Automated Performance and Dependability Evaluation Using Model Checking

Christel Baier[1], Boudewijn Haverkort[2], Holger Hermanns[3⋆], and Joost-Pieter Katoen[3]

[1] Institut für Informatik I, University of Bonn
Römerstraße 164, D-53117 Bonn, Germany
[2] Dept. of Computer Science, RWTH Aachen
Ahornstraße 55, D-52056 Aachen, Germany
[3] Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. Markov chains (and their extensions with rewards) have been widely used to determine performance, dependability and performability characteristics of computer communication systems, such as throughput, delay, mean time to failure, or the probability to accumulate at least a certain amount of reward in a given time.
Due to the rapidly increasing size and complexity of systems, Markov chains and Markov reward models are difficult and cumbersome to specify by hand at the state-space level. Therefore, various specification formalisms, such as stochastic Petri nets and stochastic process algebras, have been developed to facilitate the specification of these models at a higher level of abstraction. Up till now, however, the specification of the measure-of-interest is often done in an informal and relatively unstructured way. Furthermore, some measures-of-interest can not be expressed conveniently at all.
In this tutorial paper, we present a logic-based specification technique to specify performance, dependability and performability measures-of-interest and show how for a given finite Markov chain (or Markov reward model) such measures can be evaluated in a fully automated way. Particular emphasis will be given to so-called path-based measures and hierarchically-specified measures. For this purpose, we extend so-called model checking techniques to reason about discrete- and continuous-time Markov chains and their rewards. We also report on the use of techniques such as (compositional) model reduction and measure-driven state-space generation to combat the infamous state space explosion problem.

1 Introduction

Over the last decades many techniques have been developed to specify and solve performance, dependability and performability models. In many cases, the models addressed possess a continuous-time Markov chain as their associated stochastic process. To avoid the specification of performance models directly at the state

⋆ Corresponding author; hermanns@cs.utwente.nl, phone: +31 53 489-4661.

M.C. Calzarossa and S. Tucci (Eds.): Performance 2002, LNCS 2459, pp. 261–289, 2002.

level, high-level specification methods have been developed, most notably those based on stochastic Petri nets, stochastic process algebras, and stochastic activity networks. With appropriate tools supporting these specification methods, such as, for instance, provided by TIPPtool [36], the PEPA workbench [23], GreatSPN [13], UltraSAN [56] or SPNP [14], it is relatively comfortable to specify performance models of which the associated CTMCs have millions of states. In combination with state-of-the-art numerical means to solve the resulting linear system of equations (for steady-state measures) or the linear system of differential equations (for time-dependent or transient measures) a good workbench is available to construct and solve dependability models of complex systems.

However, whereas the specification of performance and dependdability models has become very comfortable, the specification of the measures of interest most often has remained fairly cumbersome. In particular, most often only simple state-based measures can be defined with relative ease.

In contrast, in the area of formal methods for system verification, in particular in the area of model checking, very powerful logic-based methods have been developed to express properties of systems specified as finite state automata (note that we can view a CTMC as a special type of such an automaton). Not only are suitable means available to express state-based properties, a logic like CTL [16] (Computational Tree Logic; see below) also allows one to express properties over state sequences. Such capabilities would also be welcome in specifying performance and dependability measures.

To fulfil this aim, we have introduced the so-called *continuous stochastic logic* (CSL) that provides us ample means to specify state- as well as path-based performance measures for CTMCs in a compact and flexible way [1,2,3,4,5]. Moreover, due to the formal syntax and semantics of CSL, we can exploit the structure of CSL-specified measures in the subsequent evaluation process, such that typically the size of the underlying Markov chains that need to be evaluated can be reduced considerably.

To further strengthen the applicability of the stochastic model checking approach we recently considered Markov models involving costs or rewards, as they are often used in the performability context. We extended the logic CSL to the *continuous stochastic reward logic* CSRL in order to specify steady-state, transient and path-based measures over CTMCs extended with a reward structure (Markov reward models) [4]. We showed that well-known performability measures, most notably also the *performability distribution* introduced by Meyer [51, 52,53], can be specified using CSRL. However, CSRL allows for the specification of new measures that have not yet been addressed in the performability literature. For instance, when rewards are interpreted as costs, we can express the probability that, given a starting state, a certain goal state is reached within t time units, thereby deliberately avoiding or visiting certain immediate states, and with a total cost (accumulated reward) below a certain threshold.

We have introduced CSL and CSRL (including its complete syntax and formal semantics) in a much more theoretical context as we do in this tutorial paper (cf. [2,3,4,5,33]).

The rest of the paper is organised as follows. In Section 2 and Section 3 we present the two system evaluation techniques that will be merged in this paper: performance and dependability evaluation and formal verification by means of model checking. We then proceed with the specification of performance measures using CSL in Section 4, and of performability measures using CSRL in Section 5. Section 6 addresses lumpability to combat the state space explosion problem; Section 7 concludes the paper.

2 Performance Modelling with Markov Chains

2.1 Introduction

Performance and dependability evaluation aim at forecasting system behaviour in a quantitative way by trying to answer questions related to the performance and dependability of systems. Typical problems that are addressed are: how many clients can this file server adequately support, how large should the buffers in a router be to guarantee a packet loss of at most 10^{-6}, or how long does it take before 2 failures have occurred? Notice that we restrict ourselves to model-based performance and dependability evaluation, as opposed to measurement-based evaluation.

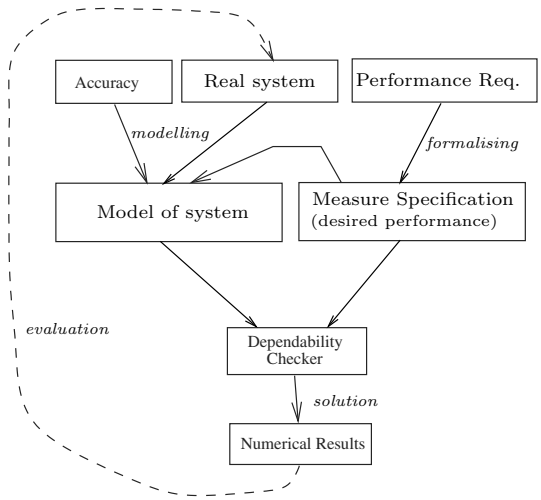


Fig. 1. The model-based performance evaluation cycle

The basic idea of model-based performance and dependability evaluation is to construct an abstract (and most often approximate) model of the system under consideration that is just detailed enough to evaluate the measures of interest (such as time-to-failure, system throughput, or number of failed components)

with the required accuracy (mean values, variances or complete distributions). The generated model is "solved" using either analytical, numerical or simulation techniques. We focus on numerical techniques as they pair a good modelling flexibility with still reasonable computational requirements. Due to the ever increasing size and complexity of real systems, performance and dependability models that are directly amenable for a numerical solution, i.e., typically continuous-time Markov chains (CTMCs), are awkward to specify "by hand" and are therefore generated automatically from high-level description/modelling languages such as stochastic Petri nets, stochastic process algebras or queueing networks [30]. The steps in the process from a system to a useful dependability or performance evaluation are illustrated in the model-based performance and dependability evaluation cycle in Fig. 1.

It remains to be stated at this point that even though good support exists for the actual model description, the specification of the measures of interest is mostly done in an informal or less abstract way.

2.2 Discrete and Continuous-Time Markov Chains

This section recalls the basic concepts of discrete- and continuous-time Markov chains with finite state space. The presentation is focused on the concepts needed for the understanding of the rest of this paper; for a more elaborate treatment we refer to [21,43,47,48,59]. We slightly depart from the standard notations by representing a Markov chain as an ordinary finite transition system where the edges are equipped with probabilistic information, and where states are labelled with atomic propositions, taken from a set AP. Atomic propositions identify specific situations the system may be in, such as "acknowledgement pending", "buffer empty", or "variable X is positive".

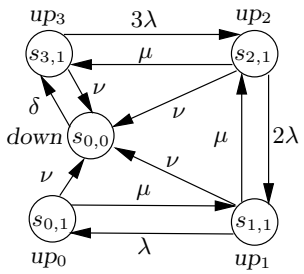
Discrete-time Markov chains. A DTMC is a triple $\mathcal{M} = (S, \mathbf{P}, L)$ where S is a finite set of *states*, $\mathbf{P} : S \times S \to [0,1]$ is the *transition probability matrix*, and $L : S \to 2^{AP}$ is the *labelling function*. Intuitively, $\mathbf{P}(s,s')$ specifies that probability to move from state s to s' in a single step, and function L assigns to each state $s \in S$ the set $L(s)$ of atomic propositions $a \in AP$ that are valid in s. One may view a DTMC as a finite state automaton equipped with transition probabilities and in which time evolves in discrete steps.

Continuous-time Markov chains. A CTMC is a tuple $\mathcal{M} = (S, \mathbf{R}, L)$ where state space S and labelling function L are as for DTMCs, and $\mathbf{R} : S \times S \to \mathbb{R}_{\geqslant 0}$ is the *rate matrix*. Intuitively, $\mathbf{R}(s,s')$ specifies that the probability of moving from state s to s' within t time-units (for positive t) is $1 - e^{-\mathbf{R}(s,s')\cdot t}$. Alternatively, a CTMC can be viewed as a finite state automaton enhanced with transition labels specifying (in a certain way) the time it takes to proceed along them. It should be noted that this definition does not require $\mathbf{R}(s,s) = -\sum_{s' \neq s} \mathbf{R}(s,s')$, as is usual for CTMCs. In the traditional interpretation, at the end of a stay in state s, the system will move to a different state. In our setting, self-loops at

state s are possible and are modelled by having $\mathbf{R}(s,s) > 0$. We thus allow the system to occupy the same state before and after taking a transition.

Let $\underline{E}(s) = \sum_{s' \in S} \mathbf{R}(s,s')$, the total rate at which any transition emanating from state s is taken.[1] More precisely, $\underline{E}(s)$ specifies that the probability of leaving s within t time-units (for positive t) is $1 - e^{-\underline{E}(s)\cdot t}$. The probability of eventually moving from state s to s', denoted $\mathbf{P}(s,s')$, 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, $\mathbf{P}(s,s') = \mathbf{R}(s,s')/\underline{E}(s)$ (except if s is an absorbing state, i.e. if $\underline{E}(s) = 0$; in this case we define $\mathbf{P}(s,s') = 0$). The matrix $\mathbf{P}$ describes an embedded DTMC of the CTMC.

Example 1. As a running example we address a *triple modular redundant system* (TMR) taken from [28], a fault-tolerant computer system consisting of three processors and a single (majority) voter. We model this system as a CTMC where state $s_{i,j}$ models that i ($0 \leqslant i < 4$) processors and j ($0 \leqslant j \leqslant 1$) voters are operational. As atomic propositions we use $AP = \{ up_i \mid 0 \leqslant i < 4 \} \cup \{ down \}$. The processors generate results and the voter decides upon the correct value by taking a majority vote. The failure rate of a single processor is λ and of the voter ν failures per hour (fph). The expected repair time of a processor is $1/\mu$ and of the voter $1/\delta$ hours. It is assumed that one component can be repaired at a time. The system is operational if at least two processors and the voter are functioning correctly. If the voter fails, the entire system is assumed to have failed, and after a repair (with rate δ) the system is assumed to start "as good as new". The details of the CTMC modelling this system are (with a clock-wise ordering of states for the matrix/vector-representation, starting with $s_{3,1}$):



$$\mathbf{R} = \begin{pmatrix} 0 & 3\lambda & 0 & 0 & \nu \\ \mu & 0 & 2\lambda & 0 & \nu \\ 0 & \mu & 0 & \lambda & \nu \\ 0 & 0 & \mu & 0 & \nu \\ \delta & 0 & 0 & 0 & 0 \end{pmatrix} \quad \text{and} \quad \underline{E} = \begin{pmatrix} 3\lambda{+}\nu \\ 2\lambda{+}\mu{+}\nu \\ \lambda{+}\mu{+}\nu \\ \mu{+}\nu \\ \delta \end{pmatrix}$$

States are represented by circles and there is an edge between state s and state s' if and only if $\mathbf{R}(s,s') > 0$. The labelling is defined by $L(s_{i,1}) = \{ up_i \}$ for $0 \leqslant i < 4$ and $L(s_{0,0}) = \{ down \}$, and is indicated near the states (set braces are omitted for singletons). For the transition probabilities we have, for instance, $\mathbf{P}(s_{2,1}, s_{3,1}) = \mu/(2\lambda{+}\mu{+}\nu)$ and $\mathbf{P}(s_{0,1}, s_{0,0}) = \nu/(\mu{+}\nu)$.

State sequences. A *path* σ through a CTMC is a (finite or infinite) sequence of states where the time spent in any of the states is recorded. For instance,

[1] Note that $\mathbf{R}$ and $\underline{E}$ just form an alternative representation of the usual infinitesimal generator matrix $\mathbf{Q}$; more precisely, $\mathbf{Q} = \mathbf{R} - \mathit{diag}(\underline{E})$. Note that this alternative representation does not affect the transient and steady-state behaviour of the CTMC, and is used for technical convenience only.

$\sigma = s_0, t_0, s_1, t_1, s_2, t_2, \ldots$ is an infinite path with for natural i, state $s_i \in S$ and time $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{R}(s_i, s_{i+1}) > 0$. We let $\sigma[i] = s_i$ denote the $(i{+}1)$-st state along a path, $\delta(\sigma, i) = t_i$, the time spent in s_i, and $\sigma@t$ the state of σ at time t. (For finite paths these notions have to be slightly adapted so as to deal with the end state of a path.) Let $Path(s)$ be the set of paths starting in s. A Borel space (with probability measure Pr) can be defined over the set $Path(s)$ in a straightforward way; for details see [2].

Steady-state and transient measures. For CTMCs, two major types of state probabilities are normally considered: steady-state probabilities where the system is considered "in the long run", i.e., when an equilibrium has been reached, and transient probabilities where the system is considered at a given time instant t. Formally, the transient probability

$$\pi(s, s', t) \;=\; \Pr\{\sigma \in Path(s) \mid \sigma@t = s'\},$$

stands for the probability to be in state s' at time t given the initial state s. We denote with $\underline{\pi}(s, t)$ the vector of state probabilities (ranging over states s') at time t, when the starting state is s. The transient probabilities are then computed from a system of linear differential equations:

$$\underline{\pi}'(s, t) = \underline{\pi}(s, t) \cdot \mathbf{Q},$$

which can be solved by standard numerical methods or by specialised methods such as *uniformisation* [45,26,25]. With uniformisation, the transient probabilities of a CTMC are computed via a uniformised DTMC which characterises the CTMC at discrete state transition epochs. Steady-state probabilities are defined as

$$\pi(s, s') = \lim_{t \to \infty} \pi(s, s', t),$$

This limit always exists for finite CTMCs. In case the steady-state distribution does not depend on the starting state s we often simply write $\pi(s')$ instead of $\pi(s, s')$. For $S' \subseteq S$, $\pi(s, S') = \sum_{s' \in S'} \pi(s, s')$ denotes the steady-state probability for the set of states S'. In this case, steady-state probabilities are computed from a system of linear equations:

$$\underline{\pi}(s) \cdot \mathbf{Q} = \underline{0} \text{ with } \sum_{s'} \pi(s, s') = 1,$$

which can be solved by direct methods (such as Gaussian elimination) or iterative methods (such as SOR or Gauss-Seidel).

Notice that the above two types of measures are truly *state based*; they consider the probability for particular states. Although this is interesting as such, one can image that for many performance and dependability questions, there is an interest in the occurrence probability of certain state *sequences*. Stated differently, we would also like to be able to express measures that address the probability on particular paths through the CTMC. Except for the recent work by Obal and Sanders [54], we are not aware of suitable mechanisms to express such measures. In the sequel, we will specifically address this issue.

3 Formal Verification with Model Checking

Whereas performance and dependability evaluation focusses on answering questions concerning quantitative system issues, traditional formal verification techniques try to answer questions related to the *functional* correctness of systems. Thus, formal verification aims at forecasting system behaviour 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., does a routing protocol eventually transfer packets to the correct destination? and (iii) fairness, e.g., will a repetitive attempt to carry out a transaction be eventually granted?

Prominent formal verification techniques are theorem proving and model checking, as well as (but to a less formal extent) testing [17,50,55,8]. Important to note at this point is that for an ever-increasing class of systems, their "formal correctness" cannot be separated anymore from their "quantitative correctness", e.g., in real-time systems, multi-media communication protocols and many embedded systems.

3.1 Model Checking

The model checking approach requires a *model* of the system under consideration together with a desired *property* and systematically checks whether the given model satisfies this property. The basic technique of model checking is a systematic, usually exhaustive, state-space search to check whether the property is satisfied in each state of the system model, thereby using effective methods to combat the infamous state-space explosion problem.

Using model checking, the user specifies a model of the system (the "possible behaviour") and a specification of the requirements (the "desirable behaviour") and leaves the verification up to the model checker. If an error is found, the model checker provides a counter-example showing under which circumstance the error can be generated. The counter-example consists of an example scenario in which the model behaves in an undesired way, thus providing evidence that the system (or the model) is faulty and needs to be revised, cf. Fig. 2. This allows the user to locate the error and to repair the system (or model specification). If no errors are found, the user can refine the model description and continue the verification process, e.g., by taking more design decisions into account, so that the model becomes more concrete and realistic.

Typically, models of systems are finite-state automata, where transitions model the evolution of the system while moving from one state to another. These automata are usually generated from a high-level description language such as Petri nets, Promela [41] or Statecharts [27]. At this point, notice the similarities with the models used for performance and dependability evaluation.

Computational Tree Logic. Required system properties can be specified in an extension of propositional logic called *temporal logic*. Temporal logics allow the formulation of properties that refer to the dynamic behaviour of a system;

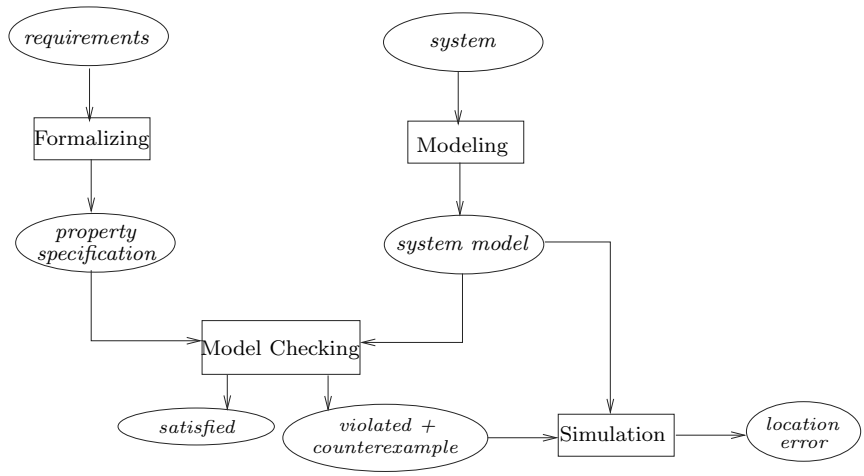


Fig. 2. The model checking approach

it allows to express for instance the temporal ordering of events. Note that the term "temporal" is meant in a qualitative sense, not in a quantitative sense. An important logic for which efficient model checking algorithms exist is CTL [16] (Computational Tree Logic). This logic allows to state properties over *states*, and over *paths* using the following syntax:

State-formulas

$$\Phi ::= a \mid \neg\Phi \mid \Phi \vee \Phi \mid \exists\varphi \mid \forall\varphi$$

a : atomic proposition
$\exists\varphi$: there *Exists* a path that fulfils φ
$\forall\varphi$: *All* paths fulfil φ

Path-formulas

$$\varphi ::= X\,\Phi \mid \Phi\,\mathcal{U}\,\Phi$$

$X\,\Phi$: the *neXt* state fulfils Φ
$\Phi\,\mathcal{U}\,\Psi$: Φ holds along the path, *Until* Ψ holds
$\Diamond\Phi$: $\mathsf{true}\,\mathcal{U}\,\Phi$, i.e., eventually Φ
$\Box\Phi$: $\neg\Diamond\neg\Phi$, i.e., invariantly Φ

The meaning of atomic propositions, negation ($\neg$) and disjunction ($\vee$) is standard; note that using these operators, other boolean operators such as conjunction ($\wedge$), implication ($\Rightarrow$), and so forth, can be defined. The state-formula $\exists\varphi$ is valid in state s if there *exists* some path starting in s and satisfying φ. The formula $\exists\Diamond$*deadlock*, for example, expresses that for some system run eventually a deadlock can be reached (potential deadlock). On the contrary, $\forall\varphi$ is valid if *all* paths satisfy φ; $\forall\Diamond$*deadlock* thus means that a deadlock is inevitable. A path satisfies an until-formula $\Phi\,\mathcal{U}\,\Psi$ if the path has an initial finite prefix (possibly

only containing state s) such that Φ holds at all states along the path until a state for which Ψ holds is encountered along the path.

Example 2. Considering the TMR system example as a finite-state automaton, some properties one can express with CTL are:

- $up_3 \Rightarrow \exists\Diamond down$:
  if the system is fully operational, it may eventually go down.
- $up_3 \Rightarrow \forall\mathsf{X}\,(up_2 \vee down)$:
  if the system is fully operational, any next step involves the failure of a component.
- $\exists\Box\neg\, down$:
  it is possible that the voter never fails.
- $\exists((up_3 \vee up_2)\,\mathcal{U}\, down)$:
  it is possible to have two or three processors continuously working until the voter fails.

Model checking CTL. A model, i.e., a finite-state automaton where states are labelled with atomic propositions, is said to satisfy a property if and only if all its initial states satisfy this property. In order to check whether a model satisfies a property Φ, the set $Sat(\Phi)$ of states that *sat*isfy Φ is computed recursively, after which it is checked whether the initial states belong to this set. For atomic propositions this set is directly obtained from the above mentioned labelling of the states; $Sat(\Phi \wedge \Psi)$ is obtained by computing $Sat(\Phi)$ and $Sat(\Psi)$, and then intersecting these sets; $Sat(\neg\Phi)$ is obtained by taking the complement of the entire state space with respect to $Sat(\Phi)$. The algorithms for the temporal operators are slightly more involved. For instance, for $Sat(\exists\mathsf{X}\,\Phi)$ we first compute the set $Sat(\Phi)$ and then compute those states from which one can move to this set by a single transition. $Sat(\exists(\Phi\,\mathcal{U}\,\Psi))$ is computed in an iterative way: (i) as a precomputation we determine $Sat(\Phi)$ and $Sat(\Psi)$; (ii) we start the iteration with $Sat(\Psi)$ as these states will surely satisfy the property of interest; (iii) we extend this set by the states in $Sat(\Phi)$ from which one can move to the already computed set by a single transition; (iv) if no new states have been added in step (iii), we have found the required set, otherwise we repeat (iii). As the number of states is finite, this procedure is guaranteed to terminate. The worst case time complexity of this algorithm (after an appropriate treatment of the $\exists\Box$-operator [16]) is linear in the size of the formula and the number of transitions in the model.

Applications. Although the model checking algorithms are conceptually relatively simple, their combination with clever techniques to combat the state-space explosion problem (such as binary decision diagrams, bit-state hashing and partial-order reduction) make model checking a widely applicable and successful verification technique. This is illustrated by the success of model checkers such as SPIN, SMV, Uppaal and Murφ, and their successful application to a large set of industrial case studies ranging from hardware verification (VHDL, Intel P7 Processor), software control systems (traffic collision avoidance and alert system

TCAS-II, storm surge barrier), and communication protocols (ISDN-User Part and IEEE Futurebus+); see for an overview [18].

4 Stochastic Model Checking CTMCs

As has become clear from the previous section, the existing approaches for formal verification using model checking and performance and dependability evaluation have a lot in common. Our aim is to integrate these two evaluation approaches even more, thereby trying to combine the best of both worlds.

4.1 A Logic for Performance and Dependability

To specify and evaluate performance and dependability measures as logical formulas over CTMCs, we describe in this section CSL [1,2], a stochastic variant of CTL, and explain how model checking this logic can be performed, summarising work reported in [2,3,46].

Syntax. CSL extends CTL with two probabilistic operators that refer to the steady-state and transient behaviour of the system being studied. Whereas the steady-state operator refers to the probability of residing in a particular set of *states* (specified by a state-formula) in the long run, the transient operator allows us to refer to the probability of the occurrence of particular *paths* in the CTMC. In order to express the time-span of a certain path, the path-operators until $\mathcal{U}$ and next X are extended with a parameter that specifies a time-interval. Let I be an interval on the real line, p a probability and $\trianglelefteq$ a comparison operator, i.e., $\trianglelefteq \in \{\leqslant, \geqslant\}$. The syntax of CSL now becomes:

State-formulas

$$\Phi ::= a \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathcal{S}_{\trianglelefteq p}(\Phi) \mid \mathcal{P}_{\trianglelefteq p}(\varphi)$$

$\mathcal{S}_{\trianglelefteq p}(\Phi)$: prob. that Φ holds in steady state $\trianglelefteq p$
$\mathcal{P}_{\trianglelefteq p}(\varphi)$: prob. that a path fulfils φ $\trianglelefteq p$

Path-formulas

$$\varphi ::= X^I \Phi \mid \Phi \mathcal{U}^I \Phi$$

$X^I \Phi$: the next state is reached at time $t \in I$ and fulfils Φ
$\Phi \mathcal{U}^I \Psi$: Φ holds along the path until Ψ holds at time $t \in I$

The state-formula $\mathcal{S}_{\trianglelefteq p}(\Phi)$ asserts that the steady-state probability for the set of Φ-states meets the bound $\trianglelefteq p$. The operator $\mathcal{P}_{\trianglelefteq p}(.)$ replaces the usual CTL path quantifiers $\exists$ and $\forall$. In fact, for most cases $\exists\varphi$ can be written as $\mathcal{P}_{>0}(\varphi)$ and $\forall\varphi$ as $\mathcal{P}_{\geqslant 1}(\varphi)$. These rules are not generally applicable due to fairness considerations [6]. $\mathcal{P}_{\trianglelefteq p}(\varphi)$ asserts that the probability measure of the paths satisfying φ meets the bound $\trianglelefteq p$. Temporal operators like $\Diamond$, $\Box$ and their real-time variants $\Diamond^I$ or $\Box^I$ can be derived, e.g., $\mathcal{P}_{\trianglelefteq p}(\Diamond^I \Phi) = \mathcal{P}_{\trianglelefteq p}(\mathsf{true}\, \mathcal{U}^I \Phi)$ and $\mathcal{P}_{\geqslant p}(\Box^I \Phi) = \mathcal{P}_{\leqslant 1-p}(\Diamond^I \neg\Phi)$. The untimed next- and until-operators are obtained by $X\Phi = X^I\Phi$ and $\Phi_1 \,\mathcal{U}\, \Phi_2 = \Phi_1 \,\mathcal{U}^I\, \Phi_2$ for $I = [0, \infty)$.

Semantics. State-formulas are interpreted over the states of a CTMC. Let $\mathcal{M} = (S, \mathbf{R}, L)$ with labels in AP. The meaning of CSL-formulas is defined by means of a so-called satisfaction relation (denoted by $\models$) between a CTMC $\mathcal{M}$, one of its states s, and a formula Φ. For simplicity the CTMC identifier $\mathcal{M}$ is often omitted as it is clear from the context. The pair (s, Φ) belongs to the relation $\models$, usually denoted by $s \models \Phi$, if and only if Φ is valid in s. For CSL state-formulas we have:

$$\begin{array}{ll} s \models a & \text{iff } a \in L(s), \\ s \models \neg\Phi & \text{iff } s \not\models \Phi, \\ s \models \Phi_1 \ \vee \ \Phi_2 & \text{iff } s \models \Phi_1 \ \vee \ s \models \Phi_2, \\ s \models \mathcal{S}_{\trianglelefteq p}(\Phi) & \text{iff } \pi(s, Sat(\Phi)) \trianglelefteq p, \\ s \models \mathcal{P}_{\trianglelefteq p}(\varphi) & \text{iff } Prob(s, \varphi) \trianglelefteq p, \end{array}$$

where $Prob(s, \varphi)$ denotes the probability of all paths $\sigma \in Path(s)$ satisfying φ when the system starts in state s, i.e.,

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

The satisfaction relation for the path-formulas is defined by a satisfaction relation (also denoted by $\models$) between paths and CSL path-formulas as follows. We have that $\sigma \models X^I \Phi$ iff

$$\sigma[1] \text{ is defined and } \sigma[1] \models \Phi \ \wedge \ \delta(\sigma, 0) \in I,$$

and that $\sigma \models \Phi_1 \, \mathcal{U}^I \, \Phi_2$ iff

$$\exists t \in I.\ (\sigma@t \models \Phi_2 \ \wedge \ \forall u \in [0, t).\, \sigma@u \models \Phi_1)\,.$$

Note that the formula $\Phi_1 \, \mathcal{U}^{\varnothing} \, \Phi_2$ cannot be satisfied.

4.2 Expressing Measures in CSL

What types of performance and dependability measures can be expressed using CSL? As a first observation, we remark that by means of the logic one does not specify a measure but in fact a constraint (or: bound) on a performance or dependability measure. Four types of measures can be identified: steady-state measures, transient-state measures, path-based measures, and nested measures. Assume that for each state s, we have a characteristic atomic proposition $in(s)$ valid in state s and invalid in any other state.

Steady-state measures. The formula $\mathcal{S}_{\trianglelefteq p}(in(s))$ imposes a requirement on the steady-state probability to be in state s. For instance, $\mathcal{S}_{\leqslant 10^{-5}}(in(s_{2,1}))$ is valid in state $s_{0,0}$ (cf. the running example) if the steady-state probability of having a system configuration in which a single processor has failed is at most 0.00001 (when starting in state $s_{0,0}$). This can be easily generalized towards selecting sets of states by using more general state-formulas. The formula $\mathcal{S}_{\trianglelefteq p}(\Phi)$ imposes a constraint on the probability to be in some Φ-state on the long run. For instance, the formula $\mathcal{S}_{\geqslant 0.99}(up_3 \ \vee \ up_2)$ states that on the long run, for at least 99% of the time at least 2 processors are operational.

Transient measures. The combination of the probabilistic operator with the temporal operator $\Diamond^{[t,t]}$ can be used to reason about transient probabilities since

$$\pi(s, s', t) = Prob(s, \Diamond^{[t,t]}\ in(s')).$$

More specifically, $\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\ in(s'))$ is valid in state s if the transient probability at time t to be in state s' satisfies the bound $\trianglelefteq p$. For instance, $\mathcal{P}_{\leqslant .2}(\Diamond^{[t,t]}\ in(s_{2,1}))$ is valid in state $s_{0,0}$ if the transient probability of state $s_{2,1}$ at time t is at most 0.2 when starting in state $s_{0,0}$. In a similar way as done for steady-state measures, the formula $\mathcal{P}_{\geqslant 0.99}(\Diamond^{[t,t]}\ up_3 \vee up_2)$ requires that the probability to have 3 or 2 processors running at time t is at least 0.99. For specification convenience, a transient-state operator

$$\mathcal{T}^{@t}_{\trianglelefteq p}(\Phi) = \mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\,\Phi)$$

could be defined. It states that the probability for a Φ-state at time t meets the bound $\trianglelefteq p$.

Path-based measures. The standard transient measures on (sets of) states are expressed using a specific instance of the $\mathcal{P}$-operator. However, by the fact that this operator allows an arbitrary path-formula as argument, much more general measures can be described. An example is the probability of reaching a certain set of states provided that all paths to these states obey certain properties. For instance,

$$\mathcal{P}_{\leqslant 0.01}((up_3 \vee up_2)\,\mathcal{U}^{[0,10]}\ down)$$

is valid for those states where the probability of the system going *down* within 10 time-units after having continuously operated with at least 2 processors is at most 0.01.

Nested measures. By nesting the $\mathcal{P}$- and $\mathcal{S}$-operators more complex measures of interest can be specified. These are useful to obtain a more detailed insight into the system's behaviour. We provide two examples. The property

$$\mathcal{S}_{\leqslant 0.9}(\mathcal{P}_{\geqslant 0.8}(\Box^{[0,10]}\neg down))$$

is valid in those states that guarantee that in equilibrium with probability at least 0.9 the probability that the system will not go down within 10 time units is at least 0.8. Conversely,

$$\mathcal{P}_{\geqslant 0.5}((\neg down)\,\mathcal{U}^{[10,20]}\,\mathcal{S}_{\geqslant 0.8}((up_3 \vee up_2)))$$

is valid for those states that with probability at least 0.5 will reach a state s between 10 and 20 time-units, which guarantees the system to be operational with at least 2 processors when the system is in equilibrium. Besides, prior to reaching state s the system must be operational continuously.

To put it in a nutshell, we believe that there are two main benefits by using CSL for specifying constraints on measures-of-interest. First, the specification is

completely formal such that the interpretation is unambiguous. Whereas this is also the case for standard transient and steady-state measures, this often does not apply to measures that are derived from these elementary measures. Such measures are typically described in an informal manner. A rigorous specification of such more intricate measures is of utmost importance for their automated analysis (as proposed in the sequel). Furthermore, an important aspect of CSL is the possibility to state performance and dependability requirements over a selective set of paths through a model, which was not possible previously. Finally, the possibility to nest steady-state and transient measures provides a means to specify complex, though important measures in a compact and flexible way.

4.3 Model Checking CSL-Specified Measures

Once we have formally specified the (constraint on the) measure-of-interest in CSL by a formula Φ, and have obtained our model, i.e., CTMC $\mathcal{M}$, of the system under consideration, the next step is to adapt the model checking algorithm for CTL to support the automated validation of Φ over a given state s in $\mathcal{M}$. The basic procedure is as for model checking CTL: in order to check whether state s satisfies the formula Φ, we recursively compute the set $Sat(\Phi)$ of states that satisfy Φ, and check whether s is a member of that set. For the non-probabilistic state-operators this procedure is the same as for CTL. The main problem we have to face is how to compute $Sat(\Phi)$ for the $\mathcal{S}$ and $\mathcal{P}$-operators. We deal with these operators separately.

Steady-state measures. For an ergodic (strongly connected) CTMC:

$$s \in Sat(\mathcal{S}_{\trianglelefteq p}(\Phi)) \text{ iff } \sum_{s' \in Sat(\Phi)} \pi(s, s') \trianglelefteq p.$$

Thus, checking whether state s satisfies $\mathcal{S}_{\trianglelefteq p}(\Phi)$, a standard steady-state analysis has to be carried out, i.e., a system of linear equations has to be solved.

In case the CTMC $\mathcal{M}$ is not strongly-connected, the approach is to determine the so-called bottom strongly-connected components (BSCCs) of $\mathcal{M}$, i.e., the set of strongly-connected components that cannot be left once they are reached. Then, for each BSCC (which is an ergodic CTMC) the steady-state probability of a Φ-state (determined in the standard way) and the probability to reach any BSCC B from state s is determined. To check whether state s satisfies $\mathcal{S}_{\trianglelefteq p}(\Phi)$ it then suffices to verify

$$\sum_{B} \left(Prob(s, \Diamond B) \cdot \sum_{s' \in B \cap Sat(\Phi)} \pi^B(s') \right) \trianglelefteq p,$$

where $\pi^B(s')$ denotes the steady-state probability of s' in BSCC B, and $Prob(s, \Diamond B)$ is the probability to reach BSCC B from state s. To compute these probabilities, standard methods for steady-state and graph analysis can be used.

Path-based measures. In order to understand how the model checking of the path-based operators is carried out it turns out to be helpful to give (recursive) characterisations of $Prob(s, \varphi)$:

$$s \in Sat(\mathcal{P}_{\trianglelefteq p}(\varphi)) \text{ iff } Prob(s, \varphi) \trianglelefteq p.$$

- **Timed Next:** For the timed next-operator we obtain that $Prob(s, X^I \Phi)$ equals

$$\left(e^{-\underline{E}(s)\cdot \inf I} - e^{-\underline{E}(s)\cdot \sup I}\right) \cdot \sum_{s' \in Sat(\Phi)} \mathbf{P}(s, s'), \tag{1}$$

  i.e., the probability to leave state s in the interval I times the probability to reach a Φ-state in one step. Thus, in order to compute the set $Sat(X^I \Phi)$ we first recursively compute $Sat(\Phi)$ and add state s to $Sat(X^I \Phi)$ if it fulfils (1); this check boils down to a matrix-vector multiplication.
- **Time-Bounded Until:** For the sake of simplicity, we only treat the case $I = [0, t]$; the general case is a bit more involved, but can be treated in a similar way [3]. The probability $Prob(s, \Phi \mathcal{U}^{[0,t]} \Psi)$ is the least solution of the following set of equations: (i) 1, if $s \in Sat(\Psi)$, (ii) 0, if $s \notin Sat(\Phi) \cup Sat(\Psi)$, and

$$\int_0^t \sum_{s' \in S} \mathbf{R}(s, s') \cdot e^{-\underline{E}(s)\cdot x} \cdot Prob(s', \Phi \mathcal{U}^{[0,t-x]} \Psi)\, dx \tag{2}$$

  otherwise. The first two cases are self-explanatory; the last equation is explained as follows. If s satisfies Φ but not Ψ, 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 $t-x$.
  It is easy to check that for the untimed until-operator (i.e., $I = [0, \infty)$) equation (2) reduces to

$$\sum_{s' \in S} \mathbf{P}(s, s') \cdot Prob(s', \Phi \mathcal{U} \Psi).$$

  Thus, for the standard until-operator, we can check whether a state satisfies $\mathcal{P}_{\trianglelefteq p}(\Phi \mathcal{U} \Psi)$ by first computing recursively the sets $Sat(\Phi)$ and $Sat(\Psi)$ followed by solving a linear system of equations.

Solution for time-bounded until. We now concentrate on numerical techniques for solving the so-called Volterra integral equation system (2) arising in the time-bounded until case.

As a first approach, numerical integration techniques can be applied. Experiments with integration techniques based on equally-sized abscissas have shown that the computation time for solving (2) is rapidly increasing when the state space becomes larger (above 10,000 states), or when the required accuracy becomes higher, e.g., between 10^{-6} and 10^{-9}. Numerical stability is another issue of concern when using this method [37].

An alternative method is to reduce the problem of computing $Prob(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi)$ to a transient analysis problem for which well-known and efficient computation techniques do exist. This idea is based on the earlier observation that for a specific instance of the time-bounded until-operator we know that it characterises a standard transient probability measure:

$$\mathcal{T}^{@t}_{\trianglelefteq p}(\Phi) = \mathcal{P}_{\trianglelefteq p}(\mathsf{true}\,\mathcal{U}^{[t,t]}\,\Phi)$$

Thus, for computing $Prob(s, \mathsf{true}\ \mathcal{U}^{[t,t]}\,\Phi)$ standard transient analysis techniques can be exploited. This raises the question whether we might be able to reduce the general case, i.e., $Prob(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi)$, to an instance of transient analysis as well. This is indeed possible: the idea is to transform the CTMC $\mathcal{M}$ under consideration into another CTMC $\mathcal{M}'$ such that checking $\varphi = \Phi\,\mathcal{U}^{[0,t]}\,\Psi$ on $\mathcal{M}$ amounts to checking $\varphi' = \mathsf{true}\ \mathcal{U}^{[t,t]}\,\Psi$ on $\mathcal{M}'$; a transient analysis of $\mathcal{M}'$ (for time t) then suffices. The question then is, how do we transform $\mathcal{M}$ in $\mathcal{M}'$? Two simple observations form the basis for this transformation. First, we observe that once a Ψ-state in $\mathcal{M}$ has been reached (along a Φ-path) before time t, we may conclude that φ holds, regardless of which states will be visited after having reached Ψ. Thus, as a first transformation we make all Ψ-states absorbing. Secondly, we observe that φ is violated once a state has been reached that neither satisfies Φ nor Ψ. Again, this is regardless of the states that are visited after having reached $\neg(\Phi \wedge \Psi)$. Thus, as a second transformation, all the $\neg(\Phi \wedge \Psi)$-states are made absorbing. It then suffices to carry out a transient analysis on the resulting CTMC $\mathcal{M}'$ for time t and collect the probability mass to be in a Ψ-state (note that $\mathcal{M}'$ typically is smaller than $\mathcal{M}$):

$$Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi) = Prob^{\mathcal{M}'}(s, \mathsf{true}\ \mathcal{U}^{[t,t]}\,\Psi).$$

In fact, by similar observations it turns out that also verifying the general $\mathcal{U}^I$-operator can be reduced to instances of (nested) transient analysis [3]. As mentioned above, the transient probability distribution can be computed via a *uniformised* DTMC which characterises the CTMC at discrete state transition epochs. A direct application of uniformisation to compute $Prob^{\mathcal{M}}(s, \Phi\,\mathcal{U}^{[0,t]}\,\Psi)$ requires to perform this procedure for each state s. An improvement suggested in [46] cumulates the entire vector $\underline{Prob}^{\mathcal{M}}(\Phi\,\mathcal{U}^I\,\Psi)$ for all states simultaneously.

For a single operator $\mathcal{U}^I$ this yields a time complexity of $\mathcal{O}(|\mathbf{R}|\cdot N_\varepsilon)$, where $|\mathbf{R}|$ is the number of non-zero entries in $\mathbf{R}$, and N_ε is the number of iterations within the uniformisation algorithm needed to achieve a given accuracy ε. The value N_ε can be computed a priori, it linearly depends on the maximal diagonal entry of the generator matrix $\underline{E}_{\max}$, and on the maximal time bound $t_{\max}$ occuring in Φ.

In total, the time complexity to decide the validity of a CSL fomula Φ on a CTMC $(S, \mathbf{R}, L)$ is $\mathcal{O}(|\Phi|\cdot(|\mathbf{R}|\cdot\underline{E}_{\max}\cdot t_{\max} + |S|^{2.81}))$, and the space complexity is $\mathcal{O}(|\mathbf{R}|)$ [5].

5 Stochastic Model Checking Markov Reward Models

5.1 Introduction

With the advent of fault-tolerant gracefully-degradable computer systems, the separation between performance and dependability aspects of a system does not make sense anymore. Indeed, fault-tolerant systems can operate "correctly" at various levels of performance, and the dependability of a system might be expressed in terms of providing a minimum performance level, rather then in terms of a certain amount of operational hardware resources. These considerations lead, in the late 1970'a and the early 1980's, to the concept of performability [51,52], in which it is investigated how well a system performs over a finite time horizon, provided (partial) system failures and repair actions are taken into account. As it turned out later, the notion of performability also fits quite naturally to the notion of quality of service as specified in ITU-T Recommendation G.106 [12]. Furthermore, as natural model for performability evaluations, so-called Markov reward models have been adopted, as will be explained below; for further details on performability evaluation, see [29].

Markov reward models. An MRM is a CTMC augmented with a *reward structure* assigning a real-valued reward to each state in the model. Such reward can be interpreted as bonus, gain, or conversely, as cost. Typical measures of interest express the amount of gain accumulated by the system, over a finite or infinite time-horizon. Formally, an MRM is a tuple $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ where $(S, \mathbf{R}, L)$ is a CTMC, and $\rho : S \to \mathbb{R}_{\geqslant 0}$ is a *reward structure* that assigns to each state s a reward $\rho(s)$, also called gain or bonus, or dually, cost.

Example 3. For the TMR example, the reward structure can be instantiated in different ways so as to specify a variety of performability measures. The simplest reward structure (leading to an availability model) divides the states into operational and non-operational ones: $\rho_1(s_{0,0}) = 0$ and $\rho_1(s_{i,0}) = 1$ for the remaining states. A reward structure in which varying levels of trustworthiness are represented is for instance based on the number of operational processors: $\rho_2(s_{0,0}) = 0$ and $\rho_2(s_{i,1}) = i$. As a third reward structure, one may consider the maintenance costs of the system, by setting: $\rho_3(s_{0,0}) = c_2$ and $\rho_3(s_{i,1}) = c_1 \cdot (3 - i)$, where c_1 is the cost to replace a processor, and c_2 the cost to renew the entire system. As a fourth option (which we do not further consider here) one can also imagine a reward structure quantifying the power consumption in each state.

Accumulating reward along a path. The presence of a reward structure allows one to reason about (at least) two different aspects of system cost/reward. One either may refer to the instantaneous reward at a certain point in time (even in steady-state), or one may refer to the reward accumulated in a certain interval of time. For an MRM $(S, \mathbf{R}, L, \rho)$, and $\sigma = s_0, t_0, s_1, t_1, s_2, t_2, \ldots$ an infinite path (through the corresponding CTMC $(S, \mathbf{R}, L)$) the instantaneuos reward at time t is given by $\rho(\sigma@t)$. The cumulated reward $y(\sigma, t)$ along σ up

to time t can be formalised as follows. For $t = \sum_{j=0}^{k-1} t_j + t'$ with $t' \leqslant t_k$ we define $y(\sigma, t) = \sum_{j=0}^{k-1} t_j \cdot \rho(s_j) + t' \cdot \rho(s_k)$. For finite paths ending at time point t the cumulated reward definition is slightly adapted, basically replacing t' by $t - \sum_{j=0}^{l-1} t_j$.

Measure specification. The specification of the measure-of-interest for a given MRM can not always be done conveniently, nor can all possible measures-of-interest be expressed conveniently. In particular, until recently it has not been possible to directly express measures where state *sequences* or paths matter, nor to accumulate rewards only in certain subsets of states, if the rewards outside these subsets are non-zero. Such measures are then either "specified" informally, with all its negative implications, or require a manual tailoring of the model so as to address the right subsets of states. Below we will address a rigorous but flexible way of expressing performability measures.

Finally, note that Obal and Sanders recently proposed a technique to specify so-called path-based reward variables [54] by which the specification of measures over state sequences becomes more convenient, because it avoids the manual tailoring of the model. In the context of the stochastic process algebra PEPA, Clark *et al.* recently proposed the use of a probabilistic modal logic to ease the specification of reward *structures* of MRM [15], as opposed to the specification of reward-based *measures*, as we do.

5.2 A Logic for Performability

The addition of rewards on the model level raises the question how they can be reflected on the measure specification level, i.e., on the level of the logic. We restrict ourselves for the moment to consider the accumulation of reward, because this turns out to be a conceptually interesting extension that fits very well to the temporal logic approach. We shall later (in Section 5.6) return to the question how to support other forms of reward quantification, such as instantaneuos reward.

Since rewards are accumulated along a path, it appears wise to extend the *path* formulas of CSL to account for the earning of reward, and this is what distinguishes CSRL from CSL. The state formulas of CSRL are unchanged relative to CSL (until Section 5.6), whereas path formulas φ now become

$$\varphi \ ::= \ X_J^I \, \Phi \ \Big| \ \Phi \, \mathcal{U}_J^I \, \Phi,$$

for intervals $I, J \subseteq \mathbb{R}_{\geqslant 0}$. In a similar way as before, we define $\Diamond_J^I \Phi = \mathsf{true} \, \mathcal{U}_J^I \, \Phi$ and $\mathcal{P}_{\trianglelefteq p}(\Box_J^I \Phi) = \neg \mathcal{P}_{\trianglelefteq p}(\Diamond_J^I \neg \Phi)$. Interval I can be considered as a timing constraint whereas J represents a bound for the cumulative reward. The path-formula $X_J^I \, \Phi$ asserts that a transition is made to a Φ-state at time point $t \in I$ such that the earned cumulative reward r until time t meets the bounds specified by J, i.e., $r \in J$. The semantics of $\Phi_1 \, \mathcal{U}_J^I \, \Phi_2$ is as for $\Phi_1 \, \mathcal{U}^I \, \Phi_2$ with the additional constraints that earned cumulative reward r at the time of reaching some Φ_2-state lies in J, i.e., $r \in J$.

Example 4. As an example property for the TMR system, $\mathcal{P}_{\geqslant 0.95}(\Diamond^{[60,60]}_{[0,200]}\mathsf{true})$ denotes that with probability of at least 0.95 the cumulative reward, e.g., the incurred costs of the system for reward structure ρ_3, at time instant 60 is at most 200. Given that the reward of a state indicates the power consumed per time-unit, property $\mathcal{P}_{<0.08}(up_3\ \mathcal{U}^{[0,30]}_{[7,\infty)}\ (down \vee\ up_2))$ expresses that with probability less than 0.08 within 30 time units at least 7 units of power have been consumed in full operational mode before some component fails. A simpler property, that only refers to reward accumulation, $\mathcal{P}_{>0.5}(\Diamond^{[0,\infty)}_{[0,10]} down)$ would say that it is likely (probability > 0.5) to spend less than 10 units of energy before a voter failure.

The semantics of the CSRL path-formulas is an extension of the CSL semantics we introduced in Section 4.1. It differs from the latter in that additional constraints are imposed reflecting that the accumulated reward on the path must be in the required interval. We have that $\sigma \models X^I_J\,\Phi$ iff

$$\sigma[1] \text{ is defined and } \sigma[1] \models \Phi \wedge \delta(\sigma,0) \in I\ \wedge\ y(\sigma,\delta(\sigma,0)) \in J$$

and that $\sigma \models \Phi_1\,\mathcal{U}^I_J\,\Phi_2$ iff

$$\exists t \in I.\,(\sigma@t \models \Phi_2\ \wedge\ (\forall u \in [0,t).\,\sigma@u \models \Phi_1)\ \wedge\ y(\sigma,t) \in J).$$

For the X^I_J case, the definition refines the one for CSL by demanding that the reward accumulated during the time $\delta(\sigma,0)$ of staying in the first state of the path lies in J, while for $\mathcal{U}^I_J$ the reward accumulated until the time t when touching a Φ_2-state must be in J.

5.3 Expressing Measures in CSRL

MRMs are an extension of CTMCs, and so is CSRL an extension of CSL. Since CSL does not allow any reference to rewards, it is obtained by putting no constraint on the reward accumulated, i.e., by setting $J = [0,\infty)$ for all sub-formulas:

$$X^I\,\Phi\ =\ X^I_{[0,\infty)}\,\Phi \qquad \text{and} \qquad \Phi_1\,\mathcal{U}^I\,\Phi_2\ =\ \Phi_1\,\mathcal{U}^I_{[0,\infty)}\,\Phi_2.$$

Similarly, we can identify a new logic CRL (continuous reward logic) in case $I = [0,\infty)$ for all sub-formulas. In CRL it is only possible to refer to the cumulation of rewards, but not to the advance of time. The formula $\mathcal{P}_{>0.5}(\Diamond^{[0,\infty)}_{[0,10]} down)$ is an example property of the CRL subset of CSRL. The CRL logic will play a special role when describing the model checking of CSRL, and therefore we will first discuss how model checking CRL can be performed, before turning our attention to CSRL. Before doing so, we list in Table 1 a variety of standard performance, dependability, and performability measures and how they can be phrased in CSRL. Here F is a generic formula playing the role of an identifier of the failed system states of the model under study (in the TMR example, F would be $down \vee up_0$). These measures correspond to basic formulas in the logic,

Table 1. Some typical performability measures

performability measure	formula	logic
steady-state availability	$\mathcal{S}_{\trianglelefteq p}(\neg F)$	**CSL**
instantaneous availability at time t	$\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\neg F)$	**CSL**
distribution of time to failure	$\mathcal{P}_{\trianglelefteq p}(\neg F\,\mathcal{U}^{\,[0,t]}\,F)$	**CSL**
distribution of reward until failure	$\mathcal{P}_{\trianglelefteq p}(\neg F\,\mathcal{U}_{[0,r]}\,F)$	**CRL**
distribution of cumulative reward until t	$\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}_{[0,r]}\mathsf{true})$	**CSRL**

and it is worth to highlight that much more involved and nested measures are easily expressible in CSRL, such as

$$\mathcal{S}_{>0.3}\left(\mathcal{P}_{<0.3}(\Diamond^{[0,85]}_{[3,5]}up_2)\Rightarrow\mathcal{P}_{>0.1}((\neg down)\,\mathcal{U}^{\,[5,\infty)}\,up_3)\right).$$

5.4 Model Checking CRL-Specified Measures

This section discusses how model checking can be performed for CRL properties, i.e., formulas which do only refer to the cumulation of rewards, but not to the advance of time. We will explain how a duality result can be used to reduce model checking of such formulas to the CSL model checking algorithm described above.

The basic strategy is the same as for CSL, and only the path operators X_J, $\mathcal{U}_J$ need specific considerations. To calculate the probability of satisfiying such a path formula we rely on a general duality result for MRMs and CSRL [4].

Duality. Assume an MRM $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ with positive reward structure, i.e., $\rho(s) > 0$ for each state s. The basic idea behind the duality phenomenon is that the progress of time can be regarded as the earning of reward and vice versa. This observation is inspired by [7]. To make it concrete, we define an MRM $\mathcal{M}^{-1} = (S, \mathbf{R}', L, \rho')$ that results from $\mathcal{M}$ by

- rescaling the transition rates by the reward of their originating state, i.e., $\mathbf{R}'(s,s') = \mathbf{R}(s,s')/\rho(s)$ and,
- inverting the reward structure, i.e., $\rho'(s) = 1/\rho(s)$.

Intuitively, the transformation of $\mathcal{M}$ into $\mathcal{M}^{-1}$ stretches the residence time in state s with a factor that is proportional to the reciprocal of its reward $\rho(s)$ if $\rho(s) > 1$, and it compresses the residence time by the same factor if $0 < \rho(s) < 1$. The reward structure is changed similarly. Note that $\mathcal{M} = (\mathcal{M}^{-1})^{-1}$.

One might interpret the residence of t time units in $\mathcal{M}^{-1}$ as the earning of t reward in state s in $\mathcal{M}$, or (reversely) an earning of a reward r in state s in $\mathcal{M}$ corresponds to a residence of r in $\mathcal{M}^{-1}$. As a consequence, the rôles of time and

reward in $\mathcal{M}$ are reversed in $\mathcal{M}^{-1}$. In terms of the logic CSRL, this corresponds to swapping reward and time intervals inside a CSRL formula, and allows one to establish that

$$Prob^{\mathcal{M}}(s, X_J^I\, \Phi) = Prob^{\mathcal{M}^{-1}}(s, X_I^J\, \Phi), \text{ and}$$

$$Prob^{\mathcal{M}}(s, \Phi_1\, \mathcal{U}_J^I\, \Phi_2) = Prob^{\mathcal{M}^{-1}}(s, \Phi_1\, \mathcal{U}_I^J\, \Phi_2).$$

As a consequence, one can obtain the set $Sat^{\mathcal{M}}(\Phi)$ (comprising the states in $\mathcal{M}$ satisfying Φ) by computing instead $Sat^{\mathcal{M}^{-1}}(\Phi^{-1})$, i.e.,

$$Sat^{\mathcal{M}}(\Phi) \quad = \quad Sat^{\mathcal{M}^{-1}}(\Phi^{-1}),$$

where Φ^{-1} is defined as Φ where for each sub-formula in Φ of the form X_J^I or $\mathcal{U}_J^I$ the intervals I and J are swapped. For the TMR example, for $\Phi = \mathcal{P}_{\geqslant 0.9}(\neg F \mathcal{U}_{[10,\infty)}^{[50,50]}\, F)$ we have $\Phi^{-1} = \mathcal{P}_{\geqslant 0.9}(\neg F \mathcal{U}_{[50,50]}^{[10,\infty)}\, F)$. We refer to [4] for a proof of this property, and to extensions of this result to some cases with zero rewards. Note that we excluded zero rewards here, since otherwise the model inversion would imply divisions by zero.

The duality result is the key to model check CRL on MRMs (satisfying the above restriction), since the swapping of formula implies that X_J turns into X^J, and $\mathcal{U}_J$ into $\mathcal{U}^J$. Hence, any CRL formula corresponds to a CSL formula interpreted on the dual MRM. As a consequence, model checking CRL can proceed via the algorithm for CSL, with some overhead (linear in the model size plus the formula length) needed to swap the model and swap the formula.

5.5 Model Checking CSRL-Specified Measures

For the general case of CSRL, model checking algorithms are more involved, and research on their effectiveness is ongoing [32,33]. In this section we describe the basic strategy and sketch three algorithms implementing this strategy. A more detailed comparison of the algorithmic intricacies can be found in [33].

Given an MRM $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ the model checking algorithm proceeds as in the earlier considered cases: in order to check whether state s satisfies the formula Φ, we recursively compute the set $Sat(\Phi)$ of states that satisfy Φ, and check whether s is a member of that set. Most of the cases have been discussed before in this paper, except for the handling of path operators with both time and reward intervals. For the sake of simplicity, we do not consider the next operator X, and we (again) restrict to formulas where all occurring intervals are of the form $[0, x]$, i.e., they impose upper bounds on the time or the cumulated reward, but no lower bound.

So, the question is how to compute $Prob(s, \Phi\, \mathcal{U}_{[0,r]}^{[0,t]}\, \Psi)$. Recall that in the CSL case, the crucial step has been to reduce the computation to instances of transient analysis. Indeed, it is possible to proceed in a similar way. In analogy to the CSL strategy, we can show that the above probability agrees with the probability $Prob(s, \mathsf{true}\, \mathcal{U}_{[0,r]}^{[t,t]}\, \Psi)$ on a transformed MRM where all Ψ-states and

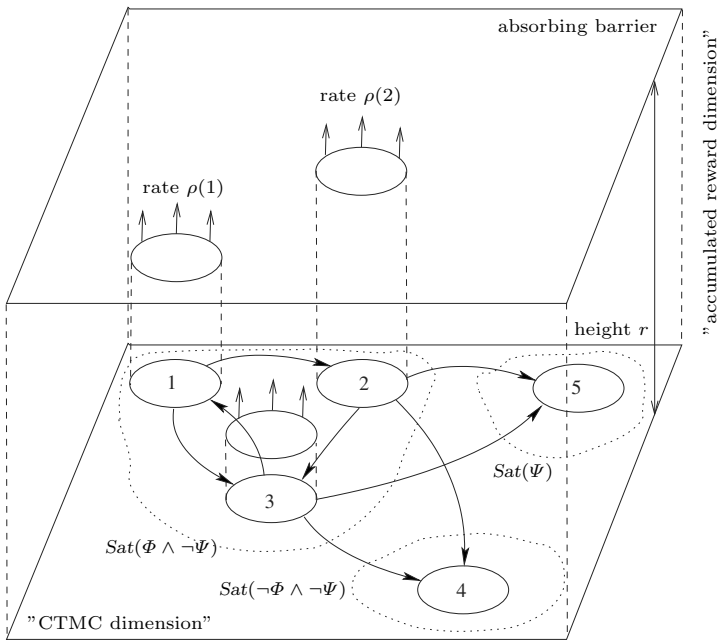


Fig. 3. Two-dimensional stochastic process $((X_t, Y_t), t \geq 0)$ for model checking $Prob(s, \Phi\, \mathcal{U}^{[0,t]}_{[0,r]}\, \Psi)$.

all $\neg(\Phi \wedge \Psi)$-states are made absorbing, and have reward 0 assigned to them. The intuitive justification is as in the CSL setting. The rewards are set to 0 since once a path reaches a Ψ-state at time $t' < t$, while not having accumulated more than r reward, it suffices to be trapped in that state until time t provided no reward will be earned anymore, i.e., $\rho(s) = 0$ for Ψ-state s. Note that we can amalgamate all states satisfying Ψ and all states satisfying $\neg(\Phi \wedge \Psi)$, thereby making the MRM considerably smaller.

Thus, we can restrict our attention to the computation of $Prob(s, \mathsf{true}\, \mathcal{U}^{[t,t]}_{[0,r]}\, \Psi)$. This probability, in turn, can be derived from the *transient accumulated reward distribution* of the MRM. (Compare this to the *transient distribution* used in the CSL case at this point.) To explain why this is the case, we consider a two-dimensional stochastic process $((X_t, Y_t), t \geq 0)$ on $S \times \mathbb{R}_{\geq 0}$, as illustrated in Figure 3. Informally speaking, this stochastic process has a discrete component that describes the transition behaviour in the original MRM, combined with a continuous component that describes the accumulated reward gained over time. For $t = 0$ we have $Y_t = 0$, and for $t > 0$ the value of Y_t increases continuously with rate $\rho(X_t)$. Hence, the discrete states of the original CTMC become "columns" of which the height models the accumulated reward. To take into account the reward bound ($\leq r$), we introduce an absorbing barrier

in the process whenever Y_t reaches the level r. Actually, we are interested in $\Pr\{Y_t \leqslant r, X_t \in S'\}$, i.e., the probability of being in a certain subset S' of states at time t, having accumulated a reward smaller than r. For our purposes, S' shall be chosen to be the set $Sat(\Psi)$ of states satisfying Ψ and we start the process in state s under consideration. We have that

$$Prob(s, \mathsf{true}\ \mathcal{U}^{[t,t]}_{[0,r]}\ \Psi) = \Pr\{Y_t \leqslant r, X_t \in Sat(\Psi)\}$$

for the transformed MRM described above. This allows us to decide the satisfaction of time- and reward-bounded until formulas via numerical recipes for calculating $\Pr\{Y_t \leqslant r, X_t \in S'\}$ on the two dimensional stochastic process (X_t, Y_t). It is worth to remark that similar processes (with mixed discrete-continuous state spaces) also emerge in the analysis of non-Markovian stochastic Petri nets (when using the supplementary variable approach, cf. [22]), Markov-regenerative stochastic Petri nets [9], and in fluid-stochastic Petri nets [42]. We briefly sketch three other approaches to compute $\Pr\{Y_t \leqslant r, X_t \in S'\}$ here, which are more directly applicable to the problem.

An Erlangian approximation. A first approach to compute $\Pr\{Y_t \leqslant r, X_t \in S'\}$ is to approximate the fixed reward bound r by a reward bound that is Erlang-k distributed with mean r. One may view this as some kind of discretisation of the continuous reward dimension into k steps. The main advantage of this approach is that the resulting model is both discrete-space and completely Markovian, and hence the techniques developed for CSL properties (cf. Section 4.3) can be used to approximate the required probabilities; reaching the reward bound in the original model corresponds to reaching a particular set of states in the approximated model. As a disadvantage we mention that an appropriate value for k (the number of phases in the Erlangian approximation) is not known *a priori*. Furthermore, when CSRL expressions are nested, it is yet unclear how the error in the approximation propagates. Furthermore, the resulting Markov chain becomes substantially larger, especially if k is large. On the other hand, the MRM can be described as a tensor product of two smaller MRMs, which can be exploited in the solution procedure (as far as the storage of the generator matrix is concerned).

With an Erlang-k distributed approximation of the reward bound together with uniformisation, the space complexity of this method is $\mathcal{O}(|S|^2 \cdot k^2)$, and the time complexity is $\mathcal{O}(N_\varepsilon \cdot |S|^2 \cdot k^2)$, where N_ε equals the number of steps required to reach a certain accuracy ε (which can be computed a priori). Note that N_ϵ determines the accuracy of only the transient analysis; it does *not* account for the (in-)accuracy of the approximation itself.

Discretisation. Recently, Tijms and Veldman [60] proposed a discretisation method for computing the transient distribution of the accumulated reward in an MRM. Their algorithm is a generalisation of an earlier algorithm by Goyal and Tantawi [24] for MRMs with only 0- and 1-rewards. The basic idea is to discretise both the time and the accumulated reward as multiples of the same step size d,

where d is chosen such that the probability of more than one transition in the MRM in an interval of length d is negligible. The algorithm allows only natural number rewards, but this is no severe restriction since rational rewards can be scaled to yield natural numbers.

The time complexity of this method is $\mathcal{O}(|S|\cdot t\cdot|(t-r)|/d^2)$ and the space complexity is $\mathcal{O}(|S|\cdot r/d)$. As the computational effort is proportional to d^{-2}, the computation time grows rapidly when a higher accuracy is required.

Occupation time distributions. In 2000, Sericola [57] derived a result for the distribution of occupation times in CTMCs prior to a given point in time t. The approach is based on uniformisation, and (as with uniformisation) it is possible to calculate an a priori error bound for the computed values. The distribution of this occupation time can be used to derive $\Pr\{Y_t \leqslant r, X_t \in S'\}$, based on the observation that if $O(s,t)$ is the occupation time of state s prior to t then $\rho(s) \cdot O(s,t)$ is the accumulated reward for this state prior to t. Summing over all states leads to the accumulated reward required.

The computation of the occupation time distribution is an iterative procedure, which in each iteration updates a linearly growing set of matrices. The computational and storage requirements of the approach are therefore considerable. If we truncate after the N_ε-th iteration, we obtain an overall time complexity of $\mathcal{O}(N_\varepsilon^3|S|^3)$ and an overall space complexity of $\mathcal{O}(N_\varepsilon^2|S|^3)$. Contrary to the Erlangian approximation, N_ϵ determines the accuracy of the entire computation procedure in this approach.

General observations. We have implemented all three algorithms, and experimented with them on a case study analysing the power consumption in ad-hoc mobile networks [33]. We can report the following observations:

- The three computational procedures converge to the same value, however, only for the occupation time distribution approach an a priori error bound (and hence a stopping criterion) is available.
- The method based on occupation time distributions is fast and accurate. In the current case study (which is small) we did not run into storage problems, however, the cubic storage requirements will limit this method to relatively small case studies.
- The discretisation method is slow when a fine-grain discretisation is used. Unfortunately, we have no method available (yet) to get a hold on the required step size to achieve a certain accuracy.
- The Erlangian approach is fast (where we did not even exploit the tensor structure in the generator matrix), but also here, we have to guess a reasonable number of phases for the approximation.
- The discretisation method suffers particularly from large time-bounds and large state spaces, as these make the number of matrices to be computed larger.
- The method based on occupation time distributions becomes less attractive when the time bound is large in comparison to the uniformisation rate. We

are currently investigating whether some kind of steady-state detection can be employed to shorten the computation in these cases.

5.6 Extending CSRL with Further Reward Operators

So far we have considered the accumulation of reward along paths, because as this is the basic novelty we support via the enriched path operators X_J^I and $\mathcal{U}_J^I$. In an orthogonal manner, it is possible to support further reward-based measures, namely by allowing further state operators.

To do so, consider state s in MRM $\mathcal{M}$. For time t and set of states S', the *instantaneous reward* $\rho^{\mathcal{M}}(s, S', t)$ equals $\sum_{s' \in S'} \pi^{\mathcal{M}}(s, s', t) \cdot \rho(s')$ and denotes the rate at which reward is earned in some state in S' at time t. The *expected (or long run) reward rate* $\rho^{\mathcal{M}}(s, S')$ equals $\sum_{s' \in S'} \pi^{\mathcal{M}}(s, s') \cdot \rho(s')$. We can now add the following state operators to our framework:

Expected reward rate $\mathcal{E}_J$: The operator $\mathcal{E}_J(\Phi)$ is true if the expected (long run) reward rate is in the interval J, if starting in state s:

$$s \models \mathcal{E}_J(\Phi) \text{ iff } \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi)) \in J.$$

Expected instantaneous reward rate $\mathcal{E}_J^t$: The operator $\mathcal{E}_J^t(\Phi)$ states that the expected instantaneous reward rate at time t lies in J:

$$s \models \mathcal{E}_J^t(\Phi) \text{ iff } \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi), t) \in J.$$

Expected cumulated reward $\mathcal{C}_J^I$: The operator $\mathcal{C}_J^I(\Phi)$ states that the expected amount of reward accumulated in Φ-states during the interval I lies in J:

$$s \models \mathcal{C}_J^I(\Phi) \text{ iff } \int_I \rho^{\mathcal{M}}(s, Sat^{\mathcal{M}}(\Phi), u)\ du \in J.$$

The inclusion of these operators in CSRL is possible because their model checking is rather straightforward. The first two formulas require the summation of the Φ-conforming steady-state or transient state probabilities multiplied with the corresponding rewards. The operator $\mathcal{C}_J^I(\Phi)$ can be evaluated using a variant of uniformisation [28,58]. Some example properties are now: $\mathcal{E}_J(\neg F)$, which expresses the expected reward rate, e.g., the system's capacity, for an operational system, $\mathcal{E}_J^t(\mathsf{true})$ expresses the expected instantaneous reward rate at time t and $\mathcal{C}_J^{[0,t]}(\mathsf{true})$ expresses the amount of accumulated reward up to time t.

The suggestion to include these operators into CSRL exemplifies how a pragmatic approach (providing new algorithms for new measures) can be combined with our logical approach, and can profit from the latter due to the ability of nesting state and path formulas in an arbitrary manner.

6 Stochastic Model Checking and Lumpability

This section is devoted to an important property that the CSRL logic family possesses. The property relates the well-known concepts of lumpability and bisimulation to the distinguishing power of the logic. We exemplify this property for CSRL, since this includes the other logics as subsets.

Bisimulation (lumping) equivalence. Lumpability enables the aggregation of CTMCs and MRMs without affecting performance properties [47,10,40,35]. We adapt the standard notion slightly in order to deal with MRMs with state-labellings. We only sketch the concepts here, and refer to the papers [4,5] for more details. For some MRM $\mathcal{M} = (S, \mathbf{R}, L, \rho)$ we say that an equivalence relation R on S is a *bisimulation* if whenever $(s, s') \in R$ then

$$L(s) = L(s') \text{ and } \rho(s) = \rho(s') \text{ and } \mathbf{R}(s, C) = \mathbf{R}(s', C) \text{ for all } C \in S/R,$$

where S/R denotes the quotient space under R and $\mathbf{R}(s, C) = \sum_{s' \in C} \mathbf{R}(s, s')$. States s and s' are said to be *bisimilar* iff there exists a bisimulation R that contains (s, s'). Thus, any two bisimilar states are equally labelled and the cumulative rate of moving from these states to any equivalence class C is equal. Since R is an equivalence relation, we can construct the quotient $\mathcal{M}/R$, often called the lumped Markov model of $\mathcal{M}$.

Example 5. The reflexive, symmetric and transitive closure of the relation

$$R = \{ (0111, 1011), (1011, 1101), (0011, 0101), (0101, 1001) \}$$

is a bisimulation on the set of states of the MRM depicted in Fig. 4. For convenience, double arrows are used to indicate that there exists a transition from a state to another state and vice versa. The lumped MRM $\mathcal{M}/R$ consists of five aggregated states, yielding, in fact, the MRM of the TMR system discussed in Example 1. For instance, state $s_{2,1}$ of the original model can be considered as the lumped state representing the three possible configurations in which, out of three, a single processor has failed. These configurations are represented in the detailed version of Fig. 4 by the states 0111, 1011, and 1101.

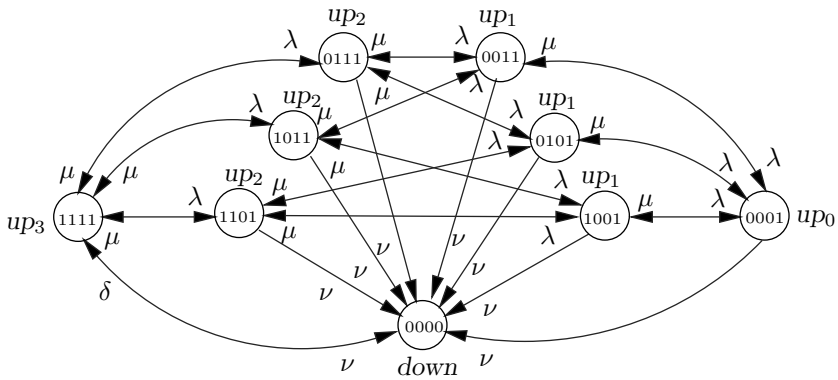


Fig. 4. A detailed version of the TMR model

It is well known that the measures derived from $\mathcal{M}$ and its quotient $\mathcal{M}/R$ are strongly related if R is a bisimulation. Without going into details, it is possible

to compute transient as well as steady state probabilities on the lumped MRM $\mathcal{M}/R$ if one is only interested in probabilities of equivalence classes. For a given MRM it is therefore possible to establish the following property [4,19,5]:

$$s \models \Phi \text{ iff } s' \models \Phi \quad \text{for all CSRL formulas } \Phi$$
$$\text{if and only if} \quad s \text{ and } s' \text{ are bisimilar.}$$

In other words, CSRL cannot distinguish between lumping equivalent states, but non-equivalent states can always be distinguished by some CSRL formula. This looks like a theoretical result, but it also has practical implications: it allows one to carry out model checking of CSRL (and CSL, and CRL) on the quotient state space with respect to lumpability. This lumped state space is often much smaller than the original one. It can be computed by a partition refinement algorithm [39,20].

7 Conclusions and Future Outlook

In this paper we have tried to give a tutorial style overview on the model checking approach to continuous time Markov chains and Markov reward models. While the logics CSL and CRL can be model checked using well-known numerical techniques to analyse Markov chains, more work is needed in the context of model checking performability properties expressible with CSRL to make the analysis effective.

Since the first paper on algorithms for CSL model checking has been published in 1999 [2] the approach has been implemented in (at least) three research tools, namely the ETMCC model checker [37], the model checker Prism, and the APNN toolbox [11]. While ETMCC is a dedicated CSL model checker based on sparse matrix data structures, Prism employs BDD based techniques to combat the state space explosion problem. The APNN toolbox uses Kronecker representations to achieve better space efficiency.

So far, we (and others) have applied stochastic model checking to various small and medium size case studies, including the analysis of a dependable workstation cluster [31], the verification of the performance of the plain ordinary telephone system protocol [3], the estimation of power consumption in mobile ad hoc networks [33], and the assessment of the survivability of the Hubble space telescope [34]. Among work that extends the basic stochastic model checking approach to a broader context, we are aware of the extension of CSL to process algebra specifications [38], to semi-Markov chains [44] and to random time bounds [49].

More work is foreseen in many exciting areas extending what has been described in this tutorial paper, ranging from research on the inclusion of nondeterminism, to efforts to improve the effectiveness of the algorithms described, to the application of stochastic model checking to realistic case studies.

Acknowledgements. The authors thank Joachim Meyer-Kayser, Markus Siegle and Lucia Cloth for valuable discussions and contributions. Holger Her-

manns is partially supported by the Netherlands Organization for Scientific Research (NWO) and Joost-Pieter Katoen is partially supported by the Dutch Technology Foundation (STW). The cooperation between the research groups in Aachen, Bonn, and Twente takes place as part of the Validation of Stochastic Systems (VOSS) project, funded by the Dutch NWO and the German Research Council DFG.

References

1. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continuous time Markov chains. *ACM Transactions on Computational Logic*, **1**(1): 162–170, 2000.
2. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Concurrency Theory*, LNCS 1664: 146–162, Springer-Verlag, 1999.
3. C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *Computer Aided Verification*, LNCS 1855: 358–372, Springer-Verlag, 2000.
4. C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *Automata, Languages, and Programming*, LNCS 1853: 780–792, Springer-Verlag, 2000.
5. C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking algorithms for continuous-time Markov chains. Technical report TR-CTIT-02-10. Centre for Telematics and Information Technology, University of Twente. 2001.
6. C. Baier and M. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Information Processing Letters*, **66**(2): 71–79, 1998.
7. M.D. Beaudry. Performance-related reliability measures for computing systems. *IEEE Transactions on Computers*, **C-27**: 540–547, 1978.
8. B. Bérard, M. Bidoit, A. Finkel, F. Laroussine, A. Petit, L. Petrucci, and Ph. Schnoebelen. *Systems and Software Verification.* Springer-Verlag, 2001.
9. A. Bobbio and M. Telek. Markov regenerative SPN with non-overlapping activity cycles. In *Proc. Int'l IEEE Performance and Dependability Symposium*: 124–133, 1995.
10. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, **31**: 59–75, 1994.
11. P. Buchholz, J.-P. Katoen, P. Kemper, and C. Tepper. Model checking large structured Markov chains. *Journal of Logic and Algebraic Programming*, to appear, 2001.
12. *CCITT Blue Book, Fascicle III.1*, International Telecommunication Union, Geneva, 1989.
13. G. Chiola, G. Franceschinis, R. Gaeta, and M. Ribaudo. GreatSPN 1.7: graphical editor and analyzer for timed and stochastic Petri nets. *Performance Evaluation*, **24** (1-2):47-68, 1995.
14. G. Ciardo, J.K. Muppala, and K.S. Trivedi. SPNP: stochastic Petri net package. In *Proc. 3rd Int. Workshop on Petri Nets and Performance Models*, pp. 142–151, IEEE CS Press, 1989.
15. G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601: 211–227, Springer-Verlag, 1999.

16. E. Clarke, E. Emerson, and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, **8**: 244–263, 1986.
17. E. Clarke, O. Grumberg, and D. Peled. *Model Checking*. MIT Press, 1999.
18. E. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, **33**(6): 61–67, 1996.
19. J. Desharnais and P. Panangaden. Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes. *Journal of Logic and Algebraic Programming*, to appear, 2001.
20. S. Derisavi, H. Hermanns, and W.H. Sanders. Optimal state space lumping in Markov models. 2002. submitted for publication.
21. W. Feller. *An Introduction to Probability Theory and its Applications*. John Wiley & Sons, 1968.
22. R. German. *Performance Analysis of Communication Systems: Modeling with Non-Markovian Stochastic Petri Nets*. John Wiley & Sons, 2000.
23. S. Gilmore and J. Hillston. The PEPA workbench: a tool to support a process algebra-based approach to performance modelling. In *Computer Performance Evaluation, Modeling Techniques and Tools*, LNCS 794: 353-368, Springer-Verlag, 1994.
24. A. Goyal and A.N. Tantawi. A measure of guaranteed availability and its numerical evaluation. *IEEE Transactions on Computers*, 37: 25–32, 1988.
25. W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In *Numerical Solution of Markov Chains*, pp. 357–371, Marcel Dekker Inc, 1991.
26. D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov chains. *Operations Research* **32**(2): 343–361, 1984.
27. D. Harel. Statecharts: a visual formalism for complex systems. *Science of Computer Programming*, **8**: 231–274, 1987.
28. B.R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1998.
29. B.R. Haverkort, R. Marie, G. Rubino, and K.S. Trivedi (editors). *Performability Modelling: Techniques and Tools*. John Wiley & Sons, 2001.
30. B.R. Haverkort and I. Niemegeers. Performability modelling tools and techniques. *Performance Evaluation*, **25**: 17–40, 1996.
31. B.R. Haverkort, H. Hermanns, and J.-P. Katoen. The use of model checking techniques for quantitative dependability evaluation. In *IEEE Symposium on Reliable Distributed Systems.*, pp. 228–238. IEEE CS Press, 2000.
32. B.R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking CSRL-specified performability properties. In *5th Int. Workshop on Performability Modeling of Computer and Communication Systems*, Erlangen, *Arbeitsberichte des IMMD*, **34** (13), 2001. 2001.
33. B.R. Haverkort, L. Cloth, H. Hermanns, J.-P. Katoen, and C. Baier. Model checking performability properties. In *Proc. IEEE Int'l Conference on Dependable Systems and Networks*, IEEE CS press, 2002.
34. H. Hermanns. Construction and verfication of performance and reliability models. *Bulletin of the EATCS*, **74**:135-154, 2001.
35. H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274(1-2):43–87, 2002.
36. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, **39**(1-4): 5–35, 2000.

37. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In *Tools and Algorithms for the Construction and Analysis of Systems*, LNCS 1785: 347–362, Springer-Verlag, 2000.
38. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In *Integrated Formal Methods*, LNCS 1945: 420–439, Springer-Verlag, 2000.
39. H. Hermanns and M. Siegle. Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In *Formal Methods for Real-Time and Probabilistic Systems*, LNCS 1601: 244–265, Springer-Verlag, 1999.
40. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
41. G.J. Holzmann. The model checker Spin. *IEEE Transactions on Software Engineering*, **23**(5): 279–295, 1997.
42. G. Horton, V. Kulkarni, D. Nicol, K. Trivedi. Fluid stochastic Petri nets: Theory, application and solution techniques. *Eur. J. Oper. Res.*, 105(1): 184–201,1998.
43. R.A. Howard. *Dynamic Probabilistic Systems; Volume 1: Markov Models.* John Wiley & Sons, 1971.
44. G.G. Infante-Lopez, H. Hermanns, and J.-P. Katoen. Beyond memoryless distributions: Model checking semi-Markov chains. In *Process Algebra and Probabilistic Methods*, LNCS 2165: 57–70, Springer-Verlag, 2001.
45. A. Jensen. Markov chains as an aid in the study of Markov processes. *Skandinavisk Aktuarietidskrift* **36**: 87–91, 1953.
46. J.-P. Katoen, M.Z. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In *Process Algebra and Probabilistic Methods*, LNCS 2165: 23–38, Springer-Verlag, 2001.
47. J.G. Kemeny and J.L. Snell. *Finite Markov Chains.* Van Nostrand, 1960.
48. V.G. Kulkarni. *Modeling and Analysis of Stochastic Systems.* Chapman & Hall, 1995.
49. M.Z. Kwiatkowska, G. Norman, and A. Pacheco. Model checking CSL until formulae with random time bounds. In *Process Algebra and Probabilistic Methods*, LNCS 2399, Springer-Verlag, 2002.
50. K.L. McMillan. *Symbolic Model Checking.* Kluwer Academic Publishers, 1993.
51. J.F. Meyer. On evaluating the performability of degradable computing systems. *IEEE Transactions on Computers*, **29**(8): 720–731, 1980.
52. J.F. Meyer. Closed-form solutions of performability, *IEEE Transactions on Computers*, 31(7): 648–657, 1982.
53. J.F. Meyer. Performability: a retrospective and some pointers to the future. *Performance Evaluation*, 14(3-4): 139–156, 1992.
54. W.D. Obal II and W.H. Sanders. State-space support for path-based reward variables. *Performance Evaluation*, **35**: 233–251, 1999.
55. D. Peled. *Software Reliability Methods.* Springer-Verlag, 2001.
56. W.H. Sanders, W.D. Obal II, M.A. Qureshi, and F.K. Widnajarko. The UltraSAN modeling environment. *Performance Evaluation*, **24**: 89–115, 1995.
57. B. Sericola. Occupation times in Markov processes. *Stochastic Models*, 16(5): 339–351, 2000.
58. E. de Souza e Silva and H.R. Gail. Performability analysis of computer systems: from model specification to solution. *Perf. Ev.*, **14**: 157–196, 1992.
59. W.J. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton University Press, 1994.
60. H.C. Tijms, R. Veldman. A fast algorithm for the transient reward distribution in continuous-time Markov chains, *Operation Research Letters*, 26: 155–158, 2000.