Performance Evaluation 68 (2011) 90–104

Contents lists available at ScienceDirect

Performance Evaluation

journal homepage: www.elsevier.com/locate/peva

The ins and outs of the probabilistic model checker MRMC☆

Joost-Pieter Katoen [a], Ivan S. Zapreev [b], Ernst Moritz Hahn [c], Holger Hermanns [c], David N. Jansen [d,*]

[a] RWTH Aachen University, 52056 Aachen, Germany
[b] CWI, 1098 XG Amsterdam, The Netherlands
[c] Saarland University, 66123 Saarbrücken, Germany
[d] Radboud Universiteit, 6500 GL Nijmegen, The Netherlands

ARTICLE INFO

Article history:
Received 20 February 2010
Accepted 29 April 2010
Available online 13 May 2010

Keywords:
Model checking
Markov chains
CTMC
DTMC
MRM
CTMDP
Numerical analysis
Discrete-event simulation
Bisimulation minimization

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 continuous-time Markov decision processes (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 minimal breakdown probabilities of, for instance, components of steer- and brake-by-wire systems in cars, is vital to fulfill 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 satisfies 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 $M \models \varphi$ (M satisfies φ) 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), 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.

☆ This research was partly performed as part of the MC = MC project financed by the Netherlands Organization for Scientific Research (NWO), and has received support from the DFG Research Training Group 623 on "Leistungsgarantien für Rechnersysteme" and the DFG Special Research Initiative SFB/TR-14 AVACS. The work has also been performed as part of the EU FP7 Project Quasimodo and the DFG–NWO bilateral project ROCKS.

* Corresponding author. Tel.: +31 24 3652271; fax: +31 24 3652525.
E-mail addresses: katoen@cs.rwth-aachen.de (J.-P. Katoen), ivan.zapreev@gmail.com (I.S. Zapreev), emh@cs.uni-saarland.de (E.M. Hahn), hermanns@cs.uni-saarland.de (H. Hermanns), D.Jansen@cs.ru.nl (D.N. Jansen).

0166-5316/$ – see front matter © 2010 Elsevier B.V. All rights reserved.
doi:10.1016/j.peva.2010.04.001

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 (by now) standard numerical analysis techniques [1,2]. 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 [3–5], most notably the verification of time- and reward-bounded reachability probabilities. This allows for verifying properties such as "The probability to reach a safe 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 [6], it has been used several times as a model-checking back-end; some example formalisms are stochastic process algebras [7,8], Petri nets [9], STATEMATE [10], discrete-time stochastic hybrid systems [11], and AADL [12]. Recently, it has been used to compute the coverage of probabilistic model checking in the software model checker Java Pathfinder [13].

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 even smaller quotients.
2. MRMC has been extended with support for continuous-time Markov decision processes (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 [14] and contains an implementation of a recent algorithm [15] to compute these probabilities for arbitrary CTMDPs.
3. MRMC includes a discrete-event simulation engine for statistical CSL model checking [16,17]. It covers bounded and unbounded until formulas, as well as steady-state properties. MRMC is thus the first tool with simulation-based model checking of full CSL.
4. MRMC includes a precise steady-state detection algorithm that is useful for time-bounded reachability properties with large time spans.

MRMC is realised in the programming language C, 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) [18], and is available for free download at [19].

This paper gives an overview of the current tool status, with a particular focus on the recent extensions. We describe MRMC v1.5 as it is planned to be released in June 2010.

Other probabilistic model checkers

In the last years, model checking of probabilistic systems has been an active research field. This has resulted in a whole variety of probabilistic model checkers, such as APMC [20], FHP-Murphi [21], Liquor [22], PASS [23], PRISM [24], VESTA [25], and Ymer [26]. Probabilistic model-checking facilities are also present in the stochastic Petri nets tools GreatSPN [27] and the APNN Toolbox [28]. 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 [29] and sequential acceptance sampling [30]. 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 [31]) discussed in [30,25]. 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. It recently has been equipped with facilities to support symmetry reduction. 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 [32]. SMART is using multi-decision diagrams (MDDs) rather than MTBDDs. MRMC can be considered as the successor of E $\vdash$ MC^2 v1.0 [33], the first CSL model checker.

Organisation 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 small examples, introduces the tool's interactive shell. The recent features of MRMC are 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 and gives some pointers for future extensions of MRMC.

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, respectively). It supports the logics:

Model	DTMC	CTMC	DMRM	CMRM	CTMDP
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 constrained (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 on leaving (or entering) a state. Key formulas 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 bound and a reward bound. Note that in CMRMs the incurred reward in a state is proportional to the reward rate associated to the state and the state residence time. Finally, MRMC supports a 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 under time-abstract schedulers.

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 as a compressed-row (compressed-column) representation (see [34] 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 [35], 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 [36]. We have extended the compressed-row representation with features that give easy access to the diagonal elements and with pointers to find all entries in a single column.

In our implementation a sparse matrix is represented by a structure containing a number of rows **nrows**; the number of columns **ncols**; and an array of structures that contain information about single states.

The structure for state i has several fields, namely the diagonal element $\mathbf{diag} = \mathbf{P}_{ii}$; information about the nonzero off-diagonal entries in matrix row $\mathbf{P}_{i\_}$: the number of entries **succ** and an array of entries **entries**, where each entry is a pair of a coordinate and a value; and finally, the state structure contains information about the nonzero off-diagonal entries in matrix column $\mathbf{P}_{\_i}$: the number of entries **pred** and an array **back_set** of *pointers* to the respective entries.[1] Row and column indices start at 0. As coordinate in an entry, we store the difference of column index minus row index. This allows us, if we know one index, to calculate the other one easily, so that one can walk through all elements of a single row or a single column quickly. The **back_set** array is used for bisimulation minimization and in the model-checking algorithms of PCTL and CSL.

Example 1. Consider the matrix **P**:

$$\mathbf{P} = \begin{pmatrix} 0.33 & 0.0 & 0.67 \\ 0.25 & 0.0 & 0.75 \\ 0.0 & 0.0 & 1.0 \end{pmatrix}.$$

[1] This is a new feature in MRMC version 1.5. Earlier versions only contained the row numbers of the entries, which required searching through all entries in that row. Because our matrices do not change often, the additional complexity of pointers is smaller than the time required by search operations.

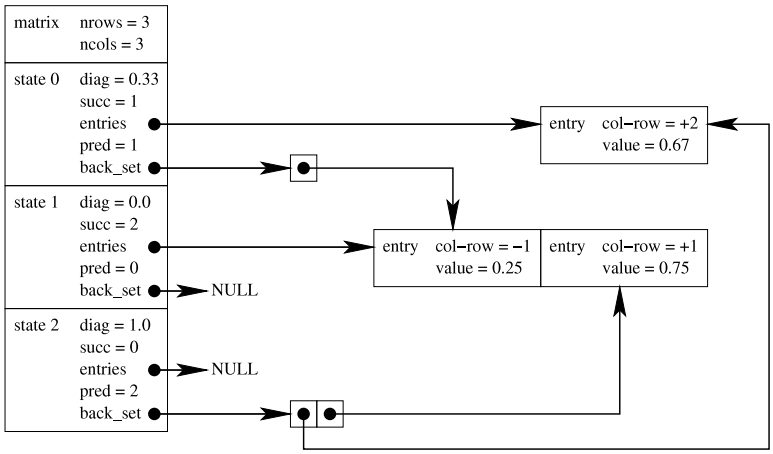


Fig. 1. An example of the sparse matrix representation used in MRMC.

Fig. 1 shows the data structures that are allocated for matrix **P** in MRMC. The matrix structure (on the top left of the figure) indicates it is a 3×3-matrix. Below the matrix structure, the state structures are drawn. For example, the structure for state 0 tells that the diagonal element $\mathbf{P}_{00} = 0.33$. There is **succ** $= 1$ other entry in row 0, namely the one pointed at by **entries**: at column (col − row) + state $= +2 + 0 = 2$, we have $\mathbf{P}_{02} = 0.67$. There is also **pred** $= 1$ off-diagonal entry in column 0, namely the one pointed at by the pointer in **back_set**: at row state − (col − row) $= 0 - (-1) = 1$, we have $\mathbf{P}_{10} = 0.25$.

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 realised in MRMC are given in [1]. Prior to checking an until formula, a graph analysis identifies the states from which the goal state is unreachable or unavoidable to reduce the number of variables in the linear equation system. MRMC extends PCTL [1] with a long-run operator, which is checked by a combination of graph analysis to find bottom strongly connected components (BSCCs, i.e., SCCs that once entered cannot be left anymore), 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 [35]. The PRCTL algorithms in MRMC are described in [4]. CSL model-checking techniques in MRMC stem from [2] and for its reward extension CSRL from [5,37,38]. For time- and reward-bounded until formulas we have implemented two algorithms: one based on discretization [39] and one based on uniformization and path truncation [3]. The algorithms for PRCTL and CSRL support both state and impulse rewards, i.e., instantaneous rewards on edges. The algorithms for maximal probabilities for time-bounded reachability on CTMDPs have been given in [14,15].

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 assumed to be 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*). We will use MRMC to study the probability to win. Table 1 lists the MRMC input files to describe the DMRM. The file `game.tra` lists the state transitions with their probabilities; the file `game.lab` contains label declarations and provides the state labelling; 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.

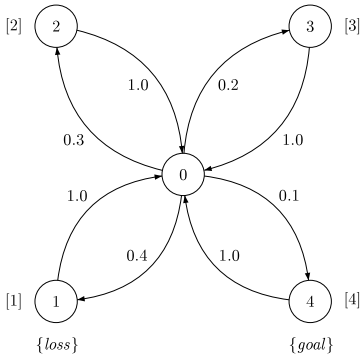


Fig. 2. A DMRM model of a simple dice game.

Table 1
MRMC input files for the dice game.

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		

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 runtime options such as the use of specific numerical algorithms or certain runtime 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 >> signaling 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 us 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 in the model. 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 $\neg \mathit{loss}\ \mathcal{U}^{[0,199]}_{[5,50]}\ \mathit{goal}$, and

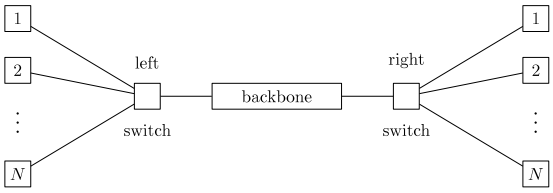
```
$STATE: { }
The model-checking time is 45 milli sec(s).
>>
```

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



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

Fig. 3. FTWC with mean fail and repair times.

interest, because it can be viewed as a common semantic model for various performance and dependability modelling formalisms including generalized stochastic Petri nets [40], Markovian stochastic activity networks [41], and interactive Markov chains [42]. 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 associate a particular probability distribution with, as that is unknown or not applicable. Such decisions 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 model satisfying a property. Instead, probabilities result from applying an entity 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 we have implemented two algorithms, one of which is specialized for *uniform* models, 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. The second algorithm is applicable to arbitrary CTMDPs.

Algorithm for uniform models

MRMC implements an algorithm that efficiently calculates time-bounded reachability probabilities in uniform CTMDPs [14]. The schedulers considered 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 suffices to consider schedulers which only know the *length* of the history. Algorithmically, a greedy backward reachability algorithm is used to compute the maximum, 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 [43]. 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 (Interactive Markov Chain) calculus as supported by the CADP toolset; details are explained in [44]. As in [43] 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 100 h to 50 000 h. For each N we display the computation time in the upper part of the row and the computed probability in the lower part.

As we can see, the uniform CTMDP algorithm performs well even for examples with very large state spaces: The model in the last row of Table 2 has about 2 million states and about 20 million nonzero matrix entries. This holds also for other examples, see [44].

The inherent non-determinism in the specification has been ignored in the original model [43] and in subsequent work, e.g., in the FTWC model used and generated by the model checker PRISM [45]. There, the non-deterministic decision of

Table 2
Statistics for the FTWC analysis.

N	Memory (MB)	States	100 h	1000 h	5000 h	10 000 h	50 000 h	
1	14.16	110	0 s	0 s	0 s	0 s	0 s	Time
			0.00	0.01	0.04	0.09	0.36	Prob.
4	14.29	818	0 s	0 s	0 s	0 s	1 s	Time
			0.00	0.02	0.09	0.18	0.62	Prob.
16	15.39	10 130	0 s	0 s	1 s	1 s	4 s	Time
			0.01	0.08	0.32	0.54	0.98	Prob.
64	32.22	151 058	0 s	1 s	5 s	10 s	46 s	Time
			0.03	0.23	0.73	0.93	1.00	Prob.
256	298.67	2 373 650	7 s	40 s	2 m 46 s	5 m 16 s	24 m 31 s	Time
			0.05	0.43	0.94	1.00	1.00	Prob.

Table 3
Statistics for task scheduling analysis.

Time bound *t*	Memory (MB)	Time	Probability
0.1	63.09	7 s	0.00
0.2	220.96	48 s	0.04
0.3	646.13	3 m 43 s	0.11
0.4	1447.37	11 m 34 s	0.22

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 [44].

Algorithm for non-uniform models

MRMC was recently extended by an implementation of an algorithm that drops the uniformity restriction. It computes time-bounded reachability probabilities in *non*-uniform, i.e., arbitrary CTMDPs [15]. As for the uniform case, this algorithm considers the class of (time-abstract) history-dependent schedulers. An important difference with the uniform setting is that schedulers that base their decision only on the length of the history do not suffice to obtain extremal values in this setting. Instead, schedulers need the entire (time-abstract) history. As a consequence, the worst-case time complexity of this algorithm is worse than for the uniform case: it is exponential in the largest outgoing rate of the CTMDP, the time bound, and the analysis precision.

Experimental results

We consider a case study from the field of stochastic scheduling in which a number of tasks with exponentially distributed processing times have to be scheduled on two identical processors [46]. Jobs may be preempted at decision epochs. Simple scheduling strategies such as "longest expected time first" suffice for minimizing the expected makespan, i.e., the completion time of the last job. We are instead interested in the maximal probability that all tasks are completed before a pre-specified deadline is reached. The scheduling policy is time-abstract: it is allowed to base its decisions on information about the history of the scheduling process, but not on timing information (e.g., when did which job finish). Notably, modelling this well-studied problem in stochastic scheduling naturally leads to a non-uniform CTMDP. As uniformization of this model could change extremal probabilities [14], we applied the recent algorithm [15] mentioned above.

Analysis results obtained with MRMC are summarized in Table 3 for different time bounds (first column). The resource usage (both time- and memory-wise) is large, even though the CTMDP model only covers four jobs, yielding 24 different states. The observed run times are high, which reflects the exponential worst-case time complexity of the algorithm. The excessive space complexity is due to the fact that entire histories need to be stored. For practical usage, we either have to restrict to small models and time bounds, or tailor the algorithm to distinguished model classes such as uniform models.

4.2. Simulation-based CTMC model checking

The numerical analysis algorithms in MRMC have recently been complemented by *simulation-based* model-checking algorithms for full CSL. As opposed to the approaches taken by Ymer [26,47] and VESTA [25] that exploit statistical hypothesis testing, MRMC uses classical discrete-event simulation (DES) techniques for CTMCs. In essence,

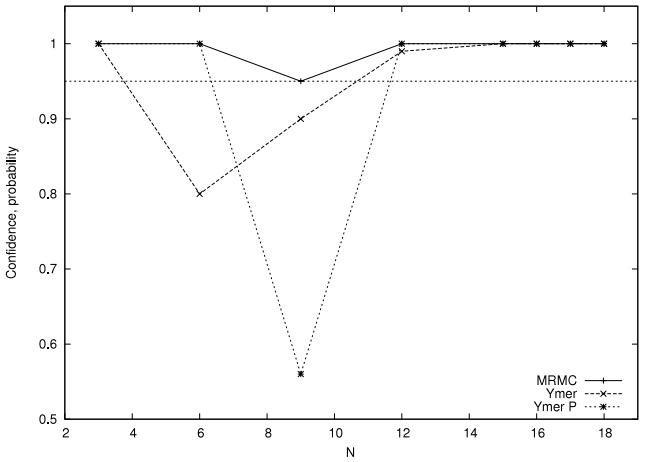


Fig. 4. Confidence levels for $\mathbb{P}_{\geq 0.99}\left(\diamond^{[40,80]} serve_1\right)$.

1. MRMC exploits continuous-time terminating simulations for *time-bounded* until formulas. This is a standard simulation technique in which the termination conditions are naturally induced by the time bound in the formula.
2. MRMC uses regenerative simulation on embedded DTMCs for *steady-state* formulas, and
3. terminating simulations on embedded DTMCs for *unbounded* until formulas.

In the current version of MRMC, it is assumed that the structure of the CTMC at hand is known prior to the simulation, i.e., it is not a fully on-the-fly simulation engine. This allows to perform a reachability and a graph analysis before the actual simulation is carried out. For instance, before starting simulation to check the validity of formula $\mathbb{P}_{>b}(\Phi U \Psi)$ where Φ and Ψ characterize sets of states, all Φ-states from which a Ψ-state cannot be reached are determined and not considered further. This preprocessing allows to focus the simulation only on the relevant states, i.e., the Φ-states that can reach a Ψ-state with positive probability. In addition, it provides a natural termination criterion for the simulation. Full details of the simulation algorithms are provided in [17, Chapter 6].

Basic simulation strategy

To check whether e.g., $s \models \mathbb{P}_{>b}(\varphi)$ for path formula φ, 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 adaption 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 (Cyclic Polling System) case study where we check $\mathbb{P}_{\geq 0.99}\left(\diamond^{[40,80]} serve_1\right)$. 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. the y-axis in Fig. 4) to a confidence which is significantly below the user-defined confidence $\xi = 0.95$ using YMER and Ymer P—the variant of YMER that uses sequential confidence-interval approach based on [48]. 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 and therefore is not included in Fig. 4.) 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. For $N = 9$, MRMC is not faster than Ymer and Ymer P, whereas for $N = 18$ (the maximum model size), MRMC is 15.5 times faster than Ymer P and 27 times slower than Ymer. 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 based on less samples. Our experiments with other cases showed that for (time-bounded) until formulas the peak-memory consumption (VSZ) of MRMC is linearly proportional to the model size.

Simple reachability properties

For unbounded conditional reachability properties, MRMC uses two statistically independent samples so as to ensure the correct confidence level. In contrast to VESTA that equips each state with an artificial "stopping" probability to guess whether from a state a goal state is never reached, MRMC uses the CTMC's underlying graph structure. By means of a graph

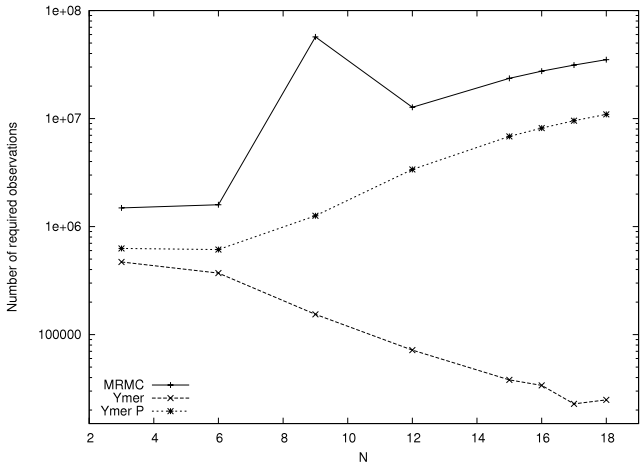


Fig. 5. Number of observations for checking $\mathbb{P}_{\geq 0.99}\left(\diamond^{[40,80]} serve_1\right)$.

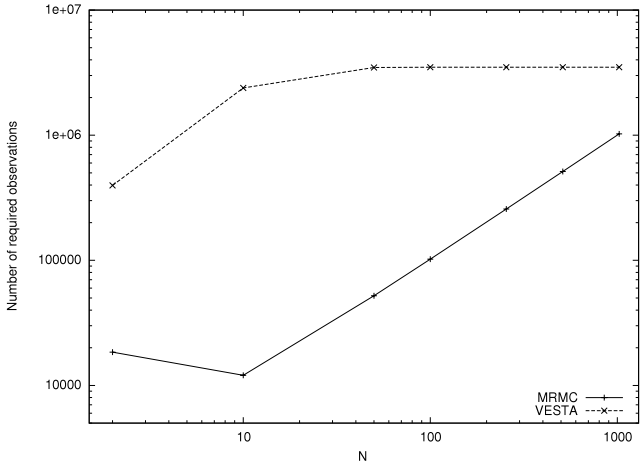


Fig. 6. Sample sizes for checking $\mathbb{P}_{\leq 0.03}\left(\neg full_1 \ \mathcal{U} \ full_2\right)$.

analysis, MRMC determines all states that reach the goal states with probability 0 or 1. For the remaining states we do simulations. This yields a natural termination criterion for the simulation as the simulation now boils down to estimating the stationary probabilities to be in a goal state in the reduced CTMC. These stationary probabilities can be estimated by the state probabilities of transient states found in the graph analysis. This yields a drastic reduction of the needed number of simulated states as indicated in Fig. 6. In fact, we simulate until the Nth 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.

Checking steady-state formulas

For steady-state formulas, 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 former ones are obtained by regeneration-based simulation of the embedded DTMC as originally proposed in [49]. 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 min. MRMC also offers the choice between dynamic (D) and constant (C) sample sizes increase. The former allows to improve tool performance on smaller models, cf. Fig. 7. Neither Ymer nor VESTA provide support for steady-state properties. Our experiments revealed that the

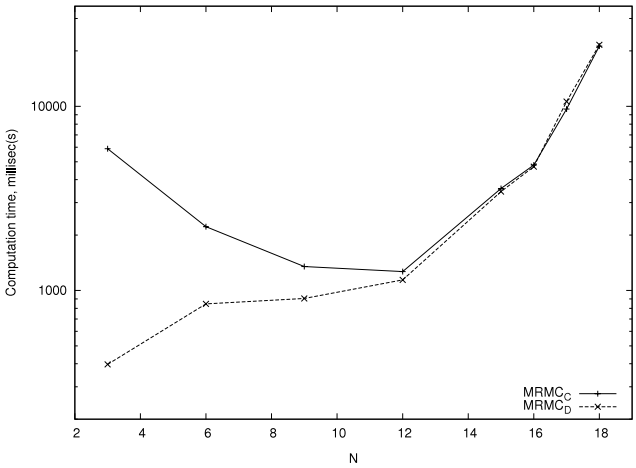


Fig. 7. Model-checking times for $\mathbb{S}_{>0.19}\,(busy_1)$ (time).

memory consumption for simulating steady-state formulas is slightly higher (about a factor 2) than for until formulas 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 [16].

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 can therefore resort to 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 [50]. 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

To store a partition, MRMC (starting from version 1.5) uses a single array of state indices sorted in a particular order: states belonging to a single subset are placed in a contiguous region of the array. A list of subset structures describes where in the array one subset ends and the next one begins. This data structure is simpler than representing each subset as a linked list of states. It is similar to the data structure described in [51].

During splitting, one has to collect the states with the same transition probability $\mathbf{P}(s, Sp)$ into separate subsets. This is equivalent to sorting multisets (i.e., sets where the same key may appear more than once [52]). Standard bisimulation minimization [50] uses splay trees to implement such a sort algorithm. A splay tree is a self-balancing binary search tree with the additional property that recently accessed elements are quick to access again [53]. If one uses a multiset sort algorithm, the worst-case time complexity of partition refinement is $O(|\mathbf{P}| \log |S|)$. It raises to $O(|\mathbf{P}| \log^2 |S|)$ with a general sort algorithm. Very recently, [54] described lumping using a general sort together with a majority candidate algorithm to achieve complexity $O(|\mathbf{P}| \log |S|)$.

In [55], 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 before version 1.5 therefore used an efficient implementation of splay trees developed by Sleator.

In version 1.5 of MRMC, splay sort has been replaced by a variant of multiset quicksort [56] with an additional optimisation: In normal sorting algorithms, the final result has to be in order; we only need to gather the states with identical $\mathbf{P}(s, Sp)$, but the order of the subsets is of no concern. This allows to save some swaps. We also experimented with heapsort.

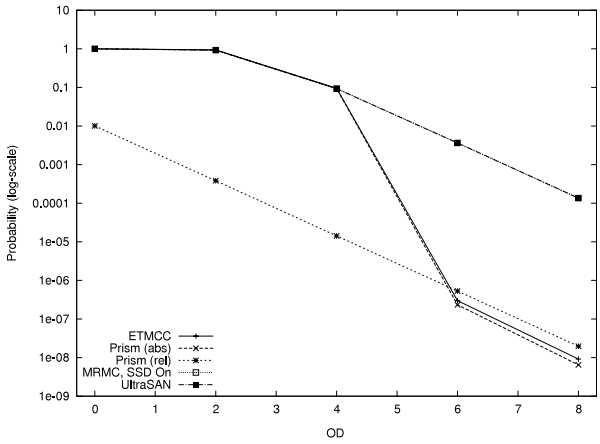


Fig. 8. Premature steady-state detection.

(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.

An additional speedup can be realised if one chooses a data structure that allows to find the predecessors of a state (the entries in a column of the transition matrix) quickly. As described in Section 2.2, we have changed the implementation of sparse matrices in version 1.5 accordingly. We have compared the old and new implementations of sparse matrices and found the expected speedup for our DTMC test suite, where the total runtime with property-independent lumping is 5% less; with property-driven lumping, the total runtime is reduced by 62%. However, for our CTMC test suite, the total runtime is increased by 8%–12%. We suspect that the new implementation needs some optimisation. We are surprised that memory usage of the new implementation is sometimes slightly larger (about 5%), because the new data structure is 4 B per state *smaller;* we assume that differences in memory layout lead to less efficient allocation.

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 – have shown the practical usability of bisimulation minimization; for full details see [57]. We often obtain significant state-space reductions, and in most cases a speedup is achieved. For DTMCs and CTMCs, the total runtime is reduced by a factor 2–20. The cyclic polling server (CPS) test was about 20% faster without minimization; randomized mutual exclusion (RME) could be checked about 1.5 times faster without minimization. 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. Property-driven bisimulation pays off in even more cases: e.g., in the RME case study, it is about two times faster than verifying the original model, and CPS becomes even 40–115 times faster.

For property-driven bisimulation on models without rewards, the speedup comes with almost no memory penalty: the peak memory use typically is reduced or remains unchanged. For ordinary bisimulation CPS showed an increase of peak memory of 20%–70%. Experiments with the P2P case study showed that bisimulation minimization leads to a significantly stronger state-space reduction than symmetry reduction [58]. Symmetry reduction is – as expected – much faster than bisimulation minimization as it operates on a syntactic 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. [59,60]. 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. [35]. MRMC is the only tool that incorporates *precise* steady-state detection, cf. [61]. This technique is based on the fact that, when checking for time-bounded reachability, the original CTMC is made absorbing. Its state space

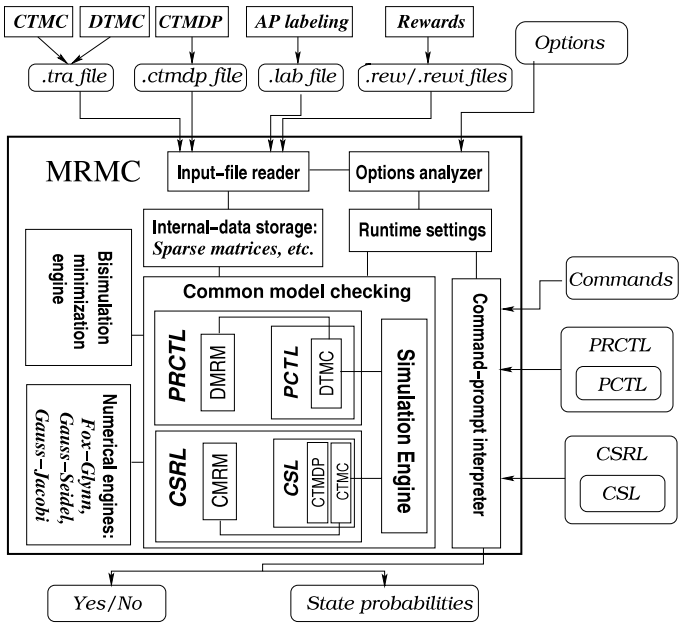


Fig. 9. Tool architecture of MRMC.

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.

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 [62]. 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., $t = 2.4$ s. This time span is extremely large, compared to the duration of each protocol's operation. Fig. 8 indicates that the results of MRMC v1.4.1, coincide with the (exact) values computed by UltraSAN [41], whereas PRISM v3.2 (absolute and relative criteria), and E $\vdash$ 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 [62], the tools' options are as in Section 3.5 of [17].

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 runtime parameters of the tool, such as the logic and the use of (property-driven) bisimulation minimization.

Runtime settings: stores the runtime 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`, `.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) [63], 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 2354. Note that values exceeding 50 are considered to indicate very complex programs. MRMC is constantly tested by an automated test suite [19]. The latter includes most of the case studies discussed in this paper.

6. Conclusions

This paper has reviewed the current status of the model checker MRMC, a tool for the automated verification of Markov models. The main contributions of this paper are: a discussion of the internals of MRMC, an extensive description of the recently added features, including two algorithms for checking time-bounded reachability properties in CTMDPs, simulation-based CTMC model checking against CSL formulas, (property-driven) bisimulation minimization, and on-the-fly steady-state detection.

The main strength of MRMC is its seamless applicability as a back-end component. It is being used as back-end of the performance modelling tools GreatSPN v2.0 [40], the PEPA Workbench [7], and in a tool chain STATEMATE [10]. 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 for model checking of AADL specifications [12]. Recent applications of MRMC are the verification of discrete-time stochastic hybrid systems [11] and the calculation of coverage of partial state-space search in Java Pathfinder [13].

Acknowledgements

We thank Maneesh Khattri (Oxford University), Christina Jansen (RWTH Aachen University), Viet Yen Nguyen (RWTH Aachen University), and Tim Kemna (University of Twente) for their implementation efforts.

References

[1] H. Hansson, B. Jonsson, A logic for reasoning about time and reliability, Formal Aspects of Computing 6 (5) (1994) 512–535.
[2] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, Model-checking algorithms for continuous-time Markov chains, IEEE Transactions on Software Engineering 29 (6) (2003) 524–541.
[3] M.A. Qureshi, W.H. Sanders, A new methodology for calculating distributions of reward accumulated during a finite interval, in: Fault-Tolerant Computing, FTCS, IEEE CS, Los Alamitos, CA, 1996, pp. 116–125.
[4] S. Andova, H. Hermanns, J.-P. Katoen, Discrete-time rewards model-checked, in: K.G. Larsen, P. Niebert (Eds.), Formal Modeling and Analysis of Timed Systems, FORMATS, in: LNCS, vol. 2791, Springer, Berlin, 2003, pp. 88–104.
[5] C. Baier, B. Haverkort, H. Hermanns, J.-P. Katoen, On the logical characterisation of performability properties, in: U. Montanari, J.D.P. Rolim, E. Welzl (Eds.), Automata, Languages, and Programming, ICALP, in: LNCS, vol. 1853, Springer, Berlin, 2000, pp. 780–792.
[6] J.-P. Katoen, M. Khattri, I.S. Zapreev, A Markov reward model checker, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2005, pp. 243–244.
[7] M. Tribastone, S. Gilmore, A new generation PEPA workbench, in: Process Algebra and Stochastically Timed Activities, PASTA, 2006. http://pastaworkshop.org/2006/.
[8] H. Hermanns, S. Johr, Uniformity by construction in the analysis of nondeterministic stochastic systems, in: Dependable Systems and Networks, DSN, IEEE CS, Los Alamitos, CA, 2007, pp. 718–728.
[9] D. D'Aprile, S. Donatelli, J. Sproston, CSL model checking for the GreatSPN tool, in: C. Aykanat, T. Dayar, İ. Körpeoğlu (Eds.), Computer and Information Sciences, ISCIS 2004, in: LNCS, vol. 3280, Springer, Berlin, 2004, pp. 543–553.
[10] E. Böde, M. Herbstritt, H. Hermanns, S. Johr, T. Peikenkamp, R. Pulungan, R. Wimmer, B. Becker, Compositional performability evaluation for statemate, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2006, pp. 167–178.
[11] A. Abate, J.-P. Katoen, J. Lygeros, M. Prandini, Approximate model checking of discrete-time stochastic hybrid systems, European Journal of Control (2010).
[12] M. Bozzano, A. Cimatti, J.-P. Katoen, V.Y. Nguyen, T. Noll, M. Roveri, The COMPASS approach: correctness, modelling and performability of aerospace systems, in: Computer Safety, Reliability and Security, SAFECOMP, in: LNCS, vol. 5775, Springer, Berlin, 2009, pp. 173–186.
[13] X. Zhang, F.V. Breugel, Probabilistic model checking with Java Pathfinder, Technical Report CSE-2010-02, York University, Toronto, Canada, 2010.
[14] C. Baier, H. Hermanns, J.-P. Katoen, B.R. Haverkort, Efficient computation of time-bounded reachability probabilities in uniform continuous-time Markov decision processes, Theoretical Computer Science 345 (1) (2005) 2–26.
[15] T. Brázdil, V. Forejt, J. Krcal, J. Kretínský, A. Kucera, Continuous-time stochastic games with time-bounded reachability, in: R. Kannan, K.N. Kumar (Eds.), Foundations of Software Technology and Theoretical Computer Science, FSTTCS, in: Leibniz International Proceedings in Informatics, LIPIcs, vol. 4, Schloss Dagstuhl—Leibniz-Zentrum fuer Informatik, 2009, pp. 61–72.
[16] J.-P. Katoen, I.S. Zapreev, Simulation-based CTMC model checking: an empirical evaluation, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2009, pp. 31–40.
[17] I.S. Zapreev, Model checking Markov chains: techniques and tools, Ph.D. Thesis, Univ. of Twente, Enschede, The Netherlands, 2008.
[18] GNU: GPL. http://www.gnu.org/copyleft/gpl.html.
[19] MRMC: downloads. http://www.mrmc-tool.org/.
[20] T. Herault, R. Lassaigne, S. Peyronnet, APMC 3.0: approximate verification of discrete and continuous time Markov chains, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2006, pp. 129–130.
[21] G.D. Penna, B. Intrigila, I. Melatti, E. Tronci, M.V. Zilli, Finite horizon analysis of Markov chains with the Murphi verifier, International Journal on Software Tools for Technology Transfer 8 (4–5) (2006) 397–409.
[22] F. Ciesinski, C. Baier, LiQuor: a tool for qualitative and quantitative linear time analysis of reactive systems, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2006, pp. 131–132.
[23] H. Hermanns, B. Wachter, L. Zhang, Probabilistic CEGAR, in: A. Gupta, S. Malik (Eds.), Computer Aided Verification, CAV, in: LNCS, vol. 5123, Springer, Berlin, 2008, pp. 162–175.
[24] A. Hinton, M. Kwiatkowska, G. Norman, D. Parker, PRISM: a tool for automatic verification of probabilistic systems, in: H. Hermanns, J. Palsberg (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, TACAS, in: LNCS, vol. 3920, Springer, Berlin, 2006, pp. 441–444.

[25] K. Sen, M. Viswanathan, G. Agha, On statistical model checking of stochastic systems, in: K. Etessami, S.K. Rajamani (Eds.), Computer Aided Verification, CAV, in: LNCS, vol. 3576, Springer, Berlin, 2005, pp. 266–280.
[26] H.L.S. Younes, Ymer: a statistical model checker, in: K. Etessami, S.K. Rajamani (Eds.), Computer Aided Verification, CAV, in: LNCS, vol. 3576, Springer, Berlin, 2005, pp. 429–433.
[27] S. Bernardi, S. Donatelli, A. Horváth, Compositionality in the Great SPN tool and its application to the modelling of industrial applications, in: K. Jensen (Ed.), Practical Use of High-level Petri Nets, no. PB-547 in DAIMI Technical Report, Univ. of Aarhus, Dept. Computer Science, Aarhus, 2000, pp. 127–146.
[28] P. Buchholz, M. Fischer, P. Kemper, C. Tepper, Model checking of CTMCs and discrete event simulation integrated in the APNN-toolbox, in: F. Bause (Ed.), Measurement, Modelling, and Evaluation of Computer-Comm. Systems, vol. 781, Fachbereich Informatik, Univ. Dortmund, 2003, pp. 30–33.
[29] G.S. Shedler, Regenerative Stochastic Simulation, Academic Press, Boston, MA, 1993.
[30] H.L.S. Younes, R.G. Simmons, Probabilistic verification of discrete event systems using acceptance sampling, in: E. Brinksma, K.G. Larsen (Eds.), Computer Aided Verification, CAV, in: LNCS, vol. 2404, Springer, Berlin, 2002, pp. 223–235.
[31] R.V. Hogg, A.T. Craig, Introduction to Math. Statistics, 4th ed., MacMillan, New York, 1978.
[32] G. Ciardo, R.L. Jones III, A.S. Miner, R.I. Siminiceanu, Logic and stochastic modeling with SMART, Performance Evaluation 63 (6) (2006) 578–608.
[33] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, M. Siegle, A Markov chain model checker, in: S. Graf, M. Schwartzbach (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, TACAS, in: LNCS, vol. 1785, Springer, Berlin, 2000, pp. 347–362.
[34] S. Pissanetzky, Sparse Matrix Technology, Academic Press, London, 1984.
[35] W.J. Stewart, Introduction to the Numerical Solution of Markov Chains, Princeton University Press, Princeton, NJ, 1994.
[36] A. Bell, Distributed evaluation of stochastic Petri nets, Ph.D. Thesis, RWTH Aachen Univ., Germany, 2004.
[37] L. Cloth, J.-P. Katoen, M. Khattri, R. Pulungan, Model checking Markov reward models with impulse rewards, in: Dependable Systems and Networks, DSN, IEEE CS, Los Alamitos, CA, 2005, pp. 722–731.
[38] B. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, C. Baier, Model checking performability properties, in: Dependable Systems and Networks, DSN, IEEE CS, Los Alamitos, CA, 2002, pp. 103–112.
[39] H.C. Tijms, R. Veldman, A fast algorithm for the transient reward distribution in continuous-time Markov chains, Operations Research Letters 26 (4) (2000) 155–158.
[40] D. Cerotti, D. D'Aprile, S. Donatelli, J. Sproston, Verifying stochastic well-formed nets with CSL model-checking tools, in: K. Goossens, L. Petrucci (Eds.), Application of Concurrency to System Design, ACSD, IEEE CS, Los Alamitos, CA, 2006, pp. 143–152.
[41] W.H. Sanders, W.D. Obal, M.A. Qureshi, F.K. Widjanarko, The UltraSAN modeling environment, Performance Evaluation 24 (1–2) (1995) 89–115.
[42] H. Hermanns, Interactive Markov Chains: The Quest for Quantified Quality, in: LNCS, vol. 2428, Springer, Berlin, 2002.
[43] B. Haverkort, H. Hermanns, J.-P. Katoen, On the use of model checking techniques for dependability evaluation, in: IEEE Symp. Reliable Distributed Systems, SRDS, IEEE CS, Los Alamitos, CA, 2000, pp. 228–237.
[44] S. Johr, Model checking compositional Markov systems, Ph.D. Thesis, Univ. des Saarlandes, Germany, 2007.
[45] Prism: workstation cluster example. http://www.prismmodelchecker.org/casestudies/cluster.php.
[46] J. Bruno, P. Downey, G.N. Frederickson, Sequencing tasks with exponential service times to minimize the expected flow time or makespan, Journal of the ACM 28 (1) (1981) 100–113.
[47] H.L.S. Younes, R.G. Simmons, Statistical probabilistic model checking with a focus on time-bounded properties, Information and Computation 204 (9) (2006) 1368–1409.
[48] A. Nadas, An extension of a theorem of Chow and Robbins on sequential confidence intervals for the mean, Annals of Mathematical Statistics 40 (2) (1969) 667–671.
[49] A. Hordijk, D.L. Iglehart, R.A. Schassberger, Discrete time methods for simulating continuous time Markov chains, Advances in Applied Probability 8 (1976) 772–788.
[50] S. Derisavi, H. Hermanns, W.H. Sanders, Optimal state-space lumping in Markov chains, Information Processing Letters 87 (6) (2003) 309–315.
[51] A. Valmari, P. Lehtinen, Efficient minimization of DFAs with partial transition functions, in: Annual Symposium on Theoretical Aspects of Computer Science, STACS, in: Leibniz International Proceedings in Informatics, LIPIcs, vol. 1, 2008, pp. 645–656. http://drops.dagstuhl.de/volltexte/2008/1328/.
[52] I. Munro, P.M. Spira, Sorting and searching in multisets, SIAM Journal on Computing 5 (1) (1976) 1–8.
[53] D.D. Sleator, R.E. Tarjan, Self-adjusting binary search trees, Journal of the ACM 32 (3) (1985) 652–686.
[54] A. Valmari, G. Franceschinis, Simple $O(m \log n)$ time Markov chain lumping, in: J. Esparza, R. Majumdar (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, TACAS, in: LNCS, Springer, Berlin, 2010.
[55] S. Derisavi, Solution of large Markov models using lumping techniques and symbolic data structures, Ph.D. Thesis, Univ. of Illinois, Urbana-Champaign, 2005.
[56] J.L. Bentley, M.D. McIlroy, Engineering a sort function, Software: Practice and Experience 23 (11) (1993) 1249–1265.
[57] J.-P. Katoen, T. Kemna, I. Zapreev, D.N. Jansen, Bisimulation minimisation mostly speeds up probabilistic model checking, in: O. Grumberg, M. Huth (Eds.), Tools and Algorithms for the Construction and Analysis of Systems, TACAS, in: LNCS, vol. 4424, Springer, Berlin, 2007, pp. 87–101.
[58] M. Kwiatkowska, G. Norman, D. Parker, Symmetry reduction for probabilistic model checking, in: T. Ball, R. Jones (Eds.), Computer Aided Verification, CAV, in: LNCS, vol. 4114, Springer, Berlin, 2006, pp. 234–248.
[59] M. Malhotra, J.K. Muppala, K.S. Trivedi, Stiffness-tolerant methods for transient analysis of stiff Markov chains, Microelectronics and Reliability 34 (11) (1994) 1825–1841.
[60] H.L.S. Younes, M. Kwiatkowska, G. Norman, D. Parker, Numerical vs. statistical probabilistic model checking, International Journal on Software Tools for Technology Transfer 8 (3) (2006) 216–228.
[61] J.-P. Katoen, I.S. Zapreev, Safe on-the-fly steady-state detection for time-bounded reachability, in: Quantitative Evaluation of Systems, QEST, IEEE CS, Los Alamitos, CA, 2006, pp. 301–310.
[62] M. Massink, J.-P. Katoen, D. Latella, Model checking dependability attributes of wireless group communication, in: Dependable Systems and Networks, DSN, IEEE CS, Los Alamitos, CA, 2004, pp. 711–720.
[63] T. Littlefair, CCCC web page: http://cccc.sourceforge.net/.

Joost-Pieter Katoen is a full professor at the RWTH Aachen University and is part-time associated with the University of Twente. His research interests include concurrency theory, model checking, timed and probabilistic systems, and semantics. He coauthored more than 130 journals and conferences papers, and co-authored a comprehensive book (with Christel Baier) on "Principles of Model Checking". Joost-Pieter is steering commitee member of QEST, ETAPS, and FORMATS, and is senior member of the ACM.

Ivan S. Zapreev is a software-design engineer at ASML. He obtained his Ph.D. at the University of Twente in 2008, and then worked as a Postdoctoral researcher at Centrum Wiskunde & Informatica (CWI). His research interests include probabilistic and hybrid systems, model checking techniques and tools, type-2 computability theory, computable model checking, temporal logics. Ivan is a program committee member of IVUS'10.

Ernst Moritz Hahn received a Bachelor in Computer Science from Oldenburg University in 2005. After this, he worked at the institute OFFIS e. V. as a member of the research staff. In 2006 he moved to Saarland University, where he received a Master (with honours) in 2008. Since then, he has been a Ph.D. student at the chair of Holger Hermanns. His research interests lie in the area of Markov chains with infinite state-space, parametric Markov models and stochastic hybrid systems.

Holger Hermanns studied at the University of Bordeaux, France, and the University of Erlangen/Nürnberg, Germany, where he received the diploma (with honours) in computer science in 1993 and a doctoral degree (with honours) from the Department of Computer Science in 1998. From 1998 to 2006, he has been with the University of Twente, The Netherlands, holding an associate professor position since October 2001. He is a full professor in computer science at Saarland University, Germany, holding the chair for Dependable Systems and Software since 2003, and he is affiliated with INRIA Grenoble–Rhône-Alpes, France, as part of the VASY team. He has published more than 100 scientific papers, holds various research grants, and has co-chaired several international conferences including CAV, CONCUR, and TACAS. His research interests include modelling and verification of concurrent systems, resource-aware embedded systems, and compositional performance and dependability evaluation.

David N. Jansen is an assistant professor in the Model-based System Development group at Radboud Universiteit Nijmegen since 2007. He studied in Bern, Switzerland and received his Ph.D. degree in Computer Science from Universiteit Twente in 2003, after which he worked a.o. in Saarbrücken and Aachen. His research interests lie in the area of probabilistic and other quantitative extensions of model checking, but he also likes excursions into the philosophy of science.