities, and ProbDiVinE [13], which focuses on parallel and distributed implementations of LTL model checking for MDPs. RAPTURE [60] and PASS [50] both provide verification of MDPs using abstraction and refinement. There are also various other probabilistic model checkers for discrete- and continuous-time Markov chains, notably MRMC [61]. For a more extensive list, see [92].

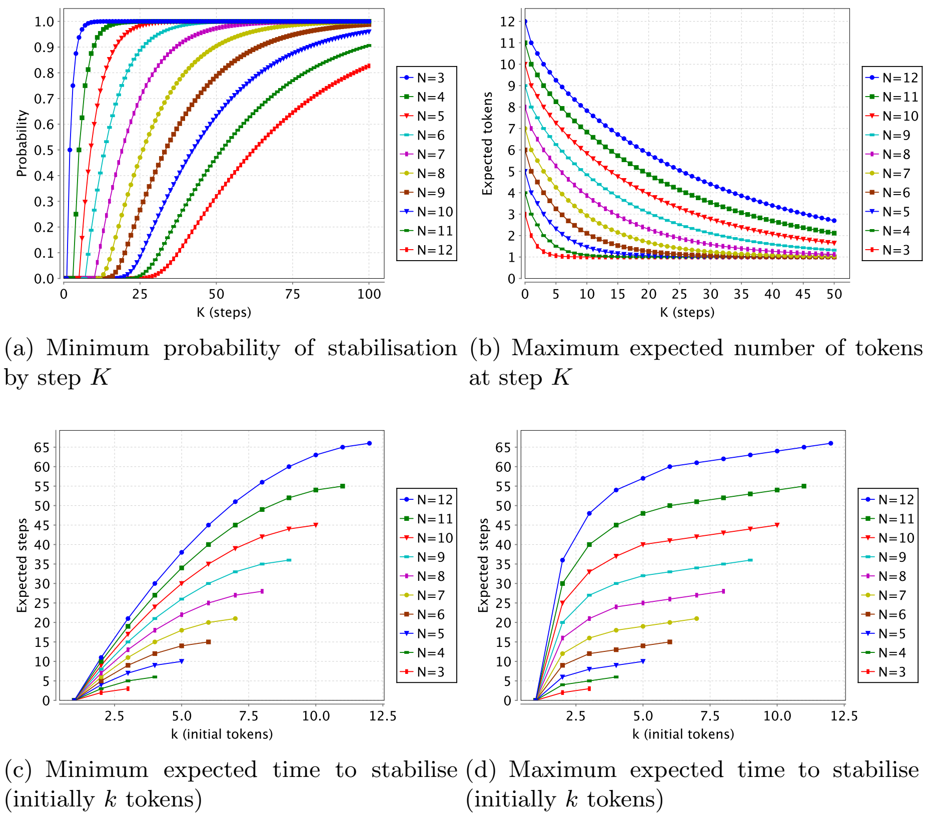
In the following sections, we present three large probabilistic model checking case studies, based on the use of Markov decision processes. A selection of further examples can be found at [90].

10.1 Case Study: Israeli and Jalfon's Self-stabilisation Protocol

A *self-stabilising protocol* for a network of processes is a protocol which transforms a system from an *unstable* state to a *stable* state in a finite number of steps and without any outside intervention. Here, we consider Israeli and Jalfon's *randomised* self-stabilising protocol [59].

The protocol of Israeli and Jalfon is designed for a network which is an oriented ring of identical processes $P_1, \ldots, P_N$ with bidirectional communication. It operates asynchronously with an arbitrary scheduler, and each process P_i has a boolean variable q_i which represents the fact that it has a token. A process is said to be *active* if it has a token and only active processes can be scheduled. When a process is scheduled, it makes a (uniform) random choice as to whether to move its token to its left or right and when tokens collide they are merged into a single one. The stable configurations are those where there is exactly one active process, i.e. exactly one token. Once a stable configuration has been reached, the token should be passed around the ring forever in a fair manner.

We first verify that the protocol does indeed stabilise, by verifying that a stable state is reached with minimum probability 1 for all possible initial configurations. This property can be expressed in PCTL as the formula $\mathtt{P}_{\geqslant 1}[\mathtt{F}\ stable]$



(a) Minimum probability of stabilisation by step K

(b) Maximum expected number of tokens at step K

(c) Minimum expected time to stabilise (initially k tokens)

(d) Maximum expected time to stabilise (initially k tokens)

Fig. 18. Israeli-Jalfon's self-stabilisation protocol: results

where *stable* is the atomic proposition representing the fact that there is only one token present in the state. We also check that the token is passed around the ring in a fair manner. Since we have already shown that, with minimum probability 1 there is eventually only one token, it is sufficient to check that each process obtains the token infinitely often. For process P_i, we use the probabilistic LTL specification $\mathtt{P}_{\geqslant 1}[\mathtt{G}\ \mathtt{F}\ active_i]$, where the atomic proposition $active_i$ indicates that P_i is active, i.e. has a token.

Next, we investigate the protocol's performance with the properties:

- the minimum probability of stabilising within K steps when starting from any initial configuration, ($\mathtt{P}_{\mathtt{min}=?}[\mathtt{F}^{\leqslant K}\ stable]$);
- the maximum expected number of tokens after K steps when starting from any initial configuration, ($\mathtt{R}^{tokens}_{\mathtt{max}=?}[\mathtt{I}^{=K}]$, where the reward structure *tokens* assigns a reward to each state corresponding to the number of tokens present in the state and there are no action rewards);
- the minimum and maximum expected time to reach a stable state given that the initial number of tokens is k ($\mathtt{R}^{steps}_{\mathtt{min}=?}[\mathtt{F}\ stable]$ and $\mathtt{R}^{steps}_{\mathtt{max}=?}[\mathtt{F}\ stable]$, where

the reward structure *steps* assigns a reward of 1 to each action and there are no state rewards).

Figure 18 presents a summary of the results as the number of processes (N) varies from 3 to 12. We see that the performance of the protocol decreases as the number of processes increases. Considering the individual properties, we observe that the probability to stabilise by step K (Figure 18(a)) and the expected number of tokens at step K (Figure 18(b)) both converge towards 1 as K increases. This is to be expected as we have already verified that the minimum probability of stabilisation is 1. Considering the expected time to stabilise (Figures 18(c)-(d)), the results show the expected time to stabilise increasing as the initial number of tokens (k) increases. Further investigation for the other properties shows similar trends as the initial number of tokens is increased.

10.2 Case Study: Dynamic Power Management

Dynamic Power Management (DPM) is a technique for saving energy in devices that can be turned on and off under operating system control. Such methods are particularly important in mobile, hand-held and embedded devices, for which minimisation of energy consumption is a key issue. DPM-enabled devices typically have several *power states* with different energy consumption rates. A DPM *policy* is used to decide when commands to transition between these states should be issued, based on the current state of the system.

The components of a DPM system are: a Service Provider (SP) representing the device under power management control; a Service Requester (SR) which sends requests to the SP; a Service Request Queue (SRQ), which stores the requests that have yet to be served; and the Power Manager (PM), which sends commands to the SP, based on observations of the system and a DPM policy.

The particular system we consider here is the IBM TravelStar VP [58], a commercially available hard disk drive. Our model is based on the one in [15]. The hard disk, i.e. the SP, can operate in five different states as shown in Table 2(a), which also provides the power dissipation in each of these states. It is only in state *active* that the device can perform data reads and writes. In state *idle* the disk is spinning but some of the electronic components of the disk drive have been switched off. The state *idlelp* ("idle low power") is similar except that it has a lower power dissipation. The states *stby* and *sleep* correspond to the disk being spun down. Transition times between states are in Figure 2(b).

We use the following reward structures during our analysis:

– *power* is used to investigate the energy consumption of the system and is defined using the power dissipation of the SP given in Figure 2(a);
– *queue* is used for analysing the size of the service request queue and is constructed by setting the reward in each state to the size of the SRQ;
– *lost* represents the loss of requests by assigning a reward of 1 to actions representing the arrival of a request when the queue is full.

(a) Average power dissipation

State	Power dissipation (W)
active	2.5
idle	1.5
idlelp	0.8
stby	0.3
sleep	0.1

(b) Expected transition times

	active	*idle*	*idlelp*	*stby*	*sleep*
active	-	1ms	5ms	2.2sec	6sec
idle	1ms	-	5ms	2.2sec	6sec
idlelp	5ms	-	-	2.2sec	6sec
stby	2.2sec	-	-	-	6sec
sleep	6sec	-	-	-	-

Table 2. Dynamic power management: properties of the states of the hard drive

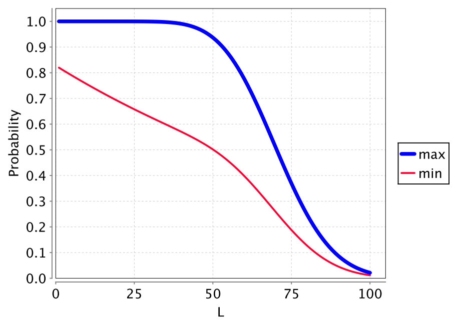
For further details of the PRISM model see [74,91].

In Figure 19 we present model checking results for computation of the minimum and maximum values for the following properties, when the maximum queue size is 2 and there is no constraint on the battery life of the system:

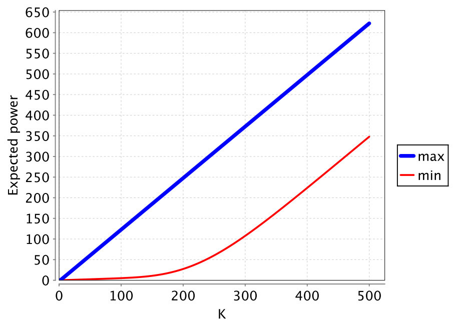
– the probability of L lost requests by step $1,000$ (e.g. $\mathtt{P}_{\mathtt{min=?}}[\mathtt{F}^{\leqslant 1000}\ lost_L]$);
– the expected energy consumed during the first K steps (e.g. $\mathtt{R}^{power}_{\mathtt{min=?}}[\mathtt{C}^{\leqslant K}]$);
– the expected queue size at step K (e.g. $\mathtt{R}^{queue}_{\mathtt{min=?}}[\mathtt{I}^{=K}]$);
– the expected number of lost requests after K steps (e.g. $\mathtt{R}^{lost}_{\mathtt{min=?}}[\mathtt{C}^{\leqslant K}]$).

The results demonstrate that, depending on the power manager's choices, there can be a large difference both in the energy consumption (Figure 19(b)) and in the performance (or quality of service) of the device (Figures 19(a), (c) and (d)). Analysing the best- and worst-case choices, i.e. generating the adversaries that yield the optimal values, we find the best-case performance and worst-case energy consumption is obtained by keeping the SP in the *active* state. On the other hand, the worst-case performance and best-case energy consumption is obtained by keeping the SP in *sleep* whenever possible and only switching to *active* when necessary (to prevent the power manager which minimises energy consumption by ignoring all requests and keeping the SP in sleep, we require the SP to be switched to *active* when the SRQ becomes full).

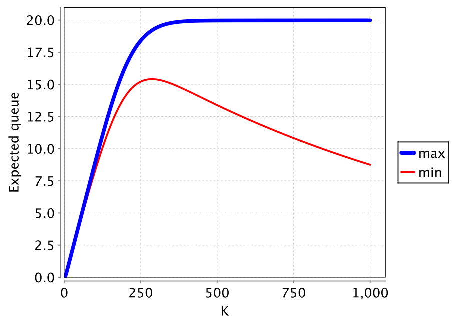
To further investigate the power-versus-performance trade-off of the system, we now apply the techniques of Section 8 and [45] (using the PRISM extension implemented in [69,45]) to perform controller synthesis on the disk-drive. To allow us to consider long-run average properties in the analysis we follow [15] and constrain the battery life of the system. Applying these techniques, we can minimise the expected energy consumption under restrictions on, for example, the probability that a request waits more than K steps, the probability that N requests are lost, the average request-queue size or the expected number of lost requests. To illustrate this approach, Figure 20(a) plots the minimum expected energy consumption under restrictions on the probability that 10 requests are lost, while Figure 20(b) plots the minimum expected energy consumption under restrictions on both the average queue size and expected number of lost requests. In both cases the maximum size of the SRQ is set to 2. These results demonstrate the familiar power-versus-performance trade-off: policies can offer improved performance, but at the expense of using more energy.



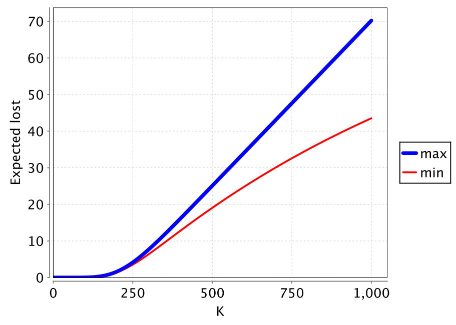
(a) Prob. L lost requests by step 1,000



(b) Expected energy consumed by step K



(c) Expected queue size at step K



(d) Expected lost requests by step K

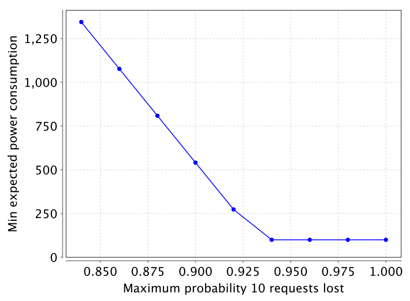
Fig. 19. Dynamic power management: model checking results

For an example of the controllers generated through this approach, consider constraints of 0.9 and 100 on the average queue size and expected number of lost requests. The optimal expected energy consumption is 1,874.6 and is achieved by the following PM:

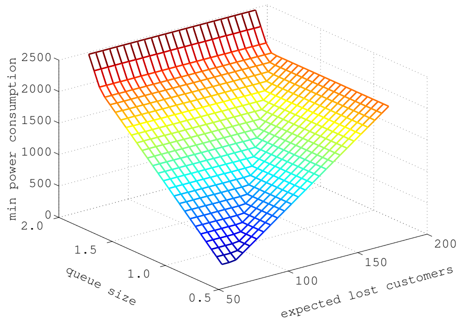
– if the SP is in *active*, the SR is in *idle* and the queue is empty, move SP to *idle*;
– if the SP is in *sleep*, the SR is in *idle* and the queue is full, then:
  • with probability 0.972958 keep SP in *sleep*
  • with probability 0.027042 move SP to *active*;
– otherwise, keep the SP in its current state.

On the other hand, if the constraints on the average queue size and the expected lost requests are 0.8 and 80 respectively, then the optimal expected energy consumption is 2,134.5 and is achieved by the following PM:

– immediately move the SP from *sleep* to *active*;
– if the SP is in *idle*, the SR is in *req* and the queue is empty, move SP to *active*;
– if the SP is in *active*, the SR is in *idle* and the queue is non-empty, then
  • with probability 0.995523 move SP to *idle*
  • with probability 0.004477 keep SP in *active*;
– otherwise, keep the SP in its current state.



(a) Constraint on maximum probability of 10 lost requests



(b) Constraints on maximum average queue size and expected lost requests

Fig. 20. Dynamic power management: controller synthesis

10.3 Case Study: Aspnes & Herlihy's Consensus Algorithm

A distributed consensus protocol is an algorithm for ensuring that a collection of distributed processes, which start in some initial value supplied by their environment, eventually terminate agreeing on the same value. In this case study, we consider the randomised distributed consensus algorithm of Aspnes & Herlihy [5] and use it to demonstrate the applicability of the compositional verification approach for safety properties introduced in Section 9.

The algorithm of Apnes & Herlihy allows N processes in a distributed network to reach a consensus and employs, in each round, a shared coin protocol parameterised by K. The safety property we will consider is that "agreement is reached by round R", where the corresponding set of bad prefixes are those that end with the action of any process entering round $R{+}1$. We will in fact use the quantitative version of the composition techniques presented in [69], which allows us to compute a lower bound on the minimum probability of satisfying the safety property, by computing an upper bound on the maximum probability of performing a bad prefix (see Proposition 1).

The probabilistic automata model of the algorithm is based on the one presented in [71]. It comprises the parallel composition of: N PAs, each representing one process, and R PAs, one for the shared coin protocol of each round. The compositional verification consists of the following steps:

- first, using the techniques of Section 7, we calculate the minimum probability that the coin protocols of rounds $1, \ldots, R{-}2$ each satisfy a safety property (the property is that the coin protocol returns the same coin value for all processes, and therefore the bad prefixes are those where the coin protocol returns different values to different processes);
- second, we combine these results through $R{-}2$ applications of of the Async rule of [69] to return a probabilistic safety property satisfied by the (asynchronous) composition of the shared coin protocols for the first $R{-}2$ rounds;
- finally, this probabilistic safety property is used as the assumption for an application of the Asym rule (see Theorem 5), yielding the final property of

Parameters			Non-compositional			Compositional		
N	K	R	Model size	Time (s)	Result[†]	LP size	Time (s)	Result[†]
2	2	3	5,158	1.6	0.108333	1,064	**0.9**	0.108333
2	20	3	40,294	108.1	0.012500	1,064	**7.4**	0.012500
2	2	4	20,886	3.6	0.011736	2,372	**1.2**	0.011736
2	20	4	166,614	343.1	0.000156	2,372	**7.8**	0.000156
2	2	5	83,798	7.7	0.001271	4,988	**2.2**	0.001271
2	20	5	671,894	1,347	0.000002	4,988	**8.8**	0.000002
3	2	3	1,418,545	18,971	0.229092	40,542	**29.6**	0.229092
3	12	3	16,674,145*	>24h	-	40,542	**49.7**	0.041643
3	20	3	39,827,233*	>24h	-	40,542	**125.3**	0.024960
3	2	4	150,487,585	78,955	0.052483	141,168	**376.1**	0.052483
3	12	4	1,053,762,385*	mem-out	-	141,168	**396.3**	0.001734
3	20	4	2,028,200,209*	mem-out	-	141,168	**471.9**	0.000623

* These models can be constructed, but not model checked, in PRISM.

[†] Results are maximum probabilities of error so actual values are these subtracted from 1.

Table 3. Randomised consensus algorithm: performance of compositional verification

interest on the combined system: the minimum probability that agreement is reached by round R.

Table 3 shows performance results (taken from [69]) for compositional verification on this case study. It gives the total time required to perform verification, both compositionally (as above) and non-compositionally (using PRISM). To give an indication of the improvements in scalability, the table shows the size of the PA (number of states) for the full system and the number of variables in the LP problems used for multi-objective model checking in the compositional case (see Section 8). As can be seen, for this case study, the compositional approach is faster and permits analysis of models that are infeasible with conventional (non-compositional) techniques. See [69] for further details.

11 Conclusions and Further Reading

This tutorial has given a general introduction to probabilistic model checking, focusing on the model of Markov decision processes. We have covered the basic underlying theory for this model, discussed techniques for model checking a wide array of quantitative properties and presented some illustrative case studies.

We conclude by briefly outlining a few of the active research topics in this area and give some pointers to further reading. In the context of automated verification of probabilistic systems, several key challenges are being addressed. One is extending the range of models to which probabilistic model checking can be applied. Another is improving the scalability of the techniques to handle larger and more complex models. There are also many other ways in which the functionality and applicability of probabilistic verification are being improved.

Models. Recent advances have been made regarding model checking for the following extensions of MDPs: probabilistic timed automata (see e.g. [70] for a survey); probabilistic hybrid systems (see e.g. [88,46]); continuous-time MDPs and continuous-time games (see e.g. [10,23,73,77]); interactive Markov chains (see e.g. [87]); and recursive MDPs and games (see e.g. [41,22]).

Scalability. A variety of approaches are being considered to improve scalability. One example is the development of *abstraction and refinement* frameworks [37,55,26,63], some of which have been applied in practice to verification of probabilistic timed automata [68], probabilistic software [62] and PRISM models [50]. Other promising directions include: *partial order reduction* [49,31], *symmetry reduction* [66,39], algorithms for *simulation* and *bisimulation* relations [25,86] and *compositional* probabilistic verification techniques [69,43,38].

Other directions. Many other interesting topics are being studied on MDPs and related models. These include: *probabilistic counterexample* generation [4,3], verification under *fairness* [8] and under *restricted classes of adversaries* [47,29], *parametric* model checking [51], *synthesis* of parameters [52] and models [28], and *run-time* probabilistic model checking [24,44].

Acknowledgments

The authors are supported in part by EU FP7 project CONNECT, ERC Advanced Grant VERIWARE and EPSRC grant EP/F001096/1. Vojtěch Forejt is also supported by a Royal Society Newton Fellowship and the Institute for Theoretical Computer Science, project no. 1M0545. In addition, we acknowledge Dinara Karkabayeva for providing a preliminary version of the results in Section 4.2 and thank the authors of [15] for making their model of the IBM TravelStar VP available to us.

References

1. de Alfaro, L.: Formal Verification of Probabilistic Systems. Ph.D. thesis, Stanford University (1997)
2. de Alfaro, L.: From fairness to chance. In: Baier, C., Huth, M., Kwiatkowska, M., Ryan, M. (eds.) Proc. 1st Int. Workshop Probabilistic Methods in Verification (PROBMIV'98). ENTCS, vol. 22. Elsevier (1998)
3. Aljazzar, H., Leue, S.: Generation of counterexamples for model checking of Markov decision processes. In: Proc. 6th Int. Conf. Quantitative Evaluation of Systems (QEST'09). pp. 197–206. IEEE CS Press (2009)
4. Andrés, M., D'Argenio, P., van Rossum, P.: Significant diagnostic counterexamples in probabilistic model checking. In: Chockler, H., Hu, A. (eds.) Proc. 4th Int. Haifa Verification Conf. Hardware and Software: Verification and Testing (HVC'08). LNCS, vol. 5394, pp. 129–148. Springer (2008)
5. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. Journal of Algorithms 15(1), 441–460 (1990)
6. Aziz, A., Singhal, V., Balarin, F., Brayton, R., Sangiovanni-Vincentelli, A.: It usually works: The temporal logic of stochastic systems. In: Wolper, P. (ed.) Proc. 7th Int. Conf. Computer Aided Verification (CAV'95). LNCS, vol. 939, pp. 155–165. Springer (1995)

7. Baier, C.: On algorithmic verification methods for probabilistic systems (1998), habilitation thesis, Fakultät für Mathematik & Informatik, Universität Mannheim
8. Baier, C., Größer, M., Ciesinski, F.: Quantitative analysis under fairness constraints. In: Liu, Z., Ravn, A. (eds.) Proc. 7th Int. Symp. Automated Technology for Verification and Analysis (ATVA'09). LNCS, vol. 5799, pp. 135–150. Springer (2009)
9. Baier, C., Größer, M., Leucker, M., Bollig, B., Ciesinski, F.: Controller synthesis for probabilistic systems. In: Lévy, J.J., Mayr, E., Mitchell, J. (eds.) Proc. 3rd IFIP Int. Conf. Theoretical Computer Science (TCS'06). pp. 493–5062. Kluwer (2004)
10. Baier, C., Hermanns, H., Katoen, J.P., Haverkort, B.: Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. Theoretical Computer Science 345(1), 2–26 (2005)
11. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press (2008)
12. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distributed Computing 11(3), 125–155 (1998)
13. Barnat, J., Brim, L., Cerna, I., Ceska, M., Tumova, J.: ProbDiVinE-MC: Multi-core LTL model checker for probabilistic systems. In: Proc. 5rd Int. Conf. Quantitative Evaluation of Systems (QEST'08). pp. 77–78. IEEE CS Press (2008)
14. Bellman, R.: Dynamic Programming. Princeton University Press (1957)
15. Benini, L., Bogliolo, A., Paleologo, G., De Micheli, G.: Policy optimization for dynamic power management. IEEE Trans. Computer-Aided Design of Integrated Circuits and Systems 8(3), 299–316 (2000)
16. Bertsekas, D.: Dynamic Programming and Optimal Control, Volumes 1 and 2. Athena Scientific (1995)
17. Bertsekas, D., Tsitsiklis, J.: An analysis of stochastic shortest path problems. Mathematics of Operations Research 16(3), 580–595 (1991)
18. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P. (ed.) Proc. 15th Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95). LNCS, vol. 1026, pp. 499–513. Springer (1995)
19. Billingsley, P.: Probability and Measure. Wiley (1995)
20. Brázdil, T., Forejt, V., Kučera, A.: Controller synthesis and verification for Markov decision processes with qualitative branching time objectives. In: Aceto, L., Damgård, I., Goldberg, L., Halldórsson, M., Ingólfsdóttir, A., Walukiewicz, I. (eds.) Proc. 35th Int. Colloq. Automata, Languages and Programming, Part II (ICALP'08). LNCS, vol. 5126, pp. 148–159. Springer (2008)
21. Brázdil, T., Brožek, V., Forejt, V., Kučera, A.: Stochastic games with branching-time winning objectives. In: 21th IEEE Symp. Logic in Computer Science (LICS 2006). pp. 349–358. IEEE CS Press (2006)
22. Brázdil, T., Brožek, V., Kučera, A., Obdržálek, J.: Qualitative reachability in stochastic BPA games. In: Albers, S., Marion, J.Y. (eds.) 26th Int. Symp. Theoretical Aspects of Computer Science (STACS'09). LIPIcs, vol. 3, pp. 207–218. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
23. Brázdil, T., Forejt, V., Krčál, J., Křetínský, J., Kučera, A.: Continuous-time stochastic games with time-bounded reachability. In: Kannan, R., Kumar, K. (eds.) Proc. 29th Int. Conf. Foundations of Software Technology and Theoretical Computer Science (FSTTCS'09). LIPIcs, vol. 4, pp. 61–72. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2009)
24. Calinescu, R., Grunske, L., Kwiatkowska, M., Mirandola, R., Tamburrelli, G.: Dynamic QoS management and optimisation in service-based systems. IEEE Transactions on Software Engineering (2010)

25. Cattani, S., Segala, R.: Decision algorithms for probabilistic bisimulation. In: Brim, L., Janar, P., Ketinsky, M., Kuera, A. (eds.) Proc. 14th Int. Conf. Concurrency Theory (CONCUR'02). LNCS, vol. 2421, pp. 371–385. Springer (August 2002)
26. Chadha, R., Viswanathan, M.: A counterexample guided abstraction-refinement framework for Markov decision processes. ACM Transactions on Computational Logic 12(1), 1–49 (2010)
27. Chatterjee, K., Henzinger, T.: Value iteration. In: Grumberg, O., Veith, H. (eds.) 25 Years of Model Checking - History, Achievements, Perspectives. LNCS, vol. 5000, pp. 107–138. Springer (2008)
28. Chatterjee, K., Henzinger, T., Jobstmann, B., Singh, R.: Measuring and synthesizing systems in probabilistic environments. In: Touili, T., Cook, B., Jackson, P. (eds.) Proc. 22nd Int. Conf. Computer Aided Verification (CAV'10). LNCS, vol. 6174, pp. 380–395. Springer (2010)
29. Cheung, L.: Reconciling Nondeterministic and Probabilistic Choices. Ph.D. thesis, Radboud University of Nijmegen (2006)
30. Ciesinski, F., Baier, C.: Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. 3rd Int. Conf. Quantitative Evaluation of Systems (QEST'06). pp. 131–132. IEEE CS Press (2006)
31. Ciesinski, F., Baier, C., Größer, M., Parker, D.: Reduction techniques for model checking Markov decision processes. In: Proc. 5th Int. Conf. Quantitative Evaluation of Systems (QEST'08). pp. 45–54. IEEE CS Press (2008)
32. Ciesinski, F., Größer, M.: On probabilistic computation tree logic. In: Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P., Siegle, M. (eds.) Validation of Stochastic Systems: A Guide to Current Research. LNCS (Tutorial Volume), vol. 2925, pp. 147–188. Springer (2004)
33. Clarke, E., Emerson, A.: Design and synthesis of synchronization skeletons using branching time temporal logic. In: Kozen, D. (ed.) Proc. Workshop Logic of Programs. LNCS, vol. 131, pp. 52–71. Springer (1981)
34. Courcoubetis, C., Yannakakis, M.: Verifying temporal properties of finite state probabilistic programs. In: Proc. 29th Annual Symp. Foundations of Computer Science (FOCS'88). pp. 338–345. IEEE CS Press (1988)
35. Courcoubetis, C., Yannakakis, M.: Markov decision processes and regular events. IEEE Trans. Automatic Control 43(10), 1399–1418 (1998)
36. Daniele, M., Giunchiglia, F., Vardi, M.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D. (eds.) Proc. 11th Int. Conf. Computer Aided Verification (CAV'99). LNCS, vol. 1633, pp. 249–260. Springer (1999)
37. D'Argenio, P., Jeannet, B., Jensen, H., Larsen, K.: Reduction and refinement strategies for probabilistic analysis. In: Hermanns, H., Segala, R. (eds.) Proc. 2nd Joint Int. Workshop Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02). LNCS, vol. 2399, pp. 57–76. Springer (2002)
38. Delahaye, B., Caillaud, B., Legay, A.: Probabilistic contracts: A compositional reasoning methodology for the design of stochastic systems. In: Proc. 10th Int. Conf. Application of Concurrency to System Design (ACSD'10). pp. 223–232. IEEE CS Press (2010)
39. Donaldson, A., Miller, A.: Symmetry reduction for probabilistic model checking using generic representatives. In: Graf, S., Zhang, W. (eds.) Proc. 4th Int. Symp. Automated Technology for Verification and Analysis (ATVA'06). LNCS, vol. 4218, pp. 9–23. Springer (2006)

40. 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)
41. Etessami, K., Yannakakis, M.: Recursive Markov decision processes and recursive stochastic games. In: Caires, L., Italiano, G., Monteiro, L., Palamidessi, C., Yung, M. (eds.) Proc. 32nd Int. Colloq. Automata, Languages and Programming (ICALP'05). LNCS, vol. 3580, pp. 891–903. Springer (2005)
42. Feller, W.: An Introduction to Probability Theory and its Applications. Wiley (1968)
43. Feng, L., Kwiatkowska, M., Parker, D.: Compositional verification of probabilistic systems using learning. In: Proc. 7th Int. Conf. Quantitative Evaluation of Systems (QEST'10). pp. 133–142. IEEE CS Press (2010)
44. Filieri, A., Ghezzi, C., Tamburrelli, G.: Run-time efficient probabilistic model checking. In: Proc. 33rd ACM/IEEE International Conference on Software Engineering (ICSE'11). ACM (2011)
45. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Quantitative multi-objective verification for probabilistic systems. In: Abdulla, P., Leino, K. (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. 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)
47. Giro, S.: On the automatic verification of distributed probabilistic automata with partial information. Ph.D. thesis, FaMAF, Universidad Nacional de Córdoba (2010)
48. van Glabbeek, R., Smolka, S., Steffen, B.: Reactive, generative, and stratified models of probabilistic processes. Information and Computation 121(1), 59–80 (1995)
49. Größer, M., Baier, C.: Partial order reduction for Markov decision processes: A survey. In: de Boer, F., Bonsangue, M., Graf, S., de Roever, W.P. (eds.) Proc. 4th Int. Symp. Formal Methods for Component and Objects (FMCO'05). LNCS, vol. 4111, pp. 408–427. Springer (2006)
50. Hahn, E., Hermanns, H., Wachter, B., Zhang, L.: PASS: Abstraction refinement for infinite probabilistic models. In: Esparza, J., Majumdar, R. (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)
51. Hahn, E., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. In: Pasareanu, C. (ed.) Proc. 16th Int. SPIN Workshop. LNCS, vol. 5578, pp. 88–106. Springer (2009)
52. Han, T., Katoen, J.P., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: Proc. IEEE Symp. Real-Time Systems (RTSS 08). pp. 173–182. IEEE CS Press (2008)
53. Hansson, H.: Time and Probability in Formal Design of Distributed Systems. Elsevier (1994)
54. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects of Computing 6(5), 512–535 (1994)
55. Hermanns, H., Wachter, B., Zhang, L.: Probabilistic CEGAR. In: Gupta, A., Malik, S. (eds.) Proc. 20th Int. Conf. Computer Aided Verification (CAV'08). LNCS, vol. 5123, pp. 162–175. Springer (2008)
56. Hinton, A., Kwiatkowska, M., Norman, G., Parker, D.: PRISM: A tool for automatic verification of probabilistic systems. In: Hermanns, H., Palsberg, J. (eds.)

Proc. 12th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'06). LNCS, vol. 3920, pp. 441–444. Springer (2006)

57. Howard, R.: Dynamic Programming and Markov Processes. The MIT Press (1960)
58. Technical specifications of hard drive IBM Travelstar VP (www.storage.ibm.com/storage/oem/data/travvp.htm)
59. Israeli, A., Jalfon, M.: Token management schemes and random walks yield self-stabilizating mutual exclusion. In: Proc. 9th Annual ACM Symp. Principles of Distributed Computing (PODC'90). pp. 119–131. ACM New York (1990)
60. Jeannet, B., D'Argenio, P., Larsen, K.: Rapture: A tool for verifying Markov decision processes. In: Cerna, I. (ed.) Proc. Tools Day, affiliated to 13th Int. Conf. Concurrency Theory (CONCUR'02). pp. 84–98. Technical Report FIMU-RS-2002-05, Faculty of Informatics, Masaryk University (2002)
61. Katoen, J.P., Hahn, E., Hermanns, H., Jansen, D., Zapreev, I.: 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 CS Press (2009)
62. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: Jones, N., Muller-Olm, M. (eds.) Proc. 10th Int. Conf. Verification, Model Checking, and Abstract Interpretation (VMCAI'09). LNCS, vol. 5403, pp. 182–197. Springer (2009)
63. 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) (2010)
64. Kemeny, J., Snell, J., Knapp, A.: Denumerable Markov Chains. Springer-Verlag, 2nd edn. (1976)
65. Kulkarni, V.: Modeling and Analysis of Stochastic Systems. Chapman & Hall (1995)
66. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Ball, T., Jones, R. (eds.) Proc. 18th Int. Conf. Computer Aided Verification (CAV'06). LNCS, vol. 4114, pp. 234–248. Springer (2006)
67. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic model checking. In: Bernardo, M., Hillston, J. (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)
68. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: Ouaknine, J., Vaandrager, F. (eds.) Proc. 7th Int. Conf. Formal Modelling and Analysis of Timed Systems (FORMATS'09). LNCS, vol. 5813, pp. 212–227. Springer (2009)
69. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Esparza, J., Majumdar, R. (eds.) Proc. 16th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10). LNCS, vol. 6105, pp. 23–37. Springer (2010)
70. Kwiatkowska, M., Norman, G., Parker, D., Sproston, J.: Modeling and Verification of Real-Time Systems: Formalisms and Software Tools, chap. Verification of Real-Time Probabilistic Systems, pp. 249–288. John Wiley & Sons (2008)
71. Kwiatkowska, M., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using Cadence SMV and PRISM. In: Berry, G., Comon, H., Finkel, A. (eds.) Proc. 13th Int. Conf. Computer Aided Verification (CAV'01). LNCS, vol. 2102, pp. 194–206. Springer (2001)
72. Legay, A., Murawski, A., Ouaknine, J., Worrell, J.: On automated verification of probabilistic programs. In: Ramakrishnan, C., Rehof, J. (eds.) Proc. 14th Int. Conf.

Tools and Algorithms for the Construction and Analysis of Systems (TACAS'08). LNCS, vol. 4963, pp. 173–187. Springer (2008)

73. Neuhäußer, M., Zhang, L.: Time-bounded reachability probabilities in continuous-time Markov decision processes. In: Proc. 7th Int. Conf. Quantitative Evaluation of Systems (QEST'10). pp. 209–218. IEEE CS Press (2010)
74. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Using probabilistic model checking for dynamic power management. Formal Aspects of Computing 17(2), 160–176 (2005)
75. Pnueli, A.: The temporal logic of programs. In: Proc. 18th Annual Symp. Foundations of Computer Science (FOCS'77). pp. 46–57. IEEE CS Press (1977)
76. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. John Wiley and Sons (1994)
77. Rabe, M., Schewe, S.: Optimal time-abstract schedulers for CTMDPs and Markov games. In: Di Pierro, A., Norman, G. (eds.) Proc. 8th Workshop Quantitative Aspects of Programming Languages (QAPL'10). EPTCS, vol. 28, pp. 144–158. Open Publishing Association (2010)
78. Rabin, M.: Probabilistic automata. Information and Control 6, 230–245 (1963)
79. Roscoe, A.: The theory and practice of concurrency. Prentice-Hall (1997)
80. Segala, R.: Modelling and Verification of Randomized Distributed Real Time Systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
81. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. Nordic Journal of Computing 2(2), 250–273 (1995)
82. Sokolova, A., de Vink, E.: Probabilistic automata: System types, parallel composition and comparison. In: Baier, C., Haverkort, B., Hermanns, H., Katoen, J.P., Siegle, M. (eds.) Validation of Stochastic Systems: A Guide to Current Research. LNCS (Tutorial Volume), vol. 2925, pp. 1–43. Springer (2004)
83. Stewart, W.: Introduction to the Numerical Solution of Markov Chains. Princeton (1994)
84. Vardi, M.: Automatic verification of probabilistic concurrent finite state programs. In: Proc. 26th Annual Symp. Foundations of Computer Science (FOCS'85). pp. 327–338. IEEE CS Press (1985)
85. Vardi, M., Wolper, P.: Reasoning about infinite computations. Information and Computation 115(1), 1–37 (1994)
86. Zhang, L., Hermanns, H.: Deciding simulations on probabilistic automata. In: Namjoshi, K., Yoneda, T., Higashino, T., Okamura, Y. (eds.) Proc. 5th Int. Symp. Automated Technology for Verification and Analysis (ATVA'07). LNCS, vol. 4762, pp. 207–222. Springer (2007)
87. Zhang, L., Neuhäußer, M.: Model checking interactive Markov chains. In: Esparza, J., Majumdar, R. (eds.) Proc. 16th Int. Conf. Tools and Algorithms for the Construction and Analysis of Systems (TACAS'10). LNCS, vol. 6105, pp. 53–68. Springer (2010)
88. Zhang, L., She, Z., Ratschan, S., Hermanns, H., Hahn, E.: Safety verification for probabilistic hybrid systems. In: Cook, B., Jackson, P., Touili, T. (eds.) Proc. 22nd Int. Conf. Computer Aided Verification (CAV'10). LNCS, vol. 6174, pp. 196–211. Springer (2008)
89. `http://www.prismmodelchecker.org/benchmarks/`
90. `http://www.prismmodelchecker.org/casestudies/`
91. `http://www.prismmodelchecker.org/files/sfm11/`
92. `http://www.prismmodelchecker.org/other-tools.php`