Computing Conditional Probabilities: Implementation and Evaluation

Steffen Märcker(✉), Christel Baier, Joachim Klein, and Sascha Klüppelholz

Institute of Theoretical Computer Science,
Technische Universität Dresden, 01062 Dresden, Germany
steffen.maercker@tu-dresden.de

Abstract. Conditional probabilities and expectations are an important concept in the quantitative analysis of stochastic systems, e.g., to analyze the impact and cost of error handling mechanisms in rare failure scenarios or for a utility-analysis assuming an exceptional shortage of resources. This paper reports on the main features of an implementation of computation schemes for conditional probabilities in discrete-time Markov chains and Markov decision processes within the probabilistic model checker Prism and a comparative experimental evaluation. Our implementation has full support for computing conditional probabilities where both the objective and condition are given as linear temporal logic formulas, as well as specialized algorithms for reachability and other simple types of path properties. In the case of Markov chains we provide implementations for three alternative methods (quotient, scale and reset). We support Prism's explicit and (semi-)symbolic engines. Besides comparative studies exploring the three dimensions (methods, engines, general vs. special handling), we compare the performance of our implementation and the probabilistic model checker Storm that provides facilities for conditional probabilities of reachability properties.

1 Introduction

Various methods and tools have been developed that support the analysis of systems against reliability and performability requirements. In this paper, we focus on the stochastic setting where finite-state Markovian models are used to carry out the system analysis (see, e.g., [29,35,44]). In the past 25 years, various algorithms to compute the probabilities for safety or liveness properties as well as time- or cost-bounded properties specified in some temporal logic have been proposed and implemented in tools. Most prominent is the probabilistic model checker Prism [39]. Among others, it provides support for the analysis of discrete- and continuous-time Markov chains as well as Markov decision processes (MDPs) against state properties specified in probabilistic computation tree logic (PCTL) [13,16,28] or ω-regular path properties specified in linear

The authors are supported by the DFG through the collaborative research centre HAEC (SFB 912) and the Excellence Initiative by the German Federal and State Governments (cluster of excellence cfaed) and projects BA 1679/11-1 and 1679/12-1.

A. Cimatti and M. Sirjani (Eds.): SEFM 2017, LNCS 10469, pp. 349–366, 2017.
DOI: 10.1007/978-3-319-66197-1_22

temporal logic (LTL) [18,46]. The focus of Prism and other probabilistic model checkers, such as MRMC [34], Storm [20], iscasMC [27] or Marcie [30], is to provide facilities for the computation of probabilities for temporal path properties, and partly also expected accumulated costs. However, tool support for conditional probabilities, i.e., the probability of an event φ given an event ψ occurs, for temporal properties or conditional expected costs, i.e., the expected costs given an event ψ occurs, in Markovian systems within probabilistic model checkers is rare (see the discussion on related work). This is unfortunate as assertions on conditional probabilities for temporal properties can provide important insights in the reliability of systems or the cost-utility balance. For instance, conditional probabilities and expectations allow to "zoom in" specific error scenarios and to analyze the impact of undetected errors or the cost of repair mechanisms in cases where an error is detected. Conditional probabilities can also serve to formalize the tradeoff between cost and utility functions, e.g., in terms of the conditional probability to achieve a sufficiently high utility level, assuming that a scenario occurs where the available energy is bounded by some constant (see [7]). For another example, forms of anonymity have been formalized by the requirement that the conditional probabilities for on observable o, given a secret s, does not depend on s (see [2]). In the context of the verification of probabilistic programs, conditional probabilities and expectations are used to formalize the semantics of loops under the assumption that the loop terminates [14,17,32].

Contribution. We present an implementation of computation schemes for conditional probabilities in Markov chains and MDPs, where both the objective and condition are LTL formulas, within the probabilistic model checker Prism [39] (Sect. 3).[1] Our implementation realizes the algorithms of [10], which rely on reductions to unconditional probabilities, and achieves a major speed-up over [10]'s prototype implementation. It uses Prism's infrastructure to translate LTL formulas into deterministic finite or ω-automata and can be used with all of Prism's analysis engines, namely explicit, mtbdd, hybrid, and sparse.[2] Besides the automata-based approach for LTL objectives and conditions, our implementation also provides specialized treatment of certain simple formula patterns, e.g., when the condition is a reachability property or an invariant. In the case of Markov chains, our implementation also supports the computation of conditional expected accumulated costs when the condition is an LTL formula.

The second contribution is a report on experiments that have been carried out to evaluate the general feasibility of the algorithms for conditional probabilities of LTL objectives and conditions as well as to compare different methods and

[1] In the context of a conditional probability $\Pr(\varphi|\psi)$ we refer to φ as the objective and to ψ as the condition.

[2] Prism's explicit engine uses sparse matrix representation for the system and carries out all computation in an explicit manner, while the other three engines use multi-terminal binary decision diagrams (MTBDDs) for the model construction. The mtbdd engine is purely MTBDD-based. The hybrid engine uses an MTBDD-representation for the system and an explicit probability vector [38], while the sparse engine uses sparse matrices for the numerical computations.

engines (Sect. 4). For Markov chains, we compare the performance of three methods: the *quotient method* (which computes conditional probabilities according to their definition), the *scale method* proposed in [10] (which transforms the Markov chain into a new one by deleting and copying certain states and rescaling the transition probabilities) and the *reset method* proposed in [10] for MDPs (which "discards" paths that violate the condition by reset transitions returning to the initial state). In the case of MDPs, we compare the running time of the reset method for computing the maximal and minimal condition probability with the running time for computing the unconditional probability for the conjunction of the objective and the condition. For both Markov chains and MDPs, we also compare the running times of our implementation for each of PRISM's engines as well as the general applicable automata-based approach and the specialized treatment of formula patterns. Finally, we also compare the performance of our implementation and that of the new probabilistic model checker STORM, which provides support for reachability objectives and conditions using its explicit engine (Sect. 5). An extended version including an appendix [41], our implementation, all benchmark models and queries as well as the raw data from the measurements are available online[3].

Related Work. For Markov chains (and other purely stochastic models without nondeterminism), the definition $\Pr(\varphi|\psi) = \Pr(\varphi \wedge \psi)/\Pr(\psi)$ of conditional probabilities yields a direct reduction to the unconditional case, called *quotient method*. Thus, all tools with facilities for computing probabilities of conjunctive properties in Markov chains provide an implicit way to compute conditional probabilities. However, this method is not applicable for conditional expectations. Moreover, tools that are restricted to branching-time logics such as PCTL or its continuous-time analogue CSL [5,8] do not support conjunctive path properties. This motivated the work by Gao et al. [24] on conditional probabilities for conditions with nested time-bounded constraints in continuous-time Markov chains. This technique has been implemented in the tool CCMC [23] and adapted in [33] for the discrete-time setting. We are not aware of an implementation of the algorithm proposed in [33]. In our previous work [10], we presented the *scale method* (see. Sect. 2) and reported on a prototypical implemented in PRISM's explicit engine. The scale method has also been implemented in STORM [20] (and its parametric branch PROPHESY [19]) for cases where both the objective φ and the condition ψ are reachability properties. While STORM generally supports both explicit and symbolic computations, the implementation for conditional probabilities and expectations in Markov chains in STORM is limited to the explicit engine using sparse data structures.

In the case of probabilistic models with nondeterminism such as MDPs, the typical task for the system analysis is to carry out a worst- or best-case analysis in terms of the maximal or minimal (unconditional or conditional) probability that can be achieved when ranging over all possible resolutions of the nondeterminism (formalized by schedulers, see Sect. 2). [1,3] proposed a model-checking algorithm for PCTL extended by constraints for conditional probabilities where

[3] https://wwwtcs.inf.tu-dresden.de/ALGI/PUB/SEFM17.

the objective and the condition are PCTL path formulas. [1,3]'s algorithm for the case where both the objective and the condition are reachability properties has exponential-time complexity, while the *reset method* proposed in [10] is polynomially time-bounded. Both the prototype reported in [10] and the tool Storm [20] realize the reset method for maximal conditional probabilities in MDPs only for reachability objectives and conditions and only non-symbolic, whereas the new implementation presented in this paper covers Prism's symbolic engines, too. To the best of our knowledge, it is the first implementation providing full support for computing maximal or minimal conditional probabilities in MDPs for LTL objectives and LTL conditions.

2 Preliminaries

We provide an informal overview of the scale and reset methods of [10] underlying our implementation. For the basic principles of Markov chains, Markov decision processes and probabilistic model checking, we refer to [22,44] and Chap. 10 of [9]. We suppose some familiarity with ω-automata (finite automata over infinite words) and linear temporal logic (LTL). See, e.g., [9,25].

Markov decision processes (MDP) can be seen as a finite-state automata model where each state has a nondeterministic choice between finitely many actions. The effect of taking an action in a state is probabilistic and formalized by a probabilistic distribution over the states. By supporting nondeterministic and probabilistic choices, MDPs are a generic model with manyfold applications, e.g., in robotics, operations research and randomized distributed systems. Probabilistic model checking can be used, e.g., to compute the maximal or minimal probability that an ω-regular path property φ holds in state s of an MDP $\mathcal{M}$, denoted $\Pr^{\max}_{\mathcal{M},s}(\varphi)$ resp. $\Pr^{\min}_{\mathcal{M},s}(\varphi)$. The extremum is taken over all resolutions of the nondeterminism in the MDP, formalized as *schedulers* that select for each finite path an action that is enabled in the last state of the path. Thus, $\Pr^{\max}_{\mathcal{M},s}(\varphi) = \max_\sigma \Pr^\sigma_{\mathcal{M},s}(\varphi)$ resp. $\Pr^{\min}_{\mathcal{M},s}(\varphi) = \min_\sigma \Pr^\sigma_{\mathcal{M},s}(\varphi)$ where σ ranges over all schedulers and $\Pr^\sigma_{\mathcal{M},s}$ denotes the probability measure induced by scheduler σ and starting state s. (The maximum and minimum indeed exist.) Discrete-time Markov chains (DTMCs) can be regarded as purely probabilistic MDPs, i.e., each state has at most one enabled action. Thus, DTMCs have a unique scheduler. Given a DTMC $\mathcal{M}$, we write $\Pr_{\mathcal{M},s}(\varphi)$ as $\Pr^{\max}_{\mathcal{M},s}(\varphi) = \Pr^{\min}_{\mathcal{M},s}(\varphi)$.

LTL formulas are built by atomic propositions for the states, the standard Boolean operators (such as $\vee$ "or" and $\wedge$ "and" and $\neg$ "negation") and the unary temporal modalities $\Diamond$ ("eventually"), $\Box$ ("globally"), $\bigcirc$ ("next") as well as the binary temporal modality U ("until"), its dual R ("release") and W ("weak until"). All path properties that are expressible in LTL are ω-regular.

Maximal reachability probabilities $\Pr^{\max}_{\mathcal{M},s}(\Diamond F)$ in MDPs are known to be computable via iterative methods (value or policy iteration) or in polynomial-time via linear programming techniques. To compute the maximal probabilities for LTL formulas, one can use standard algorithms to translate the given LTL formula φ into a deterministic ω-automaton $\mathcal{A}$ and then compute $\Pr^{\max}_{\mathcal{M},s}(\varphi)$ as

the maximal probability of a reachability property in the product-MDP $\mathcal{M} \otimes \mathcal{A}$. Minimal probabilities for reachability properties or LTL formulas can be computed in a similar way or using the fact that $\Pr^{\min}_{\mathcal{M},s}(\varphi) = 1 - \Pr^{\max}_{\mathcal{M},s}(\neg\varphi)$.

If $\mathcal{M}$ is a DTMC, the *conditional probability* of objective φ given that condition ψ holds, starting from state s, is given by:

$$\Pr_{\mathcal{M},s}(\varphi \mid \psi) \quad \stackrel{\text{def}}{=} \quad \frac{\Pr_{\mathcal{M},s}(\varphi \wedge \psi)}{\Pr_{\mathcal{M},s}(\psi)} \qquad (*)$$

where we suppose that φ and ψ are LTL formulas (or, more generally, ω-regular path properties) such that $\Pr_{\mathcal{M},s}(\psi) > 0$.

Quotient Method. For DTMCs, (*) directly provides a computation scheme for conditional probabilities via the computation of the unconditional probabilities for the conjunction of the objective and condition and for the condition.

Scale Method. [10] presents an alternative method for the computation of the conditional probabilities $\Pr_{\mathcal{M},s}(\varphi \mid \psi)$ in DTMCs, which relies on a reduction to unconditional probabilities in a transformed DTMC $\mathcal{M}_\psi$. We sketch the ideas for the case where $\psi = \Diamond C$ is a reachability condition ("eventually some state in C is reached"). In a first step, the probabilities $x_s = \Pr_{\mathcal{M},s}(\Diamond C)$ are computed for all states s in $\mathcal{M}$. Intuitively, the new DTMC $\mathcal{M}_\psi$ simulates $\mathcal{M}$ using in two modes: "before C" and "after C". The structure of $\mathcal{M}_\psi$'s "before C" mode is obtained from $\mathcal{M}$ by removing all states t with $x_t = 0$ and re-scaling the transition probabilities for the remaining states. More precisely, if $x_s > 0$ and $s \notin C$ then the transition probability from s to t in the "before C" mode of $\mathcal{M}_\psi$ is obtained by multiplying the transition probability from s to t in $\mathcal{M}$ with x_t/x_s. If $s \in C$ then $\mathcal{M}_\psi$ switches to the "after C" mode where it behaves exactly as $\mathcal{M}$, i.e., the structure of $\mathcal{M}_\psi$'s "after C" mode is a copy of $\mathcal{M}$. Then, $\Pr_{\mathcal{M},s}(\varphi \mid \Diamond C) = \Pr_{\mathcal{M}_\psi,s_b}(\varphi)$ for each measurable path property φ and each state s with $x_s > 0$ where s_b means the copy of s in the "before C" mode. The case where ψ is an arbitrary LTL formula is reducible to the case of a reachability condition using a product construction with a deterministic ω-automaton for ψ. As $\mathcal{M}_\psi$ is independent of the objective, it can also be used to compute conditional expected accumulated costs in $\mathcal{M}$. It is also applicable for continuous-time Markov chains and (time-abstract) LTL conditions.

Reset Method. The reset method has been proposed in [10] to compute maximal conditional probabilities for ω-regular objectives and conditions in MDPs via a reduction to unconditional probabilities. It supposes a fixed initial state s_{init} and relies on a transformation of $\mathcal{M}$ into an MDP $\mathcal{N} = \mathcal{M}_{\varphi|\psi}$ that depends on both the objective and the condition. We first summarize the ideas of the reset method when applied to a DTMC and then sketch the main steps for MDPs.

Let $\mathcal{M}$ be a DTMC. For reachability objectives and conditions, say $\varphi = \Diamond F$ and $\psi = \Diamond C$, where we suppose that C is reachable from the initial state s_{init}, the reset method first performs a reachability analysis to identify all states s that cannot reach C, in which case $\Pr_{\mathcal{M},s}(\psi) = 0$. It then discards the outgoing

transitions of all states s with $\Pr_{\mathcal{M},s}(\psi) = 0$ and introduces a reset transition from these states s to s_{init} with transition probability 1. If $\mathcal{N}$ is the resulting DTMC then $\Pr_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) = \Pr_{\mathcal{N},s_{init}}(\varphi)$. Thus, the reset method provides a reduction of conditional probabilities to unconditional ones. The essential idea of the switch from $\mathcal{M}$ to $\mathcal{N}$ is that via the reset transitions the probability mass of all paths that do not satisfy the condition ψ are transferred to the initial state, where it is distributed over all paths satisfying ψ.

Given an MDP $\mathcal{M}$ with a initial state s_{init}, we address the task to compute:

$$\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) \quad \stackrel{\text{def}}{=} \quad \max_{\sigma} \Pr^{\sigma}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) \quad = \quad \max_{\sigma} \frac{\Pr^{\sigma}_{\mathcal{M},s_{init}}(\varphi \wedge \psi)}{\Pr^{\sigma}_{\mathcal{M},s_{init}}(\psi)}$$

for given LTL formulas φ and ψ. The maximum ranges over all schedulers σ with $\Pr^{\sigma}_{\mathcal{M},s_{init}}(\psi) > 0$. If there is at least one scheduler where $\Pr^{\sigma}_{\mathcal{M},s_{init}}(\psi) > 0$ then the maximum exists. Minimal conditional probabilities are defined analogously and can be handled using $\Pr^{\min}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) = 1 - \Pr^{\max}_{\mathcal{M},s_{init}}(\neg\varphi \,|\, \psi)$. The quotient and scale methods are not applicable for MDPs, as the task is to maximize the quotient of the probabilities for $\varphi \wedge \psi$ and ψ w.r.t. the same scheduler.

The reset method of [10] for computing $\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi)$, where $\varphi = \Diamond F$ and $\psi = \Diamond C$ are reachability properties, works as follows. It first applies standard algorithms to check whether $\Pr^{\max}_{\mathcal{M},s_{init}}(\psi) > 0$, as otherwise the maximal conditional probability is undefined. If so, $\mathcal{M}$ is transformed into a normal form MDP $\mathcal{M}'$ with $\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) = \Pr^{\max}_{\mathcal{M}',s_{init}}(\varphi' \,|\, \psi')$ where $\varphi' = \Diamond F'$ and $\psi' = \Diamond C'$ are reachability properties with $F' \subseteq C'$ and where all states in C' are traps, i.e., they do not have enabled actions. $\mathcal{M}'$ contains a further trap state *fail* and enjoys the property $\Pr^{\max}_{\mathcal{M}',s}(\Diamond C') > 0$ for all states $s \neq \mathit{fail}$. Having constructed $\mathcal{M}'$, the reset method transforms $\mathcal{M}'$ into another MDP $\mathcal{N}$ that is obtained by adding new transitions, called reset transitions, from *fail* and from all other states s in $\mathcal{M}'$ with $\Pr^{\min}_{\mathcal{M}',s}(\Diamond C') = 0$ to s_{init}. The reset transitions have a special action name. For the states s in $\mathcal{M}'$ with $s \neq \mathit{fail}$ and $\Pr^{\min}_{\mathcal{M}',\mathit{fail}}(\Diamond C') = 0$, the reset transition is an additional nondeterministic alternative. Then, $\Pr^{\max}_{\mathcal{M},s_{init}}(\varphi \,|\, \psi) = \Pr^{\max}_{\mathcal{N},s_{init}}(\Diamond F')$. The case where φ, ψ are ω-regular properties is reducible to the base case where the condition and objective are reachability properties using deterministic ω-automata for φ and ψ.

3 Implementation

The foundation of our new implementation is the development branch of PRISM 4.3. We implemented the three methods (quotient, scale and reset) for DTMCs as well as the reset method for DTMCs and MDPs where the objective and the condition are given as LTL formulas. All methods can be employed using PRISM's explicit, mtbdd, hybrid and sparse engine. We rely on PRISM's infrastructure for generating deterministic finite or ω-automata from LTL formulas. To facilitate a more direct and performant implementation of model transformations used in the scale and the reset method, we developed a declarative framework based on atomic operations like *union of models* and *state space restriction*.

Specialized Treatment of Formula Pattern. Besides the automata-based approach of the scale method for DTMCs to treat LTL conditions, our implementation provides direct support for conditions of the form $\Diamond A$ (reachability), $\Box A$ (invariance), $\bigcirc A$ (next), $A \,\mathrm{U}\, B$ (until), $A \,\mathrm{W}\, B$ (weak until), and $A \,\mathrm{R}\, B$ (release), where A and B are sets of states. The underlying algorithms rely on the same concepts as the treatment of reachability conditions in the scale method for Markov chains presented in [10]. In Sect. 4, we refer to these specialized treatment of the scale method as scale.pattern.

Likewise, for the reset method in DTMCs or MDPs, our implementation provides direct support to treat conditions and/or objectives that are reachability, invariance, (weak) until, or release properties without translating them into deterministic ω-automata. Altogether this yields the following four variants of the reset method. The original method of [10] is denoted by reset.ltl-ltl and uses automata for both the objective and the condition. We shall write reset.pattern-ltl for the reset method that treats the objective directly (assuming the objective is a reachability, invariance, (weak) until, or release property), while the condition is translated to an automaton. reset.ltl-pattern and reset.pattern-pattern have analogous meanings.

Infrastructure Improvements. To increase the efficiency of Prism's computation of unconditional probabilities, we addressed several weaknesses in Prism's explicit engine: (i) the replacement of quadratic-time with linear-time algorithms used in a preprocessing step that identifies all states s where a reachability condition holds with probability 1 under all or some schedulers; (ii) techniques to restrict products of models and ω-automata to the product states reachable from the initial states; (iii) the conversion of graph representations of DTMCs into more compact sparse matrices enabling a faster algorithmic treatment.

4 Evaluation and Comparative Studies

4.1 Methodology

We carried out all measurements on a compute server[4] and configured CUDD (the package for manipulating (MT)BDDs used in Prism) and the JVM to use up to 31 GB RAM each. As recommended in production environments, we fixed the JVM heap size and set the integer cache to hold all boxed integers from 0 to $30 \cdot 10^6$. All experiments timed out after 3600 s. Multiple runs using Prism's (semi-)symbolic engines yielded the same times and were only negligibly effected by running them in parallel. However, the explicit engine showed deviations, mainly due to JIT warmup effects. Hence, in order to obtain reliable measurements, we ran each property and model instance at least twice, and refrained from running the explicit experiments in parallel. Generally, we consider times ≤ 1 s below the precision of measurements and time differences of a few seconds insignificant.

[4] 2 × Intel Xeon E5-2680 (Octa Core, Sandy Bridge) @ 2.70 GHz, 384 GB RAM; Turbo Boost and Hyper Threading enabled; Debian GNU/Linux 8.3.

At the core of probabilistic model checking is the computation of reachability probabilities. PRISM's default methods for this task are different variants of *value iteration*. With the standard termination criterion, the value iteration may abort prematurely and return incorrect results [26]. In [12], we developed a prototype of the *interval iteration* proposed in [26] addressing this issue. However, it is not yet integrated into the current implementation discussed in this paper, but spot tests suggested that our benchmarks do not suffer from that issue.

4.2 Considered Models and Properties

Our experiments aim for an exhaustive examination of the implemented methods and patterns with respect to their performance in comparison to each other. We use established models from the PRISM benchmark suite [40] that allow the formulation of meaningful conditional queries. We included three DTMCs: the *bounded retransmission protocol* [31] (brp), the *crowds protocol for anonymity* [45] (crowds) and the *probabilistic contract signing protocol* [21,43] from Even, Goldreich & Lempel (egl), as well as two MDP models: the *randomized consensus shared coin protocol* [4,36] (consensus) and a model of *CSMA/CA from the IEEE standard 802.11* [37] (wlan). For each model, we combined various temporal path formulas for the objective and condition to cover all the instantiations of the implemented patterns. As we do not want to evaluate the translation from LTL to automata we considered only LTL path formulas of medium complexity that yielded automata with at most 16 states. Our selection includes all models (brp, crowds, wlan) and conditional queries previously used for benchmarking in [10,19]. In total, there are 190 runs required for each instance of a DTMC and 79 runs for each MDP instance. Among the considered properties are the following examples of interest that cannot be computed efficiently without the transformation-based methods. In the case of the *brp* protocol one might be interested in "the expected energy consumption of a successful transfer given the transfer is indeed successful" which has a probability <1. For the *consensus* protocol where multiple processes have to agree on a common date based on coin-flipping, we ask for "the maximal probability that the processes eventually disagree given a defective process always flips coin".

4.3 Results

We present numbers for the explicit and the mtbdd engine in this section and cover the hybrid and sparse engine in [41].

Methods. As the size of the state space and hence the runtimes grow exponentially, we use log-scale for the charts. Figure 1 compares the scale method against the quotient method in DTMCs. It incorporates the results from all DTMC model instances and all properties. The quotient method serves as reference and is plotted on the diagonal. We compare against the minimal time over all patterns of the scale method (scale.min(*)). For the majority of experiments, the scale method is faster than the quotient method. The margin increases with

(a) explicit engine (b) mtbdd engine

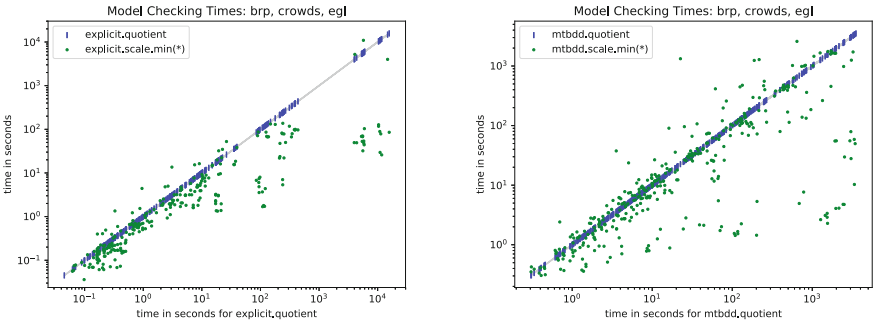
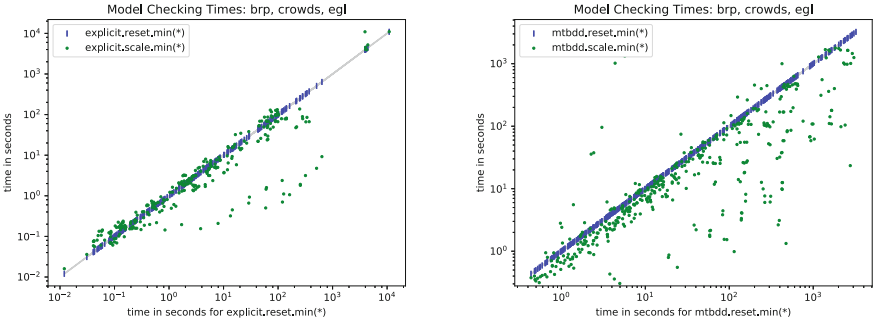


Fig. 1. Runtimes of methods: scale vs quotient (DTMCs)



(a) explicit engine (b) mtbdd engine

Fig. 2. Runtimes of methods: reset vs scale (DTMCs)

(a) explicit engine (b) mtbdd engine

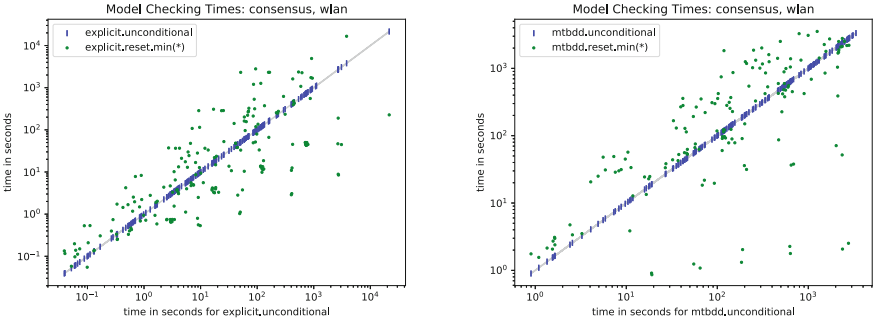


Fig. 3. Runtimes of methods: reset vs objective $\wedge$ condition (MDPs)

the size of the model up to two orders of magnitude for the explicit engine. This is mainly caused by the increasing costs of the quotient method to handle the conjunction of the objective and condition. Besides a few more instances where the quotient method is faster, using the mtbdd engine yields similar results. This observation transfers to the (semi-)symbolic engines, see [41].

Pitted against the reset method in DTMCs, the scale method is faster in almost all experiments. We plot the minimal times for the scale method against the minimal times of the reset method as reference in Fig. 2. Often the difference is not more than one order of magnitude with some notable exceptions. On the one hand, the cases where the reset method is beaten more strikingly are caused by the more costly handling of queries featuring non-simple path properties as objective and condition. On the other hand, certain objectives require three instead of two model-checking runs to compute the probabilities required to construct the normal form MDP $\mathcal{M}'$. However, the transformed model can be very small, e.g., only 6 states and hence, the final computation step fast enough to compensate for the expenses of the reset method. In very large realistic models this might be crucial to enable the computation of conditional probabilities after all.

In the absence of alternative algorithms for conditional probabilities in MDPs, we compare the reset method against the computation of the unconditional probability for the conjunction of the objective and condition. Figure 3 shows the results for the minimal time over all applicable patterns. The majority of the measurements is relatively close to the reference times, i.e., less than an order of magnitude faster or slower. This observation applies to all four engines and yields evidence that computing conditional probabilities in MDPs can be expected to be roughly as time-consuming as computing the unconditional probability of the conjunction of the objective and the condition. Furthermore, treating the general variant reset.ltl-ltl requires at most the amount of memory needed to construct the product of the original model and the automata for the objective and condition, respectively, and less if a specialized pattern applies.

(a) explicit engine (b) mtbdd engine

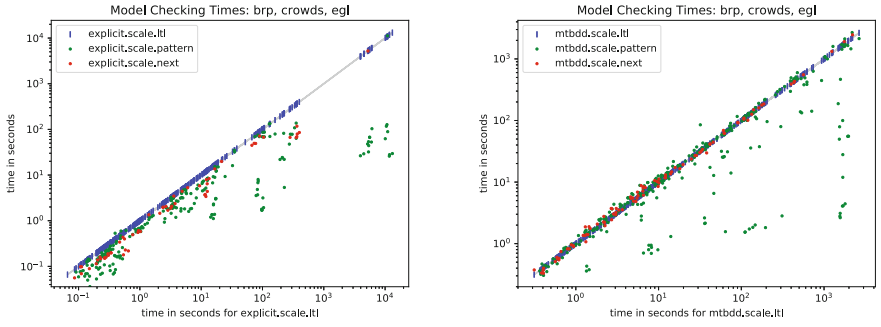


Fig. 4. Runtimes of patterns: scale.* vs scale.ltl (DTMCs) (Color figure online)

(a) explicit engine (b) mtbdd engine

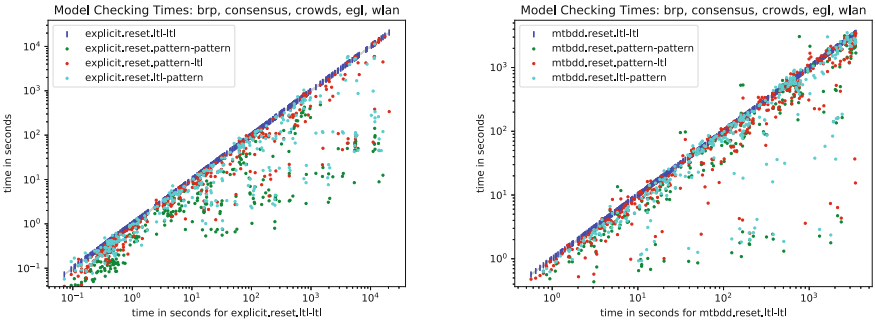


Fig. 5. Runtime of patterns: reset.* vs reset.ltl-ltl (DTMCs & MDPs)

Patterns. Figure 4 compares the runtimes of the specialized treatment of patterns scale.pattern against the general case using scale.ltl (patterns are depicted in different colors). Dots below the diagonale depict runtimes that are better than the general case which is on the diagonale. In general, the specialized methods show significant superiority over the automata-based approach. For the explicit engine (on the left), the special handling gains up to two and a half orders of magnitude. The few exceptions in the lower left are caused by fluctuations below the precision of measurements of about 1 s. Using the mtbdd engine (on the right) the picture is not as clear, since most of the measurements are closer to the general case. The reason for this is that in a purely symbolic approach, the computation time of the probabilities for the scaling step becomes an even more dominating factor. For the semi-symbolic engines hybrid and sparse this effect is not that distinct, and therefore, the results for these engines are similar to the explicit case, see [41]. Though, some cases perform considerably better with an advantage of up to three orders of magnitude (lower right).

Figure 5 considers the specialized patterns of the reset methods for both, DTMCs and MDPs. For all engines, the specialized patterns are clearly faster than the most general pattern reset.ltl-ltl, in larger model instances even up to 3 orders of magnitude. In fact, all specialized methods, e.g., reset.ltl-pattern, outperform the more general patterns, like reset.ltl-ltl, see [41] for details. There are a few, though non-representative, exceptions, most notable using the mtbdd engine. They issue from the wlan protocol and two properties having a certain (simple) until path property as objective. The specialized patterns reset.pattern-pattern and reset.pattern-ltl require the computation of minimal probabilities for the set of states falsifying until objectives, whereas the other patterns don't. As a result, in that MDP this specific computation is more time-consuming than building a product model from a simple automaton.

Engines and Model Sizes. The enhanced explicit engine showed decent performance for model sizes up to a few million states, see, e.g., Sect. 5, Table 2.

Furthermore, it beats the performance of PRISM's symbolic engines in the protocols brp, crowds, and consensus. The protocols egl and wlan however proved to be particularly suitable for symbolic treatment. Here, the symbolic engines achieved runtimes of only a couple of seconds and we were able to scale our experiments up to egl instances of $32 \cdot 10^{12}$ states and wlan instances with 10^9 states. Table 1 provides an overview of the maximal model sizes that occurred in our experiments (see [41] for similar statistics on the MTBDD size). It lists the largest model instances where all queries have successfully been computed and the minimal, maximal and mean size of all transformed models.

Table 1. Number of states

Model		Original #	Transformed: reset			Transformed: scale		
			min	mean	max	min	mean	max
DTMCs	brp	$113 \cdot 10^3$	5	$25 \cdot 10^3$	$119 \cdot 10^3$	$12 \cdot 10^3$	$106 \cdot 10^3$	$223 \cdot 10^3$
	crowds	$189 \cdot 10^6$	6	$5 \cdot 10^6$	$16 \cdot 10^6$	$11 \cdot 10^6$	$86 \cdot 10^6$	$189 \cdot 10^6$
	egl	$32 \cdot 10^{12}$	6	$332 \cdot 10^3$	$2 \cdot 10^6$	$62 \cdot 10^6$	$18 \cdot 10^{12}$	$32 \cdot 10^{12}$
MDPs	consensus	$17 \cdot 10^6$	$1 \cdot 10^6$	$6 \cdot 10^6$	$20 \cdot 10^6$	n.a.	n.a.	n.a.
	wlan	$4 \cdot 10^6$	18	$413 \cdot 10^3$	$12 \cdot 10^6$	n.a.	n.a.	n.a.

Bad Convergence of Reset Method. The reset method's core idea is to redistribute the probability mass of paths not satisfying the condition ψ over the paths satisfying it. This can impose problems in the application of approximation techniques like value iteration. If the model features a rather chain-like topology, it might take a huge number of iteration steps to eventually reach convergence. This phenomenon occurred in the brp protocol, even in a comparably small instance with only about a hundred thousand states. For five properties sharing the same globally condition, the transformation time is only a couple of seconds but the time required to compute the reachability probability in the transformed models is several hundred seconds [41]. However, the other models of our benchmarks are not effected.

5 Comparison Prism vs Storm

In [19], Dehnert et al. report on an experimental evaluation of an early version of their model checker STORM that was included in the PROPHESY tool and our previous prototypical implementation [10] for PRISM's explicit engine. For two benchmark models—brp and crowds—they were able to demonstrate a significant performance advantage of the STORM implementation.

In this section, we provide a comparison of our new implementation with the current, stand-alone release (Version 1.0) of STORM [19]. It supports the computation of conditional probabilities in DTMCs and MDPs but only if both, the objective and the condition, are reachability properties. For MDPs, only maximal probabilities are supported. The computation is based on the algorithms of [10],

i.e., the scale method for DTMCs and the reset method for MDPs. Support is limited to Storm's explicit engine (named "sparse" in Storm).

Engines. We compare the performance of Prism's and Storm's explicit engines and include our measurements for Prism's symbolic engines as well. Additionally, we provide a comparison against our former prototypical implementation [10] (denoted by explicit'14), as well as a port of that prototype to Prism 4.3 and the enhanced infrastructure (explicit*). This will allow to distinguish performance improvements due to the general enhancements and those related to the enhanced methods for the computation of condition probabilities.

Solvers. Storm provides a multitude of implementations of solvers for systems of linear equations (SLE), partially relying on highly-efficient external libraries. We provide a comparison (i) using the *default solvers* of both Prism and Storm, as well as (ii) using the *same solvers* for both model checkers. This allows to determine the impact of solvers on the performance. For *default solvers*, Prism uses the Jacobi method in the symbolic engines and the Gauss-Seidel method in the explicit engine in combination with value iteration in the case of MDPs. For DTMCs, Storm defaults to the *generalized minimal residual method* (GMRES) and preconditioning with *incomplete LU decomposition* (ILU). For MDPs, Storm uses value iteration with the power method. For the comparison using the *same solvers* for SLEs and linear programs (LP), we changed the settings for Prism to use value iteration with the power method for MDPs, and for Storm to use the Gauss-Seidel method for DTMCs.

Table 2. Sum of model-checking times, default solvers

Model		Prism						Storm
		hybrid	mtbdd	sparse	explicit	explicit*	explicit'14	explicit
DTMCs	brp	349 s	18,413 s	334 s	45 s	44 s	3515 s	4 s
	crowds	309 s	409 s	322 s	148 s	124 s	1933 s	55 s
	egl	242 s	13 s	299 s	242 s	412 s	2117 s	74 s
MDPs	consensus	319 s	2518 s	106 s	41 s	68 s	116 s	120 s
	wlan	70 s	16 s	46 s	45 s	264 s	241 s	26 s

Results. We present an aggregated comparison, one for the *default settings* (Table 2) and one with the *same solvers* (Table 3). Comprehensive tables can be found in [41]. Comparing the performance of explicit'14 and explicit*, the significant impact of the infrastructure improvements is apparent. In our observation this is mostly due to the computations of the state sets having probability 0 or 1 and the use of sparse data structures for DTMCs. Comparing explicit and explicit*, we see a considerable performance improvement for some of the models due the special handling of formula patterns and further enhancements in the computation of conditional probabilities (see Sect. 3), especially in MDPs. The new implementation employs additional graph analysis to reduce the size of the transformed models which does not pay off in the crowds protocol.

Comparing PRISM's and STORM's explicit engines using the *default settings* (Table 2), we see that STORM has better performance for the considered DTMC instances. Comparing this to STORM's performance when forcing the *same solvers* (Table 3), we can see that this advantage is due to the integration of the highly optimized linear equation solvers in STORM. For the egl protocol, STORM is still faster in this setting. But we suspect this is mainly due to the size of the model, which causes PRISM to operate close to the memory limit. This has a considerable performance impact caused by excessive garbage collection.

For MDPs, we achieve better performance using the explicit engine for the consensus case study, while STORM achieves better performance in the wlan protocol. Switching to the *same solvers*, i.e., value iteration using the power method, only has an impact on the consensus results for PRISM's explicit engine.

Taking both scenarios into account, we conclude that efficient solution methods for SLEs and LPs are a dominating factor besides sophisticated methods for the computation of conditional probabilities. Offering a variety of efficient solvers the user can choose between, STORM excels in this respect [20]. This is illustrated by the observation that STORM's internal value iteration is much slower for certain protocols, e.g., wlan, than the default, external implementation. However, the experiments also suggest that for the implementation of model-checking algorithms, choosing Java must not be a disadvantage performance-wise. Further optimized implementations can be expected to approach the performance of implementations in the C programming language [15,42]. A comparison of the build times of PRISM and STORM is presented in [41]. Furthermore, experiments on another computer resulted in surprisingly huge time differences, although the general trend does not change, see [41].

Table 3. Sum of model-checking times, same solvers

Model		PRISM						STORM
		hybrid	mtbdd	sparse	explicit	explicit*	explicit'14	explicit
DTMCs	brp	349 s	18,413 s	334 s	45 s	44 s	3515 s	228 s
	crowds	309 s	409 s	322 s	148 s	124 s	1933 s	192 s
	egl	242 s	13 s	299 s	242 s	412 s	2117 s	89 s
MDPs	consensus	310 s	2684 s	109 s	79 s	128 s	177 s	120 s
	wlan	78 s	16 s	47 s	48 s	262 s	229 s	26 s

6 Conclusion

Despite the significance of conditional probabilities and the numerous tools for computing unconditional probabilities for temporal properties in Markovian models, tool support for conditional probabilities was very limited. To the best of our knowledge, our implementation is the first that supports the computation of conditional probabilities in Markovian models where the objective and the condition are temporal properties specified as LTL formulas. We evaluated different

methods (quotient, scale and reset for DTMCs) and explicit and (semi-)symbolic engines. The experiments indicate a superiority of the scale method over the reset method, which again performs better than the naive quotient method in most cases. Further performance improvements are obtained by using specialized (automata-less) treatments of simple path properties rather than the generic automata-based reduction to reachability properties. In many cases, the performance of the reset method for maximal conditional probabilities in MDPs was fairly in the same order as the computation of unconditional probabilities for conjunctive events. As an overhead compared to the unconditional case is unavoidable, this can be taken as evidence for the general feasibility of reasoning about extremal conditional probabilities for temporal properties in MDPs.

Our implementation allows to compute conditional expected accumulated costs in Markov chains using the scale method. In future work we plan to mature and integrate the prototypical implementation of the algorithm [11] for the computation of maximal expected accumulated costs in MDPs. Other future directions include the integration of our implementation in the public version of PRISM to ease links to other facilities, such as the interval iteration [12,26] or the Hanoi framework [6] to provide support for the full class of ω-regular objectives and conditions.

References

1. Andrés, M.: Quantitative analysis of information leakage in probabilistic and nondeterministic systems. Ph.D. thesis, UB Nijmegen (2011)
2. Andrés, M.E., Palamidessi, C., van Rossum, P., Sokolova, A.: Information hiding in probabilistic concurrent systems. Theor. Comput. Sci. **412**(28), 3072–3089 (2011)
3. Andrés, M.E., van Rossum, P.: Conditional probabilities over probabilistic and nondeterministic systems. In: Ramakrishnan, C.R., Rehof, J. (eds.) TACAS 2008. LNCS, vol. 4963, pp. 157–172. Springer, Heidelberg (2008). doi:10.1007/978-3-540-78800-3_12
4. Aspnes, J., Herlihy, M.: Fast randomized consensus using shared memory. J. Algorithms **11**(3), 441–461 (1990)
5. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.K.: Model-checking continous-time Markov chains. ACM Trans. Comput. Logic **1**(1), 162–170 (2000)
6. Babiak, T., Blahoudek, F., Duret-Lutz, A., Klein, J., Křetínský, J., Müller, D., Parker, D., Strejček, J.: The Hanoi omega-automata format. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 479–486. Springer, Cham (2015). doi:10.1007/978-3-319-21690-4_31
7. Baier, C., Dubslaff, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Probabilistic model checking for energy-utility analysis. In: Breugel, F., Kashefi, E., Palamidessi, C., Rutten, J. (eds.) Horizons of the Mind. A Tribute to Prakash Panangaden. LNCS, vol. 8464, pp. 96–123. Springer, Cham (2014). doi:10.1007/978-3-319-06880-0_5
8. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.: Model-checking algorithms for continuous-time Markov chains. IEEE Trans. Softw. Eng. **29**(6), 524–541 (2003)
9. Baier, C., Katoen, J.-P.: Principles of Model Checking. MIT Press, Cambridge (2008)

10. Baier, C., Klein, J., Klüppelholz, S., Märcker, S.: Computing conditional probabilities in Markovian models efficiently. In: Ábrahám, E., Havelund, K. (eds.) TACAS 2014. LNCS, vol. 8413, pp. 515–530. Springer, Heidelberg (2014). doi:10.1007/978-3-642-54862-8_43
11. Baier, C., Klein, J., Klüppelholz, S., Wunderlich, S.: Maximizing the conditional expected reward for reaching the goal. In: Legay, A., Margaria, T. (eds.) TACAS 2017. LNCS, vol. 10206, pp. 269–285. Springer, Heidelberg (2017). doi:10.1007/978-3-662-54580-5_16
12. Baier, C., Klein, J., Leuschner, L., Parker, D., Wunderlich, S.: Ensuring the reliability of your model checker: interval iteration for Markov decision processes. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10426, pp. 160–180. Springer, Cham (2017). doi:10.1007/978-3-319-63387-9_8
13. Baier, C., Kwiatkowska, M.Z.: Model checking for a probabilistic branching time logic with fairness. Distrib. Comput. **11**(3), 125–155 (1998)
14. Barthe, G., Espitau, T., Ferrer Fioriti, L.M., Hsu, J.: Synthesizing probabilistic invariants via Doop's decomposition. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 43–61. Springer, Cham (2016). doi:10.1007/978-3-319-41528-4_3
15. Besset, D.H.: Object-Oriented Implementation of Numerical Methods: An Introduction with Java and Smalltalk. Morgan Kaufmann Publishers Inc., Burlington (2000)
16. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed.) FSTTCS 1995. LNCS, vol. 1026, pp. 499–513. Springer, Heidelberg (1995). doi:10.1007/3-540-60692-0_70
17. Chatterjee, K., Fu, H., Goharshady, A.K.: Termination analysis of probabilistic programs through Positivstellensatz's. In: Chaudhuri, S., Farzan, A. (eds.) CAV 2016. LNCS, vol. 9779, pp. 3–22. Springer, Cham (2016). doi:10.1007/978-3-319-41528-4_1
18. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J. ACM **42**(4), 857–907 (1995)
19. Dehnert, C., Junges, S., Jansen, N., Corzilius, F., Volk, M., Bruintjes, H., Katoen, J.-P., Ábrahám, E.: PROPhESY: a PRObabilistic ParamEter SYnthesis tool. In: Kroening, D., Păsăreanu, C.S. (eds.) CAV 2015. LNCS, vol. 9206, pp. 214–231. Springer, Cham (2015). doi:10.1007/978-3-319-21690-4_13
20. Dehnert, C., Junges, S., Katoen, J.P., Volk, M.: A storm is coming: a modern probabilistic model checker. In: Majumdar, R., Kunčak, V. (eds.) CAV 2017. LNCS, vol. 10427, pp. 592–600. Springer, Cham (2017). doi:10.1007/978-3-319-63390-9_31
21. Even, S., Goldreich, O., Lempel, A.: A randomized protocol for signing contracts. Commun. ACM **28**(6), 637–647 (1985)
22. Forejt, V., Kwiatkowska, M.Z., Norman, G., Parker, D.: Automated verification techniques for probabilistic systems. In: Bernardo, M., Issarny, V. (eds.) SFM 2011. LNCS, vol. 6659, pp. 53–113. Springer, Heidelberg (2011). doi:10.1007/978-3-642-21455-4_3
23. Gao, Y., Hahn, E.M., Zhan, N., Zhang, L.: CCMC: a conditional CSL model checker for continuous-time Markov chains. In: Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 464–468. Springer, Cham (2013). doi:10.1007/978-3-319-02444-8_36
24. Gao, Y., Xu, M., Zhan, N., Zhang, L.: Model checking conditional CSL for continuous-time Markov chains. Inf. Process. Lett. **113**(1–2), 44–50 (2013)
25. Grädel, E., Thomas, W., Wilke, T. (eds.): Automata Logics, and Infinite Games. LNCS, vol. 2500. Springer, Heidelberg (2002). doi:10.1007/3-540-36387-4

26. Haddad, S., Monmege, B.: Reachability in MDPs: refining convergence of value iteration. In: Ouaknine, J., Potapov, I., Worrell, J. (eds.) RP 2014. LNCS, vol. 8762, pp. 125–137. Springer, Cham (2014). doi:10.1007/978-3-319-11439-2_10
27. Hahn, E.M., Li, Y., Schewe, S., Turrini, A., Zhang, L.: iscasMc: a web-based probabilistic model checker. In: Jones, C., Pihlajasaari, P., Sun, J. (eds.) FM 2014. LNCS, vol. 8442, pp. 312–317. Springer, Cham (2014). doi:10.1007/978-3-319-06410-9_22
28. Hansson, H., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput. **6**, 512–535 (1994)
29. Haverkort, B.R.: Performance of Computer Communication Systems: A Model-Based Approach. Wiley, Hoboken (1998)
30. Heiner, M., Rohr, C., Schwarick, M.: MARCIE – model checking and reachability analysis done efficiently. In: Colom, J.-M., Desel, J. (eds.) PETRI NETS 2013. LNCS, vol. 7927, pp. 389–399. Springer, Heidelberg (2013). doi:10.1007/978-3-642-38697-8_21
31. Helmink, L., Sellink, M.P.A., Vaandrager, F.W.: Proof-checking a data link protocol. In: Barendregt, H., Nipkow, T. (eds.) TYPES 1993. LNCS, vol. 806, pp. 127–165. Springer, Heidelberg (1994). doi:10.1007/3-540-58085-9_75
32. Jansen, N., Kaminski, B.L., Katoen, J., Olmedo, F., Gretz, F., McIver, A.: Conditioning in probabilistic programming. In: Mathematical Foundations of Programming Semantics (MFPS), ENTCS, vol. 319, pp. 199–216 (2015)
33. Ji, M., Wu, D., Chen, Z.: Verification method of conditional probability based on automaton. J. Netw. **8**(6), 1329–1335 (2013)
34. Katoen, J.-P., Zapreev, I.S., Hahn, E.M., Hermanns, H., Jansen, D.N.: The ins and outs of the probabilistic model checker MRMC. Perform. Eval. **68**(2), 90–104 (2011)
35. Kulkarni, V.G.: Modeling and Analysis of Stochastic Systems. Chapman & Hall, Boca Raton (1995)
36. Kwiatkowska, M., Norman, G., Segala, R.: Automated verification of a randomized distributed consensus protocol using cadence SMV and PRISM? In: Berry, G., Comon, H., Finkel, A. (eds.) CAV 2001. LNCS, vol. 2102, pp. 194–206. Springer, Heidelberg (2001). doi:10.1007/3-540-44585-4_17
37. Kwiatkowska, M., Norman, G., Sproston, J.: Probabilistic model checking of the IEEE 802.11 wireless local area network protocol. In: Hermanns, H., Segala, R. (eds.) PAPM-PROBMIV 2002. LNCS, vol. 2399, pp. 169–187. Springer, Heidelberg (2002). doi:10.1007/3-540-45605-8_11
38. Kwiatkowska, M.Z., Norman, G., Parker, D.: Probabilistic symbolic model checking with PRISM: a hybrid approach. Int. J. Softw. Tools Technol. Transf. (STTT) **6**(2), 128–142 (2004)
39. Kwiatkowska, M.Z., Norman, G., Parker, D.: PRISM 4.0: verification of probabilistic real-time systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 585–591. Springer, Heidelberg (2011). doi:10.1007/978-3-642-22110-1_47
40. Kwiatkowska, M.Z., Norman, G., Parker, D.: The PRISM benchmark suite. In: 9th International Conference on Quantitative Evaluation of SysTems (QEST), pp. 203–204. IEEE Computer Society (2012)
41. Märcker, S., Baier, C., Klein, J., Klüppelholz, S.: Computing conditional probabilities: implementation and evaluation (extended version) (2017). http://wwwtcs.inf.tu-dresden.de/ALGI/PUB/SEFM17/
42. Nikishkov, G.P.: Programming Finite Elements in Java™, 1st edn. Springer, London (2010)

43. Norman, G., Shmatikov, V.: Analysis of probabilistic contract signing. In: Abdallah, A.E., Ryan, P., Schneider, S. (eds.) FASec 2002. LNCS, vol. 2629, pp. 81–96. Springer, Heidelberg (2003). doi:10.1007/978-3-540-40981-6_9
44. Puterman, M.L., Processes, M.D.: Discrete Stochastic Dynamic Programming. Wiley, Hoboken (1994)
45. Reiter, M.K., Rubin, A.D.: Crowds: anonymity for web transactions. ACM Trans. Inf. Syst. Secur. (TISSEC) **1**(1), 66–92 (1998)
46. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite-state programs. In: 26th IEEE Symposium on Foundations of Computer Science (FOCS), pp. 327–338. IEEE Computer Society (1985)