Performance Evaluation := (Process Algebra + Model Checking) × Markov Chains

Holger Hermanns and Joost-Pieter Katoen

Formal Methods and Tools Group, Faculty of Computer Science, University of Twente
P.O. Box 217, 7500 AE Enschede, The Netherlands

Abstract. Markov chains are widely used in practice to determine system performance and reliability characteristics. The vast majority of applications considers continuous-time Markov chains (CTMCs). This tutorial paper shows how successful model specification and analysis techniques from concurrency theory can be applied to performance evaluation. The specification of CTMCs is supported by a stochastic process algebra, while the quantitative analysis of these models is tackled by means of model checking. Process algebra provides: (i) a high-level specification formalism for describing CTMCs in a precise, modular and constraint-oriented way, and (ii) means for the automated generation and aggregation of CTMCs. Temporal logic model checking provides: (i) a formalism to specify complex measures-of-interest in a lucid, compact and flexible way, (ii) automated means to quantify these measures over CTMCs, and (iii) automated measure-driven aggregation (lumping) of CTMCs. Combining process algebra and model checking constitutes a coherent framework for performance evaluation based on CTMCs.

1 Introduction

What is performance evaluation? Performance evaluation aims at analysing quantitative system aspects that are related to its performance and dependability – what is the frequency of anomalous behaviour?, or, is correct and timely packet delivery guaranteed in at least 92% of all cases? Major performance evaluation approaches are *measurement-based* and *model-based* techniques. In measurement-based techniques, controlled experiments are performed on a concrete (prototypical) realisation of the system, and gathered timing information is analysed to evaluate the measure(s) of interest such as time-to-failure, system throughput, or number of operational components. In model-based performance evaluation, an abstract (and most often approximate) model of the system is constructed that is just detailed enough to evaluate the measure(s) of interest with the required accuracy. Depending on modelling flexibility and computational requirements, either analytical, numerical or simulative techniques are used to evaluate the required measure(s). We focus on model-based performance evaluation and their numerical analysis.

Models and measures. Continuous-time Markov chains (CTMCs) are a widely used performance evaluation model. They can be considered as labelled transition systems, where the transition labels – rates of exponential distributions – indicate the speed of the system evolving from one state to another. Using specification techniques such as queueing networks [19], stochastic Petri nets [1] or stochastic networks [42], CTMCs can be described in a quite comfortable way. Typical performance measures of CTMCs are based on steady-state and transient-state probabilities. Steady-state probabilities refer to the system behaviour on the "long run", i.e., when the system has reached an equilibrium. Transient-state probabilities consider the system at a fixed time instant t. State-of-the-art numerical algorithms allow the computation of both kinds of probabilities with relative ease and comfortable run times, and in a quantifiable precise manner. Several software-tools are available to support the specification and analysis of CTMCs.

Performance evaluation and concurrency theory: a couple? Given the success of CTMCs and their wide industrial applications, it is stunning that these models have received scant attention in concurrency theory for a long time: where probabilistic aspects have come into play, they were mostly of a purely discrete nature. This is even more remarkable, as model specification and analysis techniques – key ingredients of performance evaluation methodology – are first class examples of the success of formal methods for concurrent or reactive systems. Moreover, in modern systems many relevant functionalities are inextricably linked to performance aspects and the difference between functional and performance properties has become blurred. While formal methods for concurrency have mainly been focused on functional features, we believe that these methods have finally reached a state in which performance aspects should play a larger rôle. As a – to our opinion – promising example of the cross-fertilisation of concurrency theory and performance evaluation, this paper surveys a formal framework for the specification and analysis of CTMCs. The proposed methodology is based on appropriate extensions of *process algebra* for the description of CTMCs and *model checking* techniques for their analysis.

Stochastic process algebra. Stochastic process algebras are extensions of process algebras such as ACP, CCS and CSP in which actions can be delayed according to some negative exponential distribution. A mapping from algebraic terms onto CTMCs is obtained by a formal semantics in traditional SOS-style. The process algebraic setting provides a specification formalism for describing CTMCs in a precise, and modular way, resembling the hierarchical nature of most modern systems. In this setting, strong bisimulation coincides with lumpability, a notion central to the aggregation of Markov chains. The computation of this bisimulation equivalence can be performed using small adaptations of existing algorithms for computing strong bisimulation without an increase of their worst-case complexity. This provides means to minimise CTMCs (w.r.t. lumpability) in an efficient and fully automated way – a result that was unknown in performance evaluation. The congruence property of bisimulation allows this minimisation to

be carried out in a compositional fashion, i.e., component-wise, thus avoiding an a priori generation of the entire (and possibly huge) state space. An appropriate choice of the basic algebraic operators supports:

- an *orthogonal* extension of traditional process algebra, yielding a *single framework* for the description of both functional and performance aspects;
- the specification of exponential and *non-exponential* distributions such as the rich class of phase-type distributions;
- the *constraint-oriented* specification of stochastic time constraints, i.e., without modifying existing untimed specifications;
- variants of *weak bisimulation* congruences (and their algorithms) that combine lumpability and abstraction of sequences of internal actions.

Model checking. The process algebraic specification of the performance model can be complemented by a specification of the performance measure(s)-of-interest in a stochastic variant of the branching-time temporal logic CTL. This logic is formally interpreted over CTMCs and allows to express quantifiable correctness criteria such as: in 99% of the cases no deadlock will be reached within t time units. The logic-based method provides ample means for the unambiguous and lucid specification of requirements on steady-state and transient-state probabilities. Besides, it allows for the specification of path-based properties, measures that in performance evaluation are described informally in an ad-hoc manner and mostly require a manual tailoring of the model. To check the validity of formulas, model checking algorithms are adapted. They are enriched with appropriate numerical means, such as simple matrix manipulations and solution techniques for linear systems of equations, to reason about probabilities. Probabilistic timing properties over paths are reduced to computing transient-state probabilities for CTMCs, for which dedicated and efficient methods such as uniformisation can be employed. The use of temporal logic and model checking supports:

- the preservation of the validity of all formulas under *lumping*;
- *automated means* to analyse state-based and path-based measures over CTMCs;
- *automated measure-driven aggregation* of CTMCs;
- *hiding specialised algorithms* from the performance engineer;
- a means to specify both functional and performance properties in a *single framework*.

Organisation of the paper. This paper gives a flavour of the approaches mentioned above. A more detailed treatment of the process algebra part can be found in [28, 31, 29]; the model checking part is described in full detail in [7, 5]. For a broader overview and introduction into formal methods and performance evaluation we refer to [13]. The paper is organised as follows. Sec. 2 presents some introductory material on CTMCs. Sec. 3 surveys stochastic process algebras and indicates the main issues involved, such as synchronisation, abstraction and interleaving. Sec. 4 discusses the temporal logic approach and some of the

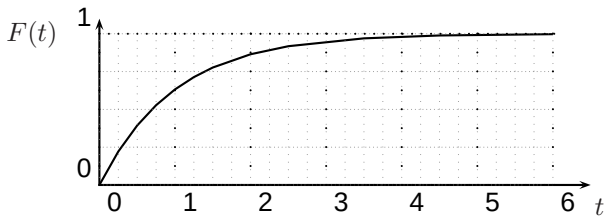
model checking algorithms. Sec. 5 concludes the paper and discusses some future research directions.

2 Continuous-time Markov chains

This section introduces exponential distributions, continuous-time Markov chains and their behaviour. This is done in an informal way, we refer to [29, 27] for details.

2.1 Exponential distributions

A probability distribution (function) is a function that assigns a probability (a real value between 0 and 1) to each element of some given set. This set is usually called the sample space, and is often interpreted as *time*, of either discrete ($\mathbb{N}$) or continuous ($\mathbb{R}_{\geqslant 0}$) nature. A specific continuous probability distribution function $F : \mathbb{R}_{\geqslant 0} \mapsto [0, 1]$ defined by $F(t) = 1 - e^{-\lambda t}$ is depicted below.



Intuitively, $F(t)$ is the probability $\mathsf{Prob}(D \leqslant t)$ that a duration D has finished at time t the latest. This specific distribution is called an exponential distribution with rate $\lambda \in \mathbb{R}_{\geqslant 0}$. Evidently, a rate uniquely characterises an exponential distribution. Note that $F(0) = 0$ (the duration surely does not finish in zero time), and $\lim_{t\to\infty} F(t) = 1$ (the duration eventually finishes).

The class of exponential distributions has some important properties that explain their prominent rôle in contemporary performance evaluation. We try to summarise them here. First, we note that the mean value of an exponential distributed duration is the reciprocal $1/\lambda$ of its rate λ. Secondly, an exponential distribution of rate λ is the most appropriate approximation[1] of a random phenomenon of which only the mean value $(1/\lambda)$ is known. Furthermore, exponential distributions possess the so-called *memory-less property*: If D is exponentially distributed then $\mathsf{Prob}(D \leqslant t+t' \mid D > t) = \mathsf{Prob}(D \leqslant t')$. This means that if we observe that D is not finished at time t, and are interested in $F(t+t')$ *under this condition*, then this is just $F(t)$: The distribution is invariant under the passage of time. In this sense, it does not possess memory. In fact, the exponential distribution is the only continuous probability distribution function that possesses this property. Other relevant properties of exponential distributions for this paper are closure properties of exponentially distributions with respect to maximum

[1] In information theoretic jargon, such an approximation maximises the entropy.

and minimum. If we are waiting for several durations to finish, then we are essentially waiting for the maximum of these durations. Exponential distributions are not closed under maximum; the maximum of several (pairwise stochastic independent) exponentially distributed durations is a phase-type distribution. If, on the other hand, we are only waiting for one out of several competing durations, the situation is different. Awaiting the minimum of several (pairwise stochastic independent) exponentially distributed durations D_i (with rate λ_i, $i \in \{0..n\}$) is itself exponentially distributed. Its rate parameter is the sum of the individual rates $\sum_{i=1}^{n} \lambda_i$. In this case the probability that a specific D_j finishes first in the race is given by $\lambda_j / \sum_{i=1}^{n} \lambda_i$.

2.2 Continuous-time Markov chains

For the purpose of this paper, a continuous time Markov chain can be viewed as a finite state machine, where transitions are labelled with rates of exponential distributions. Intuitively, such state machines evolve as follows. Whenever a state is entered, a race starts between several exponentially distributed durations, given by the (rate labels of) transitions leaving this state. As soon as some duration D_j finishes, the state machine moves to the target state of the transition belonging to this D_j (labelled by λ_j). Clearly, a certain successor state is chosen with probability $\lambda_j / \sum_{i=1}^{n} \lambda_i$, and the time until this happens is exponentially distributed with rate $\sum_{i=1}^{n} \lambda_i$.

The memory-less property carries over from the distributions to the Markov chain[2]: If we know that the chain has been in the current state for some time already (or that it was in a specific state at a specific time in the past), then this knowledge is irrelevant for its future behaviour. The chain's behaviour is history independent, only the identity of the state currently occupied is decisive for the future behaviour.

Usually, a CTMC is characterised by its so-called generator matrix $\mathbf{Q}$ and its initial distribution. The entries of the generator matrix $\mathbf{Q}$ specify the transition rates: $\mathbf{Q}(s, s')$ denotes the rate of moving from state s to state s', where $s \neq s'$.

Definition 1. *(Generator matrix) For some finite set of states S, a square matrix $\mathbf{Q}$ of size $|S| \times |S|$ is the (infinitesimal) generator matrix of a CTMC iff, for all $s \in S$, $\mathbf{Q}(s, s') \geqslant 0$ $(s \neq s')$, and $\mathbf{Q}(s, s) = -\sum_{s' \neq s} \mathbf{Q}(s, s')$.*

While the off-diagonal entries of this matrix specify *individual* rates of *entering* a new state, the diagonal entries specify the converse, a *cumulative* rate of *leaving* the current state, so to speak. Together with an initial state (or an initial probability distribution on states) the generator matrix gives all the necessary information to determine the transient and steady-state probabilistic behaviour of the chain.

[2] The standard definition of CTMCs proceeds in the reverse way, i.e., it defines a memory-less discrete-state continuous-time stochastic process, and derives from this that exponential distributions need to govern its behaviour.

Example 1. In the leftmost column of Fig. 1 we have depicted the generator matrix of a CTMC by means of the usual state-transition representation, where s and s' are connected by a transition labelled λ iff $\mathbf{Q}(s, s') = \lambda > 0$. The initial state is coloured black. In order to illustrate how the probability mass spreads over states as time passes, we represent it as pie-charts of black colour. All black colour is initially (at time 0) in state s_0. From left to right the figure depicts snapshots at different times, where the pie-charts indicate the amount of probability $\pi_{s'}(s_0, t)$ of being in state s' at time t. The rightmost column depicts the limits of these probabilities as $t \to \infty$.

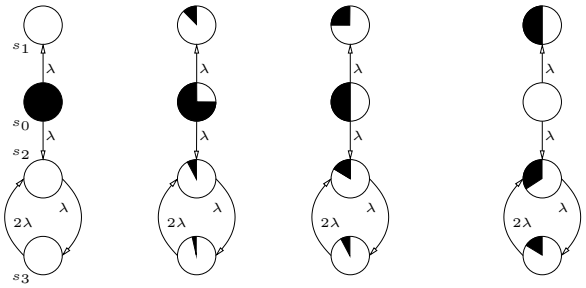


Fig. 1. Transient and steady-state behaviour of a CTMC (from left to right: transient probabilities $\underline{\pi}(s_0, 0)$, $\underline{\pi}(s_0, \ln(4/3)/(2\lambda))$, $\underline{\pi}(s_0, \ln(2)/(2\lambda))$, and steady-state probability $\underline{\pi}(s)$).

The vector $\underline{\pi}(s, t) = (\pi_{s'}(s, t))_{s' \in S}$ is the *transient probability* vector at time t if starting in state s at time 0. The vector $\underline{\pi}(s) = (\lim_{t \to \infty} \pi_{s'}(s, t))_{s' \in S}$ is called the *steady-state probability* vector. Such a limit exists for arbitrary finite CTMCs, and may depend on the starting state. Efficient numerical algorithms exist to compute steady-state as well as transient probability vectors [27].

Lumpability. We conclude this section by a remark on an important concept that allows one to aggregate states of a CTMC without affecting transient and steady-state probabilities. This concept, called *lumpability* is defined as follows [40, 15].

Definition 2. *(Lumpability.) For $\mathcal{S} = \{ S_1, \ldots, S_n \}$ a partitioning of the state space S of a CTMC, the CTMC is* lumpable *with respect to $\mathcal{S}$ if and only if for any partition $S_i \subseteq S$ and states $s, s' \in S_i$:*

$$\forall 0 < k \leqslant n.\ \sum_{s'' \in S_k} \mathbf{Q}(s, s'') = \sum_{s'' \in S_k} \mathbf{Q}(s', s'').$$

That is, for any two states in a given partition the cumulative rate of moving to any other partition needs to be equal. Under this condition, the performance measures of a CTMC and its lumped quotient are strongly related. First, the

quotient stochastic process (defined on a state space $\mathcal{S}$) is a CTMC as well. In addition, the probability of the lumped CTMC being in the quotient state S_i equals the sum of the probability of being in any of the original states $s \in S_i$ in the original chain. This correspondence holds for transient and steady-state probabilities.

3 Process algebra for CTMCs

In this section we discuss various issues one faces when designing a process algebraic formalism for CTMCs. We only summarise the crucial considerations, and refer to [29, 12] for more elaborate discussions.

3.1 CTMC algebra

To begin with we introduce a small, action-less process algebra to generate CTMCs.

Syntax. Let X be drawn from a set of process variables, and I drawn from a set of finite sets of indices. Furthermore let $\lambda_i \in \mathbb{R}_{\geqslant 0}$ for $i \in I$. The syntax of the algebra MC is

$$P ::= \sum_{i \in I} (\lambda_i) \,.\, P \;\Big|\; X \;\Big|\; recX.P$$

The term $recX.P$ defines a recursive process X by P, that possibly contains occurrences of X. If I consists of two elements we use binary choice $+$, if I is empty we write $\mathbf{0}$. The meaning of summation is as follows: For I a singleton set, the term $(\lambda) \,.\, P$ denotes a process that evolves into P within t time units ($t \geqslant 0$) according to an exponential distribution of rate λ. That is, it behaves like P after a certain delay D that is determined by $\mathsf{Prob}(D \leqslant t) = 1 - e^{-\lambda t}$ for positive t.[3] In general, the term $\sum_{i \in I} (\lambda_i) \,.\, P_i$ offers a *timed probabilistic* choice among the processes P_i. As in a CTMC, a race is assumed among competing delays. Intuitively, a successor state P_j is entered with probability $\lambda_j / \sum_{i \in I} \lambda_i$, and the time until this happens is exponentially distributed with rate $\sum_{i \in I} \lambda_i$. We restrict MC to closed expressions given by the above grammar.

Semantics. The structured operational semantics of MC is presented below. The inference rules define a mapping of this algebra onto CTMCs (as we will see).

$$\sum_{i \in I} (\lambda_i) \,.\, P_i \stackrel{\lambda_j}{\longmapsto}_j P_j \quad (j \in I) \qquad\qquad \frac{P\{\, recX.P/X \,\} \stackrel{\lambda}{\longmapsto}_i P'}{recX.P \stackrel{\lambda}{\longmapsto}_i P'}$$

[3] The prefix $(\lambda) \,.\, P$ can be considered as the probabilistic version of the timed prefix $(t) \,.\, P$ that typically occurs in timed process algebras, like in TCCS [44] or in Timed CSP [50].

The rule for recursion is standard; we just recall that $P\{\, Q/X \,\}$ denotes term P in which all (free) occurrences of process variable X in P are replaced by Q. The rule for choice requires some explanation. Consider $\sum_{i\in I} (\lambda_i) \,.\, P_i$. At execution, the fastest process, that is, the process that is enabled first, is selected. This is reflecting the race condition described above. The probability of choosing a particular alternative, P_j say, equals $\lambda_j / \sum_{i\in I} \lambda_i$, assuming that summands with distinct indices are distinct.

The transitions are decorated with an auxiliary label indicated as subscript of the transition relation. It is used to distinguish between different deduction trees of a term. In absence of such mechanism, we would, for instance, for $(\lambda_1) \,.\, P + (\lambda_2) \,.\, P$, obtain two distinct transitions, except if $\lambda_1 = \lambda_2$. In that specific case we would obtain two different deduction trees for the same transition labelled λ_1 (or λ_2); this, however, does suggest that P can be reached with rate λ_1 (or λ_2), whereas this should be rate $\lambda_1 + \lambda_2$. A similar mechanism is rather standard in probabilistic process calculi like PCCS [24].

The above operational semantics maps a term onto a transition system where transitions are labelled by rates. It is not difficult to check that by omitting self-loops and replacing the set of transitions from s to s' by a single transition with the sum of the rates of the transitions from s to s', a CTMC is obtained.

Example 2. The leftmost CTMC in Fig. 1 is generated from the semantics of

$$(\lambda) \,.\, \mathbf{0} + (\lambda) \,.\, recX.(\lambda) \,.\, (2\lambda) \,.\, X$$

Lumping equivalence. Lumping equivalence is defined in the same style as Larsen-Skou's probabilistic bisimulation [41] and Hillston's strong equivalence [36]. Let $\{\!|\, \ldots \,|\!\}$ denote multi-set brackets.

Definition 3. *(Lumping equivalence.) An equivalence relation $\mathcal{S}$ on MC is a* lumping equivalence *iff for any pair $(P, Q) \in \mathsf{MC} \times \mathsf{MC}$ we have that $(P, Q) \in \mathcal{S}$ implies for all equivalence classes $C \in \mathsf{MC}/_{\mathcal{S}}$:*

$$\gamma(P, C) = \gamma(Q, C) \text{ with } \gamma(R, C) = \sum_i \{\!|\, \lambda \mid R \stackrel{\lambda}{\longmapsto}_i R', R' \in C \,|\!\}.$$

Processes P and Q are lumping equivalent*, denoted $P \sim Q$, if $(P, Q) \in \mathcal{S}$ with $\mathcal{S}$ a lumping equivalence.*

Here, we use $\mathsf{MC}/_{\mathcal{S}}$ to denote the set of equivalence classes induced by $\mathcal{S}$ over MC. Stated in words, P and Q are lumping equivalent if the total rate of moving to equivalence class C under $\sim$ is identical for all such classes. As the name suggests, this bisimulation-style definition is in close correspondence to the concept of lumpability on CTMCs (cf. Def. 2). As first pointed out by Buchholz [16] and Hillston [36] (in settings similar to ours) $P \sim Q$ if and only if their underlying CTMCs can be partitioned into isomorphic lumpable partitionings.

Example 3. The term $(\lambda) \,.\, \mathbf{0} + (\lambda) \,.\, recX.(\lambda) \,.\, (2\lambda) \,.\, X$ is equivalent to the chain

$$(\lambda)\,.\,\mathbf{0} + (\lambda)\,.\,recX.\left(\left(\frac{1}{3}\lambda\right)\,.\,(2\lambda)\,.\,X + \left(\frac{2}{3}\lambda\right)\,.\,(2\lambda)\,.\,(\lambda)\,.\,(2\lambda)\,.\,X\right)$$

To illustrate this, both chains are depicted in Fig. 2. Let $\mathcal{S}$ be the equivalence relation containing (exactly) those pairs of states in Fig. 2 that are shaded with identical patterns. It is easy to check that $\mathcal{S}$ is a lumping equivalence, and it equates the initial states. Thus, the latter can be lumped into the former.

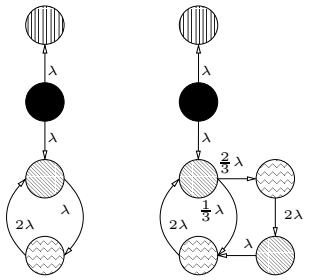


Fig. 2. Lumping equivalence classes.

The fact that lumpability is nowadays known to have a bisimulation-style (coinductive) definition is a prime example for cross-fertilisation from concurrency theory to performance evaluation. In particular, partition refinement algorithms for bisimulation can be adapted to carry out the *best possible lumping* on finite CTMCs [35]. This improves performance evaluation of large CTMCs, where the question of how to determine a lumpable partitioning of the state space (let alone the best possible one) was for a long time based on modeller's ingenuity and experience.

Equational theory. Since it can be shown that lumpability is a congruence with respect to the operators of MC, we may strive for a sound and complete axiomatisation of $\sim$ for MC. Such an axiomatisation facilitates the lumping of CTMCs at a syntactic level. The axioms for sequential finite terms are listed as (B1) through (B4) below.

$$\text{(B1) } P + Q = Q + P \qquad \text{(B2) } (P + Q) + R = P + (Q + R) \qquad \text{(B3) } P + \mathbf{0} = P$$
$$\text{(B4)} \qquad (\lambda + \mu)\,.\,P = (\lambda)\,.\,P + (\mu)\,.\,P$$

The axioms (B1) through (B3) are well known from classical process calculi. Axiom (B4) is a distinguishing law for our calculus and can be regarded as a replacement in the Markovian setting of the traditional idempotency axiom for choice ($P + P = P$). Axiom (B4) reflects that the resolution of choice is modelled by the minimum of (statistically independent) exponential distributions.

Together with standard laws for handling recursion on classical process calculi these axioms can be shown to form a sound and complete axiomatisation of MC.

Interleaving. To illustrate another feature of CTMCs from the concurrency theoretical perspective, we add a simple parallel composition operator to our calculus, denoted by $||$. Intuitively, the term $P \,||\, Q$ can evolve while either P evolves or Q evolves independently from (and concurrently to) each other. Due to the memory-less property of CTMCs, the behaviour of parallel CTMCs can be interleaved. This is different from a deterministic time setting where parallel processes typically are forced to synchronise on the advance of time, as in TCCS [44]. The operational rules are:

$$\frac{P \overset{\lambda}{\longmapsto}_i P'}{\begin{array}{c} P \,||\, Q \overset{\lambda}{\longmapsto}_{(i,*)} P' \,||\, Q \\ Q \,||\, P \overset{\lambda}{\longmapsto}_{(*,i)} Q \,||\, P'. \end{array}}$$

(Notice that we create new auxiliary labels of the form $(i, *)$ and $(*, i)$ in order to obtain a multi-transition relation.) To understand the meaning of the memory-less property in our context consider the process $(\lambda) \,.\, P \,||\, (\mu) \,.\, Q$ and suppose that the delay of the left process finishes first (with rate λ). Due to the memory-less property, the remaining delay of Q is determined by an exponential distribution with (still!) rate μ, exactly the delay prior to the enabling of these delays before the delay of the first process has been finished. Therefore, the parameters of transitions do not need any adjustment in an interleaving semantics. One of the consequences of this independent delaying is that an expansion law is obtained rather straightforwardly. For $P = \sum_i (\lambda_i) \,.\, P_i$ and $Q = \sum_j (\mu_j) \,.\, Q_j$ we have:

$$P \,||\, Q = \sum_i (\lambda_i) \,.\, (P_i \,||\, Q) + \sum_j (\mu_j) \,.\, (P \,||\, Q_j).$$

3.2 Interaction in CTMCs

The algebra MC is lacking any notion of action, and hence provides only a restricted way of specifying CTMCs in a compositional way. For instance, interaction between different parallel chains cannot be adequately described. Two different approaches can be identified when it comes to the integration of actions into CTMC algebra. One way [25, 36, 10] is to combine delays and actions in a single *compound* prefix $(a, \lambda) \,.\, P$. The intuitive meaning of this expression is that the process $(a, \lambda) \,.\, P$ is ready to engage in action a after a delay determined by an exponential distribution with rate λ and afterwards behaves like P. We refer to [12] for a discussion of this approach. Here we focus on a different, orthogonal approach [28, 11] where the action prefix $a \,.\, P$ known from standard process algebra is added to CTMC algebra, to complement the existing delay prefix $(\lambda) \,.\, P$ with separate means to specify actions.

Syntax. We equip this new algebra, called IMC (Interactive Markov Chains) with a TCSP-style parallel composition operator parametrised with a set A of synchronising actions. Accordingly we have:

$$P ::= \sum_{i \in I} a_i \,.\, P + \sum_{i \in I'} (\lambda_i) \,.\, P \;\Big|\; X \;\Big|\; recX.P \;\Big|\; P \,||_A\, P$$

Semantics. The semantics is given by the rules listed for MC (where $||$ is now understood as $||_A$ for an arbitrary set A of actions) plus the standard rules known from process algebra: (In mixed summations like $a \,.\, P + (\lambda) \,.\, Q$ the respective summation rules are applied elementwise.)

$$\sum_{i \in I} a_i \,.\, P_i \xrightarrow{a_j} P_j \quad (j \in I) \qquad\qquad \frac{P\{\, recX.P/X \,\} \xrightarrow{a} P'}{recX.P \xrightarrow{a} P'}$$

$$\frac{P \xrightarrow{a} P'}{\begin{array}{l} P \,||_A\, Q \xrightarrow{a} P' \,||_A\, Q \\ Q \,||_A\, P \xrightarrow{a} Q \,||_A\, P' \end{array}} \quad (a \notin A) \qquad\qquad \frac{P \xrightarrow{a} P', Q \xrightarrow{a} Q'}{P \,||_A\, Q \xrightarrow{a} P' \,||_A\, Q'} \quad (a \in A)$$

Equational theory. Since the calculus extends both standard process algebra and CTMC algebra, it is possible to integrate bisimulation and lumping equivalence. This can be done for strong, weak and other bisimulation equivalences [28]; we omit the operational definitions here for brevity. Instead we characterise the resulting equational theory for weak (lumping) bisimulation congruence by the set of axioms satisfied by finite expressions. It is given by the axioms (B1) through (B4) listed for CTMC algebra, and the following additional laws.

$$\text{(B5)}\ a \,.\, P = a \,.\, P + a \,.\, P \qquad \text{(P1)}\ (\lambda) \,.\, P + \tau \,.\, Q = \tau \,.\, Q$$

$$\begin{array}{lll} (\tau 1)\ \tau \,.\, P = P + \tau \,.\, P & (\tau 2)\ a \,.\, (P + \tau \,.\, Q) = a \,.\, (P + \tau \,.\, Q) + a \,.\, Q \\ (\tau 3)\ a \,.\, P = a \,.\, \tau \,.\, P & (\tau 4) \qquad (\lambda) \,.\, P = (\lambda) \,.\, \tau \,.\, P \end{array}$$

Axiom (B5) replaces the traditional idempotence axiom for choice ($P + P = P$) which is not sound for delay prefixed expressions (cf. axiom (B4)). The (P1) axiom realises *maximal progress*: A process that has something internal to do will do so, without letting time pass. No time will be spent in the presence of an internal action alternative. The axioms ($\tau 1$) through ($\tau 3$) are well known for weak bisimulation on standard process algebra [43], and ($\tau 4$) extends ($\tau 3$) to delay prefixed expressions. Together with additional laws to handle recursion (and divergence [34]), this set gives rise to a sound and complete axiomatisation for finite state IMC, illustrating that process algebra smoothly extends to this orthogonal union of CTMC algebra and process algebra. We refer to [28] for further discussion.

3.3 Time constraints and phase-type distributions

In this section, we illustrate two important features of IMC. We show how more general continuous probability distributions can be embedded into the calculus, and we illustrate how such general distributions can be used to constrain the behaviour of an IMC in a modular, constraint-oriented style. The approach presented here is detailed out in [31].

Phase-type distributions. Phase-type distributions can be considered as matrix generalisations of exponential distributions, and include frequently used distributions such as Erlang, Cox, hyper- and hypo-exponential distributions. Intuitively, a phase-type distribution can be considered as a CTMC with a single absorbing state (a state with $\mathbf{Q}(s, s') = 0$ for all s). The time until absorption determines the phase-type distribution [45]. In terms of CTMC algebra, phase-type distributions can be encoded by explicitly specifying the structure of the CTMC using summation, recursion, and termination ($\mathbf{0}$), as in the MC term $\tilde{Q}$ given by $(\lambda) \,.\, recX.(\mu) \,.\, (\mu) \,.\, X + (\lambda) \,.\, \mathbf{0}$. The possibility of specifying phase-type distributions is of significant interest, since phase-type distributions can approximate arbitrary distributions arbitrarily close [45] (i.e., it is a dense subset of the set of continuous distributions). In other words, MC and IMC can be used to express arbitrary distributions, by choosing the appropriate absorbing Markov chain, and (mechanically) encoding it in MC.

Time constraints. In IMC, phase-type distributions can govern the timing of actions. The main idea is to intersperse action sequences (such as $a \,.\, b \,.\, \mathbf{0}$) with specific phase-type distributions (such as the above $\tilde{Q}$) in order to delay the occurrences of the actions in the sequence appropriately, such as delaying the occurrence of b after a by $\tilde{Q}$. This can be achieved by explicitly changing the structure of the process into $a \,.\, (\lambda) \,.\, recX.(\mu) \,.\, (\mu) \,.\, X + (\lambda) \,.\, b \,.\, \mathbf{0}$, but this approach will be cumbersome in general.

To enhance specification convenience, we introduce the *elapse* operator that is used to impose phase-type distributed time constraints on specific occurrences of actions. The elapse operator facilitates the description of such time constraints in a *modular* way, that is, as separated processes that are constraining the behaviour by running in parallel with an untimed (or otherwise time-constrained) process. To introduce this operator, we use much of the power of process algebra, since the semantics of the operator is defined by means of a translation into the basic operators of IMC. Due to the compositional properties of IMC, important properties (congruence results, for instance) carry over to this operator in a straightforward manner.

We shall refer to a time constraint as a delay that necessarily has to elapse between two kinds of actions, unless some action of a third kind occurs in the meanwhile. In order to facilitate the definition of such time constraints, the elapse operator is an operator with four parameters, syntactically denoted by [**on** S **delay** D **by** Q **unless** B]:

– a phase-type distribution Q (represented as an MC term) that determines the duration of the time constraint,
– a set of actions S (start) that determines when the delay (governed by Q) starts,
– a set of actions D (delay) which have to be delayed, and
– a set of actions B (break) which may interrupt the delay.

Thus, for instance, [**on** $\{a\}$ **delay** $\{b\}$ **by** $\tilde{Q}$ **unless** $\varnothing$] imposes the delay of $\tilde{Q}$ between a and b. We claim that a wide range of practical timing scenarios can be covered by this operator (in particular if non-empty intersections between the action sets are allowed). This is illustrated in [31] where this operator is used to impose various time constraints on an existing, untimed process algebraic specification (of more than 1500 lines of LOTOS code) of the plain ordinary telephone system.

Semantically, the intuition behind this operator is that it enriches the chain Q with some synchronisation potential, that is used to initialise and reset the time constraint in an appropriate way. The time constraint is imposed on a process P by means of parallel composition, such as in

$$P \,||_{S \cup D \cup B}\, [\textbf{on } S \textbf{ delay } D \textbf{ by } Q \textbf{ unless } B].$$

The elapse operator is an auxiliary operator that can be defined using sequential composition and disrupt, LOTOS-operators that can be easily added to IMC [31]. For instance, the semantics of $a \,.\, b \,.\, \mathbf{0} \,||_{\{a,b\}}\, [\textbf{on } \{a\} \textbf{ delay } \{b\} \textbf{ by } \tilde{Q} \textbf{ unless } \varnothing]$ agrees with $a \,.\, (\lambda) \,.\, recX.(\mu) \,.\, (\mu) \,.\, X + (\lambda) \,.\, b \,.\, \mathbf{0}$ up to weak bisimulation.

3.4 Compositional aggregation

Interactive Markov chains can be used to specify CTMCs, but due to the presence of nondeterminism (inherited from standard process algebra), the model underlying IMC is richer, it is the class of continuous time Markov decision chains [49], a strict superset of CTMCs. Nondeterminism is one of the vital ingredients of process algebra and hence of IMC, though it appears as an additional hurdle when it comes to performance evaluation, because the stochastic behaviour may be underspecified. In order to eliminate nondeterminism – and to aggregate the state space – we have developed a general recipe leading from an IMC specification to a CTMC:

1. Develop a specification of the system under investigation using the operators provided by IMC. A possible approach is to start from an existing process algebraic specification and to enrich the specification by incorporating time constraints. The elapse operator is convenient for this purpose.
2. Close the specification by abstracting from all actions using the standard abstraction (encapsulation) operator of process algebra.
3. Apply weak bisimulation congruence to aggregate (lump) the state space, to eliminate action transitions, and to remove nondeterminism. Due to the

congruence property, this aggregation step is preferably done compositionally, by applying it to components of the specification prior to composition. In this way, the state space explosion problem can be diminished [31].

If the aggregated, minimal transition system does not contain action transitions, it trivially corresponds to a lumped CTMC. If, on the other hand, the resulting transition system still contains action transitions the stochastic process is under-specified, it is a continuous time Markov decision chain, because nondeterminism is present. The above recipe has been exercised in practice successfully [31]. The necessary algorithms for state space generation, and efficient aggregation are implemented [30, 17], and compositional aggregation is supported.

4 Model Checking CTMCs

Once a CTMC has been generated, the next step is to evaluate the measure(s) of interest such as time to failure, system throughput or utilisation, with the required accuracy. In this section, we use temporal logic to express constraints (i.e., bounds) on such measures and show how model checking techniques can be employed for the automated analysis of these constraints. We only summarise the crucial considerations, and refer to [7, 5] for more elaborate discussions.

4.1 CTMC temporal logic

To specify performance and dependability measures as logical formulas over CTMCs, we assume the existence of a set AP of atomic propositions with $a \in AP$ and 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. These labelled CTMCs can be viewed as Kripke structures with transitions labelled by rates.

Syntax. CSL (Continuous Stochastic Logic) is a branching-time temporal logic à la CTL [23] based on [4] that is interpreted on CTMCs. Let p be a probability ($p \in [0, 1]$) and $\trianglelefteq$ a comparison operator, i.e., $\trianglelefteq \in \{ \leqslant, \geqslant \}$. CSL state-formulas are constructed according to the following syntax:

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

The two probabilistic operators $\mathcal{S}$ and $\mathcal{P}$ refer to the steady-state and transient behaviour, respectively, of the CTMC being studied. Whereas the steady-state operator $\mathcal{S}$ refers to the probability of residing in a particular set of *states* (specified by a state-formula) on the long run, the transient operator $\mathcal{P}$ allows us to refer to the probability of the occurrence of particular *paths* in the CTMC, similar to [26]. The operator $\mathcal{P}_{\trianglelefteq p}(.)$ replaces the usual CTL path quantifiers $\exists$ and $\forall$. In fact, for most cases (up to fairness) $\exists\varphi$ can be written as $\mathcal{P}_{>0}(\varphi)$ and $\forall\varphi$ as $\mathcal{P}_{\geqslant 1}(\varphi)$. For instance, $\mathcal{P}_{>0}(\Diamond a)$ is equivalent to $\exists\Diamond a$ and $\mathcal{P}_{\geqslant 1}(\Diamond a)$ stands for $\forall\Diamond a$ given a fair interpretation of the CTL-formula $\forall\Diamond a$.

For I an interval on the real line ($I \subseteq \mathbb{R}_{\geqslant 0}$), the syntax of CSL path-formulas is

$$\varphi ::= \mathcal{X}^I\ \Phi \ \Big|\ \Phi\, \mathcal{U}^I\ \Phi.$$

The operators $\mathcal{X}^I$ and $\mathcal{U}^I$ are the timed variants of the usual next-operator and until-operator, respectively. Similar timed variants of these operators appear in timed CTL [2].

Semantics. State-formulas are interpreted over the states of a CTMC; for s a state of the CTMC under consideration and Φ a state-formula, $s \models \Phi$, if and only if Φ is valid in s. The semantics of the Boolean operators is standard (i.e., $s \models a$ iff $s \in L(s)$, $s \models \neg\Phi$ iff $s \not\models \Phi$, and $s \models \Phi_1 \vee \Phi_2$ iff $s \models \Phi_1 \vee s \models \Phi_2$.) The state-formula $\mathcal{S}_{\trianglelefteq p}(\Phi)$ asserts that the steady-state probability for the set of Φ-states meets the bound $\trianglelefteq p$:

$$s \models \mathcal{S}_{\trianglelefteq p}(\Phi) \quad \text{if and only if} \quad \sum_{s' \models \Phi} \pi_{s'}(s) \trianglelefteq p$$

where we recall that $\pi_{s'}(s)$ equals $\lim_{t\to\infty} \pi_{s'}(s,t)$, where $\pi_{s'}(s,t)$ denotes the probability to be in state s' at time t when starting in state s. Finally, $\mathcal{P}_{\trianglelefteq p}(\varphi)$ asserts that the probability measure of the paths satisfying φ meets the bound $\trianglelefteq p$. Let $Prob(s,\varphi)$ denote the probability of all paths satisfying φ when the system starts in state s. (The probability measure $Prob$ is formally defined in [7].) Then:

$$s \models \mathcal{P}_{\trianglelefteq p}(\varphi) \quad \text{if and only if} \quad Prob(s,\varphi) \trianglelefteq p$$

A path σ in a CTMC is an alternating sequence of the form $s_0\, t_0\, s_1\, t_1\, \ldots$ where $t_i \in \mathbb{R}_{\geqslant 0}$ indicates the amount of time stayed in state s_i.[4] Let $\sigma[i]$ denote the $(i{+}1)$-state in σ and let $\sigma@t$ denote the state occupied by σ at time t. The satisfaction relation for the path-formulas is defined as follows. The path-formula $\mathcal{X}^I\,\Phi$ asserts that a transition is made to a Φ-state at some time point $t \in I$:

$$\sigma \models \mathcal{X}^I\ \Phi \quad \text{if and only if} \quad \sigma[1] \models \Phi\ \wedge\ \delta(\sigma,0) \in I$$

where $\delta(\sigma,0) = t_0$, the duration of staying in the initial state s_0 of σ. The path-formula $\Phi\,\mathcal{U}^I\,\Psi$ asserts that Ψ is satisfied at some time instant in the interval I and that at all preceding time instants Φ holds:

$$\sigma \models \Phi_1\,\mathcal{U}^I\,\Phi_2 \quad \text{if and only if} \quad \exists t \in I.\ (\sigma@t \models \Phi_2\ \wedge\ \forall u \in [0,t).\, \sigma@u \models \Phi_1)\,.$$

The usual (untimed) next- and until-operator are obtained as $\mathcal{X}\,\Phi = \mathcal{X}^{[0,\infty)}\,\Phi$, and $\Phi\,\mathcal{U}\,\Psi = \Phi\,\mathcal{U}^{[0,\infty)}\,\Psi$. Other Boolean connectives are derived in the usual way (e.g. $\Phi \vee \Psi = \neg(\neg\Phi \wedge \neg\Psi)$). Temporal operators like $\Diamond^I$ ("eventually in I") and $\Box^I$ ("always in I") are derived by, for example: $\mathcal{P}_{\trianglelefteq p}(\Diamond^I\,\Phi) = \mathcal{P}_{\trianglelefteq p}(\mathrm{tt}\,\mathcal{U}^I\,\Phi)$ and $\mathcal{P}_{\geqslant p}(\Box^I\,\Phi) = \mathcal{P}_{\leqslant 1-p}(\Diamond^I\,\neg\Phi)$.

[4] For simplicity, we assume all paths to be infinite.

Expressiveness. Besides standard state-based performance measures such as steady-state and transient-state probabilities, the logic-based approach allows one to specify bounds on the occurrence probability of certain (sets of) paths. We exemplify the type of properties that one can express using CSL by considering a simple re-configurable fault tolerant system. The system can be either *Up* or *Down*, and it may (or may not) be in a phase of (initial or re-)configuration (*Config*). Thus we consider $AP = \{ \mathit{Up}, \mathit{Down}, \mathit{Config} \}$.

Example 4. Assume that the states of the CTMC in Fig. 1 are labelled – from top to bottom – by $\{ \mathit{Down} \}$, $\{ \mathit{Up}, \mathit{Config} \}$, $\{ \mathit{Up} \}$, and $\{ \mathit{Down}, \mathit{Config} \}$. For instance, $L(s_0) = \{ \mathit{Up}, \mathit{Config} \}$.

As an overview of some well-known performance and dependability measures [51] and their formulation in terms of CSL we list the following CSL formulas:

(a)	steady-state availability	$\mathcal{S}_{\trianglelefteq p}(\mathit{Up})$
(b)	transient configuration probability at time t	$\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\mathit{Config})$
(c)	instantaneous availability at time t	$\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\mathit{Up})$
(d)	distribution of time to failure	$\mathcal{P}_{\trianglelefteq p}(\mathit{Up}\,\mathcal{U}^{[0,t]}\,\mathit{Down})$

Measure (a) expresses a bound on the steady-state availability of the system and (b) expresses a bound on the transient-state probability of (re-)configuring the system at time t. Measure (c) states (a bound on) the probability to be in a non-failed state at time t, i.e., the instantaneous availability at time t and (d) expresses, indirectly, the time until a failure, starting from a non-failed state. That is, evaluating this measure for varying t, gives the distribution of the time to failure.

The above standard transient measures are expressed using only simple instances of the $\mathcal{P}$-operator. However, since this operator allows an arbitrary path-formula as argument, much more general measures can be described and nested.

Example 5. An example of an interesting non-standard measure is the probability of reaching a certain set of states provided that all paths to these states obey certain properties. For instance,

$$\neg \mathit{Config} \Rightarrow \mathcal{P}_{\geqslant 0.99}(\mathit{Up}\ \mathcal{U}^{[0,20]}\ \mathit{Config})$$

states that the probability to turn from a non-configuring state into a reconfiguration in no more than 20 time units without any system down time on the way is more than 99%. As another example, we may require that in the steady-state, there is a chance of at most 10% that a down time is likely (that is, has more than half of the probability) to occur within 5 and 10 time units from now.

$$\mathcal{S}_{\leqslant 0.1}(\mathcal{P}_{>0.5}(\Diamond^{[5,10]}\mathit{Down}))$$

Lumpability revisited. In the same spirit as the relations between bisimulation and CTL (and CTL*) equivalence [14] and between Larsen-Skou's probabilistic bisimulation and PCTL-equivalence [3], there exists a relation between lumping equivalence and CSL-equivalence. This is illustrated by means of a slight variant of the earlier treated notion of lumping equivalence, cf. Def. 2.

Definition 4. *(F-Lumpability.) For* $\mathcal{S} = \{ S_1, \ldots, S_n \}$ *a partitioning of the state space* S *of a CTMC and* F *a set of CSL state-formulas, the CTMC is* F-lumpable *with respect to* $\mathcal{S}$ *if and only if for any partition* $S_i \subseteq S$ *and states* $s, s' \in S_i$*:*

$$\forall 0 < k \leqslant n.\ \sum_{s'' \in S_k} \mathbf{Q}(s, s'') = \sum_{s'' \in S_k} \mathbf{Q}(s', s'')$$

and

$$\{ \Phi \mid s \models \Phi \} \cap F = \{ \Phi \mid s' \models \Phi \} \cap F.$$

That is, for any two states in a given partition the cumulative rate of evolving to another partition (like before) must be equal, and the set of formulas in F that are fulfilled must coincide. Clearly, if a CTMC is F-lumpable with respect to some partitioning of its state space, then it is also lumpable. A CTMC and its lumped quotient are strongly related with respect to the validity of CSL formulae. In particular, a (state in a) CTMC and its (quotient state in the) AP-lumped quotient – obtained by the above notion where the set F equals the set of atomic propositions – satisfy the same CSL-formulas [5]. This result can be exploited by aggregating the CTMC as far as possible during checking the validity of CSL-formulas, or prior to this process by considering its quotient with respect to the coarsest AP-lumping.

4.2 CTMC model checking

There are two distinguishing benefits when using CSL for specifying constraints on measures-of-interest over CTMCs: (i) the specification is entirely formal such that the interpretation is unambiguous, and (ii) it allows the possibility to state performance and dependability requirements over a selective set of paths (similar to [47]) through a model. These features are paired with the (practically most relevant) possibility to check CSL-formulas in a completely automated manner. This can be done by combining model checking techniques with numerical solution techniques. The basic procedure is as for model checking CTL: in order to check whether state s satisfies the formula Φ, we recursively compute the sets $Sat(\Psi) = \{s' \in S \mid s' \models \Psi\}$ of all states that satisfy Ψ, for the subformulas Ψ of Φ, and eventually check whether $s \in Sat(\Phi)$. For atomic propositions and Boolean connectives this procedure is exactly the same as for CTL. Next and (unbounded) until-formulas can be treated in a similar way as in the discrete-time probabilistic setting [26]. Checking steady-state properties reduces to solving a system of linear equations combined with standard graph analysis methods [7].

Fixed-point characterisation. Most interesting (and complicated) though is the handling of time-bounded until-formulas, as their treatment require to deal with the interplay of timing and probabilities. For the sake of simplicity, we treat the case $I = [0, t]$; the general case is a bit more involved, but can be treated in a similar way [5]. Let $\varphi_t = \Phi\, \mathcal{U}^{[0,t]}\, \Psi$. We have from the semantics that

$$s \in Sat(\mathcal{P}_{\trianglelefteq p}(\varphi_t)) \quad \text{if and only if} \quad Prob(s, \varphi_t) \trianglelefteq p$$

The probability $Prob(s, \varphi_t)$ is the least solution of the following set of equations:

$$Prob(s,\varphi_t) = \begin{cases} 1 & \text{if } s \in Sat(\Psi) \\ 0 & \text{if } s \notin Sat(\Phi) \cup Sat(\Psi) \\ \displaystyle\int_0^t \sum_{s' \in S} \mathbf{T}(s,s',x) \cdot Prob(s', \varphi_{t-x})\ dx & \text{otherwise} \end{cases}$$

where $\mathbf{T}(s, s', x)$ denotes the density of moving from state s to state s' in x time-units and can be derived from the matrix $\mathbf{Q}$. 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 $t-x$.

This recursive integral characterisation provides the theoretical basis for model checking time-bounded until-formulas over CTMCs in the same way as the fixed-point characterisations for CTL provide the basis for the model checking algorithms for usual until-formulas [18].

Algorithmic procedure. To illustrate how performance evaluation recipes can be exploited for model checking purposes, we now sketch an efficient and numerically stable strategy for model checking the time-bounded until-formulas [5]. As lumping preserves the validity of all CSL-formulas, a first step is to switch from the original state space to the (possibly much smaller) quotient space under lumping. Next, prior to computing the exact set of states that satisfy φ_t, the states fulfilling the (fair) CTL-formula $\exists(\Phi \,\mathcal{U}\, \Psi)$ is determined. For states not in this set, the respective probabilistic until-formula will have probability 0. In a similar way, the set of states satisfying $\forall(\Phi \,\mathcal{U}\, \Psi)$ (up to fairness, cf. [8]) is computed; these states satisfy φ_t with probability 1. As a result, the actual computation of the system of Volterra integral equations needs to be done only for the remaining states. How to do this? The basic idea is to employ a transformation of the CTMC and the formula at hand, such that a transient analysis problem is obtained for which well-known and efficient computation techniques do exist. This idea is based on the observation that formulas of the form $\mathcal{P}_{\trianglelefteq p}(\Diamond^{[t,t]}\Phi)$ characterise transient probability measures, and their validity (in some state s) can be decided on the basis of the transient probability vector $\underline{\pi}(s,t)$. This vector can be calculated by transient analysis techniques. For $\mathcal{P}_{\trianglelefteq p}(\Phi\ \mathcal{U}^{[0,t]}\, \Psi)$ the CTMC $\mathcal{M}$ under consideration is transformed into another CTMC $\mathcal{M}'$ such that checking $\varphi_t = \Phi\, \mathcal{U}^{[0,t]}\, \Psi$ on $\mathcal{M}$ amounts to checking $\varphi'_t = \Diamond^{[t,t]}\Psi$ 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 $\mathcal{M}'$? Concerning a $(\Phi \wedge \neg\, \Psi)$-state, two simple observations form the basis for this transformation:

- 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 afterwards. This justifies making all Ψ-states absorbing.

– once a state has been reached that neither satisfies Φ nor Ψ, φ is violated regardless of which states will be visited afterwards. This justifies making all $\neg(\Phi \wedge \Psi)$-states 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 $\mathcal{M}$):

$$Prob^{\mathcal{M}}(s, \Phi\, \mathcal{U}^{[0,t]}\, \Psi) = Prob^{\mathcal{M}'}(s, \Diamond^{[t,t]}\Psi) = \sum_{s' \models \Psi} \pi_{s'}(s,t).$$

In fact, by similar observations it turns out that also verifying the general $\mathcal{U}^I$-operator can be reduced to instances of (a two-phase) transient analysis [5].

Example 6. In order to check one of the above mentioned requirements on the CTMC of Fig. 1, one needs to check $s_2 \models \mathcal{P}_{\geqslant 0.99}(\mathit{Up}\ \mathcal{U}^{[0,20]}\ \mathit{Config})$. To decide this, it is sufficient to compute $\underline{\pi}(s_2, 20)$ on a CTMC where state s_0 and s_3 (as well as s_1) are made absorbing, and to check $\pi_{s_0}(s_2, 20) + \pi_{s_3}(s_2, 20) \geqslant 0.99$.

The transformation of the model checking problem for the time-bounded until-operator into the transient analysis of a CTMC has several advantages: (i) it avoids awkward numerical integration, (ii) it allows us to use efficient and numerically stable transient analysis techniques, such as uniformisation [38], and (iii) it employs a measure-driven transformation (aggregation) of the CTMC. The fact that a dedicated and well-studied technique in performance evaluation such as uniformisation can be employed for model checking is a prime example for the cross-fertilisation from performance evaluation to concurrency theory.

Efficiency. The worst-case time complexity of model checking CSL is

$$\mathcal{O}(|\Phi| \cdot (M \cdot q \cdot t_{max} + N^{2.81}))$$

where M is the number of non-zero entries in $\mathbf{Q}$, q is the maximal diagonal entry of $\mathbf{Q}$, t_{max} is the maximum time bound of the time-bounded until sub-formulas occurring in Φ, and N is the number of states. If we make the practically often justified assumption that $M < kN$ for a constant k then the space complexity is linear in N using a sparse matrix data structure. The space complexity is polynomial in the size of the CTMC. The model checking algorithms have been implemented both using sparse matrix data structures [32] and using BDD-based data structures [39].

5 Research perspectives

This paper has presented how two important branches of formal methods for reactive systems – process algebra and model checking – can be exploited for performance and dependability modelling and analysis. The stochastic process

algebra approach is a prominent example of cross-fertilisation of formal specification techniques and performance modelling techniques, whereas the quantitative model checking approach is a promising combination of computer-aided verification technology and performance analysis techniques. We believe that the developments in these areas mark the beginning of a new paradigm for the modelling and analysis of systems in which qualitative and quantitative aspects are studied from an integrated perspective. We hope that the further work towards the realisation of this goal will be a growing source of inspiration and progress for both communities. Examples of issues for future work in this direction are:

- *Specification:* in order to bridge the gap towards (performance) engineers, and to obtain a better integration into the design cycle, efforts should be made to the usage of (appropriate extensions of) specification languages such as UML and SDL for the description of performance models. Similarly, the usage of temporal logic by performance engineers needs to be simplified, for instance, using dedicated specification patterns [22].
- *Verification:* similar to the development of model checking techniques, smart improvements of both algorithms and data structures are needed to make the verification approach more successful. Initial investigations show that symbolic data structures (such as multi-terminal BDDs) and tailored variants of existing techniques ("backwards" uniformisation) yield a substantial efficiency improvement [39]. Promising alternative techniques, such as Kronecker representations [21], and/or refinement techniques for probabilistic systems as recently proposed in [20] could be beneficial in this context as well.
- *Extensions:* several extensions of the process algebra and model checking techniques are worthwhile to investigate. For instance, Markov reward models – an important extension of CTMCs with costs – are not yet satisfactorily treated in a process algebraic or logic-based setting, although some initial attempts have been made [6, 9]. In addition, the application of model checking to non-memoryless models, such as (generalised) semi-Markov processes, remains an interesting topic for further research. Initial work in this direction is reported in [37].

Finally, we highlight two gaps between the process algebraic and model checking approach we discussed: (i) whereas the formal model specifications are behaviour-oriented (i.e., action based), the temporal logic approach is state-based, and (ii) the semantic model of the process algebra may contain nondeterminism, whereas the verification is based on a fully probabilistic model. The first problem can be handled by considering an action-based variant of CSL. Although it turns out that a transformation of this logic into CSL (à la the relationship between CTL and its action-based variant [46]) is possible, it is more beneficial to use direct model checking techniques – basically a tailored version of the CSL model checking algorithms. Details are in [33]. This yields a combination with stochastic process algebras that treat actions and stochastic delays as a single compound entity [10, 25, 36]. In order to close the gap w.r.t. our process algebra IMC that strictly distinguishes between action occurrences

and time delays, we are currently investigating model checking procedures for continuous-time Markov decision chains.

Acknowledgements. Ed Brinksma (Univ. Twente) and Ulrich Herzog (Univ. Erlangen) have contributed to the work on stochastic process algebra reported here. The model checking research reported in this paper has been developed in collaboration with Christel Baier (Univ. Bonn) and Boudewijn Haverkort (RWTH Aachen). The first author is supported by the Netherlands Organisation of Scientific Research (NWO).

References

1. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modeling with Generalized Stochastic Petri Nets*. John Wiley & Sons, 1995.
2. R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104(1):2–34, 1993.
3. A. Aziz, V. Singhal, F. Balarin, R. Brayton and A. Sangiovanni-Vincentelli. It usually works: the temporal logic of stochastic systems. In *CAV'95*, LNCS 939:155–165. Springer, 1995.
4. 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.
5. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model checking continuous time Markov chains by transient analysis. In *CAV 2000*, LNCS 1855:358–372. Springer, 2000.
6. C. Baier, B.R. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *ICALP 2000*, LNCS 1853:780–792. Springer, 2000.
7. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR'99*, LNCS: 1664:146–162. Springer, 1999.
8. C. Baier and M. Kwiatkowska. Model checking for a probabilistic branching time logic with fairness. *Distributed Computing*, 11:125–155, 1998.
9. M. Bernardo. An algebra-based method to associate rewards with EMPA terms. In *ICALP'97*, LNCS 1256:358–368. Springer, 1997.
10. M. Bernardo and R. Gorrieri. A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Theoretical Computer Science*, 202:1–54, 1998.
11. H.C. Bohnenkamp and B.R. Haverkort. Semi-numerical solution of stochastic process algebra models. In *ARTS'99*, LNCS 1601:228–243. Springer, 1999.
12. E. Brinksma and H. Hermanns. Process algebra and Markov chains. In [13].
13. E. Brinksma, H. Hermanns, and J.-P. Katoen, editors. *Lectures on Formal Methods and Performance Analysis*, LNCS 2090. Springer, 2001.
14. M. Brown, E. Clarke, O. Grumberg. Characterizing finite Kripke structures in propositional temporal logic. *Theoretical Computer Science*, 59: 115–131, 1988.
15. P. Buchholz. Exact and ordinary lumpability in finite Markov chains. *Journal of Applied Probability*, 31–75:59–75, 1994.
16. P. Buchholz. Markovian Process Algebra: composition and equivalence. In U. Herzog and M. Rettelbach, editors, *Proc. of PAPM'94*, Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, 1994.

17. M. Cherif, H. Garavel, and H. Hermanns. bcg_min – Minimization of normal, probabilistic, or stochastic labeled transitions systems encoded in the BCG format. `http://www.inrialpes.fr/vasy/cadp/man/bcg_min.html`.
18. 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.
19. A.E. Conway and N.D. Georganas. *Queueing Networks: Exact Computational Algorithms*. MIT Press, 1989.
20. P.R. D'Argenio, B. Jeannet, H.E. Jensen, and K.G. Larsen. Reachability analysis of probabilistic systems by successive refinements. In *PAPM/PROBMIV'01*, LNCS. Springer, 2001. To appear.
21. M. Davio. Kronecker Products and Shuffle Algebra. *IEEE Transactions on Computers*, C-30(2):116–125, 1981.
22. M.B. Dwyer, G.S. Avrunin, and J.C. Corbett. Property specification patterns for finite-state verification. In *Formal Methods in Software Practice*. ACM Press, 1998.
23. E.A Emerson and E.M. Clarke. Using branching time temporal logic to synthesize synchronization skeletons. *Science of Computer Programming*, 2: 241–266, 1982.
24. R.J. van Glabbeek, S.A. Smolka, and B. Steffen. Reactive, generative, and stratified models of probabilistic processes. *Information and Computation*, 121:59–80, 1995.
25. N. Götz, U. Herzog, and M. Rettelbach. Multiprocessor and distributed system design: The integration of functional specification and performance analysis using stochastic process algebras. In *Tutorial Proc. of PERFORMANCE '93*, LNCS 729:121-146. Springer, 1993.
26. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing* 6: 512–535, 1994.
27. B. Haverkort. Markovian models for performance and dependability evaluation. In [13].
28. H. Hermanns. *Interactive Markov Chains*. PhD thesis, Universität Erlangen-Nürnberg, September 1998. Arbeitsberichte des IMMD 32/7.
29. H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 2001. To appear.
30. H. Hermanns, U. Herzog, U. Klehmet, M.Siegle, and V. Mertsiotakis. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1-4):5–35, 2000.
31. H. Hermanns and J.-P. Katoen. Automated compositional Markov chain generation for a plain-old telephony system. *Science of Computer Programming*, 36(1):97–127, 2000.
32. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov chain model checker. In *TACAS 2000*, LNCS 1785:347–362, 2000.
33. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. Towards model checking stochastic process algebra. In *IFM 2000*, LNCS 1945:420–439. Springer, 2000.
34. H. Hermanns and M. Lohrey. Priority and maximal progress are completely axiomatisable. In *CONCUR'98*, LNCS 1466:237–252. Springer, 1998.
35. H. Hermanns and M. Siegle. Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In *ARTS'99*, LNCS 1601:244–264. Springer, 1999.
36. J. Hillston. *A Compositional Approach to Performance Modelling*. Cambridge University Press, 1996.
37. G.G. Infante-Lopez, H. Hermanns, and J.-P. Katoen. Beyond memoryless distributions: model checking semi-Markov chains. In *PAPM/PROBMIV'01*, LNCS. Springer, 2001. To appear.

38. A. Jensen. Markov chains as an aid in the study of Markov processes. *Skand. Aktuarietidskrift*, 3: 87–91, 1953.
39. J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and symbolic CTMC model checking. In *PAPM/PROBMIV'01*, LNCS. Springer, 2001. To appear.
40. J.G. Kemeny and J.L. Snell. *Finite Markov Chains*. Springer, 1976.
41. K. Larsen and A. Skou. Bisimulation through probabilistic testing. *Information and Computation*, 94(1):1–28, September 1991.
42. J.F. Meyer, A. Movaghar and W.H. Sanders. Stochastic activity networks: structure, behavior and application. In *Proc. Int. Workshop on Timed Petri Nets*, pp. 106–115, IEEE CS Press, 1985.
43. R. Milner. *Communication and Concurrency*. Prentice Hall, London, 1989.
44. F. Moller and C. Tofts. A temporal calculus for communicating systems. In *CONCUR'90*, LNCS 458:401–415. Springer, 1990.
45. M.F. Neuts. *Matrix-geometric Solutions in Stochastic Models–An Algorithmic Approach*. The Johns Hopkins University Press, 1981.
46. R. De Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Concurrency*, LNCS 469: 407–419, 1990.
47. W.D. Obal and W.H. Sanders. State-space support for path-based reward variables. *Performance Evaluation*, 35: 233–251, 1999.
48. B. Plateau and K. Atif, Stochastic automata networks for modeling parallel systems. *IEEE Transactions on Software Engineering*, 17(10): 1093–1108, 1991.
49. M.L. Puterman. *Markov Decision Processes: Discrete Stochastic Dynamic Programming*. John Wiley & Sons, 1994.
50. S. Schneider. An operational semantics for timed CSP. *Information and Computation*, 116:193–213, 1995.
51. R.M. Smith, K.S. Trivedi and A.V. Ramesh. Performability analysis: measures, an algorithm and a case study. *IEEE Trans. on Comp.*, 37(4): 406–417, 1988.