Chapter 28
Model Checking Probabilistic Systems

Christel Baier, Luca de Alfaro, Vojtěch Forejt, and Marta Kwiatkowska

Abstract The model-checking approach was originally formulated for verifying qualitative properties of systems, for example safety and liveness (see Chap. 2), and subsequently extended to also handle quantitative features, such as real time (see Chap. 29), continuous flows (see Chap. 30), as well as stochastic phenomena, where system evolution is governed by a given probability distribution. Probabilistic model checking aims to establish the correctness of probabilistic system models against quantitative probabilistic specifications, such as those capable of expressing, for example, the probability of an unsafe event occurring, expected time to termination, or expected power consumption in the start-up phase. In this chapter, we present the foundations of probabilistic model checking, focusing on finite-state Markov decision processes as models and quantitative properties expressed in probabilistic temporal logic. Markov decision processes can be thought of as a probabilistic variant of labelled transition systems in the following sense: transitions are labelled with actions, which can be chosen nondeterministically, and successor states for the chosen action are specified by means of discrete probabilistic distributions, thus specifying the probability of transiting to each successor state. To reason about expectations, we additionally annotate Markov decision processes with quantitative costs, which are incurred upon taking the selected action from a given state. Quantitative properties are expressed as formulas of the probabilistic computation tree logic (PCTL) or using linear temporal logic (LTL). We summarise the main model-checking algorithms for both PCTL and LTL, and illustrate their working through examples. The chapter ends with a brief overview of extensions to more expressive models and temporal logics, existing probabilistic model-checking tool support, and main application domains.

C. Baier
Technische Universität Dresden, Dresden, Germany

L. de Alfaro
University of California, Santa Cruz, Santa Cruz, CA, USA

V. Forejt · M. Kwiatkowska (✉)
University of Oxford, Oxford, UK
e-mail: marta.kwiatkowska@cs.ox.ac.uk

E.M. Clarke et al. (eds.), *Handbook of Model Checking*,
DOI 10.1007/978-3-319-10575-8_28

28.1 Introduction

Markovian stochastic models, i.e., state-transition graphs annotated with probabilities to model and reason about stochastic phenomena, are central to many applications. Traditionally, purely stochastic models such as Markov chains [96] have been applied in, for example, queueing theory, performance evaluation, and the modelling of telecommunication systems and networks [13, 21, 61], but they are also widely used in other contexts. Dependability properties such as reliability and availability are expressed probabilistically. In systems biology, for example, stochastic models can be used to reason about biological populations and the evolution of concentrations of molecules in biological signalling networks [62]. Probabilistic models with nondeterminism, for example Markov decision processes (abbreviated as MDPs) [99], which are the main focus of this chapter, are central to the modelling of distributed coordination protocols that use randomization for medium access control for wireless networks [85], breaking the symmetry in leader election algorithms [68], or modelling security, anonymity and privacy protocols [90], among many examples. MDPs are also widely used in operations research, economics, robotics, and related disciplines that crucially rely on the concept of decision making so as to choose the next action to optimize a certain goal function. Another application of MDPs is modelling distributed systems that operate with unreliable components. For instance, for systems with communication channels that might corrupt or lose messages, or interact with sensors that deliver wrong values in certain cases, probability distributions can be used to specify the frequency of faulty behaviour. Considering stochastic models more generally, further examples are—ranking algorithms in search engines for the Internet, the analysis of soccer or baseball matches, reasoning about the stochastic growth of waves of influenza or the population dynamics of other pathogenic germs, speech recognition, and signature recognition via biometric identification features. We give a brief overview of related models at the end of this chapter.

28.1.1 Temporal Logics for Specifying Probabilistic Properties

Probabilistic temporal logics arise as generalisations of established temporal logics such as computation tree logic (CTL) and linear temporal logic (LTL). Probabilistic computation tree logic (PCTL) [15, 17, 59] is a probabilistic variant of CTL that replaces the usual path quantifiers, with which one can reason about all or some paths satisfying a certain condition, with operators instead imposing quantitative constraints on the proportion of paths that satisfy this condition. More specifically, PCTL provides a probabilistic operator whose role is to specify lower or upper probability bounds for reachability properties, in the sense of requiring that the probability of reaching a given set of states is above or below a given threshold value. The reachability properties can be constrained using the CTL path modality "until" U or its step-bounded variant $\mathsf{U}^{\leq k}$. For instance, using the probabilistic operator one might formally establish the guarantee that a system failure will occur within the next 100 steps with probability 10^{-8} or less, or that a leader will eventually be elected almost surely, that is, with probability 1. Besides the probability operator, expected cost operators can also be defined, which allow for reasoning, for example, about the average cost to reach a certain set of target states, or the accumulated cost within the next k steps. The cost operators can, for instance, be used to assert that the expected energy consumption within the next 100 steps is less than a given threshold. For Markov decision processes decorated with costs, model checking reduces to the computation of the minimum or maximum probability/expectation values, over the possible resolutions of nondeterminism.

While PCTL is a branching-time logic and its formulas express properties that a state of a probabilistic model might or might not have, probabilistic systems can also be analysed using purely linear-time (path-based) formalisms such as LTL or automata over infinite words [11, 40, 97, 105, 106]. We will restrict our attention to the logic LTL in this chapter. Unlike PCTL, it does not admit path quantifiers, but it allows us to express more elaborate properties, because it is possible to combine temporal operators. One can then, for example, express a path property "whenever button 1 is pressed, the system will be operational until button 2 is pressed". Such a property would not be expressible in PCTL. Since the underlying model is probabilistic, after fixing an LTL formula we are interested in quantitatively reasoning about the proportion of the paths satisfying the specification, analogously to PCTL. For this purpose we introduce *LTL state properties*, which are given by an LTL formula and a probability bound, and are true in a state if the maximum probability of the formula being satisfied is lower than the bound given. The solution methods we present in this chapter also allow us to ask "quantitative" questions, i.e., to directly compute the maximum probability that a given LTL formula is satisfied.

The two ways of reasoning about properties of MDPs which we study in this chapter, i.e., PCTL and LTL state properties, offer different expressive power. Essentially, the properties one can capture are in the same spirit as those in the non-probabilistic variants, and hence we refer the reader to Chap. 2 for a comprehensive overview. As in the non-probabilistic case, the properties expressed using LTL are perhaps easier to obtain from requirements expressed in natural language than PCTL formulas, but PCTL admits better complexity of model-checking algorithms, which are also easier to implement. We note that the two logics, PCTL and LTL, can be combined into a logic PCTL*.

In this chapter we will present the model-checking approach for Markov decision processes (MDPs) [4, 87, 99], which for the purposes of the model-checking algorithms discussed here are equivalent to probabilistic automata due to Segala [101, 102]. MDPs are of fundamental importance in probabilistic verification, since they not only serve as a natural representation of many real-world applications, for example distributed network protocols, but are also key to formulating abstractions for more complex models which incorporate dense real time and probability, such as continuous-time Markovian models and probabilistic variants of timed automata. Both PCTL and LTL can be used for reasoning about qualitative and quantitative properties of MDPs. Several variants of PCTL and LTL have been proposed for the analysis of probabilistic models that rely on a dense time domain. These will be briefly addressed in Sect. 28.9.

28.1.2 Model-Checking Algorithms for Probabilistic Systems

For finite-state Markov decision processes, the quantitative analysis against PCTL or LTL specifications mainly relies on a combination of graph algorithms, automata-based constructions, and (numerical) algorithms for computing the minimum and maximum probabilities and expectation values. Compared to the non-probabilistic case, there is the additional difficulty of solving linear programs, and also the required graph algorithms are more complex. This makes the state space explosion problem even more serious than in the non-probabilistic case, and the feasibility of algorithms for quantitative analysis crucially depends on good heuristics to increase efficiency. Hence, model-checking tools usually implement advanced versions of algorithms we present in this chapter, and use intricate data structures to tackle the state space explosion problem, such as multi-terminal binary decision diagrams [54] and sparse matrices. We give a more detailed overview of the implementation approaches in Sect. 28.7.1.

28.1.3 Outline

The remaining sections of this chapter are organized as follows. Section 28.2 presents the definition of Markov decision processes and explains the main concepts that are relevant for PCTL and LTL model checking. The syntax and semantics of PCTL will be provided in Sect. 28.3. Section 28.4 summarizes the main steps of the PCTL model-checking algorithm for MDPs. Section 28.5 introduces the syntax and semantics of LTL and Sect. 28.6 describes the model-checking algorithm. Section 28.7 gives a brief overview of available tools and interesting case studies; it also mentions outstanding challenges of modelling and verification of probabilistic systems. Section 28.8 summarises related models and logics, and Sect. 28.9 concludes the chapter.

28.2 Modelling Probabilistic Concurrent Systems

Markov decision processes [4, 87, 99], which are similar to probabilistic automata [101, 102], are a convenient representation for distributed or concurrent systems in which the system evolution is described by discrete probabilities. Intuitively, a Markov decision process can be understood as a probabilistic variant of a labelled transition system with transitions and states labelled with action labels and atomic propositions, respectively. For each state s and action α that is enabled in state s, a discrete probability distribution specifies the probabilities for the α-labelled transitions emanating from s. This corresponds to the so-called reactive model in the classification of [55]. In addition, a real-valued cost can be associated with each state s and action α, representing the price one has to pay whenever executing action α in state s. Dually, the cost assigned to (s, α) can also be viewed as a reward that is earned when firing action α in s. To keep the presentation simple, in this chapter we restrict ourselves to cost functions whose range is the non-negative integers. Furthermore, we assume that all transition probabilities in the MDP are rational.

28.2.1 Preliminaries

Let X be a countable set. A *(probability) distribution* on X denotes a function $\mathsf{D} : X \to [0, 1]$ such that

$$\sum_{x \in X} \mathsf{D}(x) = 1.$$

The set $\mathsf{Supp}(\mathsf{D}) \overset{\text{def}}{=} \{x \in X : \mathsf{D}(x) \neq 0\}$ is called the support of D. A distribution D is *Dirac* if its support is a singleton. We write $\mathsf{Distr}(X)$ to denote the set of all distributions on X.

As usual, $\mathbb{N}$ denotes the set of natural numbers $0, 1, 2, \ldots$ and $\mathbb{Q}$ the set of rational numbers.

28.2.2 Markov Decision Processes

A *Markov decision process* is a tuple $\mathscr{M} = (S, \mathsf{Act}, \mathsf{P}, s_{\mathsf{init}}, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ where

- S is a countable non-empty set of *states*,
- Act is a finite non-empty set of *actions*,
- $\mathsf{P} : S \times \mathsf{Act} \times S \to [0, 1] \cap \mathbb{Q}$ is the *transition probability function* such that

$$\sum_{s' \in S} \mathsf{P}\big(s, \alpha, s'\big) \in \{0, 1\} \quad \text{for all states } s \in S \text{ and actions } \alpha \in \mathsf{Act},$$

- $s_{\mathsf{init}} \in S$ is the *initial state*,
- AP is a finite set of *atomic propositions*,
- $\mathsf{L} : S \to 2^{\mathsf{AP}}$ is a *labelling function* that labels a state s with those atomic propositions in AP that are supposed to hold in s,
- $\mathsf{C} : S \times \mathsf{Act} \to \mathbb{N}$ is a *cost function*.

$\mathcal{M}$ is called finite if the state space S and the set of actions Act are finite. In this chapter we assume that all MDPs are finite, unless specified otherwise. If $s \in S$ then $\mathsf{Act}(s)$ denotes the set of actions that are *enabled* in state s, i.e.

$$\mathsf{Act}(s) \overset{\text{def}}{=} \{\alpha \in \mathsf{Act} : \mathsf{P}(s, \alpha, s') > 0 \text{ for some } s' \in S\}.$$

For technical reasons, we suppose that there are no terminal states, i.e., for each state $s \in S$ the set $\mathsf{Act}(s)$ is non-empty. Furthermore, we require that $\mathsf{C}(s, \alpha) = 0$ if α is an action that is not enabled in s, i.e., if $\alpha \notin \mathsf{Act}(s)$.

The intuitive operational behaviour of an MDP can be described as follows. The MDP starts its computation in the initial state s_{init}. If after n steps the current state is s_n then, first, an enabled action $\alpha_{n+1} \in \mathsf{Act}(s_n)$ is chosen nondeterministically. Firing α_{n+1} in state s_n incurs the cost $\mathsf{C}(s_n, \alpha_{n+1})$. The effect of taking action α_{n+1} in state s_n is given by the distribution $\mathsf{P}(s_n, \alpha_{n+1}, \cdot)$. The next state s_{n+1} belongs to the support of $\mathsf{P}(s_n, \alpha_{n+1}, \cdot)$ and is chosen probabilistically. The resulting infinite sequence of states and actions $\pi = s_0\, \alpha_1\, s_1\, \alpha_2\, s_2\, \alpha_3 \ldots \in (S \times \mathsf{Act})^\omega$ is called an (infinite) path of $\mathcal{M}$. More generally, any alternating sequence $\pi = s_0\, \alpha_1\, s_1\, \alpha_2\, s_2\, \alpha_3 \ldots \in (S \times \mathsf{Act})^\omega$, with $\mathsf{P}(s_n, \alpha_{n+1}, s_{n+1}) > 0$ for all $n \geq 0$, is called a *path* of state s_0, and will be written in the form

$$\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$$

$\mathsf{Paths}^{\mathcal{M}}(s)$, or for short $\mathsf{Paths}(s)$, denotes the set of all paths of $\mathcal{M}$ starting in state s, and $\mathsf{Paths}^{\mathcal{M}}$, or Paths, denotes the set of all paths. If π is as above then $\pi{\uparrow^n}$ denotes the infinite suffix of π that starts in the $(n+1)$-th state s_n, i.e. for the above π we have

$$\pi{\uparrow^n} \overset{\text{def}}{=} s_n \xrightarrow{\alpha_{n+1}} s_{n+1} \xrightarrow{\alpha_{n+2}} s_{n+2} \xrightarrow{\alpha_{n+3}} \ldots$$

Similarly, $\pi{\downarrow_n}$ denotes the finite prefix that ends in s_n, i.e.,

$$\pi{\downarrow_n} \overset{\text{def}}{=} s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots \xrightarrow{\alpha_n} s_n.$$

We refer to the finite prefixes of (infinite) paths as *finite paths* and denote the set of finite paths starting in state s by $\mathsf{FinPaths}^{\mathcal{M}}(s)$, or for short $\mathsf{FinPaths}(s)$, and we denote the set of all finite paths by $\mathsf{FinPaths}^{\mathcal{M}}$ or $\mathsf{FinPaths}$. The length of a finite path ς is given by the number of transitions taken in ς and denoted by $|\varsigma|$; the length of an infinite path is ω. We use the notation $\mathsf{last}(\varsigma)$ for the last state of a finite path ς. Similarly, $\mathsf{first}(\cdot)$ is used to refer to the first state of a finite or infinite path. The $(n+1)$-th state of a path is denoted by $\pi[n]$. Thus, if π is as above then $\pi[0] = \mathsf{first}(\pi) = s_0$, $|\pi{\downarrow_n}| = n$ and $\pi[n] = \mathsf{first}(\pi{\uparrow^n}) = \mathsf{last}(\pi{\downarrow_n}) = s_n$ for all $n \in \mathbb{N}$.

Given a finite path $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n$, the *total* or *cumulated cost* of ς is defined by

$$\mathsf{cost}(\varsigma) \overset{\text{def}}{=} \sum_{i=1}^{n} \mathsf{C}(s_{i-1}, \alpha_i).$$

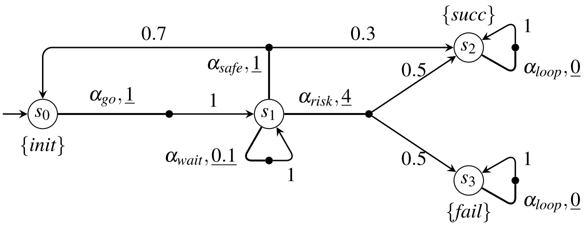
0.7
0.3
$\{succ\}$
1
$\alpha_{safe}, \underline{1}$
s_2
0.5
$\alpha_{loop}, \underline{0}$
$\alpha_{go}, \underline{1}$
1
$\alpha_{risk}, \underline{4}$
s_0
s_1
$\{init\}$
$\alpha_{wait}, \underline{0.1}$
1
0.5
1
s_3
$\{fail\}$
$\alpha_{loop}, \underline{0}$

Fig. 1 A running example of a Markov decision process annotated with costs

In addition to the cost function $\mathsf{C}(s,\alpha)$ that assigns values to the pairs consisting of a state and an enabled action, one can also define cost functions just for the states $\mathsf{C}_{\mathsf{st}} : S \to \mathbb{N}$, with the intuitive meaning that each visit to state s incurs the cost $\mathsf{C}_{\mathsf{st}}(s)$. Such cost functions are supported, for example, by the tool PRISM (see Sect. 28.7), but are omitted here since they can be encoded in the variant of MDPs presented in this chapter. If a cost function C_{st} for the states, rather than for pairs of states and actions, is given, then we might switch from C_{st} to $\mathsf{C} : S \times \mathsf{Act} \to \mathbb{N}$ as follows

$$\mathsf{C}(s,\alpha) \overset{\text{def}}{=} \begin{cases} \mathsf{C}_{\mathsf{st}}(s) & \text{if } \alpha \in \mathsf{Act}(s) \\ 0 & \text{otherwise} \end{cases}$$

to meet the syntax of the MDP definition. Given an MDP as defined in Sect. 28.2.2 and an additional cost function $\mathsf{C}_{\mathsf{st}} : S \to \mathbb{N}$ that specifies the cost incurred upon visiting state s, the effect of C and C_{st} can be mimicked by using the single cost function $\mathsf{C}' : S \times \mathsf{Act} \to \mathbb{N}$ given by

$$\mathsf{C}'(s,\alpha) \overset{\text{def}}{=} \mathsf{C}_{\mathsf{st}}(s) + \mathsf{C}(s,\alpha).$$

Example 1 (Running Example) Consider the MDP $\mathscr{M} = (S, \mathsf{Act}, \mathsf{P}, s_0, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ from Fig. 1. The MDP models a simple system in which, after some initial step, two kinds of decisions can be taken. One results in success with relatively high probability, but can fail completely, and another gives a smaller probability of immediate success, but cannot result in a non-recoverable failure. Formally, $S = \{s_0, s_1, s_2, s_3\}$, $\mathsf{Act} = \{\alpha_{go}, \alpha_{wait}, \alpha_{safe}, \alpha_{risk}, \alpha_{loop}\}$, and P is as given by the numbers on arrows originating from the dots, e.g., $\mathsf{P}(s_1, \alpha_{safe}, s_0) = 0.7$. Atomic propositions are $\{init, succ, fail\}$, where the labels of states are as shown in the picture, e.g., $\mathsf{L}(s_0) = \{init\}$. Costs of the actions are shown in the picture as underlined numbers, e.g., $\mathsf{C}(s_1, \alpha_{wait}) = 0.1$.

Observe that there is a non-trivial choice of an action only in the state s_1, where one can choose between α_{wait}, α_{safe} and α_{risk}. Consider the path

$$\pi = s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{safe}} s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{risk}} s_2 \xrightarrow{\alpha_{loop}} s_2 \xrightarrow{\alpha_{loop}} \cdots .$$

We have $\pi\uparrow^2 = s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{risk}} s_2 \xrightarrow{\alpha_{loop}} s_2 \xrightarrow{\alpha_{loop}} \cdots$ and $\pi\downarrow_2 = s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{safe}} s_0$. For the finite path $\pi\downarrow_2$ we have that the total or cumulated cost $\mathsf{cost}(\pi\downarrow_2) = \mathsf{C}(s_0, \alpha_{go}) + \mathsf{C}(s_1, \alpha_{safe}) = 2$. □

28.2.3 Markov Chains

Markov chains can be viewed as special instances of Markov decision processes, where in each state exactly one action is enabled. Thus, there are no nondeterministic choices in a Markov chain and the operational behaviour is purely probabilistic. Since, in the above definition of an MDP, the actions are used just to name the nondeterministic alternatives and group together probabilistic transitions that belong to the same alternative, the concept of actions is irrelevant for Markov chains. Thus, the transition probabilities of a Markov chain $\mathscr{C}$ can be specified by a function $\mathsf{P}^{\mathscr{C}} : S \times S \to [0, 1]$. Paths are then just sequences $s_0\, s_1\, s_2 \ldots$ of states such that

$$\mathsf{P}^{\mathscr{C}}(s_i, s_{i+1}) > 0 \quad \text{for all } i \geq 0.$$

Using standard concepts of measure and probability theory, any Markov chain naturally induces a *probability space*, i.e., a triple consisting of the set of *outcomes* Ω, the set of *events* $\mathscr{F} \subseteq 2^{\Omega}$ which contains $\emptyset$ and is closed under complements and countable unions, and a *probability measure* $\mathsf{Pr} : \mathscr{F} \to [0, 1]$ which is countably additive and satisfies $\mathsf{Pr}(\Omega) = 1$. More concretely, in the induced probability space the outcomes are the (infinite) paths and the events can be understood as linear-time properties, i.e., conditions that an infinite path might satisfy or not (indeed, all LTL formulas, PCTL path formulas, and even all ω-regular languages over sets of atomic propositions specify measurable sets of paths [40, 105]). For details we refer to textbooks on Markov chains and probability theory, see, for example, [50, 75, 77], and just sketch the main ideas. The underlying σ-algebra is the smallest σ-algebra that contains the *cylinder sets*, namely, the sets containing all paths that have a common prefix, i.e., the sets

$$\mathsf{Cyl}(\varsigma) \stackrel{\text{def}}{=} \left\{\pi \in \mathsf{Paths}^{\mathscr{C}} : \varsigma \text{ is a prefix of } \pi\right\}$$

for all finite paths ς in $\mathscr{C}$. Using Carathéodory's measure extension theorem [7], the probability measure $\mathsf{Pr}^{\mathscr{C}}$ is the unique probability measure on the σ-algebra such that for each finite path $\varsigma = s_0\, s_1\, s_2\, \ldots\, s_n$ starting in $\mathscr{C}$'s initial state $s_0 = s_{\mathsf{init}}$ we have:

$$\mathsf{Pr}^{\mathscr{C}}\big(\mathsf{Cyl}(\varsigma)\big) = \mathsf{P}^{\mathscr{C}}(s_0, s_1) \cdot \mathsf{P}^{\mathscr{C}}(s_1, s_2) \cdot \ldots \cdot \mathsf{P}^{\mathscr{C}}(s_{n-1}, s_n).$$

If ς is a finite path that does not start in the initial state then $\mathsf{Pr}^{\mathscr{C}}(\mathsf{Cyl}(\varsigma)) = 0$.

28.2.4 Schedulers

Reasoning about probabilities in an MDP relies on a *decision-making* approach that resolves the nondeterministic choices—answering the question which action will be performed in the current state—and turns an MDP into an infinite tree-like Markov chain. We give here just a brief summary of the main concepts. Details can be found in any textbook on Markov decision processes, e.g., [99].

The decision-making approach can be formalized with the help of the mathematical notion of a *scheduler*, often called *policy* or *adversary*. Intuitively, a scheduler takes as input the "history" of a computation—namely, a finite path ς—and chooses the next action according to some distribution. Formally, a *history-dependent randomized* scheduler, for short called a scheduler, is a function

$$\mathcal{U} : \mathsf{FinPaths}^{\mathcal{M}} \to \mathsf{Distr}(\mathsf{Act})$$

such that $\mathsf{Supp}(\mathcal{U}(\varsigma)) \subseteq \mathsf{Act}(\mathsf{last}(\varsigma))$ for all finite paths ς. A (finite or infinite) path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \ldots$ is said to be a $\mathcal{U}$-path, if

$$\mathcal{U}(s_0 \xrightarrow{\alpha_1} \ldots \xrightarrow{\alpha_i} s_i)(\alpha_{i+1}) > 0 \quad \text{for all } 0 \le i < |\pi|.$$

A scheduler $\mathcal{U}$ is called *deterministic* if $\mathcal{U}(\varsigma)$ is a Dirac distribution for all finite paths ς, i.e., for each finite path ς there is some action α with $\mathcal{U}(\varsigma)(\alpha) = 1$, and $\mathcal{U}(\varsigma)(\beta) = 0$ for all actions $\beta \in \mathsf{Act} \setminus \{\alpha\}$. Scheduler $\mathcal{U}$ is called *memoryless* if

$$\mathcal{U}(\varsigma) = \mathcal{U}\bigl(\varsigma'\bigr) \text{ for all finite paths } \varsigma, \varsigma' \text{ such that } \mathsf{last}(\varsigma) = \mathsf{last}\bigl(\varsigma'\bigr).$$

Deterministic schedulers are given as functions $\mathcal{U} : \mathsf{FinPaths}^{\mathcal{M}} \to \mathsf{Act}$. Memoryless randomized schedulers can be viewed as functions $\mathcal{U} : S \to \mathsf{Distr}(\mathsf{Act})$. Memoryless deterministic schedulers, also called *simple schedulers*, are specified as functions $\mathcal{U} : S \to \mathsf{Act}$. We write Sched to denote the set of all schedulers.

28.2.5 Probability Measures in MDPs

Given an MDP $\mathcal{M}$ and a scheduler $\mathcal{U}$, the behaviour of $\mathcal{M}$ under $\mathcal{U}$ can be formalized by an infinite-state tree-like Markov chain $\mathcal{C} = \mathcal{M}|_{\mathcal{U}}$. The states of that Markov chain represent the finite $\mathcal{U}$-paths. The successor states of

$$\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n$$

have the form $\varsigma' = \varsigma \xrightarrow{\beta} s$ and the transition probability for moving from ς to ς' is given by

$$\mathcal{U}(\varsigma)(\beta) \cdot \mathsf{P}(s_n, \beta, s).$$

We write $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}$, or for short $\mathsf{Pr}^{\mathcal{U}}$, to denote the standard probability measure $\mathsf{Pr}^{\mathcal{C}}$ on that Markov chain. Thus, the probability measure $\mathsf{Pr}^{\mathcal{U}}$ for a given scheduler $\mathcal{U}$ is the unique probability measure on the σ-algebra generated by the finite $\mathcal{U}$-paths such that

$$\mathsf{Pr}^{\mathcal{U}}\bigl(\mathsf{Cyl}(\varsigma)\bigr) = \prod_{i=1}^{n} \mathcal{U}(\varsigma\downarrow_{i-1})(\alpha_i) \cdot \mathsf{P}(s_{i-1}, \alpha_i, s_i)$$

if $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n$ is a $\mathcal{U}$-path starting in $s_0 = s_{\mathsf{init}}$.

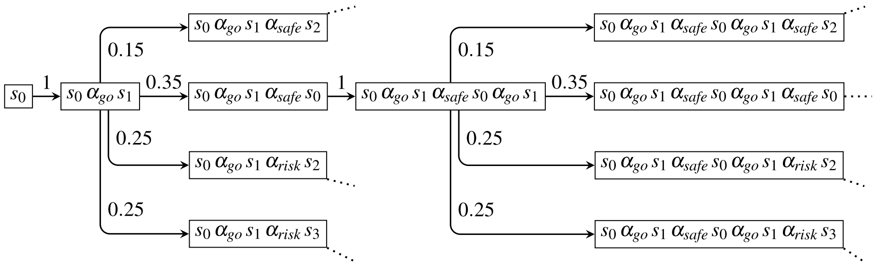
s_0
1
$s_0\,\alpha_{go}\,s_1$
0.15
0.35
0.25
0.25
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_2$
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0$
$s_0\,\alpha_{go}\,s_1\,\alpha_{risk}\,s_2$
$s_0\,\alpha_{go}\,s_1\,\alpha_{risk}\,s_3$
1
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0\,\alpha_{go}\,s_1$
0.15
0.35
0.25
0.25
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_2$
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0$
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0\,\alpha_{go}\,s_1\,\alpha_{risk}\,s_2$
$s_0\,\alpha_{go}\,s_1\,\alpha_{safe}\,s_0\,\alpha_{go}\,s_1\,\alpha_{risk}\,s_3$

Fig. 2 A Markov chain for the running example and the scheduler from Example 2

Given a state s of $\mathscr{M}$, we denote by $\mathsf{Pr}_s^{\mathscr{U}}$ the probability measure that is obtained by $\mathscr{U}$ viewed as a scheduler for the MDP $\mathscr{M}_s$ that agrees with $\mathscr{M}$, except that s is the unique initial state of $\mathscr{M}_s$. That is, if $\mathscr{M} = (S, \mathsf{Act}, \mathsf{P}, s_{\mathsf{init}}, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ then $\mathscr{M}_s = (S, \mathsf{Act}, \mathsf{P}, s, \mathsf{AP}, \mathsf{L}, \mathsf{C})$. Note that if $\mathscr{U}$ is a deterministic scheduler then

$$\mathsf{Pr}_s^{\mathscr{U}}\big(\mathsf{Cyl}(\varsigma)\big) = \prod_{i=1}^{n} \mathsf{P}(s_{i-1}, \alpha_i, s_i)$$

if $\varsigma = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots \xrightarrow{\alpha_n} s_n$ is a $\mathscr{U}$-path with $\mathsf{first}(\varsigma) = s_0 = s$. Given an MDP $\mathscr{M}$, a scheduler $\mathscr{U}$ and a measurable path property E, then

$$\mathsf{Pr}_s^{\mathscr{U}}(E) \overset{\text{def}}{=} \mathsf{Pr}_s^{\mathscr{U}}\big\{\pi \in \mathsf{Paths}^{\mathscr{M}} \,\big|\, \pi \text{ satisfies } E\big\}$$

denotes the probability that the path property E holds in $\mathscr{M}$ when starting in s and using scheduler $\mathscr{U}$ to resolve the nondeterministic choices.

Example 2 Consider again the MDP $\mathscr{M}$ from Fig. 1, together with the scheduler $\mathscr{U}$ that for every path ending in s_1 picks the action α_{safe} or α_{risk}, both with probability 0.5. This scheduler is memoryless, but not deterministic, and gives rise to the Markov chain $\mathscr{M}|_{\mathscr{U}}$ whose initial fragment is drawn in Fig. 2. For the finite path $\pi = s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{safe}} s_0$ we have

$$\begin{aligned}\mathsf{Pr}^{\mathscr{U}}\big(\mathsf{Cyl}(\pi)\big) &= \mathscr{U}(s_0)(\alpha_{go}) \cdot \mathsf{P}(s_0, \alpha_{go}, s_1) \cdot \mathscr{U}(s_0 \xrightarrow{\alpha_{go}} s_1)(\alpha_{safe}) \cdot \mathsf{P}(s_1, \alpha_{safe}, s_0) \\ &= 1 \cdot 1 \cdot 0.5 \cdot 0.7 = 0.35,\end{aligned}$$

and for the set of paths R which never reach s_2 or s_3 we have $\mathsf{Pr}^{\mathscr{U}}(R) = 0$. □

28.2.6 Maximal and Minimal Probabilities for Path Events

A typical task for the quantitative analysis of an MDP is to compute minimal or maximal probabilities for some given property E when ranging over all schedulers.

If s is a state in $\mathcal{M}$ then we define

$$\mathsf{Pr}_s^{\max}(E) \stackrel{\text{def}}{=} \sup_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}_s^{\mathcal{U}}(E) \quad \text{and} \quad \mathsf{Pr}_s^{\min}(E) \stackrel{\text{def}}{=} \inf_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}_s^{\mathcal{U}}(E).$$

This corresponds to the worst- or best-case analysis of an MDP. If, for example, E stands for the undesired behaviours then E is guaranteed not to hold with probability at least $1 - \mathsf{Pr}_s^{\max}(E)$ under all schedulers, that is, even for the worst-case resolution of the nondeterministic choices. For instance, many relevant properties fall under the class of *reachability probabilities* where one has to establish a lower bound for the minimal probability to reach a certain set F of "good" target states, possibly with some side-constraints on the cumulated cost until an F-state has been reached.

28.2.7 Maximal and Minimal Expected Cost

Another typical task for analysing an MDP against cost-based properties is to compute the minimal or maximal *expected cumulated cost* with respect to certain objectives. For reachability objectives, we consider a set F of target states. Given a path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$, we write $\pi \models \Diamond F$ if and only if π eventually visits F, i.e., there is an i such that $s_i \in F$. The cumulated cost of π to reach F is defined as follows. If $\pi \models \Diamond F$ then

$$\mathsf{cost}[\Diamond F](\pi) = \mathsf{cost}(\pi{\downarrow_n}) = \sum_{i=1}^{n} \mathsf{C}(s_{i-1}, \alpha_i)$$

where $s_n \in F$ and $\{s_i : 0 \leq i < n\} \cap F = \varnothing$. If π never visits a state in F then $\mathsf{cost}[\Diamond F](\pi)$ is defined as ∞, irrespective of whether only finitely many actions in π have nonzero cost (in which case the total cost of π would be finite). Given a scheduler $\mathcal{U}$ for $\mathcal{M}$ and a state s in $\mathcal{M}$, the expected cumulated cost for reaching F from s, denoted $\mathsf{Ex}_s^{\mathcal{U}}(\mathsf{cost}[\Diamond F])$, is the expected value of the random variable $\pi \mapsto \mathsf{cost}[\Diamond F](\pi)$ in the stochastic process (i.e., the Markov chain) induced by $\mathcal{U}$.

- If $\mathsf{Pr}_s^{\mathcal{U}}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond F\}) = 1$ then

  $$\mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\Diamond F]\big) = \sum_{\varsigma} \mathsf{Pr}_s^{\mathcal{U}}\big(\mathsf{Cyl}(\varsigma)\big) \cdot \mathsf{cost}(\varsigma)$$

  where the sum is taken over all finite $\mathcal{U}$-paths ς with $\mathsf{first}(\varsigma) = s$ and $\mathsf{last}(\varsigma) \in F$, while all other states of ς are in $S \setminus F$.
- If $\mathsf{Pr}_s^{\mathcal{U}}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond F\}) < 1$ then with positive probability $\mathcal{U}$ schedules paths that never visit F. Since the total cost of such paths is infinite, we have $\mathsf{Ex}_s^{\mathcal{U}}(\mathsf{cost}[\Diamond F]) = \infty$.

The extremal expected cumulated cost for reaching F is then obtained by

$$\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[\Diamond F]\big) \overset{\text{def}}{=} \sup_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\Diamond F]\big)$$

$$\mathsf{Ex}_s^{\min}\big(\mathsf{cost}[\Diamond F]\big) \overset{\text{def}}{=} \inf_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\Diamond F]\big).$$

Note that $\mathsf{Ex}_s^{\max}(\mathsf{cost}[\Diamond F]) = \infty$ if $\mathsf{Pr}_s^{\min}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond F\}) < 1$; the other direction also holds, i.e., $\mathsf{Pr}_s^{\min}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond F\}) = 1$ implies that $\mathsf{Ex}_s^{\max}(\mathsf{cost}[\Diamond F])$ is finite, although the proof is not as obvious.

Similarly, minimal and maximal expected cost for other objectives can be defined. If an MDP is used as a discrete-time model then one might be interested in the average cost within certain time intervals. This, for instance, permits us to establish lower or upper bounds on the expected power consumption over one time unit. For the *cost cumulated up to time point k* we use the random variable $\pi \mapsto \mathsf{cost}[\leq k](\pi)$ that assigns to each path the total cost for the first k steps, i.e., if $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \ldots$ then

$$\mathsf{cost}[\leq k](\pi) \overset{\text{def}}{=} \sum_{i=1}^{k} \mathsf{C}(s_{i-1}, \alpha_i).$$

Let $\mathsf{Ex}_s^{\mathcal{U}}(\mathsf{cost}[\leq k])$ denote the expected value of the random variable $\mathsf{cost}[\leq k]$ under scheduler $\mathcal{U}$ in the MDP $\mathcal{M}_s$, i.e.,

$$\mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\leq k]\big) = \sum_{\varsigma} \mathsf{Pr}_s^{\mathcal{U}}\big(\mathsf{Cyl}(\varsigma)\big) \cdot \mathsf{cost}(\varsigma)$$

where the sum is taken over all finite $\mathcal{U}$-paths ς of length k starting in state s. The supremum and infimum over all schedulers yields the extremal cumulated costs within the first k steps

$$\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[\leq k]\big) \overset{\text{def}}{=} \sup_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\leq k]\big)$$

$$\mathsf{Ex}_s^{\min}\big(\mathsf{cost}[\leq k]\big) \overset{\text{def}}{=} \inf_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[\leq k]\big).$$

When we specify costs for the states by the function $\mathsf{C}_{\mathsf{st}} : S \to \mathbb{N}$, then it is also possible to reason about *instantaneous costs* in the k-th step. This can be defined with the random variable $\pi \mapsto \mathsf{cost}[=k](\pi)$ that assigns to each path π the cost associated with the k-th action of π. If $\mathsf{Ex}_s^{\mathcal{U}}(\mathsf{cost}[=k])$ denotes the expected value of random variable $\mathsf{cost}[=k]$ under scheduler $\mathcal{U}$ then

$$\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[=k]\big) = \sup_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[=k]\big)$$

$$\mathsf{Ex}_s^{\min}\big(\mathsf{cost}[=k]\big) = \inf_{\mathcal{U}\in\mathsf{Sched}} \mathsf{Ex}_s^{\mathcal{U}}\big(\mathsf{cost}[=k]\big)$$

stand for the extremal average instantaneous costs incurred at the k-th step. These values can be of interest, for example, when reasoning about the minimal or maximal expected queue size at some time point k. For this purpose, we work with the cost function $\mathsf{C}(t, \alpha) = \mathsf{C}_{\mathsf{st}}(t)$ for all actions α that are enabled in state t, where $\mathsf{C}_{\mathsf{st}}(t)$ denotes the current queue size in state t.

Example 3 Let us return to our running example from Fig. 1, and for clarity of notation write just s instead of the singleton set $\{s\}$. We have that the maximal probability of reaching s_3, i.e., $\mathsf{Pr}^{\max}_{s_0}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond s_3\})$, is equal to 0.5. A (deterministic) scheduler that always chooses α_{risk} in paths ending with s_1 witnesses that $\mathsf{Pr}^{\max}_{s_0}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond s_3\}) \geq 0.5$; to see that this probability cannot be higher, observe that upon taking α_{risk} half of the paths transition to s_2, and both s_2 and s_3 have self-loops. On the other hand, $\mathsf{Pr}^{\min}_{s_0}(\{\pi \in \mathsf{Paths} \mid \pi \models \Diamond s_3\}) = 0$, as witnessed by the scheduler that never chooses α_{risk} with nonzero probability.

For maximal expected cost, let us consider a single target state s_2. We have $\mathsf{Ex}^{\max}_{s_0}(\mathsf{cost}[\Diamond s_2]) = \infty$, because there exists a scheduler that with nonzero probability does not reach s_2. For minimal expected cost $\mathsf{Ex}^{\min}_{s_0}(\mathsf{cost}[\Diamond s_2])$, we obtain the value equal to $\frac{20}{3}$, as witnessed by the scheduler that always chooses α_{safe}; to see that no scheduler can yield a better value is a simple exercise.

As an example of instantaneous cost, let us analyse the value $\mathsf{Ex}^{\max}_{s_0}(\mathsf{cost}[=3])$. It is equal to 4, which can be seen by considering a scheduler that picks α_{wait} in $s_0 \xrightarrow{\alpha_{go}} s_1$, and α_{risk} in $s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{wait}} s_1$. This is also the maximal value, because there is in fact no higher cost in the MDP.

28.3 Probabilistic Computation Tree Logic

In this section we present the syntax and semantics of Probabilistic Computation Tree Logic (PCTL), which is a probabilistic counterpart of the well-known logic CTL, introduced in Chap. 2. Formulas of this logic aim to express quantitative probabilistic properties such as "with probability at least 0.99, if we reach a bad state, we can recover with nonzero probability". PCTL is a widely used specification language in many contexts, including verification of purely probabilistic systems or systems with probability as well as nondeterminism, and for both finite- and infinite-state probabilistic systems [15, 22, 81]. Our presentation will focus on the logic PCTL interpreted over finite-state Markov decision processes.

28.3.1 Syntax of PCTL

As in CTL, the syntax of PCTL has two levels: one for the state formulas (denoted by uppercase Greek letters Φ, Ψ) and one for the path formulas (denoted by lower-case Greek letters φ, ψ). The abstract syntax of state and path formulas is as follows

$$\Phi ::= \mathsf{tt} \mid a \mid \Phi_1 \wedge \Phi_2 \mid \neg\Phi \mid \mathbb{P}_{\sim p}(\varphi) \mid \mathbb{E}_{\sim c}(\Diamond\Phi) \mid \mathbb{E}_{\sim c}(\leq k) \mid \mathbb{E}_{\sim c}(=k)$$

$$\varphi ::= \bigcirc\Phi \mid \Phi_1 \mathsf{U} \Phi_2 \mid \Phi_1 \mathsf{U}^{\sim c} \Phi_2$$

where tt stands for the constant truth value "true" and a is a state predicate, i.e., an atomic proposition in AP. The other symbols are explained below.

The operators $\mathbb{P}_{\sim p}(\cdot)$ and $\mathbb{E}_{\sim c}(\cdot)$ are called the *probability* and *expectation* operators. The subscripts $\sim p$ and $\sim c$ specify strict or non-strict lower or upper bounds for probabilities or costs, respectively. Formally, $\sim$ is a comparison operator $\leq$, $<$, $\geq$ or $>$, $p \in [0, 1] \cap \mathbb{Q}$ a rational threshold for probabilities, and $c \in \mathbb{N}$ a non-negative integer that serves as a lower or upper bound for cumulated or instantaneous cost.

The PCTL state formula $\mathbb{P}_{\sim p}(\varphi)$ asserts that, under all schedulers, the probability for the event expressed by the path formula φ meets the bound specified by $\sim p$. Thus, the probability operator imposes a condition on the probability measures $\Pr_s^{\mathcal{U}}$ for all schedulers $\mathcal{U}$. The probability bounds "$\sim p$" can be understood as quantitative counterparts to the CTL path quantifiers $\exists$ and $\forall$. Intuitively, the lower probability bounds $\geq p$ (with $p > 0$) or $> p$ (with $p \geq 0$) can be understood as the quantitative counterpart to existential path quantification. (See also Remark 1.)

As in CTL, path formulas are built from one of the temporal modalities $\bigcirc$ (next) or U (until), where the arguments of the modalities are state formulas. No Boolean connectors or nesting of temporal modalities are allowed in the syntax of path formulas. In addition to the standard until-operator, the above syntax for path formulas includes a cost-bounded version of until.[1] The intuitive meaning of the path formula $\Phi_1 \mathsf{U}^{\sim c} \Phi_2$ is that a Φ_2-state (i.e., some state where Φ_2 holds) will be reached from the current state along a finite path ς that yields a witness of minimal length for the path formula $\Phi_1 \mathsf{U} \Phi_2$ (i.e., ς ends in a Φ_2-state and all other states satisfy the formula $\Phi_1 \wedge \neg\Phi_2$) and where the total cost of ς meets the constraint $\sim c$.

The expectation operator $\mathbb{E}_{\sim c}(\cdot)$ enables the specification of lower or upper bounds for the expected cumulated or instantaneous cost. The state formula $\mathbb{E}_{\sim c}(\Diamond\Phi)$ holds if the expected cumulated cost until a Φ-state is reached meets the requirement given by "$\sim c$" under all schedulers. Similarly, the state formulas $\mathbb{E}_{\sim c}(\leq k)$ and $\mathbb{E}_{\sim c}(=k)$ assert that the cost accumulated in the first k steps and the instantaneous cost at the k-th step, respectively, belong to the interval specified by "$\sim c$".

28.3.2 Semantics of PCTL

Given an MDP, the satisfaction relation $\models$ for state and path formulas is formally defined below, in accordance with the above intuitive semantics. Let $\mathcal{M}$ be an MDP

[1] We did not introduce the step-bounded version of the until operator. This, however, can be derived using the cost-bounded until operator and changing the MDP to the one with unit cost, i.e., $\mathsf{C}(s, \alpha) = 1$ for all states s and actions $\alpha \in \mathsf{Act}(s)$.

as in Sect. 28.2.2 and s a state in $\mathscr{M}$.

$$\begin{array}{lll}
s \models \mathsf{tt} & & \\
s \models a & \text{iff} & a \in \mathsf{L}(s) \\
s \models \Phi_1 \wedge \Phi_2 & \text{iff} & s \models \Phi_1 \text{ and } s \models \Phi_2 \\
s \models \neg\Phi & \text{iff} & s \not\models \Phi \\
s \models \mathbb{P}_{\sim p}(\varphi) & \text{iff} & \mathsf{Pr}_s^{\mathscr{U}}(\varphi) \sim p \text{ for all schedulers } \mathscr{U} \\
 & & \text{where } \mathsf{Pr}_s^{\mathscr{U}}(\varphi) \overset{\text{def}}{=} \mathsf{Pr}_s^{\mathscr{U}}\{\pi \in \mathsf{Paths} \mid \pi \models \varphi\} \\
s \models \mathbb{E}_{\sim c}(\lozenge\Phi) & \text{iff} & \mathsf{Ex}_s^{\mathscr{U}}(\mathsf{cost}[\lozenge\mathsf{Sat}(\Phi)]) \sim c \text{ for all schedulers } \mathscr{U} \\
 & & \text{where } \mathsf{Sat}(\Phi) \overset{\text{def}}{=} \{s \in S \mid s \models \Phi\} \\
s \models \mathbb{E}_{\sim c}(\leq k) & \text{iff} & \mathsf{Ex}_s^{\mathscr{U}}(\mathsf{cost}[\leq k]) \sim c \text{ for all schedulers } \mathscr{U} \\
s \models \mathbb{E}_{\sim c}(= k) & \text{iff} & \mathsf{Ex}_s^{\mathscr{U}}(\mathsf{cost}[= k]) \sim c \text{ for all schedulers } \mathscr{U}
\end{array}$$

MDP $\mathscr{M}$ is said to satisfy a PCTL state formula Φ, denoted $\mathscr{M} \models \Phi$, if $s_{\mathsf{init}} \models \Phi$. The semantics of the next- and until-operators is exactly as in CTL. If $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ is an infinite path in $\mathscr{M}$ then

$$\begin{array}{lll}
\pi \models \bigcirc\Phi & \text{iff} & s_1 \models \Phi \\
\pi \models \Phi_1 \mathsf{U} \Phi_2 & \text{iff} & \text{there exists } k \in \mathbb{N} \text{ with } s_k \models \Phi_2 \text{ and } s_i \models \Phi_1 \text{ for all } 0 \leq i < k.
\end{array}$$

The semantics of the cost-bounded until-operator is as for the standard until-operator, except that we require that the shortest prefix of π that ends in a Φ_2-state meets the cost-bound. Formally,

$$\begin{array}{lll}
\pi \models \Phi_1 \mathsf{U}^{\sim c} \Phi_2 & \text{iff} & \text{there exists } k \in \mathbb{N} \text{ such that} \\
 & & \text{(1)} \;\; s_k \models \Phi_2 \\
 & & \text{(2)} \;\; s_i \models \Phi_1 \wedge \neg\Phi_2 \text{ for all } 0 \leq i < k \\
 & & \text{(3)} \;\; \mathsf{cost}(\pi{\downarrow_k}) \sim c.
\end{array}$$

We now justify the above definitions. First, using [40, 105] we get that the set consisting of all paths where a PCTL path formula holds is indeed measurable. Second, we observe that

$$\begin{array}{lll}
s \models \mathbb{P}_{\leq p}(\varphi) & \text{iff} & \mathsf{Pr}_s^{\max}\{\pi \in \mathsf{Paths} \mid \pi \models \varphi\} \leq p \\
s \models \mathbb{P}_{< p}(\varphi) & \text{iff} & \mathsf{Pr}_s^{\max}\{\pi \in \mathsf{Paths} \mid \pi \models \varphi\} < p.
\end{array}$$

The first statement is obvious. The second statement follows from the fact that, for the events that can be specified by some PCTL path formula φ, there exists a scheduler that maximizes the probability for φ, and so the supremum defining $\mathsf{Pr}_s^{\max}$ can in fact be replaced with the maximum (see, e.g., [99]). For the next- and unbounded until-operators such a scheduler can in fact be assumed to be simple.

An analogous statement holds for strict or non-strict lower probability bounds and $\mathsf{Pr}_s^{\min}$ rather than $\mathsf{Pr}_s^{\max}$. Similarly, we have

$$s \models \mathbb{E}_{\leq c}(C) \quad \text{iff} \quad \mathsf{Ex}_s^{\max}(C) \leq c$$

$$s \models \mathbb{E}_{<c}(C) \quad \text{iff} \quad \mathsf{Ex}_s^{\max}(C) < c$$

and the analogous statement for lower cost bounds, where C stands for one of the three options $\Diamond\Phi$, $\leq k$, or $=k$. Here, again, minimal or maximal expected cost for the random variable associated with C can be achieved by some scheduler, and in the case of $\Diamond\Phi$ simple schedulers suffice.

Although the above semantics of the probabilistic and expectation operators relies on universal quantification over all schedulers, the existence of at least one scheduler satisfying a certain condition can be expressed using negation in front of the operator. For instance, $\neg\mathbb{P}_{\leq p}(\varphi)$ asserts the existence of a scheduler $\mathscr{U}$ where φ holds with probability $> p$.

Since probabilities are always values in the interval [0, 1], there are some trivial combinations of $\sim$ and p. For instance, $\mathbb{P}_{\geq 0}(\varphi)$ and $\mathbb{P}_{\leq 1}(\varphi)$ are tautologies, while $\mathbb{P}_{<0}(\varphi)$ and $\mathbb{P}_{>1}(\varphi)$ are not satisfiable. In what follows, we write $\mathbb{P}_{=1}(\varphi)$ for $\mathbb{P}_{\geq 1}(\varphi)$ and $\mathbb{P}_{=0}(\varphi)$ for $\mathbb{P}_{\leq 0}(\varphi)$. Similarly, as the cost function assigns non-negative cost to all transitions, the total cost can never be negative. Hence, formulas of the form $\mathbb{E}_{<0}(\cdot)$ are not satisfiable.

28.3.3 Derived Operators

Other Boolean operators can be derived from negation and conjunction as usual, e.g.,

$$\mathsf{ff} \overset{\text{def}}{=} \neg\mathsf{tt} \quad \text{and} \quad \Phi_1 \vee \Phi_2 \overset{\text{def}}{=} \neg(\neg\Phi_1 \wedge \neg\Phi_2).$$

The eventually operator $\Diamond$, a modality for path formulas, can be obtained as in CTL or LTL by

$$\Diamond\Phi \overset{\text{def}}{=} \mathsf{tt}\,\mathsf{U}\,\Phi,$$

and an analogous definition can be derived for the cost-bounded variant

$$\Diamond^{\sim c}\Phi \overset{\text{def}}{=} \mathsf{tt}\,\mathsf{U}^{\sim c}\Phi.$$

The always operator $\Box$ and its cost-bounded variant $\Box^{\sim c}$ can be derived using the duality of lower and upper probability bounds. For instance, $\mathbb{P}_{\leq p}(\Box\Phi)$ can be defined as $\mathbb{P}_{\geq 1-p}(\Diamond\neg\Phi)$, and $\mathbb{P}_{>p}(\Box^{\sim c}\Phi)$ as $\mathbb{P}_{<1-p}(\Diamond^{\sim c}\neg\Phi)$.

Example 4 (PCTL Formulas for the Running Example) First, we give examples of properties expressible in PCTL. The property "with probability at least 0.99, whenever we reach a bad state, we can recover with nonzero probability" from the beginning of this section can be stated as the formula $\mathbb{P}_{\geq 0.99}(\Box(\mathit{bad} \to \mathbb{P}_{>0}\Diamond\neg\mathit{bad}))$. Another property is "the expected energy consumption in the first 100 steps is at most 20 units", which is expressed by $\mathbb{E}_{\leq 20}(\leq 100)$, assuming that the relevant cost function quantifies the energy consumed at every step. Further, the formula $\mathbb{P}_{\leq 0.1}(\neg \mathit{initialised}\,\mathsf{U}\,\mathit{request})$ states that the probability of a request being made before the system initialisation phase completes is at most 0.1.

Now, let us return to the MDP from Example 1 to analyse some PCTL formulas more thoroughly. Consider the formula $\Phi \equiv \mathbb{P}_{\leq 0.6}(\neg \mathit{succ}\mathsf{U}^{\leq 5}\mathit{fail})$. First, observe that the formula $\neg\mathit{succ}$ holds in the states s_0, s_1 and s_3, whereas the formula fail holds only in the state s_3. Paths that satisfy $\neg\mathit{succ}\mathsf{U}^{\leq 5}\mathit{fail}$ are exactly the paths that reach s_3 and whose cost is at most 5. It is easy to see that the probability of these paths is maximal under any scheduler that always chooses α_{risk} deterministically, in which case these paths have probability 0.5. Thus, for any $\mathcal{U}$, we have $\mathsf{Pr}^{\mathcal{U}}_{s_0}(\neg\mathit{succ}\mathsf{U}^{\leq 5}\mathit{fail}) \leq 0.6$ and the formula Φ is satisfied.

On the other hand, the formula $\mathbb{E}_{\leq 5}(\leq 4)$ is not satisfied. Consider, for example, the scheduler that chooses α_{safe} in the path $s_0 \xrightarrow{\alpha_{go}} s_1$ and α_{risk} in the path $s_0 \xrightarrow{\alpha_{go}} s_1 \xrightarrow{\alpha_{safe}} s_0 \xrightarrow{\alpha_{go}} s_1$. Under this scheduler, the expected cost cumulated in 4 steps is 5.5, whereas the required upper bound is 5.

Remark 1 (Qualitative Properties) The conditions imposed by PCTL formulas of the form $\mathbb{P}_{>0}(\varphi)$ or $\mathbb{P}_{=1}(\varphi)$ are often called *qualitative properties*. Their meaning is quite close to CTL formulas $\exists\varphi$ and $\forall\varphi$ which are defined to be true if and only if for every scheduler $\mathcal{U}$ there is a $\mathcal{U}$-path satisfying φ (resp. all $\mathcal{U}$-paths satisfy φ in the case of $\forall\varphi$).

Indeed, if φ is a CTL path formula of the form $\bigcirc a$, $a\mathsf{U}b$ or $a\mathsf{U}^{\sim c}b$ where a, b are atomic propositions, then the PCTL formula $\mathbb{P}_{>0}(\varphi)$ is equivalent to the CTL formula $\exists\varphi$ (interpreted as described above). This is a consequence of the observation that the set of paths where φ holds can be written as a disjoint union of cylinder sets, and hence the requirement to have at least one path π with $\pi \models \varphi$ is equivalent to the requirement that the probability measure of the paths that satisfy φ is positive. Similarly, the PCTL formula $\mathbb{P}_{=1}(\Box a)$ and the CTL formula $\forall\Box a$ are equivalent: if there is a path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \dots$ where some s_i does not satisfy a, then no path starting with $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \dots s_i$ satisfies $\Box a$, and so the probability of paths satisfying $\Box a$ is strictly lower than 1. The same equality holds for $\mathbb{P}_{=1}(\bigcirc a)$ and $\forall \bigcirc a$.

However, there is a mild difference between the meaning of the PCTL formula $\mathbb{P}_{=1}(\Diamond a)$ and the CTL formula $\forall\Diamond a$, because the quantification over *"all paths"* is more restrictive than that over *"almost all paths"* in the case of reachability. Observe that state s satisfies the CTL formula $\forall\Diamond a$ if and only if all paths starting from s will eventually enter an a-state (i.e., a state s' with $s' \models a$). Satisfaction of the PCTL formula $\mathbb{P}_{=1}(\Diamond a)$ in state s means that almost all paths will eventually visit an a-state, in the sense that the probability measure of the paths π starting in s and satisfying φ equals 1; this includes paths that never enter an a-state, as long as their total probability measure is zero. □

28.4 Model-Checking Algorithms for MDPs and PCTL

We now present an algorithm that, given a PCTL state formula and a Markov decision process, decides whether the formula holds in the MDP or not. The algorithm, similarly to the algorithm for CTL model checking from Chap. 2, consists of separate subprocedures for each (temporal or Boolean) connective. Instead of computing the validity of a formula in the initial state directly, for each subformula we use the appropriate subprocedure and compute the set of all states in which the subformula holds. We start with the smallest subformulas and then proceed to the larger ones, using the sets of states already computed. Let us now describe the algorithm more formally, including the aforementioned subprocedures.

The main procedure to check whether a given PCTL state formula Φ_0 holds for an MDP relies on the same concepts as for CTL. An iterative approach is used to compute the satisfaction sets $\mathsf{Sat}(\Phi) = \{s \in S \mid s \models \Phi\}$ of all subformulas Φ of Φ_0. The treatment of the propositional logic fragment of PCTL follows directly from the definition of the semantics. We will concentrate here on explaining how to deal with probabilistic features. The algorithms we give run in polynomial time if the cost bounds and cost functions are given in unary. Hence, checking whether a given formula holds can be done in polynomial time under these assumptions.

In the sequel, let $\mathscr{M} = (S, \mathsf{Act}, \mathsf{P}, s_{\mathsf{init}}, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ be an MDP as in Sect. 28.2.2.

28.4.1 Probability Operator

Suppose that $\Phi = \mathbb{P}_{\sim p}(\varphi)$. We consider here the case of upper probability bounds, i.e., $\sim \in \{\leq, <\}$, so the task is to compute maximal probabilities of satisfying φ for every state. The set $\mathsf{Sat}(\Phi)$ can then be identified easily, as we have

$$\mathsf{Sat}(\Phi) = \bigl\{s \in S \bigm| \mathsf{Pr}_s^{\max}(\varphi) \sim p\bigr\}.$$

Lower probability bounds (i.e., the case when $\sim \in \{\geq, >\}$) can be treated similarly, but using minimum probability instead (see, e.g., [4, 99] for details). We distinguish three possible cases for the outermost operator of the path formula φ. For the proper state subformulas of φ, we can assume that the satisfaction sets $\mathsf{Sat}(\varphi)$ have already been computed. This allows us to treat them as atomic propositions.

First, we consider the **next-operator**. If $\varphi = \bigcirc\Psi$ then the maximal probabilities for φ are obtained by

$$\mathsf{Pr}_s^{\max}(\varphi) = \max_{\alpha \in \mathsf{Act}(s)} \mathsf{P}\bigl(s, \alpha, \mathsf{Sat}(\Psi)\bigr)$$

where $\mathsf{P}(s, \alpha, \mathsf{Sat}(\Psi)) = \sum_{t \in \mathsf{Sat}(\Psi)} \mathsf{P}(s, \alpha, t)$. An optimal simple scheduler simply assigns an action α to the state s that maximizes the value $\mathsf{P}(s, \alpha, \mathsf{Sat}(\Psi))$.

We now address the **until-operator** and suppose that $\varphi = \Phi_1 \mathsf{U} \Phi_2$. We first apply graph algorithms to compute the sets

$$S_0 = \{ s \in S \mid \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2) = 0 \}$$
$$S_1 = \{ s \in S \mid \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2) = 1 \}.$$

Note that S_0 is equal to the set $\{s \in S \mid \forall \pi \in \mathsf{Paths}(s) \mid \pi \not\models \Phi_1 \mathsf{U} \Phi_2\}$ which can be obtained using standard algorithms for non-probabilistic model checking (see Chap. 2). The set S_1 can be computed by iterating the following steps (1) and (2), where we start with the set of all states and keep pruning all actions and states that might lead to not satisfying the formula. Step (1) removes all states t from which no path satisfying $\Phi_1 \mathsf{U} \Phi_2$ starts. Step (2) considers all the remaining states s and removes all actions α from $\mathsf{Act}(s)$ such that $\mathsf{P}(s, \alpha, t) > 0$ for some state t that has been removed in step (1). The set of states that are not removed after repeating steps (1) and (2) constitutes the set S_1.

Let $S_? = S \setminus (S_0 \cup S_1)$ and $x_s = \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2)$ for $s \in S$. Clearly, $x_s = 0$ if $s \in S_0$, $x_s = 1$ if $s \in S_1$ and[2] $0 < x_s = \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} S_1) < 1$ if $s \in S_?$. The values x_s for $s \in S_?$ are obtained as the unique solution of the *linear program* [72] given by the inequalities

$$x_s \geq \sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot x_t + \mathsf{P}(s, \alpha, S_1) \quad \text{for all } \alpha \in \mathsf{Act}(s)$$

where $\sum_{s \in S_?} x_s$ is minimal and where $\mathsf{P}(s, \alpha, S_1) = \sum_{u \in S_1} \mathsf{P}(s, \alpha, u)$.

Intuitively, the inequalities of the above form capture the idea that the probability in state s must be at least the weighted sum of probabilities of the one-step successors, for any action α. Notice that every state is considered at most once in the sum, since $S_? \cap S_1 = \emptyset$.

A simple scheduler $\mathscr{U}$ with $\mathsf{Pr}_s^{\mathscr{U}}(\Phi_1 \mathsf{U} \Phi_2) = x_s = \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2)$ is obtained by carefully choosing, for each state $s \in S_1$, an action α with $\mathsf{P}(s, \alpha, S_1) = 1$ and, for each state $s \in S_?$, an action α that maximizes the value

$$\sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot x_t + \mathsf{P}(s, \alpha, S_1).^{3}$$

Some care is needed to ensure that the chosen action indeed makes some "progress" towards reaching a Φ_2-state. More formally, it is necessary to ensure that the actions taken will not avoid a Φ_2 state forever (the condition which captures this can be found in [4]). To illustrate the possible problem, consider the MDP from Fig. 3.

[2] The notation $\Phi_1 \mathsf{U} S_1$ is a shorthand for $\Phi_1 \mathsf{U} a$ where a is an atomic proposition satisfying $a \in L(s)$ if and only if $s \in S_1$.

[3] For the states $s \in S_0$ an arbitrary action can be chosen.

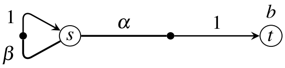

Fig. 3 An MDP showing that care needs to be taken when computing a scheduler $\mathcal{U}$ with $\mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} \Phi_2) = \mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2)$

Here, a simple scheduler that maximizes the probability for $\mathsf{tt}\,\mathsf{U}\,b$ must not take the action β for s, although $\mathsf{P}(s, \beta, S_1) = 1$ since $S_1 = \{s, t\}$.

Recall that all coefficients (transition probabilities in the MDP and the probability bound p) are rational, and hence the linear program above can be constructed in time polynomial in the size of $\mathcal{M}$. Because the linear program can be solved in polynomial time [72], the complexity of the problem to check whether an MDP satisfies a PCTL formula of the form $\mathbb{P}_{\leq p}(\Phi_1 \mathsf{U} \Phi_2)$ or $\mathbb{P}_{<p}(\Phi_1 \mathsf{U} \Phi_2)$ is also polynomial in the size of $\mathcal{M}$, assuming that the satisfaction sets for Φ_1 and Φ_2 are given.

Besides using well-known linear programming techniques to compute the vector $\vec{x} = (x_s)_{s \in S_?}$, one can use iterative approximation techniques. Most prominent are value and policy iteration, see, e.g., [99, 100].

In the *value iteration* approach, one starts with $x_s^{(0)} = 1$ for all $s \in S_1$ and $x_s^{(0)} = 0$ for all $s \in S_? \cup S_0$, and then successively computes

$$x_s^{(n+1)} \overset{\text{def}}{=} \max_{\alpha \in \mathsf{Act}(s)} \sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot x_t^{(n)} + \mathsf{P}(s, \alpha, S_1) \quad \text{for all } s \in S_?$$

until $\max_{s \in S_?} |x_s^{(n+1)} - x_s^{(n)}| < \varepsilon$ for some predefined tolerance $\varepsilon > 0$.

The idea of *policy iteration* is as follows. In each iteration, we select a simple scheduler $\mathcal{U}$ and compute the probabilities $\mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1)$ for $s \in S_?$ in the induced Markov chain (this can be done by solving a linear equation system). The method then "improves" the current simple scheduler $\mathcal{U}$ by searching for some state $s \in S_?$ such that

$$\mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1) < \max_{\alpha \in \mathsf{Act}(s)} \sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot \mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1) + \mathsf{P}(s, \alpha, S_1).$$

It then replaces $\mathcal{U}$ with $\mathcal{V}$ where $\mathcal{U}$ and $\mathcal{V}$ agree, except that $\mathcal{V}(s) = \alpha$ for some action $\alpha \in \mathsf{Act}(s)$ that maximizes $\sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot \mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1) + \mathsf{P}(s, \alpha, S_1)$. The next iteration is then performed with scheduler $\mathcal{V}$. If no improvement is possible, i.e., if

$$\mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1) = \max_{\alpha \in \mathsf{Act}(s)} \sum_{t \in S_?} \mathsf{P}(s, \alpha, t) \cdot \mathsf{Pr}_s^{\mathcal{U}}(\Phi_1 \mathsf{U} S_1) + \mathsf{P}(s, \alpha, S_1)$$

for all $s \in S_?$, then $\mathcal{U}$ maximizes the probability of $\Phi_1 \mathsf{U} \Phi_2$.

In practice, both value iteration and policy iteration outperform the linear-programming method, which does not scale to large models. The relative performance of value iteration and policy iteration varies by model, but the space and time efficiency of value iteration can be easily improved so that it outperforms policy iteration. Interested readers are referred to [52] for a brief comparison.

It remains to explain the treatment of the **cost-bounded until-operator**. We consider here just the case of non-strict upper cost bounds. The task is to compute $\mathsf{Pr}_s^{\max}(\varphi)$ for all states $s \in S$, where $\varphi = \Phi_1 \mathsf{U}^{\leq c} \Phi_2$ and $c \in \mathbb{N}$. For $s \in S$ and $d \in \mathbb{N}$ we define

$$x_s(d) \overset{\text{def}}{=} \mathsf{Pr}_s^{\max}\big(\Phi_1 \mathsf{U}^{\leq d} \Phi_2\big).$$

Then, we have $x_s(d) = 1$ for each state $s \in \mathsf{Sat}(\Phi_2)$ and each cost bound $d \in \mathbb{N}$. Similarly, $x_s(d) = 0$ for each $d \in \mathbb{N}$ and state s satisfying $\mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2) = 0$. Suppose now that $\mathsf{Pr}_s^{\max}(\Phi_1 \mathsf{U} \Phi_2) > 0$ and $s \not\models \Phi_2$. Thus, the recursive equations

$$x_s(d) = \max\left\{ \sum_{t \in S} \mathsf{P}(s, \alpha, t) \cdot x_t\big(d - \mathsf{C}(s, \alpha)\big) \;\Big|\; \alpha \in \mathsf{Act}(s), \mathsf{C}(s, \alpha) \leq d \right\}$$

hold true, where the maximum over the empty set is defined to be 0. That is, $x_s(d) = 0$ if $\mathsf{C}(s, \alpha) > d$ for all actions $\alpha \in \mathsf{Act}(s)$. Assuming that $\mathsf{C}(s, \alpha) > 0$ for all states s and enabled actions α, the above formulas for $x_s(d)$ can be computed by an iterative procedure, e.g., by employing a dynamic programming approach using the above equations. This yields the desired values $\mathsf{Pr}_s^{\max}(\varphi) = x_s(c)$. If $\mathsf{C}(s, \alpha) = 0$ for some states s and some actions $\alpha \in \mathsf{Act}(s)$ then the solution can be obtained as a solution to the linear program L_c which minimises $\sum_{s \in S} \sum_{0 \leq d \leq c} x_s(d)$, subject to

$$\begin{aligned}
&x_s(d) = 0 \quad \text{for } d < 0\\
&x_s(d) = 1 \quad \text{for } d \geq 0 \text{ and } s \in \mathsf{Sat}(\Phi_2)\\
&x_s(d) \geq \sum_{t \in S} P(s, \alpha, t) \cdot x_t\big(d - \mathsf{C}(s, \alpha)\big) \quad \text{for } d \geq 0,\, s \notin \mathsf{Sat}(\Phi_2) \text{ and } \alpha \in \mathsf{Act}(s)
\end{aligned}$$

where L_c contains variables $x_s(d)$ for $-M \leq d \leq c$ where M is the maximal number assigned by C. This approach can be optimised to consecutively solving $d + 1$ linear programs $L'_0, \ldots, L'_c$, where $L'_0 = L_0$ and for $1 \leq i \leq c$ the linear program L'_i is obtained from L_i by turning the variables $x_s(j)$ for $j < i$ into constants whose values were already computed earlier.

28.4.2 Expectation Operator

Suppose now that the task is to compute the satisfaction set $\mathsf{Sat}(\mathbb{E}_{\sim c}(C))$, where C is the random variable $\mathsf{cost}[\cdot]$ associated with the reachability condition $\Diamond \Psi$, the total cost within the first k steps (i.e., C is "$\leq k$"), or the instantaneous cost incurred by the k-th step (i.e., C is "$=k$"). Again, we just consider the case of maximal expected cost where the goal is to compute $\mathsf{Ex}_s^{\max}(C)$ for all states s. The set $\mathsf{Sat}(\mathbb{E}_{\sim c}(C))$ is then obtained by collecting all states s where $\mathsf{Ex}_s^{\max}(C) \sim c$.

Let us first address the case of **cumulated cost within** k **steps**. We can rely on the iterative computation scheme

$$\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[\le n]\big) = \max_{\alpha \in \mathsf{Act}(s)} \Big(\mathsf{C}(s,\alpha) + \sum_{t \in S} \mathsf{P}(s,\alpha,t) \cdot \mathsf{Ex}_t^{\max}\big(\mathsf{cost}[\le n-1]\big)\Big)$$

for $1 \le n \le k$ and $\mathsf{Ex}_s^{\max}(\mathsf{cost}[\le 0]) = 0$.

In the case of **instantaneous cost at time step** k, the equations have the form

$$\begin{aligned}\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[=1]\big) &= \max_{\alpha \in \mathsf{Act}(s)} \mathsf{C}(s,\alpha)\\ \mathsf{Ex}_s^{\max}\big(\mathsf{cost}[=n]\big) &= \max_{\alpha \in \mathsf{Act}(s)} \sum_{t \in S} \mathsf{P}(s,\alpha,t) \cdot \mathsf{Ex}_t^{\max}\big(\mathsf{cost}[=n-1]\big)\end{aligned}$$

for $1 < n \le k$.

We now sketch the main steps for the computation of the **maximal expected cost for the reachability objective** $\Diamond\Psi$. We first apply techniques for the standard until-operator (see Sect. 28.3) to compute $\mathsf{Pr}_s^{\min}(\Diamond\Psi)$ for all states s in $\mathscr{M}$.

If t is a state in $\mathscr{M}$ with $\mathsf{Pr}_t^{\min}(\Diamond\Psi) < 1$ then there exists a scheduler $\mathscr{U}$ such that $\mathsf{Pr}_t^{\mathscr{U}}(\Diamond\Psi) < 1$. But then $\mathsf{Ex}_t^{\mathscr{U}}(\mathsf{cost}[\Diamond\Psi])$ is infinite, and therefore

$$\mathsf{Ex}_t^{\max}\big(\mathsf{cost}[\Diamond\Psi]\big) = \infty.$$

The remaining task is to compute $\mathsf{Ex}_s^{\max}(\mathsf{cost}[\Diamond\Psi])$ for all states $s \in S'$ where

$$S' = \big\{ s \in S \,\big|\, \mathsf{Pr}_s^{\min}(\Diamond\Psi) = 1 \big\}.$$

Note that, if $s \in S' \setminus \mathsf{Sat}(\Psi)$, then for all actions $\alpha \in \mathsf{Act}(s)$ and all states u with $\mathsf{P}(s,\alpha,u) > 0$ we have $u \in S'$. The enabled actions of the states $s \in \mathsf{Sat}(\Psi)$ are irrelevant. We may suppose that for these s, $\mathsf{Act}(s)$ is a singleton set $\{\alpha\}$ with $\mathsf{P}(s,\alpha,s) = 1$. Clearly, for $s \in \mathsf{Sat}(\Psi)$ we have $\mathsf{Ex}_s^{\max}(\mathsf{cost}[\Diamond\Psi]) = 0$. For all other states $s \in S' \setminus \mathsf{Sat}(\Psi)$, we have

$$\mathsf{Ex}_s^{\max}\big(\mathsf{cost}[\Diamond\Psi]\big) = \max_{\alpha \in \mathsf{Act}(s)} \Big(\mathsf{C}(s,\alpha) + \sum_{u \in S'} \mathsf{P}(s,\alpha,u) \cdot \mathsf{Ex}_u^{\max}\big(\mathsf{cost}[\Diamond\Psi]\big)\Big).$$

These values can again be computed using linear programming techniques or the value or policy iteration schemes.

Example 5 Consider the MDP from Example 1 and the formula $\mathbb{E}_{\le 5}(\le 4)$. For all $0 \le i \le 4$, let x^i denote the tuple

$$\big(\mathsf{Ex}_{s_0}^{\max}\big(\mathsf{cost}[\le i]\big), \mathsf{Ex}_{s_1}^{\max}\big(\mathsf{cost}[\le i]\big), \mathsf{Ex}_{s_2}^{\max}\big(\mathsf{cost}[\le i]\big), \mathsf{Ex}_{s_3}^{\max}\big(\mathsf{cost}[\le i]\big)\big).$$

We iteratively compute the following tuples by applying value iteration

$$\begin{aligned}x^1 &= (\ \ 1,\ \ \ 4, 0, 0\ )\\ x^2 &= (\ \ 5,\ \ \ 4, 0, 0\ )\\ x^3 &= (\ \ 5, 4.5, 0, 0\ )\\ x^4 &= (\ 5.5, 4.5, 0, 0\ )\end{aligned}$$

and we conclude that the formula $\mathbb{E}_{\leq 5}(\leq 4)$ is not satisfied, because the maximal cumulated cost in s_0 is 5.5.

Next, consider again the same MDP, but this time together with the formula $\mathbb{P}_{\leq 0.5}(\neg \mathit{init} \,\mathsf{U}\, \mathit{succ})$, and suppose we want to know precisely the states in which the formula holds. We start by parsing the formula from the smallest subformulas. The subformula *init* is satisfied in s_0, and *succ* in s_2. Further, the subformula $\neg \mathit{init}$ is satisfied in the states s_1, s_2, and s_3. A more demanding task is to compute $\mathsf{Pr}_s^{\max}(\neg \mathit{init} \,\mathsf{U}\, \mathit{succ})$. We compute the sets S_0 and S_1, which are

$$S_0 = \{s_0, s_3\} \quad \text{and} \quad S_1 = \{s_2\}.$$

This leaves us with the set $S_? = \{s_1\}$. We construct the following simple linear program

$$\begin{aligned}&\text{minimize } x_{s_1} \text{ subject to}\\ &\quad x_{s_1} \geq x_{s_1}\\ &\quad x_{s_1} \geq 0.3.\end{aligned}$$

The solution to the above program is $x_{s_1} = 0.3$, and hence we can conclude that the formula $\mathbb{P}_{\leq 0.5}(\neg \mathit{init} \,\mathsf{U}\, \mathit{succ})$ holds in states s_0, s_1 and s_3.

28.5 Linear Temporal Logic

We continue this chapter with a brief overview of model checking Markov decision processes against properties expressed in linear temporal logic (LTL). In this section we define the logic and in the next section we show how the model-checking algorithm works. The logic LTL that we will use is standard, as defined in Chap. 2, except that we use only a subset of LTL which does not allow us to reason about the past, and whose predicates are actions of an MDP. Having predicates over actions and not over states is only a matter of convention; all the constructions and algorithms we present here can be easily modified to work with state predicates.

28.5.1 Syntax of LTL

For the purposes of this chapter, the syntax of LTL is as follows,

$$\varphi ::= \mathsf{tt} \mid \alpha \mid \varphi_1 \wedge \varphi_2 \mid \neg\varphi \mid \bigcirc \varphi \mid \varphi_1 \mathsf{U} \varphi_2$$

where tt stands for the constant truth value "true", and α is an action, i.e., an element of the set of actions Act. We write U to denote the *until*-operator, instead of $\mathcal{U}$ used in Chap. 2.

28.5.2 Semantics of LTL

The semantics of our logic LTL is defined on *traces* of paths of an MDP. A trace for an infinite path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \ldots$ is the infinite word $\mathsf{trace}(\pi) = \alpha_1\alpha_2\alpha_3\ldots$ of actions. Let $w = \alpha_0\alpha_1\ldots$ be an infinite word over the alphabet of actions Act, and let $w{\uparrow}^n$ denote the suffix of w starting with α_n. Then,

$$\begin{array}{lll}
w \models \mathsf{tt} & & \\
w \models \alpha & \text{iff} & \alpha = \alpha_0 \\
w \models \neg\phi & \text{iff} & w \not\models \phi \\
w \models \varphi_1 \wedge \varphi_2 & \text{iff} & w \models \varphi_1 \text{ and } w \models \varphi_2 \\
w \models \bigcirc\varphi & \text{iff} & w{\uparrow}^1 \models \varphi \\
w \models \varphi_1 \mathsf{U} \varphi_2 & \text{iff} & \text{there exists } k \in \mathbb{N} \text{ with } w{\uparrow}^k \models \varphi_2 \text{ and } w{\uparrow}^i \models \varphi_1 \text{ for } 0 \le i < k.
\end{array}$$

As in the case of PCTL, it can be shown that the set of all infinite paths that satisfy a given LTL formula is always measurable.

28.5.3 Derived Operators

Similarly to PCTL, we can define Boolean operators such as ff, $\vee$ and $\rightarrow$ from negation and conjunction, for example

$$\varphi_1 \vee \varphi_2 \overset{\text{def}}{=} \neg(\neg\varphi_1 \wedge \neg\varphi_2) \quad \text{and} \quad \varphi_1 \rightarrow \varphi_2 \overset{\text{def}}{=} (\neg\varphi_1) \vee \varphi_2.$$

The eventually-operator $\Diamond$ and the always-operator $\Box$ are obtained by

$$\Diamond\varphi \overset{\text{def}}{=} \mathsf{tt}\,\mathsf{U}\varphi \quad \text{and} \quad \Box\varphi \overset{\text{def}}{=} \neg\Diamond\neg\varphi.$$

For simplicity, we did not introduce a cost-bounded version of the until-operator $\mathsf{U}^{\sim c}$, but in principle there is nothing preventing us from doing so. We point out that the notation would become cumbersome; in particular, the definition of the Rabin automaton below would then need to take costs of state-action pairs into consideration.

28.5.4 LTL Model-Checking Problem

Let $\mathcal{M} = (S, \mathsf{Act}, \mathsf{P}, s_{\mathsf{init}}, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ be an MDP and $\mathbb{P}_{\sim p}(\varphi)$ an *LTL state property*, where $\sim$ is a comparison operator $\le$ or $<$, $p \in [0, 1] \cap \mathbb{Q}$ and φ is an LTL formula.

The *LTL model-checking problem* is to decide whether

$$\mathsf{Pr}^{\max}_{s_{\mathsf{init}}}\bigl\{\pi \in \mathsf{Paths} \bigm| \mathsf{trace}(\pi) \models \varphi\bigr\} \sim p.$$

We can define the model-checking problem similarly for the comparison operators $\geq$ or $>$; in that case we ask whether

$$\mathsf{Pr}^{\min}_{s_{\mathsf{init}}}\bigl\{\pi \in \mathsf{Paths} \bigm| \mathsf{trace}(\pi) \models \varphi\bigr\} \sim p.$$

Because the LTL formulas are closed under negation, we have

$$\begin{aligned}
&\mathsf{Pr}^{\min}_{s_{\mathsf{init}}}\bigl(\bigl\{\pi \in \mathsf{Paths} \bigm| \mathsf{trace}(\pi) \models \varphi\bigr\}\bigr)\\
&\quad = 1 - \mathsf{Pr}^{\max}_{s_{\mathsf{init}}}\bigl(\bigl\{\pi \in \mathsf{Paths} \bigm| \mathsf{trace}(\pi) \not\models \varphi\bigr\}\bigr)\\
&\quad = 1 - \mathsf{Pr}^{\max}_{s_{\mathsf{init}}}\bigl(\bigl\{\pi \in \mathsf{Paths} \bigm| \mathsf{trace}(\pi) \models \neg\varphi\bigr\}\bigr)
\end{aligned}$$

and so without loss of generality we can restrict our interest to the case of computing maximal probabilities.

28.6 Model-Checking Algorithms for MDPs and LTL

In this section we describe a model-checking algorithm for MDPs and LTL. Before going into formal definitions, let us describe it informally. We solve the LTL model-checking problem using ω-regular automata. Every LTL formula can be transformed into an automaton which accepts exactly the words on which the formula holds. We then build the *product* of the MDP and the automaton, and show that the problem of computing the optimal probability with which the automaton accepts traces of the MDP is equal to the problem of computing the optimal probability of reaching certain states in the product. The latter can be computed using algorithms from previous sections. The reader may observe that the outline of the algorithm is similar to the (non-probabilistic) LTL model-checking algorithm from Chap. 2. The major difference is that, instead of looking for one path in the product (called synchronous composition in Chap. 2), we need to determine the probability of certain paths. It turns out that, for this purpose, the definition of a just discrete system is not sufficient. The solution we present here uses Rabin automata, whose crucial property is that it has no nondeterminism.

The algorithm runs in time polynomial in the size of the MDP and doubly-exponential in the size of the LTL formula. From the complexity-theoretic point of view, the complexity bound is optimal since the model-checking problem for Markov decision processes and LTL state properties is known to be complete for the complexity class 2EXPTIME, even for qualitative LTL state properties [40].

Let us now describe the algorithm formally. We begin by introducing the notion of *deterministic Rabin automata* and stating that, for every LTL formula φ, there is a deterministic Rabin automaton that accepts exactly the set of words satisfying φ.

Definition 1 (Deterministic Rabin Automaton (DRA)) A *deterministic Rabin automaton* is a tuple $\mathscr{A} = (Q, \mathsf{Act}, \delta, q_{init}, Acc)$, where Q is a finite set of states, $q_{init} \in Q$ is an initial state, Act is a finite input alphabet, $\delta : Q \times \mathsf{Act} \to Q$ is a transition function, and $Acc = \{(L_1, K_1), (L_2, K_2), \dots, (L_k, K_k)\}$, for $k \in \mathbb{N}$ and $L_i, K_i \subseteq Q$, $1 \le i \le k$, is a set of accepting tuples of states.

We do not study Rabin automata in detail here and only mention their properties directly relevant to LTL model checking. We refer the reader to Chap. 4 or to [56] for additional details.

Let $\mathscr{A} = (Q, \mathsf{Act}, \delta, q_{init}, Acc)$ be a DRA. For every infinite word $w = \alpha_0\alpha_1\alpha_2\dots$ over the input alphabet Act there is a unique sequence $q_0\alpha_0q_1\alpha_1q_2\alpha_2\cdots$ where $q_0 = q_{init}$, and $\delta(q_i, \alpha_i) = q_{i+1}$ for all $i \ge 0$. The word w is *accepted* by $\mathscr{A}$ if there is $(L, K) \in Acc$ such that $q_i \in L$ for only finitely many i, and $q_j \in K$ for infinitely many j. The set of all infinite words over Act that $\mathscr{A}$ accepts is called the *language* of $\mathscr{A}$ and is denoted $\mathscr{L}(A)$.

As mentioned above, for every LTL formula φ we can construct a DRA $\mathscr{A}_\varphi$ with the input alphabet Act such that for all $w = \alpha_1\alpha_2\dots$ we have

$$w \models \varphi \iff w \in \mathscr{L}(\mathscr{A}_\varphi).$$

The construction of $\mathscr{A}_\varphi$ is non-trivial and we do not present it in this chapter, referring the reader to [14, 41, 107]. Note that, in general, the size of $\mathscr{A}_\varphi$ can be up to doubly exponential in the size of φ. In practice, however, this is often not a serious problem since LTL formulas expressing useful properties tend to be small compared to the size of the MDP.

Having defined the DRA $\mathscr{A}_\varphi$, we reduce the problem of computing the maximal probability of paths satisfying φ in $\mathscr{M}$ to the problem of reaching a certain set of states in a *product* MDP. The product MDP is defined so that its behaviour mimics that of the original MDP, but in addition it remembers the state of the automaton in which it ends after reading the sequence of actions performed so far.

Definition 2 (Product of an MDP and a DRA) Let $\mathscr{M} = (S, \mathsf{Act}, \mathsf{P}, s_{\mathsf{init}}, \mathsf{AP}, \mathsf{L}, \mathsf{C})$ be an MDP and $\mathscr{A} = (Q, \mathsf{Act}, \delta, q_{init}, Acc)$ a DRA. Their *product* $\mathscr{M} \otimes \mathscr{A}$ is the MDP $(S \times Q, \mathsf{Act}, \mathsf{P}', (s_{\mathsf{init}}, q_{init}), \mathsf{AP}, \mathsf{L}', \mathsf{C}')$ where for any $(s, q) \in S \times Q$ and $\alpha \in \mathsf{Act}$ we define

$$\mathsf{P}'\big((s, q), \alpha, (s', q')\big) = \begin{cases} \mathsf{P}(s, \alpha, s') & \text{if } \delta(q, \alpha) = q' \\ 0 & \text{otherwise.} \end{cases}$$

The elements L' and C' are defined arbitrarily.

A path $(s_0, q_0) \xrightarrow{\alpha_1} (s_1, q_1) \xrightarrow{\alpha_2} (s_2, q_2) \xrightarrow{\alpha_3} \dots$ in a product MDP is accepting if there is $(L, K) \in Acc$ such that $q_i \in L$ for only finitely many i and $q_j \in K$ for infinitely many j.

It can be proved that, for every state s and scheduler $\mathscr{U}$ in $\mathscr{M}$, there is a scheduler $\mathscr{V}$ in $\mathscr{M}\otimes\mathscr{A}$ such that

$$\begin{aligned}&\mathsf{Pr}^{\mathscr{M},\mathscr{U}}\big(\big\{\pi \in \mathsf{Paths}^{\mathscr{M}}(s) \mid \mathsf{trace}(\pi) \in \mathscr{L}(\mathscr{A})\big\}\big)\\ &\quad = \mathsf{Pr}^{\mathscr{M}\otimes\mathscr{A},\mathscr{V}}\big(\big\{\pi \in \mathsf{Paths}^{M\otimes\mathscr{A}}\big((s, q_{init})\big) \mid \pi \text{ is an accepting path}\big\}\big).\end{aligned}$$

This is essentially because the product only extends the original by keeping track of a computation of a DRA, and does not alter the power of schedulers.

So far, we have reduced the problem of LTL model checking to the problem of determining the maximal probability of accepting paths in a product MDP. To determine this probability, we introduce the notion of accepting end components, which identify the states for which there is a scheduler ensuring that almost all paths starting in these states are accepting. An *accepting end component* (EC) of $\mathscr{M}\otimes\mathscr{A}$ is a pair $(\bar{S}, \bar{\mathsf{P}})$ comprising a subset $\bar{S} \subseteq S\times Q$ of states and partial transition probability function $\bar{\mathsf{P}} : \bar{S}\times\mathsf{Act}\times\bar{S} \to [0, 1]\cap\mathbb{Q}$ satisfying the following conditions:

1. $(\bar{S}, \bar{\mathsf{P}})$ determines a sub-MDP of $\mathscr{M}\otimes\mathscr{A}$, i.e., for all $s' \in \bar{S}$ and $\alpha \in \mathsf{Act}$ we have $\sum_{s''\in\bar{S}} \bar{\mathsf{P}}(s', \alpha, s'') = 1$, and, if $\bar{\mathsf{P}}(s', \alpha, s'')$ is defined, then $\bar{\mathsf{P}}(s', \alpha, s'') = \mathsf{P}'(s', \alpha, s'')$;
2. the underlying graph of $(\bar{S}, \bar{\mathsf{P}})$ is strongly connected;
3. there is $(L, K) \in Acc$ such that:
   a. all $(s, q) \in \bar{S}$ satisfy $q \notin L$;
   b. there is $(s, q) \in \bar{S}$ satisfying $q \in K$.

Using the above condition for an accepting path, together with the property that, once an end component is entered, all its states can be visited infinitely often almost surely [4], we can further show the following. Let $T \subseteq S \times Q$ such that $(s', q') \in T$ if and only if (s', q') appears in some accepting end component of $\mathscr{M}\otimes\mathscr{A}$, then we have

$$\begin{aligned}&\mathsf{Pr}_s^{\max}\big\{\pi \in \mathsf{Paths}^{\mathscr{M}}(s) \mid \mathsf{trace}(\pi) \in \mathscr{L}(\mathscr{A})\big\}\\ &\quad = \mathsf{Pr}_{(s,q_{init})}^{\max}\big(\big\{\pi \in \mathsf{Paths}^{\mathscr{M}\otimes\mathscr{A}}\big((s, q_{init})\big) \mid \pi \text{ contains a state from } T\big\}\big).\end{aligned}$$

Thus, we have reduced model checking of LTL properties to (i) the computation of accepting end components in $\mathscr{M}\otimes\mathscr{A}_\varphi$, and (ii) the computation of maximum probabilities of reaching these end components. The second step is a special case of the problems studied in Sect. 28.4. The first step can be done efficiently using the results of [4, 14]; an approach which is simpler to comprehend, but less efficient, is to use PCTL model checking to identify all the states that lie in an accepting component and satisfy the condition 3b. above. These are exactly the states (s, q) for which there is $(L, K) \in Acc$ such that $q \in K$ and it is possible to return to (s, q) with probability 1, passing only through states (s', q') with $q' \notin L$. A state (s, q) satisfies this condition if and only if it satisfies a formula $\neg\mathbb{P}_{<1}(\bigcirc\neg\mathbb{P}_{<1}\, p_{\neg L} \,\mathsf{U}\, p_{(s,q)})$ for some $(L, K) \in Acc$ with $q \in K$, where $p_{(s,q)}$ holds only in (s, q) and $p_{\neg L}$ holds in all states (s', q') with $q' \notin L$. In step (ii) it is then sufficient to maximise the probability of reaching such states.

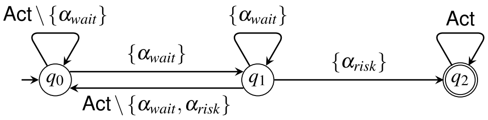
$\mathsf{Act} \setminus \{\alpha_{wait}\}$
$\{\alpha_{wait}\}$
$\{\alpha_{wait}\}$
Act
q_0
q_1
q_2
$\{\alpha_{risk}\}$
$\mathsf{Act} \setminus \{\alpha_{wait}, \alpha_{risk}\}$

Fig. 4 A DRA for the formula $\Diamond(\alpha_{wait} \wedge \bigcirc\alpha_{risk})$

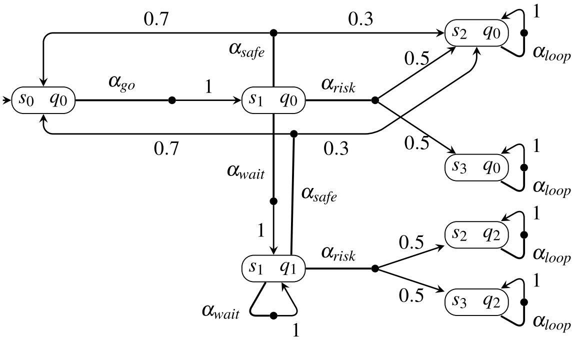

Fig. 5 The product MDP $\mathcal{M}\otimes\mathcal{A}$ for Example 6

Example 6 Consider the MDP from Example 1 together with the formula $\Phi = \Diamond(\alpha_{wait} \wedge \bigcirc\alpha_{risk})$, and assume we want to compute the maximal probability of satisfying this formula. We follow the procedure described above and first convert Φ to an equivalent DRA $\mathcal{A} = (Q, \mathsf{Act}, \delta, q_{init}, Acc)$. Using one of the cited methods, we might, for example, obtain the automaton shown in Fig. 4. Here, $Q = \{q_0, q_1, q_2\}$, $q_{init} = q_0$, $\delta(q, \alpha) = q'$ whenever there is an arrow from q to q' labelled with a set containing α, and $Acc = \{(\emptyset, \{q_2\})\}$.

Next, we construct the product of $\mathcal{M}$ and $\mathcal{A}$, yielding the MDP $\mathcal{M}\otimes\mathcal{A}$ from Fig. 5 (note that only the states reachable from the initial state (s_0, q_0) are drawn). The MDP $\mathcal{M}\otimes\mathcal{A}$ contains two accepting end components, one containing the state (s_2, q_2) and a self-loop, and the other containing the state (s_3, q_2) and a self-loop.

It is now easy to apply the algorithms from Sect. 28.4 and calculate that the maximal probability of reaching one of these end components from the initial state is equal to 1.

28.7 Tools, Applications and Model Construction

28.7.1 Tool Support

There are several software tools which implement probabilistic model checking for Markov decision processes. One of the most widely used is PRISM [81], an open-source tool available from [98] which supports both PCTL and LTL model checking as described here, including the probabilistic and expectation operators. PRISM uses a probabilistic variant of reactive modules as a modelling notation, and additionally supports model checking for discrete- and continuous-time Markov chains and probabilistic timed automata. The tools LIQUOR [38] and ProbDiVinE [16] implement

LTL model checking for MDPs: LIQUOR uses Probmela, which is a variant of the SPIN Promela modelling language, whereas ProbDiVinE provides a parallel implementation. RAPTURE [70] and PASS [58] apply abstraction-refinement techniques.

A key challenge when implementing the algorithms is the state-explosion problem, well known from other fields of model checking, and also discussed in Chap. 8 of this book. Different tools take a different approach to overcome this problem. The tool PRISM, for example, mainly uses a symbolic approach (see [6, 9] or Chap. 31) and instead of storing the state space explicitly it stores it using a variant of *binary decision diagrams* [54]. ProbDiVinE makes use of *distributed model checking*, while LIQUOR applies *partial-order reduction techniques* (see Chap. 6) to reduce the state space. Several other methods to tackle the state-explosion problem have been proposed for probabilistic model checking, including *symmetry reduction* [44, 78], game-based *quantitative abstraction refinement* [74, 80], compositional probabilistic verification [42, 51, 82, 83], or algorithms for *simulation* and *bisimulation* relations [31, 110]. Techniques to improve efficiency of probabilistic model checking include *approximate probabilistic model-checking* [88], *statistical model checking* [19, 25, 89, 108, 109] and *incremental verification* [86].

28.7.2 Applications

Probability is pervasive, and Markov decision processes underpin modelling and analysis of a wide range of applications [99]. Probabilistic model checking, and PRISM in particular, has been successfully applied to analyse and in some cases detect flaws in a variety of application domains, including analysis of communication, security, privacy and anonymity protocols, efficiency of power management protocols, correctness of randomised coordination algorithms, performance of computer systems and nanotechnology designs, *in silico* exploration of biological signalling, detecting design flaws in DNA circuits, analysis of spread of diseases, scheduling, planning, and controller synthesis (see, e.g., [45, 62, 79, 94]). More case studies are available at the PRISM tool website [98].

28.7.3 Construction of Probabilistic Models

The usefulness and precision of the results obtained by the probabilistic model-checking techniques presented here crucially depend on the adequacy of the model, and in particular on the probability values. Several methods have been proposed in the literature that support the stepwise and compositional design of probabilistic models for systems with many components, ranging from approaches that use stochastic process algebras (see, e.g., [3, 71]), probabilistic variants of Petri nets (see, e.g., [2]), or bespoke models (see, e.g., [5]) to high-level modelling languages with guarded commands, probabilistic choice, and imperative programming language concepts [8, 20, 60, 66, 73]. Such approaches can indeed be very helpful when constructing reasonable models that reflect the architectural structure of the system to be analysed, the control flow of its components, the interaction mechanism, and dependencies between components where the probabilities are known or given, as is the case in randomised protocols. However, such formal modelling approaches do not support the choice of the probability values. Estimating probability distributions is one of the core problems studied in statistics. Indeed, for many application areas, well-engineered statistical methods are available to derive good estimates for the probability values in the models used for the quantitative analysis. But even without the application of advanced statistical methods, probabilistic model-checking techniques can yield useful information on the quantitative system behavior. Repeated application of probabilistic model-checking techniques on models that only differ in the probability values might give insights into the significance or irrelevance of certain probabilistic parameters. The model of Markov decision processes also permits the representation of incomplete information on the probability values by nondeterministic choices between several probabilistic distributions. The results obtained by probabilistic model checking are lower or upper bounds for all models that result by resolving the nondeterministic choices using any convex combination of the chosen distributions. Alternatively, there are also methods that deal with probability intervals rather than specific probability values, and methods that operate with parametrized probabilistic models, see, e.g., [37, 43, 57, 103].

28.8 Extensions of the Model and Specification Notations

There are various models that extend Markov decision processes, such as *stochastic games* [33, 34, 36], in which there are two kinds of nondeterminism (sometimes called "angelic" and "demonic" nondeterminism), or *probabilistic timed automata* [84, 95], which extend timed automata as defined in Chap. 29 and allow for reasoning about time by adding real-time constraints on actions. Another class of related models are *continuous-time Markov Chains* and *continuous-time Markov decision processes* [99] in which we add a notion of time into the system and assume that the steps from one state to another are taken with delays governed by an exponential probability distribution. Continuous-time Markov Chains find applications, for example, in biochemistry (see, e.g., [29, 30, 39, 63, 92]). Note that MDPs as defined in this chapter are sometimes called *discrete-time* MDPs to reflect the intuition that each of their steps takes exactly 1 time unit. Also note that adding an exponential distribution on time makes it more difficult to define parallel composition of two systems, leading to an alternative model of *interactive Markov chains* (see, e.g., [28, 64]).

Probabilistic models with infinite state space have also been studied, where examples include models generated by pushdown systems (see, e.g., [22, 26, 49]) or lossy channel systems [1, 10, 69].

Recently [32], alternatives to deterministic Rabin automata, such as generalized Rabin automata [47, 76], have been shown suitable for probabilistic model checking. These automata can be smaller by orders of magnitude and thus induce a smaller product to be analyzed. See [18] for an overview of available tools for conversion of LTL to different types of Rabin automata and their performance.

The logics LTL and PCTL can be naturally combined into the logic PCTL* [17], which is itself a probabilistic variant of the logic CTL* [46]. There are also numerous reward-based properties not included in our definition of PCTL, for example a discounted reward or long-run average reward [4, 99]. There also exist different logics that allow us to reason about probabilities, one example being the works [67, 91, 93] which study a probabilistic variant of μ-calculus (see Chap. 26). A new direction started recently concerns studying multi-objective model-checking problems for Markov decision processes [23, 35, 48, 53].

A related problem is that of *controller synthesis*, where the question is whether there *exists* a satisfying scheduler (as opposed to the model-checking problem, where we ask whether *all* schedulers satisfy the formula). For the unrestricted controller-synthesis problem, an alternative semantics of PCTL has been studied [12, 24, 27], yielding undecidability results.

28.9 Conclusion

In this chapter, we have given an overview of probabilistic model checking, focusing on Markov decision processes as an operational model for nondeterministic-probabilistic systems against specifications given in temporal logics PCTL and LTL. The PCTL model-checking algorithm is similar to that for the logic CTL, where the parse tree of the formula is traversed bottom up and each subformula is treated separately. Model checking for the probabilistic and expectation operator reduces to a linear programming problem, which can be solved using a variety of methods.

In the case of LTL, we first translate the LTL formula into an equivalent deterministic Rabin automaton, and then reduce the model-checking problem to the problem of calculating the probability of reaching accepting end components in a product of the MDP and the automaton. The construction of a deterministic Rabin automaton for a given LTL formula can cause a doubly exponential blowup.

We have also presented a brief summary of tools that implement and extend the algorithms presented in this chapter, and listed various related formalisms that exist in the area of probabilistic model checking.

References

1. Abdulla, P., Baier, C., Iyer, P., Jonsson, B.: Reasoning about probabilistic lossy channel systems. In: Palamidessi, C. (ed.) Proc. CONCUR'00. LNCS, vol. 1877, pp. 320–330. Springer, Heidelberg (2000)

2. Ajmone-Marsan, M., Balbo, G., Conte, G., Donatelli, S., Franceschinis, G.: Modelling with Generalized Stochastic Petri Nets. Wiley Series in Parallel Computing. Wiley, New York (1995)
3. Aldini, A., Bernardo, M., Corradini, F.: A Process Algebraic Approach to Software Architecture Design. Springer, Heidelberg (2010)
4. de Alfaro, L.: Formal verification of probabilistic systems. Ph.D. thesis, Stanford University, Department of Computer Science (1997)
5. de Alfaro, L., Henzinger, T.A., Jhala, R.: Compositional methods for probabilistic systems. In: Larsen, K.G., Nielsen, M. (eds.) CONCUR. LNCS, vol. 2154, pp. 351–365. Springer, Heidelberg (2001)
6. de Alfaro, L., Kwiatkowska, M., Norman, G., Parker, D., Segala, R.: Symbolic model checking of probabilistic processes using MTBDDs and the Kronecker representation. In: Graf, S., Schwartzbach, M.I. (eds.) Proc. Tools and Algorithms for Construction and Analysis of Systems (TACAS). LNCS, vol. 1785, pp. 395–410. Springer, Heidelberg (2000)
7. Ash, R., Doléans-Dade, C.: Probability and Measure Theory. Harcourt/Academic Press, San Diego (2000)
8. Baier, C., Ciesinski, F., Größer, M.: ProbMeLa: a modeling language for communicating probabilistic systems. In: Proc. of the 2nd ACM-IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE), pp. 57–66. IEEE, Piscataway (2004)
9. Baier, C., Clarke, E., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds.) Proc. International Colloqium on Automata, Languages and Programming (ICALP). LNCS, vol. 1256, pp. 430–440. Springer, Heidelberg (1997)
10. Baier, C., Engelen, B.: Establishing qualitative properties for probabilistic lossy channel systems: an algorithmic approach. In: Katoen, J.-P. (ed.) Intl. AMAST Workshop, ARTS. LNCS, vol. 1601, pp. 34–52. Springer, Heidelberg (1999)
11. Baier, C., Größer, M., Ciesinski, F.: Model checking linear time properties of probabilistic systems. In: Droste, M., Kuich, W., Vogler, H. (eds.) Handbook of Weighted Automata, Monographs in Theoretical Computer Science. An EATCS Series, pp. 519–570. Springer, Heidelberg (2009)
12. 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 Academic, Dordrecht (2004)
13. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Commun. ACM **53**(9), 76–85 (2010)
14. Baier, C., Katoen, J.P.: Principles of Model Checking. MIT Press, Cambridge (2008)
15. Baier, C., Kwiatkowska, M.: Model checking for a probabilistic branching time logic with fairness. Distrib. Comput. **11**, 125–155 (1998)
16. Barnat, J., Brim, L., Černá, I., Češka, M., Tůmová, J.: ProbDiVinE-MC: multi-core LTL model checker for probabilistic systems. In: Proceedings of the 2008 Fifth International Conference on Quantitative Evaluation of Systems, pp. 77–78. IEEE, Washington (2008)
17. Bianco, A., De Alfaro, L.: Model checking of probabilistic and non-deterministic systems. In: Thiagarajan, P.S. (ed.) Proceedings of Foundations of Software Technology and Theoretical Computer Science. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995)
18. Blahoudek, F., Kretínský, M., Strejcek, J.: Comparison of LTL to deterministic Rabin automata translators. In: McMillan, K.L., Middeldorp, A., Voronkov, A. (eds.) Logic for Programming, Artificial Intelligence, and Reasoning—Proceedings of the 19th International Conference, LPAR-19, Stellenbosch, South Africa, December 14–19, 2013. LNCS, vol. 8312, pp. 164–172. Springer, Heidelberg (2013)
19. Bogdoll, J., Fioriti, L.M.F., Hartmanns, A., Hermanns, H.: Partial order methods for statistical model checking and simulation. In: Bruni, R., Dingel, J. (eds.) FMOODS/FORTE. LNCS, vol. 6722, pp. 59–74. Springer, Heidelberg (2011)

20. Bohnenkamp, H., D'Argenio, P., Hermanns, H., Katoen, J.P.: MODEST: a compositional modeling formalism for hard and softly timed systems. IEEE Trans. Softw. Eng. **32**(10), 812–830 (2006)
21. Bolch, G., Greiner, S., de Meer, H., Trivedi, K.: Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications. Wiley-Interscience, New York (1998)
22. Brázdil, T.: Verification of probabilistic recursive sequential programs. Ph.D. thesis, Masaryk University (2007)
23. Brázdil, T., Brožek, V., Chatterjee, K., Forejt, V., Kučera, A.: Two views on multiple mean-payoff objectives in Markov decision processes. In: Proceedings of LICS'11, pp. 33–42. IEEE, Piscataway (2011)
24. 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, Piscataway (2006)
25. Brázdil, T., Chatterjee, K., Chmelík, M., Forejt, V., Křetínský, J., Kwiatkowska, M., Parker, D., Ujma, M.: Verification of Markov decision processes using learning algorithms. In: Cassez, F., Raskin, J. (eds.) Proc. 12th International Symposium on Automated Technology for Verification and Analysis (ATVA'14). LNCS, vol. 8837, pp. 98–114. Springer, Heidelberg (2014)
26. Brázdil, T., Esparza, J., Kiefer, S., Kučera, A.: Analyzing probabilistic pushdown automata. Form. Methods Syst. Des. **43**(2), 124–163 (2013)
27. 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, Heidelberg (2008)
28. Brázdil, T., Hermanns, H., Krčál, J., Křetínský, J., Řehák, V.: Verification of open interactive Markov chains. In: D'Souza, D., Kavitha, T., Radhakrishnan, J. (eds.) FSTTCS. LIPIcs, vol. 18, pp. 474–485. Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, Dagstuhl (2012)
29. Brim, L., Češka, M., Dražan, S., Šafránek, D.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: Sharygina and Veith [104], pp. 107–123
30. Cardelli, L.: Artificial biochemistry. In: Condon, A., Harel, D., Kok, J.N., Salomaa, A., Winfree, E. (eds.) Algorithmic Bioprocesses. Natural Computing Series, pp. 429–462. Springer, Heidelberg (2009)
31. Cattani, S., Segala, R.: Decision algorithms for probabilistic bisimulation. In: Brim, L., Jančar, P., Křetínský, M., Kučera, A. (eds.) Proc. 14th Int. Conf. Concurrency Theory (CONCUR'02). LNCS, vol. 2421, pp. 371–385. Springer, Heidelberg (2002)
32. Chatterjee, K., Gaiser, A., Křetínský, J.: Automata with generalized Rabin pairs for probabilistic model checking and LTL synthesis. In: Sharygina and Veith [104], pp. 559–575
33. Chatterjee, K., Jurdzinski, M., Henzinger, T.: Simple stochastic parity games. In: Baaz, M., Makowsky, J.A. (eds.) Proceedings of the International Conference for Computer Science Logic (CSL). LNCS, vol. 2803, pp. 100–113. Springer, Heidelberg (2003)
34. Chatterjee, K., Jurdzinski, M., Henzinger, T.: Quantitative stochastic parity games. In: Munro, J.I. (ed.) Proceedings of the Annual Symposium on Discrete Algorithms (SODA), pp. 121–130. SIAM, Philadelphia (2004)
35. Chatterjee, K., Majumdar, R., Henzinger, T.A.: Markov decision processes with multiple objectives. In: Durand, B., Thomas, W. (eds.) STACS. LNCS, vol. 3884, pp. 325–336. Springer, Heidelberg (2006)
36. Chen, T., Forejt, V., Kwiatkowska, M., Parker, D., Simaitis, A.: Automatic verification of competitive stochastic systems. Form. Methods Syst. Des. **43**(1), 61–92 (2013)
37. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: Proc. 7th International Symposium on Theoretical Aspects of Software Engineering (TASE'13), pp. 85–92. IEEE, Piscataway (2013)

38. Ciesinski, F., Baier, C.: LiQuor: a tool for qualitative and quantitative linear time analysis of reactive systems. In: Proc. QEST 2007, pp. 131–132. IEEE, Piscataway (2007)
39. Ciocchetta, F., Hillston, J.: Bio-PEPA: a framework for the modelling and analysis of biological systems. Theor. Comput. Sci. **410**(33–34), 3065–3084 (2009)
40. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
41. Daniele, M., Giunchiglia, F., Vardi, M.: Improved automata generation for linear temporal logic. In: Halbwachs, N., Peled, D. (eds.) Proc. International Conference on Computer Aided Verification (CAV). LNCS, vol. 1633, pp. 249–260. Springer, Heidelberg (1999)
42. 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, Piscataway (2010)
43. Delahaye, B., Katoen, J.P., Larsen, K., Legay, A., Pedersen, M., Sher, F., Wasowski, A.: Abstract probabilistic automata. In: Jhala, R., Schmidt, D.A. (eds.) 12th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI). LNCS, vol. 6538, pp. 324–339. Springer, Heidelberg (2011)
44. 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, Heidelberg (2006)
45. Duflot, M., Kwiatkowska, M., Norman, G., Parker, D.: A formal analysis of Bluetooth device discovery. Int. J. Softw. Tools Technol. Transf. **8**(6), 621–632 (2006)
46. Emerson, E.A.: Temporal and modal logic. In: van Leeuwen, J. (ed.) Handbook of Theoretical Computer Science, vol. B: Formal Models and Semantics, pp. 996–1072. Elsevier, Amsterdam (1990). Chap. 14
47. Esparza, J., Křetínský, J.: From LTL to deterministic automata: a Safraless compositional approach. In: Biere, A., Bloem, R. (eds.) Computer Aided Verification—Proceedings of the 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18–22, 2014. LNCS, vol. 8559, pp. 192–208. Springer, Heidelberg (2014)
48. Etessami, K., Kwiatkowska, M.Z., Vardi, M.Y., Yannakakis, M.: Multi-objective model checking of Markov decision processes. Log. Methods Comput. Sci. **4**(4) (2008)
49. Etessami, K., Yannakakis, M.: Model checking of recursive probabilistic systems. ACM Trans. Comput. Log. **13**(2), 1–40 (2012)
50. Feller, W.: An Introduction to Probability Theory and Its Applications. Wiley, New York (1950)
51. 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, Piscataway (2010)
52. Forejt, V., Kwiatkowska, M., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) Formal Methods for Eternal Networked Software Systems (SFM'11). LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011)
53. 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 International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'11). LNCS, vol. 6605, pp. 112–127. Springer, Heidelberg (2011)
54. Fujita, M., McGeer, P.C., Yang, J.C.Y.: Multi-terminal binary decision diagrams: an efficient data structure for matrix representation. Form. Methods Syst. Des. **10**(2/3), 149–169 (1997)
55. van Glabbeek, R., Smolka, S.A., Steffen, B., Tofts, C.M.N.: Reactive, generative, and stratified models of probabilistic processes. In: Proc. 5th Annual Symposium on Logic in Computer Science (LICS), pp. 130–141. IEEE, Piscataway (1990)

56. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata, Logics, and Infinite Games: A Guide to Current Research [outcome of a Dagstuhl seminar, February 2001]. LNCS, vol. 2500. Springer, Heidelberg (2002)
57. Hahn, E., Hermanns, H., Wachter, B., Zhang, L.: PARAM: a model checker for parametric Markov models. In: Touili, T., Cook, B., Jackson, P. (eds.) 22nd International Conference on Computer Aided Verification (CAV). LNCS, vol. 6174, pp. 660–664. Springer, Heidelberg (2010)
58. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PASS: abstraction refinement for infinite probabilistic models. In: Esparza, J., Majumdar, R. (eds.) Proc. TACAS 2010. LNCS, vol. 6015, pp. 353–357. Springer, Heidelberg (2010)
59. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Form. Asp. Comput. **6**(5), 512–535 (1994)
60. Hartonas-Garmhausen, V., Campos, S., Clarke, E.: ProbVerus: probabilistic symbolic model checking. In: Katoen, J. (ed.) 5th International AMAST Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS). LNCS, vol. 1601, pp. 96–110. Springer, Heidelberg (1999)
61. Haverkort, B.: Performance of Computer Communication Systems: A Model-Based Approach. Wiley, Chichester (1998)
62. Heath, J., Kwiatkowska, M., Norman, G., Parker, D., Tymchyshyn, O.: Probabilistic model checking of complex biological pathways. Theor. Comput. Sci. **319**(3), 239–257 (2008)
63. Henzinger, T.A., Mateescu, M.: Propagation models for computing biochemical reaction networks. In: Fages, F. (ed.) Proc. CMSB'11, pp. 1–3. ACM, New York (2011)
64. Hermanns, H., Katoen, J.P.: The how and why of interactive Markov chains. In: de Boer, F.S., Bonsangue, M.M., Hallerstede, S., Leuschel, M. (eds.) FMCO'09. LNCS, vol. 6286, pp. 311–337. Springer, Heidelberg (2010)
65. Hermanns, H., Segala, R. (eds.): Proc. 2nd Joint Int. Workshop Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM-PROBMIV). LNCS, vol. 2399. Springer, Heidelberg (2002)
66. Hurd, J., McIver, A., Morgan, C.: Probabilistic guarded commands mechanized in HOL. Theor. Comput. Sci. **346**(1), 96–112 (2005)
67. Huth, M., Kwiatkowska, M.: Quantitative analysis and model checking. In: Proc. 12th Annual IEEE Symposium on Logic in Computer Science (LICS'97), pp. 111–122. IEEE, Piscataway (1997)
68. Itai, A., Rodeh, M.: Symmetry breaking in distributed networks. Inf. Comput. **88**(1), 60–87 (1990)
69. Iyer, P., Narasimha, M.: Probabilistic lossy channel systems. In: Bidoit, M., Dauchet, M. (eds.) TAPSOFT '97: Theory and Practice of Software Development. LNCS, vol. 1214, pp. 667–681. Springer, Heidelberg (1997)
70. Jeannet, B., d'Argenio, P.R., Larsen, K.G.: Rapture: A tool for verifying Markov Decision Processes. In: Tools Day'02, Technical Report. Masaryk University, Brno (2002)
71. Jonsson, B., Larsen, K., Yi, W.: Probabilistic extensions of process algebras. In: Bergstra, J.A., Pomse, A., Smolka, S.A. (eds.) Handbook of Process Algebra, pp. 685–710. Elsevier, Amsterdam (2001)
72. Karloff, H.: Linear Programming. Birkhäuser, Boston (1991)
73. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: Abstraction refinement for probabilistic software. In: Jones, N., Müller-Olm, M. (eds.) Proc. 10th International Conference on Verification, Model Checking, and Abstract Interpretation (VMCAI'09). LNCS, vol. 5403, pp. 182–197. Springer, Heidelberg (2009)
74. Kattenbelt, M., Kwiatkowska, M., Norman, G., Parker, D.: A game-based abstraction-refinement framework for Markov decision processes. Form. Methods Syst. Des. **36**(3), 246–280 (2010)
75. Kemeny, J., Snell, J.: Finite Markov Chains. Van Nostrand, Princeton (1960)

76. Komárková, Z., Křetínský, J.: Rabinizer 3: Safraless translation of LTL to small deterministic automata. In: Cassez, F., Raskin, J. (eds.) Automated Technology for Verification and Analysis—Proceedings of the 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3–7, 2014. LNCS, vol. 8837, pp. 235–241. Springer, Heidelberg (2014)
77. Kulkarni, V.: Modeling and Analysis of Stochastic Systems. Chapman & Hall, London (1995)
78. Kwiatkowska, M., Norman, G., Parker, D.: Symmetry reduction for probabilistic model checking. In: Ball, T., Jones, R.B. (eds.) Proc. of the 18th International Conference on Computer Aided Verification (CAV). LNCS, vol. 4144, pp. 234–248. Springer, Heidelberg (2006)
79. Kwiatkowska, M., Norman, G., Parker, D.: Using probabilistic model checking in systems biology. ACM SIGMETRICS Perform. Eval. Rev. **35**(4), 14–21 (2008)
80. Kwiatkowska, M., Norman, G., Parker, D.: Stochastic games for verification of probabilistic timed automata. In: Ouaknine, J., Vaandrager, F.W. (eds.) Proc. 7th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'09). LNCS, vol. 5813, pp. 212–227. Springer, Heidelberg (2009)
81. Kwiatkowska, M., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) Proc. 23rd International Conference on Computer Aided Verification (CAV'11). LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011)
82. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Assume-guarantee verification for probabilistic systems. In: Esparza, R.M.J. (ed.) 16th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 6015, pp. 23–37 (2010)
83. Kwiatkowska, M., Norman, G., Parker, D., Qu, H.: Compositional probabilistic verification through multi-objective model checking. Inf. Comput. **232**, 38–65 (2013)
84. Kwiatkowska, M., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. Theor. Comput. Sci. **282**, 101–150 (2002)
85. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: Hermanns, H., Segala, R. (eds.) Proc. 2nd Joint International Workshop on Process Algebra and Probabilistic Methods, Performance Modeling and Verification (PAPM/PROBMIV'02). LNCS, vol. 2399, pp. 169–187. Springer, Heidelberg (2002)
86. Kwiatkowska, M., Parker, D., Qu, H.: Incremental quantitative verification for Markov decision processes. In: Proc. IEEE/IFIP International Conference on Dependable Systems and Networks (DSN-PDS'11), pp. 359–370. IEEE, Piscataway (2011)
87. Larsen, K., Skou, A.: Bisimulation through probabilistic testing. Inf. Comput. **94**(1), 1–28 (1991)
88. Lassaigne, R., Peyronnet, S.: Approximate verification of probabilistic systems. In: [65], pp. 213–214 (2002)
89. Lassaigne, R., Peyronnet, S.: Approximate planning and verification for large Markov decision processes. In: Proceedings of the 27th Annual ACM Symposium on Applied Computing, SAC '12, pp. 1314–1319. ACM, New York (2012)
90. Lynch, N.: Distributed Algorithms. Morgan Kaufmann, San Francisco (1996)
91. McIver, A., Morgan, C.: Games, probability and the quantitative μ-calculus qMμ. In: Baaz, M., Voronkov, A. (eds.) Proc. LPAR 2002. LNCS, vol. 2514, pp. 292–310. Springer, Heidelberg (2002)
92. Mikeev, L., Sandmann, W., Wolf, V.: Numerical approximation of rare event probabilities in biochemically reacting systems. In: Gupta, A., Henzinger, T.A. (eds.) Proc. CMSB 2013. LNCS, vol. 8130, pp. 5–18. Springer, Heidelberg (2013)
93. Mio, M.: Probabilistic modal μ-calculus with independent product. Log. Methods Comput. Sci. **8**(4), 1–36 (2012)

94. Norman, G., Parker, D., Kwiatkowska, M., Shukla, S., Gupta, R.: Using probabilistic model checking for dynamic power management. In: Leuschel, M., Gruner, S., Presti, S.L. (eds.) Proc. 3rd Workshop on Automated Verification of Critical Systems (AVoCS'03), Technical Report DSSE-TR-2003-2, University of Southampton, pp. 202–215 (2003)
95. Norman, G., Parker, D., Sproston, J.: Model checking for probabilistic timed automata. Form. Methods Syst. Des. **43**(2), 164–190 (2013)
96. Norris, J.R.: Markov chains. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, Cambridge (1998)
97. Pnueli, A., Zuck, L.: Probabilistic verification by tableaux. In: Proc. Annual Symposium on Logic in Computer Science (LICS), pp. 322–331. IEEE, Piscataway (1986)
98. PRISM web site. www.prismmodelchecker.org. Accessed 20 August 2013
99. Puterman, M.: Markov Decision Processes: Discrete Stochastic Dynamic Programming. Wiley, New York (1994)
100. Schrijver, A.: Combinatorial Optimization: Polyhedra and Efficiency. Springer, Heidelberg (2003)
101. Segala, R.: Modeling and verification of randomized distributed real-time systems. Ph.D. thesis, Massachusetts Institute of Technology (1995)
102. Segala, R., Lynch, N.: Probabilistic simulations for probabilistic processes. In: Jonsson, B., Parrow, J. (eds.) Proc. CONCUR '94. LNCS, vol. 836, pp. 481–496. Springer, Heidelberg (1994)
103. Sen, K., Viswanathan, M., Agha, G.: Model-checking Markov chains in the presence of uncertainties. In: Hermanns, H., Palsberg, J. (eds.) 12th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS). LNCS, vol. 3920, pp. 394–410. Springer, Heidelberg (2006)
104. Sharygina, N., Veith, H. (eds.): Computer Aided Verification—Proceedings of the 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13–19, 2013. LNCS, vol. 8044. Springer, Heidelberg (2013)
105. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: Proc. 26th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 327–338. IEEE, Piscataway (1985)
106. Vardi, M.Y., Wolper, P.: An automata-theoretic approach to automatic program verification. In: Symposium on Logic in Computer Science (LICS'86), pp. 332–345. IEEE, Piscataway (1986)
107. Vardi, M.Y., Wolper, P.: Reasoning about infinite computations. Inf. Comput. **115**(1), 1–37 (1994)
108. Younes, H., Kwiatkowska, M., Norman, G., Parker, D.: Numerical vs. statistical probabilistic model checking. Int. J. Softw. Tools Technol. Transf. **8**(3), 216–228 (2006)
109. Younes, H., Simmons, R.: Probabilistic verification of discrete event systems using acceptance sampling. In: Brinksma, E., Larsen, K.G. (eds.) Proc. 14th International Conference on Computer Aided Verification (CAV). LNCS, vol. 2404, pp. 223–235. Springer, Heidelberg (2002)
110. 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, Heidelberg (2007)