Int J Softw Tools Technol Transfer (2003) 4: 153–172 / Digital Object Identifier (DOI) 10.1007/s100090100072

A tool for model-checking Markov chains

Holger Hermanns[1], **Joost-Pieter Katoen**[1], **Joachim Meyer-Kayser**[2], **Markus Siegle**[2]

[1] Formal Methods and Tools Group, Faculty of Computer Science, University of Twente, P.O. Box 217
7500 AE Enschede, The Netherlands; E-mail: {hermanns,katoen}@cs.utwente.nl
[2] Lehrstuhl für Informatik 7, University of Erlangen-Nürnberg, Martensstrasse 3, 91058 Erlangen, Germany
E-mail: {jmmeyerk,siegle}@informatik.uni-erlangen.de

Published online: 19 November 2002 –

Abstract. Markov chains are widely used in the context of the performance and reliability modeling of various systems. Model checking of such chains with respect to a given (branching) temporal logic formula has been proposed for both discrete [10, 34] and continuous time settings [7, 12]. In this paper, we describe a prototype model checker for discrete and continuous-time Markov chains, the *Erlangen–Twente Markov Chain Checker* ($\mathrm{E} \vdash \mathrm{MC}^2$), where properties are expressed in appropriate extensions of CTL. We illustrate the general benefits of this approach and discuss the structure of the tool. Furthermore, we report on successful applications of the tool to some examples, highlighting lessons learned during the development and application of $\mathrm{E} \vdash \mathrm{MC}^2$.

Keywords: Markov chain – Model checking – Numerical mathematics – Performance evaluation – Probabilistic systems – Temporal logic

1 Introduction

Traditional formal verification techniques try to answer questions related to the functional correctness of systems. Stated otherwise, formal verification aims at predicting system behavior in a *qualitative* way. Typical problems that are addressed by formal verification are:

(i) Safety: e.g., does a given mutual exclusion algorithm guarantee mutual exclusion?
(ii) Liveness: e.g., will a packet transferred via a routing protocol eventually arrive at the correct destination?
(iii) Fairness: e.g., will a repeated attempt to carry out a transaction be eventually granted?

Correspondence to: Holger Hermanns

Prominent formal verification techniques are theorem proving, model checking, and simulation/testing.

Model checking. Model checking [22] is a system validation technique that has received 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 nondeterministic 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 composed of concurrent entities and are often generated from a high-level description language such as Petri nets, process algebra, PROMELA [47] or Statecharts [38]. 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 an 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 of the model, thereby using effective methods (such as symbolic data structures [54], partial-order reduction [56] or clever hashing techniques [46]) to combat the state-space explosion problem.

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 stochastic nature, such as message loss or garbling and the like, and correctness – "with 99% change 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 [29]. Typical examples are distributed algorithms such as leader election or consensus algorithms where probabilities are used to break the "symmetry" between the processes such that, for example, consensus will eventually be reached with probability one.
- Modeling unreliable and unpredictable system behavior. Phenomena such as 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 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 like waiting time, queue length, time between failure, and so on.

Probabilistic models. There are different ways in which finite-state automata can be adapted 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*. In particular, we consider DTMCs and their real-time variant, continuous-time Markov chains (CTMCs).

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 any operational gyroscope the telescope will crash. In practice, three servicing missions (1993, 1997, 1999) have been carried out so far.

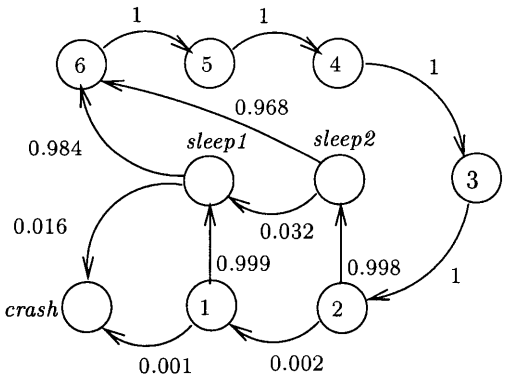


Fig. 1. DTMC of the Hubble space telescope

The DTMC modeling the failure and repair of the HST gyroscopes is depicted in Fig. 1. This model is adopted from [44]. 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.

Model checking discrete-time Markov chains. With traditional model checking approaches, properties like:

"the telescope will eventually crash"

can be formally specified and automatically checked. In a branching-time temporal logic like CTL [21] these properties can be required to hold for all or for some paths. For instance, the CTL-formula

$$\forall \Diamond \, crash \tag{1}$$

is valid in a state if all paths starting in that state will eventually lead to a crash of the telescope. Clearly, this

[1] Note: verifying probabilistic systems should not be confused with probabilistic verification, a model-checking technique (such as [46]) based on a partial state-space search.

property is invalid for the initial state as there exists an infinite path such as

$$6 \to 5 \to 4 \to 3 \to 2 \to 1 \to \mathit{sleep1} \to 6 \to \ldots \quad (2)$$

where the telescope is never in the *crash* state. Under a fair interpretation though, the possibility to jump to the *crash* mode (that is present each time while visiting state 1 or *sleep1*) has to be taken at least once. Thus, under a rather weak fair interpretation – a transition that is enabled infinitely often should be taken eventually – property (1) is valid.

The above property does not refer to the probabilistic aspects of the model and its validity can thus be assessed while ignoring the transition labels. As a first step towards checking properties that involve some information about the likelihood of certain scenarios to happen, one may check whether a certain property holds with either probability 0 or 1. For instance, a property such as

> "the telescope will eventually crash with probability 1"

is evidently valid for the initial state of the telescope as the probability of an infinite path that does not visit *crash*, such as (2), is 0. An interesting observation is that the fairness assumptions needed for the verification in the non-probabilistic case are superfluous in the probabilistic setting as unfair computations are typically scenarios that happen with zero probability [13]. Although these properties refer to the extreme probabilities 0 or 1, their verification thus boils down to checking whether the corresponding non-probabilistic formula holds under a fair interpretation. Thus, standard model checking procedures can be followed. This form is known as *qualitative* model checking of probabilistic systems.

As a DTMC contains quantitative information that enables us 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 [34]. 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 of probabilistic systems. Prototype implementations of their algorithms have been reported earlier in [31, 35, 53].

Continuous-time Markov chains. DTMCs are memoryless since probabilistic decisions 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. DTMCs are mostly applied to strictly synchronous scenarios, while CTMCs have been shown to fit in well with (interleaving) asynchronous scenarios.

A CTMC is a finite-state automaton where transitions are labelled by (the rates of) negative exponential distributions. A formal definition of CTMCs is given in Sect. 2. 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 real-valued parameter) is given by:

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

In this case, the expected value of X is $\frac{1}{\lambda}$.

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 year (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 $\frac{10}{1002}$. 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.

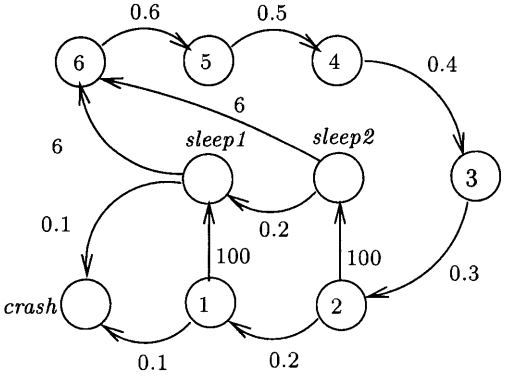


Fig. 2. CTMC of the Hubble space telescope

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 various performance characteristics; 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 the presence of efficient analysis methods, the vast majority of applications of Markov chain modeling involve CTMCs as opposed to DTMCs. This might surprise the reader, as exponential distributions seem at first sight not to be of much practical value, despite their mathematical tractability. This is misleading. Exponential distributions are known to be an appropriate means to adequately model many phenomena with a stochastic nature, such as system lifetimes (failure rates), job arrival processes (inter-arrival times), and the like. Besides, if only the mean value of a random phenomenon is known – a situation that frequently occurs in practice – then the exponential distribution is the most indeterminate distribution, i.e., the distribution with the highest degree of randomness, that describes this phenomenon. Thus, it is the most appropriate distribution when just mean values are known. Technically speaking, the exponential distribution maximizes the entropy [60], a well-known notion from information theory.

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 [24], stochastic Petri nets [1], stochastic process algebras [16, 40, 45], or from semi-formal software development techniques such as UML (The Unified Modeling Language) [58] or SDL (Specification and Description Language) [28]. The tool-development for these techniques and their success in several case studies of industrial importance during recent years has provided strong evidence that these solutions are indeed very promising.

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 in 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-level probabilities. So far, the specification of the measure-of-interest for a given CTMC cannot always be done conveniently, nor can all possible measures-of-interest 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 astronomical event, such as the appearance of an interesting comet in the coverage of the telescope, happens in, say, five years, 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% chance 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 worthwhile studying 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.

Contributions of this paper. Model checking of CTMCs has been discussed in [12], introducing a (branching) temporal logic called *continuous-time stochastic logic* (CSL) to express properties over CTMCs. This logic is an extension of the (equally named) logic by Aziz et al. [7, 8] with an operator to reason about steady-state probabilities.

In this paper, we describe the *Erlangen–Twente Markov Chain Checker* ($\mathrm{E} \vdash \mathrm{MC}^2$), to our knowledge the first implementation of a model checker for CTMCs, see Fig. 3. It uses the methods proposed in [11, 12] to model check CSL-formulas. Apart from standard graph algorithms, model checking involves matrix-vector multiplications (for next-formulas), solutions of linear systems of equations (for until- and steady-state formulas), solutions of systems of Volterra integral equations or, alternatively, uniformization (for time-bounded until). Linear systems of equations are iteratively solved by standard numerical methods [61]. Systems of integral equations are iteratively solved by piecewise integration or by uniformization.

$\mathrm{E} \vdash \mathrm{MC}^2$ is also capable of model checking DTMCs against properties expressed in PCTL [34]. This is not surprising, taking into account that the algorithms needed for CSL are a superset of what is needed to check PCTL. The tool has been implemented in JAVA (version 1.2), and uses sparse matrix representations. The paper illustrates how the tool can be linked (among others) to generalized stochastic Petri nets (GSPN) and to Markovian queueing

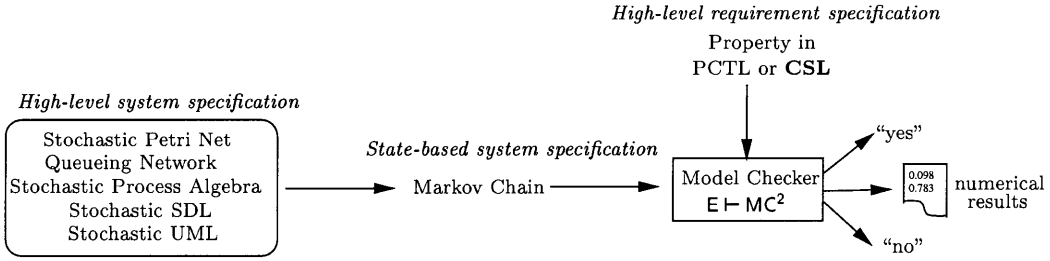


Fig. 3. Model checking CTMCs with $\mathrm{E} \vdash \mathrm{MC}^2$

networks. It reports on the model checking of a GSPN-model of a cyclic server system and of a tandem queueing network.

Organization of the paper. Section 2 introduces CTMCs and CSL. Section 3 discusses the tool architecture together with the model checking algorithm and some implementation details. Section 4 reports on practical experiences with two case studies and Sect. 5 puts the tool in the context of related work. Section 6 concludes the paper.

2 The model

This section introduces and explains the basic Markov chain model. We consider a slight extension of the standard CTMC model, namely CTMCs whose states are labelled by *atomic propositions*. These labels indicate, for instance:

- The status of buffers or other system resources,
- The value of important system variables, or
- The status of the system itself (waiting for messages, or similar).

Note however, that the labelling has no influence on the stochastic behavior of the CTMC; it is used purely for the purpose of identifying elementary properties of states.

2.1 Continuous-time Markov chains

Let AP be a fixed, finite set of atomic propositions. We define a labelled continuous-time Markov chain as a tuple $\mathcal{M} = (S, \boldsymbol{R}, L)$ where

- S is a finite set of *states*,
- $\boldsymbol{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ is the *rate matrix*, and
- $L : S \to 2^{AP}$ is the *labelling* function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions $a \in AP$ that are valid in s.

The rate matrix $\boldsymbol{R}$ characterizes the transitions between the states of the CTMC. If $\boldsymbol{R}(s, s') > 0$ then it is possible that a transition from state s to state s' takes place. Conversely, if $\boldsymbol{R}(s, s') = 0$ then no such transition is possible. If state s has only a single possible successor state s', then the probability of moving from state s to s' within t time units (for non-negative t) is given by $1 - \mathrm{e}^{-\boldsymbol{R}(s,s') \cdot t}$. This expression is the cumulative probability distribution function of an exponential distribution with rate $\boldsymbol{R}(s, s')$.

In the case where $\boldsymbol{R}(s, s') > 0$ for more than one state s', a competition between the transitions exists, also called a *race*. Let $\boldsymbol{E}(s) = \sum_{s' \in S} \boldsymbol{R}(s, s')$, the total rate at which any transition emanating from state s is taken. This rate is the reciprocal of the mean sojourn time in s. More precisely, $\boldsymbol{E}(s)$ specifies that the probability of leaving s within t time units is $1 - \mathrm{e}^{-\mathbf{E}(s) \cdot t}$, due to the fact that the minimum of exponential distributions (competing in a race) is again exponentially distributed, and characterized by the sum of their rates.

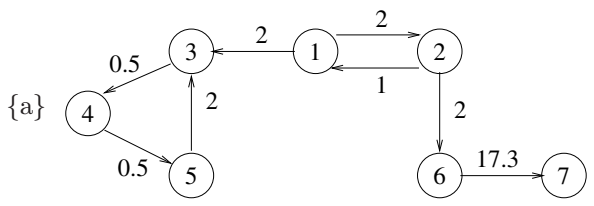
Consequently, the probability of moving from state s to s' in a single step, denoted $\boldsymbol{P}(s, s')$, is determined by the probability that the delay of going from s to s' finishes before the delays of other outgoing edges from s; formally, $\boldsymbol{P}(s, s') = \boldsymbol{R}(s, s') / \boldsymbol{E}(s)$ (except if s is an absorbing state, i.e., if $\boldsymbol{E}(s) = 0$; in this case we define $\boldsymbol{P}(s, s') = 0$).

Figure 4 shows an example labelled CTMC over $AP = \{a, b, c, d\}$, its rate matrix $\boldsymbol{R}$, its probability matrix $\boldsymbol{P}$, and vector $\boldsymbol{E}$. Each state s is decorated with $L(s)$. This example will be used throughout the remainder of this paper to illustrate various issues.

Note that in Fig. 4 all states are reachable from states 1 and 2. However, once state 3 is entered, the CTMC will remain within the subset of states $\{3, 4, 5\}$ forever. Such a subset (which cannot be left and whose states are all mutually reachable) is called a *bottom strongly connected component* (BSCC). Whenever state 6 is entered, the next transition will inevitably lead to state 7 which does not possess any outgoing transition. Such a state which cannot be left is called *absorbing*. An absorbing state can also be viewed as a BSCC containing only a single state. States which do not belong to a BSCC are called transient.

2.2 Evolution in time

One is often interested how the probability mass flows through the CTMC as time passes. If the system is started in some state $s \in S$ at time 0 (i.e., the probability of being in state s is 1 at time 0), the vector $\boldsymbol{\pi}^s(t) = (\pi^s_{s'}(t))_{s' \in S}$ denotes the distribution of probability among the states s' at time t, where t is a non-negative



$$\boldsymbol{R}=\begin{pmatrix} 0 & 2 & 2 & 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 2 & 0 \\ 0 & 0 & 0 & 0.5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0.5 & 0 & 0 \\ 0 & 0 & 2 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 17.3 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \boldsymbol{P}=\begin{pmatrix} 0 & \frac{1}{2} & \frac{1}{2} & 0 & 0 & 0 & 0 \\ \frac{1}{3} & 0 & 0 & 0 & 0 & \frac{2}{3} & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & 0 & 0 & 0 & 0 \end{pmatrix} \quad \boldsymbol{E}=\begin{pmatrix} 4 \\ 3 \\ 0.5 \\ 0.5 \\ 2 \\ 17.3 \\ 0 \end{pmatrix}$$

Fig. 4. Example labelled CTMC and corresponding transition rate matrix $\boldsymbol{R}$, probability matrix $\boldsymbol{P}$, and vector $\boldsymbol{E}$

real number. Obviously, $\pi^s_{s'}(0) = 1$ if $s = s'$ and 0 otherwise. To illustrate how the vectors $\boldsymbol{\pi}^s(t)$ characterize the flow of probability mass over time, the vectors

$$\boldsymbol{\pi}^1(t) = \begin{array}{|c|c|c|c|c|c|c|} \hline 1 & .961 & .677 & .049 & .000 & .000 & 0 \\ 0 & .193 & .141 & .085 & .000 & .000 & 0 \\ 0 & .195 & .161 & .398 & .266 & .267 & \frac{4}{15} \\ 0 & .000 & .004 & .136 & .266 & .267 & \frac{4}{15} \\ 0 & .000 & .000 & .018 & .066 & .067 & \frac{1}{15} \\ 0 & .000 & .009 & .011 & .000 & .000 & 0 \\ 0 & .000 & .006 & .301 & .399 & .400 & \frac{2}{5} \\ \hline \end{array}$$

$(t{=}0)$ $(t{=}.01)$ $(t{=}.1)$ $(t{=}1)$ $(t{=}10)$ $(t{=}10^2)$ $(t{=}\infty)$

indicate this flow from state 1 in Fig. 4 towards the BSCCs as time passes. The last column indicates the limiting probability distribution as $t \to \infty$. This limit exists for arbitrary finite CTMCs, and will be denoted $\boldsymbol{\pi}^s$. It is usually called the *steady-state distribution*, as opposed to the time-dependent distributions, which are called *transient distributions*. In general both, transient and steady-state distributions are dependent on the initial state s occupied at time 0: in the above example, $\boldsymbol{\pi}^5(t)$ and $\boldsymbol{\pi}^5$ are clearly different from $\boldsymbol{\pi}^6(t)$ and $\boldsymbol{\pi}^6$, respectively.

2.3 Computing probability distributions

To efficiently determine transient and steady-state distributions of a given CTMC requires different recipes. We sketch the main steps here, and postpone more details on the numerical algorithms needed for this purpose to Sect. 3.1.

– In order to explain the computation of the steady-state probability vector $\boldsymbol{\pi}^s$, we consider a special case first. If the CTMC consists of a single BSCC, the steady-state distribution is independent of $\boldsymbol{\pi}^s(0)$ (i.e., of s) and can be obtained by solving the linear system of equations $\boldsymbol{\pi}^s \cdot \boldsymbol{Q} = 0$, which has a unique solution independent of s. Here, $\boldsymbol{Q}$ denotes the infinitesimal generator matrix of the CTMC, whose non-diagonal elements are equal to the non-diagonal elements of $\boldsymbol{R}$ and whose diagonal elements are given by the negative row sums of $\boldsymbol{R}$, i.e., $\boldsymbol{Q}(s,s') = \boldsymbol{R}(s,s')$ for $s \neq s'$ and

$$\boldsymbol{Q}(s,s) = -\sum_{\substack{s' \in S \\ s \neq s'}} \boldsymbol{R}(s,s')$$

In general, the picture is slightly more complicated: After an infinite time, the CTMC is certainly no longer in any transient state, but will be in one of its BSCCs and remains there forever. The probability of reaching a particular BSCC can be calculated easily. For example, in the CTMC depicted in Fig. 4, the probability of reaching BSCC $\{3,4,5\}$, provided that the initial state is state 1, is given by

$$\frac{1}{2} + \frac{1}{2} \cdot \frac{1}{3} \cdot \frac{1}{2} + \left(\frac{1}{2} \cdot \frac{1}{3}\right)^2 \cdot \frac{1}{2} + \cdots = \frac{1}{2} \sum_{k=0}^{\infty} \left(\frac{1}{6}\right)^k = \frac{3}{5}.$$

Likewise it can be established that the probability of reaching BSCC $\{7\}$ equals $\frac{2}{5}$. Within each given BSCC one can compute the steady state distribution by solving a linear system of equations (in the size of the BSCC) as described above. Altogether, the probability that the CTMC is in state i after an infinite time is equal to the probability of reaching the corresponding BSCC, multiplied by the steadystate probability of state i within that BSCC.

– The calculation of the transient probability vectors $\boldsymbol{\pi}^s(t)$ of a CTMC proceeds in a different way. Transient probability distributions can be determined via a system of differential equations

$$\frac{\mathrm{d}\boldsymbol{\pi}^s(t)}{\mathrm{d}t} = \boldsymbol{\pi}^s(t) \cdot \boldsymbol{Q}$$

with the fixed boundary condition $\boldsymbol{\pi}^s(0)$. The unique solution of these so-called Kolmogorov differential equations is

$$\boldsymbol{\pi}^s(t) = \boldsymbol{\pi}^s(0) \cdot \mathrm{e}^{\boldsymbol{Q} \cdot t},$$

where the matrix exponential is defined by $\mathrm{e}^{\boldsymbol{Q} \cdot t} = \sum_{k=0}^{\infty} (\boldsymbol{Q} \cdot t)^k / k!$. In Sect. 3.1 we shall explain how this infinite sum can be truncated and computed in a numerically stable way, using a technique called uniformization.

2.4 The continuous stochastic logic CSL

The continuous stochastic logic CSL is a CTL-like temporal logic which is interpreted over a (labelled) CTMC. CSL allows one to specify (state) formulas, denoted by Φ, which may or may not be satisfied by a particular state of a CTMC.

Definition 1. *For $a \in AP$, $p \in [0,1]$, $t \in \mathbb{R}_{\geqslant 0}$, and $\bowtie \in \{\leqslant, <, \geqslant, >\}$, the state-formulas of CSL are defined by the grammar*

$$\Phi ::= a \mid \Phi \wedge \Phi \mid \neg\Phi \mid \mathcal{S}_{\bowtie p}(\Phi) \mid \mathcal{P}_{\bowtie p}(X\,\Phi) \mid \mathcal{P}_{\bowtie p}(\Phi\,\mathcal{U}\,\Phi) \mid \mathcal{P}_{\bowtie p}(\Phi\,\mathcal{U}^{\leqslant t}\,\Phi)$$

Boolean connectives other than conjunction ($\wedge$) and negation ($\neg$) can be derived in the usual way. For instance, $\Phi \vee \Psi$ can be obtained by $\neg(\neg\Phi \wedge \neg\Psi)$. The probabilistic operator $\mathcal{P}_{\bowtie p}(.)$ replaces the usual CTL path quantifiers $\exists$ and $\forall$ which can be re-invented – up to fairness [13] – as the extremal probabilities $\mathcal{P}_{>0}(.)$ and $\mathcal{P}_{\geqslant 1}(.)$. Formula $\mathcal{P}_{\bowtie p}(\varphi)$, where $\varphi = X\,\Phi$ or $\varphi = \Phi\,\mathcal{U}\,\Psi$, asserts that the probability measure of the paths satisfying φ is within the interval specified by $\bowtie p$. The meaning of X ("next step") and $\mathcal{U}$ ("until") is standard: the temporal operator $\mathcal{U}^{\leqslant t}$ is the real-time extension of $\mathcal{U}$; path formula $\varphi = \Phi\,\mathcal{U}^{\leqslant t}\,\Psi$ asserts that $\Phi\,\mathcal{U}\,\Psi$ will be satisfied in the time interval $[0,t]$; i.e., there is some $x \in [0,t]$ such that Φ continuously holds during the interval $[0,x[$ and Ψ becomes true at time instant x. The state formula $\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability for a Φ-state is within the interval specified by $\bowtie p$. Temporal operators such as $\Diamond$ ("eventually"), $\Box$ ("always") and their real-time variants $\Diamond^{\leqslant t}$ or $\Box^{\leqslant t}$ can be derived, e.g., $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t}\,\Phi) = \mathcal{P}_{\bowtie p}(true\,\mathcal{U}^{\leqslant t}\,\Phi)$ and $\mathcal{P}_{\geqslant p}(\Box\,\Phi) = \mathcal{P}_{\leqslant 1-p}(\Diamond\,\neg\Phi)$.

2.5 Semantics of CSL

This section gives an informal explanation of the semantics of CSL. For the full formal semantics of the logic the reader is referred to [12].

For a CTMC $\mathcal{M} = (S, \boldsymbol{R}, L)$ with proposition labels in AP, we define

$$Sat(\Phi) = \{\, s \in S \mid s \models \Phi \,\}.$$

The semantics for atomic propositions, negation, and conjunction is standard [21]. For the remaining operators, the basic object of the semantics are time-stamped paths through $\mathcal{M}$. These are (usually infinite) sequences of states of the model, connected by time-stamped arrows, as in

$$s_0 \xrightarrow{t_0} s_1 \xrightarrow{t_1} s_2 \xrightarrow{t_2} s_3 \xrightarrow{t_3} \cdots.$$

Each time-stamp t_i indicates the time spent in state s_i before moving to the next state s_{i+1}. $Path(s)$ is used to denote the set of paths starting in a state $s \in S$. In general, this set is infinite, not only because there might be infinitely many different sequences of states starting in s, but also because of the continuous nature of exponential distributions. For instance, $Path(6) = \{6 \xrightarrow{t} 7 \mid t \in \mathbb{R}_{\geqslant 0}\}$ for the example in Fig. 4. Each individual path σ in such a set has probability zero, but with the aid of some arguments from measure theory one can define a probability measure, called Pr, on all relevant sets of paths starting in a state of a CTMC [12]. This enables us to define the semantics of the steady-state operator by:

$$s \models \mathcal{S}_{\bowtie p}(\Phi) \text{ iff } \pi^s_{Sat(\Phi)} \bowtie p$$

where π^s_B (for some $B \subseteq S$) denotes the steady-state probability of B when starting in s:

$$\pi^s_B = \lim_{t\to\infty} \Pr\{\, \sigma \in Path(s) \mid \sigma@t \in B \,\}.$$

In this definition, $\sigma@t$ denotes the state occupied in path σ at time t, and is defined relative to the *cumulated* time-stamps along σ, see [12] for details. For instance if $\sigma = s_0 \xrightarrow{1.7} s_1 \xrightarrow{2.4} s_2$, then $\sigma@1 = s_0$, $\sigma@2 = s_1 = \sigma@3.9$, and $\sigma@t = s_2$ for $t > 4.1$.

To introduce the semantics of the remaining three probabilistic operators of the form $\mathcal{P}_{\bowtie p}(\varphi)$, we define $Prob(s,\varphi)$ as the probability measure of all paths $\sigma \in Path(s)$ satisfying φ, formally:

$$Prob(s,\varphi) = \Pr\{\, \sigma \in Path(s) \mid \sigma \models \varphi \,\}$$

and say that

$$s \models \mathcal{P}_{\bowtie p}(\varphi) \text{ iff } Prob(s,\varphi) \bowtie p$$

This definition relies on a satisfaction relation for paths as follows:

- $\sigma \models X\,\Phi$ iff the next state in σ satisfies Φ,
- $\sigma \models \Phi\,\mathcal{U}\,\Psi$ iff there is a state in σ satisfying Ψ and all earlier states satisfy Φ, and
- $\sigma \models \Phi\,\mathcal{U}^{\leqslant t}\,\Psi$ iff there is a time point x – at most t – such that Ψ is satisfied in the state occupied at time x, and all states occupied before time x satisfy Φ.

The above semantics of next and until-formulas is standard [21], while the interpretation of the time-bounded until operator is borrowed from timed CTL [5].

2.6 Some illustrative examples

In order to illustrate the logic CSL we give some examples based on the labelled CTMC from Fig. 4.

- $\Phi_1 = \mathcal{P}_{>0}(Xa)$ holds for all states except states 6 and 7.
- $\Phi_2 = \mathcal{P}_{>0.7}(Xa)$ holds for states 2, 3, 4, and 5, but not for state 1, since the probability of moving from state 1 to its only a-labelled successor state (state 3) is only 0.5.
- $\Phi_3 = \mathcal{P}_{>0}(a\,\mathcal{U}\,b)$ holds for all states except state 2.
- $\Phi_4 = \mathcal{P}_{<0.65}(a\,\mathcal{U}\,b)$ holds for state 2, as no path starting from this state satisfies $a\,\mathcal{U}\,b$, and holds for state 1, as the probability of going from state 1 to state 5 without visiting state 2 equals $\frac{1}{2}$. The property is

refuted by all other states as they satisfy $a\,\mathcal{U}\,b$ with probability 1.

- The checking of $\Phi_5 = \mathcal{P}_{\geqslant 0.5}(a\,\mathcal{U}^{\leqslant 4.0}\,b)$ is more involved. Consider, for example, state 4. The probability that it reaches state 5 within 4.0 time units is given by $1-\mathrm{e}^{-0.5\cdot 4.0} \approx 0.865$, so state 4 does indeed satisfy property Φ_5. For state 3 one has to check whether the probability that the sum of two independent exponential distributions with the same rate parameter (i.e., an Erlang-2 distribution) takes a value of less than 4.0 time units is at least 0.5. This probability is given by $1-\mathrm{e}^{-0.5\cdot 4.0}(1+0.5\cdot 4.0) \approx 0.594$, so it turns out that state 3 also satisfies property Φ_5.
- Let $\Phi_6 = \mathcal{S}_{<0.7}(b)$. The steady-state probability vector in BSCC $\{3,4,5\}$ is $(\frac{4}{9},\frac{4}{9},\frac{1}{9})$, so the probability that b holds in steady state when starting from one of the states in this BSCC is equal to $\frac{1}{9} \approx 0.111$. Therefore, states 3, 4, and 5 do satisfy Φ_6. The probability that b holds in the other BSCC, consisting of only state 7, is equal to 1.0. Thus, starting from states 6 or 7 the probability that b holds in steady state is 1.0, i.e., states 6 and 7 do not satisfy property Φ_6. The remaining states, 1 and 2, require taking into account the probabilities of ending up in each of the two BSCCs in the long run. For state 1 these probabilities are $\frac{3}{5}$ and $\frac{2}{5}$, and therefore the steady-state probability that b holds is $\frac{3}{5}\cdot\frac{1}{9}+\frac{2}{5}\cdot 1.0 = \frac{7}{15} \approx 0.467$, i.e., state 1 does satisfy property Φ_6. Starting from state 2, the steady-state probability that b holds is $\frac{1}{5}\cdot\frac{1}{9}+\frac{4}{5}\cdot 1.0 = \frac{37}{45} \approx 0.822$, i.e., state 2 does not satisfy property Φ_6.
- As a final example, consider $\Phi_7 = \mathcal{S}_{<0.7}(\mathcal{S}_{<0.7}(b))$, i.e., $\Phi_7 = \mathcal{S}_{<0.7}(\Phi_6)$. We know that Φ_6 is valid in states 1, 3, 4, and 5. As a consequence, the steady-state probability of Φ_6 is 1.0 inside the BSCC $\{3,4,5\}$, and 0 in the other BSCC. As a result, Φ_7 holds in state 1, 2, 6, and 7. Note that state 1 satisfies Φ_6 with probability $\frac{3}{5}$ in the long run, which is below the threshold 0.7.

3 The model checker E ⊢ MC²

E ⊢ MC² is a prototype tool supporting the verification of CSL-properties over CTMCs. It is a *global* model checker, i.e., it checks the validity of a formula for all states in the model. E ⊢ MC² has been developed as a model checker that can easily be linked to a wide range of existing high-level modeling tools based on, for instance, stochastic process algebras, stochastic Petri nets, or queueing networks. A whole variety of such tools exists [37], most of them using dedicated formats to store the rate matrix $\boldsymbol{R}$ which is obtained from the high-level specification. The matrix $\boldsymbol{R}$, together with the proposition-labelling function L, constitutes the interface between the high-level formalism at hand and the model checker E ⊢ MC². Currently, E ⊢ MC² accepts CTMCs represented in the `tra`-format as generated by the stochastic process algebra tool TIPPtool [41], but the tool is designed in such a way that it enables a filter plug-in functionality to bridge to various other input formats. The stochastic Petri net tool DaNAMiCS [18] has recently been extended to generate input for E ⊢ MC².

3.1 The model-checking algorithm

Once the matrix $\boldsymbol{R}$ and the labelling L of a CTMC $\mathcal{M}$ have been initialized, the model checking algorithm implemented in E ⊢ MC² essentially proceeds in the same way as for model checking CTL [21]. For a given formula Φ it recursively computes the sets of states $Sat(.)$ satisfying the sub-formulas of Φ, and constructs the set $Sat(\Phi)$ from them. The verification of probabilistic and steady-state properties relies on the constructive characterizations as established in [12].

Steady-state properties. For calculating $\mathcal{S}_{\bowtie p}(\Phi)$ the tool follows a two-phase approach, already indicated in Sect. 2.3. First, the bottom strongly connected components of $\mathcal{M}$ are determined by a standard graph algorithm [62], and the steady-state probability distributions inside each individual BSCC are calculated. Each of these tasks requires the solution of a *linear system of equations* in the size of the respective BSCC. As a second step the probabilities of reaching the individual BSCCs from a given state s is calculated. More precisely, assume B is a BSCC of $\mathcal{M}$ reachable from s. We use an auxiliary atomic proposition to label the states in B, i.e., $a_B \in L(s')$ iff $s' \in B$. Then $\Diamond a_B$ is a path-formula in CSL and $Prob(s,\Diamond a_B)$ is the probability of reaching B from s at some time t. For $s' \in B$, $\pi^s_{s'}$ is given by

$$\pi^s_{s'} = Prob(s,\Diamond a_B)\cdot \pi^B_{s'}$$

where $\pi^B_{s'}=1$ if $B=\{s'\}$, and otherwise $\boldsymbol{\pi}^B$ is a vector of size $|B|$ satisfying the linear system of equations[2]

$$\sum_{\substack{s\in B\\ s\neq s'}} \pi^B_s \cdot \boldsymbol{R}(s,s') = \pi^B_{s'} \cdot \sum_{\substack{s\in B\\ s\neq s'}} \boldsymbol{R}(s',s) \qquad (3)$$

$$\text{such that} \quad \sum_{s\in B} \pi^B_s = 1.$$

All states not contained in any BSCC have steady-state probability 0, independent of the starting state.

Returning to the example in Fig. 4, we get the following linear system of equations for the BSCC $B=\{3,4,5\}$:

$$(\pi^B_3,\pi^B_4,\pi^B_5)\cdot\begin{pmatrix}-0.5 & 0.5 & 0\\ 0 & -0.5 & 0.5\\ 2 & 0 & -2\end{pmatrix} = \begin{pmatrix}0\\0\\0\end{pmatrix},$$

[2] In [12] the above linear system of equations is defined in a slightly different way, by characterizing the steady-state probabilities in terms of the embedded DTMC.

which has the solution $(\frac{4}{9}, \frac{4}{9}, \frac{1}{9})$ as mentioned earlier.

Linear systems of equations can be solved either directly (e.g., Gaussian elimination or LU-decomposition) or by iterative methods such as the power method, Jacobi iteration, and Gauss-Seidel iteration [61]. Iterative methods compute approximations to the exact result up to a prespecified accuracy ε. Although (except for the power method) convergence of the iterative methods is not guaranteed, this problem only appears for pathological cases in practice. The major advantage of these methods is that the involved matrices do not change during the computation (i.e., fill-in is avoided), and hence the buildup of rounding errors is nonexistent [36, 61]. In addition, direct methods are known to be only practical for state spaces of up to a few hundred states, while iterative methods have successfully been applied for much larger systems (up to 10^7 states) [27]. For these reasons, E⊢MC^2 supports all of the above mentioned *iterative* methods to solve (3), the linear system of equations arising from the steady-state operator. (The default option is the Gauss-Seidel iteration.)

Probabilistic next. Recall that $s \models \mathcal{P}_{\bowtie p}(X\,\Phi)$ if and only if $Prob(s, X\,\Phi) \bowtie p$. Calculating the probabilities $Prob(s, X\,\Phi)$ boils down to a single matrix-vector product, multiplying the transition probability matrix $\boldsymbol{P}$ with the (Boolean) vector $\mathbf{i}_\Phi = (i_\Phi(s))_{s\in S}$ characterizing $Sat(\Phi)$, i.e., $i_\Phi(s) = 1$ if $s \models \Phi$, and 0 otherwise. For the example in Fig. 4, for instance $\mathbf{i}_b = (0,0,0,0,0,1,0,1)$, and $\boldsymbol{P}\cdot(\mathbf{i}_b)^T = (0,0,0,0,1,0,1,0)$ gives the probabilities of Xb.

Probabilistic until. Computing $Prob(s, \Phi\,\mathcal{U}\,\Psi)$ proceeds as in the discrete-time case, [25, 34], by solving a linear system of equations of the form

$$\mathbf{x} = \overline{\boldsymbol{P}}\cdot\mathbf{x} + \mathbf{i}_\Psi \tag{4}$$

where $\overline{\boldsymbol{P}}(s,s') = \boldsymbol{P}(s,s')$ if $s \models \Phi \wedge \neg\Psi$ and 0 otherwise. $Prob(s, \Phi\,\mathcal{U}\,\Psi)$ is the least solution of this set of equations [9]. Again returning to the example in Fig. 4, we calculate in this way the probabilities for $a\,\mathcal{U}\,b$ to be $(0.5, 0, 1, 1, 1, 1, 1, 1)$, via a matrix $\overline{\boldsymbol{P}}$ that agrees with $\boldsymbol{P}$ except that $\overline{\boldsymbol{P}}(2,1) = \overline{\boldsymbol{P}}(2,6) = \overline{\boldsymbol{P}}(5,3) = 0$ and vector $\mathbf{i}_b$ as above. E⊢MC^2 computes the least solution of equation (4) by one of the standard iterative methods mentioned above for the steady-state operator.

Time-bounded until. The time-bounded until operator is the most expensive operator from the point of view of analysis effort. Two alternative ways to compute $Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$ have been proposed so far, and both are implemented in E⊢MC^2. The first approach, suggested in [12] uses numerical integration to approximate the values of $Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$. The second approach [11] reduces the model checking problem to a transient analysis of a transformed Markov chain, i.e., to a computation of $\boldsymbol{\pi}^s(t)$. Whichever alternative is selected, it is easy to see that

$$Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi) = 1 \text{ if } s \models \Psi, \text{ and}$$
$$Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi) = 0 \text{ if } s \not\models \Phi \vee \Psi.$$

Thus, the differences concern the cases where $s \models \Phi \wedge \neg\Psi$. In this case, $Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$ satisfies the recursive Volterra integral equation

$$Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi) = \int_0^t \sum_{s'\in S} \boldsymbol{R}(s,s')\cdot \mathrm{e}^{-\boldsymbol{E}(s)\cdot x}\cdot Prob(s', \Phi\,\mathcal{U}^{\leqslant t-x}\,\Psi)\,\mathrm{d}x \tag{5}$$

(or, more precisely, it is the least solution thereof [12]). Expressed in words, the equation describes that the probability of reaching a Ψ-state from s within t time units equals the probability of reaching some direct successor state s' of s within x time units ($x \leqslant t$), multiplied by the probability to reach a Ψ-state from s' in the remaining time-span $t-x$ along Φ-states – for all possible values of x.

We now review the two alternative techniques for solving the Volterra integral equation system (5).

Numerical integration. The above equational characterization can be turned into an iterative method to approximate the solution of (5): setting $F_0(s,t) = 0$ for all s (with $s \models \Phi \wedge \neg\Psi$), and t and iterating

$$F_{k+1}(s,t) = \sum_{s'\in S} \boldsymbol{R}(s,s')\cdot \int_0^t \mathrm{e}^{-\boldsymbol{E}(s)\cdot x}\cdot F_k(s', t-x)\,\mathrm{d}x$$

approaches the values of $Prob(s, \Phi\,\mathcal{U}^{\leqslant t}\,\Psi)$ as k tends to ∞. In [12], we proposed solving these integrals numerically based on quadrature formulas with, say, $N+1$ equally spaced interpolation points $x_m = m\cdot\frac{t}{N}$ ($0 \leqslant m \leqslant N$) such as trapezoidal, Simpson, or Romberg integration schemes [59]. For the trapezoidal method, for instance, this amounts to approximate $F_{k+1}(s, x_m)$ by

$$\sum_{s'\in S} \boldsymbol{R}(s,s')\cdot \sum_{j=0}^{m} \alpha_j\cdot \mathrm{e}^{-\boldsymbol{E}(s)\cdot x_j}\cdot F_k(s', x_m - x_j)$$

where for fixed m, $\alpha_0 = \alpha_m = \frac{t}{2N}$ and $\alpha_j = \frac{t}{N}$ for $0 < j < m$. However, practical experiments during the development of our tool revealed that these schemes may result in inaccurate results by overestimating the impact of the 'leftmost' intervals. We therefore take a different route by using piecewise integration, and approximating $F_{k+1}(s, x_m)$ by

$$\sum_{s'\in S} \boldsymbol{R}(s,s')\cdot \sum_{j=0}^{m} \int_{x_j-\beta_j}^{x_j+\beta_{j+1}} \mathrm{e}^{-\boldsymbol{E}(s)\cdot x}\,\mathrm{d}x\cdot F_k(s', x_m - x_j)$$

where $\beta_0 = \beta_{m+1} = 0$ and $\beta_j = \frac{t}{2N}$ for $0 < j \leqslant m$. Note that the resulting integrals are easily solved exactly because they only involve exponential distributions. Thus,

discretization is used merely to restrict the impact of possible state changes to the interpolation points $x_0, \ldots, x_N$. N is a parameter of the algorithm, prespecified by the user. The influence of the number of interpolation points on the accuracy and the run-time of the algorithm is one of the interesting aspects discussed in Sect. 4.

Solution via transient analysis. An alternative method implemented in the tool is based on a reduction of the problem of calculating the probabilities of $\Phi \mathcal{U}^{\leqslant t} \Psi$ to a transient analysis problem. The idea is to transform the CTMC $\mathcal{M}$ under consideration into another CTMC $\overline{\mathcal{M}}$ such that checking $Prob(s, \Phi \mathcal{U}^{\leqslant t} \Psi)$ on $\mathcal{M}$ amounts to accumulating the probabilities of Ψ-states in $\overline{\mathcal{M}}$ at time t [11]. For the latter it suffices to calculate $\boldsymbol{\pi}^s(t)$ in $\overline{\mathcal{M}}$, for which well-known and efficient computation techniques exist.

For a transformation from $\mathcal{M}$ to $\overline{\mathcal{M}}$ we define $\overline{\boldsymbol{R}}(s, s') = \boldsymbol{R}(s, s')$ if $s \models \Phi \wedge \neg \Psi$ and 0 otherwise, and consider $\overline{\mathcal{M}} = (S, \overline{\boldsymbol{R}}, L)$. It then suffices to carry out a transient analysis on the resulting $\overline{\mathcal{M}}$ for time t and collect the probability mass to be in a Ψ-state:

$$Prob(s, \Phi \mathcal{U}^{\leqslant t} \Psi) = \sum_{s' \in Sat(\Psi)} \pi^s_{s'}(t)\,. \tag{6}$$

To compute the transient probabilities $\boldsymbol{\pi}^s(t)$ (on $\overline{\mathcal{M}}$) at time t, E ⊢ MC^2 uses an efficient and numerically stable technique, known as *uniformization* or Jensen's method [49]. As mentioned in Sect. 2.3, $\boldsymbol{\pi}^s(t) = \boldsymbol{\pi}^s(0) \cdot e^{\boldsymbol{Q} \cdot t}$, where $e^{\boldsymbol{Q} \cdot t} = \sum_{k=0}^{\infty} (\boldsymbol{Q} \cdot t)^k / k!$. For the practical computation of $\boldsymbol{\pi}^s(t)$ one constructs the matrix $\boldsymbol{U} = \boldsymbol{Q}/q + \boldsymbol{I}$, where q is at least the maximum exit rate and $\boldsymbol{I}$ is an identity matrix of the appropriate dimension. This construction is called uniformization. It then holds that

$$\boldsymbol{\pi}^s(t) = \boldsymbol{\pi}^s(0) \cdot \sum_{k=0}^{\infty} \boldsymbol{U}^k \cdot \frac{(q \cdot t)^k}{k!} \cdot \mathrm{e}^{-q \cdot t}, \tag{7}$$

where the weight factors $\frac{(q \cdot t)^k}{k!} \cdot e^{-q \cdot t}$ are known as the Poisson probabilities. The matrix exponentiation needed for $\boldsymbol{U}^k$ is numerically well-behaved, because $\boldsymbol{U}$ is a stochastic matrix. Furthermore, it suffices to evaluate a finite number of terms of this infinite sum, and the number of terms needed can be calculated *a priori*, relative to a prespecified accuracy ε. For the efficient evaluation of the Poisson probabilities we have implemented the approximation of Fox/Glynn [30].

The runtime of the uniformization algorithm is linear in q and also in the time point t under consideration (for large $q \cdot t$). For large t the CTMC may already have reached an equilibrium before t. Therefore, we have integrated an on-the-fly steady-state detection into the algorithm for transient analysis [55]. The effect of this detection on the verification run-time is further discussed in Sect. 4. A further optimization that we have implemented in E ⊢ MC^2 has recently been proposed in [50]. This optimization amountsto combining and re-organizing the computations in (6) and (7). This avoids the need for a computation for each state, and yields an efficiency improvement that is proportional to the number of states in the CTMC under consideration.

3.2 Preprocessing until-formulas

As in [34] for until and time-bounded until some preprocessing is done by the tool before the actual model checking is carried out. First, we determine the set of states for which the (fair) CTL-formula $\exists(\Phi \mathcal{U} \Psi)$ is valid, i.e., we compute $Sat(\exists(\Phi \mathcal{U} \Psi))$. This is done in the usual iterative way [34]. For states not in this set the respective probabilistic until-formula will have probability 0. In a similar way, we compute the set of states for which the probability of these properties will be 1. This is done by computing the set of states $Sat(\forall(\Phi \mathcal{U} \Psi))$ (up to fairness, see [13]) in the usual iterative way [34]. As a result, the actual computation – it being the solution of the linear system of equations in case of an unbounded until or the solution of the system of Volterra integral equations in case of the time-bounded until – can be restricted to the remaining states. This not only reduces the number of states, but also speeds up the convergence of the iterative algorithms.

3.3 Tool architecture

E ⊢ MC^2 has been written entirely in JAVA (version 1.2), an object-oriented language known 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 are 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 of E ⊢ MC^2 took approximately 12 man-months, with about 9000 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, see Fig. 5.

Graphical User Interface (see Fig. 6) enables the user to create, load, and save verification projects, consisting of a model $\mathcal{M}$ (which contains a rate matrix $\boldsymbol{R}$ and a labelling L), and the properties to be checked. The GUI prints results on screen or writes them into a file and allows the user to construct CSL-formulas by the 'CSL Property Manager'. Several verification parameters for the numerical analysis, such as solution method, accuracy ε and, (in case numerical integration is selected), number of interpolation points N, can be set by the user.

Tool Driver controls the model checking procedure. It parses a CSL-formula and generates the corresponding parse tree. Subsequent evaluation of the parse tree results in calls to the respective verification ob-

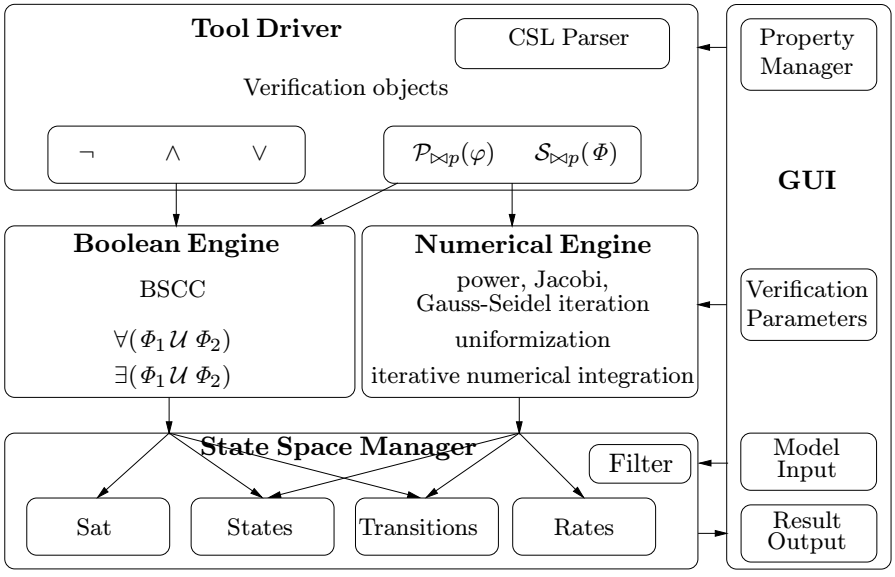


Fig. 5. The tool architecture

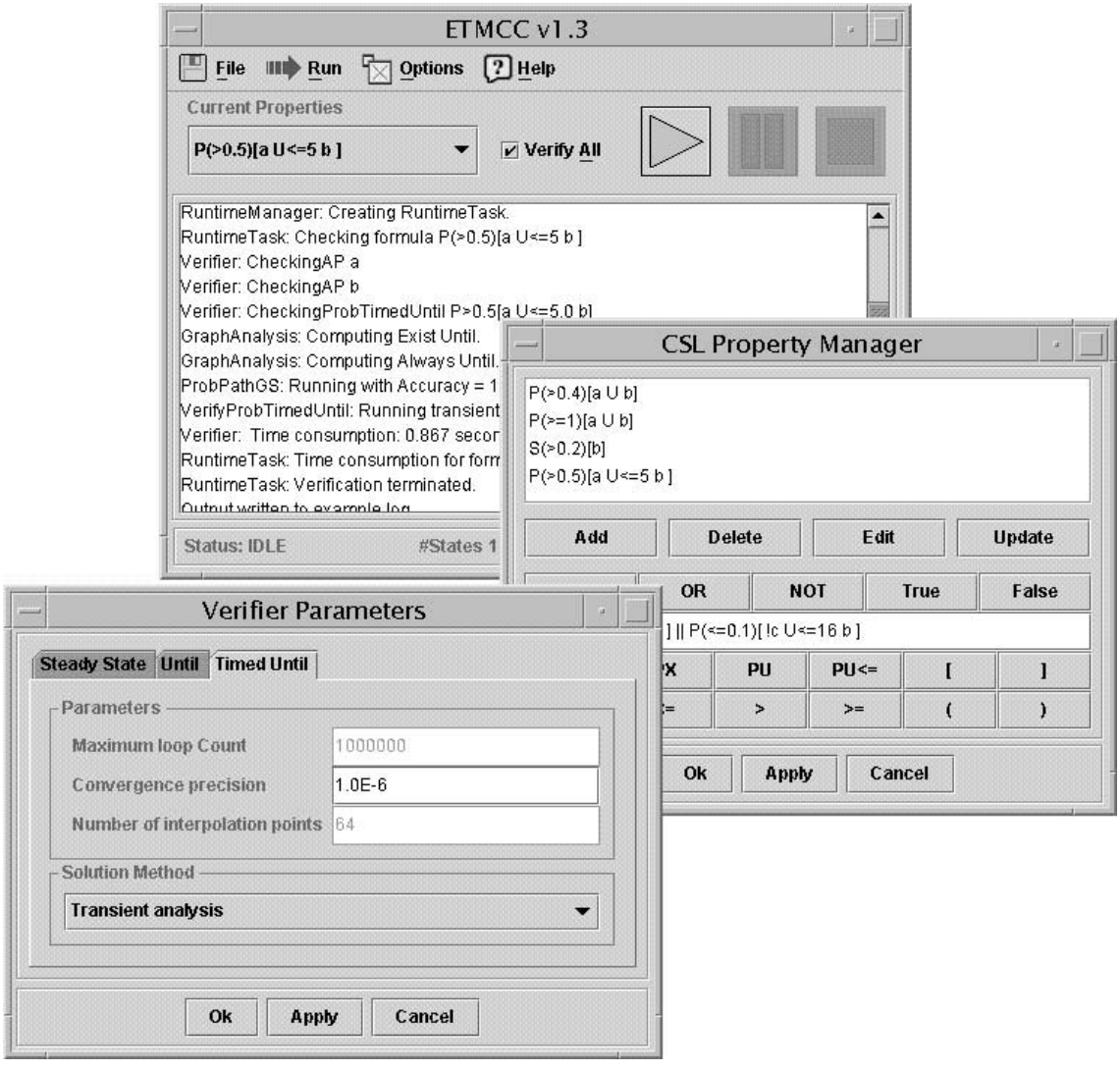


Fig. 6. User interface of E ⊢ MC^2

jects that encapsulate the verification sub-algorithms. These objects, in turn, use the Boolean and/or numerical engine. For instance, checking $\mathcal{P}_{\leqslant p}(\Phi\,\mathcal{U}\,\Psi)$ involves a pre-processing step (as mentioned above) that isolates states satisfying the Boolean $\exists(\Phi\,\mathcal{U}\,\Psi)$ and $\forall(\Phi\,\mathcal{U}\,\Psi)$. The results of this step are passed to the numerical engine that computes the remaining non-trivial probabilities.

Numerical Engine is the numerical analysis component of $\mathrm{E}\vdash\mathrm{MC}^2$. It computes the solution of linear systems of equations, offering a selection of well-established numerical algorithms. For the solution of Volterra integral equation systems, the user can choose either uniformization, or iterative numerical integration, as explained above. A variety of parameters of the algorithms can be influenced by the user via the GUI, such as the accuracy ε, the 'maximum loop count' (the maximal number of iterations before termination), or the number N of interpolation points used for the piecewise integration.

Boolean Engine is the component that supports graph algorithms, for instance, to compute the BSCCs in case of steady-state properties, and standard model checking algorithms for CTL-like until-formulas. The latter algorithms are not only used as a pre-processing phase of checking until-formulas (as explained above), but they also take care of instances of the special cases $\mathcal{P}_{\geqslant 1}(\varphi)$ and $\mathcal{P}_{>0}(\varphi)$ where the numerical analysis tends to produce 'wrong' results (such as $0.99999\ldots$ rather than 1.0) due to machine imprecision. As mentioned in the introduction, such qualitative properties can be verified without the need for numerical recipes.

State Space Manager represents DTMCs and CTMCs in a uniform way. In fact, it provides an interface between the various checking and analysis components of $\mathrm{E}\vdash\mathrm{MC}^2$ and the way in which DTMCs and CTMCs are actually represented. This eases the use of different, possibly even symbolic state space representations. It is designed to support input formats of various kinds, by means of a simple plug-in-functionality (realized via JAVA's dynamic class loading capability). It maintains information about the validity of atomic propositions and of sub-formulas for each state, encapsulated in a 'Sat' sub-component. 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 [61]. The rate matrix $\boldsymbol{R}$ (and its transpose $\boldsymbol{R}^T$) are stored, while the entries of $\boldsymbol{E}$ and $\boldsymbol{P}$ are computed on demand. All real values are stored in the IEEE 754 floating point format with double precision (64 bit).

3.4 $\mathrm{E}\vdash\mathrm{MC}^2$ *as a model checker for discrete time*

So far we have described $\mathrm{E}\vdash\mathrm{MC}^2$ as a model checker for CTMCs. However, it is equally well suited as a model checker for DTMCs with respect to the logic PCTL [34]. This logic arises from CSL by omitting the time-bounded until operator and the steady-state operator. The crucial property that makes $\mathrm{E}\vdash\mathrm{MC}^2$ a DTMC model checker is that it is possible to consider a DTMC as a CTMC without change. To make this more precise, we introduce (labelled) DTMCs formally.

Let AP be a fixed, finite set of atomic propositions. We define a labelled discrete-time Markov chain as a tuple $\mathcal{M} = (S, \boldsymbol{P}, L)$ where:

- S is a finite set of *states*,
- $\boldsymbol{P} : S \times S \to [0,1]$ is the *transition probability matrix*, such that for all $s \in S$, $\sum_{s' \in S} \boldsymbol{P}(s,s') \in \{\,0, 1\,\}$, and
- $L : S \to 2^{AP}$ is the *labelling* function which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions $a \in AP$ that are valid in s.

Two important observations can now be made:

- If $\mathcal{M} = (S, \boldsymbol{R}, L)$ is a CTMC, and $\boldsymbol{P}$ is the probability matrix associated with $\boldsymbol{R}$ then $\boldsymbol{P}$ defines a DTMC $(S, \boldsymbol{P}, L)$, usually called the *embedded* DTMC.
- If, in addition $s \in S$, and Φ is a PCTL formula, then

  $$s \models \Phi \quad \text{if and only if} \quad s \models_{PCTL} \Phi$$

  where $\models_{PCTL}$ is the original satisfaction relation of [34] on the embedded DTMC $(S, \boldsymbol{P}, L)$.

As a consequence, in order to model check a given DTMC $(S, \boldsymbol{P}, L)$ with transition probability matrix $\boldsymbol{P}$, we have to feed a CTMC $(S, \boldsymbol{R}, L)$ into the model checker, that is defined in such a way that $\boldsymbol{P}$ is the matrix associated with the rate matrix $\boldsymbol{R}$. There is some freedom in fixing this matrix $\boldsymbol{R}$ (in particular in the vector $\boldsymbol{E}$), but the easiest is to choose $\boldsymbol{R} = \boldsymbol{P}$. The reader is invited to check that for a CTMC $(S, \boldsymbol{P}, L)$, the embedded DTMC is $(S, \boldsymbol{P}, L)$. Thus, in summary, DTMCs can be fed into the model checker $\mathrm{E}\vdash\mathrm{MC}^2$ *as if they were* CTMCs. An optional check is provided by the tool to ensure that a loaded project describes a DTMC and that all formulas are within PCTL.

4 Application case studies

In this section, we report on experiences with $\mathrm{E}\vdash\mathrm{MC}^2$ in the context of model-checking two Markov chain models that have been generated from different high-level formalisms, namely queueing networks and generalized stochastic Petri nets. Based on these experiments we assess the sensitivity of the model checker with respect to various parameters. We ran the experiments on a 300 MHz SUN Ultra 5/10 workstation with 256 MB memory under the Solaris 2.6 operating system. In the case studies we solve linear systems of equations by means of the Gauss–Seidel method. All recorded execution times are wall clock times.

4.1 A tandem queue system

As a first, simple example we consider a queueing network (with blocking) taken from [43]. It consists of a $M/Cox_2/1$-queue sequentially composed with a $M/M/1$-queue, see Fig. 7. For a thorough introduction to networks of queues we refer to [24]. Both queueing stations have a capacity of c jobs, $c > 0$. Jobs arrive at the first queueing station with rate λ. The server of the first station executes jobs in one or two phases; that is, with probability $b_1 = 1 - a_1$ a job is served with rate μ_1 only, and with probability a_1, the job has to pass an additional phase with rate μ_2. Once served, jobs leave the first station, and are queued in the second station where service takes place with rate κ. In case the second queueing station is fully occupied, i.e., its server is busy and its queue is full, the first station is said to be blocked. Note that in this situation, the second phase of the first server is blocked and the first server can only pass a job that just finished the first phase to the second phase (which happens with probability a_1), but the "bypass" of the second phase is also blocked.

The CTMC of the tandem network for $c = 1$ is depicted in Fig. 8. State labels are of the form (n_1, p, n_2) where n_1 ($n_1 \leqslant c$) indicates the number of jobs in the first station, $p \in \{0, 1, 2\}$ the status of servicing in the first station (0 means that no service is going on, 1 means that a job is in the first phase, and 2 means that a job is in the second phase) and n_2 ($n_1 \leqslant c$) indicates the number of jobs in the second station.

Parameters for the experiments. For the experiments we take the following values for the parameters of the queue: $\lambda = 4 \cdot c$, $\mu_1 = 2$, $\mu_2 = 2$, $\kappa = 4$, and $a_1 = 0.1$. Note that the arrival rate equals the maximal service delay in the first station for $c = 1$. We consider the following configurations: $c = 2$, which amounts to 15 states and 33 transitions, $c = 5$, i.e., 66 states and 189 transitions and $c = 20$, i.e., 861 states and 2851 transitions. The following atomic propositions are considered:

- *block* is valid iff each queueing system contains c jobs and the first station is serving in the second phase,
- *fst* is valid iff no new arriving job (with rate λ) can be accepted anymore, because the first queue is entirely populated.
- *snd* is valid iff the first queueing station is blocked, because the second queue is entirely populated.

It should be noticed that *block* characterizes a single state, and hence, for large c identifies a rare event, a situation that appears with very low probability. For example, in Fig. 8 state 121 satisfies *block*, states 110, 120, 111, and 121 satisfy *fst*, and states 111 and 121 satisfy *snd*.

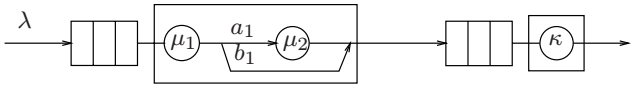


Fig. 7. A simple tandem network with blocking [43]

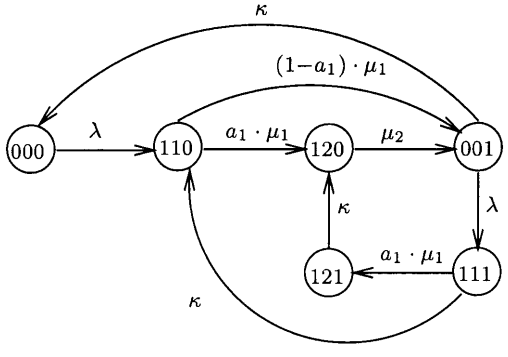


Fig. 8. CTMC of the tandem network for $c = 1$

Steady-state properties. The following steady-state properties are checked: $\mathcal{S}_{\bowtie p}(block)$, $\mathcal{S}_{\bowtie p}(fst)$, and $\mathcal{S}_{\bowtie p}(\mathcal{P}_{\bowtie q}(X\, snd))$, for arbitrary p and q. The latter property is valid if the steady-state probability to be in a state that can reach a state in which the first queueing station is blocked in a single step with probability $\bowtie q$ satisfies $\bowtie p$. We do not instantiate p and q, as the execution times and computed probabilities will be the same for all p and q (except for the extremal cases 0 and 1); only the comparison with the bounds might lead to a different outcome. Thus, $p, q \in]0, 1[$. For the steady-state properties we vary the accuracy ε of the computed probability, which is a parameter to the model checker. The results are listed in Table 1. The third column indicates the number of iterations needed to reach the result with the desired accuracy. Recall that the model checker checks the validity of CSL-formulas for all states in the CTMC.

Real-time probabilistic path properties. The verification times for probabilistic path-formulas can be quite different, as we will see. Using this small example we analyze the dependency of the verification time on:

- Solution method (numerical integration versus transient analysis),
- Precision (the number of interpolation points in case of integration), and
- The structure of the formula under consideration.

The following probabilistic path properties are used for these purposes:

- $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t}\, block)$, referring to the probability of having a fully occupied tandem network within t time units,
- $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} fst)$, referring to the probability of an entirely populated first queueing station within t time units, and
- $\mathcal{P}_{\bowtie p}(snd\, \mathcal{U}^{\leqslant t}\, \neg snd)$, which refers to the probability of leaving a situation in which the second queue is entirely populated.

All path-properties are checked with accuracy $\varepsilon = 10^{-6}$. We vary the time-span t over 2, 10, 100, and 1000 (for transient analysis).

Table 1. Statistics for checking steady-state properties on the tandem queue

# states	ε	# iterations	$\mathcal{S}_{\bowtie p}(block)$ time (in *sec*)	$\mathcal{S}_{\bowtie p}(fst)$ time (in *sec*)	$\mathcal{S}_{\bowtie p}(\mathcal{P}_{\bowtie q}(X\,snd))$ time (in *sec*)
15	10^{-4}	62	0.012	0.012	0.013
	10^{-6}	107	0.016	0.017	0.016
($c=2$)	10^{-8}	146	0.017	0.018	0.019
66	10^{-4}	77	0.028	0.028	0.065
	10^{-6}	121	0.041	0.042	0.076
($c=5$)	10^{-8}	159	0.048	0.085	0.181
861	10^{-4}	74	0.569	0.498	1.567
	10^{-6}	118	0.644	0.643	1.935
($c=20$)	10^{-8}	158	0.811	0.778	2.369

Numerical integration. The number of interpolation points for the piecewise integration is varied from 64 up to 1000. The results for numerical integration with $c=2$ are listed in the upper part of Table 2. Note the difference in computation time for the different properties. Whereas $\mathcal{P}_{\bowtie p}(snd\,\mathcal{U}^{\leqslant t}\,\neg snd)$ can be checked rather quickly, calculating the probability for reaching a *fst*-state within a certain time bound, and – in particular – until reaching a *block*-state takes significantly more time. Since the CTMC is strongly connected, a *block*- or *fst*-state can (eventually) be reached from any other state, and hence for all states the probability for reaching these states within time t must be calculated. In addition, the probability of reaching the single *block*-state is low, especially for larger c, and quite a number of iterations are needed in that case to obtain results with the desired precision. Since there are several *fst*-states in the CTMC, this effect is less apparent for $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} fst)$. For the last property (last two columns), probabilities need only be computed for *snd*-states rather than for all states, and convergence is reached rather quickly as the real probabilities are close to 1. These effects become more apparent when increasing the state space. This is reflected by the results in the lower part of Table 2 where we considered a CTMC of almost 1000 states.

Uniformization. The verification times for checking the probabilistic path-formulas while exploiting the reduction to transient analysis (i.e., uniformization rather than numerical integration) are listed in Table 3. The first observation is that run-times tend to be less dependent on t, especially for larger t. This is well reflected by the run-times for the CTMC of 861 states, where run-times become constant from a certain t onwards. This

Table 2. Statistics for checking probabilistic path-formulas on the tandem queue with numerical integration

# states	t	# interpolation points	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} block)$ # iter.	time (in *sec*)	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} fst)$ # iter.	time (in *sec*)	$\mathcal{P}_{\bowtie p}(snd\,\mathcal{U}^{\leqslant t}\,\neg snd)$ # iter.	time (in *sec*)
15	2	64	18	2.497	11	1.045	4	0.144
		128	18	9.762	11	4.082	4	0.566
		256	18	22.19	11	16.30	4	2.248
		512	18	156.2	11	69.04	4	9.067
($c=2$)		1000	18	602.3	11	248.6	4	34.27
15	10	64	45	6.506	12	1.140	4	0.145
		128	43	24.00	12	4.575	4	0.568
		256	43	52.85	12	17.94	4	2.309
		512	43	383.1	12	75.13	4	8.994
($c=2$)		1000	43	1433	12	274.9	4	34.38
15	100	64	472	104.6	12	2.133	4	0.229
		128	344	284.9	12	7.682	4	0.817
		256	285	958.1	12	31.07	4	3.361
		512	260	3582	12	123.8	4	13.51
($c=2$)		1000	252	13 201	12	493.8	4	51.49
861	2	64	36	448.3	29	347.3	21	9.608
		128	36	1773	29	1336	21	38.90
		256	36	7028	29	5293	21	150.5
($c=20$)		512	36	28 189	29	21 914	21	600.1

Table 3. Statistics for checking probabilistic path-formulas on the tandem queue with transient analysis

# states	t	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} block)$ time (in *sec*)	$\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} fst)$ time (in *sec*)	$\mathcal{P}_{\bowtie p}(snd\,\mathcal{U}^{\leqslant t}\,\neg snd)$ time (in *sec*)
15 ($c=2$)	2	0.013	0.008	0.005
	10	0.039	0.006	0.005
	100	0.052	0.006	0.006
	1000	0.055	0.007	0.008
861 ($c=20$)	2	0.453	0.049	0.122
	10	0.559	0.098	0.114
	100	0.577	0.129	0.294
	1000	0.644	0.186	0.784

phenomenon is caused by the on-the-fly steady-state detection while carrying out the uniformization. Thus, if the CTMC reaches a steady state before the bound t is reached, no further computations are needed, as the state probability mass will not be affected anymore. A second observation is that, as for the numerical integration technique, the run-times are quite dependent on the formula at hand. This effect is caused by the fact that the transformation (from $\mathcal{M}$ to $\overline{\mathcal{M}}$) of the CTMC prior to the uniformization depends on the path-formula. Thus, the size of the resulting CTMC used for the uniformization differs per formula. Checking $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} block)$ does not give rise to any reduction of the CTMC (only the single *block* state becomes absorbing), while the transformation for the other two path-formulas leads to a significant reduction in size. A final observation is that uniformization is much faster than numerical integration, for the tandem queue even up to a factor 500–1000.

4.2 A cyclic server polling system

In this section, we consider a cyclic server polling system consisting of d stations and a server, modeled as a GSPN.[3] The example is taken from [48], where a detailed explanation can be found. For $d=2$, i.e., a two-station polling system, the GSPN model is depicted in Fig. 9. For a d-station polling system, the Petri net is extended in the obvious way. Place $idle_i$ represents the condition that station i is idle, and place $busy_i$ represents the condition that station i has generated a job. The server visits the stations in a cyclic fashion. After polling station i (place $poll_i$), the server serves station i (place $serve_i$) and then proceeds to poll the next station. The times for generating a message, for polling a station, and for serving a job are all distributed exponentially with parameters λ_i, γ_i, and μ_i, respectively. In case the server finds station i idle, the service time is zero which is modeled by the immediate transition $skip_i$ and the inhibitor arc from place $busy_i$ to transition $skip_i$. In this study we consider polling systems with $d=$ 3, 5, 7, and 10 stations (as in [48]), and the case $d=13$. The corresponding CTMCs have 36, 240, 1344, 15 360, and 163 840 states (84, 800, 5824, 89 600, and 1 286 144 transitions). The polling system is assumed to be symmetric, i.e., all λ_i have the same numerical value, and the same applies to $\gamma_i=200$ and $\mu_i=1$. We set $\lambda_i=\mu_i/d$.

In the context of GSPNs, it is rather natural to identify the set of places that possess a token in a given marking – a state of our CTMC – with the set of atomic propositions valid in this state. Based on these atomic propositions, we check the following properties on the polling system: $\neg(poll_1 \wedge poll_2)$, stating that the server never polls both stations at the same time; $\mathcal{P}_{\bowtie p}(\neg serve_2\,\mathcal{U}\,serve_1)$, i.e., with probability $\bowtie p$ station 1 will be served before station 2; $busy_1 \Rightarrow \mathcal{P}_{\geqslant 1}(\Diamond poll_1)$, so once station 1 has become busy, it will eventually be polled; $busy_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} poll_1)$, once station 1 has become busy, with probability $\bowtie p$ it will be polled within t time units (we let $t=1.5$). The following steady-state formulas are considered: $\mathcal{S}_{\bowtie p}(busy_1 \wedge \neg serve_1)$, which says that the probability of station 1 being waiting for the server is $\bowtie p$; and $\mathcal{S}_{\bowtie p}(idle_1)$, stating that the probability of station 1 being idle is $\bowtie p$. As before, $p \in]0,1[$. All path-properties were

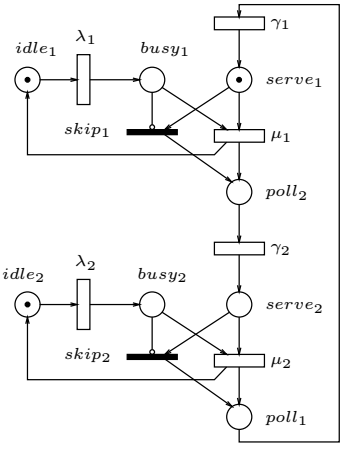


Fig. 9. The cyclic server polling system with 2 stations [48]

[3] We refer to [1] for details on the semantics of GSPNs. In particular, the existence of immediate transitions (the black transitions) leads to so-called vanishing markings in the reachability graph which, however, can be eliminated easily. Our model checker works on the resulting tangible reachability graph which is isomorphic to a CTMC.

Table 4. Statistics for the verification of the polling system

d	# states	$\neg(poll_1 \wedge poll_2)$ time (in *sec*)	$\mathcal{P}_{\bowtie p}(\neg serve_2 \,\mathcal{U}\, serve_1)$ time (in *sec*)	$busy_1 \Rightarrow \mathcal{P}_{\geqslant 1}(\Diamond poll_1)$ time (in *sec*)
3	36	0.002	0.031	0.005
5	240	0.002	0.171	0.009
7	1344	0.005	2.460	0.011
10	15 360	0.020	43.820	0.080
13	163 840	0.040	650.570	1.140

		$busy_1 \Rightarrow \mathcal{P}_{\bowtie p}(\Diamond^{\leqslant 1.5} poll_1)$		
		numerical integration		transient analysis
d	# states	# iter.	time (in *sec*)	time (in *sec*)
3	36	8	2.308	0.068
5	240	12	30.92	0.233
7	1344	14	308.5	0.840
10	15 360	18	7090	11.730
13	163 840	—	—	172.170

		$\mathcal{S}_{\bowtie p}(busy_1 \wedge \neg serve_1)$		$\mathcal{S}_{\bowtie p}(idle_1)$	
d	# states	# iter.	time (in *sec*)	# iter.	time (in *sec*)
3	36	39	0.044	39	0.038
5	240	61	0.103	61	0.102
7	1344	80	0.677	80	0.658
10	15 360	107	11.010	107	9.770
13	163 840	130	196.980	130	199.430

checked with accuracy $\varepsilon = 10^{-6}$, and the number of interpolation points for numerical integration was set to 64. The steady-state properties were checked for $\varepsilon = 10^{-8}$. The execution times for checking these properties are given in Table 4.

4.3 Assessment of the tool

General. From the results of our case studies we observe that checking CSL-formulas consisting of just atomic propositions and logical connectives is very fast. The verification time to check steady-state properties, and unbounded and time-bounded until-formulas is proportional in the state-space of the CTMC. For the polling system with about 1.6 million states, checking such formulas takes a few minutes. Measurements have shown that the performance of our tool's steady-state solution algorithm is comparable to the one of TIPPtool [41] which is based on a sophisticated sparse matrix library implemented in C. The same applies to our transient analysis algorithm. Furthermore, the tool is quite memory efficient, e.g., the 15 360-state and 163 840-state cyclic polling system models only take 1.48 and 20.8 Mb, respectively. These numbers include storage of bit-vectors needed for state-labellings for sub-formulas. Thus, memory consumption and run-times of $\mathrm{E} \vdash \mathrm{MC}^2$ allow us to verify CTMCs of up to a few million states.

Time-bounded until. $\mathrm{E} \vdash \mathrm{MC}^2$ incorporates two algorithms for checking time-bounded until-formulas: numerical integration and transient analysis. Our empirical results show that transient analysis clearly outperforms the use of numerical integration – see Table 4 (middle) – both in verification times and in accuracy of numerical results (see next paragraph).

For numerical integration, each iteration in the piecewise integration takes $\mathcal{O}(N^2 \cdot K)$ time in the worst case, where K is the number of transitions and N is the number of interpolation points. The number of iterations strongly depends on the required accuracy, the number of states, and the structure of the CTMC. The verification time also depends on the formula under consideration, and the time-span (i.e., the parameter t). For instance, checking $\mathcal{P}_{\bowtie p}(\Diamond^{\leqslant t} \Psi)$ involves a computation for each state that has a non-zero and non-trivial probability of reaching a Ψ-state, while checking $\mathcal{P}_{\bowtie p}(a \,\mathcal{U}^{\leqslant t}\, \Psi)$ only involves a computation for the a-labelled states (of this set). This effect is shown in Table 2.

For determining the states satisfying $\mathcal{P}_{\bowtie p}(\Phi \,\mathcal{U}^{\leqslant t}\, \Psi)$ using uniformization, $\mathcal{O}(K \cdot q \cdot t)$ time is needed in the worst case, where t is the time-bound of the formula and q is the uniformization rate of the CTMC under consideration [50]. Due to a built-in steady-state detection, for large t the run-times will be more or less constant. An important advantage of transient analysis is that the number of computation steps can be determined prior

to performing any computations, based on the required accuracy. This is much harder, if possible at all, to incorporate in the numerical integration approach.

Accuracy of numerical results. In order to assess the numerical accuracy and the execution times of the algorithms for time-bounded until, we used E ⊢ MC^2 to compute the cumulative distribution function of the Erlang k-distribution, that is, a convolution of k identical exponential distributions. The model is shown in Fig. 10. It has k+1 states, where states 1 through k are labelled with a and state k+1 is labelled with b. Transitions lead from state i to i+1 ($i = 1, \ldots, k$) with rate λ. We checked the formula $\mathcal{P}_{\bowtie p}(a\,\mathcal{U}^{\leqslant t}\,b)$ which for state 1 yields the value of the Erlang-k cumulative distribution function for t, i.e.,

$$F_{E_k}(t) = Prob(E_k \leqslant t) = 1 - \mathrm{e}^{-\lambda t}\left(\sum_{i=0}^{k-1}\frac{(\lambda t)^i}{i!}\right).$$

Table 5 shows the results of this study. For a varying number of Erlang phases k and different time instants t, the 'exact' result (computed according to the above closed formula) and the results of our two algorithms are shown. The value of λ was set to 1.0.

For the numerical integration algorithm, it can be observed that for small k the results are quite accurate, even for a small number of interpolation points N (the accuracy for the termination condition was set to 10^{-6}). The accuracy further improves as N is increased, as expected. For $k \geqslant 100$ and small N, however, the accuracy of the results is unacceptable, while for large N the runtime becomes excessive.

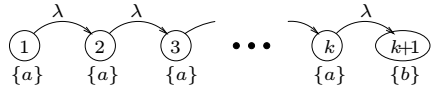


Fig. 10. State diagram of the Erlang-k distribution

The picture is much more favorable for the transient analysis algorithm, where the accuracy parameter was set to 10^{-6}. The precision of the results is extremely good in all cases, and the execution times are almost negligible, even for the $k = 200$ case. In summary, transient analysis yields much better results than numerical integration and is also dramatically faster.

Although this Erlang example is a bit artificial, some of its features, especially the sparseness of its transition rate matrix, are shared by quite a lot of examples from real applications. If the matrix $\boldsymbol{U}$ is sparse, the iterative matrix exponentiation $\boldsymbol{U}^k$, which is the most expensive step within the uniformization procedure, involves only a few multiplications and is therefore very fast.

Accuracy of verification results. Another issue – which is inherently present in all model checking approaches that rely on numerical recipes – is to avoid wrong outcomes of comparisons with a probability bound p in a sub-formula, that is then propagated upwards. Because round-off and truncation errors cannot be avoided (due to machine imprecision), this effect can happen if the computed value is very close to the bound p. For the extremal probability bounds (i.e., bounds > 0 and $\geqslant 1$), we have circumvented this problem by applying the standard model checking algorithms for $\forall$ and $\exists$ as in [34]. Furthermore, we intend to use a three-valued logic such that the tool can avoid potentially wrong results, and answers 'don't know' in case some calculated (i.e., approximated) probability is within some tolerance

Table 5. Accuracy of the algorithm for bounded until (Erlang-k example)

k	t	'exact'	transient analysis		numerical integration N=64		numerical integration N=512	
5	1	0.003660	0.003659	0.00 sec	0.003432	0.14 sec	0.003630	7.1 sec
	5	0.559507	0.559506	0.00 sec	0.545953	0.14 sec	0.557795	9.7 sec
	10	0.970747	0.970747	0.00 sec	0.967667	0.15 sec	0.970375	10.6 sec
10	2	0.000046	0.000046	0.00 sec	0.000041	0.40 sec	0.000046	33.6 sec
	10	0.542070	0.542070	0.00 sec	0.523807	0.43 sec	0.539645	34.4 sec
	20	0.995005	0.995004	0.00 sec	0.994028	0.43 sec	0.994889	33.9 sec
20	4	0.000000	0.000000	0.01 sec	0.0	1.3 sec	0.0	110 sec
	20	0.529742	0.529742	0.00 sec	0.509592	1.6 sec	0.526387	132 sec
	40	0.999823	0.999823	0.00 sec	0.999755	1.6 sec	0.999816	136 sec
50	10	0.000000	0.000000	0.01 sec	0.0	5.5 sec	—	
	50	0.518808	0.518808	0.01 sec	0.545528	9.9 sec	0.514448	806 sec
	100	0.999999	0.999999	0.01 sec	0.999999	9.9 sec	—	
100	20	0.000000	0.000000	0.03 sec	0.0	17.7 sec	—	
	100	0.513298	0.513298	0.02 sec	0.764289	38.1 sec	0.511909	3200 sec
	200	0.999999	0.999999	0.02 sec	0.999999	40.0 sec	—	
200	40	0.000000	0.000000	0.20 sec	—		—	
	200	0.509403	0.509403	0.10 sec	—		—	
	400	1.0	1.0	0.05 sec	—		—	

to a probability bound p occurring in a (sub-)formula to be checked.

5 Related work

Model checking probabilistic systems. Methods to verify a DTMC (or the like) against a linear-time temporal logic (LTL) formula (sometimes specified as a Büchi automaton) have been considered by, for example, [26, 57, 63]. The basis of these works is the (non-trivial) reduction of the model checking problem to the computation of the probabilities to reach a certain set of states (mostly, BSCCs). [25] describes an algorithm for checking whether a DTMC satisfies a probabilistic LTL-formula.

As stated in the introduction, PCTL model checking has been developed further by Hansson and Jonsson [34]. In a similar way to CTL* containing both LTL and CTL, the logic PCTL* contains both LTL and PCTL. PCTL* model checking is studied in [6, 14, 17]. Its basic idea is the reduction to the verification of quantitative LTL properties.

For work on the branching-time model checking of Markov decision processes we refer to [13, 14]. Here, non-determinism is resolved by adversaries. The model checking of until-formulas reduces to the computation of a minimum (or maximum) probability, depending whether one quantifies over all or some adversaries, respectively.

Model checking real-time probabilistic systems. A qualitative model checking algorithm for a (continuous) probabilistic variant of timed automata, finite-state automata equipped with real-time clocks, has been proposed in [4]. This technique is based on the so-called region technique, a finite partition of the infinite continuous-time domain tailored to the property and model under consideration. Recently, this approach has been adopted for quantitative model checking of (discrete) probabilistic timed automata [51] and a continuous variant thereof [52]. The verification of real-time properties over MDPs has been considered in, for example, [2, 15]. Note that our approach for CTMCs is not based on region-like constructions.

Tools for verifying probabilistic systems. Most work on the verification of probabilistic systems has been focused on theory. Tool development has received far less attention. Notable exceptions are the earlier works by Martin [53] on the tool VOPP (Verification tOol for Probabilistic Processes) and by Fredlund [31] on the tool TPWB (Timed Probabilistic Workbench). VOPP is a dedicated command-line tool (implemented in C) that supports the verification of equivalences between DTMCs and a tableau-based model checking procedure for a probabilistic modal logic by L. Christoff [19, 20]. TPWB is a command-line tool based on the well-known Concurrency Workbench [23] for CCS and is implemented in Standard ML. It is based on the model checking of Timed PCTL properties [33] over discrete-time probabilistic CCS processes. These tools use sparse-matrix representations. The symbolic model checker PROBVERUS for verifying PCTL over DTMCs has recently been reported in [35]. It is built upon the VERUS verification tool and uses multi-terminal BDDs (MTBDDs) as data structure. The theoretical foundations of PROBVERUS have been laid down in [10]. The symbolic model checker PRISM (PRobabIlistic Symbolic Model checker) for MDPs has been recently presented in [3]. It uses MTBDDs as data structures and the algorithms of [14]. Note that all these tools are based on *discrete-time* probabilistic models. An earlier version of our tool has been reported in [39].

6 Conclusion

In this paper, we have presented a model checker for (state labelled) discrete and continuous-time Markov chains. We reported on the structure of the tool, and on experiments using the model checker to verify CTMCs derived from high-level formalisms such as stochastic Petri nets and queueing networks. As far as we know, $\mathrm{E} \vdash \mathrm{MC}^2$ is the first implementation of a bridge between such high-level specification formalisms for CTMCs and model checking.

$\mathrm{E} \vdash \mathrm{MC}^2$ is a prototype. In particular, for the moment, it does not use symbolic, i.e., (MT)BDD-based, data structures. Although our own experience (and of others, see [32]) has shown that very compact encodings of Markov chains are possible with MTBDDs and similar data structures [43], and symbolic model checking algorithms for CTMCs do exist [12], we favor a separation of concerns: to our belief the issues of numerical stability, convergence, accuracy, and efficiency are worth studying in isolation, without interference of the (sometimes unpredictable) effects of BDD-based computations. In addition, none of the high-level modeling tools for generating CTMCs uses BDD-based data structures, as far as we know.

Our decision to implement the model checker $\mathrm{E} \vdash \mathrm{MC}^2$ in JAVA turned out to be a good choice. In particular it allowed us to develop an easy-to-use user interface along with the model checker engine. In addition, the numerical computations have very good performance in JAVA; e.g., the computation of steady-state properties is comparable to (optimized) existing C implementations.

$\mathrm{E} \vdash \mathrm{MC}^2$ currently allows to model-check CSL-formulas over CTMCs of up to a few million states. Due to the fact that the validity of CSL-formulas is preserved under lumpability [11] – an equivalence relation on Markov chains to aggregate state spaces that can be viewed as a continuous probabilistic variant of strong bisimulation – it is possible to minimize the CTMC under consideration (with respect to lumpability) prior to carrying out the model checking. In this way, state spaces of up to hun-

dreds of millions of states can be handled; see the analysis of the telephone system in [42].

For further information and to download $\mathrm{E} \vdash \mathrm{MC}^2$, the reader is invited to consult our web-page at:

`www7.informatik.uni-erlangen.de/etmcc`

Acknowledgements. The authors thank Lennard Kerber (Erlangen) for his contribution to assessing the accuracy of the tool output, and Christel Baier (Bonn) and Boudewijn Haverkort (Aachen) for their valuable contributions and discussions. Holger Hermanns is supported by the Netherlands Organization for Scientific Research (NWO). Joachim Meyer-Kayser is supported by the German Research Council (DFG) under HE 1408/6-1. The co-operation between the research groups in Twente and Erlangen-Nürnberg takes place in the context of the project Validation of Stochastic Systems which is funded by the Dutch NWO and the German DFG.

References

1. Ajmone Marsan, M., Conte, G., Balbo, G.: A class of generalized stochastic Petri nets for the performance evaluation of multiprocessor systems. ACM Trans Comp Syst 2(2):93–122, 1984
2. de Alfaro, L.: How to specify and verify the long-run average behavior of probabilistic systems. In: Proc. 13th IEEE Symposium on Logic in Computational Science, pp. 454–465, IEEE CS, 1998
3. de Alfaro, L., Kwiatkowska, M.Z., Norman, G., Parker, D., Segala, R.: Symbolic model checking for probabilistic processes using MTBDDs and the Kronecker representation. In: Graf, S., Schwartzbach, M. (eds), Tools and Algorithms for the Analysis and Construction of Systems, Lecture Notes in Computer Science, vol. 1785. Springer, Berlin Heidelberg New York, 2000, pp. 395–410
4. Alur, R., Courcoubetis, C., Dill, D.: Model-checking for probabilistic real-time systems. In: Albert, J.L., Monien, B., Rodríguez-Artalejo, M. (eds), Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 510. Springer, Berlin Heidelberg New York, 1991, pp. 115–126
5. Alur, R., Courcoubetis, C., Dill, D.: Model checking in dense real-time. Inf Comput 104:2–34, 1993
6. Aziz, A., Singhal, V., Balarin, F., Brayton, R., Sangiovanni-Vincentelli, A.: It usually works: the temporal logic of stochastic systems. In: Wolper, P. (ed), Computer-Aided Verification, Lecture Notes in Computer Science, vol. 939. Springer, Berlin Heidelberg New York, 1995, pp. 155–165
7. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Verifying continuous time Markov chains. In: Alur, R., Henzinger, T.A. (eds), Computer-Aided Verification, Lecture Notes in Computer Science, vol. 1102. Springer, Berlin Heidelberg New York, 1996, pp. 269–276
8. Aziz, A., Sanwal, K., Singhal, V., Brayton, R.: Model checking continuous time Markov chains. ACM Trans Comput Logic 1(1):162–170, 2000
9. Baier, C.: On algorithmic verification methods for probabilistic systems. Habilitation thesis, Univ. of Mannheim, 1999
10. Baier, C., Clarke, E., Hartonas-Garmhausen, V., Kwiatkowska, M., Ryan, M.: Symbolic model checking for probabilistic processes. In: Degano, P., Gorrieri, R., Marchetti-Spaccamela, A. (eds), Automata, Languages and Programming, Lecture Notes in Computer Science, vol. 1256. Springer, Berlin Heidelberg New York, 1997, pp. 430–440
11. Baier, C., Haverkort, B.R., Hermanns, H., Katoen, J.-P.: Model checking continuous-time Markov chains by transient analysis. In: Emerson, E.A., Sistla, A.P. (eds), Computer-Aided Verification, Lecture Notes in Computer Science, vol. 1855. Springer, Berlin Heidelberg New York, 2000, pp. 358–372
12. Baier, C., Katoen, J.-P., Hermanns, H.: Approximate symbolic model checking of continuous-time Markov chains. In: Baeten, J., Mauw, S. (eds), Concurrency Theory, Lecture Notes in Computer Science, vol. 1664. Springer, Berlin Heidelberg New York, 1999, pp. 146–162
13. Baier, C., Kwiatkowska, M.Z.: On the verification of qualitative properties of probabilistic processes under fairness constraints. Inf Proc Lett 66(2):71–79, 1998
14. Baier, C., Kwiatkowska, M.Z.: Model checking for a probabilistic branching time logic with fairness. Distrib Comput 11:125–155, 1998
15. Beauquier, D., Slissenko, A.: Polytime model checking for timed probabilistic computation tree logic. Acta Inf 35:645–664, 1998
16. Bernardo, M., Gorrieri, R.: A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities, and time. Theor Comp Sci 202:1–54, 1998
17. Bianco, A., de Alfaro, L.: Model checking of probabilistic and nondeterministic systems. In: Thiagarajan, P.S. (ed), Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 1026. Springer, Berlin Heidelberg New York, 1995, pp. 499–513
18. Changuion, B., Davies, I., Nelte, M.: DaNAMiCS – a Petri Net Editor. `http://www.cs.uct.ac.za/Research/DNA/DaNAMiCS/`
19. Christoff, I., Christoff, L.: Reasoning about safety and liveness properties for probabilistic systems. In: Shyamasundar, R.K. (ed), Foundations of Software Technology and Theoretical Computer Science, Lecture Notes in Computer Science, vol. 652. Springer, Berlin Heidelberg New York, 1992, pp. 342–355
20. Christoff, L.: Specification and verification methods for probabilistic processes. PhD thesis, Technical Report DoCs 93/37, Uppsala University, 1993
21. Clarke, E., Emerson, E., Sistla, A.: Automatic verification of finite-state concurrent systems using temporal logic specifications. ACM Trans Program Lang Syst 8:244–268, 1998
22. Clarke, E., Grumberg, O., Peled, D.: Model checking. MIT, Cambridge, Mass., USA, 1999
23. Cleaveland, W.R., Parrow, J., Steffen, B.: The concurrency workbench: a semantics-based tool for the verification of concurrent systems. ACM Trans Program Lang Syst 15(1):36–72, 1993
24. Conway, A.E., Georganas, N.D.: Queueing networks – exact computational algorithms. MIT Cambridge, Mass., USA, 1989
25. Courcoubetis, C., Yannakakis, M.: Verifying temporal properties of finite-state probabilistic programs. In: Proc. IEEE Symposium on the Foundations of Computational Science, pp. 338–345, IEEE CS, 1988
26. Courcoubetis, C., Yannakakis, M.: The complexity of probabilistic verification. J ACM 42(4):857–907, 1995
27. Deavours, D.D., Sanders, W.H.: An efficient disk-based tool for solving very large Markov models. Performance Eval 33(1):67–84, 1998
28. Diefenbruch, M., Hintelmann, J., Müller-Clostermann, B.: The QUEST-approach for the performance evaluation of SDL-systems. In: Gotzhein, R., Bredereke, J. (eds), Formal Description Techniques IX, pp. 229–244, 1996
29. Fischer, M., Lynch, N.A., Paterson, M.: Impossibility of distributed consensus with one faulty process. J ACM 32:374–382, 1985
30. Fox, B.L., Glynn, P.W.: Computing Poisson probabilities. Commun ACM 31(4):440–445, 1998
31. Fredlund, L.: The timing and probability workbench: a tool for analysing timed processes. Technical Report No. 49, Uppsala University, 1994
32. Hachtel, G., Macii, E., Padro, A., Somenzi, F.: Markovian analysis of large finite-state machines. IEEE Trans CAD Integrated Circuits Sys 15(12):1479–1493, 1996
33. Hansson, H.A.: Time and probability in formal design of distributed systems. PhD thesis, Technical Report DoCs 91/27, Uppsala University, 1991
34. Hansson, H.A., Jonsson, B.: A logic for reasoning about time and reliability. Formal Aspects Comput 6(5):512–535, 1994
35. Hartonas-Garmhausen, V., Campos, S., Clarke, E.M.: ProbVerus: probabilistic symbolic model checking. In: Katoen, J.-P. (ed), Formal Methods for Real-Time and Probabilistic Systems, Lecture Notes in Computer Science, vol. 1601. Springer, Berlin Heidelberg New York, 1999, pp. 96–111

36. Haverkort, B.R.: Performance of computer communication systems: a model-based approach. Wiley, New York, 1998
37. Haverkort, B.R., Niemegeers, I.G.: Performability modelling tools and techniques. Performance Eval 25:17–40, 1996
38. Harel, D.: Statecharts: a visual formalism for complex systems. Sci Comput Program 8:231–274, 1987
39. Hermanns, H., Katoen, J.-P., Meyer-Kayser, J., Siegle, M.: A Markov chain model checker. In: Graf, S., Schwartzbach, M.I. (eds), Tools and Algorithms for the Construction and Analysis of Systems, Lecture Notes in Computer Science, vol. 1785. Springer, Berlin Heidelberg New York, 2000, pp. 347–362
40. Hermanns, H., Herzog, U., Katoen, J.-P.: Process algebra for performance evaluation. Theoret Comput Sci 274(1–2): 43–87, 2002
41. Hermanns, H., Herzog, U., Klehmet, U., Mertsiotakis, V., Siegle, M.: Compositional performance modelling with the TIPPtool. Performance Eval 39(1–4):5–35, 2000
42. Hermanns, H., Katoen, J.-P.: Automated compositional Markov chain generation for a plain-old telephone system. Sci Comput Program 36(1):97–127, 2000
43. Hermanns, H., Meyer-Kayser, J., Siegle, M.: Multi-terminal binary decision diagrams to represent and analyse continuous-time Markov chains. In: Proc. 3rd Int. Workshop on the Numerical Solution of Markov Chains, pp. 188–207, 1999
44. Hermanns, H.: Construction and verification of performance and reliability models. Bull ETACS 74:135–153, 2001
45. Hillston, J.: A compositional approach to performance modelling. Cambridge University, 1996
46. Holzmann, G.J.: An improved protocol reachability analysis technique. Software Pract Exper 18(2):137–161, 1988
47. Holzmann, G.J.: Design and validation of computer protocols. Prentice-Hall, Englewood Cliffs, N.J., USA, 1991
48. Ibe, O.C., Trivedi, K.S.: Stochastic Petri net models of polling systems. IEEE J Selected Areas Commun 8(9):1649–1657, 1990
49. Jensen, A.: Markov chains as an aid in the study of Markov processes. Skand Aktuarietidskrift 3:87–91, 1953
50. Katoen, J.-P., Kwiatkowska, M.Z., Norman, G., Parker, D.: Faster and symbolic CTMC model checking. In: de Alfaro, L., Gilmore, S. (eds), Process Algebra and Probabilistic Method, Lecture Notes in Computer Science, vol. 2165. Springer, Berlin Heidelberg New York, 2001, pp. 23–38
51. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Automatic verification of real-time systems with discrete probability distributions. In: Katoen, J.-P. (ed), Formal Methods for Real-Time and Probabilistic Systems, Lecture Notes in Computer Science, vol. 1601. Springer, Berlin Heidelberg New York, 1999, pp. 75–95
52. Kwiatkowska, M.Z., Norman, G., Segala, R., Sproston, J.: Verifying quantitative properties of continuous probabilistic timed automata. In: Palamidessi, C. (ed), Concurrency Theory, Lecture Notes in Computer Science, vol. 1877. Springer, Berlin Heidelberg New York, 2000, pp. 123–137
53. Martin, P.-E.: Vopp: a verification tool for probabilistic processes. MSc thesis, Uppsala University, 1993
54. McMillan, K.L.: Symbolic model checking. Kluwer Academic, Boston, Mass., USA, 1993
55. Muppala, J.K., Trivedi, K.S.: Numerical transient solution of finite Markovian queueing systems. In: Bhat, U. (ed), Queueing and Related Models, Oxford University, 1992
56. Peled, D.: Combining partial order reductions with on-the-fly model checking. Formal Methods Syst Des 8:39–64, 1996
57. Pnueli, A., Zuck, L.: Probabilistic verification. Inf Comput 103:1–29, 1993
58. Pooley, R., King, P.: Derivation of Petri net performance models from UML specifications of communications software. In: Haverkort, B.R., Bohnenkamp, H.C., Smith, C.U. (eds), Computer Performance Evaluation, Lecture Notes in Computer Science, vol. 1786. Springer, Berlin Heidelberg New York, 2000, pp. 262–276
59. Press, W., Flannery, B., Teukolsky, S., Vetterling, W.: Numerical recipes in C: the art of scientific computing. Cambridge University, 1989
60. Shiryaev, A.N.: Probability. Graduate Texts in Mathematics, Springer, Berlin Heidelberg New York, 1989
61. Stewart, W.: Introduction to the numerical solution of Markov chains. Princeton University, 1994
62. Tarjan, R.E.: Depth-first search and linear graph algorithms. SIAM J Comput 1:146–160, 1972
63. Vardi, M.Y.: Automatic verification of probabilistic concurrent finite state programs. In: Proc. IEEE Symposium on the Foundations of Computational Science, pp. 327–338, 1985