Implementing a Model Checker for Performability Behaviour*

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

We describe a novel model checking algorithm for analysing the behaviour of stochastic systems with respect to their performability. Systems are modelled as action-labelled CTMCs, and the properties to be verified are specified with the help of the action-based temporal logic **aCSL**. The technique is currently being implemented in our freely available prototype tool ETMCC.

1 Introduction

Performance and dependability analysis are crucial tasks during the design cycle of parallel and distributed IT systems. The common approach to performability modelling is to use some high-level formalism for model specification (e.g. SPN [1] or SPA [6]), to derive a flat CTMC from the model and to analyse the CTMC with numerical methods. The definition of the aim of analysis, i.e. the specification of the measures of interest, has not been formalised in the past. Recently, temporal logics have been proposed for the specification of performability properties, and it has been shown how such properties can be verified with the help of model checking techniques [3].

Stochastic process algebra support compositional, *behaviour-oriented* performance and reliability modelling. In a nutshell, a process algebra model specifies sequences of actions which a process may perform. With *stochastic* process algebra, actions are associated with stochastic delays. Their behaviour-oriented view supports composition, and is in stark contrast with the state-oriented view inherent in Petri nets or queueing networks, where tokens or jobs flow through some static structure, and where a state is naturally identified with a snapshot of the populations residing at different locations of this structure.

Ironically, 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 [12]. 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.

*supported by the DFG-NWO bilateral cooperation program (VOSS).

†supported by the Netherlands Organisation for Scientific Research (NWO).

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

Similar to [4], we argue that a behaviour-oriented approach to the specification and analysis of performability properties is more appropriate for SPAs. To this aim, we have developed an action-based, branching-time stochastic logic, called **aCSL** (action-based Continuous Stochastic Logic) [9] that is inspired by the logic **CSL** [3]. The logic **aCSL** provides means to reason about continuous-time Markov chains but, opposed to **CSL**, it is not state-oriented. Its basic constructors are sets of actions, instead of atomic state propositions. **aCSL** provides means to specify temporal and timed properties, and means to quantify their probability. It allows one to specify properties such as *"Once action SEND has occured, there is at least a 30% chance that action ACK will be observed within at most* 4 *time units"*.

The analysis of such properties can be entirely automated. We have implemented the **CSL** model checker ETMCC [8] which is currently being extended so that it can also check properties specified in **aCSL**. This paper serves two purposes. On the one hand (Sec. 3) we illustrate the use of a behaviour-oriented view by quoting from a case study (from [9]). Furthermore we discuss how known model checking algorithms need to be adapted to the behaviour-oriented setting (Sec. 4).

2 An action-based continuous stochastic logic

Stochastic process algebras can be used to generate CTMCs in a compositional manner. More precisely, they generate action-labelled CTMCs, such as the one shown below in Fig. 2, i.e. a directed graph whose transitions are labelled with tuples from the set $Act \times I\!R$, i.e. tuples of the form (action, rate).

We briefly introduce the logic **aCSL** which allows one to specify performability properties. Let $p \in [0,1]$ and $\bowtie \in \{ \leq, <, \geq, > \}$. The state-formulas Φ of **aCSL** are defined by the grammar

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

where φ stands for a path-formula as defined below. A state formula is either the constant $true$, the conjunction of two state formulas, the negation of a state formula, a steady-state formula ($\mathcal{S}$) or a probabilistically quantified path formula ($\mathcal{P}$). The other Boolean connectives such as $\vee$ and $\Rightarrow$ are derived in the obvious way. $\mathcal{S}_{\bowtie p}(\Phi)$ 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 $\bowtie p$.

Let $A, B \subseteq Act$ and $t \in I\!R_{>0} \cup \{ \infty \}$. Path formulas φ are defined by the grammar

$$\varphi ::= \Phi \;{}_{A}\mathcal{U}^{<t}\; \Phi \;\Big|\; \Phi \;{}_{A}\mathcal{U}_{B}^{<t}\; \Phi$$

where $\mathcal{U}$ is to be read as "until". The path-formula $\Phi_1 \;{}_{A}\mathcal{U}^{<t}\; \Phi_2$ is satisfied by a path that visits only Φ_1-states before it eventually reaches a Φ_2-state, while taking only A-transitions; in addition, the time until reaching the Φ_2-state has to be less than t time units. The formula $\Phi_1 \;{}_{A}\mathcal{U}_{B}^{<t}\; \Phi_2$ has a related meaning, but requires in addition that (i) a transition to a Φ_2-state is actually made and that (ii) this transition is labelled by an action in B. Further operators can be derived, such as the operators "possibly" ($\langle . \rangle$) and "necessarily" ($[.]$) which are defined as follows:

$$\langle A \rangle\, \Phi \;=\; \mathcal{P}_{>0}\,(true \;{}_{\emptyset}\mathcal{U}_{A}^{<\infty}\; \Phi) \qquad \text{and} \qquad [A]\, \Phi \;=\; \neg \langle A \rangle\, \neg \Phi.$$

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.

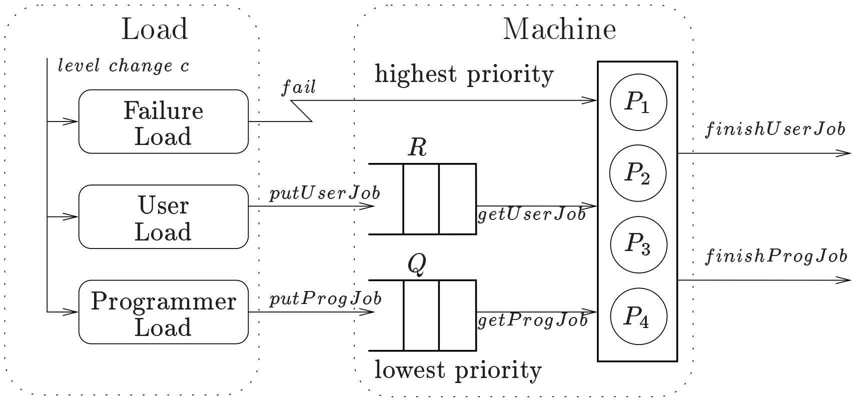


Figure 1: Multiprocessor mainframe model structure

3 Application example: Multiprocessor mainframe with failures

To illustrate the power of the behaviour-oriented logics approach, we consider a multiprocessor mainframe, see Fig. 1, which was first introduced in [10] and has since then served as a standard SPA example, see e.g. [7, 5]. 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.

Component *Load* represents the system load caused by the database users, the programmers and the failures. The three different system load components are modelled as Markov modulated Poisson processes, such that the intensity of the load changes between different levels. The mainframe itself is modelled by the *Machine* component which 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 lowest priority, while failures have highest priority. 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*). We refer to [10] for details of the SPA specification.

In order to illustrate the use of **aCSL** for specifying performance and reliability properties, we specify some properties of interest for the multiprocessor mainframe model. For each property, a description in plain English, its **aCSL** formulation and some explanation are given. For $A \subseteq \mathit{Act}$ we let $\overline{A}$ denote $\mathit{Act} \setminus A$, and we define the set of actions $Fin := \{finishUserJob, finishProgJob\}$.

Φ_1: In steady state, the probability that at least two processors are occupied by user jobs is greater than 0.002.

$$\mathcal{S}_{>0.002}\left(\langle finishUserJob\rangle\ \langle finishUserJob\rangle\ \mathit{true}\right)$$

Thus, states where at least two processors are occupied by user jobs are characterised by the ability to perform action $finishUserJob$ twice.

Φ_2: There is at least a 30% chance that some job will be finished within at most 4 time units.

$$\mathcal{P}_{\geq 0.3}\left(true\ {}_{\overline{Fin}}\mathcal{U}^{<4}_{Fin} true\right)$$

This formula characterises paths on which an action from set Fin occurs within 4 time units, but may be preceded by an arbitrary number of actions from $\overline{Fin}$.

Φ_3: In steady state, the probability of the system being unavailable (i.e. waiting for repair) is at most 0.05.

$$\mathcal{S}_{\leq 0.05}\left(\mathcal{P}_{>0}\left(true\ {}_{\overline{\{fail\}}}\mathcal{U}_{\{repair\}}\, true\right)\right)$$

States where the system is unavailable are thus characterised by the fact that action $repair$ will eventually occur without action $fail$ having to occur first.

Φ_4: After a system failure, there is a chance of more than 90% that it will come up again within the next 5 time units.

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

So, once action $fail$ has occured, with probability greater than 0.9 action $repair$ must occur within 5 time units, but arbitrary other actions may occur in between.

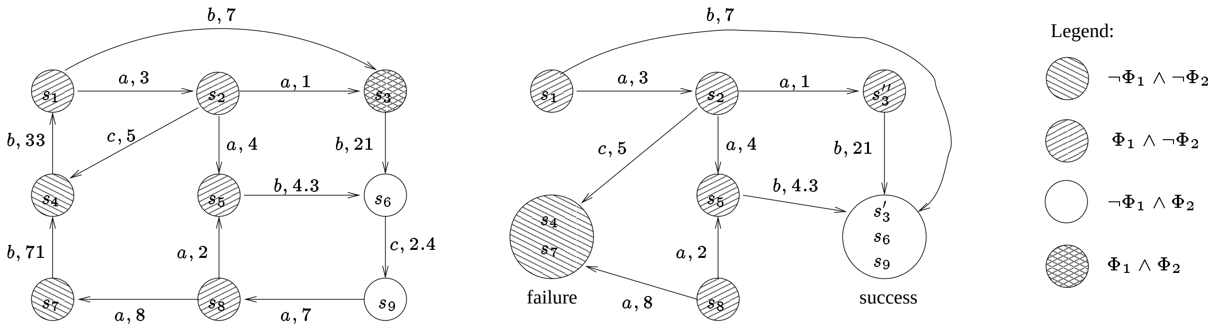
4 Implementing a model checker for aCSL

The expressive power of the logic **aCSL** is paired with the possibility to fully automate the model checking of **aCSL** properties (on finite-state action-labelled CTMCs). In this way, the analysis of involved performability properties (such as Φ_4) becomes push-button technology. We employed our model checker ETMCC to check the above[1] and other properties on different instances of the multiprocessor mainframe model. We varied the size of the queues, which led to different state spaces of up to 110946 states and 761989 transitions. The results are presented in [9]. As an example, for the largest model, checking Φ_1 took 18.814 seconds, while checking Φ_4 took 11.603 seconds on a standard SUN workstation.

We now briefly sketch the different algorithmic ingredients needed to check **aCSL**. They are similar to the **CSL** setting, but the behaviour-oriented view requires some distinct considerations. These differences are the focus of the discussion that follows. For a given formula Φ, a parse tree is generated from the formula first. Then the model checker recursively evaluates the parse tree, and subformulas are checked, starting at the leaves of the tree and finishing at the root. The subformulas are state formulas complying to the above grammar. Checking of the Boolean connectives is standard. Checking of steady-state properties requires the solving of a linear system of equations for which the tool ETMCC employs the iterative schemes of either Jacobi or Gauss-Seidel. As a preprocessing step for steady-state calculation, the strongly connected components of the CTMC are determined with the help of a graph analysis algorithm. Checking time-bounded path formulas is the most complicated. In the general case, a system of Volterra integral equations has to be solved, for which the tool ETMCC offers two solution methods: (i) numerical integration, working on discretised representations of distribution functions, and (ii) transient analysis with the help of the uniformisation method, working on a modified CTMC.

Due to limited space, we cannot present details of the model checking algorithms here. Instead, we now discuss the particularities of the behaviour-oriented approach by means of a small example. It illustrates the various steps that are carried out when checking an until-formula, decorated with sets of actions. We discuss the checking of the **aCSL** formula $\Phi = \mathcal{P}_{>0.5}\left(\Phi_1\ {}_{\{a\}}\mathcal{U}^{<t}_{\{b\}}\ \Phi_2\right)$ for both the untimed case (i.e. $t = \infty$) and the time-bounded

[1] Since ETMCC does not yet support full **aCSL**, some properties were translated manually into **CSL**.



(a) original

(b) modified

Figure 2: Example action-labelled CTMC

case (i.e. for finite $t < \infty$). For checking this formula on the model shown in Fig. 2, the following steps are performed mechanically:

1. Determine the set of states $\mathcal{E}$ from which there originates a path that functionally satisfies $\Phi_1 \ {}_{\{a\}}\mathcal{U}_{\{b\}} \ \Phi_2$. The set $\mathcal{E}$ is initialised as $\emptyset$. First, Φ_1-states from which there is a b-transition to a Φ_2-state are added to $\mathcal{E}$ (in the example s_1, s_3, s_5). Then those Φ_1-states are added to $\mathcal{E}$ which possess an a-transition to a state already in $\mathcal{E}$ (i.e. s_2 and s_8 are added to $\mathcal{E}$). So finally $\mathcal{E} = \{s_1, s_2, s_3, s_5, s_8\}$.

2. For the untimed case, determine the set of states $\mathcal{A} \subseteq \mathcal{E}$ whose outgoing paths *all* satisfy $\Phi_1 \ {}_{\{a\}}\mathcal{U}_{\{b\}} \ \Phi_2$. Set $\mathcal{A}$ is computed by removing states from $\mathcal{E}$. In the example, s_2 is removed first, since it can can make a c-transition to s_4. Afterwards, s_1 is removed, since it can move to s_2 which is no longer in $\mathcal{A}$. Furthermore, s_8 is removed since it can make an a-transition to s_7 which is not a Φ_1-state. So finally $\mathcal{A} = \{s_3, s_5\}$.

   Next, for all states, we determine the probability with which the state satisfies $\Phi_1 \ {}_{\{a\}}\mathcal{U}^{<\infty}_{\{b\}} \ \Phi_2$, i.e. regardless of the time that passes until reaching the Φ_2-state. States not in $\mathcal{E}$ have probability 0. States from $\mathcal{A}$ have probability 1.0. In general, the probabilities for the states from $\mathcal{E} \setminus \mathcal{A}$ (in the example, $\mathcal{E} \setminus \mathcal{A} = \{s_1, s_2, s_8\}$) are determined by solving a linear system of equations for which the tool ETMCC again uses Jacobi or Gauss-Seidel. In the example, the probability for state s_2 is simply $1/10 + 4/10 = 0.5$, the probability for state s_8 is $2/10 = 0.2$, and the probability for state s_1 is $7/10$ plus $3/10$ times the probability which was just computed for s_2, i.e. $7/10 + 3/10 * 0.5 = 0.85$.

3. For the time-bounded case, the algorithm essentially follows the strategy of [2], but the action-based view requires some modifications. When determining the set of states that satisfy the original formula $\Phi = \mathcal{P}_{>0.5}\left(\Phi_1 \ {}_{\{a\}}\mathcal{U}^{<t}_{\{b\}} \ \Phi_2\right)$ for finite $t < \infty$, it is obvious that states not in $\mathcal{E}$ cannot satisfy Φ, so no numerical calculations will have to be performed for states s_4, s_6, s_7, s_9. The CTMC is now modified, such that states which satisfy $\neg\Phi_1 \wedge \neg\Phi_2$ (that is states s_4, s_7) or $\neg\Phi_1 \wedge \Phi_2$ (that is states s_6, s_9) are made absorbing, and states which satisfy $\Phi_1 \wedge \Phi_2$ are duplicated, such that if reached by a b-transition they are made absorbing, while if reached by an a-transition they retain their original outgoing transitions (in the example, s_3 would be duplicated, which yields s_3' and s_3''). All absorbing states which satisfy Φ_2 are collected in a "success" macro state, while those absorbing states which satisfy neither Φ_1 nor Φ_2 are collected in a "failure" macro state. Then transient analysis (via uniformisation) is performed on this modified model. The probability with which a state s satisfies

$\Phi_1 \ {}_{\{a\}}\mathcal{U}^{<t}_{\{b\}} \ \Phi_2$ is calculated as the probability of being in the "success" macro state at time t, provided that the system started in state s at time 0. A clever cumulation of intermediate numerical results [11] can be adopted to our case such that these time-dependent probabilites are calculated for all states by a single uniformisation. This cumulation is already implemented in ETMCC.

5 Conclusion

We presented a new method for the specification of performability measures with the help of the action-based stochastic temporal logic **aCSL**. This logic is well suited for specifying requirements of SPA models. Their analysis via model checking can be fully automated. We provided some example property specifications and discussed by example the particular algorithmic steps needed to verify properties of type time-bounded until decorated with action sets. These model checking algorithms are currently being implemented in our tool ETMCC. We are also working on extensions of the logic in order to be able to specify an even broader class of performability measures. For academic purposes the current version of ETMCC can be downloaded free of charge via `http://www7.informatik.uni-erlangen.de/etmcc/`. The extension to **aCSL** will be made available via this web-page.

References

[1] M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with generalized stochastic Petri nets.* Wiley, 1995.

[2] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model Checking Continuous Time Markov Chains by Transient Analysis. In *CAV'2000*, pages 358–372. Springer LNCS 1855, 2000.

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

[4] G. Clark, S. Gilmore, and J. Hillston. Specifying performance measures for PEPA. In J.-P. Katoen, editor, *5th Int. AMAST Workshop on Real-Time and Probabilistic Systems (ARTS'99)*, pages 211–227. Springer LNCS 1601, 1999.

[5] P.R. D'Argenio, J.P. Katoen, and E. Brinksma. General purpose discrete–event simulation using SPADES. In C. Priami, editor, *6th Int. Workshop on Process Algebras and Performance Modelling*, pages 85–102. Universita di Verona, 1998.

[6] H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Th. Comp. Sci.*, 2000. to appear.

[7] H. Hermanns, U. Herzog, and V. Mertsiotakis. Stochastic Process Algebras as a Tool for Performance and Dependability Modelling. In *Proc. of IEEE International Computer Performance and Dependability Symposium*, pages 102–111, Erlangen, April 1995. IEEE Computer Society Press.

[8] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. A Markov Chain Model Checker. In S. Graf and M. Schwartzbach, editors, *TACAS'2000*, pages 347–362. Springer LNCS 1785, 2000.

[9] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In W. Grieskamp, T. Santen, and B. Stoddart, editors, *2nd Int. Conference on Integrated Formal Methods*, pages 420–439, Dagstuhl, November 2000. Springer LNCS 1945.

[10] U. Herzog and V. Mertsiotakis. Applying Stochastic Process Algebras to Failure Modelling. In *Proc. of the 2nd Workshop on Process Algebras and Performance Modelling*, pages 107–126. Arbeitsberichte des IMMD, Universität Erlangen-Nürnberg, July 1994.

[11] J.-P. Katoen, M. Kwiatkowska, G. Norman, and D. Parker. Faster and Symbolic CTMC Model Checking. In *PAPM/PROBMIV'01.* Springer LNCS, 2001. to appear.

[12] W.J. Stewart. *Introduction to the numerical solution of Markov chains.* Princeton University Press, 1994.