Chapter 13: Model Checking Linear-Time Properties of Probabilistic Systems

Christel Baier, Marcus Größer, and Frank Ciesinski

Technische Universität Dresden, Fakultät Informatik,
Institut für Theoretische Informatik,
01062 Dresden, Germany
baier@tcs.inf.tu-dresden.de
groesser@tcs.inf.tu-dresden.de
ciesinsk@tcs.inf.tu-dresden.de

1 Introduction

This chapter is about the verification of Markov decision processes (MDPs) which are one of the fundamental models for reasoning about probabilistic phenomena of computer systems. MDPs have first been studied by Bellmann [14] and Howard [54] in the 1950s. While this early work on MDPs was motivated by optimization problems that appear in the context of operations research, nowadays MDPs are used in a variety of areas, including robotics, stochastic planning, control theory, reinforcement learning, economics, manufacturing, and semantics of randomized protocols. In the context of finite-state acceptors and transducers, MDPs served as basis for the introduction of probabilistic automata [81, 73], which again interact with the theory of weighted automata by generalizing the concept of probabilities to weights in an arbitrary semiring.

In this chapter, Markov decision processes are used as an operational model for "probabilistic systems". Probabilism appears as a rather natural concept when providing the semantics of randomized algorithms or multi-agent systems with unreliable hardware components. In randomized algorithms, coin-tossing is used as an explicit probabilistic algorithmic concept. Examples for sequential randomized algorithms are the prominent primality tests by Miller and Rabin or by Solovay and Strassen or algorithms that operate with randomized data structures such as skip lists or universal hashing. For distributed systems, there is a wide range of coordination protocols to solve, e.g., mutual exclusion or leader election problems that utilize coin-tossing actions for symmetry breaking (see [44, 68]). For systems that operate with faulty components, such as communication channels that might corrupt or lose messages, or sensors that deliver wrong values in rare cases, probabilities can be used to specify the frequency of such exceptional behaviors. Probabilistic semantic models also play a central role in reasoning about systems that interact with an environment on which only partial information by means of stochastic assumptions about the I/O-operations of its interface is available.

The most popular operational models that support reasoning about probabilistic behaviors are Markovian models (Markov chains or Markov decision processes) where discrete probabilities are attached to the transitions. They enjoy the memoryless property stating that the future system evolution just depends on the current system state, but not on the specific steps that have been performed in the past. The memoryless property is inherent also in most non-stochastic automata models, such as labeled transition systems or weighted automata. In the stochastic setting, however, the memoryless property asserts that not only the enabled actions and the successor states are uniquely determined by the current state, but also the probabilities for the transitions. Markov chains are purely probabilistic, i.e., the possible behavior in each state is specified by a probabilistic distribution for the successor states. They can serve to formalize the stepwise behavior of sequential randomized algorithms. For modeling probabilistic parallel systems, Markov chains are not expressive enough to provide an *interleaving semantics*, which relies on the representation of concurrent (independent) actions α and β, executed by different processes, by a non-deterministic choice between the action sequences $\alpha\beta$ and $\beta\alpha$. Thus, models where probabilism and non-determinism co-exist are required to provide an operational semantics of randomized distributed algorithms. Stochastic models with nondeterminism are also needed for abstraction purposes (e.g., in the context of data abstraction, data-dependent conditional branching might be replaced with non-deterministic branching) or to model the potential interactions with an unpredictable environment (e.g., a human user). The operational semantics of such systems with probabilistic and non-deterministic behaviors can be described by a *Markov decision process* which is a stochastic model where the behavior in each state s is given

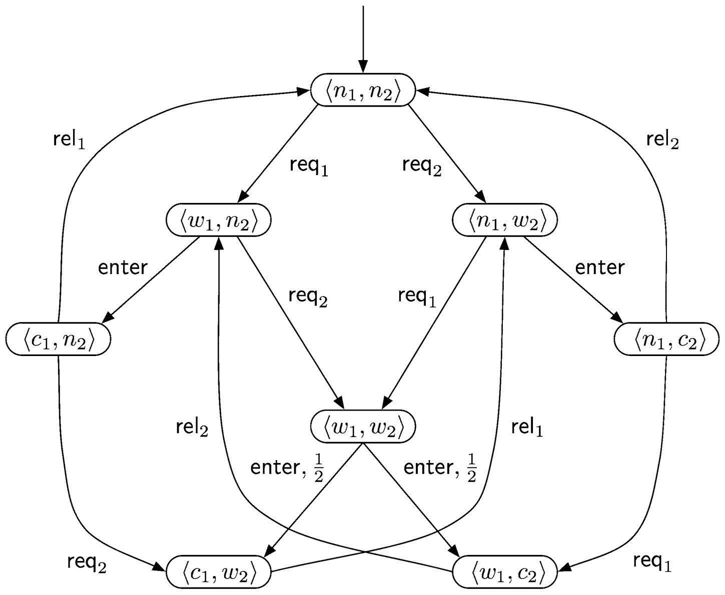

$\langle n_1, n_2\rangle$
$\langle w_1, n_2\rangle$
$\langle n_1, w_2\rangle$
$\langle c_1, n_2\rangle$
$\langle n_1, c_2\rangle$
$\langle w_1, w_2\rangle$
$\langle c_1, w_2\rangle$
$\langle w_1, c_2\rangle$
rel$_1$
rel$_2$
req$_1$
req$_2$
enter
enter
req$_2$
req$_1$
rel$_2$
rel$_1$
enter, $\frac{1}{2}$
enter, $\frac{1}{2}$
req$_2$
req$_1$

Fig. 1. MDP for a randomized mutual exclusion protocol

by a set of enabled actions which are augmented by distributions that specify the probabilistic effect of the execution of the enabled actions in state s. The idea is that when entering state s, first an enabled action α is chosen nondeterministically which is then executed and the associated distribution yields the probability for the next state. A Markov chain arises as a special case of an MDP where for each state the set of enabled actions is a singleton.

Example 1.1 (A randomized mutual exclusion protocol). We consider here a simple randomized mutual exclusion protocol for two concurrent processes $\mathcal{P}_1$ and $\mathcal{P}_2$. When process $\mathcal{P}_i$ is in its non-critical section (represented by location n_i), it can perform a request (via action req_i) and then move to a waiting section (location w_i). In the waiting section, the process waits for permission given by an arbiter to enter its critical section (location c_i) from where it can release and move to its non-critical section again (action rel_i). The arbiter that coordinates the access to the critical sections is randomized and permits process $\mathcal{P}_i$ to enter its critical section if the other process is in its non-critical section. If both processes request access to the critical section, the arbiter tosses a fair coin to decide which of the two processes has to wait and which process may enter the critical section. The composite behavior of the two concurrent processes and the arbiter can be modeled by the MDP depicted in Fig. 1. Note that the actions req, enter, and rel do not have a proper probabilistic effect, and thus yield a unique successor (the transition probabilities that equal 1 are omitted from the figure). Only in state $\langle w_1, w_2\rangle$ there is a proper probabilistic choice performed by the arbiter to select the next process to enter the critical section. In all other states of the MDP where at least one of the processes is in its non-critical location, there is a non-deterministic choice between the enabled actions of the processes $\mathcal{P}_1$ and $\mathcal{P}_2$.

One can think of an MDP as a directed graph where the edges are augmented with action names and probabilities, i.e., positive real numbers ≤ 1, satisfying the side condition that for each state s and action name α either the probabilities attached to the outgoing α-transitions of s sum up to 1 (which ensures that they represent a probabilistic distribution) or s has no outgoing α-transitions at all (in which case α is not enabled in s). Thus, an MDP can be seen as a special instance of a *weighted automaton* where the weights are elements of the interval $]0,1]$. Applying the classical approach of weighted automata (see Chap. 3 of this handbook [35]) to an MDP (where we deal with the semiring $[0,1]$ with standard multiplication and where maximum serves as plus-operation), the weight of a finite path π is obtained by the product of the weights of the transitions on π and can be understood as the probability for π. To reason about the probabilities for properties over infinite paths (which are crucial for properties that impose conditions on the "long-run behaviors" such as liveness properties), the special interpretation of the weights as probabilities permits us to apply standard techniques of measure and probability theory. More precisely, the standard approach to define probabilities for events in an MDP relies on the sigma-algebra over infinite paths generated by the cylinder sets spanned by finite paths, and the probability measure is defined using Carathéodory's measure extension theorem. A summary of the relevant measure-theoretic concepts is presented in the appendix; see Sect. 8. This is an alternative approach to the interpretation of infinite words over weighted automata discussed in [35, 31] where special algebraic assumptions on the underlying semiring are made in order to define the weights of infinite paths and words. In particular, the semiring has to permit an infinite sum as well as a countably infinite product operation satisfying several commutativity, associativity, and distributivity laws. Another approach to interpret infinite words over weighted automata uses discounting, which is a well-known concept in mathematical economics as well as systems theory in which later events get less value than earlier ones [32–34].

The typical task for verifying a probabilistic system modeled by an MDP is to prove that certain temporal properties hold *almost surely*, i.e., with probability 1, or with some *high probability*, no matter how the non-determinism is resolved. The notion *qualitative property* is used when a certain event is required to hold almost surely (or dually with zero probability). Thus, qualitative properties assert that a certain path condition E holds for almost all or almost no paths. Depending on the type of condition E, the concept of qualitative properties is different from reasoning by means of purely functional properties that require a certain event E to hold for all paths or for no path. For example, if B is a set of states, then the qualitative reachability property asserting that "almost all paths will enter a state in B" is slightly weaker than the functional reachability property requiring that "all paths will enter a state in B". The notion *quantitative property* refers to a condition that requires a lower or upper bound in the open interval $]0,1[$ for the probability of a certain event or imposes some conditions on the mean value of a random function. For instance, typical requirements for a randomized mutual exclusion protocol are the qualitative property stating that "almost surely each waiting process will eventually get the grant to enter its critical section" and the quantitative property stating that "each waiting process has the chance of at least 99% to enter its critical section after having announced its demand five times". In the case of a communication protocol with a lossy channel, a natural quantitative requirement could be that "with probability ≥ 0.98, any message will be delivered correctly after at most three attempts to send it".

When speaking about probability bounds in the context of verification methods for MDPs, we range over all possible resolutions of the non-determinism. This corresponds to a *worst-case* analysis. Thus, if the given MDP is an interleaving model for a probabilistic multi-processor system, then the worst-case analysis ranges over all orders of concurrent actions and does not impose any restrictions on the relative speed of the processors. In cases where some non-deterministic choices in an MDP stand for the potential behaviors of an unknown environment, the worst-case analysis takes all possible activities of the environment into account. Similarly, when certain non-deterministic branches result from abstractions, then the worst-case analysis covers all concrete behaviors. By requiring that the probabilities for a given event E are 1 or sufficiently close to 1, the event E is supposed to characterize the "good" (desired) behaviors. Verification problems for MDPs can also be rephrased in the opposite way: if E describes the "bad" (undesired) behaviors, then the goal is to prove that E holds with probability 0 or some sufficiently small probability.

The notion *qualitative analysis* is used if the goal is to show that a certain event E appears with probability 0 or 1, while the notion *quantitative analysis* refers to the task of computing the maximal or minimal probabilities for E, when ranging over all schedulers, i.e., instances that resolve the non-determinism. Sometimes, some mild conditions on the resolution of non-deterministic choices along infinite paths are imposed, such as fairness assumptions. Other instances of a quantitative analysis are obtained when the goal is to establish lower or upper bounds for certain mean values, e.g., the average power consumption of a complex task or the expected number of rounds required to find a leader when imposing a leader election protocol. Such variants of the verification problem for MDPs will not be addressed in this chapter. Instead, we will concentrate on the quantitative reasoning by means of extremal probabilities under the full class of schedulers.

In the literature, many variants of classical temporal logics for non-probabilistic systems have been adapted to specify the requirements of a probabilistic system. One prominent example is the probabilistic variant of computation tree logic (PCTL) [47, 15, 12] which yields an elegant formalism to specify lower or upper probability bounds for reachability properties within a logical framework. While PCTL is a representative for logics that are based on the branching time view, one can also use purely path-based formalisms, such as linear temporal logic (LTL) or automata over infinite words, to specify the desired/good or undesired/bad event E, which is then the subject of a qualitative or quantitative analysis [79, 90, 91, 23]. For finite-state MDPs, the quantitative analysis against PCTL or LTL specifications mainly relies on a combination of graph algorithms, automata-based constructions and (numerical) algorithms for solving linear programs. Consequently, compared to the non-probabilistic case, there is the additional difficulty to solve linear programs, and also the required graph algorithms are more complex. This renders the state space explosion problem even more serious than in the non-probabilistic case and the feasibility of algorithms for the quantitative analysis crucially depends on good heuristics to increase efficiency. Among other features, the tool PRISM [61] contains a PCTL model checker for MDPs which has been successfully applied to, e.g., a series of randomized coordination algorithms, communication, and security protocols. To tackle the state space explosion problem, PRISM uses a tricky combination of data structures (multi-terminal binary decision diagrams and sparse matrices) for the internal representation of the MDP and the numerical computations required for a quantitative analysis [72]. Motivated by the success of the non-probabilistic model checker SPIN [51] where partial order reduction techniques [74, 52, 87, 38] for the verification of interleaving models are realized, the concept of partial order reduction has been adapted for the quantitative analysis of MDPs against linear-time specifications [9, 26] and implemented in the model checker LIQUOR [3, 5]. Several other techniques that attempt to speed up the verification algorithms for MDPs and/or to decrease the memory requirements have been proposed in the literature, such as symmetry reduction [62], iterative abstraction-refinement algorithms [25, 50], reduction techniques for linear programs [24, 5], and many others. Most of these techniques are orthogonal to the symbolic approach of PRISM and the partial order reduction approach of LIQUOR and can be applied in combination with them.

Somehow dual to verification problems are *controller synthesis problems* where one is typically interested in *best-case* scenarios (rather than the worst case) and attempts to construct a scheduler where the probabilities for the desired behaviors, say formalized by a linear-time property E, are maximal. Assuming that all non-deterministic choices are controllable and that the controller has complete knowledge of the computation leading to the current state, then the methods for computing maximal probabilities for the event E can easily be extended for constructing a scheduler that maximizes the probabilities for E. However, the assumption that complete knowledge about the history is available is unrealistic for multi-agent systems when controllers for a single agent or a coalition of agents are wanted. In this case, the adequate model are *partially observable* MDPs [84, 70, 71] that extend ordinary MDPs by an equivalence relation $\sim$ on the states. The idea is that equivalent states are not

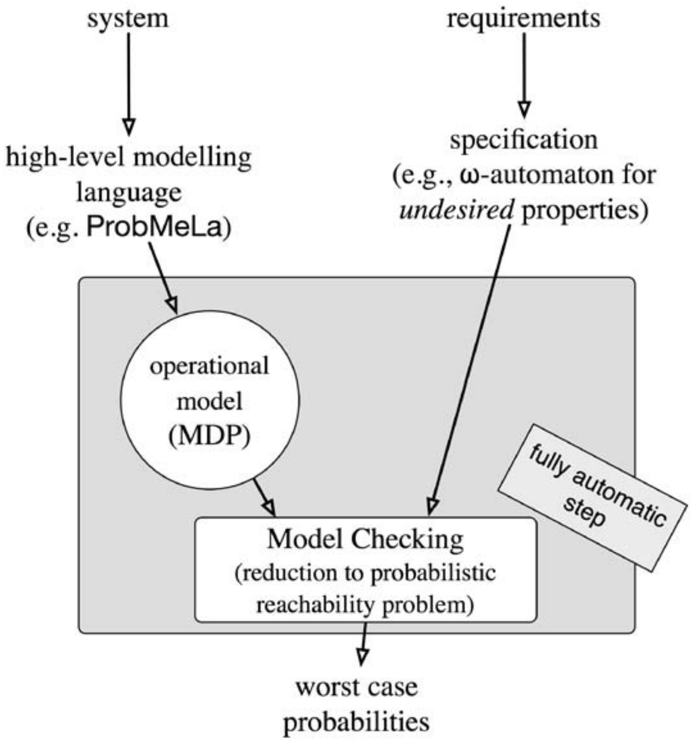

system
requirements
high-level modelling language (e.g. ProbMeLa)
specification (e.g., ω-automaton for undesired properties)
operational model (MDP)
fully automatic step
Model Checking (reduction to probabilistic reachability problem)
worst case probabilities

Fig. 2. Schema for the quantitative analysis

distinguishable by the controller and the task is to construct a controller (i.e., *observation-based scheduler*) for the partially observable MDP that maximizes the probabilities for E, where "observation-based" means that the scheduler can only observe the equivalence classes of the states that have been visited in the past, but not the states themselves. In general, the controller synthesis problem cannot be solved algorithmically as there is an undecidability result for the qualitative controller synthesis problem that asks for the existence of an observation-based scheduler where the probability to visit a certain set of states infinitely often is positive [2]. However, for simpler properties (e.g., safety properties) and special patterns of liveness properties, the qualitative controller synthesis problem is decidable.

About This Chapter

In the remaining sections of this chapter, we will present the main concepts of Markov decision processes as an operational model for probabilistic systems and present the basic steps for the (qualitative or quantitative) analysis against linear-time properties. Branching time properties will not be addressed here. For the basic steps to verify PCTL-like specifications for MDPs and the symbolic MTBDD-based approach, we refer to [82] and Chap. 10 in [11], as well as the literature mentioned there. We will start in Sect. 2 with the formal definition of an MDP and related notions (paths, schedulers and their induced probability measure) and illustrate the use of MDPs as an operational model for probabilistic systems by means of a few examples. The core problem of any quantitative analysis in an MDP is the problem of computing extremal reachability probabilities by means of optimization techniques. This will be explained in Sect. 3. The general case of ω-regular properties will be addressed in Sect. 4. We follow here the automata-based approach [90, 91, 23, 12] where we are given a deterministic ω-automata representing a linear-time property E. The maximal probability for E can be computed by a product construction and a reduction to the probabilistic reachability problem (see Fig. 2). The purpose of Sect. 5 is to explain the partial order reduction approach that attempts to derive the maximal probabilities for E from a "small" fragment of the MDP, thus avoiding the construction and analysis of the full MDP. In Sect. 6, we introduce the model of partially observable MDPs and report on results for special instances of the qualitative controller synthesis problem. Some concluding remarks are given in Sect. 7. The chapter ends with an appendix (Sect. 8) that contains the definition of Markov chains and explains the mathematical details of the stochastic process induced by a scheduler of an MDP.

2 Markov Decision Processes

Throughout this chapter, we will use Markov decision processes (MDPs) as an operational model for probabilistic systems. As in [80, 65, 27], the states of an MDP might have several enabled actions. Each of the actions that are enabled in state s is associated with a probability distribution which yields the probabilities for the successor states. This corresponds to the so-called reactive model in the classification of [89]. In addition, we assume here a labeling function that attaches to any state s a set of atomic propositions that are assumed to be fulfilled in state s. The atomic propositions will serve as atoms in the formal specifications for properties. For instance, to formalize deadlock freedom "processes $\mathcal{P}_1$ and $\mathcal{P}_2$ are never simultaneously in their critical sections" or starvation freedom "whenever $\mathcal{P}_1$ is in his waiting section, then $\mathcal{P}_1$ will eventually enter its critical section" for the randomized mutual exclusion protocol in Fig. 1, we can deal with temporal formulas that use the atomic propositions $\mathsf{wait}_i, \mathsf{crit}_i$ for $i = 1, 2$ which are attached to all states where the local state of process $\mathcal{P}_i$ is w_i or c_i, respectively. We will now give the formal definition of an MDP. For further basic definitions of, e.g., probability distribution, or Markov chain, we refer to the appendix (Sect. 8).

Definition 2.1 ((State-labeled) Markov decision process (MDP)). *A Markov decision process is a tuple*

$$\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L),$$

where:

- *S is a finite non-empty set of states.*
- Act *is a finite non-empty set of actions.*
- *$\delta : S \times \mathsf{Act} \times S \to [0, 1]$ is a transition probability function such that for each $s \in S$ and $\alpha \in \mathsf{Act}$, either $\delta(s, \alpha, .)$ is a probability distribution on S or $\delta(s, \alpha, .)$ is the null-function (i.e., $\delta(s, \alpha, t) = 0$ for any $t \in S$).*
- *μ is a probability distribution on S (called the initial distribution).*
- AP *is a finite set of atomic propositions.*
- *$L : S \to 2^{\mathsf{AP}}$ is a labeling function that labels a state s with those atomic propositions in* AP *that are supposed to hold in s.*

$\mathsf{Act}(s) = \{\alpha \in \mathsf{Act} \mid \exists t \in S : \delta(s, \alpha, t) > 0\}$ *denotes the set of actions that are enabled in state s. We require that* $\mathsf{Act}(s)$ *is non-empty for each state $s \in S$.*

The intuitive operational behavior of an MDP is the following. If s is the current state, then at first one of the actions $\alpha \in \mathsf{Act}(s)$ is chosen non-deterministically. Secondly, action α is executed leading to state t with probability $\delta(s, \alpha, t)$.

Action α is called a *probabilistic* action if it has a random effect, i.e., if there is at least one state s where α is enabled and that has two or more α-successors (an α-successor of state s is a state t such that $\delta(s, \alpha, t) > 0$). Otherwise, α is called *non-probabilistic.*

If all actions in Act are non-probabilistic and the initial distribution is a Dirac distribution, i.e., a probabilistic distribution that assigns probability 1 to some particular state, then our notion of an MDP reduces to an ordinary transition system with at most one outgoing α-transition per state and action α and exactly one initial state.

Example 2.2 (The Monty Hall problem). Before we proceed, let us have a look at a small example of an MDP. We consider here the Monty Hall problem: "Suppose you are a contestant on a game show, and you are given the choice of three doors: behind one door is a car, behind the others, goats. You choose a door, but you do not open it. Then the host, who knows what is behind the doors, has to open another door which has a goat behind it (if you initially picked the door with the car behind it, the host randomly chooses one of the other doors). He then asks whether you want to change your choice to the other unopened door. After either sticking to your first choice or switching to the other unopened door, you win what is behind the door that you have finally chosen. Considering that your goal is to win the car, is it to your advantage to switch your first choice?"

In Fig. 3, we depict an MDP for an abstraction of this problem where due to symmetry the information which exact door reveals the car is neglected. So, in the initial state s, the contestant chooses one of the three doors, each with equal probability $\frac{1}{3}$. State t_1 represents the case where the contestant has

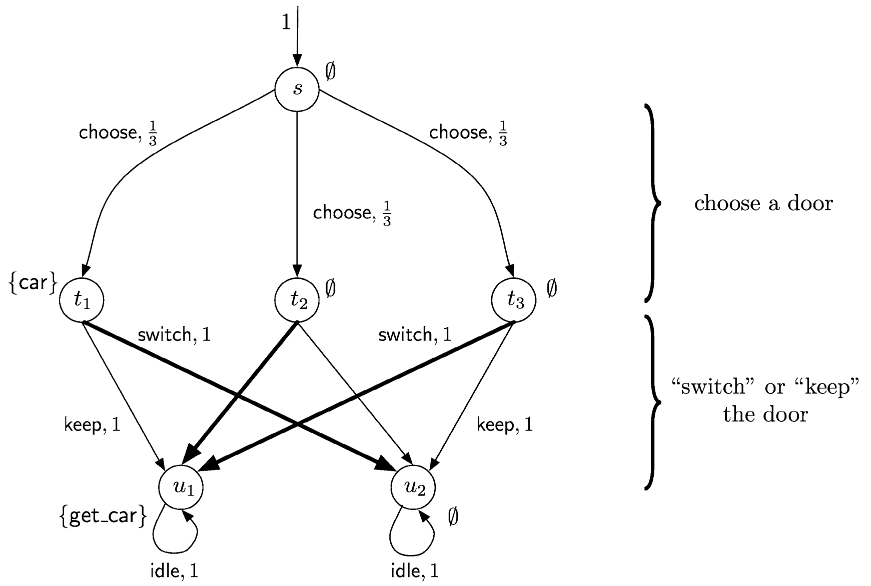


Fig. 3. The abstract "Monty Hall problem"

chosen the door with the car behind it, states t_2 and t_3 represent the other cases where the contestant has chosen a door with a goat behind it. Thus, state t_1 is labeled with the atomic proposition car, whereas t_2 and t_3 are labeled with $\emptyset$. After this first choice, the game show host opens a door which has a goat behind it. Note that now there are exactly two closed doors, one with a car behind it (represented by state u_1 and the labeling $\{\mathsf{get\_car}\}$) and one door with a goat behind it (represented by state u_2 and the labeling $\emptyset$). Now the contestant has the alternative to either stick to her/his chosen door or to switch to the other closed door. So, in states t_1, t_2, and t_3, there is a non-deterministic choice between the actions switch and keep, leading to u_1 or u_2. Note that the actions switch and keep are non-probabilistic as there are exactly two closed doors. After choosing switch or keep, the game is over. To complete the MDP, we added an idling self-loop to the states u_1 and u_2. For the sake of readability, we depict the transitions of the switch-action a little thicker and we omit the labeling of the actions of state t_2.

Paths and Schedulers of an MDP

Definition 2.3 (Path and corresponding notations). *An infinite path of an MDP is an infinite sequence* $\pi = ((s_0, \alpha_1), (s_1, \alpha_2), \ldots) \in (S \times \mathsf{Act})^\omega$ *such that* $\delta(s_i, \alpha_{i+1}, s_{i+1}) > 0$ *for* $i \in \mathbb{N}_{\geq 0}$. *We write paths in the form*

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

A finite path is a finite prefix of an infinite path that ends in a state. We use the notation $\mathsf{last}(\pi)$ *for the last state of a finite path* π *and* $|\pi|$ *for the length (number of actions) of a finite path. We denote by* $\mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}}$, *resp.* $\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}$, *the set of all finite, resp. infinite, paths of* $\mathcal{M}$.

In order to be able to talk about the probability, e.g., to get the car in the Monty Hall problem, we need another concept of the theory of MDPs, namely the concept of schedulers. Schedulers are a means to resolve the non-determinism in the states, and thus yield a discrete Markov chain and a probability measure on the paths. Intuitively, a scheduler takes as input the "history" of a computation (formalized by a finite path π) and chooses the next action (resp. a distribution on actions).

Definition 2.4 (Scheduler). *For a given MDP* $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$, *a history dependent randomized scheduler is a function*

$$\mathcal{U} : \mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}} \to \mathsf{Distr}(\mathsf{Act}),$$

such that $\mathsf{supp}(\mathcal{U}(\pi)) \subseteq \mathsf{Act}(\mathsf{last}(\pi))$ *for each* $\pi \in \mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}}$. *Here,* $\mathsf{Distr}(\mathsf{Act})$ *denotes the set of probability distributions on* Act *while* $\mathsf{supp}(\mathcal{U}(\pi))$ *denotes the support of* $\mathcal{U}(\pi)$, *i.e., the set of actions* $\alpha \in \mathsf{Act}$ *such that* $\mathcal{U}(\pi)(\alpha) > 0$.

A scheduler $\mathcal{U}$ is called *deterministic*, if $\mathcal{U}(\pi)$ is a Dirac distribution for each $\pi \in \mathsf{Path}_{\mathsf{fin}}$, i.e., $\mathcal{U}(\pi)(\alpha) = 1$ for some action α, while $\mathcal{U}(\pi)(\beta) = 0$ for every other action $\beta \neq \alpha$. Scheduler $\mathcal{U}$ is called *memoryless*, if $\mathcal{U}(\pi) = \mathcal{U}(\mathsf{last}(\pi))$ for each $\pi \in \mathsf{Path}_{\mathsf{fin}}$ (note that $\mathsf{last}(\pi)$ is a path of length 0). We denote by Sched, the set of all (history dependent, randomized) schedulers. We write $\mathsf{Sched}_{\mathsf{D}}$ to denote the set of deterministic schedulers, $\mathsf{Sched}_{\mathsf{M}}$ for the set of memoryless schedulers, and $\mathsf{Sched}_{\mathsf{MD}}$ for the set of memoryless deterministic schedulers. Note that the following inclusions hold.

- $\mathsf{Sched}_{\mathsf{M}} \subseteq \mathsf{Sched}$ and $\mathsf{Sched}_{\mathsf{D}} \subseteq \mathsf{Sched}$
- $\mathsf{Sched}_{\mathsf{MD}} \subseteq \mathsf{Sched}_{\mathsf{M}}$ and $\mathsf{Sched}_{\mathsf{MD}} \subseteq \mathsf{Sched}_{\mathsf{D}}$

A (finite or infinite) path $s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ is called a $\mathcal{U}$-path, if $\mathcal{U}(s_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_i} s_i)(\alpha_{i+1}) > 0$ for every $0 \leq i < |\pi|$.

Given an MDP $\mathcal{M}$ and a scheduler $\mathcal{U}$, the behavior of $\mathcal{M}$ under $\mathcal{U}$ can be formalized by a (possibly infinite-state) discrete Markov chain. By $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}$, we denote the standard probability measure on the standard σ-algebra of the infinite paths of $\mathcal{M}$. Given a state s of $\mathcal{M}$, we denote by $\mathsf{Pr}_s^{\mathcal{M},\mathcal{U}}$ the probability measure that is obtained if $\mathcal{M}$ is equipped with the initial Dirac distribution μ_s, with $\mu_s(s) = 1$. A detailed definition of $\mathcal{M}$ and $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}$ is provided in the appendix (Sect. 8). We also fix the following notation for convenience. Given an MDP $\mathcal{M}$, a scheduler $\mathcal{U}$, and a measurable path property E, we will write

$$\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(E) \overset{\text{def}}{=} \mathsf{Pr}^{\mathcal{M},\mathcal{U}}\big(\{\pi \in \mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}} \mid \pi \text{ satisfies } E\}\big)$$

for the probability that the property E holds in $\mathcal{M}$ under the scheduler $\mathcal{U}$.

Let us consider again the Monty Hall problem depicted in Fig. 3. It is easy to see that under the scheduler $\mathcal{U}$ with[1]

$$\mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{keep}) = \mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{switch}) = \mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{switch}) = 1$$

the contestant will win the car with probability 1. This is, of course, an unrealistic scheduler as it uses the knowledge of whether the contestant has chosen the door with the car behind it or not. If the door with the car was chosen (state t_1), then the scheduler decides to keep the door, if a door with a goat was chosen (states t_2, t_3), then the scheduler decides to switch the door. As the contestant does not know whether he has chosen the door with the car behind it, the only realistic schedulers (that model a contestant's choice) are the two schedulers $\mathcal{U}_\mathsf{s}$ and $\mathcal{U}_\mathsf{k}$ with

$$\mathcal{U}_\mathsf{s}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{switch}) = \mathcal{U}_\mathsf{s}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{switch}) = \mathcal{U}_\mathsf{s}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{switch}) = 1$$

and

$$\mathcal{U}_\mathsf{k}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{keep}) = \mathcal{U}_\mathsf{k}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{keep}) = \mathcal{U}_\mathsf{k}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{keep}) = 1$$

where the contestant either decides to switch the door or to keep it. Simple computations show that with $\mathcal{M}$ being the MDP of Fig. 3

$$\begin{aligned}
\mathsf{Pr}^{\mathcal{M},\mathcal{U}_\mathsf{s}}(\Diamond\ \mathsf{get\_car}) = \mathsf{Pr}^{\mathcal{M},\mathcal{U}_\mathsf{s}}\big(\big\{ & s \xrightarrow{\mathsf{choose}} t_1 \xrightarrow{\mathsf{keep}} u_1 \xrightarrow{\mathsf{idle}} \cdots, \\
& s \xrightarrow{\mathsf{choose}} t_2 \xrightarrow{\mathsf{switch}} u_1 \xrightarrow{\mathsf{idle}} \cdots, \\
& s \xrightarrow{\mathsf{choose}} t_3 \xrightarrow{\mathsf{switch}} u_1 \xrightarrow{\mathsf{idle}} \cdots \big\}\big) \\
= {} & 1 \cdot \tfrac{1}{3} \cdot 0 \cdot 1 + 1 \cdot \tfrac{1}{3} \cdot 1 \cdot 1 + 1 \cdot \tfrac{1}{3} \cdot 1 \cdot 1 \\
= {} & \tfrac{2}{3}
\end{aligned}$$

and similarly

$$\mathsf{Pr}^{\mathcal{M},\mathcal{U}_\mathsf{k}}(\Diamond\ \mathsf{get\_car}) = \frac{1}{3}.$$

This shows that it is an advantage to switch the door. Here, we used the LTL-like notation $\Diamond$ get_car to denote the reachability property stating that eventually a state where get_car holds will be reached. That is, given the atomic proposition get_car, a path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots$ satisfies $\Diamond$ get_car if and only if there exists an index $i \in \mathbb{N}_{\geq 0}$ such that $\mathsf{get\_car} \in L(s_i)$.

Specifying Systems with an MDP-Semantics

In the literature, various modeling languages for probabilistic systems whose stepwise behavior can be formalized by an MDP have been proposed. Such languages can serve as a starting point for model checking tools that take as

[1] Note that this completely determines the scheduler as there are no other paths that end in a state with more than one enabled action.

```
msg = ...; // the data to be sent
msg_sent = false;
do:: msg_sent == false ->
        pif :0.9: -> msg_sent = true;
            :0.1: -> skip
        fip
od
```

Fig. 4. ProbMeLa-code for an unbounded retransmission protocol

input a description of the processes of a (parallel) system and a formalization of the property to be checked. Semantic rules are used to construct the MDP automatically. Then in further steps, model checking algorithms are applied. An example for such a modeling language is ProbMeLa [4], which is the input language of the model checker LIQUOR that supports the qualitative and quantitative analysis of MDPs against linear-time properties. ProbMeLa is a probabilistic version of the process meta-language ProMeLa which serves as modeling language in connection with the prominent model checker SPIN for non-probabilistic systems [51]. The core language of ProMeLa and ProbMeLa relies on Dijkstra's guarded commands `guard -> cmd` which can be used in loops (`do` ... `od`) and conditional commands (`if` ... `fi`). One of the additional probabilistic features of ProbMeLa is a probabilistic choice operator (`pif` ... `fip`) which can be seen as a probabilistic variant of conditional commands as probabilities are assigned to the commands rather than Boolean guards. Commands with the probabilistic choice operator have the form

$$\begin{array}{l} \texttt{pif :}\mathtt{p}_1\texttt{: -> cmd}_1\texttt{;} \\ \qquad\qquad \vdots \\ \quad\;\; \texttt{:}\mathtt{p}_n\texttt{: -> cmd}_n \\ \texttt{fip} \end{array}$$

where the $\mathtt{p}_i$'s are values in $]0, 1]$ that sum up to 1 and specify a probabilistic distribution for the commands $\mathtt{cmd}_i$.

For instance, the ProbMeLa-code in Fig. 4 might be a fragment of an unbounded retransmission protocol where some process sends data (variable `msg`) over a faulty medium such as a wireless radio connection or an otherwise noisy connection. This ProbMeLa-code specifies a process that iteratively attempts to send the message until it has been delivered correctly, where in each iteration of the loop the message will be delivered with probability 0.9 and will be lost with probability 0.1 (which is modeled here by the command `skip`). Termination of this protocol is guaranteed with probability 1 which means that almost surely the message will be delivered eventually. Note that there is an infinite computation where the message is lost in all iterations, but the probability for this to happen is 0. This simple example illustrates the difference between the functional and qualitative properties: the functional property requiring termination along all paths does not hold for the protocol,

```
do
 pif:0.5:-> first_fork = left; second_fork = right
    :0.5:-> first_fork = right; second_fork = left
 fip
 if ::forks[first_fork]==false -> forks[first_fork]=true;
      if
       ::forks[second_fork]==false -> forks[second_fork]=true;
         // philosopher is eating
         forks[second_fork]=false; forks[first_fork]=false
       ::forks[second_fork]==true -> forks[first_fork]=false
      fi
    ::forks[first_fork]==true -> skip
 fi
od
```

Fig. 5. ProbMeLa-code for each philosopher

while the qualitative property requiring termination along almost all paths (i.e., with probability 1) holds. Furthermore, we can establish the quantitative property stating that with probability 0.999 the message will be delivered within the first three iterations.

In the context of distributed systems, such as mutual exclusion protocols, leader election, or Byzantine agreement, coin tossing offers an elegant way to design coordination algorithms that treat all processes equally, but avoid deadlock situations since symmetry breaking is inherent in the random outcomes of the coin tossing experiment. A prominent example is the randomized solution of the dining philosophers problem suggested by Lehmann and Rabin [66]. The philosophers are sitting at a round table and neighboring philosophers have access to a shared resource (a fork). Each philosopher attempts to alternate infinitely often between a thinking and an eating phase, where the latter requires that the philosopher has picked up the fork to his left and the fork to his right. The Lehmann–Rabin protocol works as follows. As soon as a philosopher gets hungry, he decides to pick up his left or right fork by tossing a fair coin. If the selected fork is available, then he picks up this fork and checks whether the other fork is available, also. If so, then he takes it and starts to eat. Otherwise, he returns the taken fork, and repeats the whole procedure. This procedure can be described in ProbMeLa as shown in Fig. 5. If all philosophers operate on the basis of this protocol, then deadlock freedom is ensured on all paths. Starvation freedom holds almost surely under all schedulers, i.e., no matter how the steps of the philosophers are interleaved, with probability 1 each philosopher gets to eat eventually, if he intends to do so. Other classes of randomized algorithms, protocols, and scenarios are modeled likewise.

3 Maximal Reachability Probabilities

Computing maximal or minimal probabilities for reachability objectives is one of the core problems for a quantitative analysis of MDPs. In this section, we summarize the main concepts that rely on linear programming or an iterative approach which is known under the key word *value iteration*. Further details and other methods (e.g., policy iteration) can be found in any textbook on Markov decision processes; see, e.g., [80].

The notion "reachability" refers to the directed graph structure that is obtained from $\mathcal{M}$ by ignoring the labels of transitions and states. That is, a set B of states is said to be reachable from a state s if there exists a finite path in $\mathcal{M}$ that starts in state s and ends in some state $s' \in B$. In what follows, we will use the LTL-like notation $\Diamond B$ to denote the property "eventually reach B" where B is a set of states. That is, given a set $B \subseteq S$ of states, a path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots$ satisfies $\Diamond B$ if and only if there exists an index $i \in \mathbb{N}_{\geq 0}$ such that $s_i \in B$. The quantitative analysis of an MDP $\mathcal{M}$ against a reachability specification amounts to establishing the best upper and/or lower probability bounds that can be guaranteed to reach a given set B of states, when ranging over all schedulers. That is, the goal is to compute

$$\sup_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond B) \quad \text{and} \quad \inf_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond B),$$

where the supremum and the infimum are taken over all schedulers $\mathcal{U}$ for $\mathcal{M}$. If $\mathcal{M}$ is clear from the context, we will omit the system $\mathcal{M}$ in the superscript of $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}$.

For the rest of this chapter, we will restrict to the computation of the supremum because the results for the infimum are analogous. It is well known [80, 15] that the history of a scheduler is of no relevance when it comes to maximizing the probability of reaching a certain set of states and also that randomization in the schedulers is not needed. Thus, the supremum is attained by some memoryless deterministic scheduler (note that there are only finitely many such schedulers). Thus,

$$\sup_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}^{\mathcal{U}}(\Diamond B) = \sup_{\mathcal{U} \in \mathsf{Sched}_{\mathsf{M}}} \mathsf{Pr}^{\mathcal{U}}(\Diamond B) = \max_{\mathcal{U} \in \mathsf{Sched}_{\mathsf{MD}}} \mathsf{Pr}^{\mathcal{U}}(\Diamond B).$$

Note that the above chain of equality does not hold for arbitrary measurable path properties.

The standard method to compute these maxima is to compute $\mathsf{Pr}^{\mathcal{U}}_s(\Diamond B)$ for each state s in $\mathcal{M}$ via a recursive equation system. Formally, let an MDP $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ and a set $B \subseteq S$ of target states be given. The obligation is to compute

$$\mathsf{Pr}^{\max}_s(\Diamond B) \overset{\text{def}}{=} \max_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}^{\mathcal{U}}_s(\Diamond B) = \max_{\mathcal{U} \in \mathsf{Sched}_{\mathsf{MD}}} \mathsf{Pr}^{\mathcal{U}}_s(\Diamond B)$$

for each state $s \in S$. Let x_s denote this maximum for $s \in S$, that is,

$$x_s \overset{\text{def}}{=} \max_{\mathcal{U} \in \mathsf{Sched}_{\mathsf{MD}}} \mathsf{Pr}_s^{\mathcal{U}}(\Diamond B).$$

Then if $s \notin B$, it evidently holds that

$$x_s = \max\left\{ \sum_{t \in S} \delta(s,\alpha,t) \cdot x_t \;\middle|\; \alpha \in \mathsf{Act}(s) \right\}.$$

Moreover, if $s \in B$, then obviously $x_s = 1$, and if there is no path from s to B, then $x_s = 0$. The following theorem [22, 80, 15] states that these characteristics are sufficient to specify the maximum values.

Theorem 3.1 (Equation system for maximal reachability probabilities). *Let $\mathcal{M}$ be an MDP with state space S and $B \subseteq S$. The vector $(x_s)_{s \in S}$ with $x_s = \mathsf{Pr}_s^{\max}(\Diamond B)$ is the unique solution of the following equation system:*

- *If $s \in B$, then $x_s = 1$.*
- *If B is not reachable from s, then $x_s = 0$.*
- *If $s \notin B$ and B is reachable from s, then*

$$x_s = \max\left\{ \sum_{t \in S} \delta(s,\alpha,t) \cdot x_t \;\middle|\; \alpha \in \mathsf{Act}(s) \right\}.$$

Obviously, $x_s = \mathsf{Pr}_s^{\max}(\Diamond B)$ is a solution of the above equation system. The proof of its uniqueness is rather technical and omitted here.

Let us again consider the MDP of the Monty Hall problem depicted in Fig. 3. So, we are interested in the set of target states $B = \{u_1\}$. Then

$$\begin{array}{lll} x_{u_1} = 1 & x_{t_1} = \max\{x_{u_1}, x_{u_2}\} & x_s = \max\{\frac{1}{3} \cdot x_{t_1} + \frac{1}{3} \cdot x_{t_2} + \frac{1}{3} \cdot x_{t_3}\} \\ x_{u_2} = 0 & x_{t_2} = \max\{x_{u_1}, x_{u_2}\} & \\ & x_{t_3} = \max\{x_{u_1}, x_{u_2}\} & \end{array}$$

and the unique solution is $x_{u_1} = x_{t_1} = x_{t_2} = x_{t_3} = x_s = 1$ and $x_{u_2} = 0$.

To actually compute the values $\mathsf{Pr}_s^{\max}(\Diamond B)$ algorithmically, one can rewrite the equation system in Theorem 3.1 into the following linear program [22]:

- If $s \in B$, then $x_s = 1$.
- If B is not reachable from s, then $x_s = 0$.
- If $s \notin B$ and B is reachable from s, then $0 \leq x_s \leq 1$ and for each action $\alpha \in \mathsf{Act}(s)$:

$$x_s \geq \sum_{t \in S} \delta(s,\alpha,t) \cdot x_t.$$

With the objective to

$$\text{minimize} \sum_{s \in S} x_s,$$

the vector $(x_s)_{s \in S}$ with $x_s = \mathsf{Pr}_s^{\max}(\Diamond B)$ is the unique solution of this linear program. Identifying the states s such that the value of x_s is not fixed to 0 or 1 by the first two items of the linear program as $S_? = \{ s \in S \setminus B \mid B$ is reachable from $s \}$, one can rewrite the third item into

$$\big(1 - \delta(s,\alpha,s)\big) \cdot x_s - \sum_{t \in S_? \setminus \{ s \}} \delta(s,\alpha,t) \cdot x_t \geq \delta(s,\alpha,B)$$

where $\delta(s,\alpha,B) = \sum_{t \in B} \delta(s,\alpha,t)$. Thus, the third item in the above theorem can be read as a linear inequality $\mathbf{A} \cdot \mathbf{x} \geq \mathbf{b}$ where $\mathbf{x}$ is the vector $(x_s)_{s \in S_?}$ and $\mathbf{A}$ is a matrix with a row for each pair (s,α) with $s \in S_?$ and $\alpha \in \mathsf{Act}(s)$, two extra rows for each state $s \in S_?$ to represent the inequality $0 \leq x_s \leq 1$ and a column for each state $s \in S_?$. The precise values for $\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B)$ can thus be computed by standard algorithms to solve linear programs, e.g., the simplex algorithm or polytime methods [83].

Corollary 3.2 (Complexity of computing maximal reachability probabilities). *For an MDP $\mathcal{M}$ with state space S, $B \subseteq S$ and $s \in S$, the values $\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B)$ can be computed in time polynomial in the size of $\mathcal{M}$.*

This result, however, is more of theoretical interest than of practical relevance. In practice, one often uses a different approach to calculate the values $\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B)$, namely an iterative approximation technique called *value iteration* (see, e.g., [80]) which is based on the following fact. The second item in Theorem 3.1 could be omitted and replaced by the requirement that the equations for x_s in the third item holds for every state $s \in S \setminus B$. However, the uniqueness of the solution vector $(x_s)_{s \in S} = (\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B))_{s \in S}$ is then no longer guaranteed, but one can show that $(x_s)_{s \in S}$ is the least solution in $[0,1]^S$. For the value iteration, one fixes the value for $s \in B$ to $x_s = 1$ and starts with an initial value of $x_s = 0$ for all states $s \notin B$. One then iteratively recalculates the value according to item 3 of Theorem 3.1. That is,

$$\begin{aligned}
x_s^i &= 1 && \text{for } s \in B \text{ and } i \in \mathbb{N}_{\geq 0} \\
x_s^0 &= 0 && \text{for } s \notin B \\
x_s^{n+1} &= \mathsf{max}\Big\{ \sum_{t \in S} \delta(s,\alpha,t) \cdot x_t^n \;\Big|\; \alpha \in \mathsf{Act}(s) \Big\} && \text{for } s \notin B.
\end{aligned}$$

For the states $s \notin B$, it can be shown that

$$\lim_{n \to \infty} x_s^n = \mathsf{Pr}^{\mathsf{max}}_s(\lozenge B).$$

Note that $x_s^0 \leq x_s^1 \leq x_s^2 \leq \cdots$. Thus, the values $\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B)$ can be approximated by successively computing the vectors

$$\big(x_s^0\big)_{s \in S}, \big(x_s^1\big)_{s \in S}, \big(x_s^2\big)_{s \in S}, \ldots ,$$

until $\mathsf{max}_{s \in S} |x_s^{n+1} - x_s^n|$ is below a termination threshold.

Let us once more consider the MDP of the Monty Hall problem depicted in Fig. 3 and the target set $B = \{u_1\}$. Thus, $x^i_{u_1} = 1$ for every $i \in \mathbb{N}_{\geq 0}$. Note that $x^i_{u_2} = x^{i-1}_{u_2}$, so $x^i_{u_2}$ will equal 0 for every $i \in \mathbb{N}_{\geq 0}$. With $x^i_{t_1} = x^i_{t_2} = x^i_{t_3} = \mathsf{max}\{1 \cdot x^{i-1}_{u_1}, 1 \cdot x^{i-1}_{u_2}\}$ we get

$$\begin{array}{lll}
x^0_{u_2} = 0 & x^0_{t_1} = x^0_{t_2} = x^0_{t_3} = 0 & x^0_s = 0 \\
x^1_{u_2} = 0 & x^1_{t_1} = x^1_{t_2} = x^1_{t_3} = 1 & x^1_s = \mathsf{max}\{\frac{1}{3} \cdot x^0_{t_1} + \frac{1}{3} \cdot x^0_{t_2} + \frac{1}{3} \cdot x^0_{t_3}\} = 0 \\
x^2_{u_2} = 0 & x^2_{t_1} = x^2_{t_2} = x^2_{t_3} = 1 & x^2_s = \mathsf{max}\{\frac{1}{3} \cdot x^1_{t_1} + \frac{1}{3} \cdot x^1_{t_2} + \frac{1}{3} \cdot x^1_{t_3}\} = 1 \\
x^3_{u_2} = 0 & x^3_{t_1} = x^3_{t_2} = x^3_{t_3} = 1 & x^3_s = \mathsf{max}\{\frac{1}{3} \cdot x^2_{t_1} + \frac{1}{3} \cdot x^2_{t_2} + \frac{1}{3} \cdot x^2_{t_3}\} = 1.
\end{array}$$

As all values did not change in the last iteration, we can conclude that the fixed point is reached.

Implementation Issues

It is obvious that for the set of states from which B is not reachable, the value iteration is not needed as $\mathsf{Pr}^{\mathsf{max}}_s(\Diamond B) = 0$ if and only if B is not reachable from s. Thus, it is advisable to first compute the set

$$S_0 \overset{\text{def}}{=} \{s \in S \mid B \text{ is not reachable from } s\}$$

with a standard backward reachability analysis and then set $x^i_s = 0$ for all states $s \in S_0$ and $i \in \mathbb{N}_{\geq 0}$. Note that this reduces the number of variables for which the value iteration has to be performed. Similarly, one can first compute the set

$$S_1 \overset{\text{def}}{=} \left\{s \in S \;\middle|\; \mathsf{Pr}^{\mathsf{max}}_s(\Diamond B) = 1\right\}$$

of states s such that $\mathsf{Pr}^{\mathsf{max}}_s(\Diamond B) = 1$. For all states $s \in S_1$, we set $x^i_s = 1$ for all $i \in \mathbb{N}_{\geq 0}$ thus reducing again the number of variables for which the value iteration has to be performed. The computation of S_1 can be done efficiently by Algorithm 1 using a nested fixpoint computation.

In the formal description of the value iteration, each variable x_s with $s \notin B$ is updated in every iteration. This might of course lead to a great amount of unnecessary updates (consider for instance a unidirectional, very long chain where the last state forms the target set B). So, when implementing the value iteration one seeks to omit updating a value x_s if there is no successor t of s such that the value x_t has been changed during the last update of x_s. This idea is reflected in Algorithm 2 which iteratively propagates probabilities by means of a backward traversal of the MDP from B. The algorithm maintains a set T of states for which the x_t value has been changed. It then successively removes a state t from the set T and updates the value x_s for every state s that has t as a successor. If such a value x_s becomes altered, the state s is added to T.

There are several variants of Algorithm 2 that differ in the data structure used for the organization of the set T. For instance, the set T can be realized as a stack, a (priority) queue or using buckets to aggregate states. For the priority queue and the bucket structure, the sorting criterion is the value $\Delta(s)$. For more information on such implementation details and other heuristics, see [5].

Algorithm 1 Computation of S_1

```
$R := S$
$done := false$
while $done = false$ do
  $R' := B$
  $done' := false$
  while $done' = false$ do
    $R'' := R' \cup \{s \mid \exists \alpha \in \mathsf{Act}(s) : \mathsf{supp}(\delta(s,\alpha,.)) \subseteq R \wedge \mathsf{supp}(\delta(s,\alpha,.)) \cap R' \neq \emptyset\}$
    if $R'' = R'$ then
      $done' := true$
    end if
    $R' := R''$
  end while
  if $R' = R$ then
    $done := true$
  end if
  $R := R'$
end while
return $R$
```

Algorithm 2 Backward value iteration

```
compute $S_0$, $S_1$ and $S_? = S \setminus (S_0 \cup S_1)$
set $x_s := 1$ for all states $s \in S_1$ and $x_s := 0$ for all states $s \in S_0$
for all states $s \in S_?$ do
  $x_s := \mathsf{max}\{\sum_{t \in S_1} \delta(s,\alpha,t) \mid \alpha \in \mathsf{Act}(s)\}$
end for
$T := \{s \in S_? \mid x_s > 0\}$
repeat
  if $T \neq \emptyset$ then
    pick some state $t \in T$ and remove $t$ from $T$
    for all states $s$ such that $\exists$ action $\alpha \in \mathsf{Act}(s)$ with $\delta(s,\alpha,t) > 0$ do
      $x'_s := \mathsf{max}\{\sum_{u \notin S_0} \delta(s,\alpha,u) \cdot x_u \mid \alpha \in \mathsf{Act}(s)\}$
      if $x'_s > x_s$ then
        $\Delta(s) := x'_s - x_s$ and $x_s := x'_s$ and add $s$ to $T$
      end if
    end for
  end if
until $T = \emptyset$ or termination threshold is matched
```

It should be noticed that if one is just interested in a qualitative reachability analysis (where the task is to check whether $\mathsf{Pr}^{\mathsf{max}}_s(\lozenge B)$ is 0 or 1), then the computation of the values $x_s = \mathsf{Pr}^{\mathsf{max}}(\lozenge B)$ is not necessary. In fact, algorithms to compute the sets S_0 and S_1 are sufficient.

4 Model Checking ω-Regular Properties

In the previous section, we have addressed the problem of computing extremal reachability probabilities in an MDP and have seen that this can be represented as a recursive equation system or a linear program which can be solved through fixpoint calculations or, e.g., the simplex algorithm. This approach can easily be extended to constrained reachability properties (that require to reach a certain target set B along finite paths that stay inside another set C) or the general class of regular safety properties (which impose conditions on the finite paths by means of a finite automaton). The quantitative analysis against reachability probabilities is also the core problem for quantitative reasoning about liveness properties or even arbitrary ω-regular linear-time properties. In this section, we explain how extremal probabilities for ω-regular properties represented by ω-automata can be calculated using a graph-based algorithm and a reduction to the problem of computing maximal reachability probabilities.

Before we proceed, we fix some notation on finite, resp. infinite words over a given alphabet. Following the standard notation, given an alphabet Σ, we write Σ^+ for the set of all non-empty finite words over Σ, Σ^* for the set of all finite words over Σ (including the empty word ε) and Σ^ω for the set of all infinite words over Σ. Given a finite non-empty word $\varsigma = \sigma_1\sigma_2\ldots\sigma_n$, the length $|\varsigma|$ of ς equals n. For an infinite word ς, the length is equal to ∞. Given a (non-empty finite or infinite) word $\varsigma = \sigma_1\sigma_2\sigma_3\ldots \in \Sigma^+ \cup \Sigma^\omega$ and $i \leq |\varsigma|$, we denote the ith letter of ς by ς_i (i.e., $\varsigma_i = \sigma_i$).

Recall from Definition 2.1 that in our approach each state of an MDP is labeled with a subset of a set AP of atomic propositions. Thus, each infinite path of such an MDP produces a trace which is an infinite word over the alphabet 2^{AP}.

Definition 4.1 (Trace of a path). *Given an MDP and an infinite path* $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$, *we define the infinite word*

$$\mathsf{trace}(\pi) \overset{\text{def}}{=} L(s_0)L(s_1)L(s_2)\ldots \in \left(2^{\mathsf{AP}}\right)^\omega$$

to be the trace of π.

Definition 4.2 (Linear-time property). *A linear-time property (LT property) over a given finite set* AP *of atomic propositions is a subset of* $(2^{\mathsf{AP}})^\omega$.

Given a state-labeled system, we say that a path π *satisfies* a given LT property E, if and only if $\mathsf{trace}(\pi) \in \mathsf{E}$. LT properties are used to specify the infinite behavior of a given system. For instance, the reachability property $\lozenge\,\mathsf{get\_car}$ is formally defined as the language

$$\mathsf{E} = \left\{\varsigma \in \left(2^{\mathsf{AP}}\right)^\omega \;\middle|\; \exists i : \mathsf{get\_car} \in \varsigma_i\right\}.$$

There are various formalisms (logics, automata, algebraic expressions) to specify LT properties. A very prominent logical formalism in model checking is the linear temporal logic (LTL) [78]. In this chapter, however, we will utilize ω-automata to specify LT properties. While finite automata serve as acceptors for languages of finite words, the semantics of an ω-automaton refers to a language over infinite words. The syntax and operational semantics of ω-automata is roughly the same as for finite automata, except that the input of an ω-automata is an infinite word and the acceptance condition imposes constraints on infinite runs rather than on finite ones. As we assume that the reader of this handbook is already familiar with the concept of ω-automata, we just provide the definitions that are relevant for our purposes. For more information on ω-automata, we refer to, e.g., [85, 40].

Definition 4.3 (Deterministic ω-automaton). *We define a deterministic ω-automaton as a tuple*

$$\mathcal{A} = (Q, \Sigma, \delta_{\mathcal{A}}, q_0, \mathsf{Acc}),$$

where:

- Q *is a finite non-empty set of states.*
- Σ *is a finite non-empty input alphabet.*
- $\delta_{\mathcal{A}} : Q \times \Sigma \to Q$ *is a transition function.*
- $q_0 \in Q$ *is the initial state.*
- Acc *is an acceptance condition.*

In this chapter, we only consider acceptance conditions of the form

$$\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$$

and interpret them with a Rabin, resp. Streett semantics. Given a subset $T \subseteq Q$ of states, we call T:

- *Rabin-accepting*, if there exists $1 \leq i \leq n$ such that
$$T \cap H_i = \emptyset \quad \text{and} \quad T \cap K_i \neq \emptyset.$$
- *Streett-accepting*, if for every $1 \leq i \leq n$
$$T \cap H_i \neq \emptyset \quad \text{or} \quad T \cap K_i = \emptyset.$$

Thus, Rabin and Streett acceptance are complementary to each other.

Given a deterministic ω-automaton $\mathcal{A}$ with an acceptance condition $\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$ and an infinite word $\varsigma = \sigma_1\sigma_2\sigma_3\ldots$ over Σ, we define the *run* for ς in $\mathcal{A}$ to be the infinite state sequence $\rho = p_0, p_1, \ldots$ such that $p_0 = q_0$ and $p_i = \delta_{\mathcal{A}}(p_{i-1}, \sigma_i)$ for every $i \geq 1$. Given a run ρ, let

$$\mathsf{inf}(\rho) = \left\{ p \in Q \;\middle|\; \overset{\infty}{\exists} i \in \mathbb{N}_{\geq 0} : p_i = p \right\}$$

denote the set of states that occur infinitely often in ρ. We call a run ρ *Rabin-, resp. Streett-accepting*, if and only if $\mathsf{inf}(\rho)$ is Rabin-, resp. Streett-accepting. The *Z-accepted language* of $\mathcal{A}$ is defined as

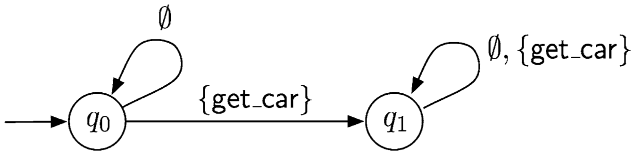

$$\begin{tikzcd} \arrow[r] & q_0 \arrow[loop above, "\emptyset"] \arrow[r, "\{\mathsf{get\_car}\}"] & q_1 \arrow[loop right, "{\emptyset, \{\mathsf{get\_car}\}}"] \end{tikzcd}$$

Fig. 6. DSA for $\Diamond$ $\mathsf{get\_car}$

$$\mathcal{L}_Z(\mathcal{A}) \overset{\text{def}}{=} \{\varsigma \in \Sigma^\omega \mid \text{the run for } \varsigma \text{ in } \mathcal{A} \text{ is Z-accepting}\}$$

for $Z \in \{Rabin, Streett\}$.

In order to argue more easily about the accepted language, we call the combination of a deterministic ω-automaton and Rabin acceptance (resp., Streett acceptance) a *deterministic Rabin automaton* (DRA) (resp., *deterministic Streett automaton* (DSA)). In the remainder of this section, we will drop the Z from Z-accepting, Z-accepted language, and $\mathcal{L}_Z(\mathcal{A})$, if it is clear from the context.

To use deterministic Rabin, resp. Streett automata as a formalism for representing LT properties, we will consider automata with the alphabet $\Sigma = 2^{\mathsf{AP}}$, where AP is the set of atomic propositions of the given system. For example, the reachability property

$$\mathsf{E} = \Diamond\ \mathsf{get\_car} = \left\{\varsigma \in \left(2^{\mathsf{AP}}\right)^\omega \mid \exists i : \mathsf{get\_car} \in \varsigma_i\right\}$$

over the set $\mathsf{AP} = \{\mathsf{get\_car}\}$ can be represented by the deterministic Streett automaton $\mathcal{A}$ shown in Fig. 6 where the acceptance condition is $\mathsf{Acc} = \{(\{q_1\}, \{q_0, q_1\})\}$. It is easy to see that $\mathcal{L}(\mathcal{A}) = \mathsf{E}$.

It is well known [85, 40] that the class of languages that are definable by deterministic Rabin (or Streett) automata coincides with the class of ω-regular languages and the class of MSO definable languages over infinite words. Thus, DRA and DSA are powerful enough to express many interesting specifications that arise in real-world scenarios. This includes simple temporal properties like "eventually" (reachability properties), "always" (safety properties), and liveness properties that result by combination of "eventually" and "always", such as "infinitely often" or "continuously from some moment on". But also more complex properties such as "each process will eventually enter its critical section" or "between two eating phases of a dining philosopher there is always a thinking phase" can be specified by DRA and DSA.

Natural requirements for MDPs and other types of probabilistic systems attach lower or upper probability bounds on such LT properties. That is, the typical task is to verify conditions such as "the probability that a waiting process is never allowed to enter its critical section is less than 0.005" for a randomized mutual exclusion protocol. But also qualitative properties play a crucial role where the goal is to establish conditions such as "with probability 1 the repeated attempt to deliver a message will eventually be successful" for a communication protocol or "with probability 1 a leader will eventually be elected" for a randomized leader election protocol.

In the remainder of this section, we explain the main steps for a quantitative analysis of an MDP $\mathcal{M}$ against an ω-regular LT property, specified by a DRA or DSA $\mathcal{A}$ (note that such properties are measurable). This requires us to compute the maximal or minimal probabilities

$$\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A}) \overset{\text{def}}{=} \sup_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}_s^{\mathcal{U}}\big(\mathcal{L}(\mathcal{A})\big) \quad \text{and} \quad \mathsf{Pr}_s^{\mathsf{min}}(\mathcal{A}) \overset{\text{def}}{=} \inf_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}_s^{\mathcal{U}}\big(\mathcal{L}(\mathcal{A})\big)$$

for the paths π of $\mathcal{M}$ that start in s and where π satisfies $\mathcal{L}(\mathcal{A})$, i.e., $\mathsf{trace}(\pi)$ belongs to $\mathcal{L}(\mathcal{A})$. As for the quantitative reachability analysis, the supremum and infimum are taken over all schedulers of the given MDP. In fact, there exist schedulers that maximize or minimize the probabilities for paths with a trace in $\mathcal{L}(\mathcal{A})$. Again, the supremum and infimum can be replaced with maximum and minimum. As in Sect. 3, we will focus here on explanations about the computation of maximal probabilities, i.e., the computation of the values $\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A})$. Analogous techniques are applicable to compute $\mathsf{Pr}_s^{\mathsf{min}}(\mathcal{A})$. However, since the class of ω-regular languages is closed under complementation, algorithms to compute $\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A})$ are even sufficient to reason about minimal probabilities. Given a deterministic ω-automaton $\mathcal{A}$ that specifies the desired behaviors, we may switch to a deterministic ω-automaton $\mathcal{B}$ for the complement language $\overline{\mathcal{L}(\mathcal{A})}$, i.e., $\mathcal{B}$ represents the undesired behaviors. (Here, we can exploit the duality of Rabin and Streett acceptance. That is, if $\mathcal{A}$ is a DSA, then we can use $\mathcal{B} = \mathcal{A}$, but treat $\mathcal{B}$ as a DRA, and vice versa.) We then apply the techniques for computing the maximal probabilities $\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{B})$ for the "bad" event specified by $\mathcal{B}$. The greatest lower bound for the probabilities that can be guaranteed for the good behaviors is then obtained by the following equation:

$$\mathsf{Pr}_s^{\mathsf{min}}(\mathcal{A}) = 1 - \mathsf{Pr}_s^{\mathsf{max}}(\mathcal{B}).$$

The key for the quantitative analysis of MDPs against ω-regular properties lies in the concept of de Alfaro's end components [27, 28]. They can be seen as the MDP counterpart to terminal strongly connected components in Markov chains. Intuitively, an end component of an MDP is a non-empty strongly connected sub-MDP, that means an end component consists of a non-empty state set $T \subseteq S$ and a non-empty action set $A(t)$ for each state $t \in T$ such that, once T is entered and only actions in $A(t)$ are chosen, the set T will not be left and any state of T can be reached from any other state in T.

Definition 4.4 (End components). *Let $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ be an MDP. Then an end component of $\mathcal{M}$ is a pair (T, A) where $\emptyset \neq T \subseteq S$ and $A : T \to 2^{\mathsf{Act}}$ is a function such that the following three conditions are satisfied:*

- *$\emptyset \neq A(s) \subseteq \mathsf{Act}(s)$ for each state $s \in T$.*
- *If $s \in T$, $t \in S$ and $\alpha \in A(s)$ such that $\delta(s, \alpha, t) > 0$ then $t \in T$.*

- *The underlying directed graph $(T, \rightarrow_A)$ of (T, A) is strongly connected.*

Here, $\rightarrow_A$ denotes the edge-relation induced by A, that is $s \rightarrow_A t$ if and only if $\delta(s, \alpha, t) > 0$ for some action $\alpha \in A(s)$.

The importance of end components is due to the following observation about the limit behavior of paths that has been established by de Alfaro [27, 28]. Given an infinite path $\pi = s_0 \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} s_2 \xrightarrow{\alpha_3} \cdots$ we denote by $\mathsf{Lim}(\pi)$ the pair (T, A) where $T = \mathsf{inf}(\pi)$ is the set of states in π that are visited infinitely often and $A : T \to 2^{\mathsf{Act}}$ is the function that assigns to any state $t \in T$ the set of actions $\alpha \in \mathsf{Act}$ such that $(s_i = t) \wedge (\alpha_{i+1} = \alpha)$ for infinitely many indices i. Given an MDP $\mathcal{M}$ and a (possibly history-dependent and randomized) scheduler $\mathcal{U}$, it holds that in the process induced by $\mathcal{U}$, almost all paths of $\mathcal{M}$ "end" in an end component, that is, their limit $\mathsf{Lim}(.)$ forms an end component.

Lemma 4.5 (Limiting behavior of MDPs and end components). *For any MDP $\mathcal{M}$ and scheduler $\mathcal{U}$,*

$$\mathsf{Pr}^{\mathcal{M},\mathcal{U}}\big(\{\pi \in \mathsf{Path}^{\mathcal{M}}_{\mathsf{inf}} \mid \mathsf{Lim}(\pi) \textit{ is an end component}\}\big) = 1.$$

This fundamental property of MDPs is one of the main features for computing $\mathsf{Pr}^{\mathsf{max}}_s(\mathcal{A})$ for a given DRA, resp. DSA $\mathcal{A}$ via a reduction to the problem of maximal reachability probabilities. Another feature is the product construction of the MDP and the automaton $\mathcal{A}$.

Definition 4.6 (Product-MDP). *As before, let $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ be an MDP and let $\mathcal{A} = (Q, 2^{\mathsf{AP}}, \delta_{\mathcal{A}}, q_0, \mathsf{Acc})$ be a DSA or DRA. The product of $\mathcal{M}$ and $\mathcal{A}$ is defined as the MDP*

$$\mathcal{M} \otimes \mathcal{A} = (S \times Q, \mathsf{Act}', \delta', \mu', Q, L')$$

where the transition function δ', the initial distribution μ', and the labeling function L' are defined as follows:

- $\mathsf{Act}' \stackrel{\text{def}}{=} \mathsf{Act}$
- $\delta'(\langle s, q\rangle, \alpha, \langle s', q'\rangle) \stackrel{\text{def}}{=} \begin{cases} \delta(s, \alpha, s') & \textit{if } q' = \delta_{\mathcal{A}}(q, L(s')) \\ 0 & \textit{otherwise} \end{cases}$
- $\mu'(\langle s, q\rangle) \stackrel{\text{def}}{=} \begin{cases} \mu(s) & \textit{if } q = \delta_{\mathcal{A}}(q_0, L(s)) \\ 0 & \textit{otherwise} \end{cases}$
- $L'(\langle s, q\rangle) \stackrel{\text{def}}{=} \{\, q \,\}$

Note that this construction requires a deterministic ω-automaton $\mathcal{A}$, as for a non-deterministic ω-automaton there is no straightforward way to define appropriate transition probabilities for the product. We may observe a one-to-one correspondence between the path

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

in the MDP $\mathcal{M}$ and the path

$$\pi^+ = \langle s_0, q_1 \rangle \xrightarrow{\alpha_1} \langle s_1, q_2 \rangle \xrightarrow{\alpha_2} \langle s_2, q_3 \rangle \xrightarrow{\alpha_3} \cdots$$

in $\mathcal{M} \otimes \mathcal{A}$ that starts in state $\langle s_0, q_1 \rangle$ where $q_1 = \delta_{\mathcal{A}}(q_0, L(s_0))$. Given a path π^+ in $\mathcal{M} \otimes \mathcal{A}$, the corresponding path in $\mathcal{M}$ is simply obtained by omitting all automata states q_i. Vice versa, given a path π as above, the corresponding path π^+ is obtained by adding the automaton states $q_{i+1} = \delta_{\mathcal{A}}(q_i, L(s_i))$ to π. Thus, π^+ emanates from π and the unique run for $\mathsf{trace}(\pi)$ in $\mathcal{A}$. Recall that the acceptance of a run imposes a condition on the set of states in the automaton that appear infinitely often in that run. Hence, whether or not $\mathsf{trace}(\pi)$ belongs to $\mathcal{L}(\mathcal{A})$ depends on the projection of $\mathsf{inf}(\pi^+)$ to the states in $\mathcal{A}$, i.e., the set $\{q \in Q \mid \exists s \in S : \langle s, q \rangle \in \mathsf{inf}(\pi^+)\}$. Since almost all paths in $\mathcal{M} \otimes \mathcal{A}$ constitute an end component (by Lemma 4.5), the algorithm to compute $\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A})$ relies on an analysis of the end components of the product where the acceptance condition of $\mathcal{A}$ holds.

Definition 4.7 (Accepting end components). *Given an MDP* $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ *and a DRA, resp. a DSA* $\mathcal{A} = (Q, 2^{\mathsf{AP}}, \delta_{\mathcal{A}}, q_0, \mathsf{Acc})$*, we call an end component* (T, A) *of* $\mathcal{M} \otimes \mathcal{A}$ *accepting if and only if the set*

$$T|_Q \overset{\text{def}}{=} \{q \in Q \mid \exists s \in S : \langle s, q \rangle \in T\}$$

is accepting in $\mathcal{A}$.

In the sequel, let AEC be the union of (the state-component of) all accepting end components in the product-MDP $\mathcal{M} \otimes \mathcal{A}$. That is, AEC consists of all states $\langle s, q \rangle \in S \times Q$ such that $\langle s, q \rangle \in T$ for at least one accepting end component (T, A) of $\mathcal{M} \otimes \mathcal{A}$. The probability for the paths in $\mathcal{M}$ under some scheduler $\mathcal{U}$ that have their trace in $\mathcal{L}(\mathcal{A})$ agrees with the probability for the paths in $\mathcal{M} \otimes \mathcal{A}$ whose limits yield an accepting end component under the corresponding scheduler $\mathcal{U}^+$ for the product. We use here the fact that each scheduler $\mathcal{U}$ for $\mathcal{M}$ can be mimicked by a scheduler $\mathcal{U}^+$ for $\mathcal{M} \otimes \mathcal{A}$. Formally, $\mathcal{U}^+$'s decision for a finite path π^+ agrees with $\mathcal{U}$'s decision for the path that results from π^+ by dropping the automaton component from the states. The maximal probabilities in $\mathcal{M} \otimes \mathcal{A}$ for paths π^+ where $\mathsf{Lim}(\pi^+)$ is an accepting end component agree with the maximal probability in $\mathcal{M} \otimes \mathcal{A}$ for reaching eventually the set AEC. Note that there is a scheduler that ensures that once AEC has been reached, the set AEC will never be left and almost surely the limiting behavior will constitute an accepting end component. This is the key property to provide a formal proof for the following theorem [27, 12].

Theorem 4.8 (Maximal probabilities for ω-regular properties). *Let* $\mathcal{M}$*,* $\mathcal{A}$*, and* AEC *be as above. Then for each state* $s \in S$*:*

$$\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A}) = \mathsf{Pr}^{\mathsf{max}}_{\langle s, \delta_{\mathcal{A}}(q_0, L(s)) \rangle}(\Diamond\, AEC).$$

On the basis of this theorem, an algorithm to compute $\mathsf{Pr}_s^{\mathsf{max}}(\mathcal{A})$ can proceed as follows. First, construct the product-MDP $\mathcal{M} \otimes \mathcal{A}$, then compute the accepting end components and finally apply the techniques explained in Sect. 4 to compute the maximal reachability probabilities for the target set AEC in the product. What remains is to explain the computation of the set AEC. The notion of an accepting end component just depends on the topological graph structure of the MDP $\mathcal{M} \otimes \mathcal{A}$ (the precise transition probabilities are irrelevant). Therefore, purely graph-based methods are sufficient to compute the set AEC. As the number of end components of an MDP $\mathcal{M}$ can grow exponentially in the size of $\mathcal{M}$, the naive approach that relies on an explicit computation of all accepting end components is hopelessly inefficient. Instead, one can use the concept of maximal end components to increase the efficiency. An end component (T, A) of an MDP $\mathcal{M}'$ is called *maximal* if there is no end component (T', A') of $\mathcal{M}'$ such that (i) $T \subseteq T'$, (ii) $A(t) \subseteq A'(t)$ for every $t \in T$, and (iii) $(T, A) \neq (T', A')$. Obviously, the state sets of maximal end components are pairwise disjoint and each end component is contained in some maximal end component. In particular, the total number of maximal end components is bounded by the number of states in $\mathcal{M}'$. We will explain first how the set of all maximal end components, which is denoted by MEC, can be computed in polynomial time.

Computing the Set of Maximal end Components

Here, we address the problem of computing the set MEC of all maximal end components in the product-MDP $\mathcal{M} \otimes \mathcal{A}$. Recall that the states contained in an end component (T, A) are strongly connected in the underlying directed graph $(T, \rightarrow_A)$ (see Definition 4.4). Furthermore, it is clear that a maximal end component in $\mathcal{M} \otimes \mathcal{A}$ is contained in some strongly connected component (SCC) of $(S \times Q, \rightarrow_{\mathcal{M} \otimes \mathcal{A}})$, the underlying directed graph of $\mathcal{M} \otimes \mathcal{A}$. Thus, an algorithmic scheme for calculating the set MEC of all maximal end components consists of the following steps [23, 27]:

1. In a first step, we compute a candidate set $\mathcal{C}$ of all possible maximal end components as follows. Compute the SCCs and define
$$\mathcal{C} = \{(T, A) \mid T \text{ is an SCC and } A(t) = \mathsf{Act}(t) \text{ for } t \in T\}.$$
2. Pick and remove a candidate (T, A) from $\mathcal{C}$.
3. For each state $t \in T$, remove all actions from $A(t)$ that violate the second condition of Definition 4.4, i.e., remove all $\alpha \in A(t)$ with $\sum_{u \in T}(t, \alpha, u) < 1$ and call that modified candidate (T, A').
4. Calculate all SCCs in the underlying directed graph $(T, \rightarrow_{A'})$ of (T, A') and insert them as new candidates into $\mathcal{C}$ (similar as in step 1, but with action sets restricted to A').
5. Repeat steps 2–4 until $\mathcal{C}$ reaches a fixpoint.

When the fixpoint is reached, the set $\mathcal{C}$ equals the set MEC. Since during each iteration for every candidate $(T, A) \in \mathcal{C}$ either there exist actions α in $A(t)$ that are removed, and thus causing a potential splitting of this candidate, or a fixpoint is reached for $\mathcal{C}$; the procedure obviously terminates. Furthermore, the number of iterations is bounded by the total number of pairs (t, α) where t is a state in $\mathcal{M} \otimes \mathcal{A}$ and α an action that is enabled in t. The costs per iteration are dominated by the calculation of the SCCs in steps 1 and 4. Thus, the complexity of the algorithm is polynomially bounded in the size of $\mathcal{M} \otimes \mathcal{A}$ (i.e., the total number of states and transitions in $\mathcal{M} \otimes \mathcal{A}$).

Remark 4.9. The algorithm to compute the maximal end components can also be used to increase the efficiency of the algorithms for computing the extremal probabilities for a reachability property $\Diamond B$. This is due to the fact that whenever s and t are states that belong to the same maximal end component, then $\mathsf{Pr}_s^{\mathsf{max}}(\Diamond B) = \mathsf{Pr}_t^{\mathsf{max}}(\Diamond B)$. Hence, the given MDP can be simplified by collapsing all states that belong to the same maximal end component into a single state. This reduces the number of variables in the equation system or linear program and can therefore lead to a major speed-up [5].

With the above algorithm for the computation of MEC, we can now explain how the set AEC can be computed for the product-MDP of a given MDP $\mathcal{M}$ and a given DRA, resp. DSA $\mathcal{A}$.

Case 1: $\mathcal{A}$ Is a Deterministic Rabin Automaton

Let $\mathsf{Acc} = \{(H_1, K_1), \ldots, (H_n, K_n)\}$ be the acceptance condition of $\mathcal{A}$ and let

$$H_i' \overset{\text{def}}{=} \{\langle s, q\rangle \mid s \in S,\ q \in H_i\} \quad \text{and}$$
$$K_i' \overset{\text{def}}{=} \{\langle s, q\rangle \mid s \in S,\ q \in K_i\}.$$

Thus, an end component (T, A) is accepting, if there is an index $1 \leq i \leq n$ such that

$$T \cap H_i' = \emptyset \quad \text{and} \quad T \cap K_i' \neq \emptyset.$$

Assume that (T, A) is accepting with respect to (H_i', K_i'). Let $\mathcal{M}_i'$ be the MDP that results from $\mathcal{M} \otimes \mathcal{A}$ by removing all states in H_i'. Then (T, A) is obviously an end component in $\mathcal{M}_i'$, and moreover the maximal end component of $\mathcal{M}_i'$ that contains (T, A) is also accepting with respect to (H_i', K_i'). Hence, the set AEC arises as the union of the sets AEC_i for $1 \leq i \leq n$ where AEC_i is the union of (the state-components of) all maximal end components (T, A) in $\mathcal{M}_i'$ where $T \cap K_i' \neq \emptyset$.

Case 2: $\mathcal{A}$ Is a Deterministic Streett Automaton

In this case, an efficient way to realize the quantitative analysis on the basis of Theorem 4.8 is obtained using the following lemma.

Lemma 4.10. *Given an MDP $\mathcal{M}'$ with state space S', a target set of states $B \subseteq S'$, and a set X with $B \subseteq X \subseteq \{t \in S' \mid \mathsf{Pr}^{\max}_t(\Diamond B) = 1\}$, then for every state $s \in \mathcal{M}'$:*

$$\mathsf{Pr}^{\max}_s(\Diamond B) = \mathsf{Pr}^{\max}_s(\Diamond X).$$

Let $\mathcal{M}' = \mathcal{M} \otimes \mathcal{A}$. We denote by *AMEC* the union of all maximal end components that have an accepting sub-component, i.e.,

$$\begin{aligned} AMEC \overset{\text{def}}{=} \big\{t \in S \times Q \mid \text{ there exists } (T, A) \in \mathsf{MEC} \text{ such that } t \in T \text{ and} \\ \text{there is some accepting end component } (T', A') \text{ with } T' \subseteq T\big\}. \end{aligned}$$

It should be noticed that the correct term for *AMEC* is not "accepting maximal end components" but "maximal end components that contain at least one accepting sub-end component". We now show that with $B = AEC$ and $X = AMEC$ the condition of Lemma 4.10 holds, i.e.,

$$AEC \subseteq AMEC \subseteq \big\{t \bigm| \mathsf{Pr}^{\max}_t(\Diamond AEC) = 1\big\}.$$

This can be seen as follows. Let $t \in AMEC$ be included in the end component (T, A) that has the accepting sub-component (T', A'). As (T, A) is an end component, there is a scheduler $\mathcal{U}$ which ensures that starting in state t, almost surely each state in T will be visited infinitely often. Since $T' \subseteq T$, the set T' will be visited almost surely. As $T' \subseteq AEC$, we get that $\mathsf{Pr}^{\mathcal{U}}_t(\Diamond AEC) = 1$. Using Lemma 4.10, we thus can reformulate Theorem 4.8 as the following theorem.

Theorem 4.11 (Maximum probability for ω-regular properties, part II). *Let $\mathcal{M}$ be as before, $\mathcal{A}$ a DSA*[2]*, and AEC and AMEC be as above. Then for every state s of $\mathcal{M}$:*

$$\mathsf{Pr}^{\max}_s(\mathcal{A}) = \mathsf{Pr}^{\max}_{\langle s, \delta_{\mathcal{A}}(q_0, L(s))\rangle}(\Diamond AEC) = \mathsf{Pr}^{\max}_{\langle s, \delta_{\mathcal{A}}(q_0, L(s))\rangle}(\Diamond AMEC).$$

This observation allows us to switch from *AEC* to the larger set *AMEC* that arises by the union of certain maximal end components. In the following, we will describe how the set *AMEC* can be computed efficiently for a given MDP $\mathcal{M}$ and a given deterministic Streett automaton $\mathcal{A}$.

Calculating the Set AMEC

To compute the set *AMEC*, it remains to check for each maximal end component $(T, A) \in \mathsf{MEC}$ if it contains an accepting end component with respect to the given Streett acceptance condition Acc. In the sequel, let AMEC be the set of all maximal end components that contain an accepting end component. (Hence, *AMEC* is the set of all states that are contained in some $(T, A) \in \mathsf{AMEC}$.) For simplicity, we suppose that Acc consists of a single acceptance pair, say $\mathsf{Acc} = \{(H, K)\}$. Let

[2] Theorem 4.11 also holds if $\mathcal{A}$ is a DRA.

$$H' \stackrel{\text{def}}{=} \{\langle s,q\rangle \mid s \in S, q \in H\} \quad \text{and}$$
$$K' \stackrel{\text{def}}{=} \{\langle s,q\rangle \mid s \in S, q \in K\}.$$

Assume that a maximal end component (T,A) violates the given Streett acceptance condition. Then $T \cap H' = \emptyset$ and no sub-component can satisfy the acceptance condition by containing an H'-state. Hence, a sub-component can only satisfy the acceptance condition Acc if it does not contain a K'-state. These ideas lead to the following procedure. Consider a maximal end component $(T,A) \in$ MEC.

1. If $T \cap H' \neq \emptyset$ or $T \cap K' = \emptyset$, then $(T,A) \in$ AMEC.
2. Otherwise let $T_1 = T \setminus K'$ and A_1 such that $A_1(t) = \{\alpha \in A(t) \mid \sum_{t \in T_1}(s,\alpha,t) = 1\}$ for each state $t \in T_1$.
3. If an end component can be found in (T_1, A_1), then $(T,A) \in$ AMEC.
4. Otherwise, $(T,A) \notin$ AMEC.

5 Partial Order Reduction

In contrast to the previous sections, where advanced solution techniques for the value iteration have been discussed, we now focus on the state space explosion problem. There exist diverse methods for tackling the state space explosion problem for non-probabilistic as well as probabilistic systems. This includes symbolic model checking methods and various reduction techniques, see, e.g., [21] for an overview. The symbolic methods are mainly based on multi-terminal binary decision diagrams and focus on a compact internal representation of the (full) system [17, 45, 45, 6, 16, 72, 49, 69, 61]. So, these methods do not aim at avoiding the state space explosion, but at using a very compact representation of the given model. For instance, the PCTL-model checkers PRISM [60], ProbVERUS [48] and RAPTURE [56] are based on a symbolic representation of the system to be analyzed. In addition, hybrid approaches that combine the compact model representation of symbolic techniques with the good performance of numerical computations of explicit techniques have been developed [61]. Somewhat orthogonal to this approach are numerous reduction techniques, where the goal is to generate only a reduced sub-system which is "equivalent" (with respect to the properties to be verified) to the original system. Then model checking is applied to the reduced system, yielding the desired answer not only for the reduced system, but also for the original one. A large class of reduction techniques are bi-simulation–minimization techniques [55, 8, 77, 18, 58] that aim to aggregate bi-similar states and to construct an "equivalent" quotient of the original model. For models with non-trivial but interchangeable components, symmetry reduction techniques have been developed that use the inherent internal symmetries to reduce the state space.

Another class of reduction techniques are partial order reduction methods which have been thoroughly studied for non-probabilistic models [74, 52, 87, 38, 39, 76, 75] and have been extended to probabilistic systems in [9, 26, 7, 41, 42]. For partial order reduction, the starting point is usually a description of an asynchronous parallel system by a representation of the sub-systems that run in parallel, e.g., as in the language ProbMeLa that has been outlined in Sect. 2. The rough idea behind partial order reduction is to construct a reduced state graph by abolishing redundancies in the transition system that originate from the interleaving of independent activities that are executed in parallel. For independent actions α and β, the interleaving semantics represents their parallel execution by the nondeterministic choice between the action sequences $\alpha\beta$ and $\beta\alpha$. If $\alpha\beta$ and $\beta\alpha$ have the same effect to the control and program variables, and thus lead to the same state, the investigation of one order ($\alpha\beta$ or $\beta\alpha$) as a representative for both suffices under certain side conditions. More general, instead of constructing the full system $\mathcal{M}$, the goal of partial order reduction is to generate an "equivalent" sub-system $\mathcal{M}_{\mathsf{red}}$ of the full transition system $\mathcal{M}$. Here, "equivalence" is considered with respect to the type of property to be verified. Of course, the algorithmic construction and analysis of $\mathcal{M}_{\mathsf{red}}$ should be more efficient than model checking the full system $\mathcal{M}$. We give a small example to illustrate these ideas. Consider two processes $\mathcal{P}_1$ and $\mathcal{P}_2$ where $\mathcal{P}_1$ increments a variable x (action α) twice and $\mathcal{P}_2$ increments a variable y (action β) twice. Assume that we are only interested in the value of y, that is, each state is labeled with its y value. Then action α does not change the labeling, but action β does. Figure 7 shows the two processes and their parallel execution, where the shade of a state node represents its y value (the greater y is, the darker the node is). Now assume that we want to check whether the property

"The value of y never decreases."

holds on any path. For the system $\mathcal{P}_1|||\mathcal{P}_2$ of the parallel execution of $\mathcal{P}_1$ and $\mathcal{P}_2$ in Fig. 7, this means

"The shades of the nodes never get lighter."

along any path. Obviously, each path of the system satisfies this property. Now this property has a remarkable feature. In order to decide whether a path satisfies the property or not, it is only relevant what changes of the labeling occur along the path, but not how often a certain labeling is repeated before it changes. The property is so-called *stutter invariant.* It cannot distinguish between two paths that follow the same pattern of changes in the labeling (but may differ in the number of repetitions of a certain labeling). Such two paths are called *stutter equivalent.* Now consider the reduced system $(\mathcal{P}_1|||\mathcal{P}_2)_{\mathsf{red}}$ in Fig. 7. As any path of $\mathcal{P}_1|||\mathcal{P}_2$ has a stutter equivalent path in $(\mathcal{P}_1|||\mathcal{P}_2)_{\mathsf{red}}$ and the property under consideration cannot distinguish between such paths, it is sufficient to check whether all paths of the reduced system satisfy the

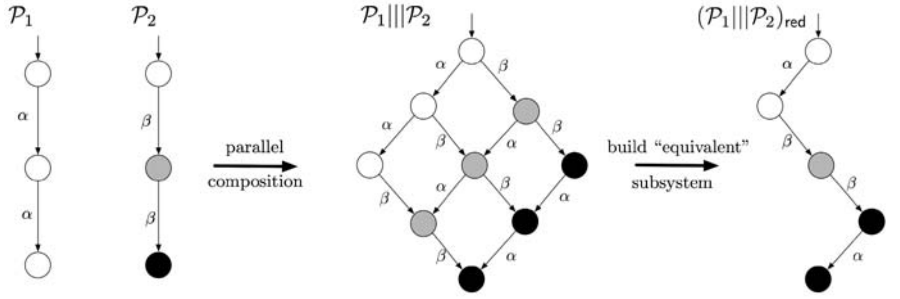

$\mathcal{P}_1$
$\mathcal{P}_2$
$\mathcal{P}_1|||\mathcal{P}_2$
$(\mathcal{P}_1|||\mathcal{P}_2)_{\mathsf{red}}$
α
β
parallel composition
build "equivalent" subsystem

Fig. 7. The idea of partial order reduction

property. If all paths of the reduced system satisfy the property, so do all paths of the original system (and vice versa as the reduced system is a subsystem of the original one). Thus, the reduced system is "equivalent" to the original system with respect to the property.

The goal of partial order reduction is to give criteria with which an "equivalent" reduced system can be generated. These criteria heavily depend on the class of properties that one wants to preserve (e.g., LT properties, branching time properties) and on the kind of model. In the early 1990s, several partial order reduction techniques have been developed for non-probabilistic systems [86, 74, 87, 52, 38, 39, 75, 88, 76]. In the last few years, one instance of partial order reduction techniques, the so-called *ample set method* has been generalized to the probabilistic setting [9, 26, 7, 41, 42].

The interested reader might want to compare our notion of independent actions with the independence relation used in Sect. 2.1 on weighted distributed systems in Chap. 10 in this handbook [36].

The Ample Set Method for MDPs and LT Properties

The rough idea of the ample set method is to assign to any reachable state s of an MDP $\mathcal{M}$ an action-set $\mathsf{ample}(s) \subseteq \mathsf{Act}(s)$ and to construct a reduced system $\mathcal{M}_{\mathsf{red}}$ that results by using the action-sets $\mathsf{ample}(s)$ instead of $\mathsf{Act}(s)$. That is, starting from the initial states of $\mathcal{M}$, one builds up $\mathcal{M}_{\mathsf{red}}$ by only applying ample transitions. The reduced system should be equivalent to the original system in the desired sense, e.g., simulation equivalent or bisimulation equivalent, etc. Depending on the desired equivalence, the defined ample sets have to fulfill certain conditions to ensure the equivalence. These equivalences typically identify those paths whose traces (i.e., words obtained from the paths by projection on the state labels) agree up to stuttering. In this context, stuttering refers to the repetition of the same state labels.

Definition 5.1 (Stutter equivalence). *Two infinite words ς_1 and ς_2 over the alphabet Σ are called stutter equivalent, denoted by*

$$\varsigma_1 \equiv_{\mathrm{st}} \varsigma_2,$$

if and only if there is an infinite word $a_1 a_2 \ldots$ over the alphabet Σ such that

$$\varsigma_1 = a_1^{k_1} a_2^{k_2} \ldots \quad \textit{and} \quad \varsigma_2 = a_1^{n_1} a_2^{n_2} \ldots,$$

where $k_i, n_i \in \mathbb{N}_{\geq 1}$. Two infinite paths π_1 and π_2 in a state-labeled MDP are called stutter equivalent, denoted by $\pi_1 \equiv_{\mathrm{st}} \pi_2$, if and only if their traces $\mathsf{trace}(\pi_1)$ and $\mathsf{trace}(\pi_2)$ over 2^{AP} are stutter equivalent.

An LT property over AP is called stutter invariant, if it cannot distinguish between stutter equivalent paths.

Definition 5.2 (Stutter invariant LT properties). *An LT property E over AP is called stutter invariant if for all stutter equivalent words $\varsigma_1, \varsigma_2 \in (2^{\mathsf{AP}})^\omega$ we have that*

$$\varsigma_1 \in \mathsf{E} \quad \textit{if and only if} \quad \varsigma_2 \in \mathsf{E}.$$

We call two MDPs stutter equivalent if for each scheduler of one of the MDPs there exists a scheduler of the other MDP such that the schedulers yield the same probabilities for any stutter invariant LT property.

Definition 5.3 (Stutter equivalence for MDPs). *Given two MDPs $\mathcal{M}_i = (S_i, \mathsf{Act}_i, \delta_i, \mu_i, \mathsf{AP}_i, L_i)$, with $i = 1, 2$, we call $\mathcal{M}_1$ and $\mathcal{M}_2$ stutter equivalent, denoted by*

$$\mathcal{M}_1 \equiv_{\mathrm{st}} \mathcal{M}_2,$$

if and only if for each scheduler $\mathcal{U}_1$ of $\mathcal{M}_1$ there exists a scheduler $\mathcal{U}_2$ of $\mathcal{M}_2$ such that,

$$\mathsf{Pr}^{\mathcal{M}_1,\mathcal{U}_1}(\mathsf{E}) = \mathsf{Pr}^{\mathcal{M}_2,\mathcal{U}_2}(\mathsf{E})$$

for each stutter invariant measurable LT property $\mathsf{E} \subseteq (2^{\mathsf{AP}})^\omega$, and vice versa.

Before explaining the ample set method, we briefly illustrate its impact on probabilistic linear-time model checking. Assume that we are given two stutter equivalent MDPs $\mathcal{M}_1, \mathcal{M}_2$ and a stutter invariant measurable LT property E. Then

$$\sup_{\mathcal{U} \in \mathsf{Sched}^{\mathcal{M}_1}} \mathsf{Pr}^{\mathcal{M}_1,\mathcal{U}}(\mathsf{E}) = \sup_{\mathcal{U} \in \mathsf{Sched}^{\mathcal{M}_2}} \mathsf{Pr}^{\mathcal{M}_2,\mathcal{U}}(\mathsf{E}).$$

The corresponding equality with inf instead of sup certainly also holds. Hence, two stutter equivalent MDPs $\mathcal{M}_1$ and $\mathcal{M}_2$ are equivalent with respect to stutter invariant measurable linear-time specifications. A prominent class of stutter invariant measurable LT properties is the LTL fragment that does not use the “NextStep”-operator [90].

The following result (Theorem 5.8 below) has been established in [9]. It requires ample sets $\mathsf{ample}(s)$ for $s \in S$ that enjoy the properties (A0)–(A4) shown in Fig. 9 (these will be explained later) and asserts the stutter equivalence of the original MDP $\mathcal{M}$ and the reduced MDP $\mathcal{M}_{\mathsf{red}}$ that arises from $\mathcal{M}$ by removing all enabled actions of a state s that are not included in the ample set of s. The precise definition of $\mathcal{M}_{\mathsf{red}}$ is as follows.

Definition 5.4 (The reduced MDP). *Let $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ be an MDP, and suppose that for each state $s \in S$, $\mathsf{ample}(s)$ is a non-empty subset of $\mathsf{Act}(s)$. Then the reduced MDP $\mathcal{M}_{\mathsf{red}}$ is the MDP $(S_{\mathsf{red}}, \mathsf{Act}, \delta_{\mathsf{red}}, \mu, \mathsf{AP}, L_{\mathsf{red}})$ where the state space of $\mathcal{M}_{\mathsf{red}}$ is the smallest sub-set S_{red} of S such that:*

- $\{s \in S \mid \mu(s) > 0\} \subseteq S_{\mathsf{red}}$.
- *Whenever $s \in S_{\mathsf{red}}$ and $\alpha \in \mathsf{ample}(s)$ then $\{s' \in S \mid \delta(s, \alpha, s') > 0\} \subseteq S_{\mathsf{red}}$.*

The transition probability function $\delta_{\mathsf{red}} : S_{\mathsf{red}} \times \mathsf{Act} \times S_{\mathsf{red}} \to [0, 1]$ is given by

$$\delta_{\mathsf{red}}(s, \alpha, s') = \begin{cases} \delta(s, \alpha, s') & \textit{if } \alpha \in \mathsf{ample}(s) \\ 0 & \textit{otherwise.} \end{cases}$$

The labeling function $L_{\mathsf{red}} : S_{\mathsf{red}} \to 2^{\mathsf{AP}}$ is the restriction of $\mathcal{M}$'s labeling function to the state space of $\mathcal{M}_{\mathsf{red}}$, i.e., $L_{\mathsf{red}}(s) = L(s)$ for all $s \in S_{\mathsf{red}}$.

Thus, $\mathcal{M}_{\mathsf{red}}$ can be obtained by an on-the-fly algorithm which first generates all initial states of $\mathcal{M}$ and then successively expands each generated state s by considering all actions $\alpha \in \mathsf{ample}(s)$ and generating the α-successors of s that have not been generated before.

The following Theorem 5.8 [9, 43] ensures that given a deterministic ω-automaton $\mathcal{A}$ that accepts a stutter invariant language, it suffices to model check $\mathcal{M}_{\mathsf{red}}$ against $\mathcal{A}$ instead of $\mathcal{M}$, provided that the ample sets satisfy the conditions (A0)–(A4) in Fig. 9. Before presenting the theorem, we define *stutter actions*, resp. *independent actions* that are used in the condition (A1), resp. (A2). Stutter actions are actions that have no effect on the state labels, no matter in which state they are taken.

Definition 5.5 (Stutter action). *Given an MDP $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$, we call an action $\alpha \in \mathsf{Act}$ a stutter action if and only if for all states $s, t \in S$,*

$$\delta(s, \alpha, t) > 0 \quad \textit{implies} \quad L(s) = L(t).$$

The main ingredient of any partial order reduction technique in the probabilistic or non-probabilistic setting is an adequate notion for independence of actions. The rough idea is a formalization of actions belonging to different processes that are executed in parallel and do not affect each other, e.g., as they only refer to local variables and do not require any kind of synchronization. In non-probabilistic systems, independence of two actions α and β means that, for any state s where both α and β are enabled, the execution of α does not affect the enabledness of β (i.e., the α-successor of s has an outgoing β-transition), and vice versa, and in addition the action sequences $\alpha\beta$ and $\beta\alpha$ lead to the same state. In the probabilistic setting, it is additionally required that $\alpha\beta$ and $\beta\alpha$ have the same probabilistic effect.

Definition 5.6 (Independence of actions, cf. [9, 26]). *Two actions α and β with $\alpha \neq \beta$ are called independent in an MDP $\mathcal{M}$ if and only if for each state $s \in S$ with $\{\alpha, \beta\} \subseteq \mathsf{Act}(s)$ it holds that:*

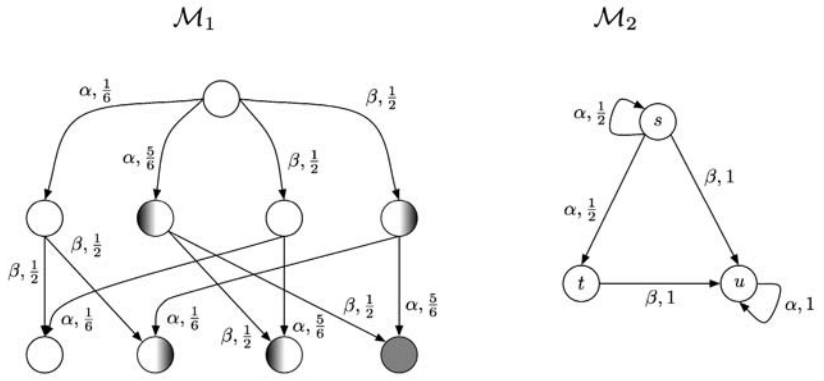


Fig. 8. Examples of independent actions

1. $\delta(s, \alpha, t) > 0$ *implies* $\beta \in \mathsf{Act}(t)$.
2. $\delta(s, \beta, u) > 0$ *implies* $\alpha \in \mathsf{Act}(u)$.
3. For each state $v \in S$:

$$\sum_{t \in S} \delta(s, \alpha, t) \cdot \delta(t, \beta, v) = \sum_{u \in S} \delta(s, \beta, u) \cdot \delta(u, \alpha, v).$$

Two different actions α *and* β *are called dependent if and only if* α *and* β *are not independent. If* $A \subseteq \mathsf{Act}$ *and* $\alpha \in \mathsf{Act} \backslash A$*, then* α *is called independent from* A *if and only if for each action* $\beta \in A$*,* α *and* β *are independent. Otherwise,* α *is called dependent on* A*.*

Example 5.7 (Independent actions). Figure 8 shows a fragment of an MDP $\mathcal{M}_1$ representing the parallel execution of independent actions α and β. For example, α might stand for the outcome of the experiment of tossing a "one" with a dice, while β stands for tossing a fair coin. In general, whenever α and β represent stochastic experiments that are independent in the classical sense, then α and β viewed as actions of an MDP are independent. However, there are also other situations where two actions can be independent that do not have a fixed probabilistic branching pattern. For instance, actions α and β in the MDP $\mathcal{M}_2$ in Fig. 8 are independent. To see this, first notice that only in state s both α and β are enabled. The α-successors t, s of s have a β-transition to state u, while the β-successor u has an α-transition to itself. The probabilistic effect under the action sequences $\alpha\beta$ and $\beta\alpha$ is the same as in either case state u is reached with probability 1.

Theorem 5.8 (Ample set method for MDPs). *Let* $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ *be an MDP and* $\mathsf{ample} : S \to 2^{\mathsf{Act}}$ *a function satisfying conditions (A0)–(A4) in Fig. 9. Then*

$$\mathcal{M} \equiv_{\mathrm{st}} \mathcal{M}_{\mathsf{red}},$$

where $\mathcal{M}_{\mathsf{red}}$ *denotes the reduced MDP that emanates from the MDP* $\mathcal{M}$ *and the ample sets defined by the function* ample *according to Definition 5.4.*

(A0) (Non-emptiness condition) For each state $s \in S$, it holds that $\emptyset \neq \mathsf{ample}(s) \subseteq \mathsf{Act}(s)$.

(A1) (Stutter condition) If $s \in S_{\mathsf{red}}$ and $\mathsf{ample}(s) \neq \mathsf{Act}(s)$, then all actions $\alpha \in \mathsf{ample}(s)$ are stutter actions.

(A2) (Dependence condition) For each path $\pi = s \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} s_n \xrightarrow{\gamma} \cdots$ in $\mathcal{M}$ where $s \in S_{\mathsf{red}}$ and γ is dependent on $\mathsf{ample}(s)$ there exists an index $i \in \{1, \ldots, n\}$ such that $\alpha_i \in \mathsf{ample}(s)$.

(A3) (End component condition) For each end component (T, A) in $\mathcal{M}_{\mathsf{red}}$ we have that: $\alpha \in \bigcap_{t \in T} A(t)$ implies $\alpha \in \bigcup_{t \in T} \mathsf{ample}(t)$.

(A4) (Branching condition) If $\pi = s \xrightarrow{\alpha_1} s_1 \xrightarrow{\alpha_2} \cdots \xrightarrow{\alpha_n} s_n \xrightarrow{\alpha} \cdots$ is a path in $\mathcal{M}$ where $s \in S_{\mathsf{red}}, \alpha_1, \ldots, \alpha_n, \alpha \notin \mathsf{ample}(s)$ and α is probabilistic, then $|\mathsf{ample}(s)| = 1$.

Fig. 9. Conditions for the ample sets of MDPs

We now provide explanations why conditions (A0)–(A4) that have been proposed in [9] ensure the stutter equivalence of $\mathcal{M}$ and $\mathcal{M}_{\mathsf{red}}$. Condition (A0) simply assures that $\mathcal{M}_{\mathsf{red}}$ is a sub-MDP of $\mathcal{M}$ (recall that in Definition 2.1 we required that all states of an MDP are non-terminal). Thus, each scheduler of $\mathcal{M}_{\mathsf{red}}$ is also a scheduler of $\mathcal{M}$. So, the interesting part is the transformation of a given scheduler $\mathcal{U}$ of $\mathcal{M}$ into an "equivalent" scheduler $\mathcal{U}_{\mathsf{red}}$ of $\mathcal{M}_{\mathsf{red}}$ (where "equivalence" is understood with respect to the probabilities of stutter invariant measurable LT properties). The details of the scheduler transformation $\mathcal{U} \to \mathcal{U}_{\mathsf{red}}$ are rather technical and will not be explained here. The main idea is an iterative approach where an infinite sequence $\mathcal{U}_0 = \mathcal{U}, \mathcal{U}_1, \mathcal{U}_2, \ldots$ of schedulers for $\mathcal{M}$ is constructed such that:

- $\mathcal{U}_i, \mathcal{U}_{i+1}, \mathcal{U}_{i+2}, \ldots$ agree on all finite paths of length at most i.
- All finite $\mathcal{U}_i$-paths of length i are paths in $\mathcal{M}_{\mathsf{red}}$.
- $\Pr^{\mathcal{M},\mathcal{U}_i}(E) = \Pr^{\mathcal{M},\mathcal{U}}(E)$ for all stutter invariant measurable LT properties E.

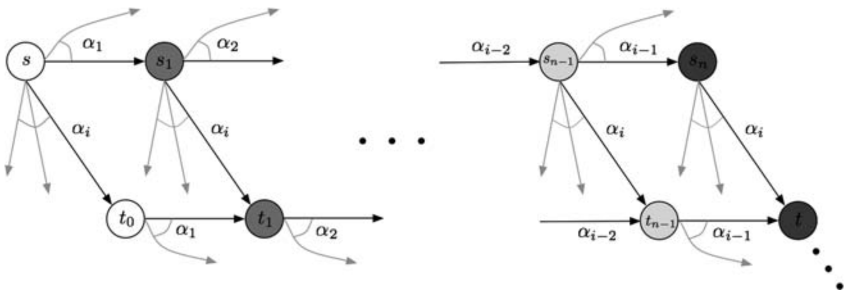
The scheduler $\mathcal{U}_{\mathsf{red}}$ is then defined to be the limit of the schedulers $\mathcal{U}_i$, that is,

$$\mathcal{U}_{\mathsf{red}}(\pi) = \mathcal{U}_{i+1}(\pi)$$

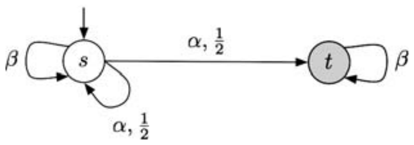
if π is a path of length i in $\mathcal{M}_{\mathsf{red}}$.

The transformations $\mathcal{U}_i \mapsto \mathcal{U}_{i+1}$ all rely on the same schema. For simplicity, we just give a very rough sketch of the idea for the case $i = 0$ and assume that $\mathcal{U}_0 = \mathcal{U}$ is a deterministic scheduler. Suppose we are given a $\mathcal{U}$-path starting in state s that relies on the action sequence $\alpha_1\alpha_2\alpha_3 \ldots$ and $\alpha_1 \notin \mathsf{ample}(s)$ (otherwise, $\mathcal{U}_{\mathsf{red}}$ just chooses α_1 with probability 1). If at least one of these actions belongs to $\mathsf{ample}(s)$, then we pick the smallest index i such that $\alpha_i \in \mathsf{ample}(s)$. Note that condition (A1) ensures that α_i is a stutter action as $\alpha_1 \notin \mathsf{ample}(s)$. Condition (A2) ensures that α_i is independent from $\alpha_1, \ldots, \alpha_{i-1}$.

Hence, we can switch from the action sequence $\alpha_1\alpha_2\ldots\alpha_{i-1}\alpha_i$ to the action sequence $\alpha_i\alpha_1\alpha_2\ldots\alpha_{i-1}$. Both action sequences can be executed from state s and yield the same distribution over the states that can be reached afterward. In addition, the action sequences $\alpha_1\alpha_2\ldots\alpha_{i-1}\alpha_i$ and $\alpha_i\alpha_1\alpha_2\ldots\alpha_{i-1}$ produce stutter equivalent paths that end in the same state (recall that α_i is a stutter action). These ideas are sketched in the following picture.



Since $\alpha_i \in \mathsf{ample}(s)$, scheduler $\mathcal{U}_1$ will choose α_i with some positive probability (the probabilities for the chosen actions rely on a rather complex formula that will not be discussed here). If the given $\mathcal{U}$-path does not contain an action in $\mathsf{ample}(s)$ then we pick an arbitrary action $\beta \in \mathsf{ample}(s)$ (this is possible by (A0)) and replace the action sequence $\alpha_0\alpha_1\ldots$ with $\beta\alpha_0\alpha_1\ldots$ (this is possible by (A2) as β is independent from each α_i). The scheduler $\mathcal{U}_1$ will then choose β with some positive probability. Note that this also yields some path that is stutter equivalent to the given $\mathcal{U}$-path. In summary, given a $\mathcal{U}$-path π starting in state s, the basic idea is to permute the first ample action of s that occurs along π to the beginning of the action sequence of π. If no such action exists, an arbitrary ample action of s is pre-pended to the action sequence of π. This step is then repeated ad infinitum to yield a scheduler $\mathcal{U}_{\mathsf{red}}$ of $\mathcal{M}_{\mathsf{red}}$. However, we cannot immediately conclude that $\mathcal{U}$ and $\mathcal{U}_{\mathsf{red}}$ yield the same probabilities for stutter invariant measurable LT properties because the generated $\mathcal{U}_{\mathsf{red}}$-paths might "delay" a certain action of a $\mathcal{U}$-path ad infinity as in the following example.



The state labeling is given by the shades of the states, thus β is a stutter action, while α is not. For $\mathsf{ample}(s) = \{\beta\}$ and scheduler $\mathcal{U}$ where $\mathcal{U}(\pi) = \alpha$ for all paths π with $\mathsf{last}(\pi) = s$, the construction sketched above (see [43] for the details) yields

$$\mathcal{U}_i(\underbrace{s \xrightarrow{\beta} s \xrightarrow{\beta} \cdots \xrightarrow{\beta} s}_{\text{length } j}) = \begin{cases} \beta & \text{for } j \leq i-1, \\ \alpha & \text{for } j = i. \end{cases}$$

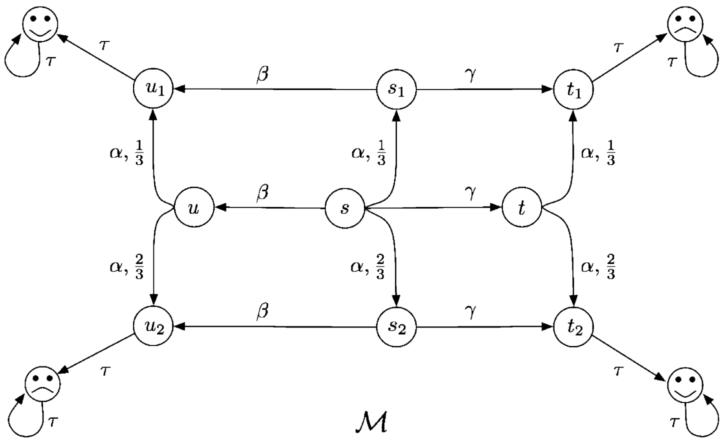


Fig. 10. Example to justify condition (A4)

Thus, scheduler $\mathcal{U}_{\mathsf{red}}$ always schedules β in the state s. In fact, $\mathcal{U}_{\mathsf{red}}$ is the only scheduler for $\mathcal{M}_{\mathsf{red}}$ as $\mathcal{M}_{\mathsf{red}}$ consists only of state s with the β-loop. Under $\mathcal{U}$ and each of the schedulers $\mathcal{U}_i$, we obtain probability 1 to reach the gray state t, while the probability to reach state t under $\mathcal{U}_{\mathsf{red}}$ is 0. However, in this example, conditions (A0), (A1), (A2), and (A4) hold, but the end component $(s, \{\beta\})$ of $\mathcal{M}_{\mathsf{red}}$ violates the end component condition (A3), which ensures that in the scheduler transformation almost surely there is no action of $\mathcal{M}$ that is postponed forever. Note that condition (A3) refers to end components in the reduced MDP $\mathcal{M}_{\mathsf{red}}$ rather than $\mathcal{M}$ (the definition of an end component has been provided in Definition 4.4).

It is worth noting that conditions (A0)–(A3) suffice in the non-probabilistic setting to ensure the equivalence between a transition system and its reduced system with respect to stutter invariant LT properties [74, 75]. However, for MDPs, we need the additional branching condition (A4). The intuitive reason for this is that the experiments

"first toss a coin, then decide between action β and γ" and
"first decide between action β and γ, then toss a coin"

are different. This becomes obvious in the example shown in Fig. 10. Starting in state s of $\mathcal{M}$, if first the coin is tossed (action α) and then, depending on its outcome, action β is chosen in state s_1 and action γ is chosen in state s_2, then this yields that a "smiling" state is reached with probability one. If, however, the choice between β and γ is resolved before the coin is tossed, that is the β-transition or the γ-transition is taken in state s, then taking α in state u, resp. state t, will not result in reaching the "smiling" states with probability one. Note that if we choose $\mathsf{ample}(s) = \{\beta, \gamma\}$ for the MDP $\mathcal{M}$ shown in Fig. 10, then conditions (A0)–(A3) are satisfied, whereas condition (A4) is violated. So, a scheduler of $\mathcal{M}$ might schedule a probabilistic non-ample action of the starting state s. Depending on the outcome (moving to state s_1 or s_2), the scheduler chooses different ample actions (of s). Thus, choosing α first, postpones the real non-deterministic decision between the ample actions β and γ. The reduced system $\mathcal{M}_{\mathsf{red}}$ is forced to decide immediately for a particular ample action β or γ of s (more precisely a distribution over the ample actions of s) in its first step before the outcome of α is known. This decision is fixed from then on. It is exactly this behavior that one has to forbid to gain stutter equivalence between the given system $\mathcal{M}$ and its reduced system. That means that if the system can branch probabilistically with non-ample actions (with respect to the starting state), then there should be only one ample action of the starting state. The additional branching condition (A4) ensures exactly this.

The above remarks only present rough explanations to justify conditions (A0)–(A4). For a full proof of Theorem 5.8, see [43].

Remark 5.9. Theorem 5.8 ensures that, given a deterministic ω-automaton that accepts a stutter invariant language, it suffices to model check $\mathcal{M}_{\mathsf{red}}$ instead of $\mathcal{M}$. As $\mathcal{M}_{\mathsf{red}}$ is in general smaller than $\mathcal{M}$, this yields a possible speed-up of the analysis. Of course, the algorithmic construction of appropriate ample sets together with the construction and the analysis of $\mathcal{M}_{\mathsf{red}}$ should be more efficient than model checking the full system $\mathcal{M}$. Note that even a reduction that eliminates only actions, but does not shrink the state space, might yield a speed-up of the analysis as the probabilistic model checking procedure relies on solving linear programs where the number of linear inequalities for any state s is given by the number of outgoing actions of s.

Experimental Results

The partial order approach for MDPs has been implemented in the model checker LIQUOR [3, 5] using heuristics for approximating the conditions (A2), (A3) and (A4) given in Fig. 9. These heuristics use a superset of the dependence relation and rely on a pre-analysis of the control flow graph induced by programs given in the specification language ProbMeLa (Sect. 2). Several case studies with LIQUOR have shown that the partial order reduction (POR) can lead to a major speed-up and can also decrease the space requirements. To give an impression on the dimension of the time and space requirements for realistic systems, the following table summarizes the results for a randomized leader election protocol (where variable N in the first column denotes the number of parallel processes in the model):

Randomized leader election

	without POR			with POR		
N	states	transitions	time	states	transitions	time
4	53621	156072	1.1 s	21063	78072	1.1 s
5	896231	$3.2 \cdot 10^6$	34 s	299670	$1.3 \cdot 10^6$	21 s
6	$1.1 \cdot 10^7$	$6.2 \cdot 10^7$	813 s	$4.1 \cdot 10^6$	$1.4 \cdot 10^7$	180 s

In other cases, for instance in models of parallel processes that share common synchronization points, the reduction can be even better. More results and more detailed information about applied heuristics and techniques can be found in [5].

6 Partially Observable MDPs

The analysis techniques of Sects. 3, 4, and 5 yield worst-case schedulers where the probability for a certain undesired event is maximal, or dually, where the probability for the desired behavior is minimal. To some extent, these techniques are also applicable to *controller synthesis problems* where the goal is to design a scheduler (i.e., a controller) that resolves the internal nondeterminism and optimizes the probabilities for a certain LT property. However, in this context, the general notion of a scheduler appears to be inadequate since it relies on the complete knowledge of the system history. Consider again the Monty Hall problem from Example 2.2 and the corresponding MDP $\mathcal{M}$ in Fig. 3. We saw in Example 2.2 that

$$\sup_{\mathcal{U} \in \mathsf{Sched}} \mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\, \mathsf{get\_car}) = 1,$$

where the supremum is attained by the scheduler $\mathcal{U}$ with

$$\mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{keep}) = 1$$

and

$$\mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{switch}) = \mathcal{U}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{switch}) = 1.$$

As already pointed out in the example, this scheduler $\mathcal{U}$ does not reflect a realistic choice of the contestant, as the contestant does not know whether she/he has chosen the door with the car behind it, or not. So, the only realistic schedulers (that model a contestant's choice) are schedulers that make the same choice for each path that ends either in state t_1, t_2, or t_3. In this case, these are the two schedulers $\mathcal{U}_{\mathsf{s}}$ and $\mathcal{U}_{\mathsf{k}}$ with

$$\mathcal{U}_{\mathsf{s}}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{switch}) = \mathcal{U}_{\mathsf{s}}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{switch}) = \mathcal{U}_{\mathsf{s}}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{switch}) = 1$$

and

$$\mathcal{U}_{\mathsf{k}}\big(s \xrightarrow{\mathsf{choose}} t_1\big)(\mathsf{keep}) = \mathcal{U}_{\mathsf{k}}\big(s \xrightarrow{\mathsf{choose}} t_2\big)(\mathsf{keep}) = \mathcal{U}_{\mathsf{k}}\big(s \xrightarrow{\mathsf{choose}} t_3\big)(\mathsf{keep}) = 1$$

where the contestant either decides to switch the door or to keep it. So, in this scenario, we are actually interested in computing the supremum, resp. infimum of $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\, \mathsf{get\_car})$ under all "realistic" schedulers. A model that allows us to express such requests is given by partially observable Markov decision processes (POMDPs) [84, 70, 71, 67].

Definition 6.1 (Partially observable Markov decision process). *A partially observable Markov decision process is a pair $(\mathcal{M}, \sim)$, where:*

- $\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L)$ *is a Markov decision process.*
- $\sim \subseteq S \times S$ *is an equivalence relation such that for all states $s, t \in S$ with $s \sim t$ we have $\mathsf{Act}(s) = \mathsf{Act}(t)$.*

If $s \in S$, then $[s]_\sim$ denotes the equivalence class of state s with respect to $\sim$.

Given a POMDP $(\mathcal{M}, \sim)$, an observation-based scheduler $\mathcal{U}$ is a scheduler for $\mathcal{M}$ that is consistent with $\sim$, i.e., $\mathcal{U}(s_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} s_n) = \mathcal{U}(t_0 \xrightarrow{\alpha_1} \cdots \xrightarrow{\alpha_n} t_n)$ if $s_i \sim t_i$ for $0 \leq i \leq n$. The set of observation-based schedulers is denoted by $\mathsf{Sched}^{(\mathcal{M},\sim)}$.

If we equip the MDP $\mathcal{M}$ for the Monty Hall problem with the equivalence relation $\sim$ given by $[s]_\sim = \{s\}$, $[u_1]_\sim = \{u_1\}$, $[u_2]_\sim = \{u_2\}$ and

$$[t_1]_\sim = [t_2]_\sim = [t_3]_\sim = \{t_1, t_2, t_3\},$$

then the deterministic observation-based schedulers of the POMDP $(\mathcal{M}, \sim)$ are the "realistic" schedulers that actually model a contestant's choice in the game. Thus, in the Monty Hall scenario, we are interested in computing

$$\sup_{\mathcal{U} \in \mathsf{Sched}_{\mathsf{D}}^{(\mathcal{M},\sim)}} \mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\, \mathsf{get\_car}),$$

resp. in computing the infimum. Here, $\mathsf{Sched}_{\mathsf{D}}^{(\mathcal{M},\sim)}$ denotes the set of deterministic observation-based schedulers of the POMDP $(\mathcal{M}, \sim)$. Unfortunately, we cannot expect to have algorithmic solutions for the task to compute extremal reachability probabilities, when ranging over observation-based schedulers. For a similar partial information model which uses distributed schedulers instead of observation-based schedulers, it has been shown that there is no algorithm that computes this supremum under all distributed schedulers. In fact, the supremum is not even approximable [37]. For the model of POMDPs, there even exist the following undecidability results for qualitative questions, which have recently been shown in [2, 43].

In what follows, we use LTL-notations to denote LT properties. The symbol $\Diamond$ stands for "eventually", $\Box$ for "always". Thus, the combination $\Box\Diamond$ denotes "infinitely often" and $\Diamond\Box$ means "continuously from some moment on".

Theorem 6.2 (Undecidability results for POMDPs). *The following problems are undecidable. Given a POMDP $(\mathcal{M}, \sim)$ and a set B of states in $\mathcal{M}$, is there a deterministic observation-based scheduler $\mathcal{U}$ for $(\mathcal{M}, \sim)$ such that:*

(a) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Box\Diamond\, B) > 0$*?*
(b) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\Box\, B) = 1$*?*

Those results as well as the undecidability result mentioned above on qualitative reachability from [37] are remarkable since the corresponding questions for (fully observable) MDPs are decidable in polynomial time (see Sect. 4). However, some other variants of qualitative verification problems for POMDPs have been shown to be decidable [29, 2, 43].

Theorem 6.3 (Decidable problems for POMDPs). *The following problems are decidable. Given a POMDP* $(\mathcal{M}, \sim)$ *and a set* B *of states in* $\mathcal{M}$*, does there exist* $\mathcal{U} \in \mathsf{Sched}^{(\mathcal{M},\sim)}$ *such that:*

(a) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Box\, B) > 0$*?*
(b) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\, B) > 0$*?*
(c) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Box\, B) = 1$*?*
(d) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\, B) = 1$*?*
(e) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Box\Diamond\, B) = 1$*?*
(f) $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(\Diamond\Box\, B) > 0$*?*

In fact, in [2, 43], it has been shown that the problems (e) and (d) are reducible to each other and that the latter one can be reduced to the similar question for (fully observable) MDPs using an advanced powerset construction. The proof of (f) (see [43]) uses the interreducibility of (d) and (e), and (a) which has been shown in [29].

7 Conclusion

In this chapter, we have summarized the main features of Markov decision processes as an operational model for parallel probabilistic systems and model checking against ω-regular linear-time properties. We have supposed here that the properties are given by deterministic ω-automata. Instead of automata specifications, any logic that can be translated into automata can be used to provide a formalization of the requirements such as linear temporal logic, the mu-calculus, or monadic second-order logic. As quantitative reasoning about probabilistic systems relies on a combination of graph-based and numerical methods, heuristics that attack the state space explosion problem are even more important than in the non-probabilistic case. In this chapter, we have explained the partial order reduction approach. Several other reduction techniques to reduce the time and space requirements such as abstraction techniques, minimization with simulation-like relations, symmetry reduction, and symbolic approaches with variants of binary decision diagrams have been discussed in the literature and are topics of current research projects (see the references given in Sects. 1 and 5).

One of the key features of model checking tools is the concept of counterexamples that can be returned to the user if the checked property does not hold for the system. In the probabilistic setting, counterexamples are more complex, as single error traces are inadequate. First results on the generation of counterexamples for probabilistic systems and their use in abstraction-refinement approaches are presented in the recent papers [1, 46, 50]. Another current research trend is the investigation of alternating-time and game-based approaches that deal with MDP-like models representing the activities of several players. The concept of partially observable MDPs is one instance thereof (see Sect. 6), another instance are stochastic $2\frac{1}{2}$-player games (see, e.g., [20, 30, 19, 53]).

The classical model of Markov decision processes is adequate for the analysis against safety and liveness properties and other conditions on the temporal order of events, but not to reason about timing constraints within a dense time domain. The treatment of continuous-time Markov decision processes or other stochastic models where time-dependent distributions are attached to the transitions (e.g., [10]) or probabilistic variants of timed automata are examples for other very active research fields [64, 57, 63].

Many concepts for reasoning about MDPs viewed as acceptors for languages over finite words (Rabin's probabilistic finite automata [81]) can be generalized rather naturally for weighted automata. Such a generalization of concepts for MDPs to weighted automata is, however, less clear for the case of infinite words. It would also be interesting to see whether the measure-theoretic concepts that yield the basis to define the probabilities for ω-regular properties can be adapted to other classes of weighted automata to reason about the weights for (measurable) sets of infinite paths. This could yield an interesting alternative to the concept of discounting which is well known for MDPs augmented with a reward function that assigns rewards to states and/or actions (see, e.g., [80]) and has been discussed recently in [32–34] for weighted automata and to the approaches investigated in Chaps. 3 and 5 that enforce convergence of infinite series by imposing certain algebraic assumptions on the semiring of a weighted automaton.

8 Appendix

In this appendix, we give the formal definitions needed for the theory of MDPs that is used in the previous sections.

Markov Chains

We first start with the definition of a probability distribution.

Definition 8.1 (Probability distribution). *Let S be a countable set. A probability distribution on S is a function*

$$\mu : S \to [0,1] \quad \textit{such that} \quad \sum_{s \in S} \mu(s) = 1.$$

Given a probability distribution μ on S, $\mathsf{supp}(\mu)$ denotes the support *of μ, i.e., the set of states $s \in S$ with $\mu(s) > 0$. For each $s \in S$, μ_s denotes the unique*

Dirac distribution on S that satisfies $\mu_s(s) = 1$. By $\mathsf{Distr}(S)$, we denote the set of all probability distributions on S.

Next, we give the definition of a discrete Markov chain, which is basically a directed graph where the edges are labeled with a probability in $[0, 1]$, such that in each state the probabilities of its outgoing edges sum up to one. Moreover, there is an initial probability distribution on the vertices of the graph.

Definition 8.2 (Discrete Markov chain). *A discrete Markov chain is a tuple*

$$\mathcal{M} = (S, \mathsf{p}, \mu),$$

where:

- *S is a countable non-empty set of states.*
- *$\mathsf{p} : S \times S \to [0, 1]$ is the so-called transition probability function such that $\mathsf{p}(s, .)$ is a probability distribution on S for each $s \in S$.*
- *μ is a probability distribution on S (called the initial distribution).*

Let $T = \{(s, t) \mid \mathsf{p}(s, t) > 0, s, t \in S\}$ be the set of transitions with positive probability. We refer to the directed graph (S, T) as the *underlying graph* of $\mathcal{M}$. Note that (S, T) has no terminal nodes. A discrete Markov chain induces a stochastic process on the set S of its states in a natural way. The probability that the process starts in a certain state (the 0th step) is determined by the starting distribution. Moreover, being in state s in the $(n-1)$st step, the probability that the process is in state t in the nth step is equal to $p(s, t)$. The fact that those probabilities do not depend on the previous steps (*history-independent* or *memoryless*) is called the *Markov property.* For a detailed discussion on Markov chains, see, e.g., [59]. Before we go on, we fix some notation for paths of a discrete Markov chain.

Definition 8.3 (Path and corresponding notation). *An (in)finite path of a discrete Markov chain $\mathcal{M}$ is an (in)finite state sequence $\pi = s_0 s_1 \ldots$ such that $p(s_i, s_{i+1}) > 0$ for all i. Given a finite path $\pi = s_0 s_1 \ldots s_n$, the length $|\pi|$ of π equals n. For an infinite path π, the length is equal to ∞. Given a path $\pi = s_0 s_1 \ldots$ and $i \leq |\pi|$, we denote the ith state of π by π_i (i.e., $\pi_i = s_i$) and the ith prefix by $\pi{\uparrow}^i = s_0, s_1, \ldots, s_i$. We denote by $\mathsf{Path}_{\mathsf{fin}}$ (resp. $\mathsf{Path}_{\mathsf{inf}}$) the set of finite (resp. infinite) paths of a given discrete Markov chain and by $\mathsf{Path}_{\mathsf{fin}}(s)$ (resp. $\mathsf{Path}_{\mathsf{inf}}(s)$) the set of finite (resp. infinite) paths starting in the state s. The empty path is denoted by ϵ.*

If necessary, then we will index Path by the corresponding system, e.g., $\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}$. We now define the probability space that formalizes the stochastic process induced by a discrete Markov chain.

Definition 8.4 (Basic cylinder). *Given a discrete Markov chain $\mathcal{M}$, we define, for every $\pi \in \mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}}$, the* basic cylinder *of π as*

$$\varDelta(\pi) = \left\{\rho \in \mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}} : \rho{\uparrow}^{|\pi|} = \pi\right\}.$$

Definition 8.5 (Probability space of a discrete Markov chain). *Given a discrete Markov chain $\mathcal{M} = (S, \mathsf{p}, \mu)$, we define a probability space*

$$\Psi = \big(\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}, \Delta, \mathsf{Pr}\big),$$

such that:

- *Δ is the σ-algebra generated by the empty set and the set of basic cylinders in $\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}$.*
- *Pr is the uniquely induced probability measure which satisfies the following: $\mathsf{Pr}(\Delta(\epsilon)) = 1$ and for every basic cylinder $\Delta(s_0, s_1, \ldots, s_n)$ over S:*

$$\mathsf{Pr}\big(\Delta(s_0, s_1, \ldots, s_n)\big) = \mu(s_0) \cdot \mathsf{p}(s_0, s_1) \cdot \cdots \cdot \mathsf{p}(s_{n-1}, s_n).$$

Given a state $s \in S$, we denote by Pr_s the probability measure that is obtained if $\mathcal{M}$ is equipped with the starting distribution μ_s, thus $\mathsf{Pr}_s(\Delta(s)) = 1$. We call the a set $P \subseteq \mathsf{Path}_{\mathsf{inf}}$ of infinite paths measurable if and only if $P \in \Delta$.

The existence of the induced probability measure Pr follows from a well-known theorem in measure theory, which is known as Carathéodory's measure extension theorem. The uniqueness follows from the fact that the set of basic cylinders is intersection-stable. For more information on measure theory, see, e.g., [13].

Markov Decision Processes

We will now explain formally the probability space that emanates from a Markov decision process and a given scheduler. Let

$$\mathcal{M} = (S, \mathsf{Act}, \delta, \mu, \mathsf{AP}, L),$$

be an MDP and $\mathcal{U}$ a scheduler that resolves the nondeterminism in $\mathcal{M}$ (for the definition of an MDP and a scheduler see Sect. 2 of this chapter). The behavior of $\mathcal{M}$ under $\mathcal{U}$ can be formalized by an infinite-state discrete Markov chain $\mathcal{M}_{\mathcal{U}} = (\mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}}, \mathsf{p}, \mu)$, where

$$\mathsf{p}(\pi, \pi') = \mathcal{U}(\pi)(\alpha) \cdot \delta\big(\mathsf{last}(\pi), \alpha, \mathsf{last}(\pi')\big),$$

for $\pi, \pi' \in \mathsf{Path}_{\mathsf{fin}}^{\mathcal{M}}$ with $|\pi'| = |\pi| + 1$, $\pi'\uparrow^{|\pi|} = \pi$ and α is the last action on the path π', i.e.,

$$\pi \xrightarrow{\alpha} \mathsf{last}(\pi') \;=\; \pi'.$$

As the states of $\mathcal{M}_{\mathcal{U}}$ are finite paths of $\mathcal{M}$, this notation is somewhat inconvenient. Consider $\Omega = (\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}_{\mathcal{U}}}, \Delta^{\mathcal{M}_{\mathcal{U}}})$ and $\Omega' = (\mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}, \Delta^{\mathcal{M}})$, where $\Delta^{\mathcal{M}_{\mathcal{U}}}$ (resp. $\Delta^{\mathcal{M}}$) is the σ-algebra generated by the empty set and the set of basic cylinders over $\mathcal{M}_{\mathcal{U}}$ (resp. $\mathcal{M}$). We define

$$\mathsf{f} : \mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}_{\mathcal{U}}} \to \mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}}$$

as $\mathsf{f}(\pi_0 \xrightarrow{\alpha_1} \pi_1 \xrightarrow{\alpha_2} \cdots) = \mathsf{last}(\pi_0) \xrightarrow{\alpha_1} \mathsf{last}(\pi_1) \xrightarrow{\alpha_2} \cdots$ (note that the π_i's are finite paths of $\mathcal{M}$). Then f is a measurable function and we define the following probability measure on $\Delta^{\mathcal{M}}$:

$$\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(A') = \mathsf{Pr}^{\mathcal{M}_{\mathcal{U}}}\big(\mathsf{f}^{-1}(A')\big), \quad \text{for } A' \in \Delta^{\mathcal{M}}.$$

Then given a scheduler $\mathcal{U}$ for $\mathcal{M}$, the probability measure $\mathsf{Pr}^{\mathcal{M},\mathcal{U}}$ formalizes the behavior of $\mathcal{M}$ under $\mathcal{U}$, where we have the convenience to talk about measures of sets of infinite paths of $\mathcal{M}$. As for discrete Markov chains, given a state $s \in S$, we denote by $\mathsf{Pr}_s^{\mathcal{M},\mathcal{U}}$ the probability measure that is obtained if $\mathcal{M}$ is equipped with the starting distribution μ_s. For a detailed discussion on MDPs, see, e.g., [80].

We also fix the following notation for convenience. Given an MDP $\mathcal{M}$, a scheduler $\mathcal{U}$, and a path property E, we will write

$$\mathsf{Pr}^{\mathcal{M},\mathcal{U}}(E) = \mathsf{Pr}^{\mathcal{M},\mathcal{U}}\big(\{\pi \in \mathsf{Path}_{\mathsf{inf}}^{\mathcal{M}} \mid \pi \text{ satisfies } E\}\big)$$

for the probability that the property E holds in $\mathcal{M}$ under the scheduler $\mathcal{U}$.

References

1. H. Aljazzar, H. Hermanns, and S. Leue. Counterexamples for timed probabilistic reachability. In *Proceedings of the 3rd International Workshop on Formal Modeling and Analysis of Timed Systems (FORMATS'05)*, volume 3829 of *Lecture Notes in Computer Science*, pages 177–195. Springer, Berlin, 2005.
2. C. Baier, N. Bertrand, and M. Grösser. On decision problems for probabilistic Büchi automata. In *Proceedings of the 11th International Conference on Foundations of Software Science and Computation Structures (FOSSACS'08)*, volume 4962 of *Lecture Notes in Computer Science*, pages 287–301. Springer, Berlin, 2008.
3. C. Baier and F. Ciesinski. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *Proceedings of the 3rd International Conference on the Quantitative Evaluation of SysTems (QEST'06)*, pages 131–132. IEEE Computer Society Press, Los Alamitos, 2006.
4. C. Baier, F. Ciesinski, and M. Grösser. ProbMeLa: A modeling language for communicating probabilistic systems. In *Proceedings of the 2nd ACM–IEEE International Conference on Formal Methods and Models for Codesign (MEMOCODE'04)*, pages 57–66. IEEE Computer Society Press, Los Alamitos, 2006.
5. C. Baier, F. Ciesinski, M. Grösser, and J. Klein. Reduction techniques for model checking Markov decision processes. In *Proceedings of the 5th International Conference on Quantitative Evaluation of SysTems (QEST'08)*, pages 45–54. IEEE Computer Society Press, Los Alamitos, 2008.

6. C. Baier, E. Clarke, V. Hartonas-Garmhausen, M. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *Proceedings of the 24th International Colloquium on Automata, Languages and Programming (ICALP'97)*, volume 1256 of *Lecture Notes in Computer Science*, pages 430–440. Springer, Berlin, 1997.
7. C. Baier, P. d'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. In *Proceedings of the 3rd Workshop on Quantitative Aspects of Programming Languages (QAPL'05)*, volume 153(2) of *Electronic Notes in Theoretical Computer Science*, pages 97–116. Springer, Berlin, 2006.
8. C. Baier, B. Engelen, and M. Majster-Cederbaum. Deciding bisimularity and similarity for probabilistic processes. *Journal of Computer and System Sciences*, 60:187–231, 2000.
9. C. Baier, M. Größer, and F. Ciesinski. Partial order reduction for probabilistic systems. In *Proceedings of the 1st International Conference on Quantitative Evaluation of SysTems (QEST'04)*, pages 230–239. IEEE Computer Society Press, Los Alamitos, 2004.
10. C. Baier, B. Haverkort, H. Hermanns, and J.P. Katoen. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theoretical Computer Science*, 345(1):2–26, 2005.
11. C. Baier and J.-P. Katoen. *Principles of Model Checking.* MIT Press, Cambridge, 2008.
12. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11(3):125–155, 1998.
13. H. Bauer. *Wahrscheinlichkeitstheorie und Grundzüge der Maßtheorie.* de Gruyter, Berlin, 1978.
14. R. Bellmann. A Markovian decision process. *Journal of Mathematics and Mechanics*, 6(4):679–684, 1957.
15. A. Bianco and L. De Alfaro. Model checking of probabilistic and nondeterministic systems. In *Proceedings of the 15th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'95)*, volume 1026 of *Lecture Notes in Computer Science*, pages 499–513. Springer, Berlin, 1995.
16. M. Bozga and O. Maler. On the representation of probabilities over structured domains. In *Proceedings of the 11th International Conference on Computer Aided Verification (CAV'99)*, volume 1633 of *Lecture Notes in Computer Science*, pages 261–273. Springer, Berlin, 1999.
17. J. Burch, E. Clarke, K. McMillan, D. Dill, and L. Hwang. Symbolic model checking: 10^{20} states and beyond. *Information and Computation*, 98(2):142–170, 1992.
18. S. Cattani and R. Segala. Decision algorithms for probabilistic bisimulation. In *Proceedings of the 13th International Conference on Concurrency Theory (CONCUR'02)*, volume 2421 of *Lecture Notes in Computer Science*, pages 371–385. Springer, Berlin, 2002.

19. K. Chatterjee. Stochastic ω-regular games. PhD thesis, University of California at Berkeley, 2007
20. K. Chatterjee, L. de Alfaro, and T. Henzinger. The complexity of stochastic Streett and Rabin games. In *Proceedings of the 32nd International Colloquium on Automata, Languages and Programming (ICALP'05)*, volume 3580 of *Lecture Notes in Computer Science*, pages 878–890. Springer, Berlin, 2005.
21. E. Clarke, O. Grumberg, and D. Peled. *Model Checking.* MIT Press, Cambridge, 1999.
22. C. Courcoubetis and M. Yannakakis. Markov decision processes and regular events (extended abstract). In *Proceedings of the 17th International Colloquium on Automata, Languages and Programming (ICALP'90)*, volume 443 of *Lecture Notes in Computer Science*, pages 336–349. Springer, Berlin, 1990.
23. C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *Journal of the ACM*, 42(4):857–907, 1995.
24. P.R. d'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reduction and refinement strategies for probabilistic analysis. In *Proceedings of the Joint International Workshop on Process Algebra and Performance Modeling and Probabilistic Methods in Verification (PAPM–PROBMIV'02)*, volume 2399 of *Lecture Notes in Computer Science*, pages 57–76. Springer, Berlin, 2002.
25. P.R. d'Argenio, B. Jeannet, H. Jensen, and K. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *Proceedings of the 1st Joint International Workshop on Process Algebra and Performance Modeling and Probabilistic Methods in Verification (PAPM–PROBMIV'01)*, volume 2165 of *Lecture Notes in Computer Science*, pages 57–76. Springer, Berlin, 2001.
26. P.R. d'Argenio and P. Niebert. Partial order reduction on concurrent probabilistic programs. In *Proceedings of the 1st International Conference on Quantitative Evaluation of SysTems (QEST'04)*, pages 240–249. IEEE Computer Society Press, Los Alamitos, 2004.
27. L. de Alfaro. Formal verification of probabilistic systems. PhD thesis, Stanford University, 1997
28. L. de Alfaro. Stochastic transition systems. In *Proceedings of the 9th International Conference on Concurrency Theory (CONCUR'98)*, volume 1466 of *Lecture Notes in Computer Science*, pages 423–438. Springer, Berlin, 1998.
29. L. de Alfaro. The verification of probabilistic systems under memoryless partial-information policies is hard. In *Proceedings of the 2nd International Workshop on Probabilistic Methods in Verification (ProbMiV'99)*, pages 19–32. Birmingham University, Research Report CSR-99-9, 1999.
30. L. de Alfaro and R. Majumdar. Quantitative solution of omega-regular games. *Journal of Computer and System Sciences*, 68:374–397, 2004.

31. M. Droste and P. Gastin. Weighted automata and weighted logics. In this *Handbook*. Chapter 5. Springer, Berlin, 2009.
32. M. Droste and D. Kuske. Skew and infinitary formal power series. *Theoretical Computer Science*, 366:199–227, 2006.
33. M. Droste and G. Rahonis. Weighted automata and weighted logics with discounting. In *Proceedings of the 12th International Conference on Implementation and Applications of Automata (CIAA'07)*, volume 4783 of *Lecture Notes in Computer Science*, pages 73–84. Springer, Berlin, 2007.
34. M. Droste, J. Sakarovitch, and H. Vogler. Weighted automata with discounting. *Information Processing Letters*, 108(1):23–28, 2008.
35. Z. Esik and W. Kuich. Finite automata. In this *Handbook*. Chapter 3. Springer, Berlin, 2009.
36. I. Fichtner, D. Kuske, and I. Meinecke. Traces, series-parallel posets, and pictures: a weighted study. In this *Handbook*. Chapter 10. Springer, Berlin, 2009.
37. S. Giro and P.R. d'Argenio. Quantitative model checking revisited: neither decidable nor approximable. In *Proceedings of the 5th International Conference on Formal Modelling and Analysis of Timed Systems (FORMATS'07)*, volume 4763 of *Lecture Notes in Computer Science*, pages 179–194. Springer, Berlin, 2007.
38. P. Godefroid. *Partial Order Methods for the Verification of Concurrent Systems: an Approach to the State Explosion Problem*, volume 1032 of *Lecture Notes in Computer Science*. Springer, Berlin, 1996.
39. P. Godefroid, D. Peled, and M. Staskauskas. Using partial-order methods in the formal validation of industrial concurrent programs. In *Proceedings of the International Symposium on Software Testing and Analysis (ISSTA'96)*, pages 261–269. ACM, New York, 1996.
40. E. Grädel, W. Thomas, and T. Wilke, editors. *Outcome of the 2001 Dagstuhl Seminar on Automata, Logics, and Infinite Games: A Guide to Current Research*, volume 2500 of *Lecture Notes in Computer Science*. Springer, Berlin, 2002.
41. M. Größer and C. Baier. Partial order reduction for Markov decision processes: A survey. In *Proceedings of the 4th International Symposium on Formal Methods for Components and Objects (FMCO'05)*, volume 4111 of *Lecture Notes in Computer Science*, pages 408–427. Springer, Berlin, 2006.
42. M. Größer, G. Norman, C. Baier, F. Ciesinski, M. Kwiatkowska, and D. Parker. On reduction criteria for probabilistic reward models. In *Proceedings of the 26th Conference on Foundations of Software Technology and Theoretical Computer Science (FSTTCS'06)*, volume 4337 of *Lecture Notes in Computer Science*, pages 309–320. Springer, Berlin, 2006.
43. M. Größer. Reduction methods for probabilistic model checking. PhD thesis, Technische Universität, Dresden, 2008
44. R. Gupta, S. Smolka, and S. Bhaskar. On randomization in sequential and distributed algorithms. *ACM Computing Surveys*, 26(1):7–86, 1994.

45. G. Hachtel, E. Macii, A. Pardo, and F. Somenzi, Probabilistic analysis of large finite state machines. In *Proceedings of the 31st Design Automation Conference (DAC'94)*, pages 270–275. ACM, New York, 1994.
46. T. Han and J.-P. Katoen. Counterexamples in probabilistic model checking. In *Proceedings of the 13th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS'07)*, volume 4424 of *Lecture Notes in Computer Science*, pages 60–75. Springer, Berlin, 2007.
47. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6(5):512–535, 1994.
48. V. Hartonas-Garmhausen, S. Campos, and E. Clarke. Probverus: Probabilistic symbolic model checking. In *Proceedings of the 5th International Workshop on Formal Methods for Real-Time and Probabilistic Systems (ARTS'99)*, volume 1601 of *Lecture Notes in Computer Science*, pages 96–110. Springer, Berlin, 1999.
49. H. Hermanns, M. Kwiatkowska, G. Norman, D. Parker, and M. Siegle. On the use of MTBDDs for performability analysis and verification of stochastic systems. *The Journal of Logic and Algebraic Programming: Special Issue on Probabilistic Techniques for the Design and Analysis of Systems*, 56:23–67, 2003.
50. H. Hermanns, B. Wachter, and L. Zhang. Probabilistic CEGAR. In *Proceedings of the 20th International Conference on Computer Aided Verification (CAV'08)*, volume 5123 of *Lecture Notes in Computer Science*, pages 162–175. Springer, Berlin, 2008.
51. G. Holzmann. *The SPIN Model Checker, Primer and Reference Manual.* Addison–Wesley, Reading, 2003.
52. G.J. Holzmann and D. Peled. An improvement in formal verification. In *Proceedings of the 7th International Conference on Formal Description Techniques (IFIP'94)*, pages 197–211. Chapman & Hall, London, 1995.
53. F. Horn. Random games. PhD thesis, RWTH Aachen and Université Paris 7, 2008
54. R. Howard. *Dynamic Programming and Markov Processes.* MIT Press, Cambridge, 1960.
55. T. Huynh and L. Tian. On some equivalence relations for probabilistic processes. *Fundamenta Informaticae*, 17:211–234, 1992.
56. B. Jeannet, P.R. d'Argenio, and K.G. Larsen. RAPTURE: A tool for verifying Markov decision processes. In *Proceedings of the International Conference on Concurrency Theory (CONCUR'02): Tools Day*, Technical Report, Faculty of Informatics, Masaryk University Brno, 2002.
57. M. Jurdzinski, F. Laroussinie, and J. Sproston. Model checking probabilistic timed automata with one or two clocks. *Logical Methods in Computer Science*, 4(3-1):4–20, 2008.
58. J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for continuous-time Markov chains. In *Proceedings of the 19th International Conference on Computer Aided Verification (CAV'07)*, vol-

ume 4590 of *Lecture Notes in Computer Science*, pages 316–329. Springer, Berlin, 2007.

59. J. Kemeny and J. Snell. *Denumerable Markov Chains*. Van Nostrand, Princeton, 1976.
60. M. Kwiatkowska, G. Norman, and D. Parker. PRISM: Probabilistic symbolic model checker. In *Proceedings of the 12th International Conference on Modelling Tools and Techniques for Computer and Communication System Performance Evaluation (TOOLS'02)*, volume 2324 of *Lecture Notes in Computer Science*, pages 113–140. Springer, Berlin, 2002.
61. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.
62. M. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *Proceedings of the 18th International Conference on Computer Aided Verification (CAV'06)*, volume 4144 of *Lecture Notes in Computer Science*, pages 238–248. Springer, Berlin, 2008.
63. M. Kwiatkowska, G. Norman, D. Parker, and J. Sproston. Verification of real-time probabilistic systems. In *Modeling and Verification of Real-Time Systems: Formalisms and Software Tools*, pages 249–288. Wiley, New York, 2008
64. M. Kwiatkowska, G. Norman, R. Segala, and J. Sproston. Automatic verification of real-time systems with discrete probability distributions. *Theoretical Computer Science*, 282:101–150, 2002.
65. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, 1991.
66. D. Lehmann and M.O. Rabin. On the advantage of free choice: A symmetric and fully distributed solution to the dining philosophers problem (extended abstract). In *Proceedings of the 8th Annual ACM Symposium on Principles of Programming Languages (POPL'81)*, pages 133–138. ACM, New York, 1981.
67. W. Lovejoy. A survey of algorithmic methods for partially observable Markov decision processes. *Annals of Operations Research*, 28(1):47–65, 1991.
68. N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, San Francisco, 1996.
69. A. Miner and D. Parker. Symbolic representations and analysis of large probabilistic systems. In *Proceedings of the GI/Dagstuhl Research Seminar on Validation of Stochastic Systems*, volume 2925 of *Lecture Notes in Computer Science*. Springer, Berlin, 2003.
70. G. Monahan. A survey of partially observable Markov decision processes: Theory, models, and algorithms. *Management Science*, 28(1):1–16, 1982.
71. C. Papadimitriou and J. Tsitsiklis. The complexity of Markov decision processes. *Mathematics of Operations Research*, 12(3):441–450, 1987.

72. D. Parker. Implementation of symbolic model checking for probabilistic systems. PhD thesis, University of Birmingham, 2002.
73. A. Paz. *Introduction to Probabilistic Automata.* Academic Press, San Diego, 1971.
74. D. Peled. All from one, one for all: On model checking using representatives. In *Proceedings of the 5th International Conference on Computer-Aided Verification (CAV'93)*, volume 697 of *Lecture Notes in Computer Science*, pages 409–423. Springer, Berlin, 1993.
75. D. Peled. Partial order reduction: Linear and branching time logics and process algebras. In *Proceedings of the DIMACS Workshop on Partial Order Methods in Verification*, volume 29(10) of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 233–257. American Mathematical Society, Providence, 1997
76. D. Peled, V. Pratt, and G. Holzmann, editors. *Proceedings of the DIMACS Workshop on Partial Order Methods in Verification*, volume 29(10) of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science.* American Mathematical Society, Providence, 1997.
77. A. Philippou, I. Lee, and O. Sokolsky. Weak bisimulation for probabilistic systems. In *Proceedings of the 11th International Conference on Concurrency Theory (CONCUR'00)*, volume 1877 of *Lecture Notes in Computer Science*, pages 334–349. Springer, Berlin, 2000.
78. A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th Symposium on the Foundations of Computer Science (FOCS'77)*, pages 46–57. IEEE Computer Society Press, Los Alamitos, 1977.
79. A. Pnueli and L. Zuck. Probabilistic verification by tableaux. In *Proceedings of the Symposium on Logic in Computer Science (LICS'86)*, pages 322–331. IEEE Computer Society Press, Los Alamitos, 1986.
80. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming.* Wiley, New York, 1994.
81. M.O. Rabin. Probabilistic automata. *Information and Control*, 6(3):230–245, 1963.
82. J.J.M.M. Rutten, M. Kwiatkowska, G. Norman, and D. Parker. *Mathematical Techniques for Analyzing Concurrent and Probabilistic Systems*, volume 23 of *CRM Monograph Series.* American Mathematical Society, Providence, 2004.
83. A. Schrijver. *Combinatorial Optimization: Polyhedra and Efficiency.* Springer, Berlin, 2003.
84. E.J. Sondik. The optimal control of partially observable Markov processes. PhD thesis, Stanford University, 1971.
85. W. Thomas. Automata on infinite objects. In *Handbook of Theoretical Computer Science, volume B*, Chapter 4, pages 133–191. Elsevier, Amsterdam, 1990.
86. A. Valmari. A stubborn attack on state explosion. *Formal Methods in System Design*, 1:297–322, 1992.

87. A. Valmari. State of the art report: Stubborn sets. *Petri-Net Newsletters*, 46:6–14, 1994.
88. A. Valmari. Stubborn set methods for process algebras. In *Proceedings of the DIMACS Workshop on Partial Order Methods in Verification*, volume 29(10) of *DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, pages 213–231. American Mathematical Society, Providence, 1997.
89. R. van Glabbeek, S. Smolka, B. Steffen, and C. Tofts. Reactive, generative, and stratified models of probabilistic processes. In *Proceedings of the 5th IEEE Symposium on Logic in Computer Science (LICS'90)*, pages 130–141. IEEE Computer Society Press, Los Alamitos, 1990.
90. M.Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *Proceedings of the 26th Symposium on Foundations of Computer Science (FOCS'85)*, pages 327–338. IEEE Computer Society Press, Los Alamitos, 1985.
91. M.Y. Vardi and P. Wolper. An automata-theoretic approach to automatic program verification. In *Proceedings of the Symposium on Logic in Computer Science (LICS'86)*, pages 332–344. IEEE Computer Society Press, Los Alamitos, 1986.