On the Use of Model Checking Techniques for Dependability Evaluation

Boudewijn R. Haverkort
Dept. of Computer Science, RWTH Aachen, D-52056 Aachen, Germany
(haverkort@cs.rwth-aachen.de)

Holger Hermanns, Joost-Pieter Katoen
Dept. of Computer Science, University of Twente, P.O. Box 217, 7500 AE Enschede, The Netherlands
({hermanns|katoen}@cs.utwente.nl)

Abstract

Over the last two decades many techniques have been developed to specify and evaluate Markovian dependability models. Most often, these Markovian models are automatically derived from stochastic Petri nets, stochastic process algebras, or stochastic activity networks. However, whereas the model specification has become very comfortable, the specification of the dependability measures of interest most often has remained fairly cumbersome. In this paper we show that our recently introduced logic CSL (continuous stochastic logic) provides ample means to specify state- as well as path-based dependability measures in a compact and flexible way. Moreover, due to the formal syntax and semantics of CSL, we can exploit the structure of CSL-specified dependability measures in the dependability evaluation process. Typically, the underlying Markov chains that need to be evaluated can be reduced considerably in size by this structure exploitation.

1. Introduction

Over the last two decades many techniques have been developed to specify and solve dependability models. These methods are either of a combinatorial nature (fault-trees, reliability block diagrams) or numerically oriented [22]. In the latter case, most of the models addressed assume as underlying stochastic process a continuous-time Markov chain. To avoid the specification of dependability models directly at the state level, high-level (dependability) model 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 e.g. provided by TIPPtool [16], UltraSAN [23] or SPNP [6], it is relatively comfortable to specify dependability models of which the underlying 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 dependability models has become very comfortable, the specification of the dependability measures of interest most often has remained fairly cumbersome. In particular, most often only simple state-based dependability 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 (Computational Tree Logic; see below) also allows one to express properties over state sequences. Such capabilities would also be welcome in specifying dependability measures.

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

We have introduced CSL (including its complete syntax

and formal semantics) in a much more theoretical context as we do here (cf. [4, 2]). Aim of the current paper is to show the suitability of our approach in an application oriented context.

2. System evaluation techniques

2.1. Performance and dependability evaluation

Performance evaluation aims 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.

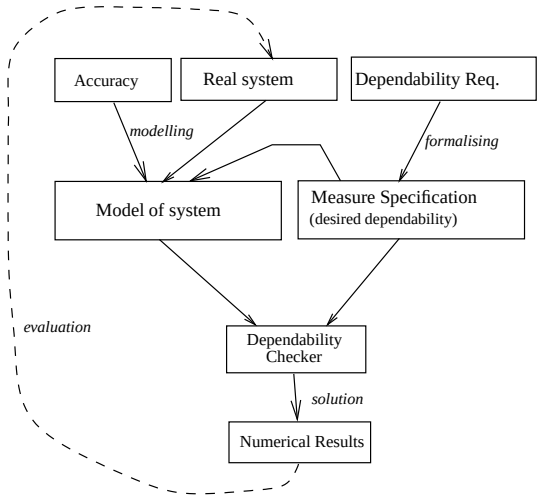

Figure 1. The model-based performance and dependability evaluation cycle

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 simulative 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 [13]. 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. In the sequel of this paper we will specifically address the question how to do this differently.

2.2. Continuous-time Markov chains

Since CTMCs play a central role in our approach, we briefly recapitulate their basic concepts and introduce some notation. A CTMC is a tuple $\mathcal{M} = (S, \mathbf{R})$ where S is a finite set of *states* 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.

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

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, $\sigma = s_0, t_0, s_1, t_1, s_2, t_2, \ldots$ 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

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

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 [4].

For a CTMC 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

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

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* [11, 10]. With uniformisation, the transient probabilities of a CTMC are computed via a uniformised DTMC which characterises the CTMC at 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 set S'. 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 [21], we are not aware of suitable mechanisms to express such measures. In what follows in this paper, we will specifically address this issue.

2.3. Formal verification

Whereas performance and dependability analysis is focused on answering questions concerning quantitative system issues, i.e., performance and dependability, 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? or (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.

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.

2.4. Model checking

What is model checking? In this paper we concentrate on model checking [8, 19]. This 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 model, thereby using effective methods to combat the infamous state-space explosion problem.

Using model checking, the user inputs a description of a model of the system (the "possible behaviour") and a description of the requirements specification (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, the counter-example provides 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) before continuing. If no errors are found, the user can refine the model description (e.g. by taking more design decisions into account, so that the model becomes more concrete/realistic) and can restart the verification process. Notice at this point the similarity between Figures 1 and 2.

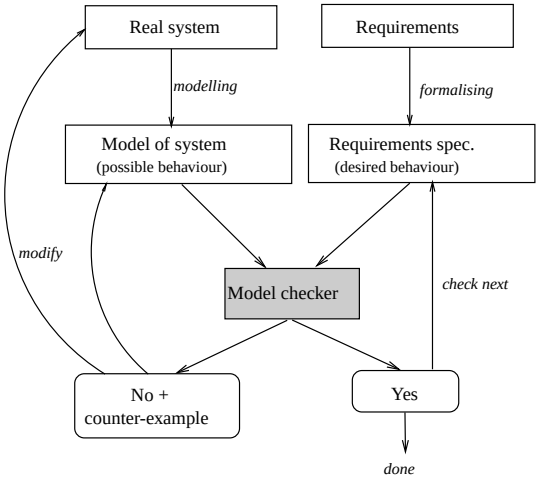

Figure 2. The model checking approach

Typically, models of systems are finite-state automata, where transitions model the evolvement 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 [18] or Statecharts [12]. 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; 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 [7] (Computational Tree Logic). This logic allows to state properties over *states*, and over *paths* using the following syntax:

State-formulas

$$\Phi ::= a \mid \neg\Phi \mid \Phi \vee \Phi \mid \mathsf{E}\,\varphi \mid \mathsf{A}\,\varphi$$

a	atomic proposition
$\mathsf{E}\,\varphi$	there *Exists* a path that fulfills φ
$\mathsf{A}\,\varphi$	*All* paths fulfill φ

Path-formulas

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

$X\,\Phi$	the *neXt* state fulfills Φ
$\Phi\,\mathcal{U}\,\Psi$	Φ holds along the path, *Until* Ψ holds
$\Diamond\Phi$	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 $\mathsf{E}\,\varphi$ is valid in state s if there *exists* some path starting in s and satisfying φ. The formula $\mathsf{E}\,\Diamond$*deadlock*, for example, expresses that for some system run eventually a deadlock can be reached (potential deadlock). On the contrary, $\mathsf{A}\,\varphi$ is valid if *all* paths satisfy φ; $\mathsf{A}\,\Diamond$*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.

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*(Φ) 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*(Φ) and *Sat*(Ψ), and then intersecting these sets; *Sat*$(\neg\Phi)$ is obtained by taking the complement of the entire state space wrt. *Sat*(Φ). The algorithm for the temporal operators is slightly more involved. For instance, for *Sat*$(\mathsf{EX}\,\Phi)$ we first compute the set *Sat*(Φ) and then compute those states from which one can move to this set by a single transition. *Sat*$(\mathsf{E}\,(\Phi\,\mathcal{U}\,\Psi))$ is computed in an iterative way: (i) as a precomputation we determine *Sat*(Φ) and *Sat*(Ψ); (ii) we start the iteration with *Sat*(Ψ) as these states will surely satisfy the property of interest; (iii) we extend this set by the states in *Sat*(Φ) that 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 $\mathsf{E}\,\Box$-operator [7]) 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 [9].

3. Model checking CSL

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.

3.1. A logic for dependability

To specify dependability measures as logical formulas over CTMCs, we have to: (i) extend CTMCs with a labelling function $L : S \to 2^{AP}$ which assigns to each state $s \in S$ the set $L(s)$ of atomic propositions that are valid in s, and (ii) extend CTL into CSL, a stochastic variant of CTL.

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 $\bowtie$ a comparison operator, i.e., $\bowtie \in \{\leqslant, \geqslant\}$. The syntax of CSL now becomes:

State-formulas

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

$\mathcal{S}_{\bowtie p}(\Phi)$	prob. that Φ holds in steady state $\bowtie p$
$\mathcal{P}_{\bowtie p}(\varphi)$	prob. that a path fulfills $\varphi \bowtie p$

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 fulfills Φ
$\Phi\,\mathcal{U}^I\,\Psi$	Φ holds along the path until Ψ holds at time $t \in I$

The state-formula $\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability for the set of Φ-states meets the bound $\bowtie\ p$. The operator $\mathcal{P}_{\bowtie p}(.)$ replaces the usual CTL path quantifiers E and A. In fact, for most cases $\mathsf{E}\,\varphi$ can be written as $\mathcal{P}_{>0}(\varphi)$ and $\mathsf{A}\,\varphi$ as $\mathcal{P}_{\geqslant 1}(\varphi)$. These rules are not generally applicable due to fairness considerations [5]. $\mathcal{P}_{\bowtie p}(\varphi)$ asserts that the probability measure of the paths satisfying φ meets the bound $\bowtie\ p$. Temporal operators like $\Diamond$, $\Box$ and their real-time variants $\Diamond^I$ or $\Box^I$ can be derived, e.g., $\mathcal{P}_{\bowtie p}(\Diamond^I\,\Phi) = \mathcal{P}_{\bowtie p}(\text{true}\ \mathcal{U}^I\,\Phi)$ 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:

$s \models a$	iff $a \in L(s)$
$s \models \neg\Phi$	iff $s \not\models \Phi$
$s \models \Phi_1 \vee \Phi_2$	iff $s \models \Phi_1 \vee s \models \Phi_2$
$s \models \mathcal{S}_{\bowtie p}(\Phi)$	iff $\pi(s, Sat(\Phi)) \bowtie p$
$s \models \mathcal{P}_{\bowtie p}(\varphi)$	iff $Prob(s,\varphi) \bowtie p$.

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

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

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.

3.2. State-based measures

What types of performance and dependability properties 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. Let us first consider how we can specify constraints on standard performance measures such as steady-state and transient-state measures.

Steady-state measures. Assume that for each state s we have an atomic proposition *in*(s) that is valid in s and

invalid in any other state. The state-formula $\mathit{in}(s)$ thus uniquely characterises the state s. The formula $\mathcal{S}_{\bowtie p}(\mathit{in}(s))$ now imposes a requirement on the steady-state probability to be in state s. For instance, $\mathcal{S}_{\leqslant 10^{-5}}(\mathit{in}(s'))$ is valid in state s if the steady-state probability of state s' (when starting in state s) is at most 0.00001.

This can be easily generalised towards selecting sets of states by using more general state-formulas than the simple atomic propositions. The formula $\mathcal{S}_{\bowtie p}(\Phi)$ imposes a constraint on the probability to be in some Φ-state in the long run. For instance, if the propositions *2up* and *3up* characterise the states of a fault-tolerant multi-processor system in which 2 resp. 3 processors are operational, the formula $\mathcal{S}_{\geqslant 0.99}(\mathit{2up} \vee \mathit{3up})$ states that in the long run, for at least 99% of the time 2 or 3 processors are *up*.

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

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

More specifically, $\mathcal{P}_{\bowtie p}(\Diamond^{[t,t]}\ \mathit{in}(s'))$ is valid in state s if the transient probability at time t to be in state s' satisfies the bound $\bowtie\ p$. For instance, $\mathcal{P}_{\leqslant .2}(\Diamond^{[t,t]}\ \mathit{in}(s))$ requires that the transient probability of state s at time t is at most 0.2.

In a similar way as done for steady-state measures, the formula $\mathcal{P}_{\geqslant 0.99}(\Diamond^{[t,t]}\ \mathit{2up} \vee \mathit{3up})$ states that the probability to have 2 or 3 processors running at time t is at least 0.99. For specification convenience, a transient-state operator

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

could be defined [2]. It states that the probability for a Φ-state at time t meets the bound $\bowtie p$.

3.3. Path-based measures

Note that 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 requires an arbitrary path-formula as an argument, much more general measures can be described as well. An example of a measure is the probability of reaching a certain set of states provided that all paths to these states obey certain properties. For instance, the formula

$$\mathcal{P}_{\leqslant 0.01}((\mathit{2up} \vee \mathit{3up})\ \mathcal{U}^{[0,10]}\ \mathit{down})$$

is valid in state s, i.e., when starting in state s, if the probability of the system being *down* in 10 time-units after continuously operating with 2 or 3 processors is at most 0.01. Of course, by nesting the $\mathcal{P}$- and $\mathcal{S}$-operators rather complex measures can be specified.

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. This is of utmost importance for the automation of the analysis of such constraints. Secondly, an interesting aspect of our specification technique is the possibility to state performance and dependability requirements over a selective set of paths through a model.

3.4. Computing 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 $\mathit{Sat}(\Phi)$ 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 $\mathit{Sat}(\Phi)$ for the $\mathcal{S}$ and $\mathcal{P}$-operators. We deal with these operators separately.

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

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

Thus, checking whether state s satisfies $\mathcal{S}_{\bowtie p}(\Phi)$, 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 there 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 $\mathcal{S}_{\bowtie p}(\Phi)$ it then suffices to verify

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

where $\pi^B(s')$ denotes the steady-state probability of s' in BSCC B, and $Prob(s, \Diamond B)$ is the probabilkity to reach

BSCC B from state s. To compute these probabilities, standard methods for steady-state and graph analysis can be used.

3.4.2. 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 $Prob(s, \varphi)$:

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

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

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

That is, 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 $\mathit{Sat}(X^I\ \Phi)$ we first recursively compute $\mathit{Sat}(\Phi)$ and add state s to $\mathit{Sat}(X^I\ \Phi)$ if it fulfills (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 $s \in \mathit{Sat}(\Psi)$, (ii) 0, if $s \notin \mathit{Sat}(\Phi) \cup \mathit{Sat}(\Psi)$, and

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

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 $\mathcal{P}_{\bowtie p}(\Phi\, \mathcal{U}\ \Psi)$ by first computing recursively the sets $\mathit{Sat}(\Phi)$ and $\mathit{Sat}(\Psi)$ followed by solving a linear system of equations.

Solving the Volterra integral equation. We now concentrate on effective techniques for solving the Volterra integral equation system (2). As a first effective 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 (i.e., between 10^{-6} and 10^{-9}). Numerical stability is another issue of concern when using this method [17].

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:

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

Thus, for computing $Prob(s, \text{true}\ \mathcal{U}^{[t,t]}\ \Phi)$ 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 $\varphi' = \text{true}\, \mathcal{U}^{[t,t]}\ \Psi$ 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 $\mathcal{M}$):

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

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 [2].

The reduction of the model checking problem for the time-bounded until-operator to the transient analysis of a CTMC has several advantages: (i) it avoids the awkward numerical integration of (2), (ii) it allows to use efficient techniques such as uniformisation, and (iii) it employs a

measure-driven transformation (mostly: reduction) of the CTMC.

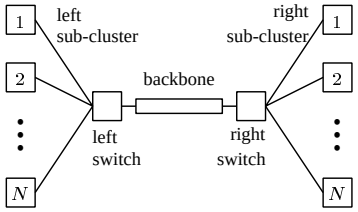

Figure 3. A dependable cluster of workstations

4. Dependability of a workstation cluster

We illustrate the model checking approach for dependability evaluation by means of a simple case study.

System description. We consider a dependable cluster of workstations, cf. Fig. 3. Two sub-clusters are connected via a backbone connection. Each sub-cluster consists of N workstations, connected in a star topology with a central switch that provides the interface to the backbone.

Each of the components of the system (workstations, switches, and backbone) can break down. There is single repair unit (not shown in Fig. 3) that takes care of repairing failed components. The given system structure is inspired by workstation clusters used in contemporary mobile telephony switching equipment [20]. The computing power of the cluster is over-dimensioned, in order to be able to accommodate varying levels of traffic volume, as well as to cope with component failures. The system operation is subject to the following informal constraints:

- In order to provide *minimum* quality of service (QoS), at least k ($k < N$) workstations have to be operational, and these workstations have to be connected to each other via operational switches. If in each sub-cluster the number of operational workstations drops below k then an operational backbone is required to ensure that in total at least k operational workstations are still connected to provide minimum service.
- *Premium* quality of service requires at least N operational workstations, with the same connectivity constraints as mentioned above.

A GSPN model. In order to assess the extent to which these constraints are met during operation, we develop a CTMC of the system, and model check different CSL requirements reflecting the above constraints on the chain. The CTMC is derived in the standard way [1] from a generalized stochastic Petri net (GSPN) representing the stochastic evolution of failures and repairs. The GSPN structure is depicted in Fig. 4 and is very similar to a model studied in [14]. The first two "rows" represent the two groups of workstations. For each of the groups, individual workstations can fail (transition WorkstationFail). Once failed, the workstation is Down, and the repair unit is needed to repair the workstation. The repair unit is depicted at the bottom of the figure. If a repair unit (token) is Available, repair starts almost immediately (transition WorkstationInspect), the workstation is InRepair. Once repaired successfully (transition WorkstationRepair), the workstation is Up again. The evolution of the other components of the system is very similar. The next two "rows" in Fig. 4 represent the behaviour of the two switches, and the last row represents the backbone. Note that a repair priority strategy could be easily introduced in the model by assigning different priorities to the "*inspect*"-transitions; we do not consider this any further here.

Dependability measures of interest. This section contains a number of interesting CSL requirements that give a flavor of the typical properties expressible in CSL. To start with, we fix the set of atomic propositions. Since the state space S of the CTMC underlying the GSPN model is given by the set of reachable markings of the GSPN, it appears natural to allow atomic propositions to range over the individual marking of each place of the GSPN. Thus, for some place P and natural number n we use $\#\mathsf{P} \bowtie n$ as the atomic proposition that is valid in all reachable markings where place P contains $\bowtie n$ tokens. We use the abbreviations as listed in Table 1.

It should be obvious that the state formulas *Minimum* and *Premium* correspond to the informal QoS constraints mentioned before. We are now in the position to study the dependability of the system, via the following example CSL requirements:

Φ_1. On the long run, premium QoS will be delivered with at least probability 0.70:

$$\mathcal{S}_{\geqslant 0.70}(\mathit{Premium})$$

Φ_2. On the long run, the chance that QoS is below minimum quality is less than 5%:

$$\mathcal{S}_{<0.05}(\neg \mathit{Minimum})$$

Φ_3. The system will always be able to offer premium QoS at some point in the future:

$$\mathcal{P}_{\geqslant 1}(\Diamond \mathit{Premium})$$

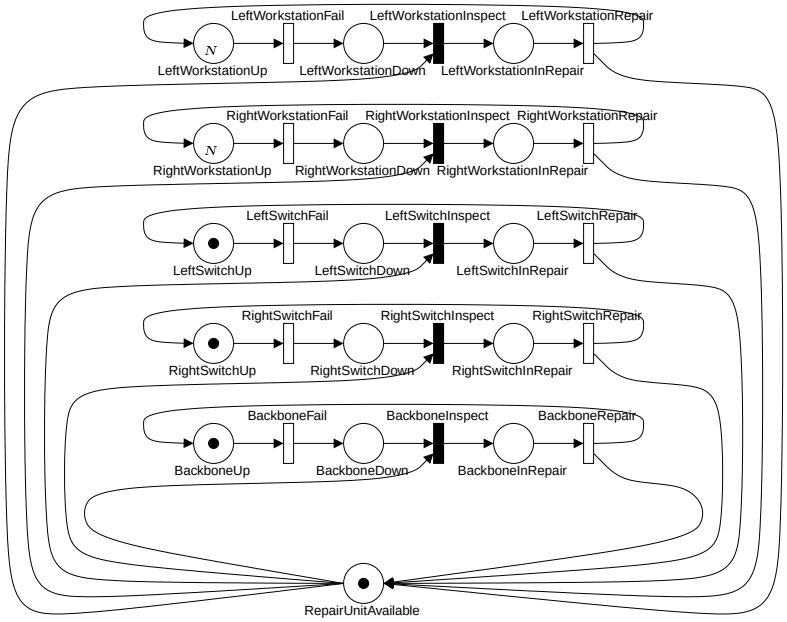

Figure 4. GSPN model of the workstation cluster

Table 1. Abbreviations of basic formulas for the workstation cluster

$$
\begin{array}{lcl}
\mathit{LeftOperational}_i & = & (\#\mathsf{LeftWorkstationUp} \geqslant i) \wedge (\#\mathsf{LeftSwitchUp} > 0) \\
\mathit{RightOperational}_i & = & (\#\mathsf{RightWorkstationUp} \geqslant i) \wedge (\#\mathsf{RightSwitchUp} > 0) \\
\mathit{Conn} & = & (\#\mathsf{LeftSwitchUp} > 0) \wedge (\#\mathsf{BackboneUp} > 0) \wedge (\#\mathsf{RightSwitchUp} > 0) \\
\mathit{Operational}_i & = & (\#\mathsf{LeftWorkstationUp} + \#\mathsf{RightWorkstationUp} \geqslant i) \wedge \mathit{Conn} \\
\mathit{Minimum} & = & \mathit{LeftOperational}_k \vee \mathit{RightOperational}_k \vee \mathit{Operational}_k \\
\mathit{Premium} & = & \mathit{LeftOperational}_N \vee \mathit{RightOperational}_N \vee \mathit{Operational}_N
\end{array}
$$

Φ_4. The chance that QoS drops below minimum quality within 85 time units is less than 10%:

$$\mathcal{P}_{<0.10}(\Diamond^{[0,85]} \neg \mathit{Minimum})$$

Φ_5. If facing insufficient QoS, the probability to face the same problem after 2 time units is less than 0.30:

$$\neg \mathit{Minimum} \Rightarrow \mathcal{P}_{<0.30}(\Diamond^{[2,2]} \neg \mathit{Minimum})$$

Φ_6. The probability to turn from minimum QoS to premium QoS without violating the minimum QoS constraint on the way is more than 99%:

$$\mathit{Minimum} \wedge \neg \mathit{Premium} \Rightarrow$$

$$\mathcal{P}_{\geqslant 0.99}(\mathit{Minimum}\ \mathcal{U}\ \mathit{Premium})$$

Φ_7. The probability to need more than 15 time units to recover from insufficient QoS is at most 20%:

$$\neg \mathit{Minimum} \Rightarrow$$

$$\mathcal{P}_{\leqslant 0.20}(\neg \mathit{Minimum}\ \mathcal{U}^{\,[15,\infty)}\ \mathit{Minimum})$$

To give a few examples of nested formulas: $\mathcal{P}_{>0.75}(\Box\ \Phi_7)$ ensures the above recovery requirement to hold for more than 75% of the future evolution, and $\mathcal{S}_{>0.75}(\Phi_7)$ requires this for steady state only.

It is worth to mention that one can omit the outermost probability bounds, because the model checking algorithm returns the actual probability of satisfying sub-formulas. In this way, queries such as: "What is the probability that ...?' can be posed instead of ''The probability is ...?".

Some experimental results. The following average durations have been assumed for the case study.

BackboneFail	5000h	BackboneRepair	8h
LeftSwitchFail	4000h	RightSwitchRepair	4h
RightSwitchFail	4000h	LeftSwitchRepair	4h
RightWorkstationFail	500h	RightWorkstationRepair	0.5h
LeftWorkstationFail	500h	LeftWorkstationRepair	0.5h

Note that the Inspect activities have an average duration of 6 minutes only. We verified the properties Φ_1 to Φ_7, using a prototypical implementation of the model checking algorithm of [2] written in C, except for Φ_3 and Φ_6. The two properties were verified using the model checker $\mathsf{E} \vdash \mathsf{MC}^2$ [17]. The measurements report average CPU runtimes on a 500Mhz Pentium III laptop with 128 MB main memory. Table 2 displays results obtained for different model sizes, determined by the number N of (both left and right) workstations initially available. The threshold k for minimum QoS has been set to 3.

All runtimes, except for Φ_3 and Φ_6, are average runtimes for checking a single state; for Φ_3 and Φ_6 all states are checked simultaneously. The latter properties do not require much verification time as no time constraints are involved in the formulas to be checked. Time-bounded until-formulas require considerably more computation time.

Table 2. Computation time (sec) to model check CSL properties

N	size	Φ_1	Φ_2	Φ_3	Φ_4	Φ_5	Φ_6	Φ_7
4	820	1.11	1.07	0.00	0.25	0.23	0.03	0.31
8	2772	3.56	3.33	0.00	2.18	1.05	0.11	1.37
16	10132	13.34	12.91	0.05	11.47	3.95	0.49	5.70
32	38676	60.35	59.10	0.21	52.21	16.28	2.79	22.33

5. Summary and outlook

In this paper we have shown that the stochastic logic CSL can be used to specify state- and path-based dependability properties of systems being modelled as CTMC (or with any high-level specification technique that maps to a CTMC) in a convenient way. As shown in detail in [2] this formal way of specifying dependability properties allows us to automatically tailor (i.e., minimise) the CTMCs to be analysed. The approach has been illustrated by means of a case study of a dependable workstation cluster.

We have recently enhanced our logic to deal with reward-based measures in order to handle performability models [3]. A first prototype tool has recently been presented [17]. We are currently considering the measure-driven generation of the CTMC from a high-level model description.

Acknowledgements: The authors thank Christel Baier, Joachim Meyer-Kayser and Markus Siegle for their contribution.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.

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

[3] C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. On the logical characterisation of performability properties. In *Automata, Languages, and Programming (ICALP)*, LNCS 1853, 2000.

[4] C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *Concurrency Theory (CONCUR)*, LNCS 1664, pp. 146–162, 1999.

[5] C. Baier and M. Kwiatkowska. On the verification of qualitative properties of probabilistic processes under fairness constraints. *Inf. Proc. Letters*, **66**(2): 71–79, 1998.

[6] G. Ciardo, J. Muppala and K.S. Trivedi, SPNP: Stochastic Petri Net Package. *Proc. 3rd Int. Workshop on Petri Nets and Perf. Models (PNPM)*, IEEE CS Press, pp. 142–151, 1989.

[7] E. Clarke, E. Emerson and A. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Trans. on Progr. Lang. and Sys.*, **8**: 244–263, 1986.

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

[9] E. Clarke and R. Kurshan. Computer-aided verification. *IEEE Spectrum*, **33**(6): 61–67, 1996.

[10] W.K. Grassmann. Finding transient solutions in Markovian event systems through randomization. In W.J. Stewart, ed, *Num. Sol. of Markov Chains*, pp. 357–371, Marcel Dekker, 1991.

[11] D. Gross and D.R. Miller. The randomization technique as a modeling tool and solution procedure for transient Markov chains. *Oper. Res.* **32**(2): 343–361, 1984.

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

[13] B.R. Haverkort and I.G. Niemegeers, Performability modelling tools and techniques. *Perf. Ev.*, **25**: 17–40, 1996.

[14] B.R. Haverkort. Approximate performability and dependability analysis using generalized Petri nets. *Perf. Ev.*, **18**: 61–78, 1993.

[15] B.R. Haverkort. *Performance of Computer Communication Systems: A Model-Based Approach*. John Wiley & Sons, 1998.

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

[17] 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 (TACAS)*, LNCS 1785, pp. 347–362, 2000.

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

[19] J.-P. Katoen. Concepts, algorithms and tools for model checking. Tech. Rep. vol. 32(1), Univ. Erlangen-Nürnberg, 1999.

[20] V. Mertsiotakis, personal communication, 2000.

[21] W.D. Obal II and W.H. Sanders. State-space support for path-based reward variables. *Perf. Ev.*, **35**: 233–251, 1999.

[22] R. Sahner, K.S. Trivedi and A. Puliafito. *Performance and Reliability Analysis of Computer Systems: An Example-Based Approach using the SHARPE Software Package*. Kluwer, 1996.

[23] W.H. Sanders, W.D. Obal II, M.A. Qureshi and F.K. Widnajarko. The UltraSAN Modeling Environment. *Perf. Ev.*, **24**: 89–115, 1995.

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