The Probabilistic Model Checking Landscape *

Joost-Pieter Katoen
RWTH Aachen University, Germany and University of Twente, The Netherlands
katoen@cs.rwth-aachen.de

Abstract

Randomization is a key element in sequential and distributed computing. Reasoning about randomized algorithms is highly non-trivial. In the 1980s, this initiated first proof methods, logics, and model-checking algorithms. The field of probabilistic verification has developed considerably since then. This paper surveys the algorithmic verification of probabilistic models, in particular probabilistic model checking. We provide an informal account of the main models, the underlying algorithms, applications from reliability and dependability analysis—and beyond—and describe recent developments towards automated parameter synthesis.

Categories and Subject Descriptors B.8 [*Performance and Reliability*]: General; F.3.1 [*Logics and Meaning of Programs*]: Mechanical Verification.; G.3 [*Probability and Statistics*]: Markov Processes.

Keywords abstraction, applications, fault trees, Markov chains, Markov decision processes, model checking, parameter synthesis, probabilistic logics

1. Introduction

Soon after the birth of model checking in 1981, the first papers on automated verification techniques for probabilistic models appeared [95, 156, 171]. They focused on almost-sure properties for probabilistic programs—does a program satisfy a property with probability one? Seminal works by Courcoubeitis & Yannakakis [59, 60] and Hansson & Jonsson [94] opened the possibility for determining the probability with which an ω-regular and probabilistic CTL formula hold, respectively. These works all focused on discrete-time Markov models. The algorithms by Baier *et al.* [14], operationalizing the decidability result of Aziz *et al.* [10], provided the starting point of model checking continuous-time Markov chains at the end of the 1990s. Powerful tool support [129] together with unremitting algorithmic improvements and the use of succinct data structures led to an impulse to the field. It is fair to say that probabilistic model checking extends and complements long-standing analysis techniques for Markov processes. Nowadays, probabilistic model checking is a well-established (and still growing) branch of computer-aided verification. This is perhaps mainly due to its wide variety of applications, most notably in dependability, reliability, and performance analysis as well as systems biology. Probabilistic model checking seems to have a bright future. New application areas such as robotics, energy-aware computing, and probabilistic programming—a dejà vu since Kozen's seminal works [120, 121]—are stimulating further advances. Richer models and more expressive properties are being considered. Clarke (2008) mentions probabilistic model checking as one of the main challenges "for the future" [57]. Recently (2015), Alur, Henzinger and Vardi [6] state: "A promising new direction in formal methods research these days is the development of probabilistic models, with associated tools for quantitative evaluation of system performance along with correctness."

This paper surveys the main probabilistic models, algorithms, abstraction techniques, applications, and gives a brief account of a promising new direction: parameter synthesis. The explanations are kept mostly informal [1].

2. The Probabilistic Models Landscape

Markov chains. Discrete-time Markov chains [113, 114] (DTMCs, for short) are the simplest possible probabilistic models. A DTMC is a Kripke structure in which all transitions are equipped with a probability. For each state, the sum of the outgoing transition probabilities equals one. The DTMC in Fig. 1 models the randomized algorithm by Knuth and Yao [118]. It aims at mimicking a six-sided die by means of a fair coin. Depending on the outcome of the coin flip in s_0, the next state is either s_1 (on "tails") or s_2 (on "heads"). We denote $\mathbf{P}(s_0, s_1){=}1/2$ and $\mathbf{P}(s_0, s_2){=}1/2$. The coin is flipped until the outcome is between ⚀ and ⚅. (For simplicity, the self-loops at these states are omitted.) Due to the cycles, it is not evident that repeatedly tossing a coin indeed yields an adequate model of a six-sided die. Is the probability of each of the outcomes indeed $1/6$? In order to answer this question, we consider infinite *paths* through the Markov chain. Thus, e.g., $s_0\, s_2\, s_5\, ⚃^\omega$ is a path. To define the probability of e.g., the set of paths that finally end up in state ⚃, one considers *cylinder sets*, sets of paths that all share a common prefix [171]. The cylinder set of $s_0 \ldots s_n$ is the set of paths that have $s_0 \ldots s_n$ as prefix. For example, the cylinder set of $s_0\, s_2\, s_6$ is the set of infinite paths consisting of $s_0\, s_2\, s_6\, ⚄^\omega$ and $s_0\, s_2\, s_6\, ⚅^\omega$. The probability of a cylinder set C of $s_0 \ldots s_n$ is defined as $\mathbf{P}(s_0, s_1) \cdot \ldots \cdot \mathbf{P}(s_{n-1}, s_n)$ if $n > 0$; for $n{=}0$, it is one. (Technically speaking, the σ-algebra on infinite paths in a DTMC is generated using cylinder sets.) Any set of paths that can be written as the complement and/or countable union of cylinder sets is now measurable.

* This work was supported by the Excellence Initiative of the German federal and state government, the CDZ project CAP (GZ 1023), the STW-ProRail partnership program ExploRail under the project ArRangeer (12238) and the ESA-funded project CATSY.

LICS '16, July 5–8, 2016, New York, NY, USA.

http://dx.doi.org/10.1145/http://dx.doi.org/10.1145/2933575.2934574

[1] More extensive and detailed treatments of verifying Markov decision processes are given in [11, 25, 79, 85]. Introductions to discrete-time and continuous-time Markov chain model checking are given in [104, 128].

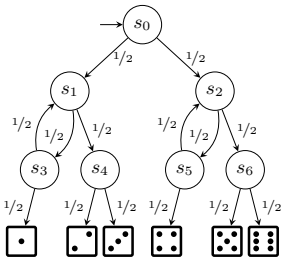


Figure 1: A Markov chain for Knuth-Yao's six-sided die.

Continuous-time Markov Chains (CTMCs). CTMCs [33, 152] extend DTMCs over state space S with a function $r : S \to \mathbb{R}_{>0}$ assigning to each state s the rate of a negative exponential distribution governing the residence time in s. Thus, the probability to reside in state s maximally d time units is $1 - e^{-r(s)\cdot d}$. Phrased alternatively, the average residence time of state s is $1/r(s)$. If $r(s) = 2{\cdot}r(s')$, the residence time in s' is—on average—half the residence time in state s. An alternative view of a CTMC is to combine the transition probabilities given by function $\mathbf{P}$ and the rate function r by: $\mathbf{R}(s,s') = \mathbf{P}(s,s'){\cdot}r(s)$, the transition rate from s to s'. Both perspectives are equivalent. In the sequel, we will use (and change between) both equivalent definitions whenever convenient. The operational interpretation of a CTMC is as follows. On entering a state s, the residence time is determined by an exponential distribution with rate $r(s)$. On leaving the state s, the probability to move to state s' equals $\mathbf{P}(s,s')$. The probability to move from s to s' in the interval $[0,t]$ is:

$$\frac{\mathbf{R}(s,s')}{r(s)} \cdot \left(1 - e^{-r(s)\cdot t}\right).$$

Whereas in a DTMC, a path is simply an infinite sequence of states, in a CTMC we need to keep track on how long we stay in each state. A (timed) path in a CTMC is thus an infinite alternating sequence of states and time delays. A probability measure over infinite timed paths is defined—as in the discrete-time setting—by cylinder sets. This is slightly more complex than before, as the residence times in the individual states along a path need to be considered as well. We do so by considering *intervals* in $\mathbb{R}_{\geqslant 0}$. Formally, the (interval) cylinder set of an (interval) path fragment $s_0\, I_0\, s_1\, I_1\, s_2\, I_2 s_3$, where I_0, I_1, and I_2 are intervals, is the set of timed paths in the CTMC that all have a prefix $s_0\, t_0\, s_1\, t_1\, s_2\, t_2\, s_3$ where $t_0 \in I_0$, $t_1 \in I_1$, and $t_2 \in I_2$. The probability of a cylinder set of $s_0\, I_0 \ldots I_{n-1} s_n$ is:

$$\prod_{j=1}^{n} \mathbf{P}(s_{j-1}, s_j) \cdot \underbrace{\int_{I_{j-1}} r(s_{j-1}){\cdot}e^{-r(s_{j-1})\cdot x}\, dx}_{\text{probability to leave } s_{j-1} \text{ in interval } I_{j-1}}$$

Any set of (timed) paths that can be written as the complement and/or countable union of (interval) cylinder sets is now measurable. Solving the integrals results in

$$\int_{I_j} r(s_j){\cdot}e^{-r(s_j)\cdot x}\, dx \;=\; e^{-r(s_j)\cdot \inf I_j} - e^{-r(s_j)\cdot \sup I_j}$$

where $\inf I_j$ and $\sup I_j$ are the infimum and supremum of the I_j, respectively. *Zeno* paths occur in CTMCs. A Zeno path is a path in which the total time that elapses converges. In case $\sum_i t_i$ does not diverge, the timed path represents an "unrealistic" computation where infinitely many transitions are taken in a finite amount of time. An example Zeno path is $s_0\, 1/2\, s_1\, 1/4\, s_2\, 1/8\, s_3 \ldots$ converging to 1. In timed automata, such executions are typically excluded from the analysis. Zeno paths may occur in CTMCs but do not harm: their probability mass is zero [14].

	Discrete time	Continuous time
Deterministic	discrete-time Markov chain (DTMC)	continuous-time MC
Nondeterministic	Markov decision process (MDP)	CTMDP
Compositional	Segala's Probabilistic Automata (PA)	Markov Automata (MA)

Figure 2: Overview of Markov models

Markov decision processes. A *Markov decision process* (MDP, for short [101, 157]) extends a Markov chain with non-determinism. In a Markov chain state s, the next state is chosen according to the probability distribution $\mathcal{D}_s = \mathbf{P}(s,\cdot)$ over the states. However, in an MDP, a state may have several distributions. On reaching a state s in an MDP, non-deterministically a distribution $\mu \in \mathcal{D}(s)$ is selected. The next state is then determined according to μ. That is, state s' is selected with probability $\mu(s')$. It is assumed that $\mathcal{D}(s) \neq \varnothing$ for each state s. Every MDP for which $|\mathcal{D}(s)| = 1$ in every state s, is a Markov chain. Paths in an MDP are infinite alternating sequences of pairs of states and distributions: (s_i, μ_i) where $\mu_i \in \mathcal{D}(s_i)$ and $\mu_i(s_{i+1}) > 0$, for each i. A probability measure on such paths can be defined using the cylinder set construction, as for Markov chains, provided for each state s_i it is known which distribution μ_i has been selected. This is formalized by a *policy*—also referred to as scheduler or adversary—that in state s_i selects a distribution $\mu \in \mathcal{D}(s_i)$. (A policy can be viewed as oracle.) Several types of policies do exist. Two ingredients are relevant: on the basis of which information does a policy make a decision, and can it use randomisation to do so, or not. Positional policies decide solely on the current state s_i and not on the history, i.e., the prefix of the path until reaching s_i. Randomized positional policies select $\mu \in \mathcal{D}(s_i)$ with a certain probability. Deterministic ones select a fixed distribution from $\mathcal{D}(s_i)$. History-dependent policies base their decision on the prefix $s_0\mu_0 \ldots \mu_{i-1} s_i$. A policy imposed on an MDP yields an (possibly infinite-state) MC.

Continuous-time MDPs (CTMDPs, for short [90]), are MDPs in which the state residence time is—as for CTMCs—governed by a negative exponential distribution. The rate of this exponential distribution depends on the discrete probability distribution that is used to determine the next state. Accordingly, the average residence time in state s under taking distribution μ is given by $1/r(s,\mu)$. Rate $r(s,\mu)$ thus determines the random residence time in state s provided distribution μ is selected in s. Paths in CTMDPs are infinite sequence of triples (s_i, t_i, μ_i) where t_i denotes the residence time in state s_i given that distribution μ_i has been selected. Policies in CTMDPs cannot only decide on the basis of the states visited and the selected distributions so far, but may also exploit the elapsed time (in every state). This gives rise to uncountably many policies. It for instance, makes a difference whether a policy decides on entering a state (early) or on leaving a state (late). A categorisation of the class of policies for CTMDPs is given in [145].

Other probabilistic models. A summary of the four described models is provided in Fig. 2. This overview is (deliberately) incomplete. A plethora of other probabilistic models do exist; for overviews see [27, 96]. Let us mention a few that have been studied in the field of formal verification. Probabilistic extensions of timed automata exist [151]; they are known as probabilistic timed automata (PTA). Their edges are discrete probability distributions over states. PTA are finite symbolic representations of uncountable MDPs—as clock valuations are real values. Non-determinism is in-

herited from timed automata. Computing reachability probabilities in PTA is decidable via a region graph-like construction. Whereas in PTA clocks are deterministic, stochastic timed automata [30] (STA) provide a stochastic interpretation to clocks. In STA, unbounded clocks are interpreted as negative exponential distributions, whereas bounded clocks obey a uniform distribution. This model is no longer Markovian. Stochastic interpretations of TA are also used in statistical model checking [63].

PTA and STA are symbolic representations of (uncountable) infinite-state probabilistic models. Other (countable) infinite-state Markov models that have been subject to verification include probabilistic push-down automata [40], (the equally expressive) recursive Markov chains and recursive MDPs [80]. Such automata are natural means to model e.g., recursive probabilistic programs. The configuration graph of a probabilistic push-down automaton is a countable infinite-state DTMC (or MDP). Alternative infinite-state models include decisive Markov chains [2], probabilistic lossy channel systems [162], and probabilistic multi-counter automata—also known as probabilistic vector-addition systems with states (pVASS) [41]. The model of pVASS is strongly related to probabilistic Petri nets [124]. Finally, we mention probabilistic ω-automata [21], probabilistic models with continuous dynamics [1], stochastic games [48], and labelled Markov processes [153].

Rewards. All Markov models can be naturally extended with the notion of a cost, or dually, a gain. This can be done in two ways. Costs that are associated to transitions—the so-called *transition rewards*—are constant non-negative real values that are incurred on taking a transition. Thus, on moving from state s to state s' via distribution μ, a transition reward $\mathit{rew}(s,\mu,s')$ is earned. Similarly, a cost rate can be associated to states—the so-called *state rewards*. In continuous-time models, residing t time units in a state with cost rate $\mathit{rew}(s,\mu)$ leads to earning a reward $\mathit{rew}(s,\mu)\cdot t$. In discrete-time models, one typically only considers transition rewards. The cumulative reward of a finite path fragment $s_0\mu_0\ldots\mu_{k-1}s_k$ in an MDP is defined as the sum of all transition rewards, that is, $\sum_{i=0}^{k-1}\mathit{rew}(s_i,\mu_i,s_{i+1})$. In a CTMDP, additionally the state rewards come into the play. That is to say, the term $\sum_{i=0}^{k-1}\mathit{rew}(s_i,\mu_i)\cdot t_i$ is added on top of the cumulative transition reward. Evidently, in absence of transition rewards and all state rewards equal to one, the cumulative reward equals the elapsed time along a finite path fragment. This simple observation can be generalized: as first observed for CTMCs in [29], the role of time and rewards can often be swapped when the reward (continuous-time) Markov model is "rescaled". For the sake of simplicity, assume all transition rewards are zero. Consider a CTMDP with $\mathit{rew}(s,\mu)>0$ for all states s and distributions $\mu\in\mathcal{D}(s)$. The dual CTMDP results by adapting the exit rates and reward function such that the reward units in state s in the original CTMDP correspond to the time units in state s in its dual version, and vice versa. This is accomplished by the following scheme:

$$\mathbf{R}^*(s,\mu,s') \;=\; \frac{\mathbf{R}(s,\mu,s')}{\mathit{rew}(s,\mu)} \quad\text{and}\quad \mathit{rew}^*(s,\mu) \;=\; \frac{1}{\mathit{rew}(s,\mu)}.$$

where $\mathbf{R}^*$ and rew^* denote the transition rate and reward function in the transformed CTMDP, respectively. Intuitively, the transformation stretches the residence time in state s with a factor that is proportional to the reciprocal of its reward $\mathit{rew}(s,\mu)$ if $0<\mathit{rew}(s,\mu)<1$. The reward function is changed similarly. Thus, all transitions with $\mathit{rew}(s,\mu)<1$ are accelerated whereas those with $\mathit{rew}(s,\mu)>1$ are slowed down. One might interpret the residence of t time units in the transformed model as the earning of t reward in state s (under distribution μ) in the original model. Conversely, earning a reward r in s under distribution μ corresponds to residing r time units in s in the transformed model [19].

Compositional models. In order to cope with the magnitude and complexity of probabilistic models for realistic systems, *compositional variants* of Markov models are considered. These are mild extensions of Markov models that facilitate the parallel composition in a process-algebraic manner like in CCS and CSP. Segala's probabilistic automata [163] (PA, for short) [2] are like MDP where the distributions $\mathcal{D}(s)$ in a state s are labelled with actions. Rather than having distributions μ_1 through μ_k in $\mathcal{D}(s)$, we have $a_1,\ldots,a_k\in\mathcal{D}(s)$. It is not just all about naming though. We stipulate that in state s several equally labelled distributions may exist, thus $a_i=a_j=b$ is possible for $i\neq j$. This is not possible in an MDP. Let $\mathcal{P}\xrightarrow{a}\mu$ denote that (the initial state of) PA P with action a selects distribution μ; state u is the next state with probability $\mu(u)$. The actions are used to define a parallel-composition operator $||_A$ that is parametrized with a set A of actions. PA $\mathcal{P}||_A\mathcal{Q}$ denotes the parallel composition of the PAs $\mathcal{P}$ and $\mathcal{Q}$. The individual PAs $\mathcal{P}$ and $\mathcal{Q}$ perform (transitions labelled with) actions that are outside A autonomously. This happens in an interleaved manner. Transitions labelled with actions in A need to be taken synchronously. Thus, $\mathcal{P}$ and $\mathcal{Q}$ must evolve simultaneously when taking an action in A. The resulting probability distribution of taking $a\in A$ is simply the product of the distributions of $\mathcal{P}$ taking a and $\mathcal{Q}$ taking a. There is one exception: the distinguished (internal) action $\tau\notin A$ can only be performed autonomously. The transition relation is thus defined as follows:

$$\frac{\mathcal{P}\xrightarrow{a}\mu}{\mathcal{P}\,||_A\,\mathcal{Q}\xrightarrow{a}\nu}\ \text{if } a\notin A \text{ s.t. } \nu(\mathcal{P}'||_A\mathcal{Q}')=\mu(\mathcal{P}')\cdot\delta_{\mathcal{Q}}$$

$$\frac{\mathcal{P}\xrightarrow{a}\mu \text{ and } \mathcal{Q}\xrightarrow{a}\mu'}{\mathcal{P}\,||_A\,\mathcal{Q}\xrightarrow{a}\nu}\ \text{if } a\in A \text{ s.t. } \nu(\mathcal{P}'||_A\mathcal{Q}')=\mu(\mathcal{P}')\cdot\mu'(\mathcal{Q}')$$

where $\delta_{\mathcal{Q}}$ denotes the Dirac distribution for PA $\mathcal{Q}$, i.e., $\delta_{\mathcal{Q}}(\mathcal{Q})=1$ and $\delta_{\mathcal{Q}}(\mathcal{R})=0$ for $\mathcal{Q}\neq\mathcal{R}$. The first inference rule corresponds to PA $\mathcal{P}$ performing action a autonomously. We have omitted the symmetric case of the first inference rule (in which $\mathcal{Q}$ autonomously performs an action) for the sake of brevity. The second inference rule covers the case in which a synchronisation between $\mathcal{P}$ and $\mathcal{Q}$ takes place. The aim is now to model a complex system by means of several components:

$$((\mathcal{P}_1\,||_{A_1}\,\mathcal{P}_2)\,||_{A_2}\,\ldots\mathcal{P}_{N-1})\;||_{A_{N-1}}\,\mathcal{P}_N$$

and then turn all actions into τ-actions, refraining from the action names. (This can be done using appropriate operators that are omitted here.) Put shortly, the actions are used for synchronization purposes only; once the entire model is defined, actions become irrelevant. The resulting model is an MDP, and all concepts and analysis algorithms for MDPs readily apply. Variants of parallel composition of PA can be found in [55, 175].

Markov automata [70, 76] (MA, for short) are the continuous-time variant of Segala's PA. Whereas PA are a mild extension to MDPs, MA are a variant of CTMDPs [3]. (Strictly speaking, Hermanns' interactive Markov chains [98] play this role; MA extend this model with probabilistic branching as in PA.) The key idea here is to separate the passage of time (denoted $\bullet\!\!\longrightarrow$) from taking an action-labelled transition (denoted $\longrightarrow$). Technically speaking, MA are Segala's PA with one extra feature—transitions that are labelled with rates of exponential distributions. One may safely assume that between any pair of states at most one such transition oc-

[2] These should not be confused by the probabilistic automata introduced by Paz [155] and Rabin [159]; these are a probabilistic extension of classical finite-state automata (with accept states).

[3] The distinction between CTMDPs and MA is however more subtle than that between PA and MDPs. This is e.g., reflected in the class of policies that attain extremal reachability probabilities.

curs as $s \bullet\!\!\xrightarrow{\lambda} s'$ and $s \bullet\!\!\xrightarrow{\lambda'} s'$ are equivalent to $s \bullet\!\!\xrightarrow{\lambda+\lambda'} s'$. MA thus have a transition relation between states and distributions over states, where the transition is either labelled by an action or by a positive real number. In the former case the target probability distribution is explicitly given, in the latter case it is implicitly given. This can be seen as follows. If $s \bullet\!\!\xrightarrow{3} u$ and $s \bullet\!\!\xrightarrow{8} v$, then this can be viewed as $s \xrightarrow{3+8} \mu$ where $\mu(u) = 3/11$ and $\mu(v) = 8/11$. The semantics of an MA is defined as follows. For states with only outgoing action-labelled transitions, the interpretation is as for PA. For states with only rate-labelled transitions, the MA evolves as a CTMC. For the moment, we refrain from treating states that have both outgoing action- and rate-labelled transitions. We get back to this issue later on. Actions may be subject to synchronisation with other MA, and thus may be subject to a delay. This happens if an MA needs to synchronise on action a, say, while its synchronization partner can only perform the a-transition after a rate-transition (a delay). Action-transitions labelled with τ—the distinguished internal action that cannot be subject to synchronization—are special. That is to say, they have priority over rate-labelled transitions. More precisely, if in a state both an τ-labelled transition (and no otherwise labelled action-transitions) and (one or more) rate-labelled transition are emanating, priority is given to the τ-transition. This is called the *maximal progress assumption*. The rationale behind this assumption is that internal (i.e., τ-labelled) transitions are not subject to interaction and thus can happen immediately, whereas the probability of a Markovian transition to immediately happen is zero. Thus, $s \bullet\!\!\xrightarrow{\lambda} s'$ almost never fires instantaneously. Note that the maximal progress assumption does not apply in case $s \bullet\!\!\xrightarrow{\lambda} s'$ and $s \xrightarrow{\alpha} \mu$ with $\alpha \neq \tau$, as α-transitions—unlike τ-transitions—can be used for synchronisation and thus be subject to a delay. In this case, the transition $s \bullet\!\!\xrightarrow{\lambda} s'$ may happen with positive probability. Note that also in this case, the maximal progress assumption applies: if $s \xrightarrow{\tau} \mu$ and s has several Markovian transitions, only the τ-transition can occur and no delay occurs in s. MA possess the same compositional features as Segala's PA. Parallel composition for MA is very similar as for PA. In fact, action transitions are defined as for PA. For real-valued transitions we have:

$$\frac{\mathcal{M} \xrightarrow{\lambda} \mu \text{ and not } \mathcal{M} \xrightarrow{\tau}}{\mathcal{M} \,||_A\, \mathcal{N} \xrightarrow{\lambda} \nu} \text{ where } \nu(\mathcal{M}'||_A\mathcal{N}') = \mu(\mathcal{M}')\cdot\delta_{\mathcal{N}'}$$

where we omit the symmetric case, for brevity's sake. As for PA, a complex system is modelled by means of several components:

$$((\mathcal{M}_1 \,||_{A_1}\, \mathcal{M}_2) \,||_{A_2}\, \ldots \mathcal{M}_{N-1}) \,||_{A_{N-1}}\, \mathcal{M}_N$$

and then all actions are turned into τ-actions, refraining from the action names. As before, the actions are used for synchronization purposes only; once the entire model is defined, actions become irrelevant. The resulting model is an MA with the property that it has no states with both outgoing action- and rate-labelled transitions. This is due to the fact that all actions are turned into τ-transitions and the maximal progress assumption. Along similar lines, compositional variants of generalized semi-Markov chains (GSMCs) have been considered [62].

Applications of compositional models. Are the compositional models such as PA and MA of pure theoretical interest? No. Various high-level description languages have been provided a semantics in terms of Segala's PA or MA. PA have been used as operational model for probabilistic process algebras, the PIOA language, and have served to reason about randomized distributed algorithms [148]. Segala [163] has studied several behavioural relations on PA such as (weak and strong) bisimulation and simulation pre-orders, as well as trace inclusions. These relations form the basis for obtaining abstractions of PA, i.e., smaller models amenable to further analysis. MA have been used to provide a semantics of Markovian process algebras à la PEPA [100], dynamic fault trees [36], data-flow languages like SDF [105], and the domain-specific language AADL [38] (see later on). The operational model of a guarded command programming language with rates for programming multi-robot systems is very similar to MA [144]. Recently, it has been shown that MA are an adequate model to treat confusion in generalised stochastic Petri nets (GSPNs) [77, 138], a problem that has been open for several decades. Probabilistic I/O automata [175] are closely related to MA.

	Branching-time	Linear-time
Discrete time	probabilistic CTL	deterministic automata (safety and LTL)
Continuous time	probabilistic timed CTL	deterministic timed automata (MITL)

Figure 3: Overview of elementary properties

3. The (How to Check) Properties Landscape

We treat the major properties in probabilistic model checking.

Qualitative reachability. One of the elementary questions for the analysis of probabilistic models is whether a certain set of goal states can be reached almost surely, i.e., with probability one, or dually, with a positive probability. For set G of target states, let $\Diamond\, G$ denote the event to reach (some state in) G eventually. The event $\Diamond\, G$ is measurable as it can be characterised as the union of all cylinders of finite path fragments that end in a state in G and do not hit G before. For finite-state Markov chains, the question whether $\Pr(\Diamond G)=1$? is equivalent to whether all strongly fair paths from the initial state eventually reach G. Thus a standard graph search algorithm suffices [95]. The same applies to the question whether events such as $\Box\, G$, $\Diamond\Box\, G$, $\Box\Diamond\, G$, or ω-regular events such as $\Diamond\Box\, G_1 \wedge \Box\Diamond\, G_2$ almost surely hold. All these objectives can be reduced to reachability objectives, as each path in a finite Markov chain eventually ends in a terminal SCC, a strongly connect component that once entered cannot be left [171]. This yields, for instance, that $\Pr(s \models \Box\Diamond\, G) = 1$ is equivalent to whether $T \cap G \neq \varnothing$ for each terminal SCC T that is reachable from state s. Or, stated differently, it is equivalent to whether the CTL-formula $\forall\Box\exists\Diamond\, G$ holds in s. Note that a graph analysis for almost-sure reachability objectives in infinite-state Markov chains does not suffice. For example, whether the origin is almost surely visited in a one-sided, one-dimensional random walk depends on the transition probabilities. If the probability to walk to the right exceeds $1/2$, the origin is not almost surely visited; if however this probability is less than $1/2$, it does.

Quantitative reachability. As opposed to qualitative reachability, we are now interested in checking whether the probability to reach G exceeds a threshold like $2/3$. More precisely, we want to determine the probability of the set of paths in $\Diamond G$, given that we start in some state s. Assuming that the Markov chain at hand has finitely many states, we let variable $x_s = \Pr(s \models \Diamond G)$ for state s. The following recursive characterization will be helpful. If G is not reachable from s, then $x_s = 0$; if $s \in G$, then $x_s = 1$. For all other cases:

$$x_s = \underbrace{\sum_{t \notin G} \mathbf{P}(s,t) \cdot x_t}_{\text{reach } G \text{ via } t \notin G} + \underbrace{\sum_{u \in G} \mathbf{P}(s,u)}_{\text{reach } G \text{ in one step}}$$

This provides the basis to show that reachability probabilities are *unique* solutions of a linear equation system. Such linear equation

system is obtained in the following way. Let $S_?$ be the set of states not in G that can reach G. Let matrix $\mathbf{A} = \big(\mathbf{P}(s,t)\big)_{s,t\in S_?}$, the transition probabilities between the states in $S_?$. Let the vector $\underline{b} = \big(b_s\big)_{s\in S_?}$, the probabilities to reach some state G in a single step, i.e., $b_s = \sum_{u\in G} \mathbf{P}(s,u)$. Then, $\underline{x} = (x_s)_{s\in S_?}$ with $x_s = \Pr(s \models \Diamond G)$ is the unique solution of:

$$\underline{x} \;=\; \mathbf{A}\cdot\underline{x} \;+\; \underline{b} \quad \text{or in short form} \quad (\mathbf{I}-\mathbf{A})\cdot\underline{x} \;=\; \underline{b}$$

where $\mathbf{I}$ is the identity matrix of cardinality $|S_?|\cdot|S_?|$.[4] To conclude: *reachability probabilities can be obtained as the unique solution of a linear equation system*. Any technique to solve such system either exactly (e.g., Gaussian elimination) or iteratively (e.g., the Power method) can be used to obtain reachability probabilities.

Bounded reachability events put an upper bound on the number of transitions that are allowed to reach G. Let $\Diamond^{\leqslant k} G$ denote the set of paths that reach G in at most k steps. For example, the path $s_0\, s_2\, s_5\, \boxdot^{\omega}$ belongs to $\Diamond^{\leqslant 3}\boxdot$, but does not belong to $\Diamond^{\leqslant 2}\boxdot$. To compute $\Pr(\Diamond^{\leqslant k} G)$ in DTMC $\mathfrak{D}$, we make all states in G absorbing, i.e., replace all outgoing transitions from every state $s \in G$ by a self-loop with probability one. Denote the resulting Markov chain by $\mathfrak{D}[G]$. Once a path reaches a state in G within k steps, that path will still be in that state in G after exactly k steps:

$$\underbrace{\Pr(s \models \Diamond^{\leqslant k}\, G)}_{\text{reachability in } \mathfrak{D}} \;=\; \underbrace{\Pr(s \models \Diamond^{=k}\, G)}_{\text{reachability in } \mathfrak{D}[G]} \;=\; \underbrace{\underline{1}_s \cdot \mathbf{P}_G^k}_{\text{in } \mathfrak{D}[G]}$$

Here $\Diamond^{=k} G$ denotes the set of infinite paths that after exactly k steps reach G, and $\underline{1}_s$ is the characteristic vector for state s. The term $\underline{1}_s\cdot\mathbf{P}_G^k$ is the *transient state distribution* of $\mathfrak{D}[G]$ (when starting in state s) at epoch k.

Like for qualitative reachability, *probabilities for ω-regular objectives* in finite Markov chains can be obtained from reachability probabilities [59]. This again is based on the fact that infinite paths eventually end up in traversing a terminal SCC. This yields, for instance, that $\Pr(s \models \Box\Diamond G) = \Pr(s \models \Diamond U)$ where U is the collection of terminal SCCs T for which $T \cap G \neq \varnothing$. For arbitrary ω-regular properties, an automaton-based approach can be taken. Given a deterministic Rabin automaton (DRA) for the property at hand, we determine the product (denoted $\otimes$) of the Markov chain and the DRA. Recall that the acceptance condition of a DRA equals $\{\,(L_i, K_i) \mid 0 < i \leqslant m\,\}$ with L_i, K_i being sets of states of the DRA. This is thus a set of pairs, where each element of a pair is a set of states. A run of DRA is accepting if for some i the run visits all states in L_i only finitely often and visits some state in K_i infinitely often. A terminal SCC in $\mathfrak{D}\otimes\mathcal{A}$ is accepting if for some i if it contains no states from L_i and some state from K_i. The accepting terminal SCCs in the product of $\mathfrak{D}$ and DRA $\mathcal{A}$ thus correspond to the possible infinite runs in $\mathfrak{D}$ that are accepted by $\mathcal{A}$. It follows that the probability of an ω-regular property in a Markov chain $\mathfrak{D}$ equals the reachability probability of an accepting SCC in the product $\mathfrak{D}\otimes\mathcal{A}$. This provides an effective way to determine the probability with which an LTL formula holds.

To reason about events in MDPs such as reachability, nondeterminism is resolved by an oracle, called a policy (also called adversary of scheduler). A policy for an MDP is a function $\mathfrak{S}$ that for a given finite path fragment through the MDP yields a distribution to take next. Due to the presence of non-determinism, the probability $\Diamond G$ depends on which distribution is selected at each state. One therefore considers reachability probabilities that are subject to a given policy. Let $\Pr^{\mathfrak{S}}(s \models \Diamond G)$ denote this for policy $\mathfrak{S}$. Core questions are then: what is the best (or, dually, the worst) achievable reachability probability for G? The *maximal* reachability probability of $G \subseteq S$ is:

$$\Pr^{\max}(s \models \Diamond G) = \sup_{\mathfrak{S}} \Pr^{\mathfrak{S}}(s \models \Diamond G)$$

where policy $\mathfrak{S}$ ranges over all, infinitely (countably) many, policies. Minimal reachability probabilities are defined similarly. Using a similar procedure as explained above, let variable $x_s = \Pr^{\max}(s \models \Diamond G)$. If $s \in G$, then $x_s = 1$ and if $s \not\models \exists\Diamond G$, then $x_s = 0$. Otherwise:

$$x_s \;=\; \max\Big\{ \sum_{t\in S} \mathbf{P}(s,\mu,t)\cdot x_t \;\mid\; \mu \in \mathcal{D}(s) \Big\}$$

This is an instance of the Bellman equation for dynamic programming. It is well known that for every finite MDP, a positional policy does exist that attains $\Pr^{\max}(s \models \Diamond G)$. Value or policy iteration, and linear programming are computational techniques to obtain these policies. Linear *in*equation systems are thus key—as linear equation systems are for Markov chains—for reachability objectives in finite-state MDPs. Repeated reachability events or persistence probabilities can be obtained by considering maximal end-components [60], the MDP counterpart to terminal SCCs. Whereas positional policies suffice for reachability (and long run) objectives, bounded reachability objectives require finite-memory policies. The same applies to ω-regular properties. For $\Diamond^{\leqslant k} G$ this can be intuitively understood as follows. Consider a state with two choices: one that almost surely leads to G but takes many steps, and one that may lead to G directly, but with a certain probability ends up in a state from which G can never be reached. Then, depending on how many steps are left to reach G, an optimal policy will decide for the (first) safe choice, whereas if only a few steps remain to reach G, it picks the (second) unsafe strategy. LTL model checking of MDPs goes along similar lines as for Markov chains using a product construction with DRA [60].

Timed reachability. Reachability objectives and their variations on finite CTMCs can be determined by considering the embedded DTMC, i.e., by only considering $\mathbf{P}$ while ignoring the rate function r. This works, as these objectives do not refer to the timing. That is to say, they do not impose any constraints on e.g., when the set G is reached. *Timed reachability* is more interesting and more involved. Decidability of (nested, constrained) timed reachability was shown in [10]. We focus on the algorithmic aspects. Consider a CTMC with finite state space and $t \in \mathbb{R}_{\geqslant 0}$. Aim: determine $\Pr(s \models \Diamond^{\leqslant t} G)$. Timed reachability probabilities can be characterised recursively by Volterra integral equation systems [14]. With every state s we associate the function $x_s(d) = \Pr(s \models \Diamond^{\leqslant d} G)$. If G is not reachable from s, then $x_s(d) = 0$ for all d; if $s \in G$. then $x_s(d) = 1$ for all d. For the remaining case we have:

$$x_s(d) \;=\; \int_0^d \sum_{u\in S} \underbrace{\mathbf{R}(s,u)\cdot e^{-r(s)\cdot y}}_{\substack{\text{probability to move to}\\ \text{state } u \text{ at time } x}} \cdot \underbrace{x_u(d-y)}_{\substack{\text{prob. to fulfill}\\ \Diamond^{\leqslant d-y} G \text{ from } u}} \; dy$$

Unlike for the discrete-time models, this recursive characterisation does not immediately provide an algorithmic approach. Solving these integral equation systems is non-trivial, inefficient, and has several pitfalls such as numerical instability. We can however reduce this to another problem for which efficient numerical techniques do exist—computing transient probabilities in CTMCs. Observe that once a path reaches G within t time, then its remaining behaviour is not important. This suggests to make all states in G

[4] It follows that all Eigenvalues of matrix $\mathbf{A}$ are strictly less than one and that the infinite sum of powers of $\mathbf{A}$, that is, $\sum_I \mathbf{A}^i$ converges to the inverse of $\mathbf{I}-\mathbf{A}$. This yields that the matrix $\mathbf{I}-\mathbf{A}$ is non-singular and the linear equation system has a single solution.

absorbing. This yields $\mathfrak{C}[G]$. Then:

$$\underbrace{\Pr(s \models \Diamond^{\leqslant t} G)}_{\text{timed reach. in } \mathfrak{C}} = \underbrace{\Pr(s \models \Diamond^{=t} G)}_{\text{timed reach. in } \mathfrak{C}[G]} = \underbrace{\underline{p}(t) \text{ with } \underline{p}(0) = \underline{1}_s}_{\text{transient prob. in } \mathfrak{C}[G]}.$$

Transient probabilities are solutions of linear differential equations: the vector $\underline{p}(t) = (p_{s_1}(t), \ldots, p_{s_k}(t))$ satisfies:

$$\underline{p}'(t) \;=\; \underline{p}(t) \cdot (\mathbf{R} - \mathbf{r}) \quad \text{given} \quad \underline{p}(0)$$

where $\mathbf{r}$ is the diagonal matrix of r (considered as vector). Solution using standard knowledge yields: $\underline{p}(t) = \underline{p}(0) \cdot e^{(\mathbf{R}-\mathbf{r})\cdot t}$. Computing the matrix exponential is a challenging numerical problem [141, 142]. These problems can be overcome by transforming the CTMC into a uniform one, a CTMC in which $r(s) = r$ for some fixed r, for all states s. Let $r \geqslant max_{s \in S}\, r(s)$. For the uniformized CTMC [87] we have $\overline{r}(s) = r$ for all $s \in S$, and:

$$\overline{\mathbf{P}}(s,s') \;=\; \begin{cases} \frac{r(s)}{r} \cdot \mathbf{P}(s,s') & \text{if } s' \neq s \\ \frac{r(s)}{r} \cdot \mathbf{P}(s,s) + 1 - \frac{r(s)}{r} & \text{otherwise} \end{cases}$$

Uniformization preserves weak bisimulation [17], see Section 4. Then $\underline{p}(t) = \underline{p}(0) \cdot e^{-r\cdot t} \cdot e^{r \cdot t \cdot \overline{\mathbf{P}}}$. Still a matrix exponential remains, but the matrix exponent now is a stochastic matrix. This yields a numerically stable technique, that can be employed for all states in the CTMC simultaneously [106]. Its complexity is linear in the time bound t, the uniformization rate r, and quadratic in $|S|$.

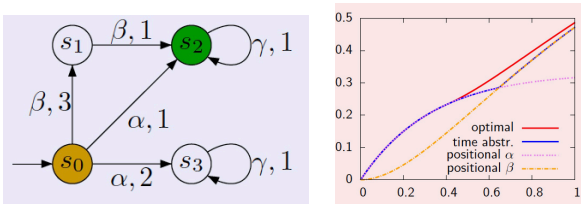
Expected time objectives on CTMCs can be characterised as solutions of sets of linear equations. Long-run average objectives—what is the fraction of time spent in some state in G in the long run?—can be determined using a two-step procedure. First, determine the limiting distribution in any terminal SCC that contains some state in G. This amounts to solving a linear equation system for the terminal SCC. The weighted sum with reachability probabilities of these terminal SCCs yields the long-run average [14]. Alternatively, long-run objectives can be described by automata that "observe" the CTMC; this technique has been advocated in [65]; a similar technique for expected delays originates from [166] and for (path) rewards in [72].

Linear-time timed objectives like "can we reach G within a deadline t while not residing more than d time units in bad states all the while"?, can be encoded as deterministic timed automata (DTA). In a deterministic TA, it is uniquely determined for any time point what will be the next location given the current location. The following procedure can be followed: (a) determine the zone graph of the DTA, (b) take the product of the CTMC and this zone graph [5], and (c) determine the probability to reach an "accepting" zone. The latter reachability probabilities can be characterised by Volterra integral equation systems of the second type [52]. For single-clock DTA, the product can be decomposed into a series of CTMCs. The reachability probability in the product can then be obtained by transient distributions of these CTMCs [26]. An extension of this technique towards infinite-state CTMCs is given in [139]. The extension to multiple rewards is covered in [86]. Checking CTMCs against linear duration properties has been reported in [53]. These properties are stated as conjunctions of linear constraints over the total duration of time spent in states that satisfy a given property.

Timed reachability in CTMDPs. As for MDPs, policies for CTMDPs are oracles to resolve the non-determinism. Whereas in MDPs, positional policies attain maximal reachability probabilities, this is not true in CTMDPs for timed reachability objectives. The *maximal* timed reachability probability of $G \subseteq S$ is:

$$\Pr^{\max}(s \models \Diamond^{\leqslant t} G) = \sup_{\mathfrak{S}} \Pr^{\mathfrak{S}}(s \models \Diamond^{\leqslant t} G)$$

[5] This yields a (simple) piecewise deterministic Markov process.



(a) (b)

Figure 4: A (left) sample CTMDP and (right) some policies for $\Diamond^{\leqslant 1}\, s_2$.

where policy $\mathfrak{S}$ ranges over all, uncountably many, policies. Minimal timed reachability probabilities are defined similarly. Using a similar procedure as explained above for CTMCs, let function $x_s(d) = \Pr^{\max}(s \models \Diamond^{\leqslant d} G)$. If $s \in G$, then $x_s(d) = 1$ and if $s \not\models \exists \Diamond G$, then $x_s(d) = 0$, for all d. Otherwise $x_s(d)$ equals

$$\max \left\{ \int_0^d \sum_{u \in S} \mathbf{R}(s,\mu,u) \cdot e^{-r(s,\mu)\cdot y} \cdot x_u(d-y)\, dy \mid \mu \in \mathcal{D}(s) \right\}$$

For finite-state CTMDPs, timed positional policies are optimal to attain maximal (or minimal) timed reachability probabilities [140, 145]. These policies decide on the total time elapsed so far, and the current state. To illustrate this, consider the CTMDP in Fig. 4(a) and objective $\Diamond^{\leqslant 1}\, s_2$. The initial state is the only non-deterministic state. By choosing distribution β, s_2 is almost surely reached, but the expected time to do so is $4/3$. Picking α reaches s_2 in expected time 1, but with probability $1/3$ only. With probability $2/3$ a state is reached from which the goal can never be reached. The optimal policy in state s_0 selects α if $1 - t_0 \leqslant \ln 3 - \ln 2$, and β otherwise, where t_0 denotes the residence time in s_0; see Fig. 4(b).

As there are uncountably many timed positional policies, one resorts to discretization to obtain ϵ-optimal policies. These policies attain the optimal timed reachability probability up to ϵ. One way to view this, is that a timed positional policies is approximated by a piecewise-continuous policy. Depending on the precision of the discretization, tight bounds can be obtained but at the expense of a considerable time penalty. Various algorithms have been developed in the last years; see e.g., [43, 82]. An efficient algorithm for uniform CTMDPs was proposed in [16]. It was recently shown that by letting the uniformization rate approaching infinity, a uniform CTMDP is obtained on which a time-abstract policy—a policy that does not base its decisions on the elapsed time—arbitrarily closely approximates the optimal timed positional policy on the original CTMDP [44]. In most cases, invoking the algorithm of [16] for various uniformization rates is more efficient than discretization.

Branching-time logics. PCTL [94] is a probabilistic variant of CTL in which the universal and existential path quantifiers are replaced by a probabilistic quantifier. The formula $[\Diamond[\Box G]_{=1}]_{\geqslant 1/2}$, e.g., expresses that with probability at least $1/2$ a state is reached from which almost surely one stays in G. CTL and the qualitative fragment of PCTL, in which only bounds $=1$ and >0 are allowed, are incomparable [11, Ch. 10]. The formulas $[\Diamond G]_{=1}$ and $\forall \Diamond G$ are not equivalent; consider a two-state DTMC where $G = \{u\}$ and $\mathbf{P}(s,u) = \mathbf{P}(s,s) = 1/2$. Then $s \models [\Diamond G]_{=1}$ and $s \not\models \forall \Diamond G$. In fact, there is no CTL formula that is equivalent to $[\Diamond G]_{=1}$, and $\forall \Diamond G$ is not expressible in PCTL. More precisely, this statement holds for infinite Markov chains. For finite DTMCs, $[\Diamond G]_{=1}$ is equivalent to the CTL-formula $\forall \Diamond G$ under strong fairness. In the example, this rules out s^ω, the only run violating $\forall \Diamond G$. PCTL is an expressive logic though. Whereas the LTL-formula $\Diamond \Box\, G$ is not expressible in CTL, it is in PCTL; indeed also $[\Diamond \Box\, G]_{\geqslant 1/2}$ is expressible in PCTL. Model checking a DTMC against a PCTL-formula proceeds by a recursive descent over the parse tree of the formula. The core pro-

cedure is determining (constrained) reachability probabilities. This goes as explained before. A safety-liveness classification for PCTL on DTMCs is given in [111]. PCTL can also be interpreted over MDPs by quantifying over policies; e.g., $[\Diamond G]_{\geqslant 1/2}$ holds if under all policies G is reached eventually with probability at least $1/2$. PCTL* [31] model checking combines the DRA procedure for the LTL sub-formulas with the recursive descent for PCTL. Extensions of these logics with dedicated operators for long-run averages, expected time objectives, reward objectives [8] and so forth can be readily defined. CSL (Continuous Stochastic Logic [10]) is a variant of PCTL that includes until-modalities with timing constraints. The core model-checking procedure is determining timed reachability objectives. Verification algorithms for MA have recently been considered in [88]. Expected time and reachability objectives can be solved by standard MDP algorithms; reward-bounded properties can be reduced to time-bounded reachability properties by exploiting the duality between progress of time and reward, and reducing long-run average properties long-run ratio objectives in MDPs.

Richer property specifications. We mention a few other property classes. Conditional probabilities like $[\Diamond G|\Diamond F]_{>1/2}$ can be efficiently dealt with using a copy-construction of the Markov model [24] or by path reductions [68]. Other properties include PCTL on MDPs with fairness constraints [12], verifying Markov chains against push-down specifications [74], and quantiles [122, 160]. Counterexample generation—how to generate diagnostic feedback in case a property is violated?—for Markov chains is surveyed in [3]. Extensions to CSL have been considered with nested until-formulas [177] and timed-automata constraints [71]. CSL extensions with rewards have been considered in [20, 22]. These model-checking algorithms exploit the duality result of Section 2. Multi-objective specifications [81] focus on questions such as: does an MDP admit a policy satisfying $[\Diamond G_1]_{>1/2}$ and $[\Diamond G_2]_{>1/2}$? Variations of multi-objective specifications have been considered in [23, 42]. An alternative [4] is to consider state distributions of Markov models at each time step, e.g., whether the probability to be in a given state is always at least $1/2$. The algorithms become quite different, and decidability is not always ensured [5]. Finally, we mention the verification of various objectives on stochastic games such as expected total rewards, expected mean-payoff, and ratios of expected mean-payoffs in `PRISM-games` [125].

4. The Abstraction Landscape

A major obstacle is the state-space explosion problem. We survey the main abstraction techniques.

Bisimulation quotienting. For a DTMC with state space S, the equivalence R on S is a *probabilistic bisimulation* [133] on S if all pairs (s,t) in R satisfy that $\mathbf{P}(s,C) = \mathbf{P}(t,C)$ for all equivalence classes $C \in S/R$ where $\mathbf{P}(s,C)$ is shorthand for $\sum_{s' \in C} \mathbf{P}(s,s')$. (If states are labelled with propositional symbols, then s and t need to be equally labelled too.) Let $\sim$ denote the largest possible probabilistic bisimulation, also known as probabilistic bisimilarity. It turns out that Paige-Tarjan's efficient algorithm for bisimulation quotienting of labelled transition systems can be adapted to Markov chains [169]. Computing probabilistic bisimilarity is P-hard, as shown in [51]. The quotient Markov chain is thus obtained in $\mathcal{O}(|\mathbf{P}| \cdot \log |S|)$ where $|\mathbf{P}|$ denotes the number of non-zero elements in the matrix $\mathbf{P}$. Every probabilistic bisimulation R induces a lumping partition [113]—lumpability is a classical concept for Markov chains—and vice versa. $\sim$ yields the coarsest lumpable partition. Altogether this means that the coarsest possible lumpable partition can be obtained in an algorithmic manner. This is an example *par excellence* of a result from formal verification that impacts Markov chain theory! Probabilistic bisimulation can easily be adapted to the other Markov models. For CTMCs, e.g., one requires $\mathbf{R}(s,C) = \mathbf{R}(t,C)$ for all equivalence classes $C \in S/R$. For MDPs, one requires for each distribution in $\mathcal{D}(s)$ the existence of a "matching" distribution in $\mathcal{D}(t)$. The bisimulation relation $\sim$ coincides with PCTL- (and PCTL*-) equivalence on all countably large Markov chains [17]. Coarser abstractions can be obtained using weak bisimulation, a stuttering variant of $\sim$. The underlying idea is that only the conditional probability to leave an equivalence class is of importance and needs to be matched. The notion for DTMC is somewhat involved [17]; for CTMC, it boils down to requiring $\mathbf{R}(s,C) = \mathbf{R}(t,C)$ for all equivalence classes $C \in S/R$, except for $C = [s]_R = [t]_R$. Weak bisimulations preserve PCTL* without next [17]. Experiments show that substantial (up to exponential) state space reductions can be obtained at almost no time penalty [107]. As shown in [107], combinations with symmetry reduction [127] turn out to be rather beneficial too. Bisimulation relations over distributions [73] (rather than over states) are related and have shown their theoretical relevance for linear-time objectives.

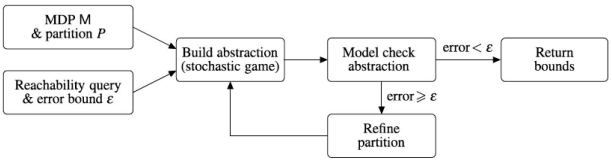


Figure 5: Abstraction-refinement using game-based abstraction (taken from [112]).

More aggressive abstraction. Most abstraction schemes are based on grouping states that are not necessarily bisimilar. Abstract and concrete models are then no longer bisimilar but they are related by a *simulation* relation. Abstraction is typically conservative in the sense that affirmative verification results for abstract models carry over to concrete models. That is to say, if the abstract model satisfies a property, the concrete one does so too. Probabilistic simulation, e.g., preserves a safe fragment of PCTL, but not the full PCTL [17]. This does not apply to negative verification results, as false negatives may occur due to over-approximation in the abstraction. Three-valued semantics, i.e., an interpretation in which a formula evaluates to either true, false, or indefinite may help out. In this setting, abstraction is conservative for both positive and negative verification results. Only if the verification of the abstract model yields an indefinite answer ("don't know"), the validity in the concrete model is unknown. This has been adopted to Markov models [110]. For a queueing system from performance evaluation, (hand-crafted) three-valued abstraction shows that 10^{278} concrete states (calculated analytically) can be reduced to 1.2 million states, while preserving six decimals accuracy for timed reachability probabilities [116]. An important question is which type of model to use as abstraction. Models that are used include interval Markov chains, abstract probabilistic automata [69] that equip PA with modalities, and two-player stochastic games [112]. The latter are used for obtaining abstractions of MDPs. One player is representing the non-determinism that is inherent in the MDP, while the other player controls the non-determinism introduced by the abstraction. Crucially, this allows lower and upper bounds to be computed for the reachability properties of the MDP. The tightness of these bounds indicate the quality of the abstraction and form the basis of refinement, see Fig. 5. Experiments show encouraging results reducing models of millions of states to hundreds of states in a few abstraction-refinement iterations [112]. A closely related approach is menu-based abstraction [174].

Compositional abstraction. In order to avoid treating the entire system model for abstraction, one can exploit composition-

ality. This amounts to applying abstraction in a component-by-component fashion. This requires the abstraction relation (such as $\sim$, weak bisimilarity, or similarity) to be a congruence with respect to parallel composition. For PA and MA, this means:

$$(\mathcal{M}_1 \sim \mathcal{N}_1 \text{ and } \mathcal{M}_2 \sim \mathcal{N}_2) \text{ implies } \mathcal{M}_1 \,||_A\, \mathcal{M}_2 \sim \mathcal{N}_1 \,||_A\, \mathcal{N}_2$$

phrased for the bisimulation $\sim$. Compositional abstraction works component-wise. Each component $\mathcal{M}_i$ is abstracted by $\alpha(\mathcal{M}_i)$. Then $\mathcal{M}_1 \,||_A \ldots ||_A\, \mathcal{M}_n$ is abstracted by $\alpha(\mathcal{M}_1) \,||_A \ldots ||_A\, \alpha(\mathcal{M}_n)$. Or, groups of parallel processes can be taken and abstracted, applying a similar regime. This has been applied to PA and (modal extensions of) MA [99, 108]. These strategies all attempt to reduce the peak memory consumption during state space generation. Exploitation of the compositional system structure in the verification using assume-guarantee verification has been considered in [119, 130].

Other techniques. There are several avenues to tackle the state space explosion problem. One can argue that bisimulation is too precise, in particular for probabilistic models. This can be remedied by considering approximate bisimulations [170]; however, quotienting is rather expensive. The usage of symbolic data structures such as multi-terminal BDDs originates from [13]. For many practical examples (see also the next section), this is quite scalable. On-the-fly partial-order reduction has been adapted to MDPs (and PA and MA) [18]. As opposed to the aforementioned techniques, this works on the high-level description of the MDP, not on the MDP itself. Experimental results show that using adequate heuristics two thirds of the total achievable reduction possible with weak bisimulation can be obtained [168]. Other techniques include parallelization [35] and Monte Carlo simulation [6]. To analyse models of many replicas, mean-field approximation can be employed [34]. The latter technique is based on counter abstraction, and yields approximate results for (in the limit infinitely) many replicas. A survey on abstraction techniques for probabilistic systems is given in [67].

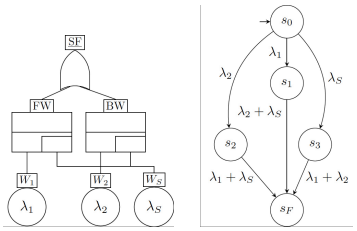
5. The Application Landscape

This section focuses on some practical applications of probabilistic model checking.

Reliability engineering. Probabilistic safety assessment is common practice for safety-critical systems, and often required by law. Typical measures of interest are the *mean time to failure*—what is the expected time of the failure?—and *reliability*—how likely is the system operational up to time t? Fault tree analysis (FTA) [165, 172] is one of the most (if not *the* most) prominent safety assessment technique. A recent detailed survey is given in [161]. It is standardised by the IEC, and deployed by many companies and institutions, like FAA, NASA, ESA, Airbus, Honeywell, etc. Fault trees (FTs) model how failures propagate through the system: FT *leaves* model component failures and are equipped with continuous probability distributions; FT *gates* model how component failures lead to system failures. FTs are one of the most prominent model to describe top-down causes for a system failure and facilitate, amongst others, the analysis of the mean time to failure and system reliability. It is a common assumption that the failure of FT leaves is governed by exponential distributions.

Remark. The incorporation of stochastic timing is motivated by the fact that failures and repairs—the key events in reliability analysis—occur randomly. Especially for failures, the negative exponential distribution is a specific, though rather reasonable

[6] Also called statistical model checking [134, 176]; this is however typically restricted to bounded reachability properties (and variants thereof) and cannot handle non-determinism.



(a) (b)

Figure 6: A (a) sample DFT with three leaves, an OR-gate (top event) and two SPARE-gates (its children), and (b) its CTMC.

choice. The exponential distribution maximises the entropy[7]: if only the mean failure rate is known, then the most appropriate continuous-time distribution is the exponential distribution with that mean [135]. For mechanical component failures, the bathtub curves states that after the burn-in the failure rate is constant until at some point wearing-off becomes a major influence [117]. For repairs, other distributions such as uniform or combinations of uniform and deterministic distributions come into the play. These distributions can however be matched arbitrarily closely with phase-type distributions (at the cost of state-space increase), distributions that are defined as the time until absorption in a CTMC [146].

Due to redundancy, not every single component failure leads to a system failure. Static fault trees have logical gates such as AND- and OR-gates, but no inverters. Their analysis is easy and can be done efficiently using BDD-based techniques. Dynamic fault trees (DFTs) [75] are directed acyclic graphs that are more expressive and more involved than static fault trees. They cater for common dependability patterns, such as spare management, functional dependencies, and sequencing. DFTs have an internal state, e.g., the order in which events fail influences the internal state, and thus whether the designated top event has failed. DFT leaves can be either dormant, standby, active, or failed. Basic events are assumed to be independent, therefore they almost surely never fail simultaneously. The failure rate depends on the status; an active leaf has failure rate λ, dormant and standby leaves have a reduced rate. The DFT in Fig. 6(a), e.g., models (part of) a motor bike with a spare wheel. If either wheel W_1 or W_2 fails, the motor bike fails. Either wheel can be replaced by the spare wheel W_S but not both. W_S is less likely to fail as long as it is not used; this is modelled by a reduced failure rate. Assume the front wheel W_1 fails. The spare wheel W_S is available and used, while its failure rate is increased, as its status changes from dormant (or standby) to active. If any other wheel fails, e.g., W_2 then no spare wheels are available any more. In that case, the SPARE BW and the DFT SF fails.

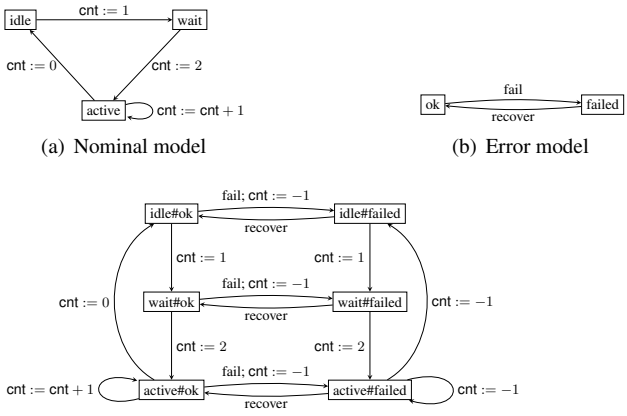
The behaviour of DFTs can be naturally described by CTMCs, where transitions correspond to the failure of a basic event. Fig. 6(b) depicts the CTMC of our sample DFT. (Strictly speaking, DFTs can exhibit non-determinism due to the presence of functional dependency-gates; Markov automata (MA) are appropriate as semantics. For the sake of simplicity, we refrain from considering such gates.) It turns out that the state space generation, i.e., the generation of the CTMC for a given DFT, is one of the main bottlenecks. By exploiting successful reduction techniques from classical model checking, such as symmetry and (static) partial-order reduction, the state-space generation can be boosted drastically. Symmetry reduction exploits the fact that many parts in DFTs are symmetric and that failures have analogous effects in symmetric parts; e.g.,

[7] According to the principle of maximum entropy, if nothing is known about a distribution except that it belongs to a certain class (usually defined in terms of specified properties or measures), then the distribution with the largest entropy should be chosen as the least-informative default.

in the example the front- and back wheel are symmetric. Symmetry reduction combined with pruning sub-DFTs that become obsolete (don't care) after the occurrence of some failures turns out to yield substantial gains. Recent experiments have shown that the state-space generation is accelerated up to five orders of magnitude by exploiting these model-checking techniques [173]. The generation of CTMCs with about 20 million states from symmetric DFTs is a matter of a few minutes. The probabilistic verification of the resulting CTMC—determining the DFT's mean-time to failure, or its reliability—can be done using the algorithms explained before. This is very fast and negligible compared to the state-space generation. Alternative techniques [36] exploit the structure of the DFT by generating the CTMC (in fact, an MA) in a compositional manner. Here, an MA is generated for each DFT gate, and is enriched with some transitions to enable its composition with the MA for other gates. During this compositional state-space generation, bisimulation quotienting is employed on the individual MA to reduce the peak memory consumption. Efficient state-space generation techniques combined with probabilistic model checking has been successfully applied to large instances of benchmarks DFTs from the literature (stemming from different application fields) as well as to safety assessment of Dutch railway systems, see e.g. [89].

Dependability. The second application is concerned with dependability modelling and analysis of spacecrafts. System dependability evaluation is tightly related to performance evaluation, but especially concerned with evaluating service continuity of systems while failures may occur. It goes without saying that spacecraft systems such as satellites, Mars pathfinders, and launchers are subject to extreme dependability requirements. Space systems engineering is an evolving field and its current state of practice is strongly influenced by software. The advent of digital interfaces of parts and equipment, and the flexibility of software-based control over analogue interfaces and electrical/mechanical control led to an exponential growth of the size of the deployed software in space crafts. In a cooperation of almost a decade with the ESA (European Space Agency), an extension of the Architecture Analysis & Design Language (AADL) [83] has been developed with accompanying tool support that includes probabilistic model checking [38]. The AADL dialect enables to express the system, the software and—most importantly—its reliability models in a single modelling language. This language is equipped with a rigorous formal semantics, that maps models onto an automata-based formalism. The automata are extended with data, continuous dynamics (to describe temperature, pressure and the like), and discrete and continuous probability distributions. The latter are primarily used for modelling the failure behaviour of (typically redundant) system components. AADL is a component-based modelling language in which hierarchical system components interact with each other in a rendezvous manner. The behaviour of component consists of two parts: its *nominal* and its *error* behaviour. The effect of an error occurrence is described by a so-called fault injection, basically an assignment to some variables in the nominal part. The nominal behaviour is given by a (possibly non-deterministic) state-transition diagram, while the error behaviour is described by a CTMC. The error model expresses how faults may affect normal operation and may lead the system into a degraded mode of operation. The parallel composition of these two "automata" together with the fault injection gives the total component's behaviour. Modes are thus pairs of nominal modes and error model states. The transition relation consists of all possible interleavings and interactions between the nominal and error model, taking failure effects into account. An example is given in Fig. 7.

Ignoring the hybrid and timed behaviour yields an MA, a CTMC with possible non-determinism. The generation and analysis of the resulting MA is supported by the COMPASS tool-set; its



(a) Nominal model (b) Error model

(c) Extended model

Figure 7: Nominal (a) and error model (b) with fault injection cnt := 1 yields extended model (c).

model checker is described in [37]. MA are reduced by weak bisimulation quotienting; in case this yields a CTMC—due to the elimination of spurious non-determinism by the quotienting—CTMC model checking is employed; otherwise MA verification is applied. A large internal case study at ESA modelled a satellite platform [39, 78]. The 90 components modelled the reaction wheels, and the attitude and orbital control system (AOCS) etc.; the 20 error models range from sensor failures to propulsion failures and AOCS break down. The nominal model already consists of 50 million states. Fault injection yields blow-ups ranging from a factor 3 (for local errors) up to 200,000 (for the AOCS). Using BDDs, combined with (weak) bisimulation quotienting, and property-based abstraction, the AADL model of the satellite has been successfully analysed [78]. The need for more effective abstraction techniques is clearly given as the reliability of the satellite in the presence of a sensor failure could not be computed in nine hours. A more detailed model of the satellite has also been analysed in [39]. The ESA activities are perhaps the largest industrial project so far [8] that has used probabilistic model checking.

Other applications and tools. Probabilistic model checking has been adopted by various *performance modelling* techniques, most notably by (generalized) stochastic Petri nets (GSPNs) [138]. Transitions in these Petri nets are equipped with exponential distributions, and safe SPNs yield finite-state CTMCs. Due to the presence of immediate transitions, GSPNs may have confusion; their marking graph is no longer a CTMC but an MA [77]. Established GSPN tools such as GreatSPN [7] nowadays include CSL model checking. We also mention a (plug-in) extension of STATEMATE that augments Statechart models with probabilistic timing information in a compositional manner—in a similar manner as described above for AADL—and exploits CTMDP model-checking algorithms [32] The availability of powerful software tools such as PRISM [129], MRMC [109], LiQuor [56], iscasMC [92], and PAT [164] has boosted the application of probabilistic model checking in a wide variety of application fields. An important application field is *systems biology*. CTMCs naturally reflect the operations of biological mechanisms such as molecular reactions. In recent years various biological systems have been studied by CTMC model checking [61, 131, 139]. These include Petri net approaches [97, 137]. A recent overview is given in [126]. In particular, computing time-bounded reachability probabilities and long-run probabilities is of importance. Other applications include: quantitative secu-

[8] See compass.informatik.rwth-aachen.de for more details.

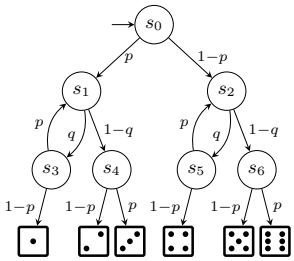


Figure 8: A variant of the Knuth-Yao die for two unfair coins.

rity [149], stochastic scheduling, planning, robotics [154], probabilistic programs [115], data-flow computations [105], user activity patterns for mobile apps [9], and so forth. Extensions with rewards have been applied, amongst others, to energy-aware computing [150]. An extensive set of case studies can be found on `www.prismmodelchecker.org`. Statistical model checkers that check a temporal logic formula using Monte Carlo simulation have emerged in the last years [134, 176].

6. The New Parameter Synthesis Landscape

The parameter synthesis problem. A major practical obstacle is that probabilistic model checking assumes that all probabilities (rates) in the Markov model are a priori known. However, at early development stages, certain system quantities such as fault rates, reaction rates, packet loss ratios, etc. are often not—or at the best partially—known. In such cases, *parametric* probabilistic models can be used, where transition probabilities are specified as arithmetic expressions over real-valued parameters. The problem of *parameter synthesis* is: Given a finite-state parametric Markov model, what are the parameter values for which a given reachability property exceeds (or is below) a given threshold β? Put differently, what probabilities in the system are tolerable such that the overall probability to reach some critical states is below a given threshold like 10^{-8}? Answering such a question may for instance allow developers of network protocols to decide for which communication channels reliability is most important, or it may help reliability engineers optimising investments by focussing on the most critical components. Due to possible dependencies between parametric transition probabilities, synthesis is intrinsically hard. Parameter synthesis has considerably advanced recently, and we survey available techniques for discrete-time parametric Markov models.

Computing symbolic reachability probabilities. As a running example we consider a parametric variant of the Knuth-Yao algorithm [118]. We assume that a finite number of parameters is given. Transitions in parametric DTMCs (pMCs, for short) are labelled with rational functions over these parameters. A rational function is a fraction of polynomials in terms of the parameters. No restrictions on the shape of multivariate polynomials are imposed; the rational functions should, of course, be on $[0, 1]$. It is assumed that pMCs are realizable, i.e., there exists a parameter evaluation such that a DTMC is induced. Checking this is NP-hard [132]. Consider two biased coins that result in heads with probability p and q, respectively. In the parametric Knuth-Yao algorithm, we throw these coins in an alternating fashion, see Fig. 8. A sample synthesis question is: For which values of p and q is the probability of eventually getting ⚀ or ⚁ at most $1/3$? Note that there are multiple trade-offs in the sense that, e.g., higher or lower values of p (or q) are not necessarily beneficial for the probability of either ⚀ or ⚁. It is possible to compute a rational function expressing the reachability probability in a pMC. In the example, this yields for ⚀ the non-linear rational function $p \cdot \frac{q \cdot (1-p)}{1-p \cdot q}$. This can be obtained—as first observed in [64] and implemented in [91]—by state elimination, a technique akin to constructing a regular expression from a finite-state automaton. We use state and transition elimination, as illustrated below. The principle is to remove states one-by-one from the pMC until only a start state and the target states remain. We show this procedure for the left sub-tree of the parametric die. Eliminating state s_3 (see Fig. 9(a)) results in transitions from s_1 to the targets of the outgoing transitions of s_3. Eliminating the self-loop at s_1 (see Fig. 9(b)) rescales all other outgoing transitions. Eliminating s_1 (see Fig. 9(c)) yields the rational function describing the probability for the event $\Diamond$⚀. Finally, eliminating state s_4 yields Fig. 9(d). This approach can be generalised to obtain rational functions for objectives such as conditional reachability probabilities [68] and expected rewards in pMCs whose rewards may be parametric too.

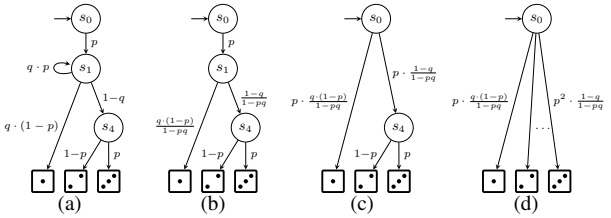


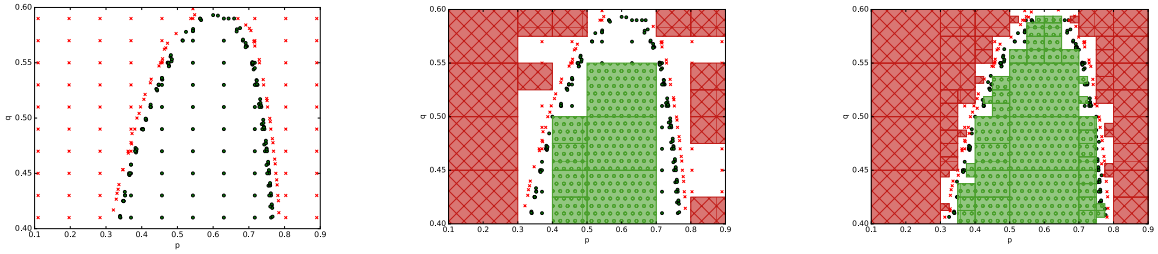
Figure 9: State-elimination for parametric Markov chains.

Depending on the structure and the size of the pMC, this procedure may yield rational functions with many high-degree multivariate polynomials. For many Markov models it is beneficial to construct the rational function by exploiting their structure. A practically viable approach is to apply a hierarchical decomposition of the pMC into SCCs [102]. This yields a tree of SCCs, the root being the pMC at hand. One can then determine the rational functions in a bottom-up fashion over this tree, starting with the leaves, i.e., the SCCs that do not contain any SCCs. The rationale behind this strategy is to keep the rational functions manageable. Together with improved gcd-computations, a bottleneck for computing rational functions, this approach scales well. In addition, probabilistic bisimulation quotienting on pMCs can be applied using the polynomial algorithm for ordinary MCs together with SMT (Satisfiability Modulo Theory) techniques [58] that simplify the rational functions and compare them. Experimental results [102] show that this approach to computing rational functions works efficiently for pMCs of up to a few million states and two parameters.

Partitioning the parameter space. The aim of parameter synthesis is to obtain all parameter values for which a given property holds. This can be conveniently represented by partitioning the parameter space, indicating for which sets of parameter values—called regions—the property holds, and for which ones it does not. The first regions are *safe*; the second ones are *unsafe*. Formally, regions are half-spaces defined by a system of linear inequalities over the parameters. A region R for threshold $\leqslant \beta$ for fixed probability β and rational function f is safe iff there is no well-formed parameter valuation $v \in R$ [9] such that f instantiated with v exceeds β. An unsafe region R does not contain a valuation v such that f instantiated by v is at most β. We present two approaches to partition the parameter space into safe and unsafe regions: generating candidate regions that are checked for (non-) safeness, and parameter lifting.

Candidate region generation and checking. To partition the parameter space into safe (indicated in green) and unsafe regions (indicated in red), one can—either algorithmically or user-guided—indicate candidate regions. To check whether a region is safe or not, satisfiability checking can be employed. This approach is based on a symbolic representation of reachability probabilities as given by the rational functions obtained by, e.g., state elimination. For our running example, the aim is to determine for a region R such as

[9] A parameter valuation is well-formed whenever it induces a Markov chain.



(a) (b) (c)

Figure 10: Sampling (a) and two snapshots (b) and (c) of incremental parameter space partitioning for $\Pr(\Diamond \boxdot) \geqslant 3/20$.

$1/3 \leqslant p \leqslant 1/2$ and $1/2 \leqslant q \leqslant 2/3$, all values of p and q such that (see Fig. 9(d)):

$$p \cdot \frac{q \cdot (1-p)}{1 - p \cdot q} + p^2 \cdot \frac{1-q}{1 - p \cdot q} \leqslant 1/3.$$

This can be done by SMT techniques over non-linear real arithmetic [103] supported by tools such as Z3 [66] and SMT-RAT [58]. If a candidate region is not safe (or not unsafe), the counterexample provided by the SMT solver can be used to split the candidate region into sub-regions. This naturally gives rise to a CEGAR[10]-like loop: once the rational function is computed, one first checks a number of instantiations of the rational function up to a user-adjustable degree. This yields a coarse abstraction of the parameter space partitioning. Based on the sampling, a safe (unsafe) candidate region is constructed. An SMT-solver then either verifies that this region is indeed safe (unsafe). Fig. 10 shows a typical initial sampling, together with an intermediate and an (almost) final partitioning for the two parameters of the parametric version of Knuth-Yao's die. The tool `PROPhESY` [68] supports this procedure.

Parameter lifting. Instead of computing a rational function and involving an SMT-solver, the new *parameter lifting* technique takes a different approach [158]. It takes as input a pMC but is—in contrast to the above technique—also directly applicable to parametric MDPs. For each unique parameter at transitions emanating from a state, we use a fresh parameter to remove parameter dependencies. This is called *relaxation*. Now, for each function over these parameters, we compute extremal values, i.e., maximal and minimal probabilities based on the region of interest. The key idea is to replace the (parametric) probabilistic choice at each state by a non-deterministic choice between these extremal values. This second step is called *substitution*. The resulting (non-parametric) MDP can be verified using the algorithms discussed before. This analysis yields sound over- and under-approximations for all parameter instantiations within the given region. Parameter lifting thus allows verifying regions via standard MDP model checking and avoids computing rational functions and SMT solving. This approach is sound if transition probabilities are linear in each variable, and regions are hyper-rectangles. These restrictions are not severe; most benchmarks from the literature adhere to this. Let us give an example. Consider Fig. 11(a) and let region $R = [0.1, 0.3] \times [0.4, 0.6]$. The relaxation (see Fig. 11(b)) replaces parameter p in s_4 by y, the bounds for y are as for p, $[0.1, 0.3]$. Now, every variable occurs only at a single state, so the choice of a variable becomes a local choice. Note that this over-approximates the possible instantiations. Substitution yields the MDP in Fig. 11(c). That is, for each outgoing transition in a state we either pick the upper or lower bound for the corresponding variable. The solid (dashed) lines depict transitions that belong to the action for the upper (lower) bound. The elegance of this approach is that it is simple and can be applied to

[10] CounterExample-Guided Abstraction-Refinement.

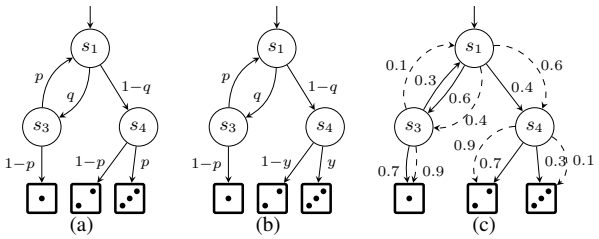


Figure 11: Illustrating (b) relaxation and (c) substitution.

parametric MDPs too. It also avoids the main practical bottlenecks of computing the rational function and SMT solving. Substitution then yields a two-player stochastic game, see [158]. The analysis of this game using standard means [84] yields upper- and lower-bounds on the maximal and minimal reachability probabilities for the pMDP. These bounds are safe by construction. The technique can also be used to verify expected reward properties. By replacing sampling of the rational function, parameter lifting can be embedded into a parameter space partitioning framework as described above. Parameter lifting enables parameter space partitioning for pMCs and pMDPs with a 95% coverage for up to a few million states and four parameters [158].

Applications. Despite the fact that parameter synthesis is a rather young research area, several applications have been reported recently. Parametric probabilistic models are used to rank patches in the repair of software [136] and are quite natural in adaptive software where "continuous" verification frequently amends system models during deployment [45]. Parameter synthesis in CTMCs has first been considered in [93]; recent extensions are reported in [46]. Early work on parameter synthesis for discrete probabilistic models occurred in [132]. Perturbation analysis [54, 167] and model-repair [28] exploit parametric Markov models.

Other synthesis works. Rather than having parametric transition probabilities, properties may be parametric. Consider the parametric LTL-formula $\Diamond_{\leqslant x} \varphi$, where x is a natural variable. The synthesis problem now is to determine the set of valuations of x such that the probability to satisfy the instantiated formula is below threshold β. This is undecidable in general but decidable for some LTL fragments and qualitative properties [47]. The controller synthesis problem is: given a (CT)MDP and a reachability objective, it is possible to effectively synthesize a controller? For MDPs and PCTL extended with long-run averages, the controller synthesis problem is decidable [123]; for a similar logic and MDPs in which some (but not all) non-determinism is controllable, it is NP-hard [15]. Other variants include the control-flow synthesis from libraries of reusable probabilistic components—which is decidable for almost-sure specification [49, 143]—the synthesis of systems in probabilistic environments [50] and the synthesis of probabilistic programs [147].

Acknowledgments

A big thanks goes to all my co-workers who joint me in the probabilistic model-checking adventure in the past years. Christian Dehnert, Sebastian Junges, Nils Jansen and Thomas Noll are thanked for their valuable feedback on a draft version.

References

[1] A. Abate, J.-P. Katoen, J. Lygeros, and M. Prandini. Approximate model checking of stochastic hybrid systems. *Eur. J. Control*, 16(6): 624–641, 2010.

[2] P. A. Abdulla, N. B. Henda, and R. Mayr. Decisive Markov chains. *Logical Methods in Computer Science*, 3(4), 2007.

[3] E. Ábrahám, B. Becker, C. Dehnert, N. Jansen, J.-P. Katoen, and R. Wimmer. Counterexample generation for discrete-time Markov models: An introductory survey. In *SFM*, volume 8483 of *LNCS*, pages 65–121. Springer, 2014.

[4] M. Agrawal, S. Akshay, B. Genest, and P. S. Thiagarajan. Approximate verification of the symbolic dynamics of Markov chains. *J. ACM*, 62(1):2:1–2:34, 2015.

[5] S. Akshay, T. Antonopoulos, J. Ouaknine, and J. Worrell. Reachability problems for Markov chains. *Inf. Process. Lett.*, 115(2):155–158, 2015.

[6] R. Alur, T. A. Henzinger, and M. Vardi. Theory in practice for system design and verification. *ACM SIGLOG News*, 3, 2015.

[7] E. G. Amparore, G. Balbo, M. Beccuti, S. Donatelli, and G. Franceschinis. 30 years of GreatSPN. In *Principles of Performance and Reliability Modeling and Evaluation*, Springer Series in Reliability Engineering, pages 227–254. Springer, 2016.

[8] S. Andova, H. Hermanns, and J.-P. Katoen. Discrete-time rewards model checked. In *FORMATS*, volume 2791 of *LNCS*, pages 88–104. Springer-Verlag, 2003.

[9] O. Andrei, M. Calder, M. Chalmers, A. Morrison, and M. Rost. Probabilistic formal analysis of app usage to inform redesign. *CoRR*, abs/1510.07898, 2015.

[10] A. Aziz, K. Sanwal, V. Singhal, and R. K. Brayton. Model-checking continous-time Markov chains. *ACM Trans. Comput. Log.*, 1(1):162–170, 2000.

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

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

[13] C. Baier, E. M. Clarke, V. Hartonas-Garmhausen, M. Z. Kwiatkowska, and M. Ryan. Symbolic model checking for probabilistic processes. In *ICALP*, volume 1256 of *LNCS*, pages 430–440. Springer, 1997.

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

[15] C. Baier, M. Größer, M. Leucker, B. Bollig, and F. Ciesinski. Controller synthesis for probabilistic systems. In *IFIP TCS*, volume 155 of *IFIP*, pages 493–506. Kluwer/Springer, 2004.

[16] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort. Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes. *Theor. Comput. Sci.*, 345(1):2–26, 2005.

[17] C. Baier, J.-P. Katoen, H. Hermanns, and V. Wolf. Comparative branching-time semantics for Markov chains. *Inf. Comput.*, 200(2): 149–214, 2005.

[18] C. Baier, P. R. D'Argenio, and M. Größer. Partial order reduction for probabilistic branching time. *Electr. Notes Theor. Comput. Sci.*, 153 (2):97–116, 2006.

[19] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Reachability in continuous-time Markov reward decision processes. In *Logic and Automata History and Perspectives*, volume 2 of *Texts in Logic and Games*, pages 53–72. Amsterdam University Press, 2008.

[20] C. Baier, L. Cloth, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Performability assessment by model checking of Markov reward models. *Formal Methods in System Design*, 36(1):1–36, 2010.

[21] C. Baier, M. Größer, and N. Bertrand. Probabilistic ω-automata. *J. ACM*, 59(1):1, 2012.

[22] C. Baier, E. M. Hahn, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking for performability. *Mathematical Structures in Computer Science*, 23(4):751–795, 2013.

[23] C. Baier, C. Dubslaff, and S. Klüppelholz. Trade-off analysis meets probabilistic model checking. In *CSL-LICS*, pages 1:1–1:10. ACM, 2014.

[24] C. Baier, J. Klein, S. Klüppelholz, and S. Märcker. Computing conditional probabilities in Markovian models efficiently. In *TACAS*, volume 8413 of *LNCS*, pages 515–530. Springer, 2014.

[25] C. Baier, L. de Alfaro, V. Forejt, and M. Kwiatkowska. Probabilistic model checking. In *Handbook of Model Checking*. Springer, 2016 (to appear).

[26] B. Barbot, T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Efficient CTMC model checking of linear real-time objectives. In *TACAS*, volume 6605 of *LNCS*, pages 128–142. Springer, 2011.

[27] F. Bartels, A. Sokolova, and E. P. de Vink. A hierarchy of probabilistic system types. *Theor. Comput. Sci.*, 327(1-2):3–22, 2004.

[28] E. Bartocci, R. Grosu, P. Katsaros, C. R. Ramakrishnan, and S. A. Smolka. Model repair for probabilistic systems. In *TACAS*, volume 6605 of *LNCS*, pages 326–340. Springer, 2011.

[29] M. D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Trans. on Computers*, 27(6):540–547, 1978.

[30] N. Bertrand, P. Bouyer, T. Brihaye, Q. Menet, C. Baier, M. Größer, and M. Jurdzinski. Stochastic timed automata. *Logical Methods in Computer Science*, 10(4), 2014.

[31] A. Bianco and L. de Alfaro. Model checking of probabilistic and nondeterministic systems. In *FSTTCS*, volume 1026 of *LNCS*, pages 499–513. Springer-Verlag, 1995.

[32] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, J.-H. Rakow, R. Wimmer, and B. Becker. Compositional dependability evaluation for STATEMATE. *IEEE Trans. Software Eng.*, 35(2):274–292, 2009.

[33] G. Bolch, S. Greiner, H. de Meer, and K. S. Trivedi. *Queueing Networks and Markov Chains: Modeling and Performance Evaluation with Computer Science Applications*. Wiley, 2006.

[34] L. Bortolussi, J. Hillston, D. Latella, and M. Massink. Continuous approximation of collective system behaviour: A tutorial. *Perform. Eval.*, 70(5):317–349, 2013.

[35] D. Bosnacki, S. Edelkamp, D. Sulewski, and A. Wijs. Parallel probabilistic model checking on general purpose graphics processors. *STTT*, 13(1):21–35, 2011.

[36] H. Boudali, P. Crouzen, and M. Stoelinga. A rigorous, compositional, and extensible framework for dynamic fault tree analysis. *IEEE Trans. Dependable Sec. Comput.*, 7(2):128–143, 2010.

[37] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, M. Roveri, and R. Wimmer. A model checker for AADL. In *CAV*, volume 6174 of *LNCS*, pages 562–565. Springer, 2010.

[38] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri. Safety, dependability and performance analysis of extended AADL models. *Comput. J.*, 54(5):754–775, 2011.

[39] M. Bozzano, A. Cimatti, J.-P. Katoen, P. Katsaros, K. Mokos, V. Y. Nguyen, T. Noll, B. Postma, and M. Roveri. Spacecraft early design validation using formal methods. *Rel. Eng. & Sys. Safety*, 132:20–35, 2014.

[40] T. Brázdil, J. Esparza, S. Kiefer, and A. Kucera. Analyzing probabilistic pushdown automata. *Formal Methods in System Design*, 43 (2):124–163, 2013.

[41] T. Brázdil, S. Kiefer, A. Kucera, P. Novotný, and J.-P. Katoen. Zero-reachability in probabilistic multi-counter automata. In *CSL-LICS*, pages 22:1–22:10. ACM, 2014.

[42] V. Bruyère, E. Filiot, M. Randour, and J.-F. Raskin. Meet your expectations with guarantees: Beyond worst-case synthesis in quantitative games. In *STACS*, volume 25 of *LIPIcs*, pages 199–213. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik, 2014.

[43] P. Buchholz and I. Schulz. Numerical analysis of continuous time Markov decision processes over finite horizons. *Computers & OR*, 38(3):651–659, 2011.

[44] Y. Butkova, H. Hatefi, H. Hermanns, and J. Krcál. Optimal continuous time Markov decisions. In *ATVA*, volume 9364 of *LNCS*, pages 166–182. Springer, 2015.

[45] R. Calinescu, C. Ghezzi, M. Z. Kwiatkowska, and R. Mirandola. Self-adaptive software needs quantitative verification at runtime. *Commun. ACM*, 55(9):69–77, 2012.

[46] M. Ceska, F. Dannenberg, M. Z. Kwiatkowska, and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *CMSB*, volume 8859 of *LNCS*, pages 86–98. Springer, 2014.

[47] S. Chakraborty and J.-P. Katoen. Parametric LTL on Markov chains. In *IFIP TCS*, volume 8705 of *LNCS*, pages 207–221. Springer, 2014.

[48] K. Chatterjee and T. A. Henzinger. A survey of stochastic ω-regular games. *J. Comput. Syst. Sci.*, 78(2):394–413, 2012.

[49] K. Chatterjee, L. Doyen, and M. Y. Vardi. The complexity of synthesis from probabilistic components. In *ICALP (2)*, volume 9135 of *LNCS*, pages 108–120. Springer, 2015.

[50] K. Chatterjee, T. A. Henzinger, B. Jobstmann, and R. Singh. Measuring and synthesizing systems in probabilistic environments. *J. ACM*, 62(1):9:1–9:34, 2015.

[51] D. Chen, F. van Breugel, and J. Worrell. On the complexity of computing probabilistic bisimilarity. In *FoSSaCS*, volume 7213 of *LNCS*, pages 437–451. Springer, 2012.

[52] T. Chen, T. Han, J.-P. Katoen, and A. Mereacre. Model checking of continuous-time Markov chains against timed automata specifications. *Logical Methods in Computer Science*, 7(1), 2011.

[53] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. Verification of linear duration properties over continuous-time Markov chains. *ACM Trans. Comput. Log.*, 14(4):33, 2013.

[54] T. Chen, Y. Feng, D. S. Rosenblum, and G. Su. Perturbation analysis in verification of discrete-time Markov chains. In *CONCUR*, volume 8704 of *LNCS*, pages 218–233. Springer, 2014.

[55] L. Cheung, N. A. Lynch, R. Segala, and F. W. Vaandrager. Switched PIOA: parallel composition via distributed scheduling. *Theor. Comput. Sci.*, 365(1-2):83–108, 2006.

[56] F. Ciesinski and C. Baier. Liquor: A tool for qualitative and quantitative linear time analysis of reactive systems. In *QEST*, pages 131–132. IEEE Computer Society, 2006.

[57] E. M. Clarke. The birth of model checking. In *25 Years of Model Checking*, volume 5000 of *LNCS*, pages 1–26. Springer, 2008.

[58] F. Corzilius, G. Kremer, S. Junges, S. Schupp, and E. Ábrahám. SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In *SAT*, volume 9340 of *LNCS*, pages 360–368. Springer, 2015.

[59] C. Courcoubetis and M. Yannakakis. Verifying temporal properties of finite-state probabilistic programs. In *FOCS*, pages 338–345. IEEE Computer Society, 1988.

[60] C. Courcoubetis and M. Yannakakis. The complexity of probabilistic verification. *J. ACM*, 42(4):857–907, 1995.

[61] F. Dannenberg, M. Z. Kwiatkowska, C. Thachuk, and A. J. Turberfield. DNA walker circuits: computational potential, design, and verification. *Natural Computing*, 14(2):195–211, 2015.

[62] P. R. D'Argenio and J.-P. Katoen. A theory of stochastic systems part I: stochastic automata. *Inf. Comput.*, 203(1):1–38, 2005.

[63] A. David, K. G. Larsen, A. Legay, M. Mikucionis, and D. B. Poulsen. Uppaal SMC tutorial. *STTT*, 17(4):397–415, 2015.

[64] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, volume 3407 of *LNCS*, pages 280–294. Springer, 2004.

[65] L. de Alfaro. How to specify and verify the long-run average behavior of probabilistic systems. In *LICS*, pages 454–465. IEEE Computer Society, 1998.

[66] L. M. de Moura and N. Bjørner. Z3: an efficient SMT solver. In *TACAS*, volume 4963 of *LNCS*, pages 337–340. Springer, 2008.

[67] C. Dehnert, D. Gebler, M. Volpato, and D. N. Jansen. On abstraction of probabilistic systems. In *ROCKS*, volume 8453 of *LNCS*, pages 87–116, 2012.

[68] C. Dehnert, S. Junges, N. Jansen, F. Corzilius, M. Volk, H. Bruintjes, J.-P. Katoen, and E. Ábrahám. PROPhESY: A PRObabilistic ParamEter SYnthesis tool. In *CAV (1)*, volume 9206 of *LNCS*, pages 214–231. Springer, 2015.

[69] B. Delahaye, J.-P. Katoen, K. G. Larsen, A. Legay, M. L. Pedersen, F. Sher, and A. Wasowski. Abstract probabilistic automata. *Inf. Comput.*, 232:66–116, 2013.

[70] Y. Deng and M. Hennessy. On the semantics of Markov automata. *Inf. Comput.*, 222:139–168, 2013.

[71] S. Donatelli, S. Haddad, and J. Sproston. Model checking timed and stochastic properties with CSL^{TA}. *IEEE Trans. Software Eng.*, 35 (2):224–240, 2009.

[72] W. Douglas Obal II and W. H. Sanders. State-space support for path-based reward variables. *Perform. Eval.*, 35(3-4):233–251, 1999.

[73] L. Doyen, T. A. Henzinger, and J.-F. Raskin. Equivalence of labeled Markov chains. *Int. J. Found. Comput. Sci.*, 19(3):549–563, 2008.

[74] C. Dubslaff, C. Baier, and M. Berg. Model checking probabilistic systems against pushdown specifications. *Inf. Process. Lett.*, 112(8-9):320–328, 2012.

[75] J. B. Dugan, S. J. Bavuso, and M. A. Boyd. Dynamic fault-tree models for fault-tolerant computer systems. *IEEE Trans. on Reliability*, 41(3):363–377, 1992.

[76] C. Eisentraut, H. Hermanns, and L. Zhang. On probabilistic automata in continuous time. In *LICS*, pages 342–351. IEEE CS, 2010.

[77] C. Eisentraut, H. Hermanns, J.-P. Katoen, and L. Zhang. A semantics for every GSPN. In *Petri Nets*, volume 7927 of *LNCS*, pages 90–109. Springer, 2013.

[78] M.-A. Esteve, J.-P. Katoen, V. Y. Nguyen, B. Postma, and Y. Yushtein. Formal correctness, safety, dependability, and performance analysis of a satellite. In *ICSE*, pages 1022–1031. IEEE, 2012.

[79] K. Etessami. Analysis of probabilistic processes and automata theory. In *Automata: from Mathematics to Applications*. 2016 (to appear).

[80] K. Etessami and M. Yannakakis. Recursive Markov decision processes and recursive stochastic games. *J. ACM*, 62(2):11, 2015.

[81] K. Etessami, M. Z. Kwiatkowska, M. Y. Vardi, and M. Yannakakis. Multi-objective model checking of Markov decision processes. *Logical Methods in Computer Science*, 4(4), 2008.

[82] J. Fearnley, M. N. Rabe, S. Schewe, and L. Zhang. Efficient approximation of optimal control for continuous-time Markov games. *Inf. Comput.*, 247:106–129, 2016.

[83] P. H. Feiler and D. P. Gluch. *Model-Based Engineering with AADL - An Introduction to the SAE Architecture Analysis and Design Language*. SEI series in software engineering. Addison-Wesley, 2012.

[84] J. Filar and K. Vrieze. *Competitive Markov Decision Processes*. Springer-Verlag, 1996.

[85] V. Forejt, M. Z. Kwiatkowska, G. Norman, and D. Parker. Automated verification techniques for probabilistic systems. In *SFM*, volume 6659 of *LNCS*, pages 53–113. Springer, 2011.

[86] H. Fu. Maximal cost-bounded reachability probability on continuous-time Markov decision processes. In *FoSSaCS*, volume 8412 of *LNCS*, pages 73–87. Springer, 2014.

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

[88] D. Guck, H. Hatefi, H. Hermanns, J.-P. Katoen, and M. Timmer. Analysis of timed and long-run objectives for Markov automata. *Logical Methods in Computer Science*, 10(3), 2014.

[89] D. Guck, J.-P. Katoen, M. I. A. Stoelinga, T. Luiten, and J. Romijn. Smart railroad maintenance engineering with stochastic model checking. In *Railways*, volume 104, pages 299–314. Civil-Comp Press, 2014.

[90] X. Guo and O. Hernandez-Lerma. *Continuous-Time Markov Decision Processes: Theory and Applications*, volume 62 of *Stochastic Modelling and Applied Probability*. Springer-Verlag, 2009.

[91] E. M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *STTT*, 13(1):3–19, 2011.

[92] E. M. Hahn, Y. Li, S. Schewe, A. Turrini, and L. Zhang. iscasMc: A web-based probabilistic model checker. In *FM*, volume 8442 of *LNCS*, pages 312–317. Springer, 2014.

[93] T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *RTSS*, pages 173–182. IEEE Computer Society, 2008.

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

[95] S. Hart, M. Sharir, and A. Pnueli. Termination of probabilistic concurrent program. *ACM Trans. Program. Lang. Syst.*, 5(3):356–380, 1983.

[96] A. Hartmanns and H. Hermanns. In the quantitative automata zoo. *Sci. Comput. Program.*, 112:3–23, 2015.

[97] M. Heiner, D. R. Gilbert, and R. Donaldson. Petri nets for systems and synthetic biology. In *SFM*, volume 5016 of *LNCS*, pages 215–264. Springer, 2008.

[98] H. Hermanns. *Interactive Markov Chains: The Quest for Quantified Quality*, volume 2428 of *LNCS*. Springer, 2002.

[99] H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephone system. *Sci. Comput. Program.*, 36(1):97–127, 2000.

[100] J. Hillston. Process algebras for quantitative analysis. In *LICS*, pages 239–248. IEEE Computer Society, 2005.

[101] R. A. Howard. *Dynamic Probabilistic Systems,* volume 2*: Semi-Markov and Decision Processes*. John Wiley & Sons, 1972.

[102] N. Jansen, F. Corzilius, M. Volk, R. Wimmer, E. Ábrahám, J.-P. Katoen, and B. Becker. Accelerating parametric probabilistic verification. In *QEST*, volume 8657 of *LNCS*, pages 404–420. Springer, 2014.

[103] D. Jovanovic and L. M. de Moura. Solving non-linear arithmetic. In *IJCAR*, volume 7364 of *LNCS*, pages 339–354. Springer, 2012.

[104] J.-P. Katoen. Model checking meets probability: A gentle introduction. In *Engineering Dependable Software Systems*, volume 34 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 177–205. IOS Press, 2013.

[105] J.-P. Katoen and H. Wu. Probabilistic model checking for uncertain scenario-aware data flow. *ACM Trans. Embedded Comp. Sys.*, pages 1–26, 2016 (to appear).

[106] J.-P. Katoen, M. Z. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In *PAPM-PROBMIV*, volume 2165 of *LNCS*, pages 23–38. Springer, 2001.

[107] J.-P. Katoen, T. Kemna, I. S. Zapreev, and D. N. Jansen. Bisimulation minimisation mostly speeds up probabilistic model checking. In *TACAS*, volume 4424 of *LNCS*, pages 87–101. Springer, 2007.

[108] J.-P. Katoen, D. Klink, and M. R. Neuhäußer. Compositional abstraction for stochastic systems. In *FORMATS*, volume 5813 of *LNCS*, pages 195–211. Springer, 2009.

[109] J.-P. Katoen, I. S. Zapreev, E. M. Hahn, H. Hermanns, and D. N. Jansen. The ins and outs of the probabilistic model checker MRMC. *Perform. Eval.*, 68(2):90–104, 2011.

[110] J.-P. Katoen, D. Klink, M. Leucker, and V. Wolf. Three-valued abstraction for probabilistic systems. *J. Log. Algebr. Program.*, 81 (4):356–389, 2012.

[111] J.-P. Katoen, L. Song, and L. Zhang. Probably safe or live. In *CSL-LICS*, pages 55:1–55:10. ACM, 2014.

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

[113] J. Kemeny and J. Snell. *Finite Markov Chains*. D. Van Nostrand, 1960.

[114] J. Kemeny and J. Snell. *Denumerable Markov Chains*. D. Van Nostrand, 1976.

[115] S. Kiefer, A. S. Murawski, J. Ouaknine, B. Wachter, and J. Worrell. APEX: an analyzer for open probabilistic programs. In *CAV*, volume 7358 of *LNCS*, pages 693–698. Springer, 2012.

[116] D. Klink, A. Remke, B. R. Haverkort, and J.-P. Katoen. Time-bounded reachability in tree-structured QBDs by abstraction. *Perform. Eval.*, 68(2):105–125, 2011.

[117] G. Klutke, P. C. Kiessler, and M. A. Wortman. A critical look at the bathtub curve. *IEEE Trans. Reliability*, 52(1):125–129, 2003.

[118] D. Knuth and A. Yao. *Algorithms and Complexity: New Directions and Recent Results*, chapter The complexity of nonuniform random number generation. Academic Press, 1976.

[119] A. Komuravelli, C. S. Pasareanu, and E. M. Clarke. Assume-guarantee abstraction refinement for probabilistic systems. In *CAV*, volume 7358 of *LNCS*, pages 310–326. Springer, 2012.

[120] D. Kozen. Semantics of probabilistic programs. *J. Comput. Syst. Sci.*, 22(3):328–350, 1981.

[121] D. Kozen. A probabilistic PDL. In *STOC*, pages 291–297. ACM, 1983.

[122] D. Krähmann, J. Schubert, C. Baier, and C. Dubslaff. Ratio and weight quantiles. In *MFCS (1)*, volume 9234 of *LNCS*, pages 344–356. Springer, 2015.

[123] A. Kucera and O. Strazovský. On the controller synthesis for finite-state Markov decision processes. *Fundam. Inform.*, 82(1-2):141–153, 2008.

[124] M. Kudlek. Probability in Petri nets. *Fundam. Inform.*, 67(1-3):121–130, 2005.

[125] M. Kwiatkowska, D. Parker, and C. Wiltsche. PRISM-Games 2.0: A tool for multi-objective strategy synthesis for stochastic games. In *TACAS*, volume 9636 of *LNCS*, pages 560–566. Springer, 2016.

[126] M. Z. Kwiatkowska and C. Thachuk. Probabilistic model checking for biology. In *Software Systems Safety*, volume 36 of *NATO Science for Peace and Security Series, D: Information and Communication Security*, pages 165–189. IOS Press, 2014.

[127] M. Z. Kwiatkowska, G. Norman, and D. Parker. Symmetry reduction for probabilistic model checking. In *CAV*, volume 4144 of *LNCS*, pages 234–248. Springer, 2006.

[128] M. Z. Kwiatkowska, G. Norman, and D. Parker. Stochastic model checking. In *SFM*, volume 4486 of *LNCS*, pages 220–270. Springer, 2007.

[129] M. Z. Kwiatkowska, G. Norman, and D. Parker. PRISM 4.0: Verification of probabilistic real-time systems. In *CAV*, volume 6806 of *LNCS*, pages 585–591. Springer, 2011.

[130] M. Z. Kwiatkowska, G. Norman, D. Parker, and H. Qu. Compositional probabilistic verification through multi-objective model checking. *Inf. Comput.*, 232:38–65, 2013.

[131] M. R. Lakin, D. Parker, L. Cardelli, M. Kwiatkowska, and A. Phillips. Design and analysis of dna strand displacement devices using probabilistic model checking. *J R Soc Interface*, 9:1470–1485, 2012.

[132] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput.*, 19(1):93–109, 2007.

[133] K. G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, 94(1):1–28, 1991.

[134] A. Legay, B. Delahaye, and S. Bensalem. Statistical model checking: An overview. In *RV*, volume 6418 of *LNCS*, pages 122–135. Springer, 2010.

[135] J. Lisman and M. van Zuylen. Note on the generation of most probable frequency distributions. *Statistica Neerlandica*, 26(1):19–23, 1972.

[136] F. Long and M. Rinard. Automatic patch generation by learning correct code. In *POPL*, pages 298–312. ACM, 2016.

[137] C. Madsen, Z. Zhang, N. Roehner, C. Winstead, and C. J. Myers. Stochastic model checking of genetic circuits. *JETC*, 11(3):23:1–23:21, 2014.

[138] M. A. Marsan, G. Conte, and G. Balbo. A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Trans. Comput. Syst.*, 2(2):93–122, 1984.

[139] L. Mikeev, M. R. Neuhäußer, D. Spieler, and V. Wolf. On-the-fly verification and optimization of DTA-properties for large Markov chains. *Formal Methods in System Design*, 43(2):313–337, 2013.

[140] B. L. Miller. Finite state continuous time Markov decision processes with a finite planning horizon. *SIAM J. on Control*, 6:266–280, 1968.

[141] C. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM Review*, 45(1): 3–49, 2003.

[142] C. B. Moler and C. F. Van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20:801–836, 1978.

[143] S. Nain, Y. Lustig, and M. Y. Vardi. Synthesis from probabilistic components. *Logical Methods in Computer Science*, 10(2), 2014.

[144] N. Napp and E. Klavins. A compositional framework for programming stochastically interacting robots. *I. J. Robotic Res.*, 30(6):713–729, 2011.

[145] M. R. Neuhäußer, M. Stoelinga, and J.-P. Katoen. Delayed nondeterminism in continuous-time Markov decision processes. In *FOSSACS*, volume 5504 of *LNCS*, pages 364–379. Springer, 2009.

[146] M. F. Neuts. *Matrix-geometric solutions in stochastic models - an algorithmic approach*. Dover Publications, 1994.

[147] A. V. Nori, S. Ozair, S. K. Rajamani, and D. Vijaykeerthy. Efficient synthesis of probabilistic programs. In *PLDI*, pages 208–217. ACM, 2015.

[148] G. Norman. Analysing randomized distributed algorithms. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 384–418. Springer, 2004.

[149] G. Norman and V. Shmatikov. Analysis of probabilistic contract signing. In *FASec*, volume 2629 of *LNCS*, pages 81–96. Springer, 2002.

[150] G. Norman, D. Parker, M. Z. Kwiatkowska, S. K. Shukla, and R. Gupta. Using probabilistic model checking for dynamic power management. *Formal Asp. Comput.*, 17(2):160–176, 2005.

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

[152] J. R. Norris. *Markov chains*. Cambridge Series in Statistical and Probabilistic Mathematics. Cambridge University Press, 1998.

[153] P. Panangaden. *Labelled Markov Processes*. Imperial College Press, 2009.

[154] S. Pathak, L. Pulina, and A. Tacchella. Evaluating probabilistic model checking tools for verification of robot control policies. *AI Commun.*, 29(2):287–299, 2016.

[155] A. Paz. *Introduction to Probabilistic Automata*. Academic Press, 1971.

[156] A. Pnueli and L. D. Zuck. Probabilistic verification by tableaux. In *LICS*, pages 322–331. IEEE Computer Society, 1986.

[157] M. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. Wiley, 1994.

[158] T. Quatmann, C. Dehnert, N. Jansen, S. Junges, and J.-P. Katoen. Parameter synthesis for Markov models: Faster than ever. *CoRR*, abs/1602.05113, 2016.

[159] M. O. Rabin. Probabilistic automata. *Inf. and Control*, 6(3):230–245, 1963.

[160] M. Randour, J.-F. Raskin, and O. Sankur. Percentile queries in multi-dimensional Markov decision processes. In *CAV (1)*, volume 9206 of *LNCS*, pages 123–139. Springer, 2015.

[161] E. Ruijters and M. Stoelinga. Fault tree analysis: A survey of the state-of-the-art in modeling, analysis and tools. *Computer Science Review*, 15:29–62, 2015.

[162] P. Schnoebelen. The verification of probabilistic lossy channel systems. In *Validation of Stochastic Systems*, volume 2925 of *LNCS*, pages 445–466. Springer, 2004.

[163] R. Segala and N. A. Lynch. Probabilistic simulations for probabilistic processes. *Nord. J. Comput.*, 2(2):250–273, 1995.

[164] S. Song, J. Sun, Y. Liu, and J. S. Dong. A model checker for hierarchical probabilistic real-time systems. In *CAV*, volume 7358 of *LNCS*, pages 705–711. Springer, 2012.

[165] M. Stamatelatos, W. Vesely, J. B. Dugan, J. Fragola, J. Minarick, and J. Railsback. *Fault Tree Handbook with Aerospace Applications*. NASA Headquarters, 2002.

[166] E. W. Stark and S. A. Smolka. Compositional analysis of expected delays in networks of probabilistic I/O automata. In *LICS*, pages 466–477. IEEE Computer Society, 1998.

[167] G. Su, D. S. Rosenblum, and G. Tamburrelli. Reliability of run-time quality-of-service evaluation using parametric model checking. In *ICSE*. ACM, 2016 (to appear).

[168] M. Timmer, J.-P. K. J. van de Pol, and M. Stoelinga. Confluence reduction for Markov automata. *Theor. Comput. Sci.*, 2016 (to appear).

[169] A. Valmari and G. Franceschinis. Simple $O(m \log n)$ time Markov chain lumping. In *TACAS*, volume 6015 of *LNCS*, pages 38–52. Springer, 2010.

[170] F. van Breugel and J. Worrell. A behavioural pseudometric for probabilistic transition systems. *Theor. Comput. Sci.*, 331(1):115–142, 2005.

[171] M. Y. Vardi. Automatic verification of probabilistic concurrent finite-state programs. In *FOCS*, pages 327–338. IEEE, 1985.

[172] W. E. Vessely, F. F. Goldberg, N. H. Roberts, and D. F. Haasl. *Fault Tree Handbook*. Office of Nuclear Regulatory Research, U.S. Nuclear Regulatory Commission, 1981.

[173] M. Volk, S. Junges, and J.-P. Katoen. Advancing dynamic fault tree analysis. In *SafeComp*, LNCS. Springer, 2016 (to appear).

[174] B. Wachter and L. Zhang. Best probabilistic transformers. In *VMCAI*, volume 5944 of *LNCS*, pages 362–379. Springer, 2010.

[175] S.-H. Wu, S. A. Smolka, and E. W. Stark. Composition and behaviors of probabilistic I/O automata. *Theor. Comput. Sci.*, 176(1-2):1–38, 1997.

[176] H. L. S. Younes and R. G. Simmons. Statistical probabilistic model checking with a focus on time-bounded properties. *Inf. Comput.*, 204 (9):1368–1409, 2006.

[177] L. Zhang, D. N. Jansen, F. Nielson, and H. Hermanns. Automata-based CSL model checking. *Logical Methods in Computer Science*, 8(2), 2011.