A Model Checker for Performance and Dependability Properties

Holger Hermanns[a], Joost-Pieter Katoen[a]
Joachim Meyer-Kayser[b], and Markus Siegle[b]

[a]Formal Methods and Tools Group, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands
[b]Lehrstuhl für Informatik 7, University of Erlangen-Nürnberg
Martensstraße 3, 91058 Erlangen, Germany

Abstract— **Markov chains are widely used in the context of performance and reliability evaluation of systems of various nature. Model checking of such chains with respect to a given (branching) temporal logic formula has been proposed for both the discrete [8] and the continuous time setting [1], [3]. In this short paper, we describe the prototype model checker $\mathsf{E} \vdash \mathsf{MC}^2$ for discrete and continuous-time Markov chains, where properties are expressed in appropriate extensions of CTL. We illustrate the general benefits of this approach and discuss the structure of the tool.**

Keywords— **Markov chain, model checking, numerical mathematics, performance evaluation, probabilistic systems, temporal logic**

I. Introduction

A. Model checking

Model checking [6] is a system validation technique that has received an increased attention during the last decade. Given a model of the system (the "possible behavior") and a specification of the property to be considered (the "desirable behavior"), model checking is a technique that systematically checks the validity of the property in the model. Models are typically non-deterministic finite-state automata, consisting of a finite set of states and a set of transitions that describe how the system evolves from one state into another. These automata are usually generated from a high-level description language such as Petri nets, process algebra, Promela [19] or Statecharts [9]. Properties are typically specified in temporal logic, an extension of propositional logic that allows one to express properties that refer to the relative order of events. Statements can either be made about states or about paths, i.e., sequences of states that model a possible evolution of the system. The basis of model checking is a systematic, usually exhaustive, state-space exploration to check whether the property is satisfied in each state or path of the model, thereby using effective methods to combat the state-space explosion problem. Typical properties that are assessed by model checking are:

(i) safety: e.g., does a given mutual exclusion algorithm guarantee mutual exclusion?

(ii) liveness: e.g., will a transmitted packet eventually arrive at its correct destination?

(iii) fairness: e.g., will a repeated attempt to carry out a transaction be eventually granted?

B. On the role of probabilities

Whereas formal verification techniques focus on the absolute guarantee of correctness — "it is impossible that the system fails" — in practice such rigid notions are hard, or even impossible, to guarantee. Instead, systems are subject to various phenomena of probabilistic nature, such as buffer overflow, message loss or garbling and the like, and correctness — "with 99% chance the system will not fail" — is becoming less absolute. In this paper we consider the automated verification of *probabilistic* systems, i.e., systems that exhibit probabilistic aspects[1]. Probabilistic aspects are essential for:

- tackling problems for which non-probabilistic solutions have been proven to be impossible. Typical examples are distributed algorithms like leader election or consensus algorithms where probabilities are used to break the "symmetry" between the processes such that e.g. consensus will eventually be reached with probability one.
- modeling unreliable and unpredictable system behavior. Phenomena like message loss, processor failure and the like can be modeled as non-deterministic scenarios. This is often appropriate in early system design phases where systems are considered at a high

[1]Note: verifying probabilistic systems should not be confused with probabilistic verification, a model checking technique (such as [18]) based on a partial state-space search using imperfect hashing

level of abstraction and where information about the likelihood is (sometimes deliberately) left unspecified. In later design stages, though, where the internal system characteristics become more dominant, probabilities are a useful vehicle to quantify (and thus refine) this information.

• model-based performance evaluation. As performance evaluation is aimed at forecasting system performance and dependability, probabilistic information — what is the distribution of the message transmission delay or what is the failure rate of a processor? — needs to be present in order to evaluate *quantitative* properties (waiting time, queue length, time between failure, and so on).

C. Probabilistic models

There are different ways of adapting finite-state automata to probabilistic phenomena. If all non-determinism is resolved by probabilities, discrete-time Markov chains (DTMCs) result; if non-determinism and probabilistic branching may co-exist, Markov decision processes (MDPs) result. In a DTMC each transition is thus equipped with a (possibly trivial) probability, in an MDP both probabilistic and non-deterministic transitions may appear. Performance and dependability analysis is mostly based on purely probabilistic models, while randomized algorithms are appropriately modeled by MDPs, as probabilities typically affect just a small part of the algorithm and non-determinism is used to model concurrency between processes ("interleaving"). As we are mainly interested in performance and dependability issues, we focus on *purely probabilistic systems*.

II. Discrete-time Markov chains

A. Modeling a telescope

As an example Markov chain we model the failure behavior of the Hubble space telescope, a well-known orbiting astronomical observatory. In particular, we focus on the steering unit which contains six gyroscopes. These gyroscopes are essential to determine where the telescope is pointing. Decisions to move or stabilize the telescope are based on their collected information. Due to their failure possibilities, the gyroscopes are arranged in such a way that any group of three gyroscopes can keep the telescope operating with full accuracy. With less than three gyroscopes the telescope turns into sleep mode and a space shuttle mission must be undertaken for repair. Without operational gyroscope the telescope runs the risk to crash. In practice, three servicing missions (1993, 1997, 1999) have been carried out so far.

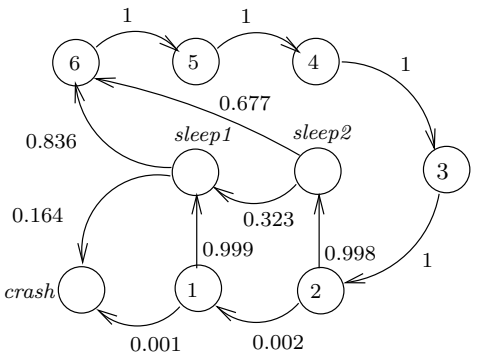

Fig. 1. DTMC of the Hubble space telescope

The DTMC modeling the failure and repair of the HST gyroscopes is depicted in Fig. 1. For convenience, each state is labelled with the number of operational gyroscopes, apart from the states in which the HST is *sleep*ing or *crash*ed. If there are more than two gyroscopes operational, no repairs can take place (as no mission is being sent) and a next gyroscope will fail with probability one. In case two gyroscopes are operational, the system can either move to the *sleep* mode with probability 0.998 or one of the remaining gyroscopes can fail with probability 0.002. Note that these probabilities do not depend on the outcome of decisions taken earlier. Instead, only the current state is decisive to completely determine the probability of evolving to a next state. This is also known as the *memoryless* property of Markov chains. Unless stated otherwise, we consider state 6, the state in which all gyroscopes are operational, as the initial state.

B. Model checking discrete-time Markov chains

With traditional non-probabilistic model checking approaches, properties like:

"the telescope will eventually crash"

can be formally specified and automatically checked. As a DTMC contains quantitative information that enables to determine the actual probability of a certain path being taken, one can gain more insight by checking whether the probability for a certain property meets a given lower- or upper-bound, such as

"the probability that the telescope crashes eventually without ever being in state 1 is at most 10^{-5}"

The temporal logic PCTL (Probabilistic CTL) supports the formal specification of such properties [8]. Clearly, in order to assess the validity of such statements, calculations involving probabilities have to be

carried out. Hansson and Jonsson showed that by a combination of graph algorithms and by solving linear systems of equations, PCTL properties over DTMCs can be automatically verified. This form of model checking is known as *quantitative* model checking. Prototype implementations of their algorithms have been earlier reported in [7], [10], [21].

III. Continuous-time Markov chains

DTMCs are memoryless since probabilistic decisions do only depend on the current state and not on decisions taken earlier. For CTMCs, the continuous-time variant of DTMCs where time ranges over (positive) reals instead of discrete subsets thereof, the memoryless property further implies that the probabilities of taking next transitions do not depend on the amount of time spent in the current state.

A. What is a CTMC?

A CTMC is a finite-state automaton where transitions are labelled by (the rates of) negative exponential distributions. Recall that a random variable X is exponentially distributed with rate λ if the probability of X being at most t (where t is a time parameter) is given by:

$$F_X(t) = \text{Prob}(X \leqslant t) = 1 - e^{-\lambda \cdot t} \text{ for } t \geqslant 0$$

and has mean $\frac{1}{\lambda}$.

To illustrate the concept of a CTMC let us return to the telescope example. We make the following (not necessarily realistic) assumptions about the timing behavior of the telescope: each gyroscope has an average lifetime of 10 years, the average preparation time of a repair mission is two months, and to turn the telescope into sleep mode takes 1/100 years (about 3.5 days) on average. Assuming a base time scale of a single year, the real-time probabilistic behavior of the failure and repair of the gyroscopes is now modeled by the CTMC of Fig. 2. This model can be understood as follows. The mean residence time of a state is the reciprocal of the sum of its outgoing transition rates. In state 6, for instance, one out of 6 gyroscopes may fail. As these failures are stochastically independent and as each gyroscope fails with rate $\frac{1}{10}$, this state has outgoing rate $\frac{6}{10}$. If less operational gyroscopes are available, these rates decrease proportionally, and state residence times become larger. Being in state 2 there are two possibilities: either one of the remaining two gyroscopes fails, or the telescope is turned into sleep mode. The mean residence time of this state is

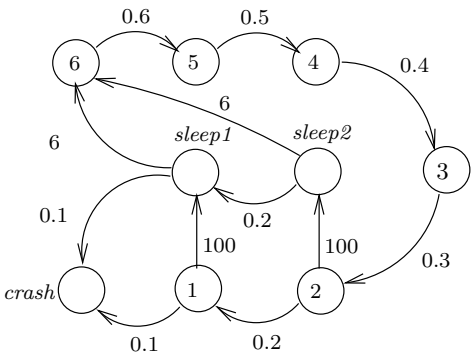

Fig. 2. CTMC of the Hubble space telescope

$\frac{10}{1002}$. The DTMC of Fig. 1 can be obtained from the CTMC of Fig. 2 by multiplying the transition rates by the mean residence time of the state from which they emanate.

B. On the applicability of CTMCs

Markov chains are widely used as simple yet adequate models in diverse areas, ranging from mathematics and computer science to other disciplines such as operations research, industrial engineering, biology and demographics. They can be used to estimate performance characteristics of various nature, for instance to quantify throughput of manufacturing systems, to locate bottlenecks in communication systems, or to estimate reliability in aerospace systems. Due to their modeling convenience and presence of efficient analysis methods, the vast majority of applications of Markov chain modeling involves CTMCs as opposed to DTMCs. Due to the rapidly increasing size and complexity of systems, specifying and analyzing stochastic models at the level of states and transitions becomes more and more cumbersome and error-prone. In order to overcome this problem, CTMCs can be generated from higher level specifications, such as queueing networks, stochastic Petri nets, stochastic process algebras, or from semi-formal software development techniques such as UML (The Unified Modeling Language) or SDL (Specification and Description Language).

C. Model checking continuous-time Markov chains

Performance and dependability analysis of CTMCs most often boils down to the calculation of *steady-state* and *transient state* probabilities. Steady-state probabilities refer to the system behavior on the "long run" while the transient probabilities consider the system at a fixed time instant t. High-level measures-of-interest are determined on the basis of these state-

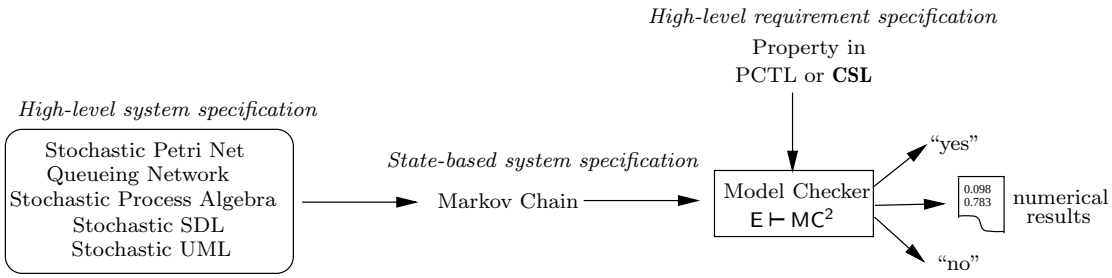

Fig. 3. Model checking CTMCs with $\mathsf{E} \vdash \mathsf{MC}^2$

level probabilities. Until so far, the specification of the measure-of-interest for a given CTMC cannot always be done conveniently, nor can all possible measures-of-interests be expressed conveniently. In particular, measures for which a selection of paths matter are usually either "specified" informally, with all its negative implications, or require a manual tailoring of the CTMC so as to address the right subsets of states.

With the use of an appropriate extension of temporal logic such measures can be specified in an unambiguous way. Let us illustrate this by means of the Hubble telescope example. In addition to the properties discussed for the DTMC model of the telescope, the presence of durations in a CTMC allows us to specify and verify properties that refer to the *time* until a certain scenario happens. Under the assumption that a rare astronomic event, such as the appearance of an interesting comet in the coverage of the telescope, happens in five years, say, it would be interesting to check whether

"the telescope is operational in exactly 5 years from now with at least probability 99%"

Another quantity of interest is the time span before the (fully operational) telescope has to be put into *sleep* mode for the first time. In reality, this happened within 2.7 years. One could check whether

"with at most 10% change the operational telescope turns into *sleep* mode within 2.7 years"

As a last example property, since the Hubble space telescope is planned to stay in orbit through 2010, it is worth to study the likelihood of a crash before that year:

"there is at most a 1% chance that the system will *crash* within the next 10 years"

given that the system was reset to state 6 in late 1999. The recently developed logic CSL (Continuous Stochastic Logic) [1], [3] is an extension of both PCTL and CTL tailored to specify quantitative properties of CTMCs.

IV. The model checker $\mathsf{E} \vdash \mathsf{MC}^2$

The *Erlangen–Twente Markov Chain Checker* ($\mathsf{E} \vdash \mathsf{MC}^2$) is a model checker for DTMCs and CTMCs that supports the automated verification of CSL and PCTL properties.

A. Main model-checking algorithms

The model checker uses numerical methods to model check PCTL and CSL-formulas, based on [8], [3], [2]. Apart from standard graph algorithms, model checking CSL involves matrix-vector multiplication, solution of linear systems of equations, and solution of systems of Volterra integral equations. Linear systems of equations are solved iteratively by standard numerical methods [22]. Two alternatives to solve systems of integral equations are implemented: One is based on piecewise integration of discretized distribution functions, the other is based on uniformisation [2], [20]. Uniformisation is the default option, because it allows the tool to a priori calculate the computational effort needed to check a given property. This effort depends on the numerical parameters of the current model, on the property to be checked, and on the required numerical precision ε (the latter is a parameter set by the user).

B. Implementation considerations

$\mathsf{E} \vdash \mathsf{MC}^2$ is a *global* model checker, i.e. it checks the validity of a formula for all states in the model. It has been developed such that it can easily be linked to a wide range of existing high-level modelling tools based on, for instance, stochastic process algebras, stochastic Petri nets, or queueing networks. A whole variety of such tools exists [11], most of them using dedicated formats to store the transition matrix $\mathbf{R}$ of the Markov chain that is obtained from a high-level specification. This matrix encodes the probabilistic behaviour of the system as time passes. Together with a labelling

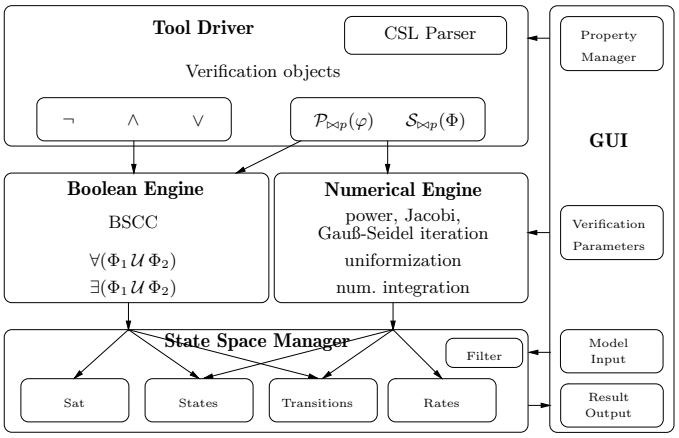

Fig. 4. The tool architecture

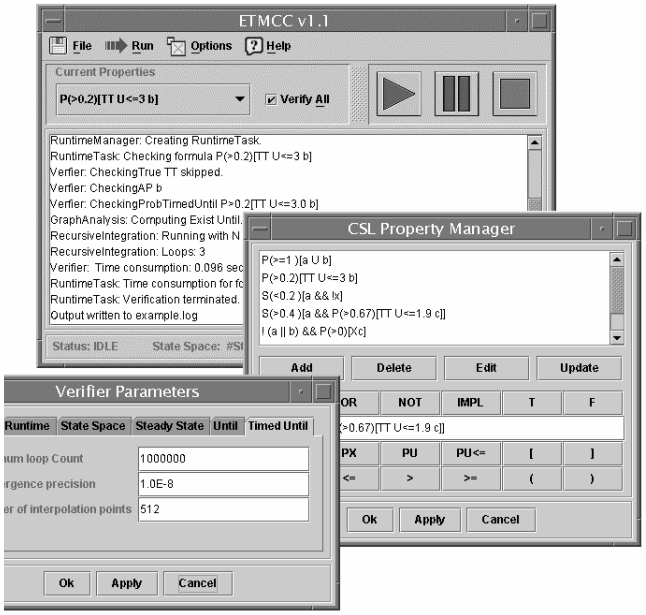

Fig. 5. User interface of $\mathsf{E} \vdash \mathsf{MC}^2$

function L, which associates the states of the Markov chain with sets of atomic propositions, the matrix $\mathbf{R}$ constitutes the interface between the high-level formalism at hand and the model checker. Currently, the tool accepts DTMCs and CTMCs represented in a format generated by the stochastic process algebra tool TIPPtool [17], but the tool is designed in such a way that it can easily bridge to various other input formats. The stochastic Petri net tool DaNAMiCS [4] has recently been extended to generate input for $\mathsf{E} \vdash \mathsf{MC}^2$. An overview of model checking with the tool is given in Fig. 3.

C. Tool architecture

The tool has been written entirely in Java (version 1.2), in order to provide platform independence and to enable fast and efficient program development. Furthermore, support for the development of graphical user interfaces as well as grammar parsers is at hand. For the sake of simplicity, flexibility and extensibility we abstained from low-level optimizations, such as minimization of object invocations. The design and implementation took approximately 15 man-months, with about 10000 lines of code for the kernel and 1500 lines of code for the GUI implementation, using the Swing library. The tool architecture consists of five components, cf. Fig. 4:

- *Graphical User Interface* (cf. Fig. 5) enables the user to load, modify and save verification projects. Each project consists of a model $\mathbf{R}$, a labelling L, and the properties to be checked. The GUI contains the 'CSL Property Manager' which allows the user to construct and edit CSL-formulas. The GUI also prints results and additional logging information on screen or writes them into file. Several verification parameters for the numerical analysis, such as solution method, precision ε, and number of interpolation points for the piecewise integration, can be set by the user.
- *Tool Driver* controls the model checking procedure. It generates the parse tree corresponding to a given CSL property. Subsequent evaluation of the parse tree issues calls to the respective verification objects that encapsulate the verification sub-algorithms. These objects, in turn, use the analysis and/or numerical engine.
- *Analysis Engine* is the engine that supports standard model checking algorithms for CTL-style until-formulas, as well as graph algorithms, for instance to compute the bottom strongly connected components of a Markov chain. The former algorithms are very useful in a pre-processing phase during the checking of probabilistic until-formulas (they may help to avoid many numerical calculations), while the latter is needed when calculating long-run average properties.
- *Numerical Engine* is the numerical analysis engine of the tool. It provides several methods for the numerical solution of linear systems, for numerical integration, and for uniformisation. These are used to solve sytems of linear or integral equations on the basis of parameters provided by the user via the GUI.
- *State Space Manager* represents DTMCs and CTMCs in a uniform way. In fact, it provides an interface between the various checking and analysis components and the way in which DTMCs and CTMCs are actually represented. This eases the use of different, possibly even symbolic (i.e. BDD-based) state

space representations. It is designed to support input formats of various kinds, by means of a simple plug-in-functionality (using JAVA's dynamic class loading capability). It maintains information about the validity of atomic propositions and of sub-formulas for each state. After checking a sub-formula, this sub-component stores the results, to be used later. In the current version of the tool, the state space is represented as a sparse matrix [22]. All real values are stored in the IEEE 754 floating point format with double precision (64 bit).

D. Case studies

Even though the tool is still a prototype, it has already been used in a number of non-trivial case studies, including

- validation and performance assessment of a cyclic server polling system [12],
- reliability estimation of the Hubble space telescope [13],
- dependability analysis of a workstation cluster [14],
- performance and availability analysis of a distributed database server [15].

V. Conclusion

This short paper described the Markov chain model checker $\mathsf{E} \vdash \mathsf{MC}^2$. For more information about the tool, the reader is invited to consult [12], or `http://www7.informatik.uni-erlangen.de/etmcc/`. For academic purposes the tool can be downloaded free of charge via this web-site. The tool is currently being extended towards verifying action-oriented properties expressed in the logic aCSL [15], [16].

Acknowledgements. Holger Hermanns is supported by the Netherlands Organisation for Scientific Research (NWO), Joost-Pieter Katoen is supported by the STW-PROGRESS project TES-4999, "HaaST: Verification of Hard and Softly Timed Systems", and Joachim Meyer-Kayser is supported by the German Research Council DFG. The cooperation between the grouos of Twente and Erlangen takes place in the context of the NWO-DFG bilateral cooperation program (VOSS).

References

[1] A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *Computer Aided Verification, CAV 96*, Springer LNCS 1102: 269–276, 1996.

[2] C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Computer Aided Verification, CAV 2000*, Springer LNCS 1855: 358-372, 2000.

[3] C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR 99*, Springer LNCS 1664: 146–162, 1999.

[4] B. Changuion, I. Davies, and M. Nelte. DaNAMiCS - a Petri Net Editor. `http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS/`.

[5] E.M. Clarke, E.A. Emerson and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Tr. on Progr. Lang. and Sys.*, **8**(2): 244-263, 1986.

[6] E. Clarke, O. Grumberg and D. Peled. *Model Checking.* MIT Press, 1999.

[7] L. Fredlund. The timing and probability workbench: a tool for analysing timed processes. Technical Report No. 49, Uppsala University, 1994.

[8] H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.*, **6**(5): 512–535, 1994.

[9] D. Harel. Statecharts: a visual formalism for complex systems. *Science of Comp. Progr.*, **8**: 231–274, 1987.

[10] V. Hartonas-Garmhausen, S. Campos and E.M. Clarke. PROBVERUS: probabilistic symbolic model checking. In *Formal Methods for Real-Time and Prob. Sys.*, LNCS 1601: 96–111, 1999.

[11] B.R. Haverkort and I.G. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation* **25**: 17–40, 1996.

[12] H. Hermanns, J.P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. In *TACAS 2000*, Springer LNCS 1785: 347–362, 2000.

[13] H. Hermanns. Performance and reliability model checking and model construction. In *Formal Methods for Industrial Critical Systems, FMICS 2000*, GMD Report 91, pages 11-28, Berlin, April 2000.

[14] B. Haverkort, H. Hermanns, and J.P. Katoen. The use of model checking techniques for quantitative dependability evaluation. In *IEEE Symposium on Reliable Distributed Systems*, IEEE CS Press, October 2000.

[15] H. Hermanns, J.P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards Model Checking Stochastic Process Algebra. In *IFM 2000*, Springer LNCS 1945: 420–439, November 2000.

[16] H. Hermanns, J.P. Katoen, J. Meyer-Kayser, and M. Siegle. Implementing a model checker for performability behaviour. In *Fifth International Workshop on Performability Modeling of Computer and Communication Systems*, Erlangen, September 2001.

[17] H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis and M. Siegle. Compositional performance modelling with the TIPPTOOL. *Performance Evaluation*, **39**(1-4): 5–35, 2000.

[18] G.J. Holzmann. An improved protocol reachability analysis technique. *Software – Practice and Experience*, **18**(2): 137–161, 1988.

[19] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice-Hall, 1991.

[20] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In *PAPM/PROBMIV'01*, LNCS 2165. Springer, 2001.

[21] P.-E. Martin. Vopp: a verification tool for probabilistic processes. MSc thesis, Uppsala University, 1993.

[22] W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.