The Ins and Outs of The Probabilistic Model Checker MRMC

Joost-Pieter Katoen*, Ivan S. Zapreev†, Ernst Moritz Hahn‡, Holger Hermanns‡, David N. Jansen§

*RWTH Aachen University, 52056 Aachen, Germany

†CWI, 1098 XG Amsterdam, The Netherlands

‡Saarland University, 66123 Saarbrücken, Germany

§Radboud Universiteit, 6500 GL Nijmegen, The Netherlands

Abstract

The Markov Reward Model Checker (MRMC) is a software tool for verifying properties over probabilistic models. It supports PCTL and CSL model checking, and their reward extensions. Distinguishing features of MRMC are its support for computing time- and reward-bounded reachability probabilities, (property-driven) bisimulation minimization, and precise on-the-fly steady-state detection. Recent tool features include time-bounded reachability analysis for uniform CTMDPs and CSL model checking by discrete-event simulation. This paper presents the tool's current status and its implementation details.

1. Introduction

Quantitative performance and dependability analysis of computerised systems is gaining considerable importance in daily life. Ensuring minimum breakdown probabilities of, for instance, components of steer- and brake-by-wire systems in cars, is vital to fulfil safety requirement specifications such as the international standard IEC 61508. Hence, answering timed reachability questions like: "What is the worst case risk to hit a safety-critical system state within 7 days of mission time?" as early as possible in the system design process is becoming indispensable.

One way to ensure that a system design meets its safety requirement specification is to create and analyse a formal model of the envisaged design. Probabilistic model checking is an automated technique to check whether $M \models \varphi$ for a (typically) Markovian model M and a temporal logic formula φ. The advance of time in M can be either continuous or discrete, the choice between probability distributions may be non-deterministic, and rewards may be attached to states and/or transitions. Logics are either linear-time (LTL) in which case the probability of M satisfying φ is computed, or a probabilistic variant of the branching-time logic CTL, such as PCTL and CSL. This spans a wide range of models and logics, each requiring tailored and specific model-checking algorithms.

The Markov Reward Model Checker (MRMC) is an explicit-state verification tool. The core of MRMC is a numerical solution engine that supports the verification of continuous- and discrete-time Markov chains (CTMCs and DTMCs) against PCTL and CSL respectively using the (by now) standard numerical analysis techniques [18], [3]. This core is enriched with the support for verifying reward extensions of CTMCs and DTMCs —so-called Markov reward models (MRMs)— against appropriate PCTL and CSL extensions [43], [1], [2], most notably the verification of time- and reward-bounded reachability probabilities. This allows for verifying properties such as "the probability to reach a bad state within d time units and E units of energy exceeds 0.98".

MRMC is a command-line tool that has an easy-to-grasp input format. The input is on the level of an explicit state space. This facilitates using MRMC as back-end to modelling tools for various modelling formalisms. It also makes the tool a potential testbed for rapid experimentation with novel and improved algorithms, because no special syntactic structure needs to be respected by the algorithm designer. This dedication has found an audience: Since its first appearance in the scientific arena [30], it has been used several times as a model-checking back-end. Some example formalisms are stochastic process algebras [50], [23], Petri nets [14], Statemate [7], and AADL [8].

The intention to provide a testbed for algorithmic advances has turned out to be a fruitful concept. Over the past few years, the core engine was extended with several distinguishing features: (1) To alleviate the state-space explosion problem, MRMC supports bisimulation minimization of the input model. The minimization can be tailored to the formula under consideration to obtain smaller quotients. Furthermore, (2) MRMC has been extended with support for CTMDPs, a variant of CTMCs with non-determinism, that can be regarded as the common semantic model of many specification formalisms for concurrent stochastic systems. MRMC implements an efficient algorithm to compute maximal time-bounded reachability probabilities in uniform CTMDPs [4]. Finally, (3) MRMC

This research was performed as part of the MC=MC project financed by the Netherlands Organization for Scientific Research (NWO) and the DFG Research Training Group 623 on Leistungsgarantien für Rechnersysteme. We thank Maneesh Khattri (Oxford Univ.), Christina Jansen (RWTH Aachen), and Tim Kemna (Univ. Twente) for their implementation efforts.

includes a discrete-event simulation engine for simulative CSL model checking [55]. MRMC is thus the first tool with simulation-based model checking of full CSL.

MRMC is realised in the C programming language, which allows the tool to be small and fast due to compiler-based optimisations and smart memory management within the implementation. To store the state space, it uses a slightly modified version of the well-known compressed-row, compressed-column representation of probability (rate) matrices that is tailored to fast matrix-vector multiplication. It supports all major platforms (Microsoft Windows, Linux and Mac OS X), is distributed under the GNU General Public License (GPL) [17], and is available for free download at [37].

This paper gives an overview of the current tool status, with a particular focus on the recent extensions. We describe MRMC v1.4.1 as it will be released in August 2009.

Other Probabilistic Model Checkers. In the last years, model checking probabilistic systems has been an active research field. This has resulted in a whole variety of probabilistic model checkers, such as APMC [21], FHP-Murphi [40], Liquor [12], PASS [24], PRISM [25], VESTA [45], and Ymer [51]. Probabilistic model checking facilities are also present in the stochastic Petri nets tools GreatSPN [6] the APNN Toolbox [9]. Some of these tools aim at specific models, logics and model-checking techniques. For example, FHP-Murphi is tailored to finite horizon safety properties, APMC uses Monte-Carlo techniques for bounded-model checking of Markov chains, and PASS exploits predicate abstraction and counterexample guided refinement for verifying probabilistic programs. Ymer implements statistical CSL model-checking techniques based on discrete event simulation [46] and sequential acceptance sampling [53]. It also incorporates simple acceptance sampling and adopted a numerical engine from PRISM. VESTA allows to verify CSL (PCTL) properties on CTMC (DTMC) models. The tool implements model-checking techniques (based on simple hypothesis testing [26]) discussed in [53] and [45]. PRISM —the most popular and advanced tool in the field— allows for numerical model checking of PCTL and CSL. It also supports the verification of expected reward measures and has some facilities for discrete-event simulation. Unlike MRMC, PRISM is a symbolic model checker using multi-terminal BDDs for representing Markov models and in contrast to MRMC supports the verification of discrete-time MDPs. Finally, we mention SMART as a related tool which supports the simulative and symbolic numerical analysis of CTMCs modelled as Petri nets, as well as CTL model-checking of such models [11]. SMART is using multi-decision diagrams (MDDs) rather than MTBDDs.

Organization of The Paper. Section 2 outlines the supported models and logical formalisms. Section 3 presents the input formats accepted by MRMC and, by means of examples, introduces the tool's interactive shell. The recent features of MRMC are thoroughly discussed in Section 4. This includes: *(i)* time-bounded reachability for CTMDPs, *(ii)* complete support for simulation-based model checking of CSL, *(iii)* using (strong) bisimulation minimization prior to model-checking, and *(iv)* precise steady-state detection for time-bounded reachability problems. Section 5 presents implementation details, such as architectural solutions and some source code metrics. Section 6 concludes.

2. Probabilistic Verification with MRMC

2.1. Models and Logics

MRMC is aimed at performance and dependability evaluation and has therefore been focused on verifying discrete- and continuous-time Markov chains (DTMCs and CTMCs) and rewards extensions thereof (DMRMs and CMRMs, repectively). It supports the logics:

model	DTMC	CTMC	DMRM	CMRM	uCTMDP
logic	PCTL	CSL	PRCTL	CSRL	time-bounded reachability

The key ingredient of these logics is the probabilistic operator, denoted $\mathbb{P}$; the formula $\mathbb{P}_{\leq \frac{1}{2}}(\Diamond \Phi)$ holds in state s whenever the total probability mass of all paths that start in s and eventually reach some Φ-state is at most $\frac{1}{2}$. (Here, it is assumed that Φ characterizes a set of states.) For DTMCs, the verification of such properties reduces to solving a system of linear equations with a variable for each state. For CTMCs, the time until reaching a Φ-state can be indicated (as parameter of the $\Diamond$ modality), and verification reduces to solving a set of Volterra integral equations, or, equivalently, to a graph transformation followed by a standard transient CTMC analysis. These properties are complemented with long-run properties.

In DMRMs, states are equipped with costs, called rewards, which are incurred in leaving (or entering) a state. Key formulae in PRCTL are besides the expected accumulated reward in Φ-states, e. g., $\mathbb{P}_{\leq \frac{1}{2}}(\Diamond_{\leq c}\, \Phi)$ which restricts attention to those paths that reach a Φ-state within a total cost of at most c. In a similar way, CSRL allows to request reachability within a time- and a reward-bound. Note that in CMRMs the incurred reward in a state is proportional to the reward associated to the state and the state residence time. Finally, MRMC supports a sub-class of CTMCs with non-determinism, viz. so-called CTMDPs. Although it does not cover a full logic, it can compute maximal probabilities for time-bounded reachability.

2.2. State-Space Representation

Storing a Markov chain may be quite a challenge, since most real-life models are represented by chains with millions of states and transitions. Fortunately, most transition matrices that appear in probabilistic model checking have a very sparse structure, i. e., contain a large number of zeroes. Therefore using sparse matrices, such

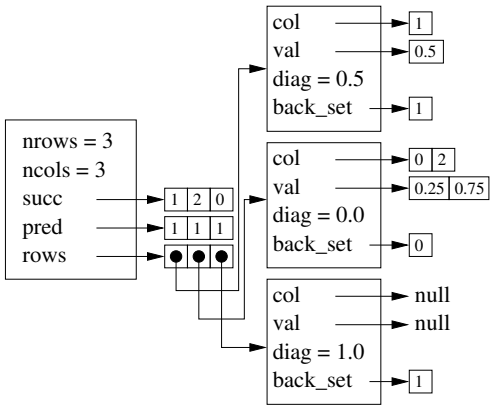


Figure 1: An example of the sparse matrix representation used in MRMC

as a compressed-row (compressed-column) representation (see [41] for more details), as a data structure for probability (rate) matrices is advantageous. These structures allow to avoid the storage of, and computation on, a large number of zeroes while keeping the manipulations of data relatively cheap.

For MRMC, as recommended in [48], we have chosen the *compressed-row representation* because it assures a high efficiency of matrix–vector multiplications which are at the core of numerical model checking. Similar data structures were implemented in the (by 2004) fastest serial and parallel explicit Markov chain solver [5].

In our implementation a sparse matrix is represented by a structure containing a number of rows $\mathbf{nrows}$; the number of columns $\mathbf{ncols}$; an array that stores the number of non-zero off-diagonal elements for each row $\mathbf{succ}$; an array of pointers to the structure representing a matrix row $\mathbf{rows}$; and an array $\mathbf{pred}$, that contains the number of non-zero off-diagonal elements for each column. Row and column indices start at 0. Note that self-loops are not taken into account by $\mathbf{succ}$ nor $\mathbf{pred}$.

The row structure for row i has several fields, namely the diagonal element $\mathbf{diag}$; an array of non-zero off-diagonal values $\mathbf{val}$; an array of corresponding column indexes $\mathbf{col}$; and an array $\mathbf{back\_set}$, that contains the row indices of non-zero off-diagonal elements in column i. The $\mathbf{back\_set}$ array is used for bisimulation minimization and in the model-checking algorithms of PCTL and CSL.

Example 1: Consider the matrix $\mathbf{P}$:

$$\mathbf{P} = \begin{bmatrix} 0.50 & 0.50 & 0.00 \\ 0.25 & 0.00 & 0.75 \\ 0.00 & 0.00 & 1.00 \end{bmatrix}$$

Figure 1 shows the data structures that are allocated for matrix $\mathbf{P}$ in MRMC. The matrix structure (on the left of the figure) has the number of columns and rows set to 3. Its $\mathbf{succ}$ array contains values 1, 2 and 0 because row zero has only one non-zero off-diagonal element 0.5, row one has two elements 0.25 and 0.75, and row two has none. The $\mathbf{pred}$ array contains ones because each column has one non-zero element (beneath the diagonal).

The array element $\mathbf{rows}[0]$ stores the pointer to the structure representing row 0. This row structure has the following fields: the array $\mathbf{col}$ contains one element (column index 1) because the only non-zero off-diagonal element of the row is $\mathbf{P}_{01}$. The value of this element is stored in the corresponding element of the array $\mathbf{val}$. The field $\mathbf{diag}$ is set to 0.5, the diagonal element $\mathbf{P}_{00}$. The array $\mathbf{back\_set}$ contains the row index 1, because $\mathbf{P}_{10}$ is non-zero.

An advantage of the compressed-row representation is that it gives an easy access to the matrix rows. The latter is crucial for the efficiency of matrix–vector multiplications, which are at the heart of the numerical model checking. Storing rows separately simplifies the procedure of making states absorbing. The fact that the matrix diagonal elements are stored separately from the non-diagonal elements facilitates optimisations of matrix transformations, such as computation of an embedded Markov chain.

2.3. Model Checking Markov Chains

The algorithms for PCTL model checking that have been realized in MRMC are given in [18]. Prior to checking an until-formula, a graph analysis identifies the states from which the goal state is unreachable to reduce the number of variables in the linear equation system. MRMC extends PCTL [18] with a long-run operator, which is checked by a combination of graph analysis (to find bottom strongly connected components [BSCCs]), reachability probabilities (of these BSCCs), and solving a system of linear equations (one per BSCC). This recipe is very similar to the treatment of the steady-state operator in CSL. MRMC uses the iterative Gauss–Seidel and Gauss–Jacobi techniques for solving systems of linear equations [48]. The PRCTL algorithms in MRMC are described in [1]. CSL model-checking techniques in MRMC stem from [3] and for its reward extension CSRL from [2], [13], [20]. For time- and reward-bounded until formulae we have implemented two algorithms: one based on discretization [49] and one based on uniformization and path truncation [43]. The algorithms for PRCTL and CSRL support both state and impulse rewards, i. e., instantaneous rewards on edges. The algorithm for maximal probabilities for time-bounded reachability on CTMDPs has been given in [4].

3. MRMC: Look and Feel

The MRMC Input Format. Consider a die with four faces numbered 1 through 4. The die is biased such that the outcomes are obtained with probability 0.4, 0.3, 0.2, and 0.1, respectively. The DMRM in Fig. 2 consists of five states where state 0 represents the toss and states 1 through 4 the possible outcomes. The state rewards, indicated as state labels between square brackets, model the gains obtained. In a game, the die is tossed repeatedly. The dice game is won if the outcome is 4 (proposition $goal$) and the gain, i. e., the accumulated reward is between 5 and 50, and lost once the outcome is 1 (proposition $loss$).

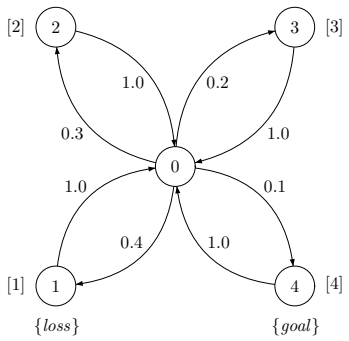


Figure 2: A DMRM model of a simple dice game

Table 1 lists the MRMC input files to describe the DMRM. The file `game.tra` lists the state transitions with their probability; the file `game.lab` contains label declarations and provides the state-labeling; finally, the file `game.rew` contains the state rewards. Possible impulse rewards (not present in the example) would be provided in a separate *filename*`.rewi` file.

`game.tra`	`game.lab`	`game.rew`
STATES 5	#DECLARATION	1 1
TRANSITIONS 8	loss goal	2 2
0 1 0.4	#END	3 3
0 2 0.3	1 loss	4 4
0 3 0.2	4 goal	
0 4 0.1		
1 0 1.0		
2 0 1.0		
3 0 1.0		
4 0 1.0		

Table 1: MRMC input files for the dice game

The MRMC Interactive Shell. MRMC is a command line tool that provides a shell-like environment (a command prompt) where a user can specify the tool run-time options such as the use of specific numerical algorithms or certain run-time parameters, and the properties to be verified. Upon startup, MRMC accepts several command-line options, e. g., that specify the model. In order to start MRMC with the aforementioned input files, the following command should be issued:

```
MRMC/bin> ./mrmc dmrm game.tra
game.lab game.rew
```

Subsequently, MRMC outputs its logo, some general information about the accepted model (like memory consumption) and the prompt >> signalling that the tool is up and running, ready to accept user commands. For every verification problem the tool lists the states satisfying the given property and, if applicable, the probability of the required path or state formula. Let's consider a small example. Suppose we want to know the answer to: "Is the probability to win this game within 100 tosses, larger than 0.5?" We can ask MRMC by entering a PRCTL formula:

```
>> P>0.5[ !loss U[0,199][5,50] goal]
```

where it should be noted that 100 tosses correspond to 199 time steps. MRMC then outputs:

```
$RESULT: ( 0.0647999, 0.0000000,
0.0959998, 0.1199998, 0.1199997 )
```

where the probability vector `$RESULT` indicates the likelihood of state i satisfying $\neg loss\ \mathcal{U}^{[0,199]}_{[5,50]}\ goal$, and

```
$STATE: {  }
The model-checking time is 45 milli
sec(s).
>>
```

where `$STATE` is the set of states satisfying the formula. As for each state the likelihood is ≤ 0.5, it is empty.

4. MRMC Recent Features

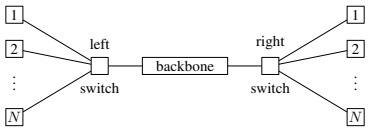
4.1. Timed Reachability in CTMDPs

A continuous-time Markov decision process (CTMDP) extends CTMCs with non-deterministic choices. As for CTMCs, the model consists of states, and the timed behaviour is governed by exponential distributions. But different from CTMCs, each state may feature a number of non-deterministic decisions of next-step distributions. The class of CTMDPs is of interest, because it can be viewed as a common semantic model for various performance and dependability modelling formalisms including generalized stochastic Petri nets [10], Markovian stochastic activity networks [44], and interactive Markov chains [22]. So far, the analysis of models developed in these and related formalisms was restricted to the subset that corresponds to CTMCs, usually referred to as "non-confused", "well-defined", or "well-specified".

Non-deterministic decisions are decisions we cannot actually connect a probability distribution with, as it is unknown or not applicable. Such choices may result from underspecification of the model, or by leaving out probabilities we do not have enough information about, like user actions or certain environmental influences. Usually, labels are used to distinguish the non-deterministic alternatives, and this is indeed also the case in the model supported by MRMC. However, MRMC also supports models where there is internal non-determinism between equally labelled next steps. In summary, a CTMDP specification consists of state transitions, corresponding distributions, and a labelling function that maps transitions to labels.

As we have non-deterministic decisions, we cannot talk about *the* probability of a property of the model. Instead, probabilities result from choosing an instance which resolves the non-deterministic decisions, a *scheduler*. The tool supports the computation of the maximum probability with which a set of target states can be reached within a given time bound. For this it assumes (but does not check) that the model is *uniform*, which means that there is a unique rate E such that for each state and each non-deterministic alternative, the total outgoing rate of this alternative is E.

Algorithmic Details. MRMC implements an algorithm that efficiently calculates time-bounded reachability probabilities in uniform CTMDPs [4]. The schedulers considered



	fail	repair
workstation	500 h	0.5 h
switch	4000 h	4 h
backbone	5000 h	8 h

Figure 3: FTWC with mean fail and repair times

are *history dependent*, i. e. they have information about all previous states visited and transitions taken, but they do not have information about the exact point of time a state is entered. To compute the maximum it is enough consider schedulers which only know the *length* of the history. Algorithmically, a greedy backwards algorithm is used to achieve this, and this is what is integrated in MRMC.

Experimental Results. As an example application, we show results obtained for a fault-tolerant workstation cluster (FTWC), originally studied in [19]. The general design of the workstation cluster is shown on the left-hand side of Fig. 3. It consists of two sub-clusters which are connected via a backbone. There are N workstations in each sub-cluster which are connected together in a star-topology with a switch as central node. The switches provide additionally the interface to the backbone. Each of the components in the fault-tolerant workstation cluster can break down (fail) and then needs to be repaired before becoming available again. The mean time to failure and the mean repair time for each component in isolation are depicted on the right-hand side of Fig. 3. They correspond to mean durations of exponential distributions. There is a single repair unit for the entire cluster, not depicted in the figure, which is only capable of repairing one failed component at a time. Essentially, this means that when multiple components are down, they must be handled in sequence, and there is a non-deterministic decision to be taken which of the failed components the repair unit is assigned to first (or next).

To generate the CTMDP representing the overall system we used the IMC calculus as supported by the CADP toolset; details are explained in [28]. As in [19] we say that our system operates in premium quality when at least N workstations are operational. These workstations have to be connected to each other via operational switches. When the number of operational workstations in one sub-cluster is below N, premium quality can be ensured by an operational backbone under the condition that there are N operational workstations in total. We are interested in the following property: *"What is the worst case probability to hit a state in which premium service is not guaranteed within t time units?"* for which we report results and statistics in Table 2. The first column depicts the number of workstations on each side, the next two columns show the overall memory required and the number of states of the CTMDP. The remaining columns show the results of the computation for time bounds ranging from 100h to 50000h. For each N we display the computation time in the upper part of the row and the computed probability in the lower part.

N	memory	states	100 h	1000 h	5000 h	10000 h	50000 h	
1	4.12 MB	110	0s 0.00	0s 0.01	0s 0.04	0s 0.09	2s 0.36	time prob.
4	4.26 MB	818	0s 0.00	0s 0.02	2s 0.09	3s 0.18	15s 0.62	time prob.
16	5.25 MB	10130	0s 0.01	2s 0.08	10s 0.32	19s 0.54	1m 34s 0.98	time prob.
64	22.16 MB	151058	6s 0.03	38s 0.23	3m 6s 0.73	5m 41s 0.93	28m 0s 1.00	time prob.
256	288.31 MB	2373650	1m 39s 0.05	12m 33s 0.43	59m 11s 0.94	118m 22s 1.00	642m 3s 1.00	time prob.

Table 2: Statistics for the FTWC analysis

As we can see, the CTMDP algorithm performs well even for examples with very large state spaces: The model in the last row has about 2 million states and about 20 million nonzero matrix entries. This holds also for other examples, see [28].

The inherent non-determinism in the specification has been ignored in the original model [19] and in subsequent work, e. g., in the FTWC model used and generated by the model checker PRISM [42]. There, the non-deterministic decision of which system to repair if several systems failed has been modelled using a very fast, but probabilistic decision, encoded via the use of very high rates (of exponential distributions) assigned to the decisive transitions. These high rates are absent in the original problem statement where the repair unit is assigned non-deterministically.

Remarkably, this modelling trick results in an overestimation of the true probabilities (obtained via the CTMC engine of MRMC, reconfirmed with PRISM). This is quite surprising, because the CTMDP algorithm accounts for the worst-case. Nothing worse is possible in the model, and we would thus expect, that this probability will be higher than in any corresponding CTMC model of the system. This overestimation, which indicates a modelling flaw in the CTMC approach, can be explained as follows. When replacing a non-deterministic selection by high rates, certain paths become possible (though with low probability), that in a non-deterministic interpretation would be absent, and thus not contribute to the reachability probability. For a more detailed explanation of this phenomenon we refer the interested reader to [28].

4.2. Simulation-Based CTMC Model Checking

The numerical analysis algorithms in MRMC have recently been complemented by *simulation-based* model-checking algorithms for CSL. As opposed to the approaches taken by Ymer [51], [54] and VESTA [45] that exploit statistical hypothesis testing, MRMC uses classical discrete-event simulation (DES) techniques for CTMCs. In essence, we exploit continuous-time terminating simulations for time-bounded until formulae (where the termination conditions are naturally induced by the formula), standard regenerative simulation on embedded DTMCs for steady-state formulae, and terminating simulations on

embedded DTMCs for unbounded until-formulae (where reaching a certain number of steps acts as terminating condition; see also below). In the current version, it is assumed that the structure of the CTMC at hand —in fact, its BSCCs— is known prior to the simulation, i. e., it is not a fully on-the-fly simulation engine. In a next version of MRMC, it is planned to obtain this during state-space generation from AADL specifications. Full details of the simulation algorithms are provided in [55, Ch. 6].

Basic Strategy. To check whether e. g., $s \models \mathbb{P}_{>b}(\varphi)$, an estimate $\widetilde{p}$ of the probability mass p of all φ-paths starting in s is determined using standard DES techniques. Let ξ be the user-specified confidence of the result and δ' the maximum width of the confidence interval. The probability of obtaining a correct answer to the model-checking problem $s \models \mathbb{P}_{>b}(\varphi)$ is now guaranteed to be at least ξ provided $\delta' \leq |b - \widetilde{p}|$.

Confidence Intervals. A slight adaptation of standard sequential confidence intervals is exploited in which the sample size and simulation depth can be adapted on demand. In combination with the Agresti–Coull confidence intervals for Bernoulli trials, quite accurate results are obtained. We illustrate this by means of the CPS case study where we check $\mathbb{P}_{\geq 0.99}\left(\Diamond^{[40,80]}\, serve_1\right)$ where the proposition $serve_i$ uniquely identifies a state in which station i is being served. For $N \in \{6, 9\}$, where N is the number of stations, $p = 0.9\underline{9}88$ and $p = 0.9\underline{8}88$, respectively, which is very close to the bound $b = 0.999$. This leads (cf. Fig. 4) to a confidence significantly below the user-defined $\xi = 0.95$ using Ymer and Ymer P —the variant of Ymer that uses sequential confidence-interval approach based on [39]. Although $\delta' > |b - \widetilde{p}|$, MRMC provides more accurate answers as its algorithm first simulates until the confidence interval is tighter than δ' and then continues simulation until it reaches the definite answer to the model-checking problem. This strategy increases the accuracy because the width of the resulting confidence interval can be much smaller than δ'. (Note that VESTA does not support intervals like $[40, 80]$ as time bounds.) The penalty for this increased accuracy is an increase in the sample size, i. e., the number of states that are visited during the simulation (cf. Fig. 5), and, accordingly, yields larger model-checking times. Despite the larger number of required states than Ymer P, MRMC is about a factor 8 times faster. Ymer needs less states for larger N since $b - \tilde{p}$ becomes larger in that case, and thus Ymer can decide the validity of the formula with less samples. Our experiments with other cases showed that for (time-bounded) until formulae the peak-memory consumption (VSZ) of MRMC is linearly proportional to the model size.

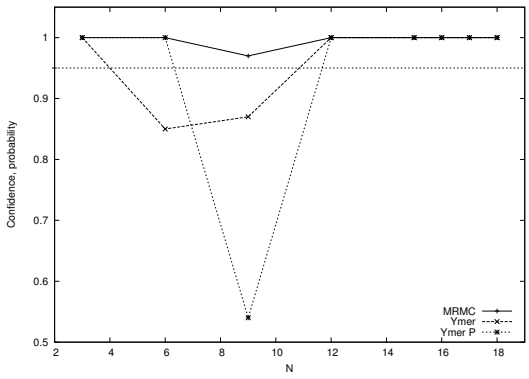


Figure 4: Confidence levels for $\mathbb{P}_{\geq 0.99}\left(\Diamond^{[40,80]}\, serve_1\right)$

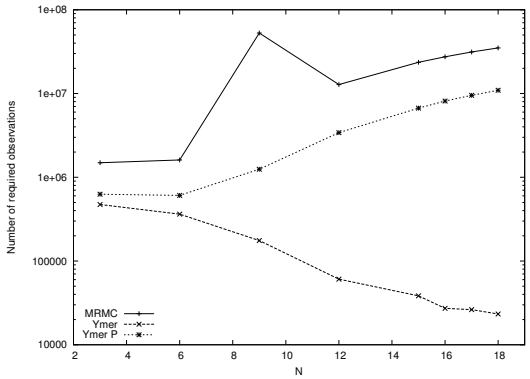


Figure 5: # observations for checking $\mathbb{P}_{\geq 0.99}\left(\Diamond^{[40,80]}\, serve_1\right)$

Simple Reachability Properties. For unbounded conditional reachability properties, MRMC uses two independent samples so as to ensure the correct confidence level. In contrast to VESTA that uses a so-called "stopping" probability to guess whether from a state a goal state is never reached, MRMC uses the CTMC's graph structure to rule out states that lie in BSCCs. Then the simulation boils down to estimating the stationary probabilities to be in a goal state, which can be estimated by the state probabilities of transient states. This yields a drastic reduction of the needed number of simulated states, cf. Fig. 6. In fact, we simulate until the N-th epoch, increasing it along with increasing the sample size M (alternating the increase of M and N). The confidence intervals are formed from the confidence interval for the probability to be in a good absorbing state (i. e., a goal state) and the probabilities to be in a good absorbing or transient state, at epoch N.

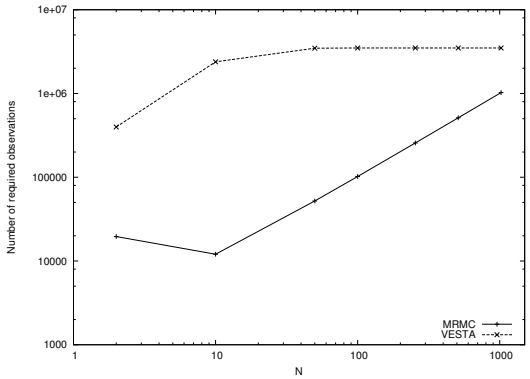


Figure 6: Sample sizes for checking $\mathbb{P}_{\leq 0.03}\left(\neg full_1\ \mathcal{U}\ full_2\right)$

Checking Steady-State Formulae. For steady-state formulae, the probability estimate $\tilde{p}$ is based on combining estimates of stationary probabilities in the BSCCs and estimates for the simple reachability probabilities of these BSCCs. The latter ones are obtained by regeneration-based simulation of the embedded DTMC as originally proposed in [27]. This approach is justified by the fact that only the exit rates of the CTMC are of relevance for obtaining a point estimate and confidence interval. For real function f on the state space and $\alpha = E[f(X)]$ where X is the random variable describing the CTMC in equilibrium, the point estimate for α is obtained by simply dividing the expected accumulated value of f along all regeneration cycles by the expected duration of such cycles. In order to select a regeneration point, MRMC offers two possibilities: a deterministic choice, or the use of a simple heuristic where it is chosen as the most recurring state in a test run preceding the verification. Experiments have shown that for ergodic Markov chains, model checking results could be obtained in a matter of seconds using the heuristic, whereas a fixed choice of regeneration point did not yield results within 15 minutes. MRMC also offers the choice between dynamic (D) and constant (C) sample-sizes increase. The latter allows to improve tool's performance on smaller models, cf. Fig. 7. Neither Ymer nor VESTA provide support for steady-

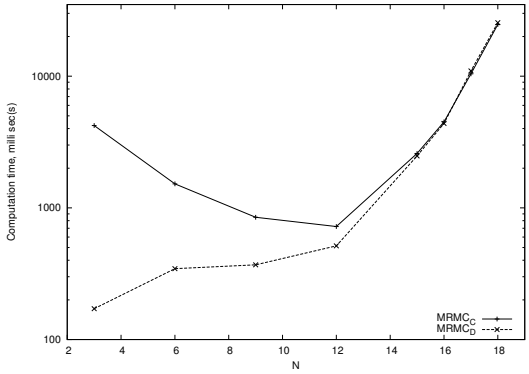


Figure 7: Model-checking times for $\mathbb{S}_{>0.19}\,(busy_1)$ (time)

state properties. Our experiments revealed that the memory consumption for simulating steady-state formulae is much higher (about a factor 5) than for until-formula due to the storage of samples. In addition, computing confidence intervals requires much more effort as estimates are conditional probabilities. For efficiency reasons, we exclude the computation of confidence intervals for BSCCs that are almost surely (non-)reachable. A more extensive empirical evaluation of MRMC's simulation engine can be found in [32].

4.3. Bisimulation Minimization

Bisimulation minimization is a (by now) standard method to reduce the size of a Markov chain while preserving interesting properties such as the validity of full CSL. Basically, states that exhibit the same probabilistic behaviour are defined to be equivalent, and one takes the quotient under this equivalence relation. The appealing feature of this abstraction technique is that it is fully automated. MRMC offers bisimulation minimization for PCTL, CSL, PRCTL and CSRL models (in the latter two cases without impulse rewards).

Algorithmic Details. MRMC implements the time-optimal partition refinement algorithm of [16]. The main step in partition refinement is *splitting*. Let Π be a partition of the state space S. A splitter for some tentative equivalence class $B \in \Pi$ is a set of states $Sp \in \Pi$ such that the probability to enter Sp is not the same for each state in B. As long as splitters exist, the algorithm splits B into subclasses such that each subclass consists of states s with identical $\mathbf{P}(s, Sp)$. This step is repeated until a fixpoint is reached. The final partition is the coarsest bisimulation that respects the initial state space partition.

Even smaller quotients can be obtained when tailoring the initial partition to the CSL/PCTL formula to be checked. Whereas ordinary bisimulation groups states with equal atomic propositions, property-driven bisimulation starts with a partition that only preserves the immediate subformulas of the property at hand (3–4 subsets).

Data Structures. The worst-case time complexity of this algorithm is $O(|\mathbf{P}| \log |S|)$ if one uses splay trees to store the subclasses. A splay tree is a self-balancing binary search tree with the additional property that recently accessed elements are quick to access again [47]. One basically has to sort a multiset, i. e., a set where the same transition probability $\mathbf{P}(s, Sp)$ may appear more than once [38], and splay trees serve to implement such a sort algorithm. With a general sort algorithm, the time complexity raises to $O(|\mathbf{P}| \log^2 |S|)$. In [15], it is suggested that red–black trees for sorting may be more efficient in practice, but our experiments have shown that this is not the case in general. This may be due to the much more complex implementation of the data structure and operations on it. MRMC therefore uses an efficient implementation of splay trees developed by Sleator. We recently experimented with other sort algorithms: heapsort and quicksort can be adapted to multisets. (Our implementation of) heapsort is approximately as fast as splay trees, and with quicksort we could accelerate the average runtime of MRMC by a factor of 1.3–2.5 for larger models.

Experimental Results. Traditional bisimulation minimization often leads to drastic (up to exponential) decreases in state-space size, but mostly comes at a time penalty, i. e., CTL and LTL model checking of the original model is faster than first minimizing and verifying the quotient. However, the time-complexity of probabilistic model checking is such that it often is advantageous to calculate the bisimulation quotient first. Extensive experiments with a wide range of benchmark case studies —with and without state rewards— has shown the practical usability of bisimulation minimization; for full details see [29].

We often obtain significant state-space reductions, and in most cases a speed-up is achieved. For DTMCs and CTMCs, the total runtime is reduced by a factor 2–10, with extremes of up to 60 (for the CPS with $N = 15$ stations and an unbounded-until formula). Randomized mutual exclusion (RME) could be checked about 3 times faster without bisimulation minimization. (Using quicksort instead of splay trees did not improve the runtime very much here.) Because checking models with rewards is rather time-consuming, astronomical savings are achieved there. The time reduction strongly depends on the number of transitions in the Markov chain, its structure, as well as on the convergence rate of numerical computations.[1]

Property-driven bisimulation pays off in even more cases: e. g., in the RME case study, it is 1–1.7 times faster than verifying the original model. For property-driven bisimulation on models without rewards, the speedup comes with almost no memory penalty: the peak memory use may be increased by up to 8%, although typically it is reduced or remains unchanged; for ordinary bisimulation some experiments showed an increase of peak memory up to 50%. In experiments with MRMs, we experienced a 21–46% reduction in peak memory use.

Experiments with the P2P case study showed that bisimulation minimization leads to a significantly stronger state-space reduction than symmetry reduction [33]. Symmetry reduction is —as expected— much faster than bisimulation minimization as it operates on a syntatic level, but this is a somewhat unfair comparison as the symmetries were indicated manually. These results suggest that it is affordable to first apply a (fast) symmetry reduction, followed by a bisimulation quotienting on the obtained reduced system.

4.4. On-The-Fly Steady-State Detection

Since verification of time-bounded reachability properties of CTMCs reduces to transient analysis, it is common practice to use —especially for large time spans— on-the-fly steady-state detection, cf. [35], [52]. The idea behind this technique is to save expensive iteration steps by detecting that the CTMC has reached its equilibrium before the end of the time bound. Most probabilistic model checkers adopt this technique *as is,* thus suffering from a possible *premature* steady-state detection, e. g., when the CTMC moves at a very slow rate. This is because for transient analysis there are no *sufficient* and verifiable convergence criteria, cf. [48]. MRMC is the only tool that incorporates *precise* steady-state detection, cf. [31]. This technique is based on the fact that, when checking for time-bounded reachability, the original CTMC is made absorbing. Its state space is then split into transient states that are neutral, and absorbing states that either satisfy or violate the property. Then, the convergence is determined by the probability mass still residing in the transient states.

1. The experiments were performed using MRMC v1.2 released in November 2006; since then the bisimulation engine has not changed.

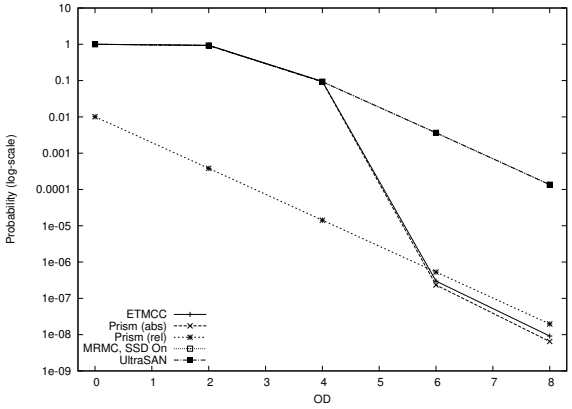


Figure 8: Premature steady-state detection

Let us consider the verification of a variant of the centralized medium access protocol of the IEEE 802.11 standard, for which Massink *et. al.* reported the premature steady-state detection in [36]. As in that work, we determine the probability that a message originating from the access point is not received by at least one station within the duration of the time-critical phase, i. e., $t = 2.4$ seconds. This time span is extremely large, compared to the duration of each protocol's operation. Figure 8 indicates that the results of MRMC v1.4.1, coincide with the (exact) values computed by UltraSAN [44], whereas PRISM v3.2 (absolute and relative criteria), and E⊢MC^2 v1.0 suffer from premature steady-state detection. The parameter OD on the x-axis stands for the omission degree of the protocol and denotes the maximal number of consecutive losses of the same message. This parameter determines the size of the state space. Note that, the protocol's model and its parameters were adopted from [36], the tools' options are as in Section 3.5 of [55].

To summarize, the steady-state detection of MRMC is precise and does not change the model-check time complexity. For runtime, it requires to store and compute one extra probability vector. The verification times, prior to steady-state, (roughly) double. If the equilibrium is reached at time t', the steady-state detection will reduce verification times for the properties with time spans $t \geq 2 \cdot t'$.

5. MRMC Architecture and Implementation

An overview of the tool architecture of MRMC is given in Fig. 9. Its main components are:

Options analyzer: is responsible for parsing the command-line options of MRMC. It invokes reading of the input files and sets the run-time parameters of the tool, such as the logic and the use of (property-driven) bisimulation minimization.

Runtime settings: stores the run-time settings of MRMC, e. g., the error bounds, the maximum number of iterations for the numerical methods.

Input-file reader: is responsible for reading the `.tra`,

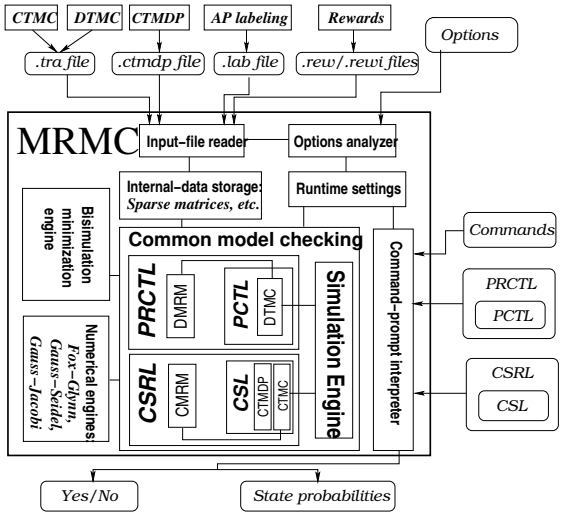


Figure 9: Tool architecture of MRMC

`.lab`, `.rew` and `.rewi` files that specify the input model.
Internal data storage: contains implementations of data structures used in MRMC, such as: sparse matrix, a bit set, structures for storing state labels, and splay trees used in bisimulation minimization.
Command-prompt interpreter: is based on `yacc` and `lex`, and is responsible for: interpreting the MRMC shell commands (such as setting error bounds, desired numerical methods) and formulas, controlling the bottom-up recursive descent over the formula, and printing the results.
Common model checking: contains a set of generally-used algorithms applied in model checking, e. g., procedures for searching BSCCs, and steers the model checking.
Bisimulation engine: provides lumping algorithms.
Numerical engines: implementations of numerical methods for computing Poisson probabilities and iterative methods for solving systems of linear equations.
Simulation engine: DES-based CTMC model checking.

According to *CCCC* (v3.1.4) [34], MRMC v1.4.1 contains about 12,000 non-blank, non-comment lines of source code, and around 15,500 lines of comments. Its McCabe's cyclomatic complexity (the number of linearly independent routes through the control flow graph) is 2,354. Note that values exceeding 50 are considered to indicate very complex programs. MRMC is constantly tested by an automated test-suite [37]. The latter includes most of the case studies discussed in this paper.

6. Conclusions

This paper presented the current status of the model checker MRMC, a tool for the automated verification of Markov reward models. The main contributions of this paper are: a first report on the internals of MRMC, and a more extensive description of the recently added features such as checking time-bounded reachability properties in (uniform) CTMDPs, simulation-based CTMC model checking, bisimulation minimization, and on-the-fly steady-state detection.

The main strength of MRMC is its easy use as a back-end component. It is being used as back-end of the performance modeling tools GreatSPN v2.0 [10], the PEPA Workbench [50], and in a tool chain STATEMATE [7]. PRISM also has an output facility to generate MRMC input files. Currently, it is used together with the model checker NuSMV in the ESA-project COMPASS [8].

References

[1] S. Andova, H. Hermanns, and J.-P. Katoen, "Discrete-time rewards model-checked," in *Formal Modeling and Analysis of Timed Systems (FORMATS)*, ser. LNCS, vol. 2791. Berlin: Springer, 2003, pp. 88–104.

[2] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "On the logical characterisation of performability properties," in *Automata, Languages, and Programming (ICALP)*, ser. LNCS, no. 1853. Berlin: Springer, 2000, pp. 780–792.

[3] ——, "Model-checking algorithms for continuous-time Markov chains," *IEEE Trans. Software Eng.*, vol. 29, no. 6, pp. 524–541, 2003.

[4] C. Baier, H. Hermanns, J.-P. Katoen, and B. R. Haverkort, "Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes," *Theor. Comp. Sc.*, vol. 345, no. 1, pp. 2–26, 2005.

[5] A. Bell, "Distributed evaluation of stochastic Petri nets," Ph.D. dissertation, RWTH Aachen Univ., Germany, 2004.

[6] S. Bernardi, S. Donatelli, and A. Horváth, "Compositionality in the GreatSPN tool and its application to the modelling of industrial applications," in *Practical Use of High-level Petri Nets*, no. PB–547. Univ. of Aarhus, Dept. Computer Science, 2000, pp. 127–146.

[7] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, R. Wimmer, and B. Becker, "Compositional performability evaluation for STATEMATE," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos, CA: IEEE CS, 2006, pp. 167–178.

[8] M. Bozzano, A. Cimatti, J.-P. Katoen, V. Y. Nguyen, T. Noll, and M. Roveri, "The COMPASS approach: Correctness, modelling and performability of aerospace systems," in *Computer Safety, Reliability and Security: 28th int'l conf., SAFECOMP*, ser. LNCS, 2009 (to appear).

[9] P. Buchholz, M. Fischer, P. Kemper, and C. Tepper, "Model checking of CTMCs and discrete event simulation integrated in the APNN-toolbox," in *Measurement, Modelling, and Evaluation of Computer-Comm. Systems*, vol. 781. Fachbereich Informatik, Univ. Dortmund, 2003, pp. 30–33.

[10] D. Cerotti, D. D'Aprile, S. Donatelli, and J. Sproston, "Verifying stochastic well-formed nets with CSL model-checking tools," in *Application of Concurrency to System Design (ACSD)*. Los Alamitos, CA: IEEE CS, 2006, pp. 143–152.

[11] G. Ciardo, R. L. J. III, A. S. Miner, and R. I. Siminiceanu, "Logic and stochastic modeling with SMART," *Performance Evaluation*, vol. 63, no. 6, pp. 578–608, 2006.

[12] F. Ciesinski and C. Baier, "LiQuor: A tool for qualitative and quantitative linear time analysis of reactive systems," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos, CA: IEEE CS, 2006, pp. 131–132.

[13] L. Cloth, J.-P. Katoen, M. Khattri, and R. Pulungan, "Model checking Markov reward models with impulse rewards," in *2005 Int'l Conf. Dependable Systems and Networks*. Los Alamitos, CA: IEEE CS, 2005, pp. 722–731.

[14] D. D'Aprile, S. Donatelli, and J. Sproston, "CSL model checking for the GreatSPN tool," in *Computer and Information Sciences, ISCIS 2004*, ser. LNCS, vol. 3280. Berlin: Springer, 2004, pp. 543–553.

[15] S. Derisavi, "Solution of Large Markov Models using Lumping Techniques and Symbolic Data Structures," Ph.D. dissertation, Univ. of Illinois, Urbana-Champaign, 2005.

[16] S. Derisavi, H. Hermanns, and W. H. Sanders, "Optimal state-space lumping in Markov chains," *Information Processing Letters*, vol. 87, no. 6, pp. 309–315, 2003.

[17] "GNU: GPL," http://www.gnu.org/copyleft/gpl.html.

[18] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[19] B. Haverkort, H. Hermanns, and J.-P. Katoen, "On the use of model checking techniques for dependability evaluation," in *The 19th IEEE Symp. Reliable Distributed Systems*. Los Alamitos, CA: IEEE CS, 2000, pp. 228–237.

[20] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier, "Model checking performability properties," in *Int'l Conf. Dependable Systems and Networks*. Los Alamitos, CA: IEEE CS, 2002, pp. 103–112.

[21] T. Herault, R. Lassaigne, and S. Peyronnet, "APMC 3.0: Approximate verification of discrete and continuous time Markov chains," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos: IEEE CS, 2006, pp. 129–130.

[22] H. Hermanns, *Interactive Markov Chains: The Quest for Quantified Quality*, ser. LNCS. Berlin: Springer, 2002, vol. 2428.

[23] H. Hermanns and S. Johr, "Uniformity by construction in the analysis of nondeterministic stochastic systems," in *Dependable Systems and Networks (DSN)*. Los Alamitos, CA: IEEE CS, 2007, pp. 718–728.

[24] H. Hermanns, B. Wachter, and L. Zhang, "Probabilistic CEGAR," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 5123. Berlin: Springer, 2008, pp. 162–175.

[25] A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker, "PRISM: A tool for automatic verification of probabilistic systems," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 3920. Berlin: Springer, 2006, pp. 441–444.

[26] R. V. Hogg and A. T. Craig, *Introduction to Math. Statistics*, 4th ed. New York, NY: MacMillan, 1978.

[27] A. Hordijk, D. L. Iglehart, and R. A. Schassberger, "Discrete time methods for simulating continuous time Markov chains," *Adv. in Appl. Prob.*, vol. 8, pp. 772–788, 1976.

[28] S. Johr, "Model Checking Compositional Markov Systems," Ph.D. dissertation, Univ. des Saarlandes, Germany, 2007.

[29] J.-P. Katoen, T. Kemna, I. Zapreev, and D. N. Jansen, "Bisimulation minimisation mostly speeds up probabilistic model checking," in *Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, ser. LNCS, vol. 4424. Berlin: Springer, 2007, pp. 87–101.

[30] J.-P. Katoen, M. Khattri, and I. S. Zapreev, "A Markov reward model checker," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos, CA: IEEE CS, 2005, pp. 243–244.

[31] J.-P. Katoen and I. S. Zapreev, "Safe on-the-fly steady-state detection for time-bounded reachability," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos, CA: IEEE CS, 2006, pp. 301–310.

[32] ——, "Simulation-based CTMC model checking: An empirical evaluation," in *Quantitative Evaluation of Systems (QEST)*. Los Alamitos, CA: IEEE CS, 2009.

[33] M. Kwiatkowska, G. Norman, and D. Parker, "Symmetry reduction for probabilistic model checking," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 4114. Berlin: Springer, 2006, pp. 234–248.

[34] T. Littlefair, "CCCC web page," http://cccc.sourceforge.net/.

[35] M. Malhotra, J. K. Muppala, and K. S. Trivedi, "Stiffness-tolerant methods for transient analysis of stiff Markov chains," *Microelectronics and Reliability*, vol. 34, no. 11, pp. 1825–1841, 1994.

[36] M. Massink, J.-P. Katoen, and D. Latella, "Model checking dependability attributes of wireless group communication," in *Dependable Systems and Networks (DSN)*. Los Alamitos, CA: IEEE CS, 2004, pp. 711–720.

[37] "MRMC: Downloads," http://www.mrmc-tool.org/.

[38] I. Munro and P. M. Spira, "Sorting and searching in multisets," *SIAM J. Computing*, vol. 5, no. 1, pp. 1–8, 1976.

[39] A. Nadas, "An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean," *Annals of Math. Statistics*, vol. 40, no. 2, pp. 667–671, 1969.

[40] G. D. Penna, B. Intrigila, I. Melatti, E. Tronci, and M. V. Zilli, "Finite horizon analysis of Markov chains with the Murphi verifier," *Software Tools for Technology Transfer*, vol. 8, no. 4–5, pp. 397–409, 2006.

[41] S. Pissanetzky, *Sparse Matrix Technology*. London, UK: Academic Press, 1984.

[42] "Prism: Workstation cluster example," `http://www.prismmodelchecker.org/casestudies/clust` http://www.`http://www.prismmodelchecker.org/casestudi` prismmodelchecker.org/casestudies/cluster.php.

[43] M. A. Qureshi and W. H. Sanders, "A new methodology for calculating distributions of reward accumulated during a finite interval," in *Fault-Tolerant Computing (FTCS)*. Los Alamitos, CA: IEEE CS, 1996, pp. 116–125.

[44] W. H. Sanders, W. D. Obal, M. A. Qureshi, and F. K. Widjanarko, "The UltraSAN modeling environment." *Performance Evaluation*, vol. 24, no. 1-2, pp. 89–115, 1995.

[45] K. Sen, M. Viswanathan, and G. Agha, "On statistical model checking of stochastic systems," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 3576. Berlin: Springer, 2005, pp. 266–280.

[46] G. S. Shedler, *Regenerative Stochastic Simulation*. Boston, MA: Academic Press, 1993.

[47] D. D. Sleator and R. E. Tarjan, "Self-adjusting binary search trees," *J. ACM*, vol. 32, no. 3, pp. 652–686, 1985.

[48] W. J. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton, NJ: Princeton Univ. Pr., 1994.

[49] H. C. Tijms and R. Veldman, "A fast algorithm for the transient reward distribution in continuous-time Markov chains," *OR Letters*, vol. 26, no. 4, pp. 155–158, 2000.

[50] M. Tribastone and S. Gilmore, "A new generation PEPA workbench," in *Process Algebra and Stochastically Timed Activities (PASTA)*. http://pastaworkshop.org/2006/, 2006.

[51] H. L. S. Younes, "Ymer: A statistical model checker," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 3576. Berlin: Springer, 2005, pp. 429–433.

[52] H. L. S. Younes, M. Kwiatkowska, G. Norman, and D. Parker, "Numerical vs. statistical probabilistic model checking," *Software Tools for Technology Transfer*, vol. 8, no. 3, pp. 216–228, 2006.

[53] H. L. S. Younes and R. G. Simmons, "Probabilistic verification of discrete event systems using acceptance sampling," in *Computer Aided Verification (CAV)*, ser. LNCS, vol. 2404. Berlin: Springer, 2002, pp. 223–235.

[54] ——, "Statistical probabilistic model checking with a focus on time-bounded properties," *Information and Computation*, vol. 204, no. 9, pp. 1368–1409, 2006.

[55] I. S. Zapreev, "Model Checking Markov Chains: Techniques and Tools," Ph.D. dissertation, Univ. of Twente, 2008.