PROPhESY:
A PRObabilistic ParamEter SYnthesis Tool[*]

Christian Dehnert, Sebastian Junges, Nils Jansen, Florian Corzilius,
Matthias Volk, Harold Bruintjes, Joost-Pieter Katoen, and Erika Ábrahám

RWTH Aachen University,
Germany

Abstract. We present PROPhESY, a tool for analyzing parametric Markov chains (MCs). It can compute a rational function (i.e., a fraction of two polynomials in the model parameters) for reachability and expected reward objectives. Our tool outperforms state-of-the-art tools and supports the novel feature of conditional probabilities. PROPhESY supports incremental automatic parameter synthesis (using SMT techniques) to determine "safe" and "unsafe" regions of the parameter space. All values in these regions give rise to instantiated MCs satisfying or violating the (conditional) probability or expected reward objective. PROPhESY features a web front-end supporting visualization and user-guided parameter synthesis. Experimental results show that PROPhESY scales to MCs with millions of states and several parameters.

1 Introduction

The mainstream model-checking methods so far focus on safety (is a "bad" state reachable?) and liveness (is some progress made?) properties. For applications in which randomization and uncertainty play an important role, probabilistic properties are of prime importance. These applications include randomized distributed algorithms (where randomization breaks the symmetry between processes), security (e.g., key generation at encryption), systems biology (where species randomly react depending on their concentration), embedded systems (interacting with unknown and varying environments), and so forth. For instance, the *crowds* protocol [1] employs random routing to ensure anonymity. Nodes randomly choose to deliver a packet or to route it to another randomly picked node. In the presence of "bad" nodes that eavesdrop, we could be interested in analyzing probabilistic safety properties such as "the probability of a bad node identifying the sender's identity is less than 5%".

This has led to the development of different automata- and tableau-based *probabilistic model-checking* techniques to prove model properties specified by, e.g., probabilistic ω-regular languages or probabilistic branching-time logics such

[*] This work was supported by the Excellence Initiative of the German federal and state government and the EU FP7 projects SENSATION and CARP.

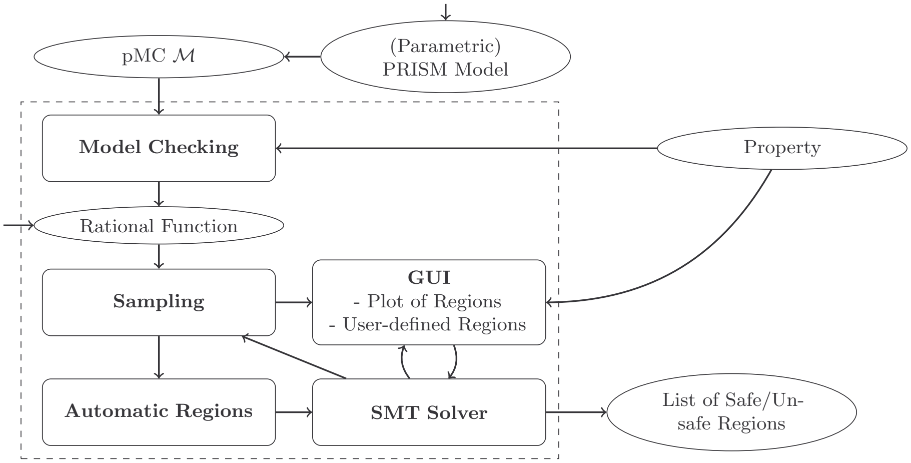


Fig. 1. The verification process of PROPhESY.

as pCTL and pCTL*. Probabilistic model checking is applicable to a plethora of probabilistic models, ranging from discrete-time Markov chains to continuous-time Markov decision processes and probabilistic timed automata, possibly extended with notions of resource consumption (such as memory footprint and energy usage) using rewards (or prices). PRISM [2], MRMC [3], CADP [4] and iscasMc [5] are mature probabilistic model checkers and have been applied successfully to a wide range of benchmarks. Recently, Alur *et al.* [6] identified probabilistic model checking as a promising new direction as it establishes correctness *and* evaluates performance aspects; see also [7].

A major practical obstacle is that probabilistic model-checking techniques and tools work under the assumption that all probabilities in models are a priori known. However, at early development stages, certain system quantities such as faultiness, reliability, reaction rates, packet loss ratios, etc. are often not—or at the best partially—known. In such cases, *parametric* probabilistic models can be used for specification, where transition probabilities are specified as arithmetic expressions using real-valued parameters. In addition to checking instantiated models for fixed parameter values, the important problem of *parameter synthesis* arises, posing the question which parameter values lead to the satisfaction of certain properties of interest. For the crowds protocol it is of interest to establish for which routing probabilities the sender's identity can be revealed in at most 5% of all protocol runs. Similar questions arise in systems biology when determining the concentration of species such that, e.g., catalytic reactions diminish other species within a given time frame with high likelihood. Parametric models are also quite natural in adaptive software where "continuous" verification frequently amends system models during deployment [8] as well as in model repair [9], where probabilities are tuned so as to satisfy a desired property. There is little work done on model checking of parametric probabilistic models, with the notable exception of the PARAM tool [10] and recently also PRISM [2].

This paper presents the tool PROPhESY for the analysis of *parametric (discrete-time) Markov chains* (*pMCs*). Inputs are a pMC (specified in the input language of PRISM) together with a requirement imposing an upper bound on the measure-of-interest, see Figure 1 depicting the workflow of the tool. Transitions in pMCs are labelled with rational functions, i.e., fractions of polynomials over a set of parameters. These measures are (conditional) reachability probabilities or expected costs to reach target states. Once the state space of a pMC is generated, the focus is on determining parameter valuations meeting the requirement, e.g., values for which (a) bad states can be reached in at most 1% of all runs, (b) the expected resource consumption to reach a successful state is within a given budget, or (c) the conditional probability to reach a good state given that eventually a terminating state is reached is above 99%. To do so, PROPhESY supports a palette of advanced techniques relying on computing and efficient manipulation of rational functions and incremental synthesis techniques (á la CEGAR).

In the next section, we will elaborate on PROPhESY's features and summarize the contributions. Section 3 lays the formal background needed for the algorithms and techniques presented in Section 4. In Section 5 we explain details on implementation issues and give a thorough experimental evaluation. Finally, in Sections 6 and 7 we discuss the related work and conclude.

2 Features and Contributions

In this overview on PROPhESY's workflow and contributions we emphasize all steps as depicted in Figure 1.

The core engine. The core *model-checking* engine of the tool determines and returns a *rational function* in terms of the parameters of the (conditional) reachability probability or expected cost. Daws [11] showed that these rational functions can be obtained using state elimination in the pMC, a technique similar to reducing finite-state automata to regular expressions. This was implemented in PARAM [10] and PRISM [2]. Note that finding the minimal-sized regular expression for an automaton is NP-complete; the efficiency of the construction strongly depends on the order in which states are eliminated [12]. We employ several dedicated heuristics in our algorithms. New techniques exploit SCC decomposition for state elimination together with advanced gcd-computations on rational functions [13].

Apart from the these techniques, the PROPhESY tool supports new algorithms dedicated to determine conditional probabilities, which are introduced in this paper. Conditional probabilities are central in—amongst others—the field of Bayesian programming.

Parameter synthesis. In general, to determine whether the given requirement is met, one has to consider all possible parameter valuations. For a feasible and usable approach, we aim for an (approximate) partitioning of the parameter space into *safe* and *unsafe regions.* Each parameter instantiation within a safe region satisfies the requirement under consideration. These *parameter synthesis* problems are challenging and substantially more complex than verifying standard MCs—just checking whether a pMC is realizable (having a parameter evaluation inducing a well-defined MC) is exponential in the number of parameters [14].

Incremental parameter synthesis. Our approach to parameter synthesis can be summarized as follows. After the model checking engine has computed a rational function for the property at hand, the first step is to *sample* the rational function up to a user-adjustable degree. This amounts to instantiating parameter values (determined by dedicated heuristics) over the entire parameter space. This yields a coarse approximation of parts of the solution space that are safe or unsafe and can be viewed as an abstraction of the true partitioning into safe or unsafe parts. The goal is now to divide the parameter space into regions which are *certified* to be safe or unsafe. This is done in an iterative CEGAR-like fashion [15]. First, a region candidate assumed to be safe or unsafe is automatically generated. An *SMT solver* is then used to verify the assumption. In case it was wrong, a *counterexample* in the form of a contradicting sample point is provided along which the abstraction/sampling is *refined*, giving a finer abstraction of the solution space. Using this, new region candidates are generated.

Sensitivity analysis. In addition to determining whether a property is satisfied, the robustness of selected parameters which are subject to perturbation is of utter importance, see [16]. That is, for a region of the parameter space, one wants to certify that changing parameter values within certain "robust" bounds has limited impact on the investigated property. A *sensitivity analysis* for parameters leads to obtaining such bounds. As an initial approach, we benefit from computing the rational function for the measure-of-interest were we simply compute the derivative of this function.

Visualization. The PROPhESY tool includes a *web front-end* as part of a service-oriented architecture for *visualization* as well as steering and guiding the verification process. Concretely, the sampling result and the final or intermediate regions can be visualized in the GUI. The user has the possibility to change the properties dynamically such that the sample points are updated. This offers a direct help to find good parameter evaluations, akin to fitting [17]. Regions in the form of convex polygons can be manually specified and again be verified by an SMT solver. At all times, intermediate results can be used by the automatic CEGAR-like synthesis procedure.

Contributions. The main contribution of this paper is a tool offering a palette of analysis techniques for parametric Markov chains. It significantly extends the efficiency, functionality, and analysis techniques of the currently available tools that can handle parameters, PRISM [2] and PARAM [10]:

– An efficient core engine based on a dedicated library for the costly arithmetic operations yielding a substantial speed up and improved scalability;

- The first algorithmic approach for computing conditional probabilities over parametric MCs. Its instantiation to ordinary (i.e., non-parametric) MCs is orders of magnitudes faster than reported in [18];
- Incremental parameter synthesis (á la CEGAR) exploiting advanced SMT techniques. For many benchmarks, over 90% of the solution space can be split into safe and unsafe regions within a minute.
- Initial support for sensitivity and perturbation analysis;
- A user-friendly GUI based on an integrated web-server for guiding the synthesis process.

3 Formal Foundations

In order for this paper to be self-contained, we briefly introduce the formal models and properties we consider. Let in the following V be a finite set of variables over the domain $\mathbb{R}$. A *valuation* for V is a function $u\colon V \to \mathbb{R}$. Following [19], we use rational functions $f = {}^{g_1}\!/_{g_2}$ over V to describe parameterized probabilities, where g_1 and g_2 are (multivariate) polynomials over V with rational coefficients. Let $\mathbb{Q}_V$ be the set of all rational functions over V. The evaluation $g(u)$ of a polynomial g under u replaces each $x \in V$ by $u(x)$. For $f = {}^{g_1}\!/_{g_2} \in \mathbb{Q}_V$ and evaluation u with $g_2(u) \neq 0$ we define $f(u) = \frac{g_1(u)}{g_2(u)} \in \mathbb{R}$.

Definition 1 (pMC). *A* parametric discrete-time Markov chain (pMC) *is a tuple $\mathcal{M} = (S, V, s_I, P)$ with a finite set of states S, a finite set of parameters $V = \{x_1, \dots, x_n\}$ with domain $\mathbb{R}$, an initial state $s_I \in S$, and a parametric transition probability matrix $P\colon S \times S \to \mathbb{Q}_V$. $\mathcal{M}$ is called a* discrete-time Markov chain (MC) *if $P\colon S \times S \to \mathbb{R}$. Together with a (state) reward function* $\mathrm{rew}\colon S \to \mathbb{R}_{\geq 0}$*, a pMC is called a* parametric Markov reward model.

For a pMC $\mathcal{M} = (S, V, s_I, P)$, the *underlying graph* of $\mathcal{M}$ is $\mathcal{G}_\mathcal{M} = (S, E)$ with $E = \big\{(s, s') \in S \times S \mid P(s, s') \not\equiv 0\big\}$. *Successor or predecessor states of* $s \in S$ are $\mathrm{succ}(s) = \{s' \in S \mid (s, s') \in E\}$ and $\mathrm{pred}(s) = \{s' \in S \mid (s', s) \in E\}$. We define $P(s, S') = \sum_{s' \in S'} P(s, s')$ and $\overline{S'} = S \setminus S'$. State s is *absorbing* iff $\mathrm{succ}(s) = \{s\}$.

A *path* of $\mathcal{M}$ is a non-empty sequence $\pi = s_0 s_1 \dots$ of states $s_i \in S$ such that $P(s_i, s_{i+1}) > 0$ for $i > 0$. A state $s' \in S$ is *reachable* from $s \in S$, written $s \rightsquigarrow s'$, iff there is a path leading from s to s'. The property $\lozenge T$ is overloaded to describe the set of paths finally reaching a set of target states $T \subseteq S$ starting from s_I.

For a pMC $\mathcal{M} = (S, V, s_I, P)$ and a valuation $u\colon V \to \mathbb{R}$ of V, the *instantiated pMC under u* is given by the tuple $\mathcal{M}_u = (S, s_I, P_u)$ with $P_u(s, s') = P(s, s')(u)$ for all $s, s' \in S$. A valuation u is *well-defined* for the pMC $\mathcal{M}$ iff $P_u(s, s') \in [0, 1]$ with $\sum_{s'' \in S} P_u(s, s'') = 1$ for all $s, s' \in S$ and $\mathcal{G}_\mathcal{M} = \mathcal{G}_{\mathcal{M}_u}$. $\mathcal{M}$ is called *realizable* iff there is a well-defined valuation for $\mathcal{M}$. We assume all pMCs to be realizable. The instantiated pMC $\mathcal{M}_u$ induced by a well-defined valuation u is an MC, enabling to use all definitions and concepts for mere MCs also for pMCs.

Example 1. Consider the pMC $\mathcal{M}$ with parameters $V = \{p, q\}$ depicted in Figure 2(a) on Page 8. The valuation $u(p) = u(q) = 0.25$ is well-defined, while $u(p) = u(q) = 0.5$ would induce probabilities larger than 1.

A unique probability measure $\mathrm{Pr}^{\mathcal{M}}$ on sets of paths is defined via the usual *cylinder set construction*, see [20]. For instance, $\mathrm{Pr}^{\mathcal{M}}(\lozenge T)$ describes the probability of reaching $T \subseteq S$ states from s_I in $\mathcal{M}$. For a set of *stochastically independent* paths, the individual probabilities of these paths can be summed.

The *conditional probability* for two reachability objectives is given by

$$\mathrm{Pr}^{\mathcal{M}}(\lozenge T \mid \lozenge C) = \frac{\mathrm{Pr}^{\mathcal{M}}(\lozenge T \cap \lozenge C)}{\mathrm{Pr}^{\mathcal{M}}(\lozenge C)}$$

for $\mathrm{Pr}^{\mathcal{M}}(\lozenge C) > 0$. Considering a (parametric) Markov reward model, the *reward* $\mathrm{rew}(s)$ is earned upon leaving s. The *expected reward* $\mathrm{ExpRew}^{\mathcal{M}}(\lozenge T)$ is the expected amount of reward that has been accumulated until a set of target states $T \subseteq S$ is reached when starting in the initial state s_I. We often omit the superscript $\mathcal{M}$ if it is clear from the context. For more details on probability measures and the considered properties we refer to [20].

Finally, we give a formal definition the model checking problems for pMCs.

Definition 2 (Parametric probabilistic model checking). *For a pMC $\mathcal{M} = (S, V, s_I, P)$ the* parametric probabilistic model checking problem *is to find either*

- *$f^r \in \mathbb{Q}_V$ for $\mathrm{Pr}^{\mathcal{M}}(\lozenge T)$ with $T \subseteq S$,*
- *$f^c \in \mathbb{Q}_V$ for $\mathrm{Pr}^{\mathcal{M}}(\lozenge T \mid \lozenge C)$ with $T, C \subseteq S$,*
- *or $f^e \in \mathbb{Q}_V$ for $\mathrm{ExpRew}^{\mathcal{M}}(\lozenge T)$ with $T \subseteq S$*

such that for all well-defined valuations u, the instantiated rational function f^r, f^c, or f^e, and the instantiated pMC $\mathcal{M}_u$ it holds that:

$$f^r_u = \mathrm{Pr}^{\mathcal{M}_u}(\lozenge T), \qquad f^c_u = \mathrm{Pr}^{\mathcal{M}_u}(\lozenge T \mid \lozenge C), \qquad f^e_u = \mathrm{ExpRew}^{\mathcal{M}_u}(\lozenge T).$$

4 Supported Techniques

In this section we briefly recall incorporated methods introduced in former works and explain new methods and concepts in detail.

4.1 Model Checking

We start by briefly explaining how model checking for a pMC $\mathcal{M} = (S, V, s_I, P)$ and the different properties as in Definition 2 is performed.

Reachability probabilities and expected rewards. 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 $s_I \notin T$. Let us briefly recall the concept of the *state elimination* [11, 19] used to compute a rational function describing reachability probabilities (**eliminate_state** in Algorithm 1). The basic idea is to "bypass" a state s by removing it from the model and increasing the probabilities $P(s_1, s_2)$ of the transitions from each predecessors s_1 to each successors s_2 by the probability of moving from s_1 to s_2 via s, possibly including a self-loop on s. Note that it is well possible to eliminate a single transition (s_1, s_2) by only calling the function **eliminate_transition**(P, s_1, s_2).

The state elimination approach can also be adapted to compute *expected rewards* [19] for Markov reward models. 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 "summarize" the reward that would have been gained from s_1 to s_2 via s.

Algorithm 1 State elimination for pMCs

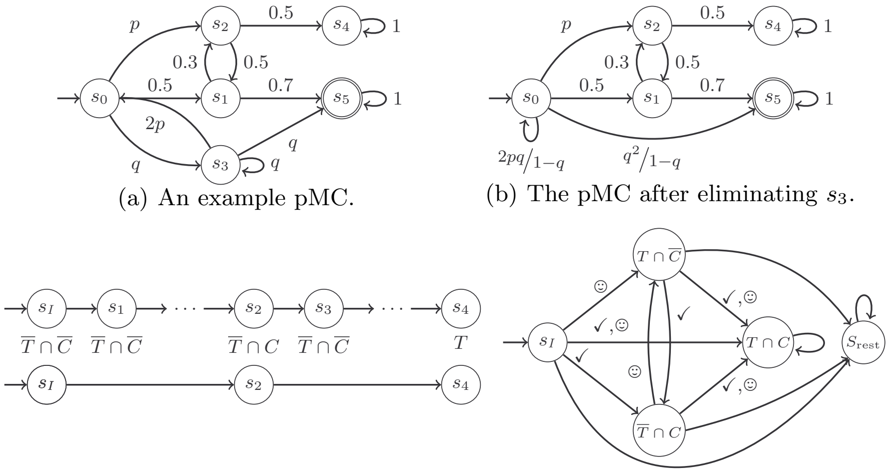
```
eliminate_state($P, s \in S$ not absorbing)
    for each $s_1 \in pred(s), s_1 \neq s$ do
        eliminate_transition($P, s_1, s$)

eliminate_transition($P, s_1 \in pred(s), s \in S$ not absorbing)
    if $s_1 \neq s$ then
        for each $s_2 \in succ(s), s \neq s_2$ do
            $P(s_1, s_2) := P(s_1, s_2) + \frac{P(s_1,s) \cdot P(s,s_2)}{1-P(s,s)}$
        $P(s_1, s) := 0$
    else
        for each $s_2 \in succ(s), s \neq s_2$ do
            $P(s, s_2) := \frac{P(s,s_2)}{1-P(s,s)}$
        $P(s, s) := 0$
```

Example 2. Consider again the pMC from Example 1. Assume, state s_3 is to be eliminated. The states that are relevant for this procedure are the only predecessor s_0 and the successors s_0 and s_5. Applying the function **eliminate_state**(P, s_3) of Algorithm 1 yields the model in Figure 2(b).

Conditional probabilities. The probability $\Pr^{\mathcal{M}}(\lozenge T \mid \lozenge C)$ measures the reachability of $T \subseteq S$ given that C is reached. We assume $s_I \notin T \cup C$, because otherwise the result is the constant one function or coincides with the probability of reaching T, respectively. We assume w. l. o. g. all states in $T \cap C$ to be absorbing.

We will now show how to compute this function using the elimination framework. Let $S_{\text{rest}} = (\overline{T} \cap \overline{C}) \setminus \{s_I\}$. Consider the path fragment in $\mathcal{M}$ as illustrated the upper part of Figure 2(c). It finally reaches T (s_4) after visiting C (s_2) and intermediately visits only S_{rest}. By eliminating the transitions to and from the *intermediate states* in S_{rest}, we essentially summarize the probability of moving from s_I to $\overline{T} \cap C$ and from there on to T; see the second path fragment in Figure 2(c). Therefore, the conditional probability from s_I remains the same.

We use this key insight to convert the shape of $\mathcal{M}$ to the one depicted in Figure 2(d). First, we bypass all non-absorbing intermediate states via state elimination and keep only the states that are relevant for the conditional probability



(a) An example pMC.

(b) The pMC after eliminating s_3.

(c) Eliminating intermediate states for conditional probabilities.

(d) Resulting shape of the pMC.

Fig. 2. pMC model checking.

and the absorbing states in S_{rest}. Then, we eliminate *backward transitions* from all states s targeting the initial state s_I by applying **eliminate_transition** from Algorithm 1. It remains to eliminate transitions between states in $T \cap \overline{C}$ as well as transitions between states in $\overline{T} \cap C$. After this final step, the shape of the resulting system is the one depicted in Figure 2(d). Note that the abstract states and transitions in this pMC correspond to sets of states and sets of transitions, respectively. We are interested in computing the fraction

$$\frac{\Pr^{\mathcal{M}}(\Diamond T \cap \Diamond C)}{\Pr^{\mathcal{M}}(\Diamond C)} =: \frac{f_1}{f_2} \qquad f_1, f_2 \in \mathbb{Q}_V$$

For the sake of clarity, we label an (abstract) transition $s \to s'$ with ☺ if $s' \in T$, and with ✓ if $s' \in C$. To this end, we notice that it suffices to consider path fragments of length two, since we have either seen both ☺ and ✓ along such a fragment (and it therefore contributes to f_1 and f_2), or we are in S_{rest}. In the latter case, we saw only ✓ (contributing to f_2), only ☺, or none of them and there is no way of reaching any one of them in the future.

The functions are computed as follows. f_2 corresponds to the probability mass of all paths along which ✓ is seen. That is, we either (i) start with a ☺ and then see a ✓, (ii) directly see both a ✓and a ☺, or (iii) encounter only a ✓along the first step. f_1 corresponds to the probability mass of all paths along which both ✓ and ☺ are seen. Such paths either (i) start with ✓, and require a subsequent ☺ (corresponding to the path from 2(c)), (ii) start with both ☺ and ✓, or (iii) start with ☺, and require a subsequent ✓. This directly leads to the following equation, where the three cases for f_1 and f_2 correspond to the three summands in the numerator and denominator in the order from left to right.

$$\frac{\Pr^{\mathcal{M}}(\lozenge T \cap \lozenge C)}{\Pr^{\mathcal{M}}(\lozenge C)} = \frac{\sum\limits_{t \in T \cap \overline{C}} P(s_I, t) \cdot P(t, C) + \sum\limits_{t \in T \cap C} P(s_I, t) + \sum\limits_{t \in \overline{T} \cap C} P(s_I, t) \cdot P(t, T)}{\sum\limits_{t \in T \cap \overline{C}} P(s_I, t) \cdot P(t, C) + \sum\limits_{t \in T \cap C} P(s_I, t) + \sum\limits_{t \in \overline{T} \cap C} P(s_I, t)}$$

The pseudo code of the elimination algorithm is given in Algorithm 2.

Algorithm 2 Computing conditional probabilities for pMCs

```
**conditional**(pMC $\mathcal{M} = (S, V, s_I, P)$, $T \subseteq S$, $C \subseteq S$)
    **while** $\exists s \in (\overline{T} \cap \overline{C}) \setminus \{s_I\}$, $s$ not absorbing **do**
        **eliminate_state**$(P, s)$
    **for each** $s_1$ with $P(s_1, s_I) > 0$ **do**
        **eliminate_transition**$(P, s_1, s_I)$
    **while** $\exists s_1, s_2 \in (T \cap \overline{C})$ or $\exists s_1, s_2 \in (\overline{T} \cap C)$ with $P(s_1, s_2) > 0$ **do**
        **eliminate_transition**$(P, s_1, s_2)$
    $g_1 \coloneqq \sum_{t \in T \cap \overline{C}} P(s_I, t) \cdot P(t, C)$        $g_2 \coloneqq \sum_{t \in T \cap C} P(s_I, t)$
    $g_3 \coloneqq \sum_{t \in \overline{T} \cap C} P(s_I, t) \cdot P(t, T)$        $g_4 \coloneqq \sum\limits_{t \in \overline{T} \cap C} P(s_I, t)$
    **return** $\frac{g_1 + g_2 + g_3}{g_1 + g_2 + g_4}$
```

Theorem 1 (Correctness). *For a given pMC $\mathcal{M} = (S, V, s_I, P)$ and sets of states $T \subseteq S$ and $C \subseteq S$, the procedure* ***conditional****$(\mathcal{M}, T, C)$ computes the rational function describing the conditional probability* $\Pr^{\mathcal{M}}(\lozenge T \mid \lozenge C)$.

The proof relies on the fact that state elimination preserves reachability probabilities [19]. As we obtain a structure as in Figure 2(d), the summation over all path fragments of length (at most) two that contribute to the conditioned probability yields the same result as in the original system.

4.2 Parameter Synthesis

Instantiating the rational functions yields model checking probabilities for the corresponding instantiated MCs. However, this only gives a very rough impression of the behavior of the pMC for different parameter values, which is unsatisfactory if one aims to certify expected behavior over a non-singular parameter space. Instead we determine which parts of the parameter space give rise to a safe system. As explained in Section 2, we do this in a CEGAR-like manner; consider again Figure 1. The underlying concepts are presented in the following.

We assume upper bounds[1] $\lambda \in [0, 1]$ for (conditional) reachability probabilities and $\kappa \in \mathbb{R}_{\geq 0}$ for expected rewards. For all parameter valuations inside a region

[1] Note that all methods are equally well applicable to lower bounds.

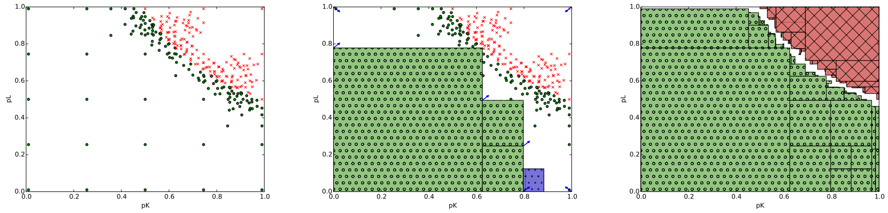


Fig. 3. Sampling and region analysis.

the bound shall either be violated or met in the corresponding instantiated MC. Typically, the parameter space consists of both safe and unsafe regions.

Formally, let a *half-space* for parameters $V = \{p_1, \dots, p_n\}$ be given by the linear inequality $a_1p_1 + \ldots + a_np_n \leq b$ with $a_1, \dots, a_n, b \in \mathbb{Q}$. A *region* is a *convex polytope* defined by m half-spaces, i. e., a system of linear inequalities $A\mathbf{p} \leq b$ with $A \in \mathbb{Q}^{m\times n}$, $\mathbf{p} = (p_1 \dots p_n)^T \in V^{n\times 1}$ and $b \in \mathbb{Q}^{m\times 1}$. Assume a rational function $f^r \in \mathbb{Q}_V$, $f^c \in \mathbb{Q}_V$, or $f^e \in \mathbb{Q}_V$ according to Definition 2 to be computed for a pMC $\mathcal{M}$ as explained in the previous section.

Definition 3 (Safe/unsafe region). *A region is* safe *iff there is no valuation u such that $A\mathbf{u} \leq b$ with $f^r_u > \lambda$, $f^c_u > \lambda$, or $f^e_u > \kappa$ with $\lambda \in [0,1]$ and $\kappa \in \mathbb{R}_{\geq 0}$ where $\mathbf{u} = (u(p_1) \dots u(p_n))^T$. A region is* unsafe *iff there is no valuation such that $f^r_u \leq \lambda$, $f^c_u \leq \lambda$, or $f^e_u \leq \kappa$. Otherwise, the region is called* undetermined.

By safe, unsafe, or undetermined we also refer to the *type* of a region. Given a region and a rational function together with a threshold, certifying the assumed type boils down to checking satisfiability of a conjunction of

- linear inequalities encoding the candidate region,
- nonlinear inequalities ensuring well-definedness of valuations, and
- a nonlinear inequality stating that the bound is violated or satisfied,

using an SMT solver such as Z3 [21]. The solver can then determine whether there exists a valuation inside the candidate region whose corresponding instantiated MC exceeds the threshold on the probability or the expected reward. If so, we obtain such a valuation from the solver and can conclude that the region is not safe. The obtained valuation serves as a *counterexample* to this region candidate.

Sampling. As a guide for determining candidates for safe or unsafe regions, we apply sampling w.r.t. the property. An initially coarse sampling is iteratively refined by adding points based on the linear interpolation between samples from a safe and an unsafe region. Sampling can either be performed by instantiating a rational function describing these reachability probabilities or by instantiating the pMC and performing (non-parametric) probabilistic model checking, e. g., via PRISM. The latter is faster for a moderate number of sample points because of the costly computation of the rational function. However, the rational function is needed for verifying the safety of a region as described above.

Figure 3(left) shows an example sampling of the *Bounded Retransmission Protocol* (BRP) benchmark [22]. Red crosses indicate that λ is exceeded (i.e. the instantiated pMC is unsafe) while green dots indicate a safe instantiation.

Finding region candidates. For the construction of region candidates based on sample points, three methods are available. It is possible to generate *half-spaces* separating safe from unsafe points, successively enlarge rectangles containing only safe or only unsafe points, a technique that is commonly referred to as *growing rectangles*, or recursively separate the search space in *quadrants* that only contain either safe or unsafe points. In each iteration, the intermediate regions are checked for either safety or unsafety, based on the information from the sampling. The middle and right images in Figure 3 show an example of region generation in the BRP benchmark, based on the initial sampling in the first figure. After 5 iterations, a large part of the solution space is already determined to be (un)safe. After 80 iterations, over 97% of the area was covered by certified safe and unsafe regions, respectively. The remaining white space indicates that not the whole parameter space could yet be categorized into safe or unsafe points, but the approximation can be further refined in subsequent iterations. Currently, only pMCs with at most two parameters are supported, but all existing benchmark models satisfy this criterion. We plan to alleviate this restriction by supporting multi-dimensional convex regions, which is a straightforward extension for the rectangle and quadrant approaches, but challenging for the hyperplane approach.

Sensitivity analysis. Besides analyzing in which regions the system behaves correctly w. r. t. the specification, it is often desirable to perform a sensitivity analysis [16], 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.

5 Implementation and Experiments

The complete tool chain is available online[2]. We implemented the model checking algorithms as described in Section 4 in the framework of a probabilistic model checker that is a redevelopment of `MRMC` [3] in `C++`. As for `PARAM` and `PRISM`, models are specified in a parametric version of `PRISM`'s input language. From the model description, we construct the explicit transition matrix, which can then be reduced w. r. t. both *strong* [23] and *weak bisimulation* [24] in order to speed up the computation. As the state elimination process frequently deletes old transitions and creates new ones, we chose not to represent the transition matrix in the

[2] `http://moves.rwth-aachen.de/prophesy/`

compressed row storage format [25], but rather implemented a hybrid between a sparse and a dense representation that only stores non-zero entries but does not store all rows consecutively in memory. Furthermore, for the representation of rational functions, we employ the newly developed modular arithmetic library CArL [26]. Since the simplification involves the costly computation of the greatest common divisor of polynomials, CArL tries to speed this up by caching and refining a partial factorization of rational functions, optimizing the ideas of [13].

The tool chain—integrating the model checking backend with the sampling algorithms, region generation and the web service—is implemented in Python using the SciPy packages [27] and the Shapely package. Currently supported SMT-solvers are Z3 [21] and SMT-RAT [28]. They are interfaced via the standard SMT-LIB format [29], in principle enabling to use all SMT solvers supporting non-linear real arithmetic. Due to numerical instabilities when sampling the rational function, we use exact arithmetics. As the performance of SciPy proved to be insufficient in this regard, we use CArL as sampling backend.

Experimental Evaluation We evaluated the performance of our model checking backend on well-known benchmark models available on PRISM's [30] and PARAM's [31] website, respectively. We compared the running times of our tool with those of PRISM and PARAM on reachability properties and expected reward properties. We ran the experiments on an HP BL685C G7 machine with 48 cores clocked with 2.0GHz each and possessing 192GB of RAM. However, we restricted the available RAM to 12GB for all experiments. We briefly explain the benchmark models, but refer to our website [32] for further details and a full list of benchmark results.

The first case study is the probabilistic *Bounded Retransmission Protocol* [22] that tries to send a file via an unreliable network. This model has two parameters: the reliability of each lossy channel. The *Crowds Protocol* [1] aims at anonymizing the sender of a message by routing it probabilistically through a larger crowd of communication parties. The parameters govern the probability that a message is once more forwarded in the crowd as well as the probability that a member of the crowd is not trustworthy. The *Zeroconf Protocol* [33] governs how hosts joining a network are being assigned a network address by probabilistically choosing one and then checking for possible collisions. This model is parametric in the probability that a collision happens and the probability that this is successfully detected. *Probabilistic Contract Signing* [34] tries to establish the commitment of two parties to a contract where no one trusts each other. It does so by revealing secrets bit by bit with a certain probability that is the single parameter of this model. Finally, *NAND Multiplexing* [35] describes fault-tolerant hardware using unreliable hardware by having copies of a NAND unit all doing the same job. Parameters are the probabilities of faultiness of the units and of erroneous inputs.

Table 1 shows the runtimes (in seconds) of PRISM, PARAM and PROPhESY on the selected benchmarks for different objectives where we chose the best-performing settings for each tool and benchmark instance. These concrete settings are given on our webpage to enable the reproducibility of our results. Note that to the best of our knowledge, no symbolic representation of pMCs is available.

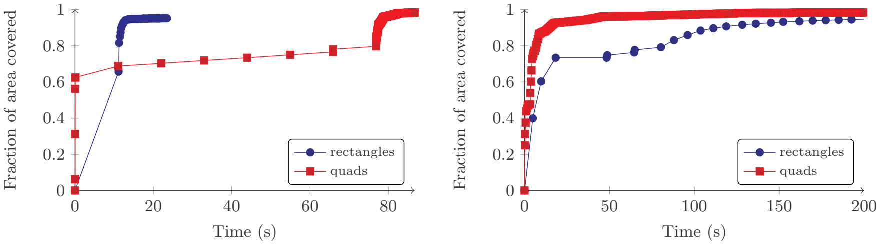
					PRISM		PARAM		PROPhESY	
		instance	#states	#trans	verif.	total	verif.	total	verif.	total
reachability	brp	(128, 5)	10376	13827	215	218	5	7	2	**3**
		(256, 5)	20744	27651	1237	1242	32	33	8	**10**
	crowds	(15, 5)	592060	1754860	TO	TO	18^*	48^*	1	**46**
		(20, 5)	2061951	7374951	TO	TO	75^*	194^*	4	**165**
	nand	(20, 2)	154942	239832	886	901	44	48	16	**22**
		(20, 5)	384772	594792	TO	TO	319	328	89	**104**
exp. reward	egl	(5, 4)	74750	75773	5	11	–	–	< 1	**5**
		(8, 4)	7536638	7602173	543	910	–	–	7	**607**
	nand	(20, 2)	154942	239832	TO	TO	264	2033	5	**12**
		(20, 5)	384772	594792	TO	TO	TO	TO	47	**64**
	zconf	(10000)	10004	20005	TO	TO	TO*	TO*	4	**4**
		(100000)	100004	200005	TO	TO	TO*	TO*	255	**263**
conditional	brp	(256, 2)	10757	13827	–	–	–	–	< 1	1
		(256, 5)	20744	27651	–	–	–	–	1	3
	crowds	(15, 5)	592060	1754860	–	–	–	–	5	50
		(20, 5)	2061951	7374951	–	–	–	–	14	174

					PRISM		Baier *et al.*[18]		PROPhESY	
		instance	#states	#trans	verif.	total	verif.	total	verif.	total
conditional	brp	(256, 2)	10757	13827	6	10	13	16	< 1	$<$ **1**
		(256, 5)	20744	27651	10	14	65	69	< 1	$<$ **1**
		(256, 10)	37389	50691	16	20	325	328	< 1	**1**
	crowds	(10, 5)	111294	261444	95	99	11	16	< 1	**1**
		(15, 5)	592060	1754860	699	702	69	84	< 1	**6**
		(20, 5)	2061951	7374951	TO	TO	184	242	1	**19**

Table 1. Runtimes of model checking on different benchmark models.

Besides the total time taken by the respective tool (columns "total"), we list the verification time (columns "verif."), i. e. the time needed to reduce the model and compute the rational function. The total time also includes the time needed to build the model. Each row of the table corresponds to one benchmark instance. As we observed that PARAM produced wrong results on some case studies when using specific settings, we list the times of the best setup that returned the correct result and marked the entries with a little star. All experiments marked with "TO" exceeded the time limit of one hour and the best total time is **boldfaced**.

When computing the rational function representing the reachability probability, PROPhESY is faster than PARAM, whereas PRISM is significantly slower than both its competitors. E. g. both PARAM and PROPhESY solve the larger crowds instances within less than four minutes, while PRISM is unable to compute a result within the time limit. Note that for the crowds case study, the raw verification times of PROPhESY are always strictly better than those of PARAM, even though the total time is not (always), because PARAM employs an efficient model building procedure that PROPhESY does not implement. Even without this technical advantage, PROPhESY beatsPARAM on almost all instances. For the expected reward benchmarks, we did not list the times for PARAM for the egl case study, because the tool produced an incorrect rational function for the smaller instance for all settings and was unable to build the model for the larger instance. Overall, we observe that PROPhESY outperforms the other tools on all benchmark models.



(a) Crowds Protocol.

(b) Bounded Retransmission Protocol.

Fig. 5. Area of solution space covered.

The runtimes of PROPhESY on the case studies of the first "conditional" section of the table illustrate that our elimination-based algorithm to compute parametric conditional probabilities on pMCs is able to solve instances with millions of states and transitions. For instance, it takes only a few seconds longer to compute the conditional probability rather than the reachability probability on the largest crowds instance despite the more complicated objective.

Thanks to the authors of [18], we could compare the performance of our algorithm for computing conditional probabilities on *non-parametric* models with both (i) the "naive" quotient method available in PRISM and (ii) the prototypical implementation used in [18]. The second section "conditional" of Table 1 shows that we are able to compute the result at least one order of magnitude faster than both PRISM and the prototypical tool of [18] for all benchmarks. Figure 4 shows a scatter plot of all mere verification times except for parametric conditional probability. The data shows how PROPhESY performs in comparison to the best competitor on any given instance. All points above the main diagonal indicate that our tool could solve the instance faster than the competitor, which is the case for all larger benchmarks; above the dashed diagonal, PROPhESY is more than ten times faster.

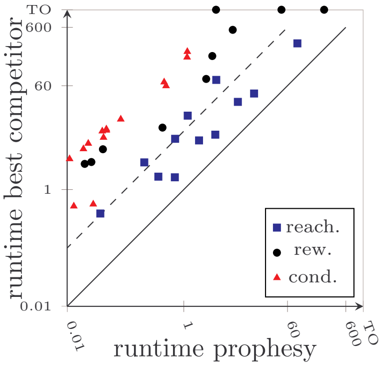


Fig. 4. The summarized results.

Finally, recall Figure 3 showing how growing rectangles cover the parameter space starting from a sampling. For a practical evaluation, see Figure 5 illustrating that large parts of the solution space are covered quickly by the growing rectangles or using quadrants, but covering more area is increasingly costly. Moreover, it depends strongly on the benchmark which of the technique performs best.

6 Related work

Parameter synthesis for probabilistic models is a relatively new and challenging field. Daws [11] proposed to represent reachability probabilities by means of rational functions, which are obtained by state elimination (as for obtaining a regular expression from automata). This technique has been improved by Hahn *et al.* [19] by directly computing and simplifying intermediate functions, as a major drawback of these techniques is the rapid growth of functions. The simplification involves the addition of functions where the costly operation of computing the greatest common divisor (gcd) needs to be performed. Jansen *et al.* [13] further improved the state elimination technique by combining it with SCC decomposition, and a dedicated gcd-computation operating on partial factorizations of polynomials. State elimination is the core of the tool `PARAM` [10] and has recently been adopted in `PRISM` [2]. These are—to the best of our knowledge—the only available tools for computing reachability probabilities and expected rewards of pMCs. Note that all these tools just output the rational function—sometimes accompanied by constraints ensuring well-definedness—while none of them directly addresses the synthesis problem.

Other works include parameter synthesis of timed reachability in parametric CTMCs [36–38], synthesis for interval MCs and ω-regular properties [39].

Seshia *et al.* [40] investigate probabilities which are modeled as convex functions over independent parameters. In model repair [9, 41], models refuting a given property are amended so as to satisfy this property. In this setting, parametric MCs are used as underlying model. The verification of MCs against parametric LTL formulas has been considered in [42]. Improved methods for single-parameter pMCs and nested reachability properties were presented in [43]. Finally, [16] presents several complexity results for perturbation analysis of pMCs.

Computing *conditional probabilities* for MCs has been considered in [44, 18]. Usually, conditional probabilities are computed by the so-called quotient method involving verifying ω-regular properties. Baier *et al.* [18] presented an elegant algorithm reducing the problem to compute reachability probabilities in the MC and a copy of it and experimentally showed the superiority of this approach.

7 Conclusion and Future Work

We presented the new tool `PROPhESY` dedicated to parameter synthesis for pMCs. Beyond the superior model checking times over existing tools, it offers automated and user-guided methods for partitioning the parameter space into safe and unsafe parts. The service oriented architecture and modularity allow for a high usability. Future work will consider the extension to parametric Markov decision processes as well as continuous-time MCs. A further important extension will be parameter synthesis for a higher number of parameters.

Acknowledgements

We want to thank Ernst Moritz Hahn for valuable discussions on computing conditional probabilities for parametric MCs.

References

1. Reiter, M.K., Rubin, A.D.: Crowds: Anonymity for web transactions. ACM Trans. on Information and System Security **1**(1) (1998) 66–92
2. Kwiatkowska, M., Norman, G., Parker, D.: Prism 4.0: Verification of probabilistic real-time systems. In: Proc. of CAV. Volume 6806 of LNCS, Springer (2011) 585–591
3. Katoen, J.P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Performance Evaluation **68**(2) (2011) 90–104
4. Garavel, H., Lang, F., Mateescu, R., Serwe, W.: CADP 2011: a toolbox for the construction and analysis of distributed processes. Software Tools for Technology Transfer **15**(2) (2013) 89–107
5. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: A web-based probabilistic model checker. In: Proc. of FM. Volume 8442 of LNCS, Springer (2014) 312–317
6. Alur, R., Henzinger, T.A., Vardi, M.: Theory in practice for system design and verification. ACM SIGLOG News **3** (2015)
7. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.P.: Performance evaluation and model checking join forces. Communications of the ACM **53**(9) (2010) 76–85
8. Calinescu, R., Ghezzi, C., Kwiatkowska, M.Z., Mirandola, R.: Self-adaptive software needs quantitative verification at runtime. Communications of the ACM **55**(9) (2012) 69–77
9. Bartocci, E., Grosu, R., Katsaros, P., Ramakrishnan, C., Smolka, S.A.: Model repair for probabilistic systems. In: Proc. of TACAS. Volume 6605 of LNCS. Springer (2011) 326–340
10. Hahn, E.M., Hermanns, H., Wachter, B., Zhang, L.: PARAM: A model checker for parametric Markov models. In: Proc. of CAV. Volume 6174 of LNCS, Springer (2010) 660–664
11. Daws, C.: Symbolic and parametric model checking of discrete-time Markov chains. In: Proc. of ICTAC. Volume 3407 of LNCS, Springer (2004) 280–294
12. Gruber, H., Johannsen, J.: Optimal lower bounds on regular expression size using communication complexity. In: Proc. of FOSSACS. Volume 4962 of LNCS, Springer (2008) 273–286
13. Jansen, N., Corzilius, F., Volk, M., Wimmer, R., Ábrahám, E., Katoen, J.P., Becker, B.: Accelerating parametric probabilistic verification. In: Proc. of QEST. Volume 8657 of LNCS, Springer (2014) 404–420
14. Lanotte, R., Maggiolo-Schettini, A., Troina, A.: Parametric probabilistic transition systems for system design and analysis. Formal Aspects of Computing **19**(1) (2007) 93–109
15. Clarke, E.M., Grumberg, O., Jha, S., Lu, Y., Veith, H.: Counterexample-guided abstraction refinement. In: Proc. of CAV. Volume 1855 of LNCS, Springer (2000) 154–169
16. Chen, T., Feng, Y., Rosenblum, D.S., Su, G.: Perturbation analysis in verification of discrete-time Markov chains. In: Proc. of CONCUR. Volume 8704 of LNCS, Springer (2014) 218–233
17. Su, G., Rosenblum, D.S.: Asymptotic bounds for quantitative verification of perturbed probabilistic systems. In: Proc. of ICFEM. Volume 8144 of LNCS, Springer (2013) 297–312
18. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in Markovian models efficiently. In: Proc. of TACAS. Volume 8413 of LNCS, Springer (2014) 515–530

19. Hahn, E.M., Hermanns, H., Zhang, L.: Probabilistic reachability for parametric Markov models. Software Tools for Technology Transfer **13**(1) (2010) 3–19
20. Baier, C., Katoen, J.P.: Principles of Model Checking. The MIT Press (2008)
21. Jovanovic, D., de Moura, L.M.: Solving non-linear arithmetic. In: Proc. of IJCAR. Volume 7364 of LNCS, Springer (2012) 339–354
22. Helmink, L., Sellink, M., Vaandrager, F.: Proof-checking a data link protocol. In: Proc. of TYPES. Volume 806 of LNCS, Springer (1994) 127–165
23. Jonsson, B., Larsen, K.G.: Specification and refinement of probabilistic processes. In: Proc. of LICS, IEEE CS (1991) 266–277
24. Baier, C., Hermanns, H.: Weak bisimulation for fully probabilistic processes. In: Proc. of CAV. Volume 1254 of LNCS, Springer (1997) 119–130
25. Barrett, R., Berry, M., Chan, T.F., Demmel, J., Donato, J., Dongarra, J., Eijkhout, V., Pozo, R., Romine, C., der Vorst, H.V.: Templates for the Solution of Linear Systems: Building Blocks for Iterative Methods, 2nd Edition. SIAM (1994)
26. `CArL` Website: (2015) `http://goo.gl/8QsVxv`.
27. Jones, E., Oliphant, T., Peterson, P., et al.: SciPy: Open source scientific tools for Python (2001–)
28. Corzilius, F., Loup, U., Junges, S., Ábrahám, E.: SMT-RAT: An SMT-compliant nonlinear real arithmetic toolbox - (tool presentation). In: Proc. of SAT. Volume 7317 of LNCS, Springer (2012) 442–448
29. Barrett, C., Stump, A., Tinelli, C.: The Satisfiability Modulo Theories Library (SMT-LIB). `www.SMT-LIB.org` (2010)
30. `PRISM` Website: (2015) `http://prismmodelchecker.org`.
31. `PARAM` Website: (2015) `http://depend.cs.uni-sb.de/tools/param/`.
32. `PROPhESY` Website: (2015) `http://moves.rwth-aachen.de/prophesy/`.
33. Bohnenkamp, H., Stok, P.V.D., Hermanns, H., Vaandrager, F.: Cost-optimization of the IPv4 zeroconf protocol. In: Proc. of DSN, IEEE CS (2003) 531–540
34. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6) (1985) 637–647
35. Han, J., Jonker, P.: A system architecture solution for unreliable nanoelectronic devices. IEEE Transactions on Nanotechnology **1** (2002) 201–208
36. Han, T., Katoen, J.P., Mereacre, A.: Approximate parameter synthesis for probabilistic time-bounded reachability. In: Proc. of RTSS, IEEE CS (2008) 173–182
37. Brim, L., Ceska, M., Drazan, S., Safránek, D.: Exploring parameter space of stochastic biochemical systems using quantitative model checking. In: Proc. of CAV. Volume 8044 of LNCS, Springer (2013) 107–123
38. Ceska, M., Dannenberg, F., Kwiatkowska, M.Z., Paoletti, N.: Precise parameter synthesis for stochastic biochemical systems. In: Proc. of CMSB. Volume 8859 of LNCS, LNCS (2014) 86–98
39. Benedikt, M., Lenhardt, R., Worrell, J.: LTL model checking of interval Markov chains. In: Proc. of TACAS. Volume 7795 of LNCS, Springer (2013) 32–46
40. Puggelli, A., Li, W., Sangiovanni-Vincentelli, A.L., Seshia, S.A.: Polynomial-time verification of PCTL properties of MDPs with convex uncertainties. In: Proc. of CAV. Volume 8044 of LNCS, Springer (2013) 527–542
41. Chen, T., Hahn, E.M., Han, T., Kwiatkowska, M., Qu, H., Zhang, L.: Model repair for Markov decision processes. In: Proc. of TASE, IEEE CS (2013) 85–92
42. Chakraborty, S., Katoen, J.P.: Parametric LTL on Markov chains. In: IFIP TCS. Volume 8705 of LNCS, Springer (2014) 207–221
43. Su, G., Rosenblum, D.S.: Nested reachability approximation for discrete-time Markov chains with univariate parameters. In: Proc. of ATVA. Volume 8837 of LNCS, Springer (2014) 364–379

44. Andrés, M.E., van Rossum, P.: Conditional probabilities over probabilistic and nondeterministic systems. In: Proc. of TACAS. Volume 4963 of LNCS, Springer (2008) 157–172