Towards model checking stochastic process algebra

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

[a] *Formal Methods and Tools Group, University of Twente P.O. Box 217, 7500 AE Enschede, The Netherlands*

[b] *Lehrstuhl für Informatik 7, University of Erlangen-Nürnberg Martensstraße 3, 91058 Erlangen, Germany*

Abstract. Stochastic process algebra have been proven useful because they allow *behaviour-oriented* performance and reliability modelling. As opposed to traditional performance modelling techniques, the behaviour-oriented style supports composition and abstraction in a natural way. However, analysis of stochastic process algebra models is state-oriented, because standard numerical analysis is typically based on the calculation of (transient and steady) state probabilities. This shift of paradigms hampers the acceptance of the process algebraic approach by performance modellers. In this paper, we develop an entirely behaviour-oriented analysis technique for stochastic process algebra. The key contribution is an action-based *temporal logic* to describe behaviours-of-interest, together with a *model checking* algorithm to derive the probability with which a stochastic process algebra model exhibits a given behaviour-of-interest.

1 Introduction

The analysis of systems with respect to their performance is a crucial aspect in the design cycle of concurrent information systems. Although huge efforts are often made to analyse and tune system performance, these efforts are usually isolated from contemporary hardware and software design methodology [15, 18, 28]. This insularity of performance analysis has numerous drawbacks. Most severe, it is unclear how to incorporate performance analysis into the early stages of a design, where substantial changes are still not too costly. In these design stages, system models are nowadays developed by means of semi-formal methods such as UML or SDL.

In order to overcome the insularity problem, there is a growing tendency towards the integration of performance modelling and analysis into (semi-)formal methods, such as Petri nets [1], process algebra [21], or SDL [12]. This integration has potential benefits for the application of both formal methods and performance analysis: Using a formal method, performance models of interest are readily

⋆ supported by the German Research Council DFG under HE 1408/6-1.

available for analysis. Conversely, the availability of quantitative insight into a design clearly adds extra value to a formal design.

Process algebra is an influential approach to the modelling of concurrent systems using formal methods. Developed in the 80ies, process algebra is radically *behaviour-oriented.* Systems are modelled by describing the possible behaviours they can exhibit to the external environment. This approach led to powerful *composition* operators as means to compose behaviours hierarchically. The behaviour-oriented approach also enables one to employ *abstraction* mechanisms to compress behaviours to only those fragments relevant in a specific environment.

In a behaviour-oriented setting, the notion of a *state* is an auxiliary one. To identify a state is of no importance, since a state is completely characterized by the behaviour it exhibits. As a consequence, states exhibiting the same behaviour are considered to be indistinguishable, and hence are (or can be) collapsed to just a single state, using an appropriate notion of equivalence (such as bisimulation).

During the last decade, *stochastic* process algebra (SPA) has emerged as a promising way to carry out compositional performance and reliability modelling, mostly on the basis of continuous-time Markov chains (CTMCs) [21]. Following the same philosophy as ordinary process algebra, the stochastic behaviour of a system is described as the composition of the stochastic behaviours of its components.

However, all standard analysis algorithms for stochastic models are purely state-based. They compute interesting information about the model on the basis of state probabilities derived by either transient or steady-state analysis [35]. As a consequence, there is a disturbing shift of paradigms when it comes to the analysis of stochastic process algebra models: While the model is specified in a behaviour-oriented style, the performance properties-of-interest are defined in terms of states, on a very different level of abstraction. This shift of paradigms clearly hampers the acceptance of the SPA approach to performance modellers.

In the context of *model checking* of ordinary (i.e. non-stochastic) process algebra models, a similar mismatch has been attacked successfully. Model checking is a successful technique to establish the *correctness* of a given model, relative to a set of *temporal logic* properties which the model should satisfy [9, 10]. The most efficient model checkers use the logics **LTL** or **CTL**. Though different in nature, both logics are state-oriented, their basic building blocks are state propositions. So, at first sight they do not fit well to a behaviour-based formalism.

To import the success of model checking to behaviour-oriented formalisms, de Nicola and Vaandrager have pioneered the development of an action-based variant of **CTL**, called **aCTL** [33, 34][2]. **aCTL** is behaviour-oriented, yet it naturally corresponds to **CTL**. In particular, [33, 34] provide a translation from **aCTL** to **CTL** that allows one to perform (behaviour-oriented) **aCTL** model checking

[2] The logic **aCTL** should not be confused with the logic **ACTL**, the restriction of **CTL** to universal path quantifiers.

by means of a (state-oriented) **CTL** model checker (on a transformed model) with only linear overhead. (It should however be noted that direct **aCTL** model checkers are more popular by now [14, 32].)

In this paper, we develop a behaviour-oriented analysis technique for CTMCs, and hence for the SPA approach modelling *and* analysis become entirely behaviour-oriented. This is the central contribution of the paper. Our analysis complements behaviour-oriented CTMC modelling with SPA in the same sense as De Nicola and Vaandrager's work complements ordinary process algebraic modelling.

We develop an action-based, branching-time stochastic logic, called **aCSL** (action-based Continuous Stochastic Logic), that is strongly inspired by **CSL**, the continuous stochastic logic first proposed in [2] and further refined in [5, 3]. Similar to **CSL**, **aCSL** provides means to reason about CTMCs, but opposed to **CSL**, it is not state-oriented. Its basic constructors are sets of actions, instead of atomic state propositions. The logic provides means to specify temporal and timed properties, and means to quantify their probability. **aCSL** allows one to specify properties such as *"there is at least a 30% chance that action SEND will be observed within at most* 4 *time units"*. After defining syntax and semantics, we develop a dedicated model-checking algorithm for **aCSL**. As an application example, we study behaviour-oriented performance and reliability properties of a multiprocessor mainframe example taken from [23]. Furthermore, we show that Markovian bisimulation, an equivalence notion that can be used to compress SPA specifications compositionally, preserves **aCSL**-formulas. This property is exploited in our case study.

For efficiency reasons, our model checking algorithm is not based on a translation from **aCSL** to **CSL**. Instead, it checks **aCSL** properties directly. A translational approach would allow one to use a state-based **CSL** model checker (such as $\mathsf{E}\vdash\mathsf{MC}^2$ [24]), but with an increase of the state space. We briefly sketch the translation from **aCSL** to **CSL**, which is inspired by Emerson and Lei [13], and discuss why the linear translation in the style of Nicola and Vaandrager [33, 34] fails in the stochastic setting.

The paper is organised as follows. Section 2 introduces action-labelled Markov chains, the basic model considered in this paper. In Section 3, we define syntax and semantics of **aCSL**, derive a number of convenient operators, and discuss Markovian bisimulation. Section 4 focuses on model checking of **aCSL**. Section 5 studies **aCSL**-properties of the multiprocessor example, while Section 6 briefly discusses the translational approach to model checking **aCSL**. Section 7 concludes the paper.

2 Action-labelled Markov chains

The operational semantics of purely[3] Markovian process algebra such as TIPP [16], PEPA [29] and (the core of) EMPA [7] is defined in terms of labelled transition systems where transitions are labelled with pairs of actions and rates. In this section we briefly recall this notion and define some notations that are convenient for our purpose.

Action-labelled Markov chains. Let Act denote a set of actions, ranged over by a, b. We will use A, B as subsets of Act and adopt the convention that for singleton sets curly brackets are omitted; i.e., we write a for $\{ a \}$.

Definition 1. An *action-labelled Markov chain* (AMC) $\mathcal{M}$ is a triple $(S, A, \longrightarrow)$ where S is a set of states, $A \subseteq \mathsf{Act}$ is a set of actions, and $\longrightarrow \; \subseteq \; S \times (A \times \mathbb{R}_{>0}) \times S$ is the transition relation.

Throughout this paper we assume that any AMC is finite, i.e., has a finite number of states and is finitely branching. Transition $s \xrightarrow{a,\lambda} s'$ denotes that the system can move from state s to s' while offering action a after a delay determined by an exponential distribution with rate λ. We use the following notations:

$$\mathbf{R}_A(s, s') = \sum_{a \in A} \{ \lambda \mid s \xrightarrow{a,\lambda} s' \}$$

$$\mathbf{E}(s) = \sum_{s' \in S} \mathbf{R}_{\mathsf{Act}}(s, s')$$

$$\mathbf{P}_A(s, s') = \mathbf{R}_A(s, s') / \mathbf{E}(s).$$

Stated in words, $\mathbf{R}_A(s, s')$ denotes the cumulative rate of moving from state s to s' while offering some action from A, $\mathbf{E}(s)$ denotes the total rate with which some transition emanating from s is taken, and finally, $\mathbf{P}_A(s, s')$ is the probability of moving from state s to s' by offering an action in A. For absorbing s, $\mathbf{E}(s) = 0$ and $\mathbf{P}_A(s, s') = 0$ for any state s' and any set A. Further note that $\mathbf{R}_\varnothing(s, s') = \mathbf{P}_\varnothing(s, s') = 0$ for any states s, s'.

Paths. An infinite *path* σ is a sequence $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \xrightarrow{a_2,t_2} \ldots$ with for $i \in \mathbb{N}$, $s_i \in S$, $a_i \in \mathsf{Act}$ and $t_i \in \mathbb{R}_{>0}$ such that $\mathbf{R}_{a_i}(s_i, s_{i+1}) > 0$. For $i \in \mathbb{N}$ let $\sigma[i] = s_i$, the $(i{+}1)$-st state of σ, and $\delta(\sigma, i) = t_i$, the time spent in s_i. For $t \in \mathbb{R}_{\geqslant 0}$ and i the smallest index with $t \leqslant \sum_{j=0}^{i} t_j$ let $\sigma@t = \sigma[i]$, the state in σ at time t.

A finite path σ is a sequence $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \ldots s_{l-1} \xrightarrow{a_{l-1},t_{l-1}} s_l$ where s_l is absorbing, and $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i < l$.

For finite σ, $\sigma[i]$ and $\delta(\sigma, i)$ are only defined for $i \leqslant l$; they are defined as above for $i < l$, and $\delta(\sigma, l) = \infty$. For $t > \sum_{j=0}^{l-1} t_j$ let $\sigma@t = s_l$; otherwise, $\sigma@t$ is as above.

[3] We call a stochastic process algebra purely Markovian if the delay of any action is governed by an exponential distribution.

We denote $\sigma[i] \xrightarrow{A} \sigma[i{+}1]$ whenever $\sigma[i]$ can move to $\sigma[i{+}1]$ by performing some action in A, i.e., if $a_i \in A$. Note that $\sigma[i] \not\xrightarrow{\varnothing}$. Let $\mathit{Path}(s)$ denote the set of paths starting in s. A Borel space over $\mathit{Path}(s)$ can be defined in a similar way as in [5] and is omitted here.

3 An action-based continuous stochastic logic

This section describes the action-based stochastic logic **aCSL** which is inspired by the action-based logic **aCTL** by De Nicola and Vaandrager [33] and the stochastic logic **CSL** by Baier *et al.* [5], which in turn is based on the work of Aziz *et al.* [2].

3.1 Syntax and semantics of aCSL

Syntax. For $p \in [0,1]$ and $\bowtie \in \{ \leqslant, <, \geqslant, > \}$, the state-formulas of **aCSL** are defined by the grammar

$$\Phi ::= \mathit{true} \;\Big|\; \Phi \wedge \Phi \;\Big|\; \neg\Phi \;\Big|\; \mathcal{S}_{\bowtie p}\,(\Phi) \;\Big|\; \mathcal{P}_{\bowtie p}\,(\varphi)$$

where path-formulas are defined for $t \in \mathrm{I\!R}_{>0} \cup \{ \infty \}$ by

$$\varphi ::= \Phi \,{}_A\mathcal{U}^{<t}\, \Phi \;\Big|\; \Phi \,{}_A\mathcal{U}^{<t}{}_B\, \Phi.$$

Note that atomic propositions are absent. The boolean connectives such as $\vee$ and $\Rightarrow$ are derived in the obvious way. The probabilistic operator $\mathcal{P}_{\bowtie p}\,(.)$ replaces the CTL path quantifiers $\exists$ and $\forall$ that can be re-invented — up to fairness [6] — as the extremal probabilities $\mathcal{P}_{>0}\,(.)$ and $\mathcal{P}_{\geqslant 1}\,(.)$. The state formulas are directly adopted from **CSL**: $\mathcal{S}_{\bowtie p}\,(\Phi)$ asserts that the steady-state probability for a Φ-state meets the bound $\bowtie p$ and $\mathcal{P}_{\bowtie p}\,(\varphi)$ asserts that the probability measure of the paths satisfying φ meets the bound $\bowtie p$.

The path-formula $\Phi_1 \,{}_A\mathcal{U}^{<t}\, \Phi_2$ is fulfilled by a path if a Φ_2-state is eventually reached via visiting only Φ_1-states before, while taking only A-transitions; besides, going from the beginning of the path until reaching the Φ_2-state should last at most t time units. The formula $\Phi_1 \,{}_A\mathcal{U}^{<t}{}_B\, \Phi_2$ requires in addition that (i) a move to a Φ_2-state is actually made and that (ii) this transition is labelled by some action in B. We remark the following. Due to the fact that the Φ_2-state *must* be reached via a B-transition, the formula $\Phi_1 \,{}_A\mathcal{U}^{<t}{}_B\, \Phi_2$ is invalid in a $(\neg\Phi_1 \wedge \Phi_2)$-state s: although the state satisfies Φ_2, it is not able to move from a Φ_1-state to a Φ_2-state via a B-transition as it does not fulfill Φ_1. The formula $\Phi_1 \,{}_A\mathcal{U}^{<t}\, \Phi_2$ is, however, valid in state s, since for the validity of this formula it is not required that a transition into a Φ_2-state is made. Thus, whereas for $\Phi_1 \,{}_A\mathcal{U}^{<t}\, \Phi_2$ it suffices to currently be in a Φ_2-state, this is not the case for

$\Phi_1 \; {}_A\mathcal{U}^{<t}{}_B \; \Phi_2$.[4]

The major differences with a 'standard' until-formula $\Phi_1 \, \mathcal{U} \, \Phi_2$ of linear and branching temporal logics are that restrictions are put on (i) the action labels of transitions to be taken and on (ii) the amount of time that is needed to reach a Φ_2-state. This can be made precise in the following way:

$$\Phi_1 \, \mathcal{U} \, \Phi_2 = \Phi_1 \; {}_{\mathsf{Act}}\mathcal{U}^{<\infty} \, \Phi_2.$$

In the sequel, we use $\Phi_1 \; {}_A\mathcal{U}_B \, \Phi_2$ as an abbreviation of $\Phi_1 \; {}_A\mathcal{U}^{<\infty}{}_B \, \Phi_2$ and $\Phi_1 \; {}_A\mathcal{U} \, \Phi_2$ as an abbreviation of $\Phi_1 \; {}_A\mathcal{U}^{<\infty} \, \Phi_2$. These are the untimed versions of the until-operators ${}_A\mathcal{U}^{<t}{}_B$ and ${}_A\mathcal{U}^{<t}$.

An interesting aspect of **aCSL** is that the following set of next-operators are all derived operators:

$$\begin{aligned} X_A^{<t} \, \Phi &= \mathit{true} \; {}_{\varnothing}\mathcal{U}^{<t}{}_A \, \Phi \\ X_A \, \Phi &= X_A^{<\infty} \, \Phi \\ X \, \Phi &= X_{\mathsf{Act}} \, \Phi. \end{aligned}$$

The formula $X_A^{<t} \, \Phi$ asserts that from the current state an A-transition can be made to a Φ-state before time t. Remark that the Φ-state must be reached by the first transition, as — due to the empty set of actions — further transitions are disallowed. X_A is the action-labelled next-operator from **aCTL**, whereas X is the traditional state-based next-operator.

Note 2. In our logic, the next operator is derived from the until-operator. In **aCTL** the reverse holds [33]. This stems from the special treatment of internal, i.e., τ-labelled, transitions in **aCTL**. For instance in **aCTL**, $X_{\varnothing} \, \Phi$ allows to reach a Φ-state by an internal transition (but not any other). In our setting, internal transitions are treated as any other transition, and accordingly, $X_{\varnothing} \, \Phi$ is invalid for any state. We have made this difference deliberately: whereas **aCTL** is aimed to characterize branching bisimulation – a slight variant of weak bisimulation equivalence – we focus on characterizing a strong equivalence like lumping equivalence (since exact weak equivalences on AMCs cannot be obtained [21]).

The temporal operator $\Diamond$ and its variants are derived in the following way:

$$\begin{aligned} {}_A\Diamond^{<t}\Phi &= \mathit{true} \; {}_A\mathcal{U}^{<t} \, \Phi \\ {}_A\Diamond \; \Phi &= {}_A\Diamond^{<\infty} \, \Phi \\ \Diamond^{<t}\Phi &= {}_{\mathsf{Act}}\Diamond^{<t} \, \Phi \end{aligned}$$

[4] If we enlarged the set of path-formulas such that conjunction and negation of path-formulas is allowed (in a similar way as for CTL*), the relationship between ${}_A\mathcal{U}^{<t}{}_B$ and ${}_A\mathcal{U}^{<t}$ could be made precise as follows:

$$\Phi_1 \; {}_A\mathcal{U}^{<t} \, \Phi_2 = \Phi_2 \vee (\Phi_1 \; {}_A\mathcal{U}^{<t}{}_A \, \Phi_2).$$

A path fulfills ${}_A\Diamond^{<t}\Phi$ if it reaches a Φ-state within t time units by only performing A-actions. Formulas ${}_A\Diamond\ \Phi$ and $\Diamond^{<t}\Phi$ denote the generalisations to infinite time and arbitrary actions. Their combination, $\Diamond\ \Phi$, corresponds to the well-known "eventually" operator. An even more discerning $\Diamond$-operator can be defined by

$$ {}_A\Diamond_B^{<t}\Phi \ = \ \mathit{true}\ {}_A\mathcal{U}^{<t}{}_B\ \Phi \ \text{ and } \ {}_A\Diamond_B\Phi \ = \ {}_A\Diamond_B^{<\infty}\ \Phi $$

Here, the path leading to the Φ-state consists of an arbitrary number of A-actions, followed by a single B-action. Dual to these $\Diamond$-operators is the set of $\Box$-operators, of which we only mention the following:

$$ \mathcal{P}_{\bowtie p}\left({}_A\Box^{<t}\Phi\right) \ = \ \neg\mathcal{P}_{\bowtie p}\left({}_A\Diamond^{<t}\neg\Phi\right) \ \text{ and } \ \mathcal{P}_{\bowtie p}\left({}_A\Box_B^{<t}\Phi\right) \ = \ \neg\mathcal{P}_{\bowtie p}\left({}_A\Diamond_B^{<t}\neg\Phi\right) $$

with the obvious generalisations to infinite time and/or arbitrary sets of actions. Finally, existential and universal quantification are introduced as

$$ \exists\varphi \ = \ \mathcal{P}_{>0}(\varphi) \ \text{ and } \ \forall\varphi \ = \ \mathcal{P}_{\geqslant 1}(\varphi) $$

Note that by this definition formula $\forall\varphi$ holds even if there exists a path that does not satisfy φ, if that path has zero probability mass.

We consider the modal operators from Hennessy-Milner logic [19] and the μ-calculus [30] as derived operators. They are obtained as follows:

$$ \langle A\rangle\,\Phi = \mathcal{P}_{>0}\left(X_A\,\Phi\right) \ \text{ and } \ [A]\,\Phi = \neg\langle A\rangle\,\neg\Phi. $$

The modal operator $\langle A\rangle\,\Phi$ states that there is some A-transition from the current state to a Φ-state, whereas $[A]\,\Phi$ states that for all A-transitions from the current state a Φ-state is reached.

Note 3. The modal operator $\langle a\rangle_p\,\Phi$ from the probabilistic modal logic **PML** [31] cannot be obtained as a derived operator in our setting. The state-formula $\langle a\rangle_p\,\Phi$ asserts that, *given that* an a-transition happens, the probability of moving to a Φ-state is at least p. This interpretation fits well to the reactive probabilistic setting used in [31] in which over each set of equally labelled transitions a discrete probability space is defined. Since we consider a generative setting — having a discrete probability space over all, possibly different labelled, transitions — the probability in a formula like $\langle a\rangle_p\,\Phi$ is relative to *all* transitions, and not just the ones labelled with a. In the continuous variant of **PML** [8] a similar approach as in [31] is taken, and a reactive interpretation is used.

Semantics. The **aCSL** state-formulas are interpreted over the states of an AMC $(S, A, \longrightarrow)$. Let $Sat(\Phi) = \{\, s\in S \mid s\models\Phi \,\}$.

$$ \begin{array}{llcll} s\models \mathit{true} & \text{for all } s\in S & & s\models\Phi_1\wedge\Phi_2 & \text{iff } s\models\Phi_i,\ \text{for } i{=}1,2 \\ s\models\neg\Phi & \text{iff } s\not\models\Phi & & s\models\mathcal{S}_{\bowtie p}(\Phi) & \text{iff } \pi(s, Sat(\Phi))\bowtie p \\ & & & s\models\mathcal{P}_{\bowtie p}(\varphi) & \text{iff } Prob(s,\varphi)\bowtie p \end{array} $$

Here, $\pi(s, S')$ denotes the steady-state probability to be in a state of set S' when starting in s. It is defined by means of a probability measure[5] Pr on the set of paths $\mathit{Path}(s)$ emanating from s.

$$\pi(s, S') = \lim_{t \to \infty} \Pr\{\, \sigma \in \mathit{Path}(s) \mid \sigma@t \in S' \,\}$$

$\mathit{Prob}(s, \varphi)$ denotes the probability measure of all paths satisfying φ given that the system starts in state s, i.e.,

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

The fact that these sets are measurable follows by easy verification from the Borel space construction given in [5].

The meaning of the path-operators is defined by a satisfaction relation, also denoted by $\models$, between a path and a path-formula. We define: $\sigma \models \Phi_1\, {}_A\mathcal{U}^{<t}\, \Phi_2$ if and only if:

$$\begin{aligned} \exists k \geqslant 0.\, \Big(& \sigma[k] \models \Phi_2 \\ & \wedge (\forall i < k.\, \sigma[i] \models \Phi_1 \wedge \sigma[i] \xrightarrow{A} \sigma[i{+}1]) \;\wedge\; t > \textstyle\sum_{i=0}^{k-1} \delta(\sigma, i)\Big) \end{aligned} \tag{1}$$

where we recall that $\delta(\sigma, i)$ denotes the sojourn time in state $\sigma[i]$. Thus, $\Phi_1\, {}_A\mathcal{U}^{<t}\, \Phi_2$ is valid for a path if at some time instant before t a Φ_2-state is reached — assume this is the $(k{+}1)$-st state (for $k \geqslant 0$) in the path so far — by visiting only Φ_1-states, while taking only A-transitions along the entire path.

For the other until-formula we have: $\sigma \models \Phi_1\, {}_A\mathcal{U}^{<t}{}_B\, \Phi_2$ if and only if:

$$\begin{aligned} \exists k > 0.\, \Big(& \sigma[k] \models \Phi_2 \;\wedge\; (\forall i < k{-}1.\, \sigma[i] \models \Phi_1 \wedge \sigma[i] \xrightarrow{A} \sigma[i{+}1]) \\ & \wedge \sigma[k{-}1] \models \Phi_1 \;\wedge\; \sigma[k{-}1] \xrightarrow{B} \sigma[k] \;\wedge\; t > \textstyle\sum_{i=0}^{k-1} \delta(\sigma, i)\Big) \end{aligned}$$

Note the subtle difference with (1): For $\Phi_1\, {}_A\mathcal{U}^{<t}{}_B\, \Phi_2$ to be valid, there *should* be a single transition leading to a Φ_2-state labelled by some action in B.

It is left to the interested reader to check that $s \models X_A^{<t}\, \Phi$ iff

$$\sigma[1] \models \Phi \;\wedge\; \sigma[0] \xrightarrow{A} \sigma[1] \;\wedge\; t > \delta(\sigma, 0).$$

This agrees with the intuitively expected semantics for $X_A^{<t}\, \Phi$.

[5] The probability measure Pr is defined by means of a Borel space construction on paths. We refer to [5] for a formal definition.

3.2 Markovian bisimulation

In this section, we show that **aCSL** is invariant under the application of Markovian bisimulation. Markovian bisimulation, a variant of Larsen-Skou bisimulation [31], is a congruence for the stochastic process algebras TIPP [16] and PEPA [29]. In the context of process algebraic composition operators, a congruence relation can be used to compress the state space of components before composition, in order to alleviate the state space explosion problem, under the condition that the relation equates only components obeying the same properties. Hence the question arises whether a Markovian bisimulation R can be applied to compress models (or model components) prior to model checking **aCSL**-formulas. In general, this requires that the validity of **aCSL**-formulas is preserved when moving from an AMC $\mathcal{M}$ to its quotient AMC $\mathcal{M}/R$. We establish this property in Theorem 5.

Definition 4. A *Markovian bisimulation* on $\mathcal{M} = (S, A, \longrightarrow)$ is an equivalence R on S such that whenever $(s, s') \in R$ then $\mathbf{R}_a(s, C) = \mathbf{R}_a(s', C)$ for all $C \in S/R$ and all $a \in \mathsf{Act}$. States s and s' are *Markovian bisimilar* iff there exists a Markovian bisimulation R that contains (s, s').

Here, S/R denotes the quotient space and $\mathbf{R}_a(s, C)$ abbreviates $\sum_{s' \in C} \mathbf{R}_a(s, s')$. Let $\mathcal{M}/R$ be the AMC that results from building the quotient space of $\mathcal{M}$ under R, i.e., $\mathcal{M}/R = (S/R, A, \longrightarrow)$. In the following we write $\models_{\mathcal{M}}$ for the satisfaction relation $\models$ (on **aCSL**) on $\mathcal{M}$.

Theorem 5. *Let R be a Markovian bisimulation on $\mathcal{M}$ and s a state in $\mathcal{M}$. Then:*

(a) For all state-formulas Φ: $s \models_{\mathcal{M}} \Phi$ iff $[s]_R \models_{\mathcal{M}/R} \Phi$
(b) For all path-formulas φ: $Prob^{\mathcal{M}}(s, \varphi) = Prob^{\mathcal{M}/R}([s]_R, \varphi)$.

In particular, Markovian bisimilar states satisfy the same **aCSL** *formulas.*

In the appendix, we sketch the proof of Theorem 5. The detailed proof can be found in [25]. This result allows to verify **aCSL**-formulas on the potentially much smaller AMC $\mathcal{M}/R$ rather than on $\mathcal{M}$. The quotient with respect to Markovian bisimilarity can be computed by a modified version of the partition refinement algorithm for ordinary bisimulation without an increase in complexity [26]. In addition, due to the congruence property of Markovian bisimularity on TIPP and PEPA, a specification can be reduced in a compositional way, thus avoiding the need to model check the (possibly very large) state space S. This feature is exploited in the case study discussed in Section 5.

4 Model checking aCSL

The general strategy for model checking **aCSL** proceeds in the standard way: For a given state formula Φ, the algorithm recursively computes the sets of states satisfying the sub-formulas of Φ, and constructs from them the set of states satisfying Φ. For boolean connectives, the strategy is obvious. Model checking steady-state properties $\mathcal{S}_{\bowtie p}(\Phi)$ involves solving linear systems of equations, after determining (bottom) strongly connected components, exactly as in the **CSL** context [5].

Model checking the probabilistic quantifier $\mathcal{P}_{\bowtie p}(\varphi)$ is the crucial difficulty. It relies on the following characterizations of $Prob(s, \varphi)$. We discuss the characterizations by structural induction over φ. For the sake of simplicity, we first treat the simple untimed until-formulas.

Untimed until. For $\varphi = \Phi_1 \, {}_A\mathcal{U} \, \Phi_2$ we have that $Prob(s, \varphi)$ is given by the following equations: $Prob(s, \varphi) = 1$ if $s \models \Phi_2$,

$$\sum_{s' \in S} \mathbf{P}_A(s, s') \cdot Prob(s', \varphi)$$

if $s \models \Phi_1 \wedge \neg \Phi_2$, and 0 otherwise. For $A = \mathsf{Act}$ we obtain the equation for standard until as for DTMCs [17].

Let $\varphi = \Phi_1 \, {}_A\mathcal{U}_B \, \Phi_2$. For $s \not\models \Phi_1$, the formula is invalid. As for $s \models \Phi_1$ the situation is more involved let us, for the sake of simplicity, assume that A and B are disjoint, i.e. $A \cap B = \varnothing$. Then the only interesting possibilities starting from s are (i) to directly move to a Φ_2-state via a B-transition, in which case the formula φ is satisfied with probability 1, or (ii) to take an A-transition leading to Φ_1-state s' which satisfies φ with probability $Prob(s', \varphi)$. Accordingly, for $A \cap B = \varnothing$, $Prob(s, \varphi)$ can be characterized by:

$$\sum_{s' \models \Phi_2} \mathbf{P}_B(s, s') + \sum_{s' \models \Phi_1} \mathbf{P}_A(s, s') \cdot Prob(s', \varphi). \tag{2}$$

In the general case we have to take into account that A and B may not be disjoint. Equation (2) does not apply now, since an $(A \cap B)$-transition into a state that satisfies both Φ_1 and Φ_2 is "counted" twice. We therefore obtain that $Prob(s, \varphi)$ is the least solution of the following set of equations:

$$\sum_{s' \models \Phi_2} \mathbf{P}_B(s, s') + \sum_{s' \models \Phi_1} \mathbf{P}_A(s, s') \cdot Prob(s', \varphi) - \sum_{s' \models \Phi_1 \wedge \Phi_2} \mathbf{P}_{A \cap B}(s, s') \cdot Prob(s', \varphi)$$

if $s \models \Phi_1$, and 0 otherwise. Note that

$$Prob(s, X_B \, \Phi) = Prob(s, true \, {}_{\varnothing}\mathcal{U}_B \, \Phi) = \sum_{s' \models \Phi} \mathbf{P}_B(s, s')$$

which coincides, for $B = \mathsf{Act}$, with the characterization of next for DTMCs [17]. Thus, the probability that s satisfies $X_B \, \Phi$ equals the sum of the probabilities to move to a Φ-state via a single B-transition. The reader is also invited to check that for $B = \varnothing$ there is no state that satisfies $\Phi_1 \, {}_A\mathcal{U}_B \, \Phi_2$ with positive probability.

Timed until. For $\varphi = \Phi_1 \,_A\mathcal{U}^{<t}\, \Phi_2$ we have that $Prob(s, \varphi)$ is the least solution of the following set of equations: $Prob(s, \varphi) = 1$ if $s \models \Phi_2$, and

$$\int_0^t e^{-\mathbf{E}(s)\cdot x} \cdot \sum_{s' \in S} \mathbf{R}_A(s, s') \cdot Prob(s', \Phi_1 \,_A\mathcal{U}^{<t-x}\, \Phi_2)\ dx$$

if $s \models \Phi_1 \wedge \neg\Phi_2$, and 0 otherwise. For state s satisfying $\Phi_1 \wedge \neg\Phi_2$, the probability of reaching a Φ_2-state within t time units from s equals the probability of reaching some direct successor s' of s within x time units, multiplied by the probability of reaching a Φ_2-state from s' within the remaining time $t{-}x$. Since there may be different paths from s to Φ_2-states, the sum is taken over all these possibilities. (Note that by taking $t = \infty$ we obtain, after some straight-forward calculations, the characterisation for untimed until $_A\mathcal{U}$ given before).

For $\varphi = \Phi_1 \,_A\mathcal{U}^{<t}{}_B\, \Phi_2$ we have that $Prob(s, \varphi)$ is the least solution of the following set of equations:

$$\int_0^t e^{-\mathbf{E}(s)\cdot x} \cdot \Bigg(\sum_{s' \models \Phi_2} \mathbf{R}_B(s, s') + \sum_{s' \models \Phi_1} \mathbf{R}_A(s, s') \cdot Prob(s', \Phi_1 \,_A\mathcal{U}^{<t-x}{}_B\, \Phi_2) \\ - \sum_{s' \models \Phi_1 \wedge \Phi_2} \mathbf{R}_{A\cap B}(s, s') \cdot Prob(s', \Phi_1 \,_A\mathcal{U}^{<t-x}{}_B\, \Phi_2) \Bigg)\ dx$$

if $s \models \Phi_1$, and 0 otherwise. This characterization can be justified in the same way as for its untimed counterpart, i.e., $\Phi_1 \,_A\mathcal{U}_B\, \Phi_2$, given the above explanation for the simpler timed until variant. Let us consider what this yields for $X_B^{<t}\, \Phi$:

$$Prob(s, X_B^{<t}\, \Phi) = Prob(s, true \,_{\varnothing}\mathcal{U}_B^{<t}\, \Phi) = \int_0^t e^{-\mathbf{E}(s)\cdot x} \cdot \sum_{s' \models \Phi} \mathbf{R}_B(s, s')$$

which, after some straight-forward calculations, leads to

$$\sum_{s' \models \Phi} \mathbf{P}_B(s, s') \cdot \left(1 - e^{-\mathbf{E}(s)\cdot t}\right).$$

The first term of the product denotes the discrete probability to move via a single B-transition to a Φ-state, whereas the second term denotes the probability to leave state s within t time units.

This equational characterization allows one to model check **aCSL** formulas by means of approximate numerical techniques. The concrete implementation closely follows the one for **CSL** outlined in [5] and implemented in [24]. We are currently investigating whether the solution of the above integral equations can be reduced to standard transient analysis via uniformisation, as in [3].

5 Case study: multiprocessor mainframe with software failures

We consider a multiprocessor mainframe which was first introduced in [27] and has since then served as a standard SPA example, see e.g. [23, 11]. Here we only briefly repeat the main features of the model.

5.1 Specification of multiprocessor mainframe

The multiprocessor mainframe serves two purposes: It has to process database transactions submitted by users, and it provides computing capacity to programmers maintaining the database. The system is subject to software failures which are modelled as special jobs. On the top level, the system is composed of two processes (cf. Fig. 1).

$$System := Load \,|[putUserJob, putProgJob, fail]|\, Machine$$

Process *Load* represents the system load caused by the database users, the programmers and the failures. The mainframe itself is modelled by the *Machine* process. The three different system load components are modelled as so-called

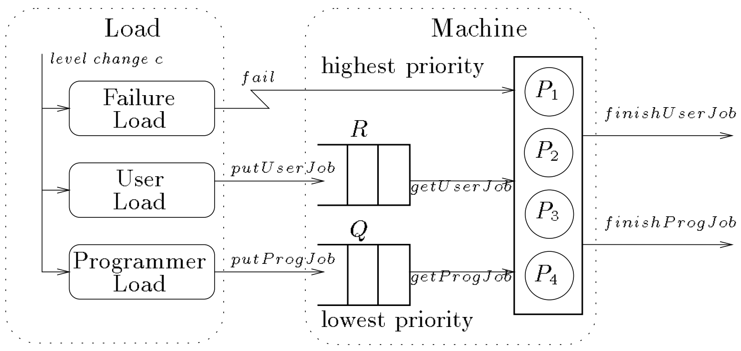

Fig. 1. Mainframe model structure

Markov modulated Poisson processes, see [27]. The intensity of the load alters between different levels. To realize a synchronous change of load level, a synchronizing action c is used.

$$Load := ProgLoad \,|[c]|\, UserLoad \,|[c]|\, FailLoad$$

The *Machine* component consists of two finite queues and four identical processors. The queues buffer incoming jobs. They are controlled by a priority mechanism to ensure that programmer jobs have the lowest priority, while failures have the highest priority. The priority mechanism is realised by appropriate synchronisation of the queue processes. For instance, process Q can only deliver a job to a processor if queue R is empty and no failure is present. Furthermore, the insertion of new jobs into the system is prohibited once a failure has occured, until the system is repaired.

Each of the four processors executes user or programmer jobs waiting in the respective queues, unless a failure occurs. As failures have preemptive priority over the other two job classes, all processors stop working once action $fail$ has occured and then wait until the system will recover (via action $repair$).

5.2 Properties of interest

This section contains some example properties which are of interest for the multiprocessor mainframe model. For each property, a description in plain English, its **aCSL** formulation and some explanation are given. We first introduce some purely functional requirements to ensure that the priority mechanism is properly realised by the model. Then we turn to the performance and reliability requirements which the system should satisfy. For $A \subseteq \mathsf{Act}$ we let $\overline{A}$ denote $\mathsf{Act} \setminus A$. We use the following sets of actions: $Get := \{getUserJob, getProgJob\}$, $Put := \{putUserJob, putProgJob\}$, $Fin := \{finishUserJob, finishProgJob\}$ and $FailRep := \{fail, repair\}$. We omit brackets for singleton sets.

Φ_1: If there are user jobs waiting, the processors will not start programmer jobs.

$$\Phi_{UserJobWaiting} \;\Rightarrow\; \neg\langle getProgJob\rangle\ true$$

where $\Phi_{UserJobWaiting}$ is defined by $\exists\ ({}_{\overline{putUserJob}}\Diamond_{getUserJob}\ true)$, characterizing at least one user job waiting in the queue.

Φ_2: Whenever a failure occurs, no jobs can be inserted into the queues until the system is repaired.

$$[fail]\ \forall\ ({}_{\overline{Put}}\Diamond_{repair}\, true)$$

Φ_3: Whenever a failure occurs, the processors will be blocked until the system is repaired.

$$[fail]\ \forall\ ({}_{\overline{Get \cup Fin}}\Diamond_{repair}\, true)$$

Φ_4: After a repair, both queues are empty.

$$[repair]\ (\ \neg\, \Phi_{UserJobWaiting}\ \wedge\ \neg\, \Phi_{ProgJobWaiting}\ )$$

where $\Phi_{ProgJobWaiting}$ characterizes at least one waiting programming job, defined in a similar way as $\Phi_{UserJobWaiting}$. This is an example of a property which is not true, since a failure does not cause the queues to be flushed.

Φ_5: In steady state, the probability of low priority programming jobs having to wait because of user jobs being served is smaller than 0.01.

$$\mathcal{S}_{<0.01}\,(\langle finishUserJob\rangle true\ \wedge\ \Phi_{\mathrm{ProgJobWaiting}})$$

Φ_6: At least two processors are occupied by user jobs.

$$\langle finishUserJob\rangle\ \langle finishUserJob\rangle\ true$$

Φ_7: In steady state, the probability that at least two processors are occupied by user jobs is greater than 0.002.

$$\mathcal{S}_{>0.002}\,(\Phi_6)$$

Φ_8: There is at least a 30% chance that some job will be finished within at most 4 time units.

$$\mathcal{P}_{\geqslant 0.3}\,({}_{\overline{Fin}}\Diamond^{<4}_{Fin}\, true)$$

Φ_9: In steady state, the probability of the system being unavailable (i.e. waiting for repair) is at most 0.05.

$$\mathcal{S}_{\leqslant 0.05}\left(\exists({}_{\overline{FailRep}}\Diamond_{repair}\, true)\right)$$

Φ_{10}: After a system failure, there is a chance of more than 90% that it will come up again within the next 5 time units.

$$[fail]\ \mathcal{P}_{>0.9}\left({}_{\overline{repair}}\Diamond^{<5}_{repair}\, true\right)$$

The fact that the above property holds for all states can be expressed by $\forall\,\Box\,\Phi_{10}$. Slightly weaker, one might require the above property to hold on the long run, formulated as $\mathcal{S}_{\geqslant 1}\,(\Phi_{10})$.

5.3 Verification results

In this section we report on our experience with the verification of the above properties. The results have been obtained by means of a trial implementation, basically an extension of the model checker $\mathsf{E}\vdash\mathsf{MC}^2$ [24]. The implementation does not yet support the full logic **aCSL**, therefore property Φ_8 has not been checked.

For the properties listed in the previous section we present the verification runtimes in Table 1. We checked three models: A small model with 4 (2) programmer (user) buffer places, an intermediate model with 10 (4) programmer (user) buffer places and a large model with 40 (10) programmer (user) buffer places. The small model has 3690 states and 24009 transitions, the intermediate model has 13530 states and 91069 transitions and the large model has 110946 states and 761989 transitions. However, we did not perform model checking on the original models but on models with compressed state spaces which we gained through the application of Markovian bisimilarity (in the example multiprocessor system, the main potential for reduction stems from the symmetry of the four identical processors). By Theorem 5, the compressed models satisfy the same properties as the original ones. After bisimilarity compression, the small model has 720 states and 3219 transitions, the intermediate model has 2640 states and 12295 transitions and the large model has 21648 states and 103471 transitions. All steady state properties given in the table were double checked with TIPPTool [22].

states (original)	3690	13530	110946
(compressed)	720	2640	21648

property	verification runtimes (in seconds)		
Φ_1	0.012	0.037	0.268
Φ_2	0.008	0.049	0.864
Φ_3	0.008	0.039	0.319
Φ_4	0.003	0.005	0.036
Φ_5	0.642	2.371	18.750
Φ_6	0.001	0.002	0.014
Φ_7	0.558	2.122	18.814
Φ_9	0.554	2.009	18.819
Φ_{10}	2.557	11.404	92.324

Table 1. Verification runtimes

6 On translating aCSL to CSL

The design of **aCSL** closely follows the work of De Nicola and Vaandrager on **aCTL** [33]. For what concerns model checking, they propose a translation $\mathcal{K}$ from **aCTL** into **CTL**, and a transformation (also denoted $\mathcal{K}$) from action-labelled to state-labelled transition systems in such a way that for an arbitrary **aCTL** formula Φ and arbitrary action-labelled transition system $\mathcal{M}$ (with the obvious notation):

$$s \models_{\mathcal{M},\mathrm{aCTL}} \Phi \quad \text{iff} \quad \mathcal{K}(s) \models_{\mathcal{K}(\mathcal{M}),\mathrm{CTL}} \mathcal{K}(\Phi) \tag{3}$$

In this way, **aCTL** model checking can be reduced to **CTL** model checking, by checking a translated formula $\mathcal{K}(\Phi)$ on a transformed model $\mathcal{K}(\mathcal{M})$. The bypass via $\mathcal{K}$ blows up the model and the formula, but only by a factor of 2, whence it follows that model checking **aCTL** has the same worst case (space and time) complexity as **CTL**. The key idea of this transformation is to break each transition of $\mathcal{M}$ in two, connected by a new auxiliary state. The new state is labelled with the action label of the original transition, playing the role of an atomic state proposition. (The original source and target states are labelled with a distinguished symbol $\perp$). Formula Φ is manipulated by $\mathcal{K}$ in such a way that starting from some state $\mathcal{K}(s)$ essentially all the labellings of original states ($\perp$) do not matter, while the ones of auxiliary states do. Unfortunately, this approach does not carry over to the Markov chain setting, because splitting a Markovian transition in two implies splitting an exponential distribution in two. However, no sequence of two exponential distribution agrees with an exponential distribution. Since **aCSL** is powerful enough to detect differences in transient probabilities, this approach is infeasible.

Even though a translation in the style of De Nicola and Vaandrager does not allow one to reduce **aCSL** to **CSL**, this does not imply that such a reduction is generally infeasible. For the sake of completeness, we remark that it is indeed possible to reduce model checking **aCSL** to model checking (slight variants of) **CSL**. We briefly sketch two possibilities:

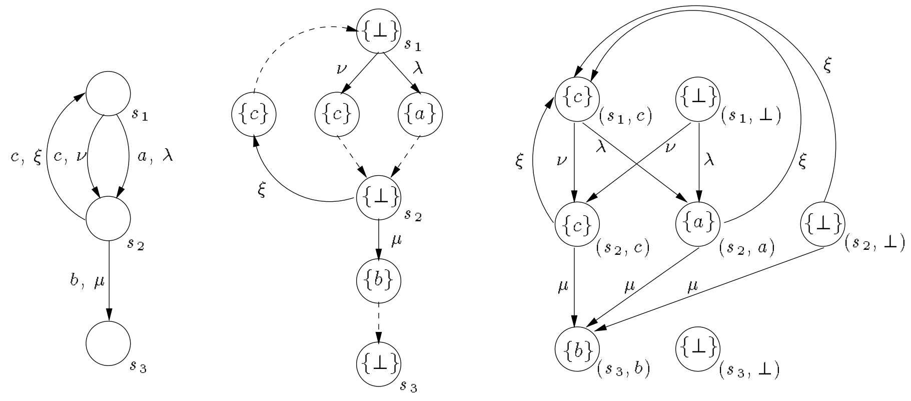

Fig. 2. Transformation example from AMC (left) to IMC (middle) and SMC (right)

- Apply the transformation of [33] and map AMCs to interactive Markov chains (IMC) [20]. This transformation is exemplified in Figure 2 (from left to middle), where state labellings appear as sets, and dashed transitions are supposed to be immediate. In general, IMC allow for nondeterminism, but this phenomenon is not introduced by the translation. Therefore, the model checking algorithm of [5] can be lifted to this subset of IMC.
- Transform AMCs to state-labelled CTMCs (SMC), using a transformation inspired by Emerson and Lei [13]. The main idea is to split each state into a number of duplicates, given by the number of different incoming actions it possesses, and label each duplicate with a different action, and distribute the incoming transitions accordingly. (In order to track the first transition delay correctly, one additional $\perp$-labelled duplicate per state is needed.) To give an intuitive idea, this transformation is depicted in Figure 2 (from left to right). A mapping $\mathcal{K}$ from **aCSL** to a minor variant of **CSL** exists that ensures $s \models_{\mathcal{M}} \phi$ to hold if and only if $(s, \perp) \models_{\mathcal{K}(\mathcal{M})} \mathcal{K}(\phi)$ holds. (The satisfaction relation $\models_{\mathcal{K}(\mathcal{M})}$ on **CSL** requires a subtle – but straight-forward to implement – modification.) Details can be found in [25]. In the worst case, the state space is blown up by a factor given by the maximal number of distinct actions entering a state.

Notice that both translations sketched above require a small modification of the model checking algorithm for **CSL** [3, 5]. Furthermore, both approaches induce a blow up of the model by a linear factor. To avoid these drawbacks, we have decided to develop a direct model checking algorithm, as sketched in Section 4. Remark that despite the aforementioned translations from **aCTL** to **CTL**, dedicated model checkers for **aCTL** are more popular by now [14, 32].

7 Concluding remarks

This paper has introduced a behaviour-oriented analysis approach for Markovian stochastic process algebra. From a conceptual as well as from a pragmatic point of view, this approach closes a disturbing gap in the process algebraic approach to performance and dependability modelling. In particular, performance engineers are no longer confronted with the need to switch from a behaviour-oriented to a state-oriented view when it comes to model analysis.

The behaviour-oriented modelling and analysis approach outlined in this paper has four ingredients: (1) A standard stochastic process algebra (such as TIPP, PEPA, EMPA) is used to model the system under consideration as an action-labelled CTMC. (2) The action-based logic **aCSL** serves as a powerful means to specify properties of interest. (3) A model checking algorithm decides which properties are satisfied by the Markov chain model. (4) Since Markovian bisimilarity preserves **aCSL** properties, it can be used to compress the model (or the model components, due to the congruence property for TIPP and PEPA) before model checking. We have illustrated all four ingredients by means of the multiprocessor mainframe case study.

$\mathbf{PML}_\mu$ [8], the continuous-time variant of **PML** [31], is another logic on action-labelled CTMCs. $\mathbf{PML}_\mu$ and **aCSL** are incomparable, because $\mathbf{PML}_\mu$ takes a reactive point of view, while our view is generative (see Note 3). $\mathbf{PML}_\mu$ is not considered in the context of model checking, instead it serves as the foundation of a formalism to assign rewards to states, i.e., to construct Markov reward models. The thus obtained models are then analyzed with standard (steady-state) numerical analysis. $\mathbf{PML}_\mu$ does neither provide means to quantify probability nor to reason about time intervals.

For the future, we intend to study to what extent **aCSL** can be extended towards the analysis of Markov reward models. In the state-based setting, we have recently developed a continuous reward logic (**CRL**) that allows bounds on rewards to be checked, and naturally combines with **CSL** [4].

Acknowledgements. The authors are grateful to Lennard Kerber, Frits Vaandrager and Moshe Vardi for highly valuable discussions.

References

1. M. Ajmone Marsan, G. Conte, and G. Balbo. A class of generalised stochastic Petri nets for the performance evaluation of multiprocessor systems. *ACM Tr. on*

Comp. Sys., **2**(2): 93–122, 1984.

2. A. Aziz, K. Sanwal, V. Singhal and R. Brayton. Verifying continuous time Markov chains. In *CAV*, LNCS 1102: 269–276, 1996.
3. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. Model checking continuous-time Markov chains by transient analysis. In *CAV*, LNCS 1855, 2000.
4. C. Baier, B.R. Haverkort, H. Hermanns and J.-P. Katoen. On the logical characterization of performability properties. In *ICALP*, LNCS 1853, 2000.
5. C. Baier, J.-P. Katoen and H. Hermanns. Approximate symbolic model checking of continuous-time Markov chains. In *CONCUR*, LNCS 1664: 146–162, 1999.
6. 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.
7. M. Bernardo and R. Gorrieri. A tutorial on EMPA: a theory of concurrent processes with nondeterminism, priorities, probabilities and time. *Th. Comp. Sc.*, **202**: 1–54, 1998.
8. G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In *ARTS*, LNCS 1601: 211–227, 1999.
9. E.M. Clarke and R.P. Kurshan. Computer-aided verification. IEEE Spectrum, **33**(6): 61–67, 1996.
10. E.M. Clarke, O. Grumberg and D. Peled. *Model Checking.* MIT Press, 1999.
11. P.R. D'Argenio, J.-P. Katoen, and E. Brinksma. Specification and analysis of soft real-time systems: Quantity and quality. In *Proc. IEEE Real-Time Systems Symp.*, pp. 104–114, IEEE CS Press, 1999.
12. M. Diefenbruch, J. Hintelmann, B. Müller-Clostermann. The QUEST-approach for the performance evemluation of SDL-systems. In *Proc. Form. Descritpion Techn. IX*, pp. 229-244, Chapman & Hall, 1996.
13. E.A. Emerson and C.-L. Lei. Temporal model checking under generalized fairness constraints. In *Proc. Hawaii Int. Conf. on System Sc.*, pp. 277–288, 1985.
14. A. Fantechi, S. Gnesi and G. Ristori. Model checking for action-based logics. *Form. Meth. in Sys. Design*, **4**: 187–203, 1994.
15. D. Ferrari. Considerations on the insularity of performance evaluation. *IEEE Tr. on Softw. Eng.*, **SE–12**(6): 678–683, 1986.
16. 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 *Performance*, LNCS 729: 121–146, 1993.
17. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Form. Asp. of Comp.*, **6**(5): 512–535, 1994.
18. C. Harvey. Performance engineering as integral part of system design. *Br. Telecom Techn. J.*, **4**:142–147, 1986.
19. M. Hennessy and R. Milner. Algebraic laws for nondeterminism and concurrency. *J. of the ACM*, **32**(1): 137–161, 1985.
20. H. Hermanns. *Interactive Markov Chains.* PhD thesis, University of Erlangen-Nürnberg, 1998.
21. H. Hermanns, U. Herzog and J.-P. Katoen. Process algebra for performance evaluation. *Th. Comp. Sci.*, 2001 (to appear).
22. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, **39**(1-4): 5–35, 2000.

23. H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic process algebras as a tool for performance and dependability modelling. In *Proc. IEEE Int. Comp. Perf. and Dependability Symp.*, pages 102–111. IEEE CS Press, 1995.
24. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. A Markov chain model checker. In *TACAS*, LNCS 1785:347–362, 2000.
25. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser and M. Siegle. Model checking stochastic process algebra. Tech. Rep. IMMD 7–2/00, University of Erlangen-Nürnberg, 2000.
26. H. Hermanns and M. Siegle. Bisimulation algorithms for stochastic process algebras and their BDD-based implementation. In *ARTS*, LNCS 1601:244–265, 1999.
27. U. Herzog and V. Mertsiotakis. Applying stochastic process algebras to failure modelling. In *Proc. Workshop on Process Algebra and Performance Modelling*, pp. 107–126, Univ. Erlangen-Nürnberg, 1994.
28. U. Herzog. Formal description, time and performance analysis. In *Entwurf und Betrieb Verteilter Systeme*, IFB 264:172–190, Springer, 1990.
29. J. Hillston. *A Compositional Approach to Performance Modelling.* Cambridge University Press, 1996.
30. D. Kozen. Results on the propositional mu-calculus. *Th. Comp. Sc.*, **27**: 333–354, 1983.
31. K.G. Larsen and A. Skou. Bisimulation through probabilistic testing. *Inf. and Comp.*, **94**(1): 1–28, 1992.
32. R. Mateescu and M. Sighireanu. Efficient on-the-fly model checking for regular alternation-free mu-calculus. *Proc. Workshop on Form. Meth. for Industrial Critical Sys.*, pp. 65-86. GMD/FOKUS, 2000.
33. R. De Nicola and F.W. Vaandrager. Action versus state based logics for transition systems. In *Semantics of Concurrency*, LNCS 469: 407–419, 1990.
34. R. De Nicola and F.W. Vaandrager. Three logics for branching bisimulation. *J. of the ACM*, **42**(2): 458–487, 1995.
35. W. Stewart. *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.

A Appendix: Sketch of the proof of Theorem 5

In order to verify Theorem 5(a), we prove that $(u, v) \in R$ implies $\forall \Phi.(u \models_{\mathcal{M}} \Phi$ iff $v \models_{\mathcal{M}} \Phi)$. We do so by structural induction on Φ. The only non-trivial cases are that Φ is of the form $\mathcal{S}_{\bowtie p}(\Psi)$, or $\mathcal{P}_{\bowtie p}(\varphi)$. In the former case, $\mathcal{S}_{\bowtie p}(\Psi)$, we use the induction hypothesis, the fact that Markovian bisimulation implies lumpability, and that lumpability ensures that steady-state probabilities can be obtained from the lumped quotient Markov chain [21]. In the latter case, $\mathcal{P}_{\bowtie p}(\varphi)$, we can apply Theorem 5(b), together with the induction hypothesis. So, only Theorem 5(b) remains to be verified. For this purpose, it is sufficient to show that $(u, v) \in R$ implies

$$\Pr\{\, \sigma \in Path(u) \;\mid\; \sigma \models \varphi \,\} \;=\; \Pr\{\, \sigma \in Path(v) \;\mid\; \sigma \models \varphi \,\}$$

We have to distinguish two cases, $\varphi = \Phi_1 \,{}_A\mathcal{U}^{<t}{}_B\, \Phi_2$ and $\varphi = \Phi_1 \,{}_A\mathcal{U}^{<t}\, \Phi_2$. Only the first of them is elaborated below. The other case proceeds in a similar, but simpler,

way. For $n \geqslant 1$ and $t > 0$ we define the set of paths $A^u_n(t)$ as

$$\begin{array}{rcl} A^u_n(t) & = & \{\, \sigma \in \mathit{Path}(u) \ \mid \ \sigma[n] \models \Phi_2 \\ & & \qquad\qquad\qquad\quad \wedge\ \forall\, 0 \leqslant i < n.\ \sigma[i] \models \Phi_1 \\ & & \qquad\qquad\qquad\quad \wedge\ \forall\, 0 \leqslant i < n-1.\ \sigma[i] \xrightarrow{A} \sigma[i+1] \\ & & \qquad\qquad\qquad\quad \wedge\ \sigma[n-1] \xrightarrow{B} \sigma[n] \\ & & \qquad\qquad\qquad\quad \wedge\ \sum_{i=0}^{n-1} \delta(\sigma, i) < t \,\} \end{array}$$

(observe the similarity to the semantics of $\Phi_1 \ {}_A\mathcal{U}^{<t}{}_B\, \Phi_2$) and the set of paths $B^u_i(t)$ for $i \geqslant 1$ by $B^u_1(t) = A^u_1(t)$, and $B^u_{n+1}(t) = A^u_{n+1}(t) \setminus \bigcup_{i=1}^n B^u_i(t)$. Intuitively, $A^u_n(t)$ is the set of paths starting in u and reaching a Φ_2-state within t time units in n steps, where the first $n-1$ steps are A-transitions and the last step is a B-transition. $B^u_n(t)$ denotes the subset of $A^u_n(t)$ consisting of paths that reach a Φ_2-state in n steps without performing an $A \cap B$-transition to a Φ_2-state in the previous steps.

Note that $B^u_i(t)$, $B^u_j(t)$ are pairwise disjoint (for $i \neq j$). By exploiting the fact that $\{\, \sigma \in \mathit{Path}(u) \ \mid \ \sigma \models \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B\, \Phi_2 \,\} \ = \ \bigcup_{n \geqslant 1} B^u_n(t)$, we obtain:

$$\Pr\{\, \sigma \in \mathit{Path}(u) \ \mid \ \sigma \models \Phi_1 \ {}_A\mathcal{U}^{<t}{}_B\, \Phi_2 \,\} \ = \ \sum_{i=1}^{\infty} \Pr\{\, \sigma \in B^u_i(t) \,\}.$$

Hence, it is sufficient to show that for arbitrary $t > 0$,

$$\sum_{i=1}^{\infty} \Pr\{\, \sigma \in B^u_i(t) \,\} \ = \ \sum_{i=1}^{\infty} \Pr\{\, \sigma \in B^v_i(t) \,\}.$$

We fix some $t > 0$, and prove the above by showing the stronger property that for all positive n, $\Pr\{\, \sigma \in B^u_n(t) \,\} \ = \ \Pr\{\, \sigma \in B^v_n(t) \,\}$. This proof proceeds by induction on n, the length of the paths in $B^u_n(t)$ and $B^v_n(t)$. So, we perform a nested induction, the (inner) induction on n is nested in the (outer) induction on the structure of the formula Φ.

In the base case $n = 1$ of the inner induction, let us first assume $u \not\models \Phi_1$. But then $v \not\models \Phi_1$ (by the outer induction hypothesis) and hence $\Pr\{\, \sigma \in B^u_1(t) \,\} = 0 = \Pr\{\, \sigma \in B^v_1(t) \,\}$. If, conversely, $u \models \Phi_1$, we obtain $v \models \Phi_1$ by the outer induction hypothesis, and therefore

$$\begin{array}{lr} \Pr\{\, \sigma \in B^u_1(t) \,\} & = \\ \sum_{w \models \Phi_2} \mathbf{P}_B(u, w) \cdot \left(1 - e^{-\mathbf{E}(u) \cdot t}\right) & = \\ \sum_{C \in \mathcal{M} \setminus R,\ C \models \Phi_2} \sum_{w \in C} \mathbf{P}_B(u, w) \cdot \left(1 - e^{-\mathbf{E}(u) \cdot t}\right) & \overset{(*)}{=} \\ \sum_{C \in \mathcal{M} \setminus R,\ C \models \Phi_2} \sum_{w \in C} \mathbf{P}_B(v, w) \cdot \left(1 - e^{-\mathbf{E}(v) \cdot t}\right) & = \\ \sum_{w \models \Phi_2} \mathbf{P}_B(v, w) \cdot \left(1 - e^{-\mathbf{E}(v) \cdot t}\right) & = \\ \Pr\{\, \sigma \in B^v_1(t) \,\} & \end{array}$$

Here, $C \models \Phi_2$ denotes that all states in the equivalence class C satisfy Φ_2, which we can assume by the outer induction hypothesis. The transformation labelled $(*)$ uses that $(u, v) \in R$ implies $\sum_{w \in C} \mathbf{P}_A(u, w) = \sum_{w \in C} \mathbf{P}_A(v, w)$, since C is the class of a Markovian bisimulation.

To complete the inner induction we now assume that for arbitrary $n > 1$ we have that $\Pr\{\, \sigma \in B^u_n(t) \,\} = \Pr\{\, \sigma \in B^v_n(t) \,\}$, and aim to show that this also holds for $n + 1$. The case $u \not\models \Phi_1$ proceeds as above. The remaining case, $u \models \Phi_1$, leads to the following transformation, using the same arguments as above.

$$\begin{array}{ll}
\Pr\{\,\sigma \in B^u_{n+1}(t)\,\} & = \\
\int_0^t e^{-\mathbf{E}(u)\cdot x} \cdot \sum_{w \models \Phi_1} \mathbf{R}_A(u,w)\cdot \Pr\{\,\sigma \in B^w_n(t-x)\,\}\; dx & = \\
\int_0^t e^{-\mathbf{E}(u)\cdot x} \cdot \sum_{C \in \mathcal{M}\setminus R,\; C \models \Phi_1} \sum_{w \in C} \mathbf{R}_A(u,w)\cdot \Pr\{\,\sigma \in B^w_n(t-x)\,\}\; dx & \overset{(*)}{=} \\
\int_0^t e^{-\mathbf{E}(v)\cdot x} \cdot \sum_{C \in \mathcal{M}\setminus R,\; C \models \Phi_1} \sum_{w \in C} \mathbf{R}_A(v,w)\cdot \Pr\{\,\sigma \in B^w_n(t-x)\,\}\; dx & = \\
\int_0^t e^{-\mathbf{E}(v)\cdot x} \cdot \sum_{w \models \Phi_1} \mathbf{R}_A(v,w)\cdot \Pr\{\,\sigma \in B^w_n(t-x)\,\}\; dx & = \\
\Pr\{\,\sigma \in B^v_{n+1}(t)\,\} &
\end{array}$$

This completes the proof sketch, details can be found in [25].

This article was processed using the LaTeX macro package with LLNCS style