Formal Methods in System Design (2024) 62:181–259
https://doi.org/10.1007/s10703-023-00442-x

Parameter synthesis for Markov models: covering the parameter space

Sebastian Junges[1] · Erika Ábrahám[3] · Christian Hensel[3] · Nils Jansen[1,2] · Joost-Pieter Katoen[3] · Tim Quatmann[3] · Matthias Volk[4]

Received: 15 July 2020 / Accepted: 28 September 2023 / Published online: 17 February 2024
© The Author(s) 2024

Abstract
Markov chain analysis is a key technique in formal verification. A practical obstacle is that all probabilities in Markov models need to be known. However, system quantities such as failure rates or packet loss ratios, etc. are often not—or only partially—known. This motivates considering parametric models with transitions labeled with functions over parameters. Whereas traditional Markov chain analysis relies on a single, fixed set of probabilities, analysing parametric Markov models focuses on synthesising parameter values that establish a given safety or performance specification φ. Examples are: what component failure rates ensure the probability of a system breakdown to be below 0.00000001?, or which failure rates maximise the performance, for instance the throughput, of the system? This paper presents various analysis algorithms for parametric discrete-time Markov chains and Markov decision processes. We focus on three problems: (a) do all parameter values within a given region satisfy φ?, (b) which regions satisfy φ and which ones do not?, and (c) an approximate version of (b) focusing on covering a large fraction of all possible parameter values. We give a detailed account of the various algorithms, present a software tool realising these techniques, and report on an extensive experimental evaluation on benchmarks that span a wide range of applications.

Keywords Formal methods · Verification · Model checking · Probabilistic systems · Parameter synthesis · Markov chains

The work has been partially supported by the DFG RTG 2236 UnRAVeL, the European Union's Horizon 2020 research and innovation programme under the Marie Skłodowska-Curie Grant Agreement No. 101008233 (Mission), and the ERC Starting Grant 101077178 (DEUCE).

✉ Nils Jansen
n.jansen@rub.de

[1] Radboud University, Nijmegen, The Netherlands

[2] Ruhr-University, Bochum, Germany

[3] Department of Computer Science, RWTH Aachen University, Aachen, Germany

[4] Eindhoven University of Technology, Eindhoven, The Netherlands

1 Introduction

Uncertainty. Probabilistic model checking subsumes a multitude of formal verification techniques for systems that exhibit uncertainties [16, 54, 98]. Such systems are typically modeled by Markov chains or Markov decision processes [121]. Applications range from reliability, dependability and performance analysis to systems biology, take for instance reliability measures such as the mean time between failures in fault trees [28, 123] and the probability of a system breakdown within a time limit.

The results of probabilistic model checking algorithms are rigorous, their quality depends solely on the system models. Yet, there is one major practical obstacle: All probabilities (or rates) in the Markov model are precisely known *a priori*. In many cases, this assumption is too severe. System quantities such as component fault rates, molecule reaction rates, packet loss ratios, etc. are often not, or at best partially, known. Let us give a few examples. The quality of service of a (wireless) communication channel may be modelled by e.g., the popular Gilbert-Elliott model, a two-state Markov chain in which packet loss has an unknown probability depending on the channel's state [112]. Other examples include the back-off probability in CSMA/CA protocols determining a node's delay before attempting a transmission [1], the bias of used coins in self-stabilising protocols [88, 105], and the randomised choice of selecting the type of time-slots (sleeping, transmit, or idle) in the birthday protocol, a key mechanism used for neighbour discovery in wireless sensor networks [110] to lower power consumption. In particular, in early stages of reliable system design, the concrete failure rate of components [55] is left unspecified. Optimally, analyses in this stage may even guide the choice of a concrete component from a particular manufacturer.

The probabilities in all these systems are deliberately left unspecified. They can later be determined in order to optimise some performance or dependability measure. Dually, some systems should be robust for all (reasonable) failure rates. For example, a network protocol should ensure a reasonable quality of service for each reasonable channel quality.

Parametric probabilistic models. What do these examples have in common? The random variables for packet loss, failure rate etc. are not fully defined, but are *parametric*. Whether a parametric system satisfies a given property or not—"is the probability that the system goes down within k steps below 10^{-8}"—depends on these parameters. Relevant questions are then: for which concrete parameter values is such a property satisfied—the *(parameter) synthesis problem*—and, in case of decision-making models, which parameter values yield optimal designs? That is, for which fixed probabilities do such protocols work in an optimal way, i.e., lead to maximal reliability, maximise the probability for nodes to be discovered, or minimise the time until stabilisation, and so on. These questions are intrinsically hard as parameters can take infinitely many different values that, in addition, can depend on each other.

This paper faces these challenges and *presents various algorithmic techniques to treat different variations of the (optimal) parameter synthesis problem*. To deal with uncertainties in randomness, *parametric* probabilistic models are adequate. These models are just like Markov models except that the transition probabilities are specified by arithmetic expressions over real-valued parameters. Transition probabilities are thus functions over a set of parameters. A simple instance is to use intervals over system parameters imposing constant lower and upper bounds on every parameter [74, 100]. The general setting as considered here is more liberal as it e.g., includes the possibility to express complex parameter dependencies. We address the analysis of *parametric* Markov models where probability distributions are functions

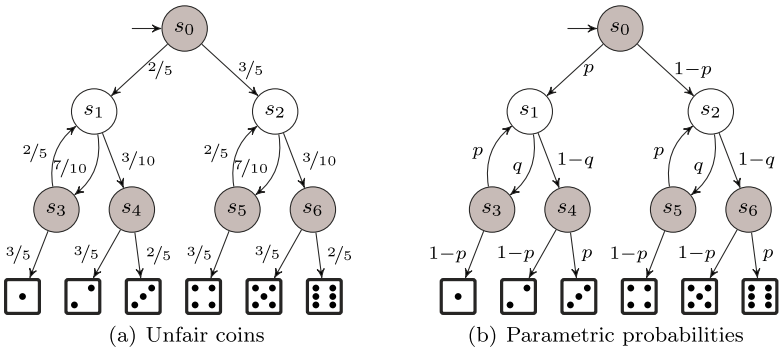


Fig. 1 A **a** biased and **b** parametric variant of Knuth-Yao's algorithm. In gray states an unfair coin is flipped with probability 2/5 for 'heads'; for the unfair coin in the white states this probability equals 7/10. On the right, the two biased coins have parametric probabilities

over system parameters, specifically, parametric discrete-time Markov *chains* (pMCs) and parametric discrete-time Markov *decision processes* (pMDPs).

Example 1 The Knuth-Yao randomised algorithm [99] uses repeated coin flips to model a six-sided die. It uses a fair coin to obtain each possible outcome ('one', 'two', ..., 'six') with probability 1/6. Figure 1a depicts a Markov chain (MC) of a variant in which two *unfair* coins are flipped in an alternating fashion. Flipping the unfair coins yields *heads* with probability 2/5 (gray states) or 7/10 (white states), respectively. Accordingly, the probability of *tails* is 3/5 and 3/10, respectively. The event of throwing a 'two' corresponds to reaching the state ⚁ in the MC. Assume now a *specification* that requires the probability to obtain 'two' to be larger than 3/20. Knuth-Yao 's original algorithm accepts this specification as using a fair coin results in 1/6 as probability to end up in ⚁. The biased model, however, does not satisfy the specification; in fact, a 'two' is reached with probability 1/10.

Probabilistic model checking. The analysis algorithms presented in this paper are strongly related to (and presented as) techniques from probabilistic model checking. *Model checking* [13, 46] is a popular approach to verify the correctness of a system by systematically evaluating all possible system runs. It either certifies the absence of undesirable (dangerous) behaviour or delivers a system run witnessing a violating system behaviour. Traditional model checking typically takes two inputs: a finite transition system modelling the system at hand and a temporal logic formula specifying a system requirement. Model checking then amounts to checking whether the transition system satisfies the logical specification, which in its simplest form describes that a particular state can (not) be reached. Model checking is nowadays a successful analysis technique adopted by mainstream hardware and software industry [49, 101].

To cope with real-world systems exhibiting random behaviour, model checking has been extended to deal with probabilistic, typically Markov, models. *Probabilistic model checking* [13, 16, 98] takes as input a Markov model of the system at hand together with a quantitative specification specified in some probabilistic extension of LTL or CTL. Example specifications are e.g., "is the probability to reach some bad (or degraded) state below a safety threshold λ?" or "is the expected time until the system recovers from a fault bounded by some threshold κ". Efficient probabilistic model-checking techniques do exist for models such as discrete-time Markov chains (MCs), Markov decision processes (MDPs), and their continuous-time

counterparts [98]. Probabilistic model checking extends and complements long-standing analysis techniques for Markov models.

It has been adopted in the field of performance analysis to analyse stochastic Petri nets [4, 38], in dependability analysis for analysing architectural system descriptions [29], in reliability engineering for fault tree analysis [27, 139], as well as in security [114], distributed computing [105], and systems biology [104]. Unremitting algorithmic improvements employing the use of symbolic techniques to deal with large state spaces have led to powerful and popular software tools realising probabilistic model checking techniques such as `PRISM` [102] and `Storm` [66].

1.1 Problem statements

We now give a more detailed description of the parameter synthesis problems considered in this paper. We start off by establishing the connection between parametric Markov models and concrete ones, i.e., ones in which the probabilities are fixed such as MCs and MDPs. Each parameter in a pMC or pMDP (where p stands for parametric) has a given parameter range. The *parameter space* of the parametric model is the Cartesian product of these parameter ranges. Instantiating the parameters with a concrete value in the parameter space to the parametric model results in an *instantiated* model. The parameter space defines all possible parameter instantiations, or equivalently, the instantiated models. A parameter instantiation that yields a Markov model, e.g., results in probability distributions, is called *well-defined*. In general, a parametric Markov model defines an uncountably infinite *family of Markov models*, where each family member is obtained by a well-defined instantiation. A *region* R is a fragment of the parameter space; it is well-defined if all instantiations in R are well-defined.

Example 2 (pMC) Figure 1b depicts a parametric version of the biased Knuth-Yao die from Example 1. It has parameters $V = \{p, q\}$, where p is the probability of outcome *heads* in gray states and q the same for white states. The parameter space is $\{(p, q) \mid 0 < p, q < 1\}$. The probability for *tails* is $1-p$ and $1-q$, respectively. The sample instantiation u with $u(p) = 2/5$ and $u(q) = 7/10$ is well-defined and results in the MC in Fig. 1a. The region

$$R = \{u \colon V \to \mathbb{R} \mid 1/10 \leq u(p) \leq 9/10 \text{ and } 3/4 \leq u(q) \leq 5/6\}$$

is well-defined. Contrarily, region

$$R' = \{u \mid 1/5 \leq u(p) \leq 6/5 \text{ and } 2/5 \leq u(q) \leq 7/10\}$$

is not well-defined, as it contains the instantiation u' with $u'(p) = 6/5$ which does not yield an MC. For pMCs whose transition probabilities are high-degree polynomials, it is not always obvious whether a region is well-defined.

We are now in a position to describe the three problems considered in this paper.

The verification problem is defined as follows:

> **The verification problem.** Given a parametric Markov model $\mathcal{D}$, a well-defined region R, and a specification φ, the *verification problem* is to check whether *all* instantiations of $\mathcal{D}$ within R satisfy φ.

Consider the following possible outcomes:

– If R only contains instantiations of $\mathcal{D}$ satisfying φ, then the verification problem evaluates to `true` and the Markov model $\mathcal{D}$ on region R *accepts* specification φ. Whenever $\mathcal{D}$ and φ are clear from the context, we call R *accepting*.

– If R contains an instantiation of $\mathcal{D}$ refuting φ, then the problem evaluates to `false`. If R contains only instantiations of $\mathcal{D}$ refuting φ, then $\mathcal{D}$ on R *rejects* φ. Whenever $\mathcal{D}$ and φ are clear from the context, we call R *rejecting*.
– If R contains instantiations satisfying φ as well as instantiations satisfying $\neg\varphi$, then $\mathcal{D}$ on R is inconclusive w. r. t. φ. In this case, we call R *inconsistent*.

In case the verification problem yields `false` for φ, one can only infer that the region R is not accepting, but not conclude whether R is inconsistent or rejecting. To determine whether R is rejecting, we need to consider the verification problem for the negated specification $\neg\varphi$. Inconsistent regions for φ are also inconsistent for $\neg\varphi$.

Example 3 (**Verification problem**) Consider the pMC $\mathcal{D}$, the well-defined region R from Example 2, and the specification $\varphi' := \neg\varphi$ that constrains the probability to reach ⚀ to be at most $3/20$. The verification problem is to determine whether all instantiations of $\mathcal{D}$ in R satisfy φ'. As there is no instantiation within R for which the probability to reach ⚀ is above $3/20$, the verification problem evaluates to `true`. Thus, R *accepts* φ'.

Typical structurally simple regions are described by hyperrectangles or given by linear constraints, rather than non-linear constraints; we refer to such regions as *simple*. A simple region comprising a large range of parameter values may likely be inconsistent, as it contains both instantiations satisfying φ, and some satisfying $\neg\varphi$. Thus, we generalise the problem to synthesise a partition of the parameter space.

The exact synthesis problem is described as follows:

The synthesis problem. Given a parametric Markov model $\mathcal{D}$ and a specification φ, the *(parameter) synthesis problem* is to partition the parameter space of $\mathcal{D}$ into an accepting region R_a and a rejecting region R_r for φ.

The aim is to obtain such a partition in an automated manner. A complete sub-division of the parameter space into accepting and rejecting regions provides deep insight into the effect of parameter values on the system's behaviour. The exact division typically is described by non-linear functions over the parameters, referred to as *solution functions*.

Example 4 Consider the pMC $\mathcal{D}$, the region R, and the specification φ as in Example 3. The solution function:

$$f_\varphi(p,q) = \frac{p \cdot (1-q) \cdot (1-p)}{1 - p \cdot q}$$

describes the probability to eventually reach ⚀. Given that φ imposes a lower bound of $3/20$, we obtain

$$R_a = \{u \mid f(u(p), u(q)) \geq 3/20\} \text{ and } R_r = R \setminus R_a.$$

The example illustrates that exact symbolic representations of the accepting and rejecting regions may be complex and hard to compute algorithmically. The primary reason is that the boundaries are described by non-linear functions. A viable alternative therefore is to consider an approximative version of the synthesis problem.

The approximate synthesis problem: As argued before, the regions obtained via exact synthesis are typically not simple. The aim of the approximate synthesis problem is to use simpler and more tractable representations of regions. As such shapes ultimately approximate the exact solution function, simple regions become infinitesimally small when getting close to

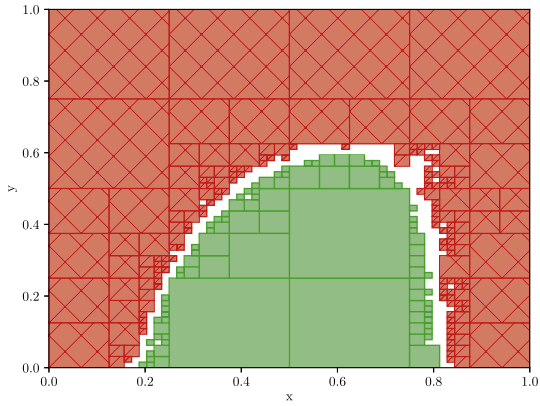


Fig. 2 Parameter space partitioning into accepting (green), rejecting (red), and unknown (white) regions

the border between accepting and rejecting areas. For computational tractability, we are thus interested in *approximating* a partition of the parameter space in accepting and rejecting regions, where we allow also for a (typically small) part to be covered by possibly inconsistent regions. Practically this means that $c\,\%$ of the entire parameter space is covered by simple regions that are either accepting or rejecting, for some adequate value of c. Altogether this results in the following problem description:

The approximate synthesis problem. Given a parametric Markov model, a specification φ, and a percentage c, the *approximate (parameter) synthesis problem* is to partition the parameter space of $\mathcal{D}$ into a simple accepting region R_a and a simple rejecting region R_r for φ such that $R_a \cup R_r$ cover at least $c\%$ of the entire parameter space.

Example 5 Consider the pMC $\mathcal{D}$, the region R, and the specification φ as in Example 3. The parameter space in Fig. 2 is partitioned into simple regions (rectangles). The green (dotted) area—the union of a number of smaller rectangular accepting regions—indicates the parameter values for which φ is satisfied, whereas the red (hatched) area indicates the set of rejecting regions for φ. The white area indicates the unknown regions. The indicated partition covers 95% of the parameter space. The sub-division into accepting and rejecting (simple) regions approximates the solution function $f_\varphi(p, q)$ given before.

1.2 Solution approaches

We now outline our approaches to solve the verification problem and the two synthesis problems. For the sake of convenience, we start with the synthesis problem.

Synthesis. The most straightforward description of the sets R_a and R_r is of the form:

$$R_a = \{u \mid \mathcal{D}[u] \text{ satisfies } \varphi\} \quad \text{and}$$
$$R_r = \{u \mid \mathcal{D}[u] \text{ satisfies } \neg\varphi\}.$$

The satisfaction relation (denoted $\models$) can be concisely described by a set of linear equations over the transition probabilities [13]. As in the parametric setting the transition probabilities are no longer fixed, but rather defined over a set of parameters, the equations become non-linear.

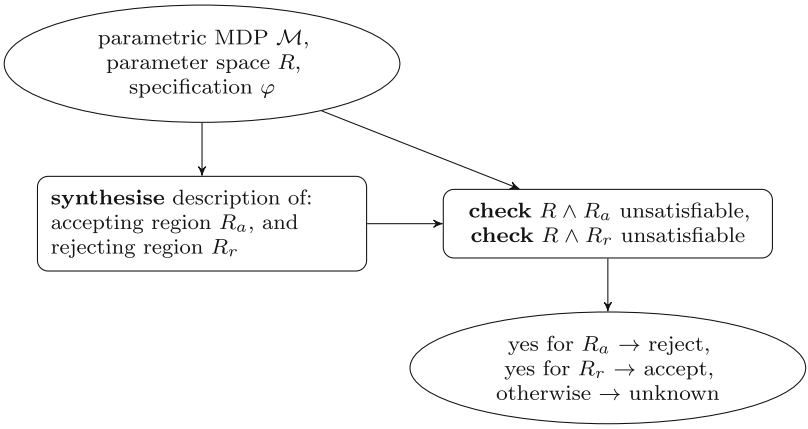


Fig. 3 Verification via exact synthesis

Example 6 (**Non-linear equations for reachability**) Take the MC from Fig. 1a. To compute the probability of eventually reaching, e.g., state ⚀, one introduces a variable p_s for each transient state s encoding that probability for s. For state s_0 and variable p_{s_0}, the corresponding linear equation reads:

$$p_{s_0} = 2/5 \cdot p_{s_1} + 3/5 \cdot p_{s_2},$$

where p_{s_1} and p_{s_2} are the variables for s_1 and s_2, respectively.

The corresponding equation for the pMC from Fig. 1b reads:

$$p_{s_0} = p \cdot p_{s_1} + (1 - p) \cdot p_{s_2}.$$

The multiplication of parameters in the model and equation variables leads to a *non-linear* equation system.

Thus, we can describe the sets R_a and R_r colloquially as:

$$R_a, R_r = \{u \mid u \text{ satisfies a set of non-linear constraints}\}.$$

We provide further details on these constraint systems in Sect. 6.

A practical drawback of the resulting equation system is the substantial number of auxiliary variables p_s, one for each state in the pMC. A viable possibility for pMCs is to simplify the equations by (variants of) *state elimination* [64]. This procedure successively removes states from the pMC until only a start and final state (representing the reachability objective) remain that are connected by a transition whose label is (a mild variant of) the solution function f_φ that *exactly* describes the probability to reach a target state:

$$R_a = \{u \mid f_\varphi(u) > 0\} \quad \text{and} \quad R_r = \{u \mid f_{\neg\varphi}(u) > 0\}.$$

We recapitulate state elimination and present several alternatives in Sect. 5.

Verification. The basic approach to the verification problem is depicted in Fig. 3. We use a description of the accepting region as computed via the synthesis procedure above. Then, we combine the description of the accepting region with the region R to be verified, as follows: A region R accepts a specification, if $R \cap R_a = R$, or equivalently, if $R \cap R_r = \emptyset$. The existence of a rejecting instance in R is thus of relevance; if such a point does not exist, the region is accepting. Using R_a and R_r as obtained above, the query "is $R \cap R_r = \emptyset$?" can

be solved via *satisfiability modulo theories* (SMT) over non-linear arithmetic, checking the conjunction over the corresponding constraints for unsatisfiability. With the help of SMT solvers over this theory like Z3 [93], MathSAT [31], or SMT-RAT [52], this can be solved in a fully automated manner. This procedure is complete, and is computationally involved. Details of the procedure are discussed in Sect. 6.

Parameter lifting [122] is an alternative, *approximative* solution to the verification problem. Intuitively, this approach over-approximates R_r for a given R, by ignoring parameter dependencies. Region R is accepted if the intersection with the over-approximation of R_r is empty. This procedure is sound but may yield false negatives as a rejecting point may lie in the over-approximation but not in R_r. Tightening the over-approximation makes the approach complete. A major benefit of parameter lifting (details in Sect. 7 and Sect. 8) is that the intersection with the over-approximation of R_r can be investigated by standard probabilistic model-checking procedures. This applicability of mature tools results—as will be shown in Sect. 11—in a practically efficient procedure.

Approximate synthesis. We solve the approximate synthesis problem with an iterative synthesis loop. Here, the central issue is to obtain representations of R_a and R_r by simple regions. Our approach for this *parameter space partitioning* therefore iteratively obtains partial partitions of the parameter space. The main idea is to compute a sequence $\left(R_a^i\right)_i$ of simple accepting regions that successively extend each other. Similarly, an increasing sequence $\left(R_r^i\right)_i$ of simple rejecting regions is computed. The typical approach is to let R_a^{i+1} be the union of R_a^i, the approximations in the previous iteration, together with some accepting region with a simple representation. Rejecting regions are handled analogously. At the i-th iteration, $R_a^i \cup R_r^i$ is the covered fragment of the parameter space. The iterative approach halts when this fragment forms at least $c\,\%$ of the entire parameter space. Termination is guaranteed. In the limit, the accepting and rejecting regions converge to the exact solution, $\lim_{i\to\infty} R_a^i = R_a$ and $\lim_{i\to\infty} R_r^i = R_r$, under some mild constraints on the ordering of the regions R^i.

Figure 4 outlines a procedure to address the approximate synthesis problem. As part of our synthesis method, we algorithmically *guess* a (candidate) region R and guess whether it is accepting or rejecting. We then exploit one of our verification methods to verify whether R is indeed accepting (or rejecting). If it is not accepting (rejecting), we exploit this information together with any additional information obtained during verification to refine the candidate region. This process is repeated until an accepting or rejecting region results. We discuss the method and essential improvements in Sect. 9.

Example 7 Consider the pMC $\mathcal{D}$ and the specification φ as in Example 2. The parameter space in Figure 2 is partitioned into regions. The green (dotted) area—the union of a number of smaller rectangular accepting regions—indicates the parameter values for which φ is satisfied, whereas the red (hatched) area indicates the set of rejecting regions for φ. Checking whether a region is accepting, rejecting, or inconsistent is done by verification. The small white area consists of regions that are *unknown* (i.e., not yet considered) or inconsistent.

1.3 Overview of the paper

Section 2 introduces the required formalisms and concepts. Section 3 defines the notion of a region and formalises the three problems: the verification problem and the two synthesis problems. It ends with a bird's eye view of the verification approaches that are later discussed in detail. Section 4 details specific region structures and procedures to check elementary

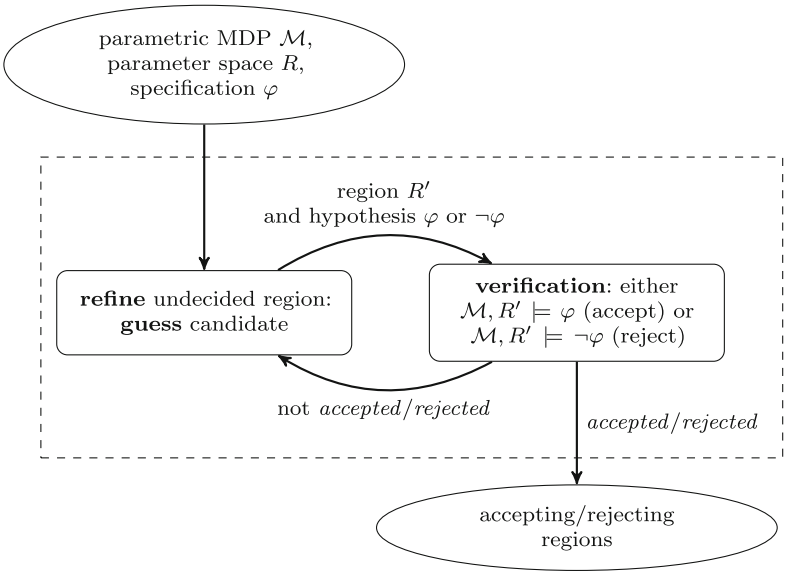


Fig. 4 Approximate synthesis process using verification as black box

region properties such as well-definedness and graph-preservedness, two prerequisites for the verification procedures. Section 5 shows how to do exact synthesis by computing the solution function. Sections 6–8 present algorithms for the verification problem. Section 9 details the approach to reduce the synthesis problem to a series of verification problems. Sections 10 and 11 contain information about the implementation of the approaches, as well as an extensive experimental evaluation. Section 12 contains a discussion of the approaches and related work. Section 13 concludes with an outlook.

1.4 Contributions of this paper

The paper is loosely based on the conference papers [65] and [122] and extends these works in the following ways. It gives a uniform treatment of the solution techniques to the synthesis problem, and treats all techniques uniformly for all different objectives—bounded and unbounded reachability as well as expected reward specifications. The material on SMT-based region verification has been extended in the following way: The paper gives the complete characterisations of the SMT encoding with or without solution function. Furthermore, it is the first to extend this encoding to MDPs under angelic and demonic non-determinism and includes an explicit and in-depth discussion on exact region checking via SMT checkers. It presents a uniform treatment of the linear equation system for Markov chains and its relation to state elimination and Gaussian elimination. It presents a novel and simplified description of state elimination for expected rewards, and a version of state elimination that is targeted towards MTBDDs. The paper contains a correctness proof of approximate verification for a wider range of pMDPs and contains proofs for expected rewards. It also supports expected-time properties for parametric continuous-time MDPs (via the embedded pMDP).

Novel heuristics have been developed to improve the iterative synthesis loop. All presented techniques, models, and specifications are realised in the state-of-the-art tool PROPhESY[1].

2 Preliminaries

2.1 Basic notations

We denote the set of real numbers by $\mathbb{R}$, the rational numbers by $\mathbb{Q}$, and the natural numbers including 0 by $\mathbb{N}$. Let $[0, 1] \subseteq \mathbb{R}$ denote the *closed interval* of all real numbers between 0 and 1, including the bounds; $(0, 1) \subseteq \mathbb{R}$ denotes the *open interval* of all real numbers between 0 and 1 excluding 0 and 1.

Let X, Y denote arbitrary sets. If $X \cap Y = \emptyset$, we write $X \uplus Y$ for the *disjoint union* of the sets X and Y. We denote the *power set* of X by $2^X = \{X' \mid X' \subseteq X\}$. Let X be a finite or countably infinite set. A *probability distribution* over X is a function $\mu \colon X \to [0, 1] \subseteq \mathbb{R}$ with $\sum_{x \in X} \mu(x) = \mu(X) = 1$.

2.2 Polynomials, rational functions

Let V denote a finite set of parameters over $\mathbb{R}$ and $\mathrm{dom}(p) \subseteq \mathbb{R}$ denote the domain of parameter $p \in V$.

Definition 1 (*Polynomial, rational function*) For a finite set $V = \{p_1, \ldots, p_n\}$ of n parameters, a *monomial* m is

$$m = p_1^{e_1} \cdot \ldots \cdot p_n^{e_n} \text{ with } e_i \in \mathbb{N}.$$

Let $Mon[V]$ denote the set of monomials over V. A *polynomial* g (over V) with t *terms* is a weighted sum of monomials:

$$g = \sum_{j=1}^{t} a_j \cdot m_j \text{ with } a_j \in \mathbb{Q} \setminus \{0\},\ m_j \in Mon[V].$$

Let $\mathbb{Q}[V]$ be the set of polynomials over V. A *rational function* $f = \frac{g_1}{g_2}$ over V is a fraction of polynomials $g_1, g_2 \in \mathbb{Q}[V]$ with $g_2 \not\equiv 0$ (where $\equiv$ states equivalence). Let $\mathbb{Q}(V)$ be the set of rational functions over V.

A monomial is *linear*, if $\sum_{i=1}^{|V|} e_i \leq 1$, and *multi-linear*, if $e_i \leq 1$ for all $1 \leq i \leq |V|$. A polynomial g is (multi-)linear, if all monomials occurring in g are (multi-)linear.

Instantiations replace parameters by constant values in polynomials or rational functions.

Definition 2 (*Parameter instantiations*) A *(parameter) instantiation* u *of parameters* V is a function $u \colon V \to \mathbb{R}$.

We abbreviate the parameter instantiation u with $u(p_i) = a_i \in \mathbb{R}$ by the n-dimensional vector $(a_1, \ldots, a_n) \in \mathbb{R}^n$ for ordered parameters $p_1, \ldots, p_n$. Applying the instantiation u on V to polynomial $g \in \mathbb{Q}[V]$ yields $g[u]$ which is obtained by replacing each $p \in V$ in g by $u(p)$, with subsequent application of $+$ and $\cdot$. For rational function $f = \frac{g_1}{g_2}$, let $f[u] = \frac{g_1[u]}{g_2[u]} \in \mathbb{R}$ if $g_2[u] \not\equiv 0$, and otherwise $f[u] = \bot$.

[1] PROPhESY is available on https://github.com/moves-rwth/prophesy.

2.3 Probabilistic models

Let us now introduce the probabilistic models used in this paper. We first define parametric Markov models and present conditions such that their instantiations result in Markov models with constant probabilities. Then, we discuss how to resolve non-determinism in decision processes.

2.3.1 Parametric Markov models

The transitions in parametric Markov models are equipped with rational functions over the set of parameters. Although this is the general setting, for some of our algorithmic techniques we will restrict ourselves to linear polynomials[2]. We consider parametric MCs and MDPs as sub-classes of a parametric version of classical two-player stochastic games [128]. The state space of such games is partitioned into two parts, $S_{\bigcirc}$ and $S_{\Box}$. At each state, a player chooses an action upon which the successor state is determined according to the (parametric) probabilities. Choices in $S_{\bigcirc}$ and $S_{\Box}$ are made by player $\bigcirc$ and $\Box$, respectively. pMDPs and pMCs are parametric stochastic one- and zero-player games respectively.

Definition 3 (*Parametric models*) A *parametric stochastic game (pSG)* is a tuple $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ with a finite set S of states with $S = S_{\bigcirc} \uplus S_{\Box}$, a finite set V of parameters over $\mathbb{R}$, an initial state $s_I \in S$, a finite set Act of actions, and a transition function $\mathcal{P}\colon S \times Act \times S \to \mathbb{Q}(V) \cup \mathbb{R} \cup \{\bot\}$ with $|Act(s)| \geq 1$ for all $s \in S$, where $Act(s) = \{\alpha \in Act \mid \exists s' \in S.\, \mathcal{P}(s, \alpha, s') \not\equiv 0\}$ is the set of *enabled* actions at state s.

- A pSG is a *parametric Markov decision process (pMDP)* if $S_{\bigcirc} = \emptyset$ or $S_{\Box} = \emptyset$.
- A pMDP is a *parametric Markov chain (pMC)* if $|Act(s)| = 1$ for all $s \in S$.

A parametric state-action *reward function* $\mathrm{rew}\colon S \times Act \to \mathbb{Q}(V) \cup \mathbb{R} \cup \{\bot\}$ associates rewards with state-action pairs[3]. It is assumed that deadlock states are absent, i.e., $Act(s) \neq \emptyset$ for all $s \in S$. Entries in $\mathbb{R} \cup \{\bot\}$ in the co-domains of the functions $\mathcal{P}$ and rew ensure that the model is closed under instantiations, see Definition 5 below. Throughout the rest of this paper, we silently assume that any given pSGs only uses constants from $\mathbb{Q}$ and rational functions $\mathbb{Q}(V)$, but no elements from $\mathbb{R} \setminus \mathbb{Q}$ or $\bot$. A model is called *parameter-free* if all its transition probabilities are constant.

A pSG intuitively works as follows. In state $s \in S_{\bigcirc}$, player $\bigcirc$ non-deterministically selects an action $\alpha \in Act(s)$. With (parametric) probability $\mathcal{P}(s, \alpha, s')$ the play then evolves to state s'. On leaving state s via action α, the reward $\mathrm{rew}(s, \alpha)$ is earned. If $s \in S_{\Box}$, the choice is made by player $\Box$, and as for player $\bigcirc$, the next state is determined in a probabilistic way. As by assumption no deadlock states occur, this game goes on forever. A pMDP is a game with one player, whereas a pMC has no players; a pMC thus evolves in a fully probabilistic way. Let $\mathcal{D}$ denote a pMC, $\mathcal{M}$ a pMDP, and $\mathcal{G}$ a pSG.

Example 8 Figure 5a–c depict a pSG, a pMDP, and a pMC respectively over parameters $V = \{p, q\}$. The states of the players $\bigcirc$ and $\Box$ are drawn as circles and rectangles, respectively. The initial state is indicated by an incoming arrow without source. We omit actions in state s if $|Act(s)| = 1$. In state s_0 of Fig. 5a, player $\bigcirc$ can select either action α or β. On selecting α, the game moves to state s_1 with probability p, and to s_2 with probability $1-p$. In state s_2, player $\Box$ can select α or β; in s_1 there is a single choice only.

[2] Most models use only simple polynomials such as p and $1-p$, and benchmarks available e.g., at the PRISM benchmark suite [103] or at the PARAM [77] web page are of this form.

[3] Recall that $\bot$ represents, e.g., $1/0$.

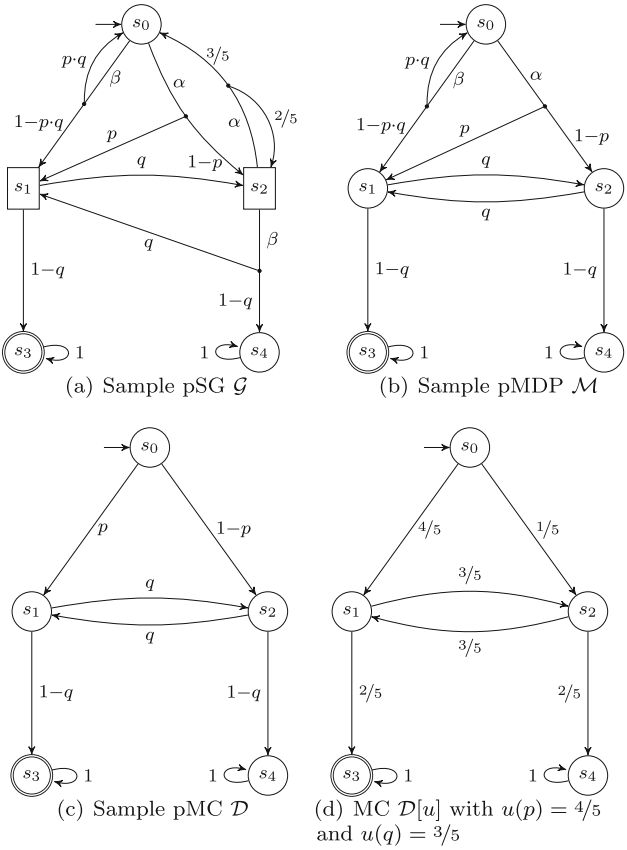


Fig. 5 The considered types of parametric probabilistic models (**a**)–(**c**) and an instantiated model (**d**)

A *transition* (s, α, s') exists if $\mathcal{P}(s, \alpha, s') \not\equiv 0$. As pMCs have a single enabled action at each state, we omit this action and just write $\mathcal{P}(s, s')$ for $\mathcal{P}(s, \alpha, s')$ if $Act(s) = \{\alpha\}$. A state s' is a *successor of* s, denoted $s' \in \text{succ}(s)$, if $\mathcal{P}(s, \alpha, s') \not\equiv 0$ for some α; in this case, $s \in \text{pred}(s')$ is a predecessor of s'.

Remark 1 Parametric stochastic games are the most general model used in this paper. They subsume pMDPs and pMCs and parameter-free SGs, which are used throughout this paper. We concisely introduce the formal foundations on this general class and indicate how these apply to subclasses. Most algorithmic approaches in this paper are not directly applicable to pSGs, but tailored to either pMDPs or pMCs. This is indicated when introducing these techniques.

Definition 4 (*Stochastic game*) A pSG $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ is a *stochastic game (SG)* if $\mathcal{P}\colon S \times Act \times S \to [0, 1]$ and $\sum_{s' \in S} \mathcal{P}(s, \alpha, s') = 1$ for all $s \in S$ and $\alpha \in Act(s)$.

A state-action reward function $\text{rew}\colon S \times Act \to \mathbb{R}_{\geq 0}$ associates (non-negative, finite) rewards to outgoing actions. Analogously, Markov chains (MCs) and Markov decision processes (MDPs) are defined as special cases of pMCs and pMDPs, respectively. We use D to denote a MC, M for an MDP and G for an SG.

2.3.2 Paths and reachability

An *infinite path* of a pSG $\mathcal{G}$ is an infinite sequence $\pi = s_0\alpha_0 s_1\alpha_1 \ldots$ of states $s_i \in S$ and actions $\alpha_i \in Act(s_i)$ with $\mathcal{P}(s_i, \alpha_i, s_{i+1}) \not\equiv 0$ for $i \geq 0$. A *finite path* of a pSG $\mathcal{G}$ is a non-empty finite prefix $s_0\alpha_0 \ldots s_n$ of an infinite path $s_0\alpha_0 \ldots s_n\alpha_n \ldots$ of $\mathcal{G}$ for some $n \in \mathbb{N}$. Let $Paths^{\mathcal{G}}$ denote the set of all finite or infinite paths of $\mathcal{G}$ while $Paths^{\mathcal{G}}_{fin} \subseteq Paths^{\mathcal{G}}$ denotes the set of all finite paths. For paths in (p)MCs, we omit the actions. The set $Paths^{\mathcal{G}}(s)$ contains all paths that start in state $s \in S$. For a finite path $\pi \in Paths^{\mathcal{G}}_{fin}$, $last(\pi) = s_n$ denotes the last state of π. The *length* $|\pi|$ of a path π is $|\pi| = n$ for $\pi \in Paths^{\mathcal{G}}_{fin}$ and $|\pi| = \infty$ for infinite paths. The *accumulated reward* along the finite path $s_0\alpha_0 \ldots \alpha_{n-1}s_n$ is given by the sum of the rewards $\mathrm{rew}(s_i, \alpha_i)$ for $0 \leq i < n$.

We denote the set of states that can reach a set of states T as follows: $\Diamond T = \{s \in S \mid \exists \pi \in Paths^{\mathcal{G}}_{fin}(s).\ last(\pi) \in T\}$. A set of states $T \subseteq S$ is *reachable* from $s \in S$, written $s \in \Diamond T$, iff there is a path from s to some $s' \in T$. A state s is *absorbing* iff $\mathcal{P}(s, \alpha, s) = 1$ for all $\alpha \in Act(s)$.

Example 9 The pMC in Fig. 5c has a path $\pi = s_0 s_1 s_3 s_3$ with $|\pi| = 3$. Thus $s_0 \in \Diamond\{s_3\}$. There is no path from s_4 to s_3, so $s_4 \notin \Diamond\{s_3\}$. States s_3 and s_4 are the only absorbing states.

2.3.3 Model instantiation

Instantiated parametric models are obtained by instantiating the rational functions in all transitions as in Definition 2.

Definition 5 (*Instantiated pSG*) For a pSG $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ and instantiation u of V, the *instantiated pSG at* u is given by $\mathcal{G}[u] = (S, s_I, Act, \mathcal{P}[u])$ with $\mathcal{P}[u](s, \alpha, s') = \mathcal{P}(s, \alpha, s')[u]$ for all $s, s' \in S$ and $\alpha \in Act$.

The instantiation of the parametric reward function rew at u is $\mathrm{rew}[u]$ with $\mathrm{rew}[u](s, \alpha) = \mathrm{rew}(s, \alpha)[u]$ for all $s \in S, \alpha \in Act$. Instantiating pMDP $\mathcal{M}$ and pMC $\mathcal{D}$ at u is denoted by $\mathcal{M}[u]$ and $\mathcal{D}[u]$, respectively.

Remark 2 The instantiation of a pSG at u is a pSG, but not necessarily an SG. This is due to the fact that an instantiation does not ensure that $\mathcal{P}(s, \alpha, \cdot)$ is a probability distribution. In fact, instantiation yields a transition function of the form $\mathcal{P}\colon S \times Act \times S \to \mathbb{R} \cup \{\bot\}$. Similarly, there is no guarantee that the rewards $\mathrm{rew}[u]$ are non-negative. Therefore, we impose restrictions on the parameter instantiations.

Definition 6 (*Well-defined instantiation*) An instantiation u is *well-defined* for a pSG $\mathcal{G}$ if the pSG $\mathcal{G}[u]$ is an SG.

The reward function rew is well-defined on u if it does only associate non-negative reals to state-action pairs.

Example 10 Consider again the pMC in Fig. 5c. The instantiation u with $u(p) = 4/5$ and $u(q) = 3/5$ is well-defined and induces the MC $\mathcal{D}[u]$ depicted in Fig. 5d.

From now on, we silently assume that every pSG we consider has at least one well-defined instantiation. This condition can be assured through checking the satisfiability of the conditions in Def. 4, which we discuss in Sect. 4.2.

Our methods necessitate instantiations that are not only well-defined, but also preserve the topology of the pSG. In particular, we are interested in the setting where reachability between two states coincides for the pSG and the set of instantiations u we consider. We detail this discussion in Sect. 4.2.

Definition 7 (*Graph preserving*) A well-defined instantiation u for pSG $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ is *graph preserving* if for all $s, s' \in S$ and $\alpha \in Act$,

$$\mathcal{P}(s, \alpha, s') \not\equiv 0 \implies \mathcal{P}(s, \alpha, s')[u] \in \mathbb{R} \setminus \{0\}.$$

Example 11 The well-defined instantiation u with $u(p) = 1$ and $u(q) = 3/5$ for the pMC in Fig. 5c is not *graph preserving*.

2.3.4 Resolving non-determinism

Strategies[4] resolve the non-deterministic choices in stochastic games with at least one player. For the objectives considered here, it suffices to consider so-called *deterministic* strategies [136]; more general strategies can be found in [13, Ch. 10]. We define strategies for pSGs and assume well-defined instantiations as in Definition 6.

Definition 8 (*Strategy*) A (deterministic) *strategy* σ_i for player $i \in \{\bigcirc, \square\}$ in a pSG $\mathcal{G}$ with state space $S = S_\bigcirc \uplus S_\square$ is a function

$$\sigma_i : \{\pi \in Paths^{\mathcal{G}}_{fin} \mid last(\pi) \in S_i\} \to Act$$

such that $\sigma_i(\pi) \in Act(last(\pi))$. Let $Str^{\mathcal{G}}$ denote the set of strategies $\sigma = (\sigma_\bigcirc, \sigma_\square)$ for pSG $\mathcal{G}$ and $Str^{\mathcal{G}}_i$ the set of strategies of player i.

A pMDP has only a player-i strategy for the player with $S_i \neq \emptyset$; in this case the index i is omitted. A player-i strategy σ_i is *memoryless* if $last(\pi) = last(\pi')$ implies $\sigma_i(\pi) = \sigma_i(\pi')$ for all finite paths π, π'. A memoryless strategy can thus be written in the form $\sigma_i : S_i \to Act$. A pSG-strategy $\sigma = (\sigma_\bigcirc, \sigma_\square)$ is memoryless if both $\sigma_\bigcirc$ and $\sigma_\square$ are memoryless.

Remark 3 From now on, we only consider memoryless strategies and refer to them as strategies.

A strategy σ for a pSG resolves all non-determinism and results in an *induced pMC*.

Definition 9 (*Induced pMC*) The *pMC* $\mathcal{G}^\sigma$ *induced by strategy* $\sigma = (\sigma_\bigcirc, \sigma_\square)$ on pSG $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ equals (S, V, s_I, P^σ) with:

$$P^\sigma(s, s') = \begin{cases} \mathcal{P}(s, \sigma_\bigcirc(s), s') & \text{if } s \in S_\bigcirc \\ \mathcal{P}(s, \sigma_\square(s), s') & \text{if } s \in S_\square. \end{cases}$$

Example 12 Let σ be a strategy for the pSG $\mathcal{G}$ in Fig. 5a with $\sigma_\bigcirc(s_0) = \alpha$ and $\sigma_\square(s_2) = \beta$. The induced pMC $\mathcal{G}^\sigma$ equals pMC $\mathcal{D}$ in Fig. 5c. Analogously, imposing strategy σ' with $\sigma'(s_0) = \alpha$ on the pMDP in Fig. 5b yields $\mathcal{M}^{\sigma'} = \mathcal{D}$.

The notions of strategies for pSGs and pMDPs and of induced pMCs naturally carry over to non-parametric models; e.g., the MC G^σ is induced by strategy $\sigma \in Str^G$ on SG G.

[4] Also referred to as policies, adversaries, or schedulers.

2.4 Specifications and solution functions

2.4.1 Specifications

Specifications constrain the measures of interest for (parametric) probabilistic models. Before considering parameters, let us first consider MCs. Let $D = (S, s_I, \mathcal{P})$ be an MC and $T \subseteq S$ a set of *target states* that (without loss of generality) are assumed to be absorbing. Let $\Diamond T$ denote the path property to reach T[5]. Furthermore, the *probability measure* $\Pr_s$ over sets of paths can be defined using a cylinder construction with $\Pr_s(s_0\alpha_0 \dots s_n) = \Pi_{i=0}^{n-1}\mathcal{P}(s_i, \alpha_i, s_{i+1})$, see [13, Ch. 10].
We consider three kinds of specifications:

1. *Unbounded probabilistic reachability* A specification $\mathbb{P}_{\leq\lambda}(\Diamond T)$ asserts that the probability to reach T from the initial state s_I shall be at most λ, where $\lambda \in \mathbb{Q} \cap [0, 1]$. More generally, specification φ^r is satisfied by MC D, written:
$$D \models \mathbb{P}_{\sim\lambda}(\Diamond T) \quad \text{iff} \quad \Pr^D_{s_I}(\Diamond T) \sim \lambda,$$
where $\Pr^D_{s_I}(\Diamond T)$ is the probability mass of all infinite paths that start in s_I and visit any state from T.
2. *Bounded probabilistic reachability* In addition to reachability, these specifications impose a bound on the maximal number of steps until reaching a target state. Specification $\varphi^b = \mathbb{P}_{\sim\lambda}(\Diamond^{\leq n} T)$ asserts that in addition to $\mathbb{P}_{\sim\lambda}(\Diamond T)$, states in T should be reached within $n \in \mathbb{N}$ steps. The satisfaction of $\mathbb{P}_{\sim\lambda}(\Diamond^{\leq n} T)$ is defined similar as above.
3. *Expected reward until a target* The specification $\mathbb{E}_{\leq\kappa}(\Diamond T)$ asserts that the expected reward until reaching a state in T shall be at most $\kappa \in \mathbb{R}$. Let $\mathrm{ER}^D_{s_I}(\Diamond T)$ denote the expected accumulated reward until reaching a state in $T \subseteq S$ from state s_I. We obtain this reward by multiplying the probability of every path reaching T with the accumulated reward of that path, up until reaching T. Details are given in [13, Chapter 10]. [6]. Then we define
$$D \models \mathbb{E}_{\sim\kappa}(\Diamond T) \quad \text{iff} \quad \mathrm{ER}^D_{s_I}(\Diamond T) \sim \kappa,$$
We do not treat the accumulated reward to reach a target within n steps, as this is not a very useful measure. In case there is a possibility to not reach the target within n steps, this yields ∞.

We omit the superscript D if it is clear from the context. We write $\neg\varphi$ to invert the relation: $D \models \neg\mathbb{P}_{\leq\lambda}(\Diamond T)$ is thus equivalent to $D \models \mathbb{P}_{>\lambda}(\Diamond T)$. An SG G satisfies specification φ under strategy σ if the induced MC $G^\sigma \models \varphi$. Unbounded reachability and expected rewards are prominent examples of *indefinite-horizon* properties – they measure behaviour up-to some specified event (the horizon) which may be reached after arbitrarily many steps.

Remark 4 Bounded reachability in MDPs can be reduced to unbounded reachability by a technique commonly referred to as *unrolling* [5]. For performance reasons, it is sometimes better to avoid this unrolling, and present dedicated approaches.

[5] Thereby overloading the earlier notation to denote the set of states for which there exists a path on which this property holds.

[6] As standard, if $\Pr^D_{s_I}(\Diamond T) < 1$ then we set $\mathrm{ER}^D_{s_I}(\Diamond T) := \infty$. The rationale is that an infinite amount of reward is collected on visiting a state (with positive reward) infinitely often from which all target states are unreachable.

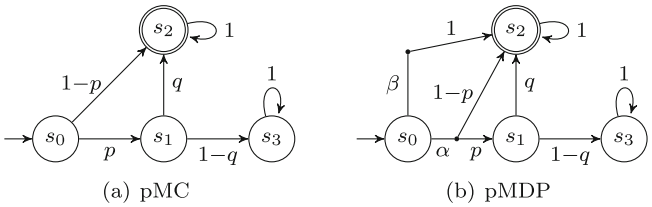


Fig. 6 Two sample parametric models

2.4.2 Solution functions

Computing (unbounded) reachability probabilities and expected rewards for MCs reduces to solving linear equation systems [13] over the field of reals (or rationals). For parametric MCs, we obtain a linear equation system over the field of the rational functions over V instead. The solution to this equation system is a rational function. (See Examples 4 and 6 on pages 6 and 7). More details on the the solution function and the equation system follow in Sects. 5 and 6, respectively.

Definition 10 (*Solution functions*) For a pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$, $T \subseteq S$ and $n \in \mathbb{N}$, a *solution function* for a specification φ is a rational function

$$\begin{aligned} f^r_{\mathcal{D},T} &\in \mathbb{Q}(V) \text{ for } \varphi = \mathbb{P}_{\sim\lambda}(\Diamond\, T) \\ f^b_{\mathcal{D},T,n} &\in \mathbb{Q}(V) \text{ for } \varphi = \mathbb{P}_{\sim\lambda}(\Diamond^{\leq n}\, T) \text{ , and} \\ f^e_{\mathcal{D},T} &\in \mathbb{Q}(V) \text{ for } \varphi = \mathbb{E}_{\sim\kappa}(\Diamond\, T), \end{aligned}$$

such that for every well-defined graph-preserving instantiation u:

$$\begin{aligned} f^r_{\mathcal{D},T}[u] &= \Pr^{\mathcal{D}[u]}_{s_I}(\Diamond\, T), \\ f^b_{\mathcal{D},T,n}[u] &= \Pr^{\mathcal{D}[u]}_{s_I}(\Diamond^{\leq n}\, T), \text{ and} \\ f^e_{\mathcal{D},T}[u] &= \mathrm{ER}^{\mathcal{D}[u]}_{s_I}(\Diamond\, T). \end{aligned}$$

Example 13 Consider the reachability probability to reach s_2 for the pMC in Fig. 6a. Any instantiation u with $u(p), u(q) \in (0, 1)$ is well-defined and graph-preserving. As the only two finite paths to reach s_2 are $s_0 s_2$ and $s_0 s_1 s_2$, we have $f^r_{\mathcal{D},\{s_2\}} = 1 - p + p \cdot q$.

For pSGs (and pMDPs), the solution function depends on the resolution of non-determinism by strategies, i. e., they are defined on the induced pMCs. Formally, a *solution function* for a pSG $\mathcal{G}$, a reachability specification $\varphi^r = \mathbb{P}_{\leq\lambda}(\Diamond\, T)$, and a strategy $\sigma \in Str^{\mathcal{G}}$ is a function $f^r_{\mathcal{G},\sigma,T} \in \mathbb{Q}(V)$ such that for each well-defined graph-preserving instantiations u it holds:

$$f^r_{\mathcal{G},\sigma,T}[u] = \Pr^{\mathcal{G}^\sigma[u]}_{s_I}(\Diamond\, T).$$

These notions are defined analogously for bounded reachability (denoted $f^b_{\mathcal{G},\sigma,T,n}$) and expected reward (denoted $f^e_{\mathcal{G},\sigma,T}$) specifications.

Example 14 For the pMDP in Fig. 6b, the solution functions for reaching s_2 are $1-p+p\cdot q$, for the strategy $\sigma_\alpha = \{s_0 \mapsto \alpha\}$, and 1 for the strategy $\sigma_\beta = \{s_0 \mapsto \beta\}$.

Remark 5 We define solution functions only for graph-preserving valuations. For the more general well-defined solutions, a similar definition can be given [94] where (solution) functions are no longer rational functions but instead a collection of solution functions obtained on the graph-preserving subsets. In particular, unless a pMC is acyclic, such a function is only semi-continuous [97]. A key reason for the discontinuity is the change of states that are in $\Diamond T$, e.g., consider instantiations with $q = 1$ in Fig. 5c. We provide the decomposition into graph-preserving subsets in Sect. 4.3.

2.5 Constraints and formulas

We consider *(polynomial) constraints* of the form $g \sim g'$ with $g, g' \in \mathbb{Q}[V]$ and $\sim \in \{<, \leq, =, \geq, >\}$. We denote the set of all constraints over V with $\mathcal{C}[V]$. A constraint $g \sim g'$ can be equivalently formulated as $g - g' \sim 0$. A *formula* ψ over a set of polynomial constraints is recursively defined: Each polynomial constraint is a formula, and the Boolean combination of formulae is also a formula.

Example 15 Let p, q be variables. $1 - p \cdot q > 0$ and $p^2 < 0$ are constraints, $\neg\left(p^2 < 0\right)$ and $(1 - p \cdot q > 0) \vee \left(p^2 < 0\right)$ are formulae.

The semantics of constraints are standard: i.e., an instantiation u satisfies $g \sim g'$ if $g[u] \sim g'[u]$. An instantiation satisfies $\psi \wedge \psi'$ if u satisfies both ψ and ψ'. The semantics for other Boolean connectives are defined analogously. Moreover, we will write $g \neq g'$ to denote the formula $g < g' \vee g > g'$.

Checking whether there exists an instantiation that satisfies a formula is equivalent to checking membership of the *existential theory of the reals* [21]. Such a check can be automated using SMT-solvers capable of handling quantifier-free non-linear arithmetic over the reals [93], such as [52, 63].

Statements of the form $f \sim f'$ with $f, f' \in \mathbb{Q}(V)$ are not necessarily polynomial constraints: however, we are not interested in instantiations u with $f[u] = \bot$, and thus later (in Sect. 4.2.2) we can transform such constraints into formulae over polynomial constraints.

3 Formal problem statements

This section formalises the three problem statements mentioned in the introduction: the verification problem and two synthesis problems. We start off by making precise what regions are and how to represent them. We then define what it means for a region to satisfy a given specification. This puts all in place to making the three problem statements precise. Finally, it surveys the verification approaches that are detailed later in the paper.

3.1 Regions

Instantiated parametric models are amenable to standard probabilistic model checking. However, *sampling* an instantiation is very restrictive—verifying an instantiated model gives results for a single point in the (uncountably large) parameter space. A more interesting problem is to determine which parts of the parameter space give rise to a model that complies with the specification. Such sets of parameter values are, inspired by their geometric interpretation, called *regions*. Regions are solution sets of conjunctions of constraints over the set V of parameters.

Definition 11 (*Region*) A *region* R over V is a set of instantiations of V (or dually a subset of $\mathbb{R}^{|V|}$) for which there exists a set $C(R) \subseteq \mathcal{C}[V]$ of polynomial constraints such that for their conjunction $\Phi(R) = \bigwedge_{c \in C(R)} c$ we have

$$R \;=\; \{u \mid \Phi(R)[u]\}.$$

We call $C(R)$ the *representation* of R.

Any region which is a subset of a region R is called a *subregion* of R.

Example 16 Let the region R over $V = \{p, q\}$ be described by

$$C(R) = \{p^2 + q^2 - 1 \leq 0,\ \ p + q - 1 \leq 0\}.$$

Thus, $R = \{\, u \mid (p^2+q^2-1)[u] \leq 0 \wedge (p+q-1)[u] \leq 0 \,\}$. The region R contains the instantiation $u = (2/5, 3/5)$ as $(2/5)^2 + (3/5)^2 - 1 \leq 0$ and $2/5 + 3/5 - 1 \leq 0$. The instantiation $u' = (1/2, 3/5) \notin R$ as $1/2 + 3/5 - 1 > 0$. Regions do not have to describe a contiguous area of the parameter space; e.g., consider the region R' described by $\{-p^2 + 1 < 0\}$ is $R' = (-\infty, -1] \cup [1, +\infty)$.

Regions are semi-algebraic sets [21] which yield the theoretical formalisation of notions such as distance, convexity, etc. It also ensures that regions are well-behaved: Informally, a region in the space $\mathbb{R}^n$ is given by a finite number of connected semialgebraic sets (*cells*[7]), and (the boundaries of) each cell can be described by a finite set of polynomials. The size $\|R\|$ of a region R is given by the Lebesgue measure. All regions are Lebesgue measurable.

A region is called well-defined if all its instantiations are well defined.

Definition 12 (*Well-defined region*) Region R is *well defined* for pSG $\mathcal{G}$ if for all $u \in R$, u is a well-defined valuation for $\mathcal{G}$.

3.2 Angelic and demonic satisfaction relations

As a next step towards our formal problem statements, we have to define what it means for a region to satisfy a specification. We first introduce two satisfaction relations—angelic and demonic—for parametric Markov models for a single instantiation. We then lift these two notions to regions.

Definition 13 (*Angelic and demonic satisfaction relations*) For pSG $\mathcal{G}$, well-defined instantiation u, and specification φ, the *satisfaction relations* $\models_a$ *and* $\models_d$ are defined by:

$$\mathcal{G}, u \models_a \varphi \quad \text{iff} \quad \exists \sigma \in \mathit{Str}^{\mathcal{G}}.\ \mathcal{G}[u]^{\sigma} \models \varphi \quad \text{(angelic)}$$
$$\mathcal{G}, u \models_d \varphi \quad \text{iff} \quad \forall \sigma \in \mathit{Str}^{\mathcal{G}}.\ \mathcal{G}[u]^{\sigma} \models \varphi \quad \text{(demonic)}.$$

The angelic relation $\models_a$ refers to the *existence* of a strategy to fulfil the specification φ, whereas the demonic counterpart $\models_d$ requires *all* strategies to fulfil φ. Observe that $\mathcal{G}, u \not\models_a \varphi$ if and only if $\mathcal{G}, u \models_d \neg\varphi$. Thus, demonic and angelic can be considered to be dual. By $\models_{\heartsuit}$ we denote the dual of $\models_{\clubsuit}$, that is, if $\clubsuit = a$ then $\heartsuit = d$ and vice versa. For pMCs, the relations $\models_a$ and $\models_d$ coincide and the subscripts a and d are omitted.

[7] Connected here intuitively refers to the fact that you can draw a path from two points in a cell that never leaves the cell.

Example 17 Consider the pMDP $\mathcal{M}$ in Fig. 6b, instantiation $u = (1/2, 1/2)$ and $\varphi = \mathbb{P}_{>4/5}(\Diamond\{s_2\})$. We have $\mathcal{M}, u \models_a \varphi$, as for strategy $\sigma_\beta = \{s_0 \mapsto \beta\}$ the state s_2 is reached with probability one; thus, $\mathcal{M}[u]^{\sigma_\beta} \models \varphi$. However, $\mathcal{M}, u \not\models_d \varphi$, as for strategy $\sigma_\alpha = \{s_0 \mapsto \alpha\}$, we have $(1-p+p\cdot q)[u] = 3/4 \not> 4/5$; thus, $\mathcal{M}[u]^{\sigma_\alpha} \not\models \varphi$. By duality, $\mathcal{M}, u \models_a \neg\varphi$.

We now lift these two satisfaction relations to regions. The aim is to consider specifications φ that hold *for all instantiations* represented by a region R of a parametric model $\mathcal{G}$. This is captured by the following satisfaction relation.

Definition 14 (*Satisfaction relation for regions*) For pSG $\mathcal{G}$, well-defined region R, and specification φ, the relation $\models_\clubsuit$, $\clubsuit \in \{a, d\}$, is defined as:

$$\mathcal{G}, R \models_\clubsuit \varphi \quad \text{iff} \quad \mathcal{G}, u \models_\clubsuit \varphi \text{ for all } u \in R.$$

Before we continue, we note the difference between $\mathcal{G}, R \not\models_\clubsuit \varphi$ and $\mathcal{G}, R \models_\clubsuit \neg\varphi$:

$$\mathcal{G}, R \models_\clubsuit \neg\varphi \text{ implies } \mathcal{G}, u \models_\clubsuit \neg\varphi \text{ for } \textit{all } u \in R,$$

whereas in constrast,

$$\mathcal{G}, R \not\models_\clubsuit \varphi \text{ implies } \mathcal{G}, u \not\models_\clubsuit \varphi \text{ for } \textit{some } u \in R.$$

Definition 15 (*Accepting/rejecting/inconsistent region*) A well-defined region R is *accepting* (for $\mathcal{G}, \varphi, \clubsuit$) if $\mathcal{G}, R \models_\clubsuit \varphi$. Region R is *rejecting* (for $\mathcal{G}, \varphi, \clubsuit$) if $\mathcal{G}, R \models_\heartsuit \neg\varphi$. Region R is *inconsistent* if it is neither accepting nor rejecting.

By the duality of $\models_a$ and $\models_d$, a region is thus rejecting iff $\forall u \in R.\ \mathcal{G}, u \not\models_\clubsuit \varphi$. Note that this differs from $\mathcal{G}, R \not\models_\clubsuit \varphi$.

Example 18 Reconsider the pMDP in Fig. 6b, with $R = [2/5, 1/2] \times [2/5, 1/2]$ and $\varphi = \mathbb{P}_{>4/5}(\Diamond\{s_2\})$. The corresponding solution functions are given in Example 14. It follows that:

- $\mathcal{M}, R \models_a \varphi$, as for strategy $\sigma_\beta = \{s_0 \mapsto \beta\}$, we have $\mathcal{M}^{\sigma_\beta}, u \models \varphi$ for all $u \in R$.
- $\mathcal{M}, R \not\models_d \varphi$, as for strategy $\sigma_\alpha = \{s_0 \mapsto \alpha\}$, $\mathcal{M}^{\sigma_\alpha}, u \not\models \varphi$ for $u = (1/2, 1/2)$.
- $\mathcal{M}, R \models_a \neg\varphi$ using strategy σ_α.

Regions can be inconsistent w. r. t. a relation, and consistent w. r. t. its dual relation. The region $(0, 1) \times (0, 1)$ is inconsistent for $\mathcal{M}$ and $\models_d$, as for both φ and $\neg\varphi$, there is a strategy that is not accepting. For $\models_a$, there is a single strategy which accepts φ; other strategies do not affect the relation.

As an example of an accepting region under the demonic relation, consider $R' = [4/5, 9/10] \times [2/5, 9/10]$. We have $\mathcal{M}, R' \models_d \varphi$, as for both strategies, the induced probability is always exceeding $4/5$.

3.3 Formal problem statements

We are now in a position to formalise the two synthesis problems and the verification problem from the introduction, page 5. We present the formal problem statements in the order of treatment in the rest of the paper.

The formal synthesis problem. Given pSG $\mathcal{G}$, specification φ, and well-defined region R, the *synthesis problem* is to partition R into R_a and R_r such that:

$$\mathcal{G}, R_a \models_{\clubsuit} \varphi \quad \text{and} \quad \mathcal{G}, R_r \models_{\heartsuit} \neg\varphi.$$

This problem is the topic of **Section 5**.

Remark 6 The solution function for pMCs precisely describes how (graph-preserving) instantiations map to the relevant measure. Therefore, comparing the solution function with the threshold divides the parameter space into an accepting region R_a and a rejecting region R_r and defines the exact result for the formal synthesis problem. Recall also Example 4.

The formal verification problem. Given pSG $\mathcal{G}$, specification φ, and well-defined region R, the *verification problem* is to check whether:

$$\begin{aligned} &\mathcal{G}, R \models_{\clubsuit} \varphi && (R \text{ is accepting}) \\ \text{or } &\mathcal{G}, R \models_{\heartsuit} \neg\varphi && (R \text{ is rejecting}) \\ \text{or } &\mathcal{G}, R \not\models_{\clubsuit} \varphi \ \wedge\ \mathcal{G}, R \not\models_{\heartsuit} \neg\varphi && (R \text{ is inconsistent}) \end{aligned}$$

where $\models_{\heartsuit}$ denotes the dual satisfaction relation of $\models_{\clubsuit}$.
This problem is the topic of **Section 6–8**.

The verification procedure allows us to utilise an approximate synthesis problem in which verification procedures are used as a backend.

The formal approximate synthesis problem. Given pSG $\mathcal{G}$, specification φ, percentage c, and well-defined region R, the *approximate synthesis problem* is to partition R into regions R_a, R_o, and R_r such that:

$$\mathcal{G}, R_a \models_{\clubsuit} \varphi \quad \text{and} \quad \mathcal{G}, R_r \models_{\heartsuit} \neg\varphi,$$

where $R_a \uplus R_r$ cover at least $c\%$ of the region R.
This problem is the topic of **Section 9**.

Note that no requirements are imposed on the (unknown, open) region R_o.

Remark 7 By definition, the angelic satisfaction relation for region R and pSG $\mathcal{G}$ is equivalent to:

$$\mathcal{G}, R \models_a \varphi \quad \text{if and only if} \quad \forall u \in R.\ \exists \sigma \in Str^{\mathcal{G}}.\ \mathcal{G}^{\sigma}, u \models \varphi.$$

An alternative notion in parameter synthesis is the existence of a *robust* strategy:

$$\exists \sigma \in Str^{\mathcal{G}}.\ \forall u \in R.\ \mathcal{G}^{\sigma}, u \models \varphi.$$

Note the swapping of quantifiers compared to $\models_a$. That is, $\mathcal{G}, R \models_a \varphi$ considers potentially different strategies for different parameter instantiations $u \in R$. The notion of robust strategies leads to a series of quite orthogonal challenges. For instance, the notion is not compositional, i.e., if robust strategies exist in R_1 and R_2, then we *cannot* conclude the existence of a robust strategy in $R_1 \cup R_2$. Moreover, memoryless strategies are not sufficient, see [9]. Robust strategies are outside the scope of this paper and are only shortly mentioned in Sect. 8.

3.4 A bird's eye view on the verification procedures

In the later sections, we will present several techniques that decide the verification problem for pMCs and pMDPs. (Recall that stochastic games were only used to define the general setting.)

The verification problem is used to analyse the regions of interest. The assumption that this region contains only well-defined instantiations is therefore natural. It can be checked algorithmically as described in Sect. 4.2 below. Many verification procedures require that the region is graph preserving. A decomposition result of well-defined into graph-preserving regions is given in Sect. 4.3.

Section 6 presents two verification procedures. The first one directly solves the non-linear equation system, see Example 6, as an SMT query. The second procedure reformulates the SMT query using the solution function. While this reformulation drastically reduces the number of variables in the query, it requires an efficient computation of the solution function, as described in Sect. 5.

Section 7 covers an *approximate* and more efficient verification procedure, called *parameter lifting*, which is tailored to multi-linear functions and closed rectangular regions. Under these mild restrictions, the verification problem for pMCs (pMDPs) can be approximated using a sequence of *standard verification analyses on non-parametric* MDPs (SGs) of similar size, respectively. The key steps here are to relax the parameter dependencies, and consider lower- and upper-bounds of parameters as worst and best cases.

4 Regions

Section 3.1 already introduced regions. This section details specific region structures such as linear, rectangular and graph-preserving regions. It then presents procedures to check whether a region is graph preserving. Finally, we describe how well-defined but not graph-preserving regions can be turned into several regions that are graph preserving.

4.1 Regions with specific structure

As defined before, a region R is a (typically uncountably infinite) set of parameter valuations described by a set $C(R)$ of polynomial constraints. Two classes of regions are particularly relevant: linear and rectangular regions.

Definition 16 (*Linear region*) A region with representation $C(R)$ is *linear* if for all $g \sim 0 \in C(R)$, the polynomial g is linear.

Linear regions describe convex polytopes. We refer to the vertices (or angular points) of the polytope as the *region vertices*.

Definition 17 (*Rectangular region*) A region R with representation

$$C(R) = \bigcup_{i=1}^{|V|} \{ -p_i + a_i \trianglelefteq_i^1 0,\ p_i + b_i \trianglelefteq_i^2 0 \}$$

with $a_i \leq b_i \in \mathbb{Q}$ and $\trianglelefteq_i^j \in \{<, \leq\}$ for $0 < i \leq |V|$ and $j \in \{1, 2\}$ is called *rectangular*. A rectangular region is *closed* if all inequalities $\trianglelefteq_i^j$ in the constraints in $C(R)$ are non-strict.

Rectangular regions are hyper-rectangles and a subclass of linear regions. A *closed* rectangular region R can be represented as $R = \times_{p\in V}[a_p, b_p]$ with parameter intervals $[a_p, b_p]$ described by the bounds a_p and b_p for all $p \in V$. For a region R, we refer to the *bounds* of parameter p by $B_R(p) = \{a_p, b_p\}$ and to the *interval* of parameter p by $I_R(p) = [a_p, b_p]$. We may omit the subscript R, if it is clear from the context. For a rectangular region R, the size $\|R\|$ equals $\prod_{p\in V}(b_p - a_p)$.

Regions represent sets of instantiations $\mathcal{G}[u]$ of a pSG $\mathcal{G}$. The notion of graph-preservation from Definition 7 lifts to regions in a straightforward manner:

Definition 18 (*Graph-preserving region*) Region R is *graph preserving* for pSG $\mathcal{G}$ if for all $u \in R$, u is a graph-preserving valuation for $\mathcal{G}$.

By this definition, all instantiations from graph-preserving regions have the same topology as the parametric model, cf. Remark 8 below. In addition, all such instantiations are well-defined.

Example 19 Let $\mathcal{D}$ be the pMC in Fig. 5c, $R = [1/10, 4/5] \times [2/5, 7/10]$ be a (closed rectangular) region, and instantiation $u = (4/5, 3/5) \in R$. Figure 5d depicts the instantiation $\mathcal{D}[u]$, an MC with the same topology as $\mathcal{D}$. As the topology is preserved for all possible instantiations $\mathcal{D}[u']$ with $u' \in R$, the region R is graph preserving. The region $R' = [0, 1] \times [0, 1]$ is not graph preserving as, e.g., the instantiation $(0, 0) \in R'$ results in an MC that has no transition from state s_1 to s_2.

Remark 8 Graph-preserving regions have the nice property that if

$$\exists u \in R, \mathcal{G}, u \models_{\clubsuit} \mathbb{P}_{=1}(\Diamond T) \text{ implies } \mathcal{G}, R \models_{\clubsuit} \mathbb{P}_{=1}(\Diamond T).$$

This property can be checked by standard graph analysis [13, Ch. 10]. It is thus straightforward to check $\mathcal{G}, R \models_{\clubsuit} \mathbb{P}_{=1}(\Diamond T)$, an important precondition for computing expected rewards. In the rest of this paper when considering expected rewards, it is assumed that within a region the probability to reach a target is one.

The following two properties of regions are frequently (and often implicitly) used in this paper.

Lemma 1 (Characterisation for inconsistent regions) *For any inconsistent region R it holds that $R = R_a \cup R_r$ for some accepting $R_a \neq \emptyset$ and rejecting $R_r \neq \emptyset$.*

Lemma 2 (Compositionality) *Region $R = R_1 \cup R_2$ is accepting (rejecting) if and only if both R_1 and R_2 are accepting (rejecting).*

The statements follow from the universal quantification over all instantiations in the definition of $\models_{\clubsuit}$.

4.2 Checking whether a region is graph preserving

The verification problem for region R requires R to be well-defined. We first address the problem on how to check this condition. In fact, we present a procedure to check graph preservation which is slightly more general and useful later, see also Remark 8. To show that region R is not graph preserving, a point in R suffices that violates the conditions in Definition 7. Using the representation of region R, the implication

$$\Phi(R) \implies R \text{ graph preserving}$$

needs to be valid since any violating assignment corresponds to a non-graph-preserving instantiation inside R. Technically, we consider satisfiability of the conjunction of:

– the inequalities $C(R)$ representing the candidate region, and
– a disjunction of (in)equalities describing violating graph-preserving.

This conjunction is satisfiable if and only if the region is not graph preserving.

4.2.1 Graph preservation for polynomial transition functions

Let us consider the above for pSGs with polynomial transition functions. The setting for pSGs with rational functions is discussed at the end of this section. The following constraints (1)–(4), which we denote *GP*, capture the notion of graph preservation:

$$\bigwedge_{\substack{s,s'\in S,\alpha\in Act(s)\\ \mathcal{P}(s,\alpha,s')\not\equiv 0}} 0 \leq \mathcal{P}(s,\alpha,s') \leq 1 \tag{1}$$

$$\wedge \bigwedge_{s\in S,\alpha\in Act(s)} \sum_{s'\in S} \mathcal{P}(s,\alpha,s') = 1 \tag{2}$$

$$\wedge \bigwedge_{s\in S,\alpha\in Act(s)} \mathrm{rew}(s,\alpha) \geq 0 \tag{3}$$

$$\wedge \bigwedge_{\substack{s,s'\in S,\alpha\in Act(s)\\ \mathcal{P}(s,\alpha,s')\not\equiv 0}} 0 < \mathcal{P}(s,\alpha,s'). \tag{4}$$

The constraints ensure that (1) all non-zero entries are evaluated to a probability, (2) transition probabilities are probability distributions, (3) rewards are non-negative, and (4) non-zero entries remain non-zero. The constraints (1)–(3) suffice to ensure well-definedness. The constrains (1)–(4) can be simplified to:

$$\bigwedge_{\substack{s,s'\in S,\alpha\in Act(s)\\ \mathcal{P}(s,\alpha,s')\not\equiv 0}} \mathcal{P}(s,\alpha,s') > 0$$

$$\wedge \bigwedge_{s\in S,\alpha\in Act(s)} \sum_{s'\in S} \mathcal{P}(s,\alpha,s') = 1$$

$$\wedge \bigwedge_{s\in S,\alpha\in Act(s)} \mathrm{rew}(s,\alpha) \geq 0.$$

Example 20 Recall the pMC from Fig. 5c.

$$GP = \; p > 0 \wedge 1-p > 0 \wedge p+1-p = 1 \wedge q > 0 \wedge 1-q > 0 \wedge q+1-q = 1.$$

This equation simplifies to $0 < p < 1 \wedge 0 < q < 1$. To check whether the region R described by $\Phi(R) = 1/10 \leq p \leq 4/5 \wedge 2/5 \leq q \leq 7/10$ is graph preserving, we check whether the conjunction $\Phi(R) \wedge \neg GP$ is satisfiable, with

$$\neg GP = \; p \leq 0 \vee p \geq 1 \vee q \leq 0 \vee q \geq 1.$$

As the conjunction is not satisfiable, the region R is graph preserving. Contrary, $R' = [0,1] \times [0,1]$ is not graph preserving as $u = (0,0)$ satisfies the conjunction $\Phi(R') \wedge \neg GP$.

Satisfiability of *GP*, or equivalently, deciding whether a region is graph preserving, is as hard as the existential theory of the reals [21], if no assumptions are made about the transition

probability and reward functions. This checking can be automated using SMT-solvers capable of handling quantifier-free non-linear arithmetic over the reals [93]. The complexity drops to polynomial time once both the region R and all transition probability (and reward) functions are linear as linear programming has a polynomial complexity and the formula is then a disjunction over linear programs (with trivial optimisation functions).

4.2.2 Graph preservation for rational transition functions

In case the transition probability and reward function of a pSG are not polynomials, the left-hand side of the statements in (1)–(4) are not polynomials, and the statements would not be constraints. We therefore perform the following transformations on (1)–(4):

- Transforming equalities:

$$\frac{g_1}{g_2} = c \quad \text{becomes} \quad g_1 - c \cdot g_2 = 0 \wedge g_2 \neq 0 \quad \text{with } c \in \mathbb{Q}.$$

- Transforming inequalities $\trianglerighteq \in \{>, \geq\}$:

$$\frac{g_1}{g_2} \trianglerighteq c \text{ becomes } g_2 \neq 0 \wedge \Big((g_2 > 0 \wedge g_1 \trianglerighteq c \cdot g_2) \vee (g_2 < 0 \wedge g_1 \not\trianglerighteq c \cdot g_2)\Big)$$

  with $c \in \mathbb{Q}$, and $\not\trianglerighteq$ equals $<$ for $\not>$ and $\leq$ for $\not\geq$.
- Transforming $<, \leq$ is analogous.
- Transforming $g \neq g'$ (i.e., $g < g' \vee g > g'$) involves transforming both disjuncts.

The result is a formula with polynomial constraints that correctly describes graph preservation (or well-definedness).

Example 21 Consider a state with outgoing transition probabilities q and $\frac{p}{1+p}$. The graph preservation statements are (after some simplification):

$$q > 0 \text{ and } \frac{p}{1+p} > 0 \text{ and } q + \frac{p}{1+p} = 1.$$

Transforming the second item as explained above yields:

$$1 + p \neq 0 \wedge \Big((1 + p > 0 \wedge p > 0) \vee (1 + p < 0 \wedge p < 0)\Big)$$

while transforming the third item yields:

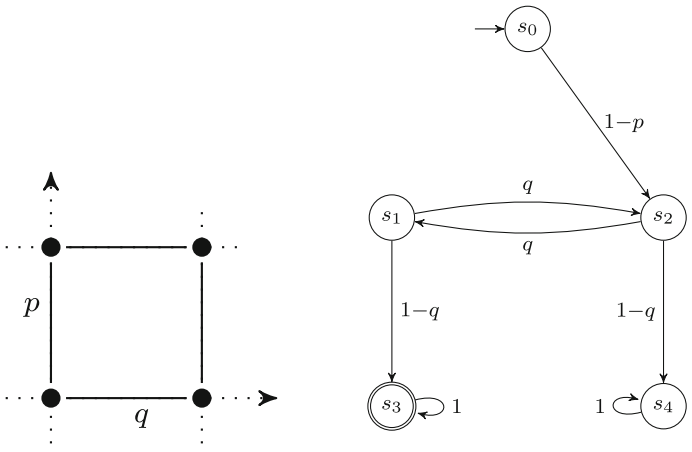
$$(1 + p \neq 0) \wedge q \cdot (1+p) - 1 = 0.$$

Finally, we obtain the following formula (after some further simplifications):

$$q > 0 \ \wedge \ (p > 0 \vee p < -1) \ \wedge \ q \cdot (1 + p) - 1 = 0.$$

4.3 Reduction to graph-preserving regions

In this section, we show how we can partition a well-defined region into a set of graph-preserving regions. This is useful, e.g., as we only define solution functions for graph-preserving regions. The decomposition in this section allows to define solution functions on each of these partitions, see also Remark 5. Before we illustrate the decomposition, we define sub-pSGs: Given two pSGs $\mathcal{G} = (S, V, s_I, Act, \mathcal{P})$ and $\mathcal{G}' = (S', V', s_I', Act', \mathcal{P}')$, $\mathcal{G}'$ is a *sub-pSG* of $\mathcal{G}$ if $S' \subseteq S, V' \subseteq V, s_I' = s_I \in S', Act' \subseteq Act$, and $\mathcal{P}'(s, \alpha, s') \in \{\mathcal{P}(s, \alpha, s'), 0\}$



(a) Subregions (b) sub-pMC for $p = 0$

Fig. 7 Ensuring graph-preservation on subregions

for all $s, s' \in S'$ and $\alpha \in Act'$. Note that for a given state $s \in S$ and action $\alpha \in Act(s)$, the sub-pSG might not contain s or α might not be enabled in s, but it is also possible that the sub-pSG omits some but not all successors of α in s.

Example 22 Reconsider the pMC $\mathcal{D}$ from Fig. 5c, and let $R = [0, 1] \times [0, 1]$, which is well-defined but not graph preserving. Region R can be partitioned into 9 regions, see Fig. 7a where each dot, line segment, and the inner region are subregions of R. All subregions are graph preserving on some sub-pMC of $\mathcal{D}$. Consider, e.g., the line-region $R' = \{u \in R \mid p[u] = 0\}$. The subregion R' is not graph preserving on pMC $\mathcal{D}$, as the transition $s_0 \xrightarrow{p} s_1$ vanishes when $p = 0$. However, R' is graph preserving on the sub-pMC $\mathcal{D}'$ in Fig. 7b, which is obtained from $\mathcal{D}$ by removing the transitions on the line-region p=0.

Let us formalise the construction from this example. For a given well-defined region R, and pSG $\mathcal{G}$, let $\mathcal{Z}_R$ describe the set of constraints:

$$\{\mathcal{P}(s, \alpha, s')=0 \mid s, s' \in S \wedge \alpha \in Act(s) \wedge \mathcal{P}(s, \alpha, s') \not\equiv 0 \wedge \exists u \in R.\, \mathcal{P}(s, \alpha, s')[u] = 0\}.$$

For $X \subseteq \mathcal{Z}_R$, the subregion $R_X \subseteq R$ is defined as:

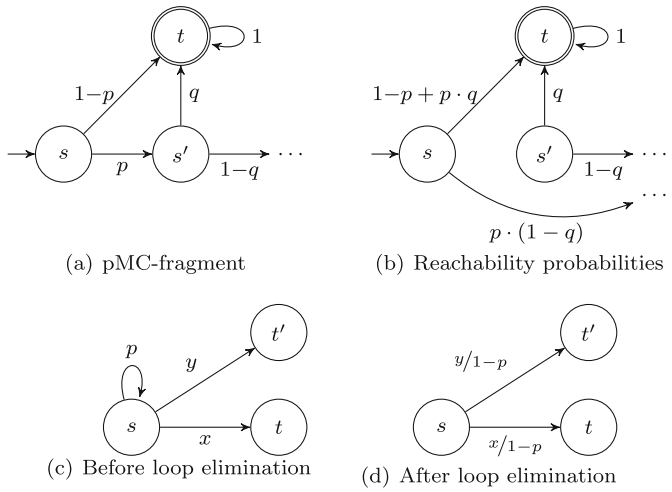
$$\Phi(R_X) \;= \Phi(R) \wedge \bigwedge_{c \in X} c \wedge \bigwedge_{c \in \mathcal{Z}_R \setminus X} \neg c.$$

It follows that X uniquely characterises which transition probabilities in $\mathcal{G}$ are set to zero. In fact, each instance in R_X is graph preserving for the unique sub-pSG $\mathcal{G}'$ of $\mathcal{G}$ obtained from $\mathcal{G}$ by removing all zero-transitions in R_X. The pSG $\mathcal{G}'$ is well-defined as R on $\mathcal{G}$ is well-defined. By construction, it holds that $\mathcal{G}[u] = \mathcal{G}'[u]$ for all instantiations $u \in R'$.

5 Exact synthesis by computation of the solution function

This section discusses *how to compute the solution function*. The solution function for pMCs describes the exact accepting and rejecting regions, as discussed in Sect. 3.3[8]. This section

[8] For pMDPs, one may compute a solution function for every strategy, but this has little practical relevance



(a) pMC-fragment (b) Reachability probabilities

(c) Before loop elimination (d) After loop elimination

Fig. 8 Essential ideas for state elimination

thus provides an algorithmic approach to the exact synthesis problem. In Sect. 6, we will also see that the solution function may be beneficial for the performance of SMT-based (region) verification.

The original approach to compute the solution function of pMCs is via *state elimination* [64, 78], and is analogous to the computation of regular expressions from nondeterministic finite automata (NFAs) [90]. It is suitable for a range of indefinite-horizon properties. The core idea behind state elimination and the related approaches presented here is based on two operations:

- *Adding short-cuts:* Consider the pMC-fragment in Fig. 8a. The reachability probabilities from any state to t are as in Fig. 8b, where we replaced the transition from s to s' by shortcuts from s to t and all other successors of s', bypassing s'. By successive application of shortcuts, any path from the initial state to the target state eventually has length 1.
- *Elimination of self-loops:* A prerequisite for introducing a short-cut is that the bypassed state is loop-free. Recall that the probability of staying forever in a non-absorbing state is zero, and justifies elimination of self-loops by rescaling all other outgoing transitions, as depicted in the transition from Fig. 8c, d.

The remainder of this section is organised as follows: Sect. ?? recaps the original state elimination approach in Sect. 5.1, albeit slightly rephrased. The algorithm is given for (indefinite) reachability probabilities, expected rewards, and bounded reachability probabilities. In the last part, we present alternative, equivalent formulations which sometimes allow for superior performance. In particular, Sect. 5.2 clarifies the relation to solving a linear equation system over a field of rational functions, and Sect. 5.3 discusses a variation of state elimination applicable to pMCs described by multi-terminal binary decision diagrams.

5.1 Algorithm based on state elimination

Let $T \subseteq S$ be a set of target states and assume w. l. o. g. that all states in T are absorbing and that $s_I \notin T$.

Algorithm 1 State elimination for pMCs

reachability(pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$, $T \subseteq S$)
  $S^? \coloneqq \{s \in S \mid s \neq s_I \wedge s \in \Diamond T \setminus T\}$
  while $S^? \neq \emptyset$ **do**
    select $s \in S^?$
    eliminate_selfloop$(\mathcal{P}, s)$
    eliminate_state$(\mathcal{P}, s)$
    $S^? \coloneqq S^? \setminus \{s\}$
  eliminate_selfloop$(\mathcal{P}, s_I)$
  // All $S^?$ eliminated. Only direct transitions to target.
  return $\sum_{t \in T} \mathcal{P}(s_I, t)$

eliminate_selfloop$(\mathcal{P}, s \in S)$
  assert $\mathcal{P}(s, s) \neq 1$
  for each $s_2 \in \text{succ}(s), s \neq s_2$ **do**
    $\mathcal{P}(s, s_2) \coloneqq \frac{\mathcal{P}(s,s_2)}{1-\mathcal{P}(s,s)}$
  $\mathcal{P}(s, s) \coloneqq 0$

eliminate_transition$(\mathcal{P}, s_1 \in S, s \in S)$
  assert $s_1 \in \text{pred}(s), \mathcal{P}(s, s) = 0$
  for each $s_2 \in \text{succ}(s)$ **do**
    $\mathcal{P}(s_1, s_2) \coloneqq \mathcal{P}(s_1, s_2) + \mathcal{P}(s_1, s) \cdot \mathcal{P}(s, s_2)$
  $\mathcal{P}(s_1, s) \coloneqq 0$

eliminate_state$(\mathcal{P}, s \in S)$
  assert $\mathcal{P}(s, s) = 0$
  for each $s_1 \in \text{pred}(s)$ **do**
    eliminate_transition$(\mathcal{P}, s_1, s)$

5.1.1 Reachability probabilities

We describe the algorithm to compute reachability probabilities based on state elimination in Algorithm 1. In the following, $\mathcal{P}$ is the transition matrix. The function **eliminate_selfloop**$(\mathcal{P}, s)$ rescales all outgoing probabilities of a non-absorbing state s by eliminating its self-loop. The function **eliminate_transition**$(\mathcal{P}, s_1, s_2)$ adds a shortcut from s_1 to the successors of s_2. Both operations preserve reachability to T. The function **eliminate_state**$(\mathcal{P}, s)$ "bypasses" a state s by adding shortcuts from all its predecessors. More precisely, we eliminate the incoming transitions of s, and after all incoming transitions are removed, the state s is unreachable. It is thereby effectively removed from the model.

After removing all non-absorbing, non-initial states $S^?$, the remaining model contains only self-loops at the absorbing states and transitions emerging from the initial state. Eliminating the self-loop on the initial state (by rescaling) yields a pMC. In this pMC, after a single step, an absorbing state is reached. These absorbing states are either a target or a sink. The solution function is then the sum over all (one-step) transition probabilities to target states.

Example 23 Consider again the pMC from Example 8, also depicted in Fig. 9a. Assume state s_2 is to be eliminated. Applying the function **eliminate_state**$(\mathcal{P}, s_2)$, we first eliminate the transition $s_1 \to s_2$, which yields Fig. 9b, and subsequently eliminate the transition $s_0 \to s_2$ (Fig. 9c). State s_2 is now unreachable, so we can eliminate s_2, reducing computational effort when eliminating state s_1. For state s_1, we first eliminate the self-loop (Fig. 9e) and then eliminate the transition $s_0 \to s_1$. The final result, after additionally removing the now

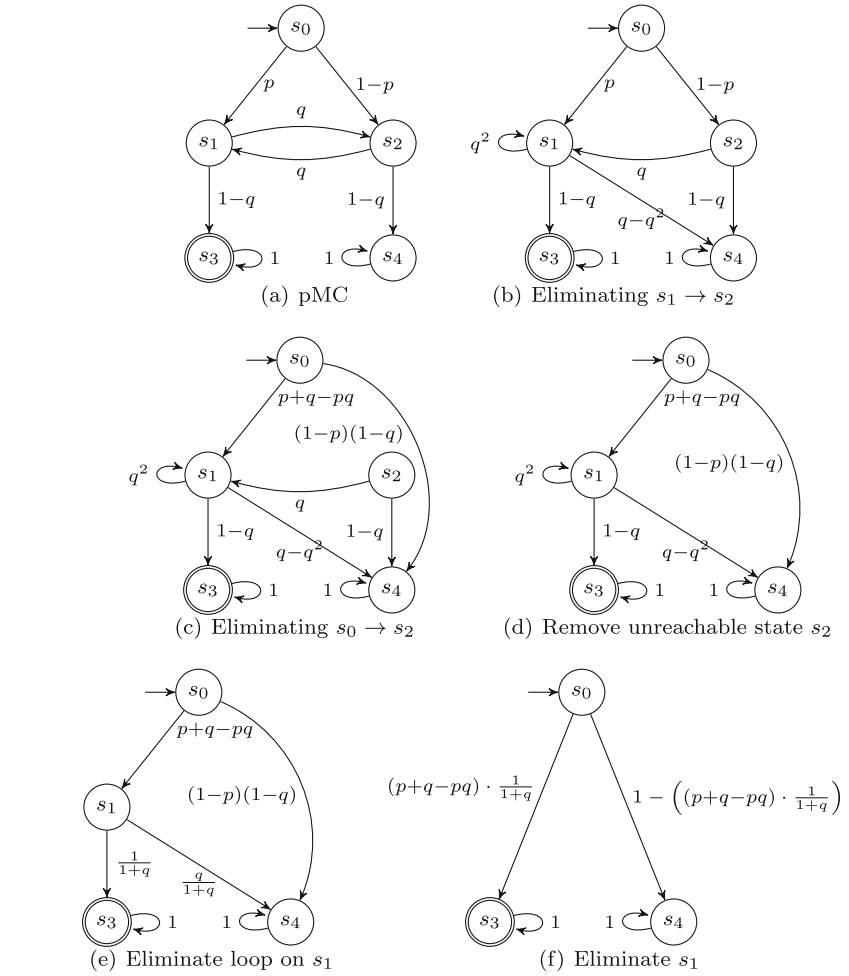


Fig. 9 State elimination exemplified

unreachable s_1, is depicted in Fig. 9f. The result, i.e., the probability to eventually reach s_3 from s_0 in the original model, can now be read from the single transition between these two states.

As for computing of regular expressions from NFAs, the order in which the states are eliminated is essential. Computing an optimal order with respect to minimality of the result, however, is already NP-hard for acyclic NFAs, see [84]. For state elimination on pMCs, the analysis is more intricate, as the cost of every operation crucially depends on the size and the structure of the rational functions. We briefly discuss the implemented heuristics in Sect. 10.2.1.

Remark 9 The elimination of self-loops yields a rational function. In order to keep these functions as small as possible, it is natural to eliminate common factors of the numerator and the denominator. Such a reduction, however, involves the computation of greatest common divisors (gcds). This operation is expensive for multivariate polynomials. In [91], data struc-

Algorithm 2 State elimination with one-step probabilities

reachability(pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$, $T \subseteq S$)
  $S^? \coloneqq \{s \in S \mid s \in \Diamond T \setminus T\}$
  // $x \colon S^? \to [0, 1]$
  $x(s) \coloneqq \sum_{t \in T} \mathcal{P}(s, t)$ for each $s \in S^?$
  $\mathcal{P}(s, t) \coloneqq 0$ for all $s \in S, t \in T$
  while $S^? \neq \emptyset$ **do**
    eliminate_state($\mathcal{P}, x, s$) for some $s \in S^?$
    $S^? \coloneqq S^? \setminus \{s\}$
  // All $S^?$ eliminated. One-step probability is reachability probability.
  return $x(s_I)$

eliminate_transition($\mathcal{P}, x, s_1 \in S, s \in S$)
  // Algorithm modifies $\mathcal{P}$ **assert** $s_1 \neq s, \mathcal{P}(s, s) \neq 1$
  $x(s_1) \coloneqq x(s_1) + \frac{\mathcal{P}(s_1, s) \cdot x(s)}{1 - \mathcal{P}(s, s)}$
  for each $s_2 \in \text{succ}(s), s \neq s_2$ **do**
    $\mathcal{P}(s_1, s_2) \coloneqq \mathcal{P}(s_1, s_2) + \frac{P(s_1, s) \cdot \mathcal{P}(s, s_2)}{1 - \mathcal{P}(s, s)}$
  $\mathcal{P}(s_1, s) \coloneqq 0$

eliminate_state($\mathcal{P}, x, s \in S$)
  // Algorithm modifies $\mathcal{P}$ **assert** $\mathcal{P}(s, s) = 0$
  for each $s_1 \in \text{pred}(s)$ **do**
    eliminate_transition($\mathcal{P}, x, s_1, s$)

tures to avoid their computation are introduced, in [17] a method is presented that mostly avoids introducing common factors.

5.1.2 Expected rewards

The state elimination approach can also be adapted to compute *expected rewards* [78]. When eliminating a state s, in addition to adjusting the probabilities of the transitions from all predecessors s_1 of s to all successors s_2 of s, it is also necessary to "summarise" the reward that would have been gained from s_1 to s_2 via s. The presentation in [78] describes these operations on so-called transition rewards. Observe that for the analysis of expected rewards in MCs, we can always reformulate transition rewards in terms of state rewards. We preprocess pMCs to only have rewards at the states: this adjustment simplifies the necessary operations considerably.

The treatment of the expected reward computation is easiest from an adapted (and more performant) implementation of state elimination, as outlined in Algorithm 2. Here, we eliminate the probabilities to reach a target state in exactly one step, and collect these probabilities in a vector x which we refer to as *one-step-probabilities*. Then, we proceed similar as before. However, the elimination of a transition from s_1 to s now has two effects: it updates the probabilities within the non-target states as before, and (potentially) updates the probability $x(s_1)$ to reach the target within one step from s_1 (with the probability that the target was reached via s in two steps). Upon termination of the outer loop, the vector x contains the probabilities from all states to reach the target, that is, $x(s_i) = x_{s_i}$.

Finally, when considering rewards, the *one-step-probabilities* contain initially the rewards for the states. Eliminating a transition then moves the (expected) reward to the predecessors by the *same* sequence of arithmetic operations.

5.1.3 Bounded reachability

As discussed in Remark 4, bounded reachability can typically be considered by an unfolding of the Markov model and considering an unbounded reachability property on that (acyclic) unfolding. In combination with state elimination, that yields the creation of many states that are eliminated afterwards, and does not take into account any problem-specific properties. Rather, and analogous to the parameter-free case [13], it is better to do the adequate matrix-vector multiplication (# number of steps often). The matrix originates from the transition matrix, the vector (after i multiplications) encodes the probability to reach a state within i steps.

5.2 Algorithm based on solving the linear equation system

The following set of equations is a straightforward adaption of the Bellman linear equation system for MCs found in, e.g., [13, 121] to pMCs. For each state s, a variable x_s is used to express the probability $\Pr_s(\Diamond T)$ to reach a state in T from the state s. Recall that we overloaded $\Diamond T$ to also denote the set of states from which T is reachable (with positive probability). Analogously, we use $\neg\Diamond T$ to denote the set of states from which T is not reachable, i. e., $\neg\Diamond T = S \setminus \Diamond T$. We have:

$$x_s = 0 \qquad \forall s \in \neg\Diamond T \tag{5}$$

$$x_s = 1 \qquad \forall s \in T \tag{6}$$

$$x_s = \sum_{s' \in S} \mathcal{P}(s, s') \cdot x_{s'} \qquad \forall s \in \Diamond T \setminus T. \tag{7}$$

This system of equations has a unique solution for every well-defined parameter instantiation. In particular, the set of states satisfying $\neg\Diamond T$ is the same for all well-defined graph-preserving parameter instantiations, as instantiations that maintain the graph of the pMC do not affect the reachability of states in T.

For pMCs, the coefficients are no longer from the field of the real numbers, but rather from the field of rational functions.

Example 24 Consider the equations for the pMC from Fig. 9a.

$$\begin{aligned} x_0 &= p \cdot x_1 + (1-p) \cdot x_2 \\ x_1 &= q \cdot x_2 + (1-q) \cdot x_3 \\ x_2 &= q \cdot x_1 + (1-q) \cdot x_4 \\ x_3 &= 1 \\ x_4 &= 0. \end{aligned}$$

Bringing the system in normal form yields:

$$\begin{aligned} x_0 - p \cdot x_1 - (1-p) \cdot x_2 &= 0 \\ x_1 - q \cdot x_2 - (1-q) \cdot x_3 &= 0 \\ -q \cdot x_1 + x_2 - (1-q) \cdot x_4 &= 0 \\ x_3 &= 1 \\ x_4 &= 0. \end{aligned}$$

Adding q times the second equation to the third equation (concerning state s_2) brings the left-hand side matrix in upper triangular form:

$$\begin{aligned} x_0 - p \cdot x_1 - (1-p) \cdot x_2 &= 0 \\ x_1 - q \cdot x_2 - (1-q) \cdot x_3 &= 0 \\ (1-q^2) \cdot x_2 - q(1-q) \cdot x_3 - (1-q) \cdot x_4 &= 0 \\ x_3 &= 1 \\ x_4 &= 0. \end{aligned}$$

The equation system yields the same result as the elimination of the transition from s_2 to s_1 (notice the symmetry between s_1 and s_2).

The example illustrates that there is no elementary advantage in doing state elimination over resorting to solving the linear equation sytem by (some variant of) Gaussian elimination. If we are only interested in the probability from the initial state, we do not need to solve the full equation system. The state-elimination algorithm, in which we can remove unreachable states, optimises for this observation, in contrast to (standard) linear equation solving. As in state elimination, the elimination order of the rows has a significant influence.

5.3 Algorithm based on set-based transition elimination

To succinctly represent large state spaces, Markov chains are often represented by multi-terminal binary decision diagrams (or variants thereof) [14]. Such a *symbolic representation* handles sets of states instead of single states (and thus also sets of transitions), and thereby exploits symmetries and similarities in the underlying graph of a model. To support efficient elimination, we describe how to eliminate sets of transitions at once. The method is similar to the Floyd-Warshall algorithm for all-pair shortest paths [51]. The transition matrix contains one-step probabilities for every pair of source and target states. Starting with a self-loop-free pMC (obtained by eliminating all self-loops from the original pMC), we iterate two operations until convergence. By doing a matrix-matrix multiplication, we effectively eliminate all transitions emanating from all non-absorbing states *simultaneously*. As this step may reintroduce self-loops, we eliminate them in a second step. As before, eventually only direct transitions to absorbing states remain, which effectively yield the unbounded reachability probabilities. The corresponding pseudo-code is given in Algorithm 3.

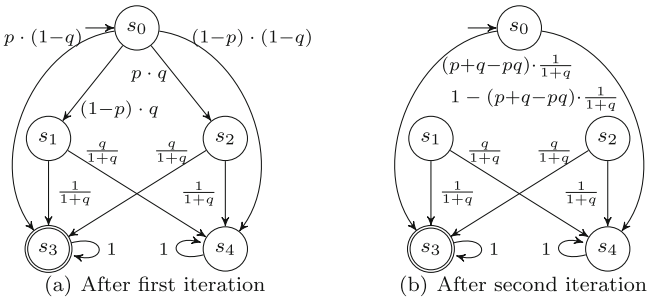
The approach of this algorithm can conveniently be explained in the equation system representation. Let us therefore conduct one step of the algorithm as an example, where we use the observation that the matrix-matrix multiplication corresponds to replacing the variables x_s by their defining equations in all *other* equations.

Example 25 Reconsider the equations from Example 24:

$$\begin{aligned} x_0 &= p \cdot x_1 + (1-p) \cdot x_2 \\ x_1 &= q \cdot x_2 + (1-q) \cdot x_3 \\ x_2 &= q \cdot x_1 + (1-q) \cdot x_4 \\ x_3 &= 1 \\ x_4 &= 0. \end{aligned}$$

Using the equations for x_0, x_1, x_2 to replace their occurrences in all *other* equations yields:

$$x_0 = p \cdot (q \cdot x_2 + (1-q) \cdot x_3) + (1-p)(q \cdot x_1 + (1-q) \cdot x_4)$$



(a) After first iteration (b) After second iteration

Fig. 10 One step of set-based transition elimination exemplified

Algorithm 3 Set-based transition elimination for pMCs

reachability(pMC $\mathcal{D} = (S, V, s_I, \mathcal{P}), T \subseteq S$)
  $S^? \coloneqq \{s \in S \mid s \neq s_I \wedge s \in \Diamond T \setminus T\}$
  for each $s \in S^?$ **do**
    // can be done in parallel for all s
    eliminate_selfloop($\mathcal{P}, s$)
  while $\exists s, s' \in S^?. \mathcal{P}(s, s') \neq 0$ **do**
    for each $s \in S^?, s' \in S$ **do**
      // can be done in parallel for all s, s'
      $\mathcal{P}'(s, s') \coloneqq \sum_{s''} \mathcal{P}(s, s'') \cdot \mathcal{P}(s'', s')$
    for each $s \in S^?$ **do**
      // can be done in parallel for all s
      eliminate_selfloop($\mathcal{P}', s$)
    $\mathcal{P} \coloneqq \mathcal{P}'$
  // All $S^?$ eliminated. Only direct paths to target.
  return $\sum_{t \in T} \mathcal{P}(s_I, t)$

$$
\begin{aligned}
x_1 &= q \cdot (q \cdot x_1 + (1-q) \cdot x_4) + (1-q) \cdot x_3 \\
x_2 &= q \cdot (q \cdot x_2 + (1-q) \cdot x_3) + (1-q) \cdot x_4 \\
x_3 &= 1 \\
x_4 &= 0
\end{aligned}
$$

which simplifies to

$$
\begin{aligned}
x_0 &= (1-p) \cdot q \cdot x_1 + p \cdot q \cdot x_2 + p \cdot (1-q) \cdot x_3 \\
&\quad + (1-p)(1-q) \cdot x_4 \\
x_1 &= \frac{1}{1+q} \cdot x_3 + \frac{q}{1+q} \cdot x_4 \\
x_2 &= \frac{q}{1+q} \cdot x_3 + \frac{1}{1+q} \cdot x_4 \\
x_3 &= 1 \\
x_4 &= 0.
\end{aligned}
$$

We depict the pMC which corresponds to this equation system in Fig. 10a. Again, notice the similarity to state elimination. For completeness, the result after another iteration is given in Fig. 10b.

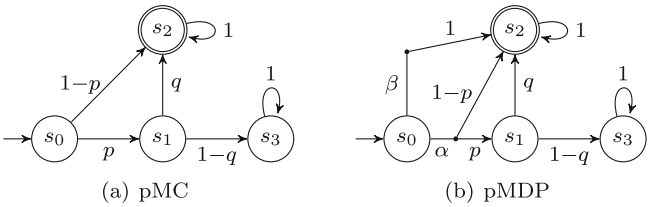


Fig. 11 Toy-examples (repeated from Fig. 6)

The correctness follows from the following argument: After every iteration, the equations describe a pMC over the same state space as before. As all absorbing states have defining equations $x_i \in \{0, 1\}$, the equation system is known to have a unique solution [13]. Moreover, as the equation system in iteration i implies the equation system in iteration $i+1$, they preserve the same (unique) solution.

6 SMT-based region verification

In this section, we discuss a complete procedure to verify regions by encoding them as queries for an SMT solver, or more precisely, in the existential theory of the reals (the `QF_NRA` theory in the SMT literature). We first introduce the constraints for verifying regions on pMCs in Sect. 6.1. The constraints are either based on the equation system encoding from Sect. 5.2 or use the solution function, which yields an equation system with less variables at the cost of precomputing the solution function. In Sect. 6.2, we then introduce the encodings for region verification on pMDPs under angelic and demonic strategies.

Throughout the section, we focus on unbounded reachability, that is, we assume $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$. As expected rewards can be described by a similar equation system, lifting the concepts is straightforward. We assume a graph-preserving region R: Assuming that R is graph preserving eases the encodings significantly, but is not strictly necessary: In [94, Ch. 4], we provide encodings for well-defined regions R.

6.1 Satisfiability checking for pMC region checking

Recall from Sect. 5.2 the equation system for pMCs, exemplified by the following running example.

Example 26 Reconsider the pMC $\mathcal{D}$ from Fig. 6a, repeated in Fig. 11a for convenience. The concrete equation system of (5)–(7) on page 32 for reaching $T = \{s_2\}$, using x_i to denote x_{s_i}, is given by:

$$\begin{aligned} x_0 &= p \cdot x_1 + (1-p) \cdot x_2 \\ x_1 &= q \cdot x_2 + (1-q) \cdot x_3 \\ x_2 &= 1 \\ x_3 &= 0. \end{aligned}$$

The conjunction of the equation system for the pMC, (5)–(7) on page 32, is an implicitly existential quantified formula to which we refer by $\Phi(\mathcal{D})$—consider the remark below. By construction, this formula is satisfiable.

Remark 10 If transitions in the pMC are not polynomial but rational functions, the equations are not polynomial constraints, hence their conjunction is not a formula (Sect. 2.5). Instead, each $x = \sum \mathcal{P}(s, s')$ has to be transformed by the rules in Sect. 4.2.2: then, their conjunction is a formula. This transformation can always be applied, in particular, in the equalities we are never interested in the evaluation of instantiations $u \in R$ with $\mathcal{P}(s, s')[u] = \bot$: Recall that we are interested in analysing this equation system on a *well-defined* parameter region R: Therefore, for any $u \in R$, $\mathcal{P}(s, s')[u] \neq \bot$ for each $s, s' \in S$. Thus, when $\Phi(\mathcal{D})$ is used in conjunction with $\Phi(R)$, we do not need to consider this special case.

We consider the conjunction of the equation system, a property and a region. Concretely, let us first consider the conjunction of:

- the equation system $\Phi(\mathcal{D})$,
- a comparison of the initial state s_I with the threshold λ, and
- a formula $\Phi(R)$ describing the parameter region R.

Satisfiability of this conjunction means that—for some parameter instantiation within the region R—the reachability probability from the initial state s_I satisfies the bound. Unlike $\Phi(\mathcal{D})$, this conjunction may be unsatisfiable.

Example 27 We continue with Example 26. Let $\varphi = \mathbb{P}_{\leq 0.4}(\Diamond\{s_2\})$ and $R = \{(p, q) \in [0.4, 0.6] \times [0.2, 0.5]\}$. We have $\Phi(R) = 0.4 \leq p \wedge p \leq 0.6 \wedge 0.2 \leq q \wedge q \leq 0.5$. We obtain the following conjunction:

$$\Phi(\mathcal{D}) \wedge x_0 \leq 0.4 \wedge \Phi(R) \tag{8}$$

where $\Phi(\mathcal{D})$ is the conjunction of the equation system, i.e.:

$$\Phi(\mathcal{D}) = \begin{pmatrix} x_0 = p \cdot x_1 + (1-p) \cdot x_2 & \wedge \\ x_1 = q \cdot x_2 + (1-q) \cdot x_3 & \wedge \\ x_2 = 1 \ \wedge \ x_3 = 0 & \end{pmatrix}.$$

Formula (8) is unsatisfiable, thus, no instance of p and q within the region R induces a reachability probability of at most $2/5$.

Towards region verification, consider that the satisfaction relations $\models_a$[9] as defined in Definition 13, we have to certify that all parameter values within a region yield a reachability probability that satisfies the threshold. Thus, we have to quantify over all instantiations u, (roughly) leading to a formula of the form $\forall u \ldots \models \varphi$. By negating this statement, we obtain the proof obligation $\neg\exists u \ldots \models \neg\varphi$: no parameter value within the region R satisfies the negated comparison with the initial state. This intuition leads to the following conjunction of:

- the equation system $\Phi(\mathcal{D})$,
- a comparison of the initial state with the threshold, by inverting the given threshold-relation, and
- a formula $\Phi(R)$ describing the parameter region.

This conjunction is formalised in the following definition.

[9] Recall that $\models_d$ coincides with $\models_a$ for pMCs.

Definition 19 (*Equation system formula*) Let $\mathcal{D}$ be a pMC, $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond T)$, and R a region. The *equation system formula* is given by:

$$\Phi(\mathcal{D}) \wedge x_{s_I} \nsim \lambda \wedge \Phi(R).$$

Theorem 1 *The equation system formula is unsatisfiable iff* $\mathcal{D}, R \models \varphi$.

Otherwise, a satisfying solution is a *counterexample*.

Example 28 We continue Example 27. We invert the relation $x_0 \leq 0.4$ and obtain:

$$\Phi(\mathcal{D}) \wedge x_0 > 0.4 \wedge \Phi(R).$$

By SMT-checking, we determine that the formula is satisfiable, e.g., with $p = 0.5$ and $q = 0.3$. Thus, $\mathcal{D}, R \not\models \varphi$. If we consider instead the region $R' = \{(p, q) \in [0.8, 0.9] \times [0.1, 0.2]\}$ with $\Phi(R') = 0.8 \leq p \wedge p \leq 0.9 \wedge 0.1 \leq q \wedge q \leq 0.2$, we obtain:

$$\Phi(\mathcal{D}) \wedge x_0 > 0.4 \wedge \Phi(R')$$

which is unsatisfiable. Hence, no point in R' induces a probability larger than $2/5$ and, equivalently, all points in R' induce a probability of at most $2/5$. Thus, $\mathcal{D}, R' \models \varphi$.

We observe that the number of variables in this encoding is $|S| + |V|$. In particular, we are often interested in systems with at least thousands of states. The number of variables is therefore often too large for SMT-solvers dealing with non-linear real arithmetic. However, many of the variables are auxiliary variables that encode the probability to reach target states from each individual state. We can get rid of these variables by replacing the full equation system by the solution function (Definition 10).

Definition 20 (*Solution function formula*) Let $\mathcal{D}$ be a pMC, $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond T)$, and R a region. The *solution function formula*[10] is given by:

$$f^r_{\mathcal{D},T} \nsim \lambda \wedge \Phi(R).$$

Corollary 1 *The solution function formula is unsatisfiable iff* $\mathcal{D}, R \models \varphi$.

Example 29 We consider the same scenario as in Example 27. The solution function is given in Example 13. The solution function formula is:

$$1 - p + p \cdot q > 0.4 \wedge \Phi(R).$$

By construction, the equation system formula and the solution function formula for pMC $\mathcal{D}$ and reachability property φ are equisatisfiable.

6.2 Existentially quantified formula for parametric MDPs

We can also utilise an SMT solver to tackle the verification problem on pMDPs. For parametric MDPs, we distinguish between the angelic and the demonic case, cf. Definition 14. We use the fact that optimal strategies for unbounded reachability objectives are memoryless and deterministic [121].

[10] Remark 10 applies also here.

6.2.1 Demonic strategies

The satisfaction relation $\models_d$ is defined by two universal quantifiers, $\forall u \forall \sigma \ldots \models \varphi$. We therefore try to refute satisfiability of $\exists u \exists \sigma \ldots \models \neg\varphi$. Put in a game-theoretical sense, the same player can choose both the parameter instantiation u and the strategy σ to resolve the non-determinism. We generalise the set of linear equations from the pMC to an encoding for pMDPs, where we define a disjunction over all possible nondeterministic choices:

$$x_s = 0 \qquad \forall s \in \neg\Diamond T \tag{9}$$

$$x_s = 1 \qquad \forall s \in T \tag{10}$$

$$\bigvee_{\alpha \in Act(s)} \left(x_s = \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot x_{s'} \right) \qquad \forall s \in \Diamond T \setminus T. \tag{11}$$

We denote the conjunction of (9)–(11) as $\Phi_d(\mathcal{M})$ for pMDP $\mathcal{M}$[11]. Instead of a single equation for the probability to reach the target from state s, we get one equation for each action. The solver can now freely choose which (memoryless deterministic) strategy it uses to refute the property.

Definition 21 (*Demonic equation system formula*) Let $\mathcal{M}$ be a pMDP, $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$, and R a region. The *demonic equation system formula* is given by:

$$\Phi_d(\mathcal{M}) \wedge x_{s_I} > \lambda \wedge \Phi(R).$$

Theorem 2 *The demonic equation system formula is unsatisfiable iff* $\mathcal{M}, R \models_d \varphi$.

Example 30 Let $\mathcal{M}$ be the pMDP from Fig. 11b. Let R, φ be as in Example 27. The demonic equation system formula is

$$\Phi_d(\mathcal{M}) \wedge x_0 > 0.4 \wedge \Phi(R)$$

with $\Phi(R)$ as before, and

$$\begin{aligned} \Phi_d(\mathcal{M}) = \quad & \Big(\big(x_0 = p \cdot x_1 + (1-p) \cdot x_2 \quad \vee \quad x_0 = x_2 \big) && \wedge \\ & x_1 = q \cdot x_2 + (1-q) \cdot x_3 && \wedge \\ & x_2 = 1 \ \wedge \ x_3 = 0 && \Big). \end{aligned}$$

Similarly, when using the (potentially exponential) set of solution functions, we let the solver choose:

Definition 22 (*Demonic solution function formula*) Let $\mathcal{M}$ be a pMDP, $\varphi = \mathbb{P}_{\sim\lambda}(\Diamond T)$, and R a region. The *demonic solution function formula* is given by:

$$\bigvee_{\sigma \in Str^{\mathcal{M}}} f^r_{\mathcal{M}^\sigma, T} \sim \lambda \wedge \Phi(R).$$

Corollary 2 *The demonic solution function formula is unsatisfiable iff* $\mathcal{M}, R \models_d \varphi$.

As the set of solution functions can be exponential, the demonic solution function formula can grow exponentially.

[11] Recall again Remark 10.

Example 31 The demonic solution function formula for $\mathcal{M}, \varphi, R$ as in Example 30, is given by:

$$\Big(1 > 0.4 \vee 1 - p + p \cdot q > 0.4\Big) \wedge \Phi(R).$$

6.2.2 Angelic strategies

The satisfaction relation $\models_a$ has two different quantifiers, $\forall u \exists \sigma \ldots \models \varphi$. Again, we equivalently try to refute the satisfiability of $\exists u \forall \sigma \ldots \models \neg\varphi$. The quantifier alternation can be circumvented by lifting the linear programming (LP) formulation for MDPs [121], where for each nondeterministic choice an upper bound on the probability variables x_s is obtained:

$$x_s = 0 \qquad \forall s \in \neg\Diamond T \tag{12}$$

$$x_s = 1 \qquad \forall s \in T \tag{13}$$

$$\bigwedge_{\alpha \in Act(s)} \Big(x_s \leq \sum_{s' \in S} \mathcal{P}(s, \alpha, s') \cdot x_{s'}\Big) \qquad \forall s \in \Diamond T \setminus T. \tag{14}$$

Intuitively, the conjunction in constraint (14) eliminates the freedom of choosing any strategy from the solver and forces it to use the strategy that minimises the reachability probability. This means that the constraint system is only satisfiable if *all* strategies violate the probability bound. We denote the conjunction of (12)–(14) as $\Phi_a(\mathcal{M})$. Notice that, as for parameter-free MDPs, the optimisation objective of the LP formulation can be substituted by a constraint on probability in the initial state.

Definition 23 (*Angelic equation system formula*) Let $\mathcal{M}$ be a pMDP, $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$, and R a region. The *angelic equation system formula* is given by:

$$\Phi_a(\mathcal{M}) \wedge x_{s_I} > \lambda \wedge \Phi(R).$$

Theorem 3 *The angelic equation system formula is unsatisfiable iff* $\mathcal{M}, R \models_a \varphi$.

Example 32 Let $\mathcal{M}, \varphi, R$ as in Example 30. The angelic equation system formula is given by

$$\Phi_a(\mathcal{M}) \wedge x_0 > 0.4 \wedge \Phi(R)$$

with

$$\Phi_a(\mathcal{M}) = \Big(\big(x_0 \leq p \cdot x_1 + (1-p) \cdot x_2 \wedge x_0 \leq x_2\big) \wedge x_1 \leq q \cdot x_2 + (1-q) \cdot x_3 \wedge x_2 = 1 \wedge x_3 = 0\Big).$$

When using the set of solution functions, all strategies have to be considered. Again, for most pMDPs, this set is prohibitively large.

Definition 24 (*Angelic solution function formula*) Let $\mathcal{M}$ be a pMDP, $\varphi = \mathbb{P}_{\leq\lambda}(\Diamond T)$, and R a region. The *angelic solution function formula* is given by:

$$\bigwedge_{\sigma \in Str^{\mathcal{M}}} f^r_{\mathcal{M}^\sigma, T} > \lambda \wedge \Phi(R).$$

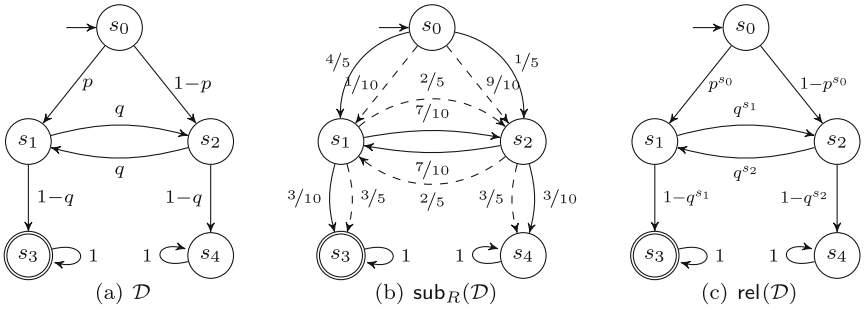


Fig. 12 A pMC $\mathcal{D}$ and its substitution $\mathsf{sub}_R(\mathcal{D})$ and its relaxation $\mathsf{rel}(\mathcal{D})$

Corollary 3 *The angelic solution function formula is unsatisfiable iff* $\mathcal{M}, R \models_a \varphi$.

Example 33 The angelic solution function formula for $\mathcal{M}, \varphi, R$ as in Example 30 is given by:

$$\Big(1 > 0.4 \wedge 1 - p + p \cdot q > 0.4\Big) \wedge \Phi(R).$$

7 Model-checking-based region verification of parametric MCs

This section discusses an abstraction (and refinement) procedure for region verification of pMCs. Intuitively, in order to bound the probability in a region from above, we bound the value induced by any instantation from above. We aim to do this by finding an instantiation that maximises the reachability probability in the region. This problem is particularly hard, as there are dependencies between the different parameters:

Example 34 Consider the pMC $\mathcal{D}$ in Fig. 12a—repeating Fig. 5c— and region $R = [1/10, 4/5] \times [2/5, 7/10]$. We again aim to reach s_3. We make two observations: s_4 is the only state from which we cannot reach s_3, furthermore, s_4 is only reachable via s_2. Hence, it is best to avoid s_2. From state s_0, it is thus beneficial if the transition probability to s_2 is as small as possible. Equivalently, it is beneficial if p is as large as possible, as this minimises the probability of reaching s_2 and as p does not occur elsewhere. Now we consider state s_1: As we want to reach s_3, the value of q should be preferably low. However, q occurs also at transitions leaving s_2. From s_2, q should be assigned a high value as we want to avoid s_4. In particular, the optimal value for q depends on the probability that we ever visit s_2, which is directly influenced by the value of p.

In a nutshell, the abstraction we propose in this section ignores the dependencies between the same occurence of a parameter. Conveniently, the abstraction transforms a pMC into an (parameter-free!) MDP whose minimal (maximal) reachability probability under-approximates (over-approximates) the reachability probability of the pMC. This result is formalised in Theorem 5, below.

Example 35 Consider the pMC in Fig. 12a and a region $R = [1/10, 4/5] \times [2/5, 7/10]$. The method creates the MDP in Fig. 12b, where different types of arrows reflect different actions. The MDP is created by adding in each state two actions: One reflecting the lower bound of the parameter range, one reflecting the upper bound. Model checking on this MDP yields a maximal probability of $47/60$. From this result, we infer that $\max_{u \in R} \Pr^{\mathcal{D}[u]}(\Diamond T) \leq 47/60$.

The essence of this construction is to consider parameter values as a local, discrete choice that we can capture with nondeterminism. To support the discretisation, we must ensure that the optimal values are taken at the bounds of the region. While this is not true in general due to the nonlinearity of the solution function, creating a suitable over-approximation, called the relaxation, enforces this property, as we show in Theorem 4, also below.

In the remainder of this section, we first clarify helpful assumptions on the type of pMCs we support in Sect. 7.1. We then construct so-called relaxed pMCs in Sect. 7.2. In Sect. 7.3, we translate relaxed pMCs to parameter-free MDPs to allow off-the-shelf MDP analysis for region verification of pMCs.

7.1 Preliminaries

We formalise the perspective that underpins our approach to region verification and introduce some assumptions.

7.1.1 A perspective for region verification

The probability $\Pr^{\mathcal{D}}(\lozenge T)$ can be expressed as a rational function $f = g_1/g_2$ with polynomials g_1, g_2 due to Definition 10. Recall that we assume region R to be graph preserving. Therefore, $g_2[u] \neq 0$ for all $u \in R$ and f is continuous on any closed region R. Hence, there is an instantiation $u \in R$ that induces the maximal (or minimal) reachability probability:

$$\sup_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T) = \max_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T) \quad \text{and} \quad \inf_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T) = \min_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T).$$

To infer that R is accepting (i.e. all instantiations $u \in R$ induce probabilities at most λ), it suffices to show that the *maximal* reachability probability over all instantiations is at most λ:

$$\mathcal{D}, R \models \mathbb{P}_{\leq \lambda}(\lozenge T) \iff \left(\max_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T)\right) \leq \lambda, \text{ and}$$

$$\mathcal{D}, R \models \neg\mathbb{P}_{\leq \lambda}(\lozenge T) \iff \left(\min_{u \in R} \Pr^{\mathcal{D}[u]}(\lozenge T)\right) > \lambda.$$

One way to determine the maximum reachability probability is to first determine which $u \in R$ induces the maximum, and then compute the probability on the instantiated model $\mathcal{D}[u]$. While we only discuss upper-bounded specifications here, the results can be analogously described for lower-bounded specifications.

Example 36 Consider $\mathcal{D}$ depicted in Fig. 11a, $\varphi = \mathbb{P}_{\leq 9/10}(\lozenge\{s_2\})$, and $R' = \{(p, q) \in [2/5, 3/5] \times [1/5, 1/2]\}$ as in Example 27. The maximum is obtained at $u = (2/5, 1/2)$ (via some oracle). We have $\mathcal{D}[u] \models \mathbb{P}_{\leq 9/10}(\lozenge\{s_2\})$, and thus, $\mathcal{D}, R' \models \mathbb{P}_{\leq 9/10}(\lozenge\{s_2\})$.

However, constructing an oracle that determines the u that induces the maximum is difficult in general. We focus on the essential idea an therefore make the following assumptions throughout the rest of this section:

Assumption 1
- We restrict the (graph-preserving) region R to be (i) rectangular, and (ii) closed. This restriction makes the bounds of the parameters independent of other parameter instantiations, and ensures that the maximum over the region exists.
- We restrict the pMC $\mathcal{D}$ to be *locally monotone*—explained in Sect. 7.1.2—to exclude difficulties from analysing single transitions.

The first assumption can be a nuisance. In particular, it is not always clear how to create an adequate closed region from an open region. The second assumption is very mild and can be accomodated for using adequate preprocessing [94, Section 5.1] that introduced additional states.

7.1.2 Locally monotone pMCs

Recall that the solution function is nonlinear. We aim to approximate this u and therefore want to exploit the structure of the pMC. Therefore, we want to make an assumption on the transition relation.

Example 37 Consider a three-state pMC where the probability from initial state s_I to target state t is a non-linear, non-monotone transition function, as, e.g., the transition probability from s_0 to s_3 of the pMC in Fig. 9f. Finding the maximum requires an analysis of the derivative of the solution function, and is (approximately) as hard as the exact verification problem.

Instead, we assume monotonic transition probabilities, and consider a slightly restricted class of pMCs.

Definition 25 (*Locally monotone pMCs*) A pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$ is *locally monotone* iff for all $s \in S$ there is a multilinear polynomial $g_s \in \mathbb{Q}[V]$ satisfying

$$\mathcal{P}(s, s') \in \{f/g_s \mid f \in \mathbb{Q}[V] \text{ is multilinear}\}$$

for all $s' \in S$.

Locally monotone pMCs include most pMCs from the literature [122][12]. Examples of the egligible transition probabilities are p, pq, $1/p$ and their complements formed by $1 - p$ etc.

Thanks to monotonicity, for a locally monotone pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$, and a closed rectangular region R we have that for all $s, s' \in S$:

$$\max_{u \in R} \mathcal{P}(s, s') = \max_{u \in B(V)} \mathcal{P}(s, s')$$

where $B(V) = \{u \mid \forall p \in V.u(p) \in B_R(p)\}$, i.e., all maxima of the individual transition probabilities are attained at the bounds of the region. However, the restriction to local monotonicity does not immediately overcome the challenge of constructing an oracle. The resulting solution function may still be highly nonlinear. In particular, Example 34 uses a locally monotone pMC and a closed rectangular region. However, as the example indicates, trade-offs in locally monotone pMCs occur due to dependencies where parameters occur at multiple states.

7.2 Relaxation

The idea of our approach, inspired by [30], is to drop the aforementioned dependencies between parameters by means of a *relaxation* of the pMC. We want to highlight that this relaxed pMC is very similar to so-called interval MCs, a detailed discussion is given in [94, Section 5.1.1.3]. Intuitively, the relaxation $\mathsf{rel}(\mathcal{D})$ is a pMC that arises from $\mathcal{D}$ to a pMC with the same state space but an updated transition relation. In particular, it introduces a

[12] It even includes the embedded pMCs of parametric continuous-time Markov chains with multilinear exit rates.

fresh copy of every parameter in every state, thereby eliminating parameter dependencies between different states (if any). This step simplifies finding an optimal instantiation (in the relaxation). However, the set of instantiated pMCs grows: some of the instantiations cannot be obtained from the original pMC. In this subsection, we first formalize the relaxation, then clarify the relation between properties being satisfied on the pMC and properties satisfied on the relaxation. We finish the subsection by discussing how to efficiently analyze a relaxed pMC.

Definition 26 (*Relaxation*) The *relaxation* of pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$ is the pMC $\mathsf{rel}(\mathcal{D}) = (S, \mathsf{rel}_{\mathcal{D}}(V), s_I, \mathcal{P}')$ with $\mathsf{rel}_{\mathcal{D}}(V) = \{p_i^s \mid p_i \in V, s \in S\}$ and $\mathcal{P}'(s, s') = \mathcal{P}(s, s')[p_1, \ldots, p_n / p_1^s, \ldots, p_n^s]$.

We extend an instantiation u for $\mathcal{D}$ to the *relaxed instantiation* $\mathsf{rel}_{\mathcal{D}}(u)$ for $\mathsf{rel}(\mathcal{D})$ by $\mathsf{rel}_{\mathcal{D}}(u)(p_i^s) = u(p_i)$ for every s. We have that for all u, $\mathcal{D}[u] = \mathsf{rel}(\mathcal{D})[\mathsf{rel}_{\mathcal{D}}(u)]$. We lift the relaxation to regions such that $B(p_i^s) = B(p_i)$ for all s, i. e., $\mathsf{rel}_{\mathcal{D}}(R) = \times_{p_i^s \in \mathsf{rel}_{\mathcal{D}}(V)} I(p_i)$. We drop the subscript $\mathcal{D}$, whenever it is clear from the context.

Example 38 Figure 12c depicts the relaxation $\mathsf{rel}(\mathcal{D})$ of the pMC $\mathcal{D}$ from Fig. 12a. For $R = [1/10, 4/5] \times [2/5, 7/10]$ and $u = (4/5, 3/5) \in R$ from Example 19, we obtain $\mathsf{rel}(R) = [1/10, 4/5] \times [2/5, 7/10] \times [2/5, 7/10]$ and $\mathsf{rel}(u) = (4/5, 3/5, 3/5)$. An instantiation $\mathsf{rel}(\mathcal{D})[\mathsf{rel}(u)]$ corresponds to $\mathcal{D}[u]$ as depicted in Fig. 5d. The relaxed region $\mathsf{rel}(R)$ contains also instantiations, e.g., $(4/5, 1/2, 3/5)$ which are not realisable in R.

For a pMC $\mathcal{D}$ and a graph-preserving region R, relaxation increases the set of possible instantiations: $\{\mathcal{D}[u] \mid u \in R\} \subseteq \{\mathsf{rel}(\mathcal{D})[u] \mid u \in \mathsf{rel}(R)\}$. Thus, the maximal reachability probability over all instantiations of $\mathcal{D}$ within R is bounded by the maximum over the instantiations of $\mathsf{rel}(\mathcal{D})$ within $\mathsf{rel}(R)$.

Lemma 3 *For pMC $\mathcal{D}$ and region R:*

$$\max_{u \in R} \left(Pr^{\mathcal{D}[u]}(\Diamond T)\right) = \max_{u \in R} \left(Pr^{\mathsf{rel}(\mathcal{D})[\mathsf{rel}(u)]}(\Diamond T)\right) \leq \max_{u \in \mathsf{rel}(R)} \left(Pr^{\mathsf{rel}(\mathcal{D})[u]}(\Diamond T)\right).$$

Consequently, if $\mathsf{rel}(\mathcal{D})$ satisfies a reachability property, so does $\mathcal{D}$.

Corollary 4 *For pMC $\mathcal{D}$ and region R:*

$$\max_{u \in \mathsf{rel}(R)} \left(Pr^{\mathsf{rel}(\mathcal{D})[u]}(\Diamond T)\right) \leq \lambda \text{ implies } \mathcal{D}, R \models \mathbb{P}_{\leq \lambda}(\Diamond T).$$

We now formalise the earlier observation: *Without parameter dependencies, finding optimal instantiations in a pMC is simpler*. Although $\mathsf{rel}(\mathcal{D})$ has (usually) more parameters than $\mathcal{D}$, finding an instantiation $u \in \mathsf{rel}(R)$ that maximises the reachability probability is simpler than in $u \in R$: For any $p_i^s \in \mathsf{rel}(V)$, we can in state s pick a value in $I(p_i^s)$ that maximises the probability to reach T from state s. There is no (negative) effect for the reachability probability at the other states as p_i^s only occurs at s. Optimal instantiations can thus be determined *locally* (at the states).

Furthermore, as both $\mathcal{D}$ is locally monotone, and there are no parameter dependencies, the maximum reachability probability is relatively easy to find: We only need to consider instantiations u that set the value of each parameter to either the lowest or highest possible value, i. e., $u(p_i^s) \in B(p_i^s)$ for all $p_i^s \in \mathsf{rel}(V)$:

Theorem 4 *Let $\mathcal{D}$ be a pMC with states S and $T \subseteq S$ and R a region subject subject to Assumption 1. There exists an instantiation $u \in \mathsf{rel}(R)$ satisfying $u(p_i^s) \in B(p_i^s)$ for all $p_i^s \in \mathsf{rel}(V)$ such that:*

$$Pr^{\mathsf{rel}(\mathcal{D})[u]}(\lozenge T) = \max_{v \in \mathsf{rel}(R)} Pr^{\mathsf{rel}(\mathcal{D})[v]}(\lozenge T).$$

To prove this statement, we consider an instantiation which assigns a value to a parameter strictly between its bounds. Any such instantiation can be modified such that all parameters are assigned to its bound, without decreasing the induced reachability probability. The essential statement is the monotonicity of a parameter without any further dependencies. The number of instantiations that must be analysed is therefore finite, compared for infinitely many candidates for non-relaxed pMCs.

Lemma 4 *Let $\mathcal{D}$ be a locally monotone pMC with a single parameter p that only occurs at one state $s \in S$, i.e. $\mathcal{P}(\hat{s}, s') \in [0, 1]$ for all $\hat{s}, s' \in S$ with $\hat{s} \neq s$. For region R and $T \subseteq S$, the probability $Pr^{\mathcal{D}}(\lozenge T)$ is monotonic on R.*

Proof W. l. o. g. let $s \notin T$ be the initial state of $\mathcal{D}$ and let T be reachable from s. Furthermore, let $\mathcal{U}$ denote the standard until-modality and $\neg T$ denote $S \setminus T$. Using the characterisation of reachability probabilities as linear equation system (cf. [13]), the reachability probability w. r. t. T (from the initial state) in $\mathcal{D}$ is given by:

$$\begin{aligned}
&\Pr^{\mathcal{D}}(\lozenge T) \\
&= \sum_{s' \in S} \mathcal{P}(s, s') \cdot \Pr^{\mathcal{D}}_{s'}(\lozenge T) \\
&= \sum_{s' \in S} \mathcal{P}(s, s') \cdot \left(\Pr^{\mathcal{D}}_{s'}(\neg s \,\mathcal{U}\, T) + \Pr^{\mathcal{D}}_{s'}(\neg T \,\mathcal{U}\, s) \cdot \Pr^{\mathcal{D}}(\lozenge T) \right) \\
&= \sum_{s' \in S} \mathcal{P}(s, s') \cdot \Pr^{\mathcal{D}}_{s'}(\neg s \,\mathcal{U}\, T) + \sum_{s' \in S} \mathcal{P}(s, s') \cdot \Pr^{\mathcal{D}}_{s'}(\neg T \,\mathcal{U}\, s) \cdot \Pr^{\mathcal{D}}(\lozenge T).
\end{aligned}$$

Transposing the equation yields

$$\Pr^{\mathcal{D}}(\lozenge T) = \frac{\sum_{s' \in S} \mathcal{P}(s, s') \cdot \Pr^{\mathcal{D}}_{s'}(\neg s \,\mathcal{U}\, T)}{1 - \sum_{s' \in S} \mathcal{P}(s, s') \cdot \Pr^{\mathcal{D}}_{s'}(\neg T \,\mathcal{U}\, s)}.$$

The denominator can not be zero as T is reachable from s. Since $\mathcal{D}$ is locally monotone, we have $\mathcal{P}(s, s') = f_{s'}/g_s$ for $s' \in S$ and multilinear functions $f_{s'}, g_s \in \mathbb{Q}[p]$. We obtain:

$$\Pr^{\mathcal{D}}(\lozenge T) = \frac{\sum_{s' \in S} f_{s'} \cdot \overbrace{\Pr^{\mathcal{D}}_{s'}(\neg s \,\mathcal{U}\, T)}^{\text{constant}}}{g_s - \sum_{s' \in S} f_{s'} \cdot \underbrace{\Pr^{\mathcal{D}}_{s'}(\neg T \,\mathcal{U}\, s)}_{\text{constant}}}.$$

Hence, $\Pr^{\mathcal{D}}(\lozenge T) = f_1/f_2$ is a fraction of two multilinear functions $f_1, f_2 \in \mathbb{Q}[p]$ and therefore monotonic on R. □

Proof of Theorem 4 We prove the statement by contraposition. Let $u \in \mathsf{rel}(R)$ with $\Pr^{\mathsf{rel}(\mathcal{D})[u]}(\lozenge T) = \max_{v \in \mathsf{rel}(R)} \left(\Pr^{\mathsf{rel}(\mathcal{D})[v]}(\lozenge T) \right)$. For the contraposition, assume that there exists a parameter $p \in \mathsf{rel}(V)$ with $u(p) \in I_R(p) \setminus B_R(p)$ such that all instantiations

$u' \in \mathsf{rel}(R)$ that set p to a value in $B_R(p)$ induce a smaller reachability probability, i.e. $u'(p) \in B_R(p)$ and $u'(q) = u(q)$ for $q \neq p$ implies

$$\Pr^{\mathsf{rel}(\mathcal{D})[u']}(\Diamond T) < \Pr^{\mathsf{rel}(\mathcal{D})[u]}(\Diamond T).$$

Consider the pMC $\hat{\mathcal{D}} = (S, \{p\}, s, \hat{\mathcal{P}})$ with the single parameter p that arises from $\mathsf{rel}(\mathcal{D})$ by replacing all parameters $q \in \mathsf{rel}(V) \setminus \{p\}$ with $u(q)$. We have $\hat{\mathcal{D}}[u] = \mathsf{rel}(\mathcal{D})[u]$. Moreover, $\Pr^{\hat{\mathcal{D}}}(\Diamond T)$ is monotonic on $I(p)$ according to Lemma 4. Thus, there is an instantiation $u' \in \mathsf{rel}(R)$ with $u'(p) \in B_R(p)$ and $u'(q) = u(q)$ for $q \neq p$ satisfying

$$\Pr^{\hat{\mathcal{D}}[u]}(\Diamond T) \leq \Pr^{\hat{\mathcal{D}}[u']}(\Diamond T) = \Pr^{\mathsf{rel}(\mathcal{D})[u']}(\Diamond T).$$

This contradicts our assumption for parameter p. □

7.3 Replacing parameters by nondeterminism

In order to determine $\max_{u \in \mathsf{rel}(R)} \Pr^{\mathsf{rel}(\mathcal{D})[u]}(\Diamond T)$, it suffices to make a discrete choice over instantiations $u\colon \mathsf{rel}(V) \to \mathbb{R}$ with $u(p_i^s) \in B(p_i)$. This choice can be made locally at every state, which brings us to the key idea of *constructing a (non-parametric) MDP out of the pMC $\mathcal{D}$ and the region R*, where nondeterministic choices represent all instantiations that have to be considered. In the following, it is convenient to refer to the parameters in a given state s by:

$$V_s = \{\, p \in V \mid p \text{ occurs in } \mathcal{D}(s, s') \text{ for some } s' \in S \,\}.$$

Definition 27 (*Substitution (pMCs)*) For pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$ and region R, let the MDP $\mathsf{sub}_R(\mathcal{D}) = (S, s_I, Act_{\mathsf{sub}}, \mathcal{P}_{\mathsf{sub}})$ with

- $Act_{\mathsf{sub}} = \biguplus_{s \in S} Act_s$ where

$$Act_s = \{u\colon V_s \to \mathbb{R} \mid \forall p \in V_s.\ u(p) \in B(p)\,\}, \text{ and}$$

-

$$\mathcal{P}_{\mathsf{sub}}(s, u, s') = \begin{cases} \mathcal{P}(s, s')[u] & \text{if } u \in Act_s, \\ 0 & \text{otherwise.} \end{cases}$$

be the *(parameter-)substitution of $\mathcal{D}$ and R*.

Thus, choosing action u in s corresponds to assigning one of the extremal values $B(p_i)$ to the parameters p_i^s. The number of outgoing actions from state s is therefore $2^{|V_s|}$.

Example 39 Consider pMC $\mathcal{D}$—depicted in Fig. 12a—with $R = [1/10, 4/5] \times [2/5, 7/10]$ as before. The substitution of $\mathcal{D}$ and R is shown in Fig. 13a. In $\mathcal{D}$, each outgoing transition of states s_0, s_1, s_2 is replaced by a nondeterministic choice in MDP $\mathsf{sub}_R(\mathcal{D})$. That is, 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. For the states s_3 and s_4, the choice is unique as their outgoing transitions in $\mathcal{D}$ are constant. Figure 13b depicts the MC $\mathsf{sub}_R(\mathcal{D})^\sigma$ which is induced by the strategy σ on MDP $\mathsf{sub}_{\mathcal{D}}(R)$ that chooses the upper bounds at s_0 and s_2, and the lower bound at s_1. Notice that $\mathsf{sub}_R(\mathcal{D})^\sigma$ coincides with $\mathsf{rel}(\mathcal{D})[v]$ for a suitable instantiation v, as depicted in Fig. 12c.

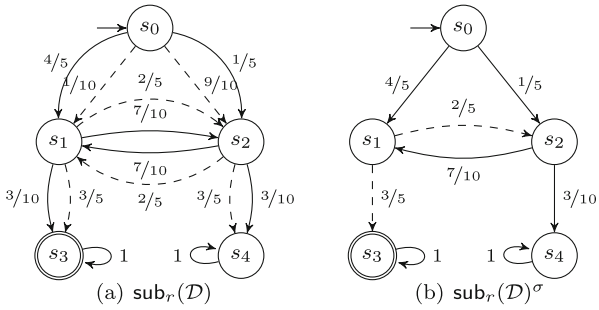


Fig. 13 Illustrating parameter-substitution

The substitution encodes the local choices for a relaxed pMC. That is, for an arbitrary pMC, there is a one-to-one correspondence between strategies σ in the MDP $\mathsf{sub}_{\mathsf{rel}(R)}(\mathsf{rel}(\mathcal{D}))$ and instantiations $u \in \mathsf{rel}(R)$ for $\mathsf{rel}(\mathcal{D})$ with $u(p_i^s) \in B(p_i)$. For better readability, we will omit the superscripts for sets of strategies Str. Combining these observations with Theorem 4, yields the following.

Corollary 5 *For a pMC $\mathcal{D}$, a graph-preserving region R, and a set T of target states of $\mathcal{D}$:*

$$\max_{u \in R} Pr^{\mathcal{D}[u]}(\Diamond T) \leq \max_{\sigma \in Str} Pr^{sub_{rel(R)}(rel(\mathcal{D}))^\sigma}(\Diamond T)$$
$$\min_{u \in R} Pr^{\mathcal{D}[u]}(\Diamond T) \geq \min_{\sigma \in Str} Pr^{sub_{rel(R)}(rel(\mathcal{D}))^\sigma}(\Diamond T).$$

Furthermore, the nondeterministic choices introduced by the substitution only depend on the values $B(p_i)$ of the parameters p_i in R. Since the ranges of the parameters p_i^s in $\mathsf{rel}(R)$ agree with the range of p_i in R, we have

$$\mathsf{sub}_{\mathsf{rel}(R)}(\mathsf{rel}(\mathcal{D})) = \mathsf{sub}_R(\mathcal{D}) \quad \text{for all graph-preserving } R. \tag{15}$$

A direct consequence of these statements yields:

Theorem 5 *Let $\mathcal{D}$ be a pMC, R a graph-preserving region, φ a reachability property, subject to Assumption 1. Then it holds:*

$$\forall \sigma \in Str.\ sub_R(\mathcal{D})^\sigma \models \varphi \implies \mathcal{D}, R \models \varphi \quad \wedge$$
$$\forall \sigma \in Str.\ sub_R(\mathcal{D})^\sigma \models \neg\varphi \implies \mathcal{D}, R \models \neg\varphi.$$

Hence, we can deduce via Algorithm 4 whether $\mathcal{D}, R \models \varphi$ by applying standard techniques for MDP model checking to $\mathsf{sub}_R(\mathcal{D})$, such as value- and policy iteration, cf. [13, 121]. We stress that while the relaxation is key for showing the correctness, equation (15) proves that this step does not actually need to be performed.

Example 40 Reconsider Example 39. From $\mathsf{sub}_R(\mathcal{D})$ in Fig. 13a, we can derive $\max_{\sigma \in Str} \Pr^{\mathsf{sub}_R(\mathcal{D})^\sigma}(\Diamond T) = 47/60$ and, by Theorem 5, $\mathcal{D}, R \models \mathbb{P}_{\leq 4/5}(\Diamond T)$ follows. Despite the large region R, we establish a non-trivial upper bound on the reachability probability over all instantiations in R.

If the over-approximation by region R is too coarse for a conclusive answer, region R can be refined, meaning that we split R into a set of smaller regions[13] [30]. We discuss

[13] Strictly speaking, these regions will overlap as we always consider closed regions. This is not a concern for correctness. When splitting, we may take this information into account, see Sect. 9.2.2.

Algorithm 4 Parameter lifting

reachability(pMC $\mathcal{D}$, $T \subseteq S$, region R, specification $\mathbb{P}_{\leq\lambda}(\Diamond T)$)
  // Check whether $\mathcal{D}$, $R \models \mathbb{P}_{\leq\lambda}(\Diamond T)$
  Construct $\mathsf{sub}_R(\mathcal{D})$
  if $\forall \sigma \in Str\ \mathsf{sub}_R(\mathcal{D}) \models \mathbb{P}_{\leq\lambda}(\Diamond T)$ **then**
    // via standard MDP model checking procedures
    return true
  else if $\forall \sigma \in Str\ \mathsf{sub}_R(\mathcal{D}) \models \mathbb{P}_{>\lambda}(\Diamond T)$ **then**
    // via standard MDP model checking procedures
    return false
  else
    return unknown

splitting strategies in Sect. 9. Intuitively, as more potential parameter values are excluded by reducing the region size, the actual choice of the parameter value has less impact on reachability probabilities. The smaller the region gets, the smaller the over-approximation: The optimal instantiation on the pMC $\mathcal{D}$ is over-approximated by some strategy on $\mathsf{sub}_R(\mathcal{D})$. The approximation error originates from choices where an optimal strategy on $\mathsf{sub}_R(\mathcal{D})$ chooses actions u_1 and u_2 at states s_1 and s_2, respectively, with $u_1(p_i^{s_1}) \neq u_2(p_i^{s_2})$ for some parameter p_i, and therefore intuitively disagree on its value. The probability mass that is affected by these choices decreases the smaller the region is. For infinitesimally small regions, the error from the over-approximation vanishes, as the actions for the upper and the lower bound of a parameter become equal up to an infinitesimal. More formally, the difference in reachability probability between two MCs corresponding to instantiations in a region tends is bounded and tends to zero if the region gets smaller [45, Lemma 9].

7.4 Expected reward properties

The reduction of bounding reachability probabilities on pMCs to off-the-shelf MDP model checking can also be applied to bound expected rewards. To see this, we have to extend the notion of locally monotone parametric Markov chains.

Definition 28 (*Locally monotone reward pMC*) A pMC $\mathcal{D} = (S, V, s_I, \mathcal{P})$ with reward function $\mathrm{rew}\colon S \to \mathbb{Q}(V)$ is *locally monotone* iff for all $s \in S$, there is a multilinear polynomial $g_s \in \mathbb{Q}[V]$ with

$$\{\mathrm{rew}(s), \mathcal{P}(s, s') \mid s' \in S\} \subseteq \{f/g_s \mid f \in \mathbb{Q}[V] \text{ multilinear}\}.$$

We now generalise relaxation and substitution to the reward models, and obtain analogous results.

Definition 29 (*Substitution for reward pMCs*) Let $\mathcal{D} = (S, V, s_I, \mathcal{P})$ be a pMC, $\mathrm{rew}\colon S \to \mathbb{Q}(V)$ a reward function, $T \subseteq S$ a set of target states, and R a region. For $s \in S$, let

$$V_s^{\mathrm{rew}} = V_s \cup \{p_i \in V \mid p_i \text{ occurs in } \mathrm{rew}(s)\}.$$

The MDP $\mathsf{sub}_R^{\mathrm{rew}}(\mathcal{D}) = (S, s_I, Act_{\mathsf{sub}}^{\mathrm{rew}}, \mathcal{P}_{\mathsf{sub}}^{\mathrm{rew}})$ with reward function $\mathrm{rew}_{\mathsf{sub}}$ is the *(parameter-)substitution of* $\mathcal{D}$, rew *on* R, where

– $Act_{\mathsf{sub}}^{\mathrm{rew}}$ and $\mathcal{P}_{\mathsf{sub}}^{\mathrm{rew}}$ are analogous to Definition 27, but over V_s^{rew}.

– rew_{sub} is given by:

$$(s, u) \mapsto \begin{cases} \text{rew}(s)[u] & \text{if } u \in Act_s^{\text{rew}}, \\ 0 & \text{otherwise,} \end{cases}$$

where Act_s^{rew} is defined analogously to Act_s in Definition 27.

The reward approximation of a pMC can be used to identify regions as accepting or rejecting for expected reward properties.

Theorem 6 *Let $\mathcal{D}$ be a pMC with locally monotone rewards rew, R a region, and φ an expected reward property, subject to Assumption 1:*

$$\forall \sigma \in Str.\ sub_R^{rew}(\mathcal{D}) \models \varphi \text{ implies } \mathcal{D}, R \models \varphi \text{ and}$$
$$\forall \sigma \in Str.\ sub_R^{rew}(\mathcal{D}) \models \neg\varphi \text{ implies } \mathcal{D}, R \models \neg\varphi.$$

The proof is analogous to the proof of Theorem 5.

8 Model-checking-based region verification of parametric MDPs

In the previous section, we approximated reachability probabilities in (locally-monotone) pMCs by considering the substitution MDP, see Definition 27. The non-determinism in the MDP encodes the finitely many parameter valuations that approximate the reachability probabilities in the pMC. By letting an adversary player resolve the non-determinism in the MDP, we obtain bounds on the reachability probabilities in the pMC. These bounds can efficiently be computed by standard MDP model checking.

In this section, we generalise the approach to pMDPs, which already contain non-determinism. The result naturally leads to a 2-player stochastic game: One player controls the nondeterminism inherent to the MDP, while the other player controls the (abstracted) parameter values. Letting the two players adequately minimise and/or maximise the reachability probabilities in the SG yields bounds on the minimal (and maximal) reachability probabilities in the pMDP. For example, if the player for the original non-determinism maximises and the parameter player minimises, we obtain a lower bound on the maximal probability. These bounds can efficiently be computed by standard SG model checking procedures.

In our presentation below, we discuss the interplay of the two sources of non-determinism. In particular, we show how the generalisation of the method yields an additional source of (over-)approximation. Then, we formalise the construction of the substitution with nondeterminism, analogous to the pMCs from the previous section. In particular, Definition 30 is analogous to Definition 27 and Theorem 7 is analogous to Theorem 5. We do not repeat relaxation, described in Sect. 7.2, as—as also discussed in the previous section—it is not a necessary ingredient for the correctness of the approach.

8.1 Two types of approximation

In the following, let $\mathcal{M} = (S, V, s_I, Act, \mathcal{P})$ be a pMDP and R a graph-preserving, rectangular, closed region.

Demonic strategies We analyse R with respect to the demonic relation $\models_d$. We have:

$$\mathcal{M}, R \models_d \varphi \iff \forall u \in R.\ \forall \sigma \in Str^{\mathcal{M}}.\ \mathcal{M}[u]^{\sigma} \models \varphi.$$

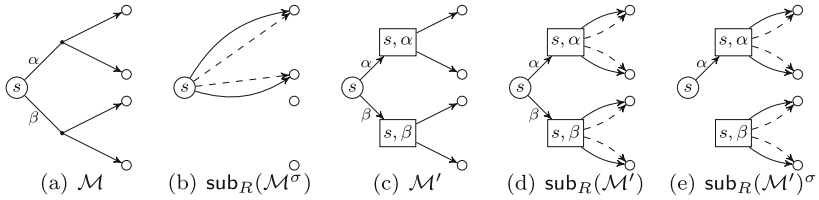


Fig. 14 Illustration of the substitution of a pMDP

The two universal quantifiers can be reordered, and in addition $\mathcal{M}[u]^\sigma = \mathcal{M}^\sigma[u]$. We obtain:

$$\mathcal{M}, R \models_d \varphi \iff \forall \sigma \in Str^{\mathcal{M}}. \forall u \in R. \underbrace{\mathcal{M}^\sigma[u]}_{\text{a pMC}} \models \varphi$$

Intuitively, the reformulation states that we have to apply pMC region verification on $\mathcal{M}^\sigma$ and R for all $\sigma \in Str^{\mathcal{M}}$. We now want to employ parameter lifting for each strategy. Thus, we want to consider the verification of the substituted pMCs $\mathsf{sub}_R(\mathcal{M}^\sigma)$. As these substituted pMCs share most of their structure, the set of all such substituted pMCs can be concisely represented as an SG, in which both players cooperate (as witnessed by the same quantifiers). In the scope of this paper, an SG with cooperating players can be concisely represented as an MDP. Consequently, for the demonic relation, pMDP verification can be approximated by MDP model checking.

Angelic strategies We now turn our attention to the angelic relation $\models_a$, cf. Definition 14.

$$\mathcal{M}, R \models_a \varphi \iff \forall u \in R. \exists \sigma \in Str^{\mathcal{M}}. \mathcal{M}[u]^\sigma \models \varphi.$$

Here, we cannot simply reorder the quantifiers. However:

$$\exists \sigma \in Str^{\mathcal{M}}. \forall u \in R. \mathcal{M}^\sigma[u] \models \varphi \implies \mathcal{M}, R \models_a \varphi.$$

Now, the left-hand side expresses again that we want to do region verification for pMCs induced by a strategy, as in the demonic case, and that we likewise want to represent by a stochastic game. As witnessed by the quantifier alternation, this SG does not reduce to an MDP; the two players have opposing objectives. Nevertheless, we can efficiently analyse this SG (with a variant of value iteration), and thus the left-hand side of the implication above.

Observe that the over-approximation actually computes a robust strategy, as discussed in Remark 7. In particular, we now have two sources of approximation:

- The approximation that originates from dropping parameter dependencies (as also in the demonic case).
- The application of the substitution of parameters with non-determinism on robust strategies rather than of the actual angelic relation.

Both over-approximations vanish with declining region size.

8.2 Replacing parameters by nondeterminism

Example 41 Consider the pMDP $\mathcal{M}$ in Fig. 14a, where the state s has two enabled actions α and β. The strategy σ given by $\{s \mapsto \alpha\}$ applied to $\mathcal{M}$ yields a pMC, which is subject to substitution, cf. Fig. 14b.

The parameter substitution of a pMDP (cf. Fig. 14a) yields an SG—as in Fig. 14d. It represents, for all strategies of the pMDP, the parameter-substitution (as in Definition 27) of each induced pMC. To ensure that in the SG each state can be assigned to a unique player, we split states in the pMDP which have both (parametric) probabilistic branching and non-determinism, such that states have either probabilistic branching or non-determinism, but not both. The reformulation is done as follows: After each choice of actions, auxiliary states are introduced, such that the outcome of the action becomes deterministic and the probabilistic choice is delayed to the auxiliary state. This construction is similar to the conversion of Segala's probabilistic automata into Hansson's alternating model [127]. More precisely, we

- split each state $s \in S$ into $\{s\} \uplus \{\langle s, \alpha\rangle \mid \alpha \in Act(s)\}$,
- add a transition with probability one for each $s \in S$ and $\alpha \in Act(s)$. The transition leads from s to $\langle s, \alpha\rangle$, and
- move the probabilistic choice at s w. r. t. α to $\langle s, \alpha\rangle$.

Applying this to the pMDP from Fig. 14a, we obtain the pMDP $\mathcal{M}'$ in Fig. 14c, where the state s has only *nondeterministic* choices leading to states of the form $\langle s, \alpha\rangle$ with only *probabilistic* choices. The subsequent substitution on the probabilistic states yields the SG $\mathsf{sub}_R(\mathcal{M}')$, where one player represents the nondeterminism of the original pMDP $\mathcal{M}$, while the other player decides whether parameters should be set to their lower or upper bound in the region R. For the construction, we generalise V_s to state-action pairs: For a pMDP, a state s and action α, let

$$V_{s,\alpha} = \{\, p \in V \mid p \text{ occurs in } \mathcal{P}(s, \alpha, s') \text{ for some } s' \in S \,\}.$$

Definition 30 (*Substitution (pMDPs)*) For pMDP $\mathcal{M} = (S, V, s_I, Act, \mathcal{P})$ and region R, let SG

$$\mathsf{sub}_R(\mathcal{M}) = (S_{\bigcirc} \uplus S_{\square}, s_I, Act_{\mathsf{sub}}, \mathcal{P}_{\mathsf{sub}})$$

with

- $S_{\bigcirc} = S$
- $S_{\square} = \{\langle s, \alpha\rangle \mid \alpha \in Act(s)\}$,
- $Act_{\mathsf{sub}} = Act \uplus \left(\biguplus_{\langle s,\alpha\rangle \in S_{\square}} Act_s^{\alpha}\right)$ where

$$Act_s^{\alpha} = \{u \colon V_{s,\alpha} \to \mathbb{R} \mid u(p) \in B(p)\ \forall p \in V_{s,\alpha}\},$$

and,

- $$\mathcal{P}_{\mathsf{sub}}(t, \beta, t') = \begin{cases} 1 & \text{if } t \in S_{\bigcirc}, \beta \in Act(t), t' = \langle t, \beta\rangle \in S_{\square}, \\ \mathcal{P}(s, \alpha, t')[\beta] & \text{if } t = \langle s, \alpha\rangle \in S_{\square}, \beta \in Act_s^{\alpha}, t' \in S_{\bigcirc}, \\ 0 & \text{otherwise.} \end{cases}$$

be the *(parameter-)substitution of* $\mathcal{M}$ *and* R.

We relate the SG $\mathsf{sub}_R(\mathcal{M})$ under different strategies for player $\bigcirc$ with the substitution in the strategy-induced pMCs of $\mathcal{M}$. We observe that the strategies for player $\bigcirc$ in $\mathsf{sub}_R(\mathcal{M})$ coincide with strategies in $\mathcal{M}$. Consider the induced MDP $(\mathsf{sub}_R(\mathcal{M}))^{\sigma}$ with a strategy σ for player $\bigcirc$. The MDP $(\mathsf{sub}_R(\mathcal{M}))^{\sigma}$ is obtained from $\mathsf{sub}_R(\mathcal{M})$ by erasing transitions not agreeing with σ. In $(\mathsf{sub}_R(\mathcal{M}))^{\sigma}$ player $\bigcirc$-state have a single enabled action, while player $\square$-states have multiple available enabled actions.

Example 42 Continuing Example 41, applying strategy σ to $\mathsf{sub}_R(\mathcal{M})$ yields $(\mathsf{sub}_R(\mathcal{M}))^\sigma$, see Fig. 14e. The MDP $(\mathsf{sub}_R(\mathcal{M}))^\sigma$ matches the MDP $\mathsf{sub}_R(\mathcal{M}^\sigma)$ apart from intermediate states of the form $\langle s, \alpha\rangle$: The outgoing transitions of s in $\mathsf{sub}_R(\mathcal{M}^\sigma)$ coincide with the outgoing transitions of $\langle s, \alpha\rangle$ in $(\mathsf{sub}_R(\mathcal{M}))^\sigma$, where $\langle s, \alpha\rangle$ is the unique successor of s.

The following corollary formalises that $(\mathsf{sub}_R(\mathcal{M}))^\sigma$ and $\mathsf{sub}_R(\mathcal{M}^\sigma)$ induce the same reachability probabilities.

Corollary 6 *For pMDP $\mathcal{M}$, graph-preserving region R, target states $T \subseteq S$, and strategies $\sigma \in Str_{\bigcirc}^{sub_R(\mathcal{M})}$ and $\rho \in Str^{sub_R(\mathcal{M}^\sigma)}$, it holds that*

$$Pr^{(sub_R(\mathcal{M}^\sigma))^\rho}(\Diamond T) = Pr^{sub_R(\mathcal{M})^{\sigma,\widehat{\rho}}}(\Diamond T)$$

with $\widehat{\rho} \in Str_{\square}^{sub_R(\mathcal{M})}$ satisfies $\widehat{\rho}(\langle s, \sigma(s)\rangle) = \rho(s)$.

Instead of performing the substitution on the pMC induced by $\mathcal{M}$ and σ, we can perform the substitution on $\mathcal{M}$ directly and preserve the reachability probability.

Consequently, and analogously to the pMC case (cf. Theorem 5), we can derive whether $\mathcal{M}, R \models_{\clubsuit} \varphi$ by analysing a stochastic game. For this, we consider various standard variants of model checking on stochastic games.

Definition 31 (*Model-relation on SGs*) For an SG $\mathcal{G}$, property φ, and quantifiers $\mathcal{Q}_1$, $\mathcal{Q}_2$, we define $\mathcal{G} \models^{\mathcal{Q}_1,\mathcal{Q}_2} \varphi$ as:

$$\mathcal{Q}_1\sigma_{\bigcirc} \in Str_{\bigcirc}^{\mathsf{sub}_R(\mathcal{M})}.\ \mathcal{Q}_2\sigma_{\square} \in Str_{\square}^{\mathsf{sub}_R(\mathcal{M})} \quad \mathcal{G}^{\sigma_{\bigcirc},\sigma_{\square}} \models \varphi$$

The order of players, for these games, does not influence the outcome [48, 128].

Theorem 7 *Let $\mathcal{M}$ be a pMDP, R a region, and φ a reachability property, subject to Assumption 1*[14]*. Then:*

$$sub_R(\mathcal{M}) \models^{\forall,\forall} \varphi \text{ implies } \mathcal{M}, R \models_d \varphi, \text{ and}$$
$$sub_R(\mathcal{M}) \models^{\exists,\forall} \varphi \text{ implies } \mathcal{M}, R \models_a \varphi.$$

Proof We only prove the second statement using $\varphi = \mathbb{P}_{>\lambda}(\Diamond T)$, other reachability properties are similar. A proof for the (simpler) first statement can be derived in an analogous manner. We have that $\mathcal{M}, R \models_a \mathbb{P}_{>\lambda}(\Diamond T)$ iff for all $u \in R$ there is a strategy σ of $\mathcal{M}$ for which the reachability probability in the MC $\mathcal{M}^\sigma[u]$ exceeds the threshold λ, i. e.,

$$\mathcal{M}, R \models_a \mathbb{P}_{>\lambda}(\Diamond T) \iff \min_{u\in R} \max_{\sigma \in Str^{\mathcal{M}}} \Pr^{\mathcal{M}^\sigma[u]}(\Diamond T) > \lambda.$$

A lower bound for this probability is obtained as follows:

$$\begin{aligned}
&\min_{u\in R} \max_{\sigma\in Str^{\mathcal{M}}} \left(\Pr^{\mathcal{M}^\sigma[u]}(\Diamond T)\right)\\
&\geq \max_{\sigma\in Str^{\mathcal{M}}} \min_{u\in R} \left(\Pr^{\mathcal{M}^\sigma[u]}(\Diamond T)\right)\\
&\overset{*}{\geq} \max_{\sigma\in Str^{\mathcal{M}}} \min_{\rho\in Str^{\mathsf{sub}_R(\mathcal{M}^\sigma)}} \left(\Pr^{(\mathsf{sub}_R(\mathcal{M}^\sigma))^\rho}(\Diamond T)\right)\\
&\overset{**}{=} \max_{\sigma\in Str_{\bigcirc}^{\mathsf{sub}_R(\mathcal{M})}} \min_{\rho\in Str_{\square}^{\mathsf{sub}_R(\mathcal{M})}} \left(\Pr^{\mathsf{sub}_R(\mathcal{M})^{\sigma,\rho}}(\Diamond T)\right).
\end{aligned}$$

[14] Straightforwardly lifting locally monotone pMCs to locally monotone pMDPs

The inequality $*$ is due to Corollary 5. The equality $**$ holds by Corollary 6. Then:

$$
\begin{aligned}
&\mathsf{sub}_R(\mathcal{M}) \models^{\exists,\forall} \mathbb{P}_{>\lambda}(\Diamond T) \\
&\quad\iff \exists \sigma \in Str_{\bigcirc}^{\mathsf{sub}_R(\mathcal{M})}.\ \forall \rho \in Str_{\square}^{\mathsf{sub}_R(\mathcal{M})} \\
&\qquad \mathcal{G}^{\sigma,\rho} \models \mathbb{P}_{>\lambda}(\Diamond T) \\
&\quad\iff \max_{\sigma \in Str_{\bigcirc}^{\mathcal{G}}} \Big(\min_{\rho \in Str_{\square}^{\mathcal{G}}} \big(\Pr^{\mathcal{G}^{\sigma,\rho}}(\Diamond T)\big)\Big) > \lambda \\
&\quad\implies \min_{u \in R} \max_{\sigma \in Str^{\mathcal{M}}} \big(\Pr^{\mathcal{M}^{\sigma}[u]}(\Diamond T)\big) > \lambda \\
&\quad\iff \mathcal{M}, R \models_a \mathbb{P}_{>\lambda}(\Diamond T).
\end{aligned}
$$

□

9 Approximate synthesis by parameter space partitioning

Parameter space partitioning is our iterative approach to the approximate synthesis problem. It builds on top of region verification, discussed above, and is, conceptually, independent of the methods used for verification discussed later.

Parameter space partitioning is best viewed as a counter-example guided abstraction refinement (CEGAR)-like [47] approach to successively divide the parameter space into accepting and rejecting regions. The main idea is to compute a sequence $\left(R_a^i\right)_i$ of simple accepting regions that successively extend each other. Similarly, an increasing sequence $\left(R_r^i\right)_i$ of simple rejecting regions is computed. At the i-th iteration, $R^i = R_a^i \cup R_r^i$ is the covered fragment of the parameter space. The iterative approach halts when R^i is at least $c\%$ of the entire parameter space. Termination is guaranteed: in the limit a solution to the exact synthesis problem is obtained as $\lim_{i\to\infty} R_a^i = R_a$ and $\lim_{i\to\infty} R_r^i = R_r$.

Let us describe the synthesis loop for the approximate synthesis as depicted in Fig. 4 in detail. In particular, we discuss how to generate *candidate regions* that can be dispatched to the verifier along with a *hypothesis* whether the candidate region is accepting or rejecting. We focus on *rectangular regions* for several reasons:

- the automated generation of rectangular regions is easier to generalise to multiple dimensions,
- earlier experiments [65] revealed that rectangular regions lead to a more efficient SMT-based verification of regions (described in Sect. 6), and
- model-checking based region verification (described in Sect. 7) requires rectangular regions.

A downside of rectangular regions is that they are neither well-suited to approximate a region partitioning given by a diagonal, nor to cover well-defined regions that are not rectangular themselves.

Remark 11 In the following, we assume that the parameter space is given by a rectangular well-defined region R. If the parameter space is not rectangular, we over-approximate R by a rectangular region $\hat{R} \supseteq R$. If the potential over-approximation of the parameter space $\hat{R}$ is not well-defined, then we iteratively approximate $\hat{R}$ by a sequence of well-defined and ill-defined[15] regions. The regions in the sequence of well-defined regions are then subject

[15] A region R is ill-defined if *no* instantiation in R is well-defined.

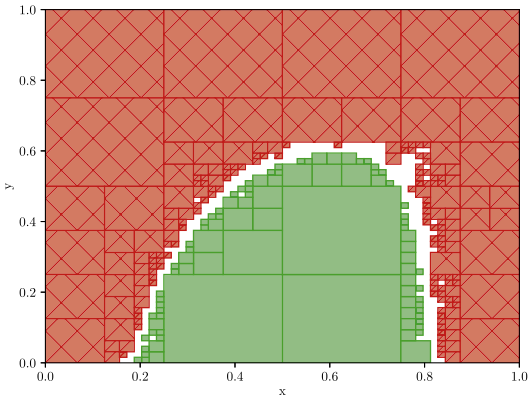


Fig. 15 Parameter space partitioning into safe and unsafe regions

to the synthesis problem. Constructing the sequence of regions is done analogously to the partitioning into accepting and rejecting regions.

Before we present the procedure in full detail, we first outline a naive refinement procedure by means of an example.

Example 43 (Naive refinement loop) Consider the parametric die from Example 5. Suppose we want to synthesise the partitioning as depicted in Fig. 2. We start by verifying the full parameter space R against φ. The verifier returns `false`, as R is not accepting. Since R (based on our knowledge at this point) might be rejecting, we invoke the verifier with R and $\neg\varphi$, yielding `false` too. Thus, the full parameter space R is inconsistent. We now split R into four equally-sized regions, all of which are inconsistent. Only after splitting again, we find the first accepting and rejecting regions. After various iterations, the procedure leads to the partitioning in Fig. 15.

Algorithm 5 describes this naive region partitioning procedure. It takes a pSG, a region R, a specification φ, and a (demonic or angelic) satisfaction relation as input. It first initialises a (priority) queue Q with R. In each iteration, a subregion R' of R is taken from the queue, the counter i is incremented, and the sequence of accepted and rejected regions is updated. There are three possibilities. Either R' is accepting (or rejecting), and R_a^i (R_r^i) extends R_a^{i-1} (R_r^{i-1}) with R', or R' is inconsistent. In the latter case, we split R' into a finite set of subregions that are inserted into the queue Q. Regions that are not extended are unchanged.

The algorithm only terminates if R_a and R_r are a finite union of hyper-rectangles. However, the algorithm can be terminated after any iteration yielding a sound approximation. The algorithm ensures $\lim_{i\to\infty} R^i = R$, if we order Q according to the size of the regions. We omit the technical proof here; the elementary property is that the regions are Lebesgue-measurable (and have a positive measure by construction).

The naive algorithm has a couple of structural weaknesses:

- It invokes the verification algorithm twice to determine that the full parameter space is inconsistent.
- It does not provide any (diagnostic) information from a verification invocation yielding `false`.
- It checks whether a region is accepting before it checks whether it is rejecting. This order is suboptimal if the region is rejecting.

Algorithm 5 Naive refinement loop

naive-refinement(pSG $\mathcal{G}$, rectangular region R, $\clubsuit \in \{a, d\}$, specification φ)
  $i := 0$
  $Q := \{R\}$, $R_a^i := \emptyset$, $R_r^i := \emptyset$
  while $Q \neq \emptyset$ **do**
    $i := i + 1$
    $R' := Q$.pop
    if $\mathcal{G}, R' \models_\clubsuit \varphi$ **then**
      $R_a^i := R_a^{i-1} \cup R'$, $R_r^i := R_r^{i-1}$
    else if $\mathcal{G}, R' \models_\clubsuit \neg\varphi$ **then**
      $R_a^i := R_a^{i-1}$, $R_r^i := R_r^{i-1} \cup R'$
    else
      $R_a^i := R_a^{i-1}$, $R_r^i := R_r^{i-1}$
      $Q := Q \cup$ **split**(R')
  return Accepting region R_a^i, Rejecting region R_r^i

- If the region is inconsistent, it splits the region into 2^n equally large regions. Instead, it might be beneficial to select a smaller number of regions (only split in one dimension).
- Uninformed splitting yields many inconsistent subregions. Splitting in only one dimension even increases the number of verification calls yielding `false`.

In the remainder of this section, we discuss ways to alleviate these weaknesses. The proposed improvements are based on empirical observations about the benchmarks and are in line with the implementation in our tool `PROPhESY`. In particular, we tailor the heuristics to "well-behaved" models and specifications, which reflect the benchmarks from various domains. The notion of being well-behaved refers to

- a limited number of connected accepting and rejecting regions with smooth (albeit highly non-linear) borders between these regions.
- a limited number of accepting (rejecting) instantiations that are close to a rejecting (accepting) instantiations. We call instantiations that form a border between R_a and R_r *border* instantiations.

The parameter space depicted in Fig. 15 is well-behaved. It features only two connected regions, with a smooth border between them. Furthermore, the regions have a considerable interior, or equivalently, many instantiations are not too close to the border. We remark that we do *rely* on these assumptions to hold, but `PROPhESY` will be slow on models that are not well-behaved.

9.1 Sampling

A simple but effective improvement is to verify an instantiated model $\mathcal{G}[u]$ for some instantiation (a *sample*) $u \in R$. The verification result either reveals that the region is not accepting, if $\mathcal{G}[u] \not\models_\clubsuit \varphi$, or not rejecting, if $\mathcal{G}[u] \models_\clubsuit \varphi$. Two samples within a region R may suffice to conclude that R is inconsistent. In order to quickly find inconsistent regions by sampling, it is beneficial to seek for border instantiations. To this end, a good strategy is to start with a coarse (random) sampling to get a first indication of border instantiations. We then select additional instantiations by intra-/extrapolation between these samples.

Example 44 We discuss how sampling may improve the naive refinement loop as discussed in Example 43. Fig. 16a shows a uniform sampling. Red crosses indicate that the instanti-

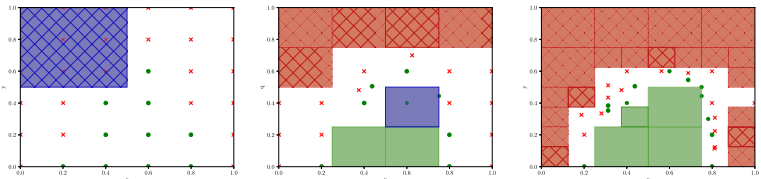


Fig. 16 Parameter space partitioning in progress: Images generated by `PROPhESY`

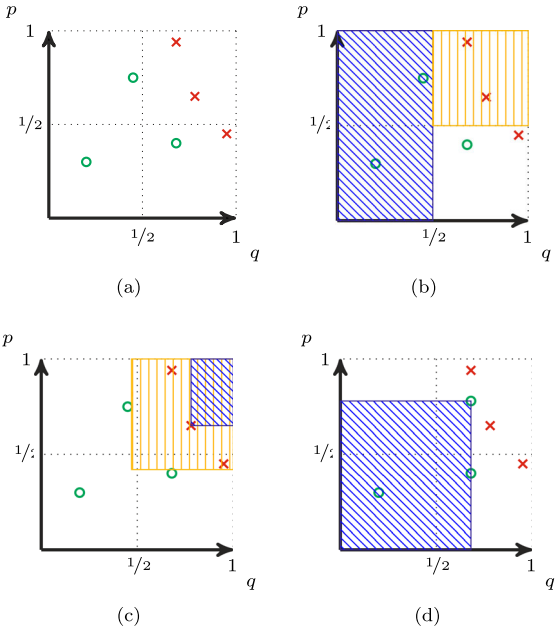


Fig. 17 Creating region candidates based on samples

ated pMC satisfies $\neg\varphi$, while green dots indicate that the instantiation satisfies φ. The blue rectangle is a candidate region (with the hypothesis $\neg\varphi$, indicated by the hatching), which is consistent with all samples.

9.2 Finding region candidates

We use the sampling results to steer the selection of a candidate region that may either be accepting or rejecting. A simple strategy is to split regions that we found to be inconsistent via sampling.

Example 45 Consider the parameter space with six samples depicted in Fig. 17a. After verifying only six instantiated models, we conclude that the parameter space is inconsistent.

The use of samples allows to improve the naive refinement scheme as given in Algorithm 5. This improvement is given in Algorithm 6. For each region R, we have a finite set X of samples. For each sample $u \in X$, it is known whether $\mathcal{G}[u] \models_{\clubsuit} \varphi$. The queue Q now contains pairs (R, X).

In each iteration, a pair (R', X') where R' is (as before) a subregion of R is taken from the queue. Then, we distinguish (again) three possibilities. Only when all samples in X' satisfy φ,

Algorithm 6 Sampling-based refinement loop

sampling-refinement(pSG $\mathcal{G}$, rectangular region R, $\clubsuit \in \{a, d\}$, specification φ)
  $i := 0$, $Q := \{(R, \textbf{sample}(R))\}$, $R_a^i := \emptyset$, $R_r^i := \emptyset$
  while $Q \neq \emptyset$ **do**
    $i := i + 1$
    $(R', X') := Q$.pop
    if $\mathcal{G}, X' \models_\clubsuit \varphi$ **and** $\mathcal{G}, R' \models_\clubsuit \varphi$ **then**
      $R_a^i := R_a^{i-1} \cup R'$, $R_r^i := R_r^{i-1}$
    else if $\mathcal{G}, X' \models_\clubsuit \neg\varphi$ **and** $\mathcal{G}, R' \models_\clubsuit \neg\varphi$ **then**
      $R_a^i := R_a^{i-1}$, $R_r^i := R_r^{i-1} \cup R'$
    else
      $R_a^i := R_a^{i-1}$, $R_r^i := R_r^{i-1}$
      $Q := Q \cup \textbf{split}(R', X')$
  return Accepting region R_a^i, Rejecting region R_r^i

it is verified whether R' is accepting. If R' is accepting, we proceed as before: R_a^i is extended by R' while R_r^i remains unchanged. In the symmetric case that all samples in X' refute φ, we proceed in a similar way by verifying whether R' rejects φ. Otherwise, R' is split into a finite set of subregions with corresponding subsets of X', and added to the queue Q. In case the verification engine provides a counterexample, we can add this counterexample as a new sample. We thus ensure that for all $(R', X') \in Q$, $u \in X'$ implies $u \in R'$. The algorithm can be easily extended such that sampling is also done once a region without samples is obtained: rather than inserting $(R', \emptyset)$ into Q, we insert the entry $(R', \textbf{sample}(R'))$.

Example 46 After several more iterations, the refinement loop started in Example 44 has proceeded to the state in Fig. 16b. First, we see that the candidate region from Fig. 16a was not rejecting. The verification engine gave a counterexample in form of an accepting sample (around $p \mapsto 0.45, q \mapsto 0.52$). Further iterations with smaller regions had some successes, but some additional samples were generated as counterexamples. The current blue candidate is to be checked next. In Fig. 16c, we see a further continuation, with even smaller regions being verified. Note the white box on the right border: It has been checked, but the verification timed out without a conclusive answer. Therefore, we do not have a counterexample in this subregion.

It remains to discuss some methods to split a region, and how we may discard some of the constructed regions. We outline more details below.

9.2.1 How to split

Splitting of regions based on the available samples can be done using different strategies. We outline two basic approaches. These approaches can be easily mixed and extended, and their performance heavily depends on the concrete example at hand.

Equal splitting. This approach splits regions in equally-sized regions; the main rationale is that this generates small regions with concise bounds (the bounds are typically powers of two). Splitting in equally sized regions can be done recursively: One projects all samples down to a single dimension, and splits if both accepting and rejecting samples are in the region. The procedure halts if all samples in a region are either accepting or rejecting. The order in which parameters are considered plays a crucial role. Typically, it is a good idea to first split along the larger dimensions.

Example 47 A split in equally-sized regions is depicted in Fig. 17b, where first the left region candidate is created. The remaining region can be split either horizontally or vertically to immediately generate another region candidate. A horizontal split in the remaining region yields a region without any samples.

The downside of equal splitting is that the position of the splits are not adapted based on the samples. Therefore, the number of splits might be significantly larger than necessary, leading to an increased number of verification calls.

Growing rectangles. This approach attempts to gradually obtain a large region candidate[16]. The underlying rationale is to quickly cover vast amounts of the parameter space. This is illustrated in Fig. 17d (notice that we adapted the samples for a consistent but concise description) where from an initial sampling a large rectangle is obtained as region candidate.

Example 48 Consider the shaded regions in Fig. 17c. Starting from vertex $v = (1, 1)$, the outer rectangle is maximised to not contain any accepting samples. Taking this outer rectangle as candidate region is very optimistic, it assumes that the accepting samples are on the border. A more pessimistic variant of growing rectangles is given by the inner shaded region. It takes a rejecting sample as vertex v' such that the v and v' span the largest region.

The growing rectangles algorithm iterates over a subset of the hyper-rectangle's vertices: For each vertex (referred to as *anchor*), among all possible sub-hyper-rectangles containing the anchor and only accepting or only rejecting samples, the largest is constructed.

Example 49 The growing rectangles approach pessimistically takes anchor $(0, 0)$ as anchor and yields the candidate region in Fig. 17d.

The verification fails more often on large regions (either due to time-outs or due to the over-approximation). Consequently, choosing large candidate regions comes at the risk of failed verification calls, and fragmentation of the parameter space in more subregions.

Furthermore, growing rectangles requires a fall-back splitting strategy: To see why, consider Fig. 15. The accepting (green) region does not contain any anchors of the full parameter space, therefore the hypothesis for any created subregion is always rejection. Thus, no subregion containing a (known) accepting sample is ever considered as a region candidate.

9.2.2 Neighbourhood analysis

Besides considering samples within a region, we would like to illustrate that analysis of a region R can and should take information from outside of R into account. First, take Fig. 17b, and assume that the left region is indeed accepting. The second generated region contains only rejecting samples, but it is only rejecting if all points, including all those on the border to the left region, are rejecting. In other words, the border between the accepting and rejecting regions needs to exactly follow the border between the generated region candidates. The latter case does not occur often, so it is reasonable to shrink or split the second generated region. Secondly, a sensible hypothesis for candidate regions without samples inside is helpful, especially for small regions or in high dimensions. Instead of spawning new samples, we take samples and decided regions outside of the candidate region into account to create a hypothesis. Concretely, we infer the hypothesis for regions without samples via the closest known region or sample.

[16] The approach shares its rationale with the approach formerly implemented in `PROPhESY` [65], but is realised slightly differently to overcome challenges for n-dimensional hyper-rectangles.

9.3 Requirements on verification back-ends

In this section, we have described techniques for iteratively partitioning the parameter space into accepting and rejecting regions. The algorithms rely on verifying regions (and sets of samples) against the specification φ. The way in which verification is used in the iterative parameter space partitioning scheme imposes the following requirements on the verification back-end:

(i) The verification should work *incrementally*. That is to say, verification results from previous iterations should be re-used in successive iterations. Verifying different regions share the same model (pMC or pMDP). A simple example of working incrementally is to reuse minimisation techniques for the model over several calls. If a subregion is checked, the problem is even incremental in a more narrow sense: any bounds etc. obtained for the super-region are also valid for the subregion.
(ii) If the verification procedure fails, i.e. if the verifier returns `false`, obtaining additional *diagnostic information* in the form of a counterexample is beneficial. A counterexample here is a sample which refutes the verification problem at hand.

This wish list is very similar to the typical requirements that theory solvers in lazy SMT frameworks should fulfil [23]. Therefore, SMT-based verification approaches naturally match the wish-list. Parameter-lifting can work incrementally: it reuses the graph-structure to avoid rebuilding the MDP, and it may use previous model checking results to improve the time until the model checker converges. Parameter-lifting, due to its approximative nature, does provide only limited diagnostic information: In particular, it provides information which parameters would be assigned with the upper or lower bounds based on the strategy that optimizes the MDP/SG.

10 Implementation

All the algorithms and constructions in this paper have been implemented, and are publicly available via `PROPhESY`[17]. In particular, `PROPhESY` supports algorithms for:

- **the exact synthesis problem:** via *computing the solution function*, using either of the three variants of state elimination, discussed in Sect. 5.
- **the verification problem:** via an *encoding to an SMT-solver* as in Sect. 6 or by employing the *parameter lifting* method as in Sects. 7 and 8.
- **the approximate synthesis problem:** via *parameter space partitioning*, that iteratively generates verification calls as described in Sect. 9.

`PROPhESY` is implemented in `python`, and designed as a flexible toolbox for developing and experimenting with parameter synthesis. `PROPhESY` internally heavily relies on high-performance routines of the probabilistic model checker `Storm` [66] and the SMT Z3. `PROPhESY` is built in a modular way, such that it is easy to use different backend solvers. The computation of the solution function and the parameter lifting presented in the experiments have been implemented in `Storm`.

`PROPhESY` can be divided in three parts:

(i) First and foremost, it presents a library consisting of: (a) data structures for parameter spaces and instantiations, solution functions, specifications, etc., built around the

[17] https://github.com/moves-rwth/prophesy, archived at https://doi.org/10.5281/zenodo.7697154.

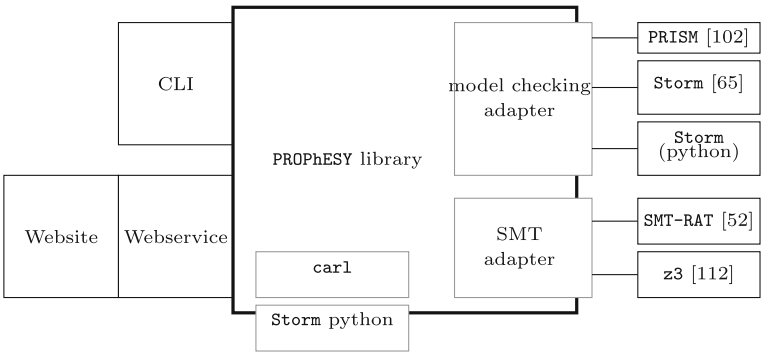


Fig. 18 High-level architecture of PROPhESY and its backends

python bindings of the library carl[18] (featuring computations with polynomials and rational functions), (b) algorithms such as guided sampling, various candidate region generation procedures, decomposition of regions, etc., methods that require tight integration with the model are realised via the python bindings of Storm[19], (c) abstract interfaces to backend tools, in particular probabilistic model checkers, and SMT-checkers, together with some concrete adapters for the different solvers, see Fig. 18.

(ii) An extensive command-line interface which provides simple access to the different core functionalities of the library, ranging from sampling to full parameter synthesis.

(iii) A prototypical web-service running on top of the library, which allows users to interact with the parameter synthesis via a web-interface.

PROPhESY is constructed in a modular fashion: besides the python bindings for carl, all non-standard packages and tools (in particular model checkers and SMT solvers) are optional. Naturally, the full power of PROPhESY can only be used if these packages are available. Besides the methods presented in this paper, PROPhESY contains two further mature parameter synthesis methods: (i) particle-swarm optimisation inspired by [43], and (ii) convex optimisation from [57].

The information in the remainder details the implementation and the possibilities provided by PROPhESY. The section contains some notions from probabilistic model checking [13, 16, 98]. We refrain from providing detailed descriptions of these notions, as it would go beyond the scope of this paper.

10.1 Model construction and preprocessing *(Realised in Storm)*

The model checker Storm supports the creation of pMCs and pMDPs from both PRISM-language model descriptions [102] and JANI-specifications [32]. The latter can be used as intermediate format to support, e.g., digital-clock PTAs with parameters written in Modest [80], or to support expected time properties of generalised stochastic Petri nets [109] with parametric rates and/or weights. Parametric models can be built using the matrix-based, explicit representation, as well as the symbolic, decision diagram (dd)-based engine built on top of sylvan [135]. Both engines support the computation of qualitative properties, an essential preprocessing step, and bisimulation minimisation on parametric models, as described in [78]. We advocate the use of the Storm-python API adapter: Its interactive

[18] https://moves-rwth.github.io/pycarl/.

[19] https://moves-rwth.github.io/stormpy/.

nature avoids the repetition of expensive steps. In particular, it allows for the incremental usage of parameter lifting and sampling.

The support for rational functions is realised via the library `carl`[20]. The rational function is stored as a tuple consisting of multivariate polynomials. These polynomials are by default stored in a partially factorised fashion, cf. [91]. Each factor (a polynomial) is stored as an ordered sparse sum of terms, each term consists of the coefficient and a sparse representation of variables with their non-zero exponents. For manipulating the (rational) coefficients, we exploit `gmp`[21] or `cln`[22]. The former is thread-safe, while the latter performs slightly better with single-thread usage. Computation of GCDs in multivariate polynomials is done either via `ginac` [22] or `cocoa` [2].

10.2 Solution function computation *(Realised in `Storm`)*

The computation of solution functions for pMCs as discussed in Sect. 5 is implemented for a variety of specifications:

- reachability and reach-avoid probabilities,
- expected rewards, including expected time of continuous-time Markov chains,
- step-bounded reachability probabilities, and
- long-run average probabilities and rewards.

The computation is realised either via state elimination, or via Gaussian elimination. An implementation of set-based transition elimination is available for symbolic representations of the pMC.

10.2.1 State elimination

As the standard sparse matrix representation used by `Storm` is not suitable for fast removal and insertion of entries, a flexible sparse matrix with faster delete and insert operations is used.

The order in which states are eliminated has a severe impact on the performance [65]. `Storm` supports a variety of static (pre-computed) and dynamic orderings for the elimination:

- several static orders (forward (reversed), backward (reversed)) based on the order of state-generation by the model construction algorithms. This latter order is typically determined by a depth-first search through the high-level model description[23],
- orders based on the topology of the pMC, e.g., based on the decomposition in strongly connected components,
- orders (Regex) which take into account the in-degree (the number of incoming transitions at a state), inspired by [84, 125],
- orders (SPen, DPen) which take into account the complexity of the rational function corresponding to the transition probability. The complexity is defined by the degree and number of terms of the occurring polynomials.

The orders are computed as penalties for states, and the order prefers states with a low penalty. For dynamic orderings (Regex, DPen), the penalties are recomputed as the in-degree of states and complexity of transition probabilities change during state elimination.

[20] https://github.com/moves-rwth/carl-storm.

[21] https://gmplib.org/.

[22] https://www.ginac.de/CLN/.

[23] This order is destroyed during the computation of a bisimulation quotient.

10.2.2 Gaussian elimination

`Storm` supports Eigen [76] as a linear equation system solver over the field of rational functions. It uses the "supernodal" (supernodes) LU factorisation. The matrix is permuted by the column approximate minimum degree permutation (COLAMD) algorithm to reorder the matrix. One advantage is that this solver is based on sparse model-checking algorithm for parameter-free models. The solver therefore, in addition to the properties supported by state elimination, supports the construction in [15] for conditional probabilities *and* rewards.

10.2.3 Set-based transition elimination

This elimination method is targeted for symbolic representations of the Markov chain. Set-based transition elimination is implemented via matrix-matrix multiplications. In every multiplication, a copy of the dd-representation of a matrix over variables $(\vec{s}, \vec{t})$ is made. The copy uses renamed dd-variables $(\vec{t}, \vec{t'})$. Then, a multiplication of the original matrix with the copy can be done on the dd level yielding a matrix $(\vec{s}, \vec{t'})$. Renaming $\vec{t'}$ to $\vec{t}$ yields a matrix on the original dd-variables.

10.3 Parameter lifting *(Realised in `Storm`)*

For parameter lifting (Sects. 7 and 8), the major effort beyond calling standard model-checking procedures is the construction of the substituted (lifted) model. As parameter lifting for different regions does not change the topology of the lifted model, it is beneficial to create a template of the lifted model once, and to substitute the values according to the region at hand. The substitution operation can be sped up by exploiting the following observation: Typically, transition probability functions coincide for many transitions. Thus, we evaluate each occurring function once and substitute the outcome directly at all occurrences. Moreover, for a growing number of regions to be checked, any one-time preprocessing of the lifted model eventually pays off. In particular, we apply minimisation techniques before construction of the lifted model. We use both bisimulation minimisation as well as state elimination of parameter-free transitions. These minimisations drastically reduce the run-time of checking a single region. We use numerical methods first: for regions that we want to classify as accepting (or rejecting) we resort to the analysis of MDPs using policy iteration with rational numbers. For that, we initialise the policy iteration with a guess based on the earlier numerical results.

10.4 SMT-based region verification *(Realised in `PROPhESY`)*

This complete region checking procedure is realised by constructing SMT queries, as elaborated in Sect. 6. When invoking the SMT solver, we use some features of the SMT-lib standard [18]. First of all, when checking several regions, we use backtrack-points to only partly reset the solver: More precisely, the problem description is given by a conjunction of subformulae, where the conjunction is represented by a stack. We first push the constraints for the problem to the stack, save a backtrack point, and then store the region. Once we have checked a particular region, we backtrack to the *backtrack point*, that is, we remove the constraints for the particular region from the problem description. This way, we reuse simplifications and data structures the solver constructed for the problem description covering the model (and not the region). To support both verifying the property and its negation, the problem description is

slightly extended. We add two Boolean variables (*accepting* and *rejecting*). The following gives an example of the encoding together with checking whether a region R_1 is accepting, and a region R_2 is rejecting, using the notation of Sect. 6.

$$x = f_{\mathcal{D},\varphi} \wedge \big(\mathit{accepting} \implies x \geq \lambda\big) \wedge \big(\mathit{rejecting} \implies x < \lambda\big)$$
(push)
$\mathit{accepting} \wedge \Phi(R_1)$
(pop) (push)
$\mathit{rejecting} \wedge \Phi(R_2)$

10.5 Sampling *(Realised in PROPhESY)*

We accelerate the selection of regions by getting a rough picture through sampling, as discussed in Sect. 9. We support two engines for computing the samples: Either via model checking, or by instantiating the solution function. Sampling on the solution function should always be done exactly, as the evaluation of the typically highly-nonlinear solution functions is (again typically) numerically unstable. In each iteration, based on the current set of samples, a new set of sampling candidates is computed. The choice of the new samples can be modified in several ways. The standard used here is via linear interpolation between accepting and rejecting samples.

10.6 Partitioning *(Realised in PROPhESY)*

For the construction of region candidates, we split the initial regions according to our heuristic (quads or growing rectangles, cf. Sect. 9.2) until none of the regions is inconsistent. We sort the candidate regions based on their size in descending order. Furthermore, we prefer regions where we deem verification to be less costly: Candidate regions that are supposed to be accepting and are further away from samples or regions that are rejecting are preferred over those regions which have rejecting samples or regions in their neighbourhood.

11 Experimental evaluation

In this section, we review the scalability of the presented approaches based on a selection of benchmarks.

11.1 Set-up

11.1.1 Benchmarks

We consider five case studies from the literature. The selection represents various application domains.

NAND multiplexing. With integrated circuits being built at ever smaller scale, they are more prone to defects and/or to exhibit transient failures [85]. One way to overcome these deficiencies is the implementation of redundancy at gate-level. In particular, one aims to construct reliable devices from unreliable components. NAND multiplexing is such a technique, orig-

inally due to von Neumann [140]. Automated analysis of NAND multiplexing via Markov chain model checking was considered first in [115]. They also studied the influence of gate failures in either of the stages of the multiplexing by sampling various values. We use the pMC from [65], that replaced fixed probabilities in the original formulation with parameters. We analyse the effect of changing failure probabilities of the gates on the reliability of the multiplexed NAND.

Herman's self-stabilising protocol. In distributed systems, tokens are used to grant privileges (e.g., access to shared memory) to processes. Randomisation is an essential technique to break the symmetry among several processes [7]. Herman's probabilistic algorithm [88] is a token circulation algorithm for ring structures. In each step, every process possessing a token passes the token along with probability p and keeps the token with probability $1-p$. The algorithm is self-stabilising, i.e., started from any illegal configuration with more than one token the algorithm recovers to a legal configuration with a unique token. The recovery time crucially depends on the probability of passing the token, and an optimal value for p depends on the size of the system [105]. We investigate the expected recovery time by parameter synthesis, inspired by [3].

Mean-time-to-failure of a computer system. In reliability engineering, fault trees are a prominent model to describe how a system may fail based on faults of its various components [24, 123]. Dynamic fault trees (DFTs, [71]) extend these fault trees with a notion of a state, and allow to model spare management and temporal dependencies in the failure behaviour. State-of-the-art approaches for dynamic fault trees translate such fault trees into Markov chains [27, 50, 139]; evaluation of the mean-time-to-failure boils down to the analysis of the underlying Markov chain. Probabilities and rewards originate from the failure rate of the components in the described system. Such failure rates are often not known (precisely), especially during design time. Therefore, they may be represented by parameters. We take the HECS DFT [137] benchmark describing the failure of a computer system with an unknown failure rate for the software interface and the spare processor, as first described in [138]. We analyse how this failure rate affects the expected time until the failure (mean-time-to-failure) of the complete computer system.

Network scheduling. This benchmark [143] concerns the wireless downlink scheduling of traffic to different users, with hard deadlines and prioritised packets. The system is time-slotted: time is divided into periods and each period is divided into an equal number of slots. At the start of each time period, a new packet is generated for each user with a randomly assigned priority. The goal of scheduling is to, in each period, deliver the packets to each user before the period ends. Packets not delivered by the end of a period are dropped. Scheduling is non-trivial, as successful transmissions are not stochastically independent, i.e., channels have a (hidden) internal state. The system is described as a partially observable Markov decision process [124], a prominent formalism in the AI community. We take the Network model from [116], and consider the pMC that describes randomised finite memory controllers that solve this scheduling problem, based on a translation from [96]. Concretely, the parameters represent how the finite memory controller randomises. We evaluate the effect of the randomisation in the scheduling on the expected packet loss.

Bounded retransmission protocol. The bounded retransmission protocol (BRP, [61, 87]) is a variant of the alternating bit protocol. It can be used as part of an OSI data link layer, to implement retransmitting corrupted file chunks between a sender and a receiver. The system contains two channels; from sender to receiver and vice versa. BRP is a famous benchmark in (non-parametric) probabilistic model checking, based on a model in [62]. We consider

the parametric version from [78]. The parameters naturally reflect the channel qualities. The model contains non-determinism as the arrival of files on the link layer cannot be influenced. This non-determinism hampers a manual analysis. The combination of parametric probabilities and non-determinism naturally yields a pMDP. We analyse the maximum probability that a sender eventually does not report a successful transmission.

Remark 12 Other benchmarks and a thorough performance evaluation have been presented before in [65] (for state elimination and parameter space partitioning) and [122] (for parameter lifting).

11.1.2 Benchmark statistics

Table 1 summarises relevant information about the concrete instances that we took from the benchmarks. The *id* is used for reference. The *benchmark* refers to the name of the benchmark-set, while the *instance* describes the particular instance from this benchmark set. We give the total number of *parameters* $|V|$ both in the transition matrix as well as in the reward structure whenever applicable. For the remainder of the columns, we give two numbers per benchmark instance: The upper row describes the original model, the latter describes the (strong) bisimulation quotient. The columns give the number of *states* and *transitions*. The last row gives the *time* (in seconds) required for constructing the model (top) and constructing the bisimulation quotient (bottom). We remark that all benchmarks have a limited number of parameters: Systems with many parameters are beyond the reach of the methods discussed here, but can be analysed with respect to simpler synthesis questions (such as finding one suitable instantiation). We refer to the related work for a discussion of such methods.

11.1.3 Evaluation

We conducted the empirical evaluation on an HP BL685C G7 with Debian 9.6. Each evaluation run could use 8 cores with 2.1GHz each. However, unless specified otherwise, algorithms use a single core. We set the timeout to 1 hour and the memory limit to 16GB. We used `PROPhESY` version 2.0, together with the `Storm`-python bindings version 1.3.1, z3 version 4.8.4. All benchmark files are made available via `PROPhESY`[24].

11.2 Exact synthesis via the solution function

To evaluate the exact synthesis approach, we use state elimination with 7 different heuristics, set-based transition elimination, and Gaussian elimination. All configurations are evaluated with and without strong bisimulation.

First, we show the sizes of the solution function: The results are summarised in Table 2. The *id* references the corresponding benchmark instance in Table 1. The BRP pMDP is not included. The set of all strategies prevents the computation of the solution function for all induced pMCs. The next four columns display properties of the resulting rational function. We give the degree of both the numerator (*degree num*) and denominator (*degree denom*), as well as the number of terms in both polynomials (*# terms num*, *# terms denom*). The next column gives the number of configurations (out of the 18) which *successfully* finished within the time limit. The last two columns indicate timings. We give the times (in seconds)

[24] Benchmarks are in the subfolder `benchmark_files`.

Table 1 Detailed information for models in the benchmark set

Id	Benchmark	Instance	$\|V\|$	States	Transitions	Time
1	BRP	MAX = 2, N = 16	2	1439	1908	0.06
				664	928	0.22
2		MAX = 2, N = 256		20639	27348	0.57
				10264	14368	370.83
3		MAX = 2, N = 512		41119	54484	1.11
				20504	28704	197.69
4		MAX = 5, N = 16		2801	3783	0.10
				1354	1912	1.23
5		MAX = 5, N = 256		40721	55143	1.15
				21034	29752	3305.07
6		MAX = 5, N = 512		81169	109927	2.25
				42026	59448	345.21
7	HECS	m = 1, k = 1, i = 1	2	129	489	0.02
				25	71	0.00
8		m = 1, k = 1, i = 2		145	589	0.02
				49	173	0.00
9	Herman	N = 3	1	9	36	0.02
				3	5	0.00
10		N = 5		33	276	0.03
				5	15	0.00
11		N = 7		129	2316	0.11
				16	137	0.02
12		N = 9		513	20196	0.92
				347	15009	0.12
13	NAND	K = 2, N = 2	2	178	243	0.03
				125	167	0.00
14		K = 2, N = 20		154942	239832	2.81
				102012	154722	0.91
15		K = 2, N = 30		681362	1065797	12.56
				474847	732768	4.65
16		K = 5, N = 10		35112	52647	0.63
				23603	34093	0.21
17		K = 5, N = 20		384772	594792	7.04
				288102	436332	3.17
18		K = 5, N = 30		1697732	2653937	31.45
				1345507	2074758	18.49
19	Network	c = 2, K = 2, T = 2	8	52	133	0.00
				52	133	0.00
20		c = 2, K = 2, T = 3	16	106	269	0.01
				106	269	0.00

Table 1 continued

Id	Benchmark	Instance	$\|V\|$	States	Transitions	Time
21		c = 2, K = 2, T = 4	24	164	411	0.01
				164	411	0.00
22		c = 2, K = 4, T = 2	20	136	365	0.01
				136	365	0.00
23		c = 2, K = 4, T = 3	36	262	691	0.01
				262	691	0.00
24		c = 2, K = 4, T = 4	52	392	1023	0.01
				392	1023	0.00

to compute the solution function (*time mc*) and the *total time* including model building, (optional) bisimulation minimisation and computing the solution function. For these timings we give two numbers per benchmark instance: The upper row describes the median value over all successful configurations and the lower row describes the best result obtained. Thus, *while functions often grow prohibitively large, medium-sized functions can still be computed.* Contrary to model checking for parameter-free models, model building is typically *not* the bottleneck.

Furthermore, we see that the selected heuristic is indeed crucial. Consider instance 11: 11 heuristics successfully compute the solution function (and most of them within a second). However, 7 others yield a timeout. That leads us to compare some heuristics in Fig. 19. The plot depicts the cumulative solving times for selected configurations over all 18 benchmark instances (excluding BRP). Gaussian and set-based refer to these approaches, respectively, all other configurations are variants of state elimination, cf. Sect. 10.2.1, (bisim) denotes that bisimulation minimisation is used. The x-axis represents the number of solved instances and the (logarithmic) y-axis represents the time in seconds. A point (x, y) in the plot represents the x fastest instances which could be solved within a total time of y seconds. For 15 instances, one of the depicted configurations was the fastest overall. *Regex* based configurations were the fastest eight times, *DPen* based ones four times and three times configurations based on *FwRev* were fastest. From these numbers, we conclude that *the selection of the heuristic is essential, and depending on the model to be analysed.* From the graph, we further observe that although using a Gaussian elimination yields good performance, state-elimination approaches can (significantly) outperform the Gaussian elimination on some benchmarks. The *DPen* solves all instances (the only configuration to do so), but *Regex* is overall (slightly) faster. The uninformed *FwRev* with bisimulation works surprisingly well for these benchmarks (but that is mostly coincidence). The set-based elimination is clearly inferior on the benchmarks considered here, but allows to analyse some models with a very regular structure and a gigantic state space, e.g., a parametric Markov chain for the analysis of the bluetooth protocol [70].

11.3 Three types of region verification

We evaluate region verification using two SMT-based approaches (SF: based on first computing the Solution Function, or ETR: encoding the equations into Existential Theory of the Reals), and PLA. In particular, we present some results for the Herman benchmark: it

Table 2 Empirical performance of computing the solution function

Id	Degree num	Degree denom	# terms num	# terms denom	Success	Time mc	Time total
7	23	24	234	247	16	2.00	2.09
						0.64	0.72
8	31	32	408	425	16	9.12	9.21
						3.00	3.08
9	0	2	1	2	18	0.00	0.09
						0.00	0.08
10	4	6	5	6	17	0.04	1.55
						0.00	0.10
11	28	30	29	30	11	0.62	0.82
						0.37	0.56
12	150	152	151	152	8	247.00	248.14
						114.49	115.64
13	10	0	32	1	18	0.00	0.11
						0.00	0.09
14	100	0	2106	1	15	43.05	46.88
						15.46	19.35
15	150	0	4653	1	13	469.29	486.74
						110.54	128.48
16	110	0	1220	1	15	6.30	7.24
						3.30	4.25
17	200	0	4640	1	13	245.47	256.05
						88.18	98.71
18	330	0	10260	1	1	3031.34	3083.88
						3031.34	3083.88
19	1	0	23	1	16	0.00	0.07
						0.00	0.06
20	1	0	111	1	16	0.01	0.08
						0.01	0.07
21	1	0	519	1	16	0.04	0.11
						0.03	0.09
22	1	0	65	1	16	0.01	0.08
						0.01	0.08
23	1	0	289	1	16	0.07	0.15
						0.03	0.10
24	1	0	1377	1	16	0.40	0.48
						0.12	0.20

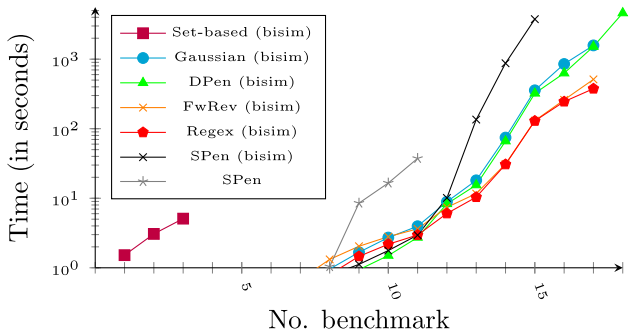


Fig. 19 Cumulative solving times for solution function computation

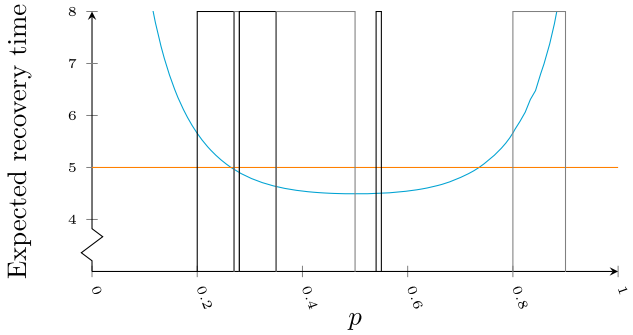


Fig. 20 Plot for Herman model with seven processes and parameter p (Benchmark Id: 11)

features a single parameter, and therefore is well-suited for the illustration of some concepts. We visualised the results for instance 11 in Fig. 20.

The x-axis represents the probability p and the y-axis the expected recovery time. We indicate the solution function in blue. The threshold in the following is set to $\lambda = 5$ and indicated by the orange horizontal line. The black columns depict six different regions[25] that are evaluated with region checking. For each region we want to verify whether the expected recovery time is at least 5. The results are summarised in (the upper part of) Table 3.The first column *id* references the benchmark instance and the second column gives the threshold λ. The next columns indicate the considered *region* and the *technique*. The last columns give the *result* of the region verification and the *time* (in seconds) needed for the computation. The timeout (TO) was set to 120 s.

For benchmark instance 11, Parameter lifting (PLA) computes a result within milliseconds and the computation time is independent of the considered region. The SMT-based techniques take longer and the SF technique in particular does not terminate within two minutes. However, the ETR technique could yield a result for region [0.28, 0.35] whereas PLA could not give a conclusive answer due to its inherent over-approximation.

We now consider the region verification on the NAND model with two parameters. We visualised the solution function for instance 13 in Fig. 21. The considered threshold is $\lambda = 0.3$. Green coloured parts indicate parameter instantiations leading to probabilities above λ and red parts lie below λ. The results of the verification for different regions are given in (the

[25] Strictly speaking, regions are given by the intervals for the parameter, we depict the columns for better visibility.

Table 3 Empirical performance of region verification algorithms

Id	λ	Region	Techn.	Result	Time
11	5	[0.20, 0.27]	ETR	Inconsistent	12,11
			PLA	Unknown	0.01
			SF	Unknown	TO
		[0.27, 0.28]	ETR	Reject	20.68
			PLA	Reject	0.01
			SF	Unknown	TO
		[0.28, 0.35]	ETR	Reject	53.47
			PLA	Unknown	0.01
			SF	Unknown	TO
		[0.35, 0.50]	ETR	Reject	23.41
			PLA	Reject	0.00
			SF	Unknown	TO
		[0.54, 0.55]	ETR	Reject	22.35
			PLA	Reject	0.01
			SF	Unknown	TO
		[0.80, 0.90]	ETR	Unknown	TO
			PLA	Accept	0.01
			SF	Unknown	TO
13	0.3	$[0.01, 0.99] \times [0.70, 0.90]$	ETR	Accept	16.20
			PLA	Unknown	0.01
			SF	Accept	0.16
		$[0.01, 0.99] \times [0.90, 0.99]$	ETR	Inconsistent	19.41
			PLA	Unknown	0.01
			SF	Inconsistent	0.04
		$[0.01, 0.50] \times [0.65, 0.70]$	ETR	Accept	45.61
			PLA	Unknown	0.01
			SF	Accept	0.13
		$[0.01, 0.50] \times [0.75, 0.90]$	ETR	Accept	4.58
			PLA	Accept	0.01
			SF	Accept	0.12
		$[0.01, 0.99] \times [0.40, 0.50]$	ETR	Reject	19.82
			PLA	Reject	0.00
			SF	Reject	0.08

lower part of) Table 3. PLA is again the fastest technique, but for larger regions close to the threshold PLA can often not provide a conclusive answer. Contrary to before, SF is superior to ETR.

The performance of the SMT-based techniques (again) greatly depends on the considered region. It is only natural that *the size of the region, and the difference to the threshold have a significant influence on the performance of region verification*. These observations are general and do *hold on all other benchmarks*. Furthermore, parameter lifting seems broadly applicable, and in the setting evaluated here, clearly faster than SMT-based approaches.

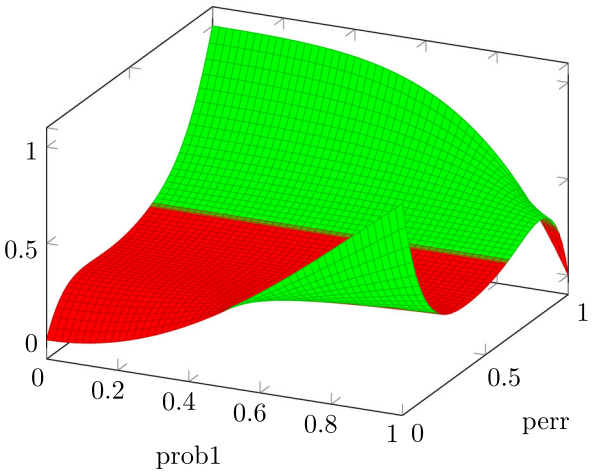


Fig. 21 Plotting the solution function for NAND $K = 2$, $N = 2$ (Benchmark Id: 13) and parameters prob1 and perr

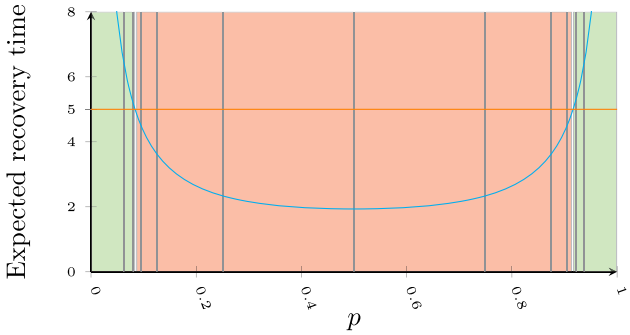


Fig. 22 Parameter space partitioning for Herman $N = 5$ (Benchmark Id: 10) with parameter p

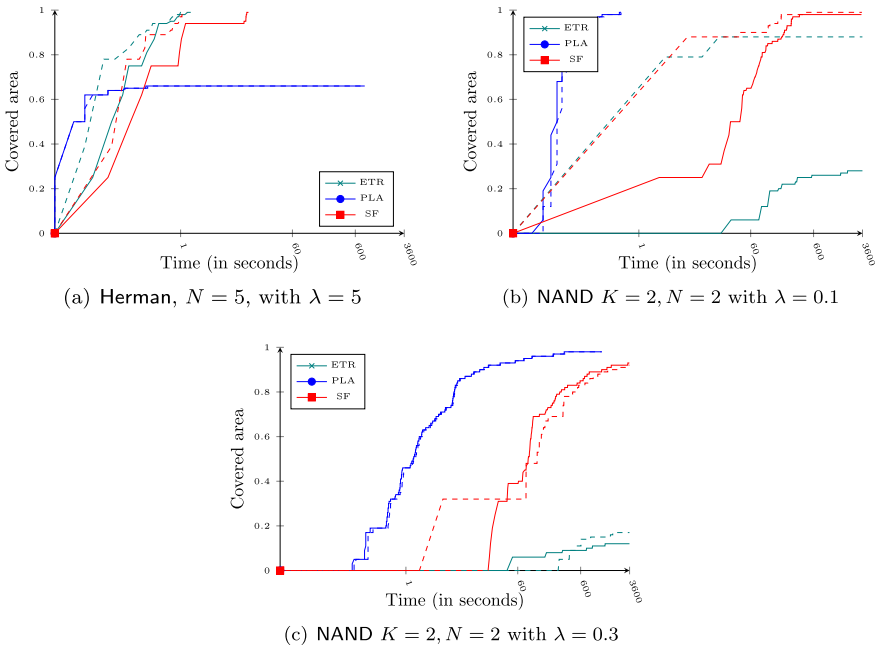
Parameter lifting over-approximates and therefore might only give a decisive result in a refinement loop such as parameter space partitioning. The SMT-based approaches are a valuable fallback. When relying on the SMT techniques, it is heavily model-dependent which performs better. Table 4 at the end of the next section gives some additional results, indicating the performance of the different verification techniques.

11.4 Approximative synthesis via parameter space partitioning

We now evaluate the parameter space partitioning. We use the implementation in `PROPhESY` with the three verification procedures evaluated above. Therefore, we focus here on the actual parameter space partitioning.

First, consider again Herman for illustration purposes. Region verification is not applicable for instance 10 (with threshold 5), as neither all instantiations accept nor all reject the specification. Instead, parameter space partitioning delivers which of these instantiations accept, and which reject the specification. The resulting parameter space partitioning is visualised in Fig. 22.

Next, we compare the three verification techniques—each with two different methods for selecting candidate regions—in Fig. 23. Figure 23a depicts the computation on the Herman



(a) Herman, $N = 5$, with $\lambda = 5$

(b) NAND $K = 2, N = 2$ with $\lambda = 0.1$

(c) NAND $K = 2, N = 2$ with $\lambda = 0.3$

Fig. 23 Covered areas for parameter space partitioning on different models and thresholds

model with 5 processes and threshold $\lambda = 5$. The plot depicts the covered area for all three techniques with both quads (straight lines) and rectangles (dashed lines) as regions. The x-axis represents the computation time (in seconds) on a logarithmic scale and the y-axis represents the percentage of covered area. A point (x, y) in the plot represents y percent of the parameter space which could be covered within x seconds.

For Herman, SMT-based techniques perform better than PLA. PLA was able to cover 64% of the parameter space within milliseconds. However, in the remaining hour only 2% more space was covered. The SMT-based techniques were able to cover at least 99% of the parameter space within 15 s. Moreover, the rectangles cover the parameter space faster than quads. We also perform the parameter space partitioning on the NAND model with two different thresholds: We compare the parameter space partitioning techniques for threshold $\lambda = 0.1$ in Fig. 23b, and for threshold $\lambda = 0.3$ in Fig. 23c. For NAND, the PLA technique performs better than the SMT-based techniques. For threshold $\lambda = 0.1$, PLA could cover at least 99% of the parameter space within 1 s. The main reason is that the border is in a corner of the parameter space. Additionally, the SMT-based techniques with rectangles are significantly faster than the quads for this threshold. For threshold $\lambda = 0.3$, more region verification steps were necessary. PLA still outperforms ETR and SF. However, the use of rectangles over quads does not lead to a better performance for this threshold. At any point in time, there can be very significant differences between the heuristics for candidate generation, especially in settings where single region verification calls become expensive.

Finally, we summarise an overview of the performance in Table 4. For brevity, we pruned some rows, especially if the present approaches already struggle with smaller instances. The *id* is a reference to the benchmark instance. The *technique* is given in the next column. In the next three columns we give for each technique the *time* (in seconds) needed to cover at

Table 4 Empirical performance of parameter space partitioning variations

Id	Techn.	Time 50%	Time 90%	Time 98%	Area cov	Area safe	Percent reg gen	Percent analysis
1	ETR	–	–	–	0.20	0.20	0.00%	99.82%
	PLA	0.04	0.19	3.09	0.99	0.83	31.65%	12.36%
	SF	–	–	–	0.00	0.00	–	–
2	ETR	–	–	–	0.00	0.00	0.00%	89.50%
	PLA	0.33	0.34	–	0.97	0.97	0.00%	81.18%
	SF	–	–	–	0.00	0.00	–	–
4	ETR	–	–	–	0.03	0.03	0.00%	99.61%
	PLA	0.06	0.30	7.86	0.99	0.63	38.45%	10.83%
	SF	–	–	–	0.00	0.00	–	–
5	ETR	–	–	–	0.00	0.00	0.00%	9.73%
	PLA	0.70	–	–	0.87	0.87	0.00%	7.72%
	SF	–	–	–	0.00	0.00	–	–
7	ETR	–	–	–	0.47	0.00	0.00%	99.64%
	PLA	0.00	0.00	0.00	1.00	0.00	0.00%	0.00%
	SF	–	–	–	0.00	0.00	–	–
8	ETR	–	–	–	0.00	0.00	0.00%	99.77%
	PLA	0.01	0.01	0.01	1.00	1.00	0.00%	0.91%
	SF	–	–	–	0.00	0.00	–	–
9	ETR	0.02	30.19	70.41	0.99	0.05	0.00%	98.89%
	PLA	0.08	–	–	0.55	0.06	0.15%	73.89%
	SF	0.02	0.09	0.23	0.99	0.05	0.00%	18.60%
10	ETR	0.12	0.45	1.29	0.99	0.16	0.00%	57.09%
	PLA	0.03	–	–	0.66	0.17	0.15%	74.84%
	SF	0.24	1.20	11.30	0.99	0.16	0.00%	90.63%
12	ETR	–	–	–	0.00	0.00	0.00%	99.66%
	PLA	1.75	–	–	0.56	0.43	0.15%	75.39%
	SF	–	–	–	0.00	0.00	0.00%	96.20%
13	ETR	–	–	–	0.28	0.28	0.00%	99.80%
	PLA	0.05	0.09	0.49	0.99	0.98	8.22%	15.53%
	SF	28.70	202.98	357.90	0.98	0.98	0.00%	96.26%
14	ETR	–	–	–	0.00	0.00	0.00%	85.15%
	PLA	3.08	16.08	152.36	0.99	0.15	32.01%	47.66%
	SF	–	–	–	0.00	0.00	0.00%	98.68%
15	ETR	–	–	–	0.00	0.00	–	–
	PLA	20.27	91.18	854.48	0.99	0.14	30.27%	61.95%
	SF	–	–	–	0.00	0.00	0.00%	92.56%
16	ETR	–	–	–	0.00	0.00	0.00%	98.95%
	PLA	0.55	4.65	55.99	0.99	0.19	33.04%	25.99%
	SF	–	–	–	0.00	0.00	0.00%	99.42%

Table 4 continued

Id	Techn.	Time 50%	Time 90%	Time 98%	Area cov	Area safe	Percent reg gen	Percent analysis
17	ETR	–	–	–	0.00	0.00	0.00%	18.75%
	PLA	8.79	40.99	326.12	0.99	0.16	33.23%	54.62%
	SF	–	–	–	0.00	0.00	0.00%	94.39%
18	ETR	–	–	–	0.00	0.00	–	–
	PLA	53.69	254.13	1861.31	0.99	0.16	33.21%	60.37%
	SF	–	–	–	0.00	0.00	–	–
19	ETR	–	–	–	0.00	0.00	0.00%	99.28%
	PLA	–	–	–	0.12	0.12	0.00%	99.54%
	SF	–	–	–	0.32	0.32	0.00%	98.22%

least 50%, 90% and 98% of the complete parameter space. The next two columns give the complete *covered area*—i.e. the sum of the sizes of all accepting or rejecting regions— when terminating the parameter space partitioning after 1h, together with the *safe area*, i.e. the sum of the sizes of all accepting regions. The last two columns indicate the percentage of the total time spent in generating the regions (*time reg gen*) and verifying the regions (*time analysis*). *PLA is almost always superior, but not on all benchmarks (and not on all (sub)regions.* Depending on the model, SF or ETR are the best SMT-based technique. There might be room for improvement by portfolios and machine-learned algorithm selection schemes.

12 Related work and discussion

We discuss related work with respect to various relevant topics.

Complexity. For graph-preserving pMCs, many complexity results are collected in [97], including results from [45]. In particular, the complement of the verification problem, i.e., the question whether there exists an instantiation in a region that satisfies a reachability property, is ETR-complete for both pMDPs and pMCs[26]. For any fixed number of parameters, the problem can be solved in polynomial time [17]. This paper also considers a richer fragment of the logic PCTL.

Computing a solution function. This approach was pioneered by [64] and significantly improved by [78]. Both `PRISM` [102] and `PARAM` [77] support the computation of a solution function based on the latter method. It has been adapted in [91] to an elimination of SCCs and a more clever representation of rational functions. This representation has been adapted by `Storm` [66]. In [72], computing a solution function via a computer algebra system was considered. That method targets small, randomly generated pMCs with many parameters. Recently, [17] explored the use of one-step fraction-free Gaussian elimination to reduce the number of GCD computations. For pMDPs, [79] experimented with the introduction of discrete parameters to reflect strategy choices—this method, however, scales poorly. In [67] and [68], variants of value iteration with a dd-based representation of the solution function are

[26] It holds that $\mathrm{P} \subset \mathrm{ETR} \subseteq \mathrm{PSPACE}$. A prominent ETR-complete problem is whether a multivariate polynomial has a real-valued root.

presented. Fast sampling on (concise representations of) the solution function is considered in [73, 89].

Equation system formulation. Regarding pMDPs, instead of introducing a Boolean structure, one can lift the linear program formulation for MDPs to a nonlinear program (NLP). This lifting has been explored in [20], and shown to be not feasible in general. A string of results rely on convex programming approaches. For instance, although the general NLP does not lie in the class of convex problems, a variety of verification related problems can be expressed by a sequence of geometric programs, which is exploited in [56]. Alternatively, finding satisfying parameter instantiations in pMDPs under demonic non-determinism and with affine transition probabilities can be approached by iteratively solving a convex-concave program that approximates the original NLP [57]. A comprehensive overview of exploiting convex programming is presented in [60]. Alternatively, more efficient solvers can be used [42] for subclasses of pMDPs. An alternative parametric model with a finite set of parameter instantiations, but without the assumption that these instantiations are graph preserving is considered in [41].

Model repair. The problem of model repair is related to parameter synthesis. In particular, for a Markov model and a refuted specification the problem is to transform the model such that the specification is satisfied. In the special case where repair amounts to changing transition probabilities, the underlying model is parametric as in this paper: the parameters are addive factors to be added to the original transition probabilities. The problem was first defined and solved either by a nonlinear program or parameter synthesis in [20]. A greedy approach was given in [117] and efficient simulation-based methods are presented in [43]. In addition, parametric models are used to rank patches in the repair of software [107].

Interval Markov chains. Instead of parametric transitions, interval MCs or MDPs feature intervals at their transitions [10, 74, 92, 141]. These models do not allow for parameter dependencies, but verification is necessarily "robust" against all probabilities within the intervals, see for instance [120], where convex optimization is utilised, and [81, 82], where efficient verification of multiple-objectives is introduced. In [6, 19], these models are extended to so-called parametric interval MCs, where interval bounds themselves are parametric. Extensions to richer models such as partially observable MDPs are considered in [59, 132].

Derivatives and monotonicity. Many systems behave monotonically in some of their system parameters. For example, most network protocols become more reliable if the communication channel reliability increases. If the solution function is monotonic, then parameter space partitioning can be accelerated [129]. Assessing monotonicity can be tightly integrated in a loop that uses parameter lifting [130]. Finally, the derivative of the solution function can be used for gradient descent whenever the goal is to find a counterexample for region verification [86].

Sensitivity analysis. Besides analysing in which regions the system behaves correctly w. r. t. the specification, it is often desirable to perform a sensitivity analysis [44, 131], i. e., to determine in which regions of the parameter space a small perturbation of the system leads to a relatively large change in the considered measure. In our setting, such an analysis can be conducted with little additional effort. Given a rational function for a measure of interest, its derivations w. r. t. all parameters can be easily computed. Passing the derivations with user-specified thresholds to the SMT solver then allows for finding parameter regions in which the system behaves robustly. Adding the safety constraints described earlier, the SMT solver can find regions that are both safe *and* robust.

Parameters with distributions. Rather than a model in which the parameter values are chosen from a set, they can be equipped with a distribution. The verification outcome consists then of confidence intervals rather than absolute guarantees. In [111], simulation based methods are used, whereas [33, 34] use statistical methods on a solution function. pMDPs with a distribution over the parameters are considered in [9]. Sampling-based methods that rely on the so-called scenario-approach [36, 37] are presented in [11, 58].

Ensuring graph preservation. Checking graph-preservation is closely related to checking whether a well-defined point instantiation exists, which has an exponential runtime in the number of parameters [106]. For parametric interval Markov chains, the question whether there exists a well-defined instantiation is referred to as *consistency* and received attention in [6, 118].

Robust strategies. Robust strategies for pMDPs, as mentioned in Remark 7, are considered in, among others, [108, 141]. These and other variants of synthesis problems on pMDPs were compared in [8]. A variant where parameters are not non-deterministically chosen, but governed by a prior over these parameters, has recently been considered [9]. In [119], data-driven bounds on parameter ranges are obtained, and properties are validated using parameter synthesis techniques.

Continuous time. Parametric CTMCs were first considered by [83]. A method using relaxations similarly to parameter lifting has been proposed in [30]. The method was improved in [39] and implemented in `PRISM-PSY` [40]. A combination with sampling-based algorithms to find good parameter instantiations is explored in [35]. Parameter synthesis with statistical guarantees has been explored in [25, 26]. Moreover, a sampling-based approach for so-called uncertain parametric CTMCs that have a distribution over the parameter values obtains statistical guarantees on reachability probabilities [12]. Finally, in [75], finding good parameter instantiations is considered by identifying subsets of parameters that have a strictly positive or negative influence on the property at hand.

Connection to other models. Furthermore, [96] establishes connections to the computation of strategies in partially observable MDPs [124], a prominent model in AI. In [142], the connection to concurrent stochastic games is shown. pMCs can be used to accelerate solving hierarchical Markov models [95, 113] and for parameter synthesis in Bayesian networks [126]. Finally, in [53], a method that maintains a belief over parameter values is introduced in a robotics context.

13 Conclusion and future work

This paper gives an extensive account of parameter synthesis for discrete-time Markov chain models. In particular, we considered three different variants of parameter synthesis questions. For each problem variant, we give an account of the available algorithms from the literature, together with several extensions from our side. All algorithms are available in the open-source tool `PROPhESY`.

Future work Future work in various directions is possible. Many of the results here can be ported to the more general setting of weighted automata over the adequate semiring [69], which can be interesting from a theoretical perspective. Algorithmically, we would like to develop methods which identify and exploit structural properties that are common to standard benchmarks for Markov chains and Markov decision processes. First steps in this direction have been taken, e.g., by exploiting monotonicity [129]. While graph-preservation is common

in many applications, this restriction is not always natural. The decomposition presented in this paper yields an exponential blow-up in the number of parameters that we would like to avoid whenever possible. However, algorithms that do not rely on graph-preservation have not yet been integrated. The techniques to cover the parameter space by sets of smaller and easy-to-verify regions are still rather naive: This is true both for region verification, where we split due to the approximation, and for parameter space partitioning. The above mentioned monotonicity is one possibility to accelerate the way we split. In general, we plan to exploit parametric models in a data-driven context, where the structure provided by parameter dependencies can be exploited to accelerate learning of probabilistic models [133, 134].

Acknowledgements The authors would like to thank Harold Bruintjes and Florian Corzilius for their contributions to `PROPhESY` 1.0, Tom Janson and Lutz Klinkenberg for their help in developing `PROPhESY` 2.0, Gereon Kremer as a long-term maintainer of `carl` and the anonymous reviewers for their thorough feedback.

Funding Open Access funding enabled and organized by Projekt DEAL.

Open Access This article is licensed under a Creative Commons Attribution 4.0 International License, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and the source, provide a link to the Creative Commons licence, and indicate if changes were made. The images or other third party material in this article are included in the article's Creative Commons licence, unless indicated otherwise in a credit line to the material. If material is not included in the article's Creative Commons licence and your intended use is not permitted by statutory regulation or exceeds the permitted use, you will need to obtain permission directly from the copyright holder. To view a copy of this licence, visit http://creativecommons.org/licenses/by/4.0/.

References

1. (1999) IEEE wireless LAN Medium Access Control (MAC) and Physical Layer (PHY) specification
2. Abbott J, Bigatti AM (2022) CoCoALib: a c++ library for doing computations in commutative algebra. http://cocoa.dima.unige.it/cocoalib
3. Aflaki S, Volk M, Bonakdarpour B, Katoen JP, Storjohann A (2017) Automated fine tuning of probabilistic self-stabilizing algorithms. In: SRDS. IEEE Computer Society, pp 94–103
4. Amparore EG, Beccuti M, Donatelli S (2014) (Stochastic) model checking in GreatSPN. In: Petri Nets, LNCS, vol 8489. Springer, Berlin, pp 354–363
5. Andova S, Hermanns H, Katoen JP (2003) Discrete-time rewards model-checked. In: FORMATS, LNCS, vol 2791. Springer, Berlin, pp 88–104
6. André É, Delahaye B (2016) Consistency in parametric interval probabilistic timed automata. In: TIME. IEEE Computer Society, pp 110–119
7. Angluin D (1980) Local and global properties in networks of processors (extended abstract). In: STOC. ACM, pp 82–93
8. Arming S, Bartocci E, Sokolova A (2017) SEA-PARAM: exploring schedulers in parametric MDPs. In: QAPL@ETAPS, EPTCS, vol 250, pp 25–38
9. Arming S, Bartocci E, Chatterjee K, Katoen JP, Sokolova A (2018) Parameter-independent strategies for pMDPs via POMDPs. In: QEST, LNCS, vol 11024. Springer, Berlin, pp 53–70
10. Bacci G, Delahaye B, Larsen KG, Mariegaard A (2021) Quantitative analysis of interval Markov chains. In: Model checking, synthesis, and learning, LNCS, vol 13030. Springer, Berlin, pp 57–77
11. Badings TS, Cubuktepe M, Jansen N, Junges S, Katoen J, Topcu U (2022) Scenario-based verification of uncertain parametric MDPs. Int J Softw Tools Technol Transf 24(5):803–819
12. Badings TS, Jansen N, Junges S, Stoelinga M, Volk M (2022) Sampling-based verification of CTMCs with uncertain rates. In: CAV (2), LNCS, vol 13372. Springer, Berlin, pp 26–47
13. Baier C, Katoen JP (2008) Principles of model checking. MIT Press, Cambridge
14. Baier C, Clarke EM, Hartonas-Garmhausen V, Kwiatkowska MZ, Ryan M (1997) Symbolic model checking for probabilistic processes. In: ICALP, LNCS, vol 1256. Springer, Berlin, pp 430–440
15. Baier C, Klein J, Klüppelholz S, Märcker S (2014) Computing conditional probabilities in Markovian models efficiently. In: TACAS, LNCS, vol 8413. Springer, Berlin, pp 515–530

16. Baier C, de Alfaro L, Forejt V, Kwiatkowska M (2018) Model checking probabilistic systems. In: Handbook of model checking. Springer, Berlin, pp 963–999
17. Baier C, Hensel C, Hutschenreiter L, Junges S, Katoen J, Klein J (2020) Parametric Markov chains: PCTL complexity and fraction-free Gaussian elimination. Inf Comput 272(104):504
18. Barrett C, Fontaine P, Tinelli C (2016) The satisfiability modulo theories library (SMT-LIB). www.SMT-LIB.org
19. Bart A, Delahaye B, Fournier P, Lime D, Monfroy E, Truchet C (2018) Reachability in parametric interval Markov chains using constraints. Theor Comput Sci 747:48–74
20. Bartocci E, Grosu R, Katsaros P, Ramakrishnan C, Smolka SA (2011) Model repair for probabilistic systems. In: TACAS, LNCS, vol 6605. Springer, Berlin, pp 326–340
21. Basu S, Pollack R, Roy MF (2006) Algorithms in real algebraic geometry (algorithms and computation in mathematics). Springer, New York
22. Bauer C, Frink A, Kreckel R (2002) Introduction to the Ginac framework for symbolic computation within the C++ programming language. J Symb Comput 33(1):1–12
23. Biere A, Heule M, van Maaren H, Walsh T (eds) (2009) Handbook of satisfiability, frontiers in artificial intelligence and applications, vol 185. IOS Press, Amsterdam
24. Bobbio A, Trivedi KS (2017) Reliability and availability engineering: modeling, analysis, and applications. Cambridge University Press, Cambridge
25. Bortolussi L, Silvetti S (2018) Bayesian statistical parameter synthesis for linear temporal properties of stochastic models. In: TACAS (2), LNCS, vol 10806. Springer, Berlin, pp 396–413
26. Bortolussi L, Milios D, Sanguinetti G (2016) Smoothed model checking for uncertain continuous-time Markov chains. Inf Comput 247:235–253
27. Boudali H, Crouzen P, Stoelinga M (2010) A rigorous, compositional, and extensible framework for dynamic fault tree analysis. IEEE Trans Depend Secure Comput 7(2):128–143
28. Bozzano M, Villafiorita A (2010) Design and safety assessment of critical systems. CRC Press, Cambridge
29. Bozzano M, Cimatti A, Katoen JP, Katsaros P, Mokos K, Nguyen VY, Noll T, Postma B, Roveri M (2014) Spacecraft early design validation using formal methods. Reliab Eng Syst Saf 132:20–35
30. Brim L, Ceska M, Drazan S, Safránek D (2013) Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: CAV, LNCS, vol 8044. Springer, Berlin, pp 107–123
31. Bruttomesso R, Cimatti A, Franzén A, Griggio A, Sebastiani R (2008) The MathSAT 4 SMT solver. In: CAV, LNCS, vol 5123. Springer, Berlin, pp 299–303
32. Budde CE, Dehnert C, Hahn EM, Hartmanns A, Junges S, Turrini A (2017) JANI: quantitative model and tool interaction. In: TACAS (2), LNCS, vol 10206, pp 151–168
33. Calinescu R, Ghezzi C, Johnson K, Pezzè M, Rafiq Y, Tamburrelli G (2016) Formal verification with confidence intervals to establish quality of service properties of software systems. IEEE Trans Reliab 65(1):107–125
34. Calinescu R, Johnson K, Paterson C (2016) FACT: a probabilistic model checker for formal verification with confidence intervals. In: TACAS, LNCS, vol 9636. Springer, Berlin, pp 540–546
35. Calinescu R, Ceska M, Gerasimou S, Kwiatkowska M, Paoletti N (2018) Efficient synthesis of robust models for stochastic systems. J Syst Softw 143:140–158
36. Campi MC, Garatti S (2008) The exact feasibility of randomized solutions of uncertain convex programs. SIAM J Optim 19(3):1211–1230
37. Campi MC, Garatti S (2011) A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality. J Optim Theory Appl 148(2):257–280
38. Cerotti D, Donatelli S, Horváth A, Sproston J (2006) CSL model checking for generalized stochastic Petri nets. In: QEST. IEEE Computer Society, pp 199–210
39. Ceska M, Dannenberg F, Kwiatkowska MZ, Paoletti N (2014) Precise parameter synthesis for stochastic biochemical systems. In: CMSB, LNCS, vol 8859. Springer, Berlin, pp 86–98
40. Ceska M, Pilar P, Paoletti N, Brim L, Kwiatkowska MZ (2016) PRISM-PSY: precise GPU-accelerated parameter synthesis for stochastic systems. In: TACAS, LNCS, vol 9636. Springer, Berlin, pp 367–384
41. Ceska M, Jansen N, Junges S, Katoen J (2019) Shepherding hordes of Markov chains. In: TACAS (2), Lecture Notes in Computer Science, vol 11428. Springer, Berlin, pp 172–190
42. Chatzieleftheriou G, Katsaros P (2018) Abstract model repair for probabilistic systems. Inf Comput 259(1):142–160
43. Chen T, Hahn EM, Han T, Kwiatkowska M, Qu H, Zhang L (2013) Model repair for Markov decision processes. In: TASE. IEEE Computer Society, pp 85–92
44. Chen T, Feng Y, Rosenblum DS, Su G (2014) Perturbation analysis in verification of discrete-time Markov chains. In: CONCUR, LNCS, vol 8704. Springer, Berlin, pp 218–233
45. Chonev V (2017) Reachability in augmented interval Markov chains. CoRR arXiv:1701.02996

46. Clarke EM, Grumberg O, Peled D (1999) Model checking. MIT Press, Cambridge
47. Clarke EM, Grumberg O, Jha S, Lu Y, Veith H (2000) Counterexample-guided abstraction refinement. In: CAV, LNCS, vol 1855. Springer, Berlin, pp 154–169
48. Condon A (1990) On algorithms for simple stochastic games. In: Advances in computational complexity theory, DIMACS/AMS, DIMACS series in discrete mathematics and theoretical computer science, vol 13, pp 51–72
49. Cook B (2018) Formal reasoning about the security of Amazon web services. In: CAV, LNCS, vol 10981. Springer, Berlin, pp 38–47
50. Coppit D, Sullivan KJ, Dugan JB (2000) Formal semantics of models for computational engineering: a case study on Dynamic Fault Trees. In: ISSRE. IEEE Computer Society, pp 270–282. https://doi.org/10.1109/ISSRE.2000.885878
51. Cormen TH, Leiserson CE, Rivest RL, Stein C (2009) Introduction to algorithms, 3rd edn. MIT Press, Cambridge
52. Corzilius F, Kremer G, Junges S, Schupp S, Ábrahám E (2015) SMT-RAT: an open source C++ toolbox for strategic and parallel SMT solving. In: SAT, LNCS, vol 9340. Springer, Berlin, pp 360–368
53. Costen C, Rigter M, Lacerda B, Hawes N (2023) Planning with hidden parameter polynomial MDPs. In: AAAI. AAAI Press, Pomona, pp 11,963–11,971
54. Courcoubetis C, Yannakakis M (1988) Verifying temporal properties of finite-state probabilistic programs. In: FOCS. IEEE Computer Society, pp 338–345
55. Cousineau D (2009) Fitting the three-parameter Weibull distribution: review and evaluation of existing and new methods. IEEE Trans Dielectr Electr Insul 16(1):281–288
56. Cubuktepe M, Jansen N, Junges S, Katoen JP, Papusha I, Poonawala HA, Topcu U (2017) Sequential convex programming for the efficient verification of parametric MDPs. In: TACAS (2), LNCS, vol 10206, pp 133–150
57. Cubuktepe M, Jansen N, Junges S, Katoen JP, Topcu U (2018) Synthesis in pMDPs: a tale of 1001 parameters. In: ATVA, LNCS, vol 11138. Springer, Berlin, pp 160–176
58. Cubuktepe M, Jansen N, Junges S, Katoen J, Topcu U (2020) Scenario-based verification of uncertain MDPs. In: TACAS (1), LNCS, vol 12078. Springer, Berlin, pp 287–305
59. Cubuktepe M, Jansen N, Junges S, Marandi A, Suilen M, Topcu U (2021) Robust finite-state controllers for uncertain POMDPs. In: AAAI. AAAI Press, Pomona, pp 11,792–11,800
60. Cubuktepe M, Jansen N, Junges S, Katoen J, Topcu U (2022) Convex optimization for parameter synthesis in MDPs. IEEE Trans Autom Control 67(12):6333–6348
61. D'Argenio PR, Katoen JP, Ruys TC, Tretmans J (1997) The bounded retransmission protocol must be on time! In: TACAS, LNCS, vol 1217. Springer, Berlin, pp 416–431
62. D'Argenio PR, Jeannet B, Jensen HE, Larsen KG (2001) Reachability analysis of probabilistic systems by successive refinements. In: PAPM-PROBMIV, LNCS, vol 2165. Springer, Berlin, pp 39–56
63. de Moura LM, Bjørner N (2008) Z3: An efficient SMT solver. In: TACAS, LNCS, vol 4963. Springer, Berlin, pp 337–340
64. Daws C (2004) Symbolic and parametric model checking of discrete-time Markov chains. In: ICTAC, LNCS, vol 3407. Springer, Berlin, pp 280–294
65. Dehnert C, Junges S, Jansen N, Corzilius F, Volk M, Bruintjes H, Katoen JP, Ábrahám E (2015) Prophesy: a probabilistic parameter synthesis tool. In: CAV, LNCS, vol 9206. Springer, Berlin, pp 214–231
66. Dehnert C, Junges S, Katoen JP, Volk M (2017) A storm is coming: a modern probabilistic model checker. In: CAV, LNCS, vol 10427. Springer, Berlin, pp 592–600
67. Delgado KV, Sanner S, de Barros LN (2011) Efficient solutions to factored MDPs with imprecise transition probabilities. Artif Intell 175(9–10):1498–1527
68. Delgado KV, de Barros LN, Dias DB, Sanner S (2016) Real-time dynamic programming for Markov decision processes with imprecise probabilities. Artif Intell 230:192–223
69. Droste M, Kuich W, Vogler H (2009) Handbook of weighted automata. Springer, Berlin
70. Duflot M, Kwiatkowska MZ, Norman G, Parker D (2006) A formal analysis of bluetooth device discovery. STTT 8(6):621–632
71. Dugan JB, Bavuso SJ, Boyd MA (1992) Dynamic fault-tree models for fault-tolerant computer systems. Trans Reliab 41(3):363–377. https://doi.org/10.1109/24.159800
72. Filieri A, Tamburrelli G, Ghezzi C (2016) Supporting self-adaptation via quantitative verification and sensitivity analysis at run time. IEEE Trans Software Eng 42(1):75–99
73. Gainer P, Hahn EM, Schewe S (2018) Accelerated model checking of parametric Markov chains. In: ATVA, LNCS, vol 11138. Springer, Berlin, pp 300–316
74. Givan R, Leach SM, Dean TL (2000) Bounded-parameter Markov decision processes. Artif Intell 122(1–2):71–109

75. Gouberman A, Siegle M, Tati B (2019) Markov chains with perturbed rates to absorption: theory and application to model repair. Perform Evaluat 130:32–50
76. Guennebaud G, Jacob B et al (2010) Eigen v3. http://eigen.tuxfamily.org
77. Hahn EM, Hermanns H, Wachter B, Zhang L (2010) PARAM: a model checker for parametric Markov models. In: CAV, LNCS, vol 6174. Springer, Berlin, pp 660–664
78. Hahn EM, Hermanns H, Zhang L (2010) Probabilistic reachability for parametric Markov models. STTT 13(1):3–19
79. Hahn EM, Han T, Zhang L (2011) Synthesis for PCTL in parametric Markov decision processes. In: NASA formal methods, LNCS, vol 6617. Springer, Berlin, pp 146–161
80. Hahn EM, Hartmanns A, Hermanns H, Katoen JP (2013) A compositional modelling and analysis framework for stochastic hybrid systems. Formal Methods Syst Des 43(2):191–232
81. Hahn EM, Hashemi V, Hermanns H, Lahijanian M, Turrini A (2017) Multi-objective robust strategy synthesis for interval Markov decision processes. In: QEST, LNCS, vol 10503. Springer, Berlin, pp 207–223
82. Hahn EM, Hashemi V, Hermanns H, Lahijanian M, Turrini A (2019) Interval Markov decision processes with multiple objectives: From robust strategies to pareto curves. ACM Trans Model Comput Simul 29(4):27:1–27:31
83. Han T, Katoen JP, Mereacre A (2008) Approximate parameter synthesis for probabilistic time-bounded reachability. In: RTSS. IEEE Computer Society, pp 173–182
84. Han Y (2013) State elimination heuristics for short regular expressions. Fundam Inform 128(4):445–462
85. Haselman M, Hauck S (2010) The future of integrated circuits: a survey of nanoelectronics. Proc IEEE 98(1):11–38
86. Heck L, Spel J, Junges S, Moerman J, Katoen J (2022) Gradient-descent for randomized controllers under partial observability. In: VMCAI, LNCS, vol. 13182. Springer, Berlin, pp 127–150
87. Helmink L, Sellink MPA, Vaandrager FW (1993) Proof-checking a data link protocol. In: TYPES, LNCS, vol 806. Springer, Berlin, pp 127–165
88. Herman T (1990) Probabilistic self-stabilization. Inf Process Lett 35(2):63–67
89. Holtzen S, Junges S, Vazquez-Chanlatte M, Millstein TD, Seshia SA, den Broeck GV (2021) Model checking finite-horizon Markov chains with probabilistic inference. In: CAV (2), LNCS, vol 12760. Springer, Berlin, pp 577–601
90. Hopcroft JE, Motwani R, Ullman JD (2003) Introduction to automata theory, languages, and computation. Addison-Wesley, Boston
91. Jansen N, Corzilius F, Volk M, Wimmer R, Ábrahám E, Katoen JP, Becker B (2014) Accelerating parametric probabilistic verification. In: QEST, LNCS, vol 8657. Springer, Berlin, 404–420
92. Jonsson B, Larsen KG (1991) Specification and refinement of probabilistic processes. In: LICS. IEEE Computer Society, pp 266–277
93. Jovanovic D, de Moura LM (2013) Cutting to the chase—solving linear integer arithmetic. J Autom Reason 51(1):79–108
94. Junges S (2020) Parameter synthesis in Markov models. Ph.D. thesis, RWTH Aachen University, Germany
95. Junges S, Spaan MTJ (2022) Abstraction-refinement for hierarchical probabilistic models. In: CAV (1), LNCS, vol 13371. Springer, Berlin, pp 102–123
96. Junges S, Jansen N, Wimmer R, Quatmann T, Winterer L, Katoen JP, Becker B (2018) Finite-state controllers of POMDPs using parameter synthesis. In: UAI. AUAI Press, pp 519–529
97. Junges S, Katoen J, Pérez GA, Winkler T (2021) The complexity of reachability in parametric Markov decision processes. J Comput Syst Sci 119:183–210
98. Katoen JP (2016) The probabilistic model checking landscape. In: LICS. ACM
99. Knuth D, Yao A (1976) Algorithms and complexity: new directions and recent results. Academic Press, chap The complexity of nonuniform random number generation
100. Kozine I, Utkin LV (2002) Interval-valued finite Markov chains. Reliable Comput 8(2):97–113
101. Kurshan RP (2018) Transfer of model checking to industrial practice. In: Handbook of model checking. Springer, Berlin, pp 763–793
102. Kwiatkowska M, Norman G, Parker D (2011) Prism 4.0: verification of probabilistic real-time systems. In: CAV, LNCS, vol 6806. Springer, Berlin, pp 585–591
103. Kwiatkowska M, Norman G, Parker D (2012a) The PRISM benchmark suite. In: QEST. IEEE Computer Society, pp 203–204
104. Kwiatkowska MZ, Norman G, Parker D (2008) Using probabilistic model checking in systems biology. SIGMETRICS Perform Eval Rev 35(4):14–21
105. Kwiatkowska MZ, Norman G, Parker D (2012) Probabilistic verification of Herman's self-stabilisation algorithm. Formal Asp Comput 24(4–6):661–670

106. Lanotte R, Maggiolo-Schettini A, Troina A (2007) Parametric probabilistic transition systems for system design and analysis. Formal Asp Comput 19(1):93–109
107. Long F, Rinard M (2016) Automatic patch generation by learning correct code. In: POPL. ACM, pp 298–312
108. Mannor S, Mebel O, Xu H (2012) Lightning does not strike twice: robust MDPs with coupled uncertainty. In: ICML. icml.cc/Omnipress
109. Marsan MA, Balbo G, Conte G, Donatelli S, Franceschinis G (1998) Modelling with generalized stochastic petri nets. SIGMETRICS Perform Evaluat Rev 26(2):2
110. McGlynn MJ, Borbash SA (2001) Birthday protocols for low energy deployment and flexible neighbor discovery in ad hoc wireless networks. In: MobiHoc. ACM, pp 137–145
111. Meedeniya I, Moser I, Aleti A, Grunske L (2014) Evaluating probabilistic models with uncertain model parameters. Softw Syst Model 13(4):1395–1415
112. Mushkin M, Bar-David I (1989) Capacity and coding for the Gilbert–Elliot channels. IEEE Trans Inf Theory 35(6):1277–1290
113. Neary C, Verginis CK, Cubuktepe M, Topcu U (2022) Verifiable and compositional reinforcement learning systems. In: ICAPS. AAAI Press, Pomona, pp 615–623
114. Norman G, Shmatikov V (2006) Analysis of probabilistic contract signing. J Comput Secur 14(6):561–589
115. Norman G, Parker D, Kwiatkowska M, Shukla S (2005) Evaluating the reliability of NAND multiplexing with PRISM. IEEE Trans Comput Aided Des Integr Circuits Syst 24(10):1629–1637
116. Norman G, Parker D, Zou X (2017) Verification and control of partially observable probabilistic systems. Real-Time Syst 53(3):354–402
117. Pathak S, Ábrahám E, Jansen N, Tacchella A, Katoen J (2015) A greedy approach for the efficient repair of stochastic models. In: NFM, LNCS, vol. 9058. Springer, Berlin, pp 295–309
118. Petrucci L, van de Pol J (2018) Parameter synthesis algorithms for parametric interval Markov chains. In: FORTE, LNCS, vol. 10854. Springer, Berlin, pp 121–140
119. Polgreen E, Wijesuriya VB, Haesaert S, Abate A (2016) Data-efficient bayesian verification of parametric Markov chains. In: QEST, LNCS, vol. 9826. Springer, Berlin, pp 35–51
120. Puggelli A, Li W, Sangiovanni-Vincentelli AL, Seshia SA (2013) Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In: CAV, LNCS, vol 8044. Springer, Berlin, pp 527–542
121. Puterman ML (1994) Markov decision processes: discrete stochastic dynamic programming. Wiley, New York
122. Quatmann T, Dehnert C, Jansen N, Junges S, Katoen JP (2016) Parameter synthesis for Markov models: Faster than ever. In: ATVA, LNCS, vol 9938, pp 50–67
123. Ruijters E, Stoelinga M (2015) Fault tree analysis: a survey of the state-of-the-art in modeling, analysis and tools. Comput Sci Rev 15–16:29–62
124. Russell SJ, Norvig P (2010) Artificial intelligence—a modern approach (3. internat. ed.). Pearson Education, London
125. Sakarovitch J (2005) The language, the expression, and the (small) automaton. In: CIAA, LNCS, vol 3845. Springer, Berlin, pp 15–30
126. Salmani B, Katoen J (2021) Fine-tuning the odds in bayesian networks. In: ECSQARU, LNCS, vol 12897. Springer, Berlin, pp 268–283
127. Segala R, Turrini A (2005) Comparative analysis of bisimulation relations on alternating and non-alternating probabilistic models. In: QEST. IEEE Computer Society, pp 44–53
128. Shapley LS (1953) Stochastic games. Proc Natl Acad Sci 39(10):1095–1100
129. Spel J, Junges S, Katoen J (2019) Are parametric Markov chains monotonic? In: ATVA, LNCS, vol 11781. Springer, Berlin, pp 479–496
130. Spel J, Junges S, Katoen J (2021) Finding provably optimal Markov chains. In: TACAS (1), LNCS, vol 12651. Springer, Berlin, pp 173–190
131. Su G, Feng Y, Chen T, Rosenblum DS (2016) Asymptotic perturbation bounds for probabilistic model checking with empirically determined probability parameters. IEEE Trans Software Eng 42(7):623–639
132. Suilen M, Jansen N, Cubuktepe M, Topcu U (2020) Robust policy synthesis for uncertain POMDPs via convex optimization. In: IJCAI, ijcai.org, pp 4113–4120
133. Suilen M, Simão TD, Parker D, Jansen N (2022) Robust anytime learning of Markov decision processes. In: NeurIPS
134. Tappler M, Aichernig BK, Bacci G, Eichlseder M, Larsen KG (2019) L^*-based learning of markov decision processes. In: FM, Lecture Notes in Computer Science, vol 11800. Springer, Berlin, pp 651–669
135. van Dijk T, van de Pol J (2017) Sylvan: multi-core framework for decision diagrams. STTT 19(6):675–696

136. Vardi MY (1985) Automatic verification of probabilistic concurrent finite-state programs. In: FOCS. IEEE Computer Society, pp 327–338
137. Vesely W, Stamatelatos M (2002) Fault tree handbook with aerospace applications. Tech. rep, NASA Headquarters, USA
138. Volk M, Junges S, Katoen JP (2016) Advancing dynamic fault tree analysis—get succinct state spaces fast and synthesise failure rates. In: SAFECOMP, LNCS, vol 9922. Springer, Berlin, pp 253–265
139. Volk M, Junges S, Katoen JP (2018) Fast dynamic fault tree analysis by model checking techniques. IEEE Trans Ind Inform 14(1):370–379
140. von Neumann J (1956) Probabilistic logics and synthesis of reliable organisms from unreliable components. In: Shannon C, McCarthy J (eds) Automata studies. Princeton University Press, Princeton, pp 43–98
141. Wiesemann W, Kuhn D, Rustem B (2013) Robust Markov decision processes. Math Oper Res 38(1):153–183
142. Winkler T, Junges S, Pérez GA, Katoen J (2019) On the complexity of reachability in parametric Markov decision processes. In: CONCUR, Schloss Dagstuhl - Leibniz-Zentrum für Informatik, LIPIcs, vol 140, pp 14:1–14:17
143. Yang L, Murugesan S, Zhang J (2011) Real-time scheduling over Markovian channels: when partial observability meets hard deadlines. In: GLOBECOM. IEEE, pp 1–5

Publisher's Note Springer Nature remains neutral with regard to jurisdictional claims in published maps and institutional affiliations.