A Stochastic Extension of the Logic PDL

Matthias Kuntz
Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik 7
Martensstrasse 3, D-91058 Erlangen
Email: mskuntz@informatik.uni-erlangen.de

Markus Siegle
Friedrich-Alexander-Universität Erlangen-Nürnberg
Institut für Informatik 7
Martensstrasse 3, D-91058 Erlangen
Email: siegle@informatik.uni-erlangen.de

Abstract— **We present stochastic PDL (SPDL), a stochastic extension of the modal logic PDL (propositional dynamic logic), which is interpreted over labelled continuous time Markov chains (CTMC). This logic allows one to specify complex path-based performability measures and check their validity automatically. We define the syntax and semantics of SPDL. In this logic, computation paths can be characterised by regular expressions, also called programs, where the executability of a regular expression may depend on the validity of test formulae. For model checking SPDL path formulae, we transform programs into a variant of deterministic finite automata, and then build the product automaton between this automaton and the labelled CTMC. The paper contains a small example that illustrates the model checking procedure**[1]**.**

I. Motivation and Introduction

Distributed, concurrent hard- and software systems have become part of our daily life and it becomes more and more important to assert that they are working correctly and that they meet high performance and dependability requirements.

In order to carry out performance, dependability and reliability analysis it is necessary to have both a model and a number of measures of interest, such as utilisation, mean number of jobs, mean time to failure, etc.. Roughly spoken, the model is derived in two steps: Firstly, some specification method such as stochastic Petri nets, stochastic process algebras, queueing networks, etc. is employed to obtain a high-level specification of the system that is to be analysed. Secondly, from this high-level specification the low-level representation is obtained. This low-level representation is normally a continuous time Markov chain (CTMC).

In the realm of functional verification, temporal logics such as CTL provide powerful means to specify complex requirements that a system has to satisfy. In the recent years big efforts have been made to provide similar means for the specification of system properties in the area of performance analysis. One result of these efforts is the logic CSL (continuous stochastic logic) introduced by [1] and extended in [2] with an operator to reason about steady state probabilities. CSL allows the specification of certain types of performability measures (cf. [3]) but the specification of these measures is completely state-oriented.

A very important branch of modelling formalisms is that of stochastic process algebras (SPA), which is action-oriented. In a nutshell this means a process is specified by the sequence of actions that it can perform. In this context, states constitute only an auxiliary means within the semantic model of SPA-processes. In contrast, using CSL, the determination of the measures of interest is state-oriented. To avoid this change of views, i.e. action- vs. state-oriented, in [4] an action-based variant of CSL, aCSL, has been proposed. In [5] it was shown how to employ this logic for performability modelling. In aCSL the requirements are completely action-oriented. As aCSL has only limited capabilities to characterise paths, it was extended in [6] to aCSL+, where paths can be characterised by regular expressions, also called programs.

In this extended abstract we propose a stochastic extension of the modal logic PDL, SPDL, which extends aCSL+. In SPDL paths can also be characterised by programs, but in addition it is possible to express that a program is executable only if the current state satisfies a given state property. This makes it possible to combine in an easy way state- and action-oriented behaviour.

SPDL is thus well suited for performability modelling, since it provides ample means to characterise paths, thereby allowing the modeller to obtain a high level of confidence in the performance and dependability of the system at hand.

This paper is organised as follows: At first we define the syntax and semantics of the logic SPDL (Sec. II). In Sec. III we illustrate the possibilities of SPDL to specify performability measures by means of an example system. Sec. IV introduces the model checking procedure of SPDL-path formulae, which can be reduced to standard transient analysis, by example. The paper concludes with a summary and an outlook on future work.

II. Syntax and Semantics of SPDL

This section introduces the syntax of SPDL formulae and programs and describes in an informal way their semantics.

A. Syntax of SPDL

Generally spoken, SPDL consists of the following ingredients: propositional logic, modal logic, probability theory, and algebra of regular expressions. SPDL expressions can be formed using as follows: Let $p \in [0,1]$ and $q \in \mathsf{AP}$, where AP is the set of atomic propositions, i.e. elementary state formulae, and let $\bowtie \in \{\leq, <, \geq, >\}$. The state formulae Φ of SPDL are

[1]©Matthias Kuntz and Markus Siegle

defined by the following grammar:

$$\Phi \quad := \quad q \big| \Phi \vee \Phi \big| \neg\Phi \big| \mathcal{S}_{\bowtie p}(\Phi) \big| \mathcal{P}_{\bowtie p}(\varphi) \big| (\Phi)$$

Thus, a state formula is either an atomic proposition, the disjunction of two state formulae, the negation of a state formula, a steady state formula ($\mathcal{S}$), or a probabilistically quantified path formula ($\mathcal{P}$). Path formulae φ are defined by:

$$\varphi \quad := \quad \Phi[\pi]^I \Phi$$

where I is the closed interval $[t, t']$. π is a program as defined in the sequel.

Let Act be a set of atomic programs, which we may also call actions, and TEST be a set of state formulae. A program π is defined by the following grammar:

$$\begin{aligned} \pi \quad &:= \quad \epsilon \big| \pi;\pi \big| \pi \cup \pi \big| \Phi?;\pi \big| \pi_1 \big| (\pi) \\ \pi_1 \quad &:= \quad a \big| \pi_1;\pi_1 \big| \pi_1 \cup \pi_1 \big| \pi_1^* \big| \Phi?;\pi_1 \big| (\pi_1) \end{aligned}$$

where $a \in \mathsf{Act}$ and $\Phi \in \mathsf{TEST}$. ϵ is the empty program, $\pi;\pi$ is the sequential execution of two programs, $\pi \cup \pi$ is the non-deterministic choice between two programs. $\Phi?;\pi$ checks whether Φ holds in the actual state, if it does, execute π, otherwise fail. Finally, π_1^* means, execute π_1 an arbitrarily chosen finite number of times, including zero times.

$\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability, i.e. the probability to reside in a particular set of states on the long run, given an initial state s, satisfies the boundary as given by $\bowtie p$. $\mathcal{P}_{\bowtie p}(\varphi)$ asserts that the probability measure of the set of paths that satisfy φ is within the bounds as given by $\bowtie p$.

B. The Semantic Model

SPDL is interpreted over an action- and state-labelled CTMC (ASMC) $\mathcal{M}$, which is a quadruple (S, Act, L, R), where

- S: finite set of states
- Act: set of action names
- L: state labelling function: $S \to 2^{\mathsf{AP}}$
- R: state transition relation : $R \subseteq S \times (A \times \mathbb{R}_{>0}) \times S$

The semantics of SPDL-state formulae is defined the standard way, details can be found in [7]. We only describe informally the semantics of SPDL-path formulae as their semantics is very different from that of CSL: A path σ of model $\mathcal{M}$ satisfies path formula $\varphi := \Phi[\pi]^I \Psi$ iff:

- a Ψ-state on path σ is reached after the passage of t'' time units, where t'' is within I.
- all preceeding states satisfy Φ
- the actions offered by σ correspond to the program π.
- all test formulae occuring in π are satisfied in the corresponding states of σ.

C. Derived Temporal Operators

The only temporal operator presented so far is $[\pi]^I$. The usual until-operator of CSL (cf. [8]) can be expressed as follows:

$$\Phi \mathsf{U}^I \Psi := \Phi[\mathsf{Act}^*]^I \Psi$$

Other operators, like 'X' ('next') and 'F' ('finally') can also be derived. Details can be found in [7].

III. Example: System Model and Requirements

In order to illustrate our approach, specifying and checking performability measures using the logic SPDL, we consider an example.

A. The System Model

The model in Fig. 1 represents a system that receives four data packets and processes them together. This behaviour is repeated indefinitely.

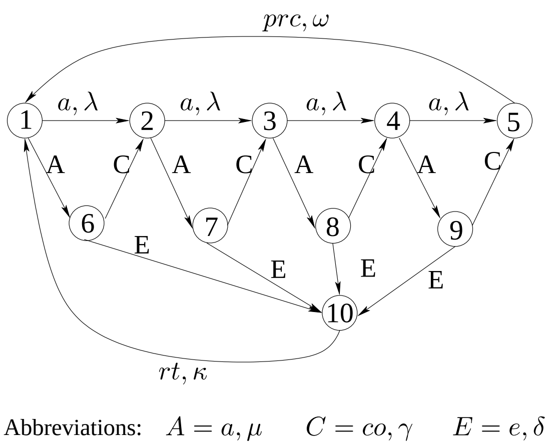

Fig. 1. System model – a 4 place buffer with erroneous arrivals

In more detail, an arrival is modelled by action a. Arrival of a data packet can be error-free (arrival rate λ) or erroneous (arrival rate μ). An erroneous data packet can be corrected (co, γ), or cannot be corrected (e, δ). If it cannot be corrected, the buffer is emptied and all data packets have to be retransmitted (rt, κ). If all data packets are error-free or correctable, then the received data can be processed (prc, ω) and the system awaits new data.

We must also provide the state labellings, i.e. atomic formulae that are valid in the states of the system model. The example system has 10 states, indexed 1 to 10.

- $L(s_1) = \{\mathsf{empty}\}$
- $L(s_2) = L(s_3) = L(s_4) = \emptyset$
- $L(s_5) = \{\mathsf{full}\}$
- $L(s_6) = L(s_7) = L(s_8) = L(s_9) = L(s_{10}) = \{\mathsf{error}\}$

A state satisfies the negations of the formulae that are not valid in it, e.g. states 2 - 4 satisfy $\{\neg\mathsf{empty}, \neg\mathsf{full}, \neg\mathsf{error}\}$.

B. Performability Measures

Now, we will give some example requirements:

1) $\Phi_1 \ := \ \mathcal{P}_{\geq 0.9}(\neg\mathsf{full}[a^*; e; rt; a^* \cup a^*]^{[0,5]}\mathsf{full})$: Is the probability to receive all data packets without error or with at most one non-correctable error within 5 time units at least 0.9?
2) $\Phi_2 := \mathcal{P}_{>0}(\mathsf{true}[a]^{[0,\infty)}\mathsf{full})$: Is the probability to reach a state in which the buffer is full with a single arrival greater than zero? Φ_2 characterises state 4 as this is the only state from which it is possible to reach a state satisfying 'full' by a single arrival action.

3) $\Phi_3 := \mathcal{P}_{\leq 0.1}(\mathsf{true}[a^*; (\Phi_2?; a; co)]^{[0,7.3]}\mathsf{full})$: Is the probability that the buffer is full after at most 7.3 time units and that the 4th packet contains a correctable error and that all preceeding packets are error free, at most ten percent?
4) $\Phi_4 := \mathcal{P}_{\geq 0.85}(\mathsf{true}[(a \cup a; co)^*]^{[0,10]}\mathsf{full})$: Is the probability to reach state 5 within 10 time units, provided no packet contains incorrectable errors, at least 85 %?

IV. Model Checking SPDL

In this section we illustrate by example how we can model check SPDL path formulae. A thorough account of how to model check SPDL can be found in [7].

A. General Aproach

Fig. 2 shows the general approach to model check SPDL path formulae. We assume that a system model $\mathcal{M}$ and a

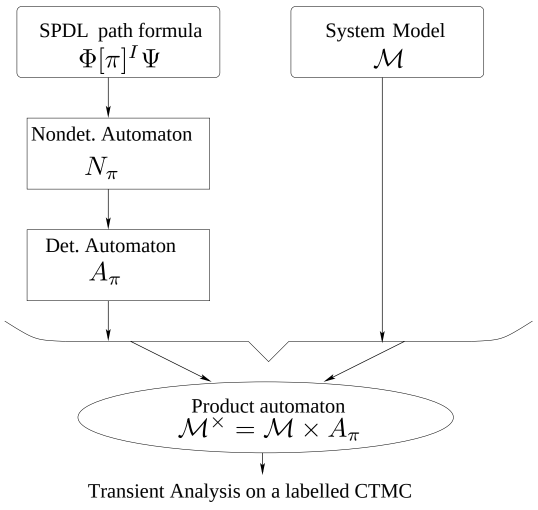

Fig. 2. Model checking SPDL path formulae – general approach

SPDL requirement $\Phi[\pi]^I\Psi$ are given. From π we derive a non-deterministic automaton, N_π, that is transformed into a deterministic one, A_π. From A_π and $\mathcal{M}$ a product automaton $\mathcal{M}^\times$ is built, which in fact is a CTMC whose transitions rates are taken from $\mathcal{M}$. Finally, on $\mathcal{M}^\times$ we can perform transient analysis to check whether the model satifies its requirement or not. Transient analysis is done by the well-known method of uniformisation.

B. Example

Consider the example system $\mathcal{M}$, from Fig. 1 and the requirement $\varphi := \mathsf{true}[a^*; (\Phi_2?; a; co)]^{[0,7.3]}\mathsf{full}$. We want to check whether $\mathcal{M}$ satisfies φ, especially for state 1. Corresponding to Fig. 2 we derive from $a^*; (\Phi_2?; a; co)$ a non-deterministic automaton N_π (cf. Fig. 3)[2]. The test Φ_2 forms together with a a single transition. Now, we have to transform N_π into a determininistic automaton A_π (cf. Fig. 4) which possesses states from the powerset of the state space of N_π. In

[2]Grey-shaded states indicate the accepting end states.

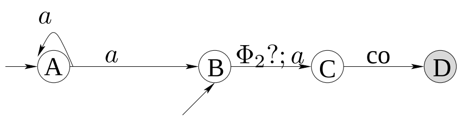

Fig. 3. Non-deterministic automaton N_π for $a^*; (\Phi_2?; a; co)$

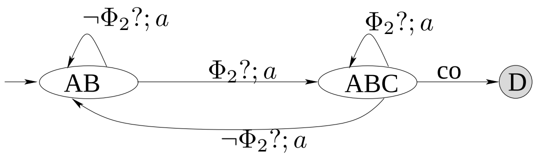

Fig. 4. Deterministic automaton A_π for $a^*; (\Phi_2?; a; co)$

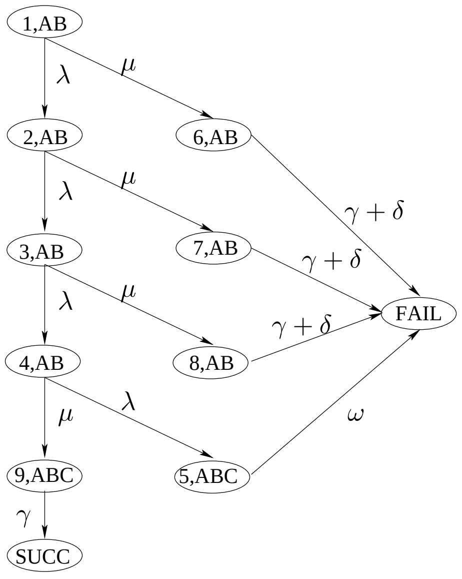

Fig. 5. Product automaton $\mathcal{M}^\times = \mathcal{M} \times A_\pi$ for checking validity of Φ_3

Fig. 4 we see that the labels of the transitions emanating from states AB and ABC are labelled with $\neg\Phi_2?; a$ resp. $\Phi_2?; a$, i.e. the tests are disjoint. We discuss this issue further below. Fig. 5 shows, how the product automaton $\mathcal{M}^\times := \mathcal{M} \times A_\pi$ is generated. The state labelled with $SUCC$ is an absorbing goal state in which the path formula functionally holds, the state labelled with $FAIL$ is an absorbing error state to which all transitions are redirected that lead to states that render the path formula unsatisfiable. The model checking itself, i.e. the check whether $\mathcal{M}$ satisfies the path formula, would be done by transient analysis.

Now, we will briefly explain why A_π possesses transitions labelled $\neg\Phi_2?; a$ and $\Phi_2?; a$, instead of a and $\Phi_2?; a$: For model checking to yield correct result it is necessary to preserve the stochastic behaviour of $\mathcal{M}$. For an artificial example, we will see that this requirement might be violated (cf. Fig. 6). If we assume that in state 4 of $\mathcal{M}$ the formula Ψ of an imaginary path formula $\varphi_{im} := \Phi[(\Theta?; a)^*; \Xi?; a]^I\Psi$ holds, then the lower path of $\mathcal{M}^\times_{N_\pi}$, the Ξ-path leads to the $FAIL$-state, as once in state B of the automaton for the program π of φ_{im} no transition is possible. In contrast, the

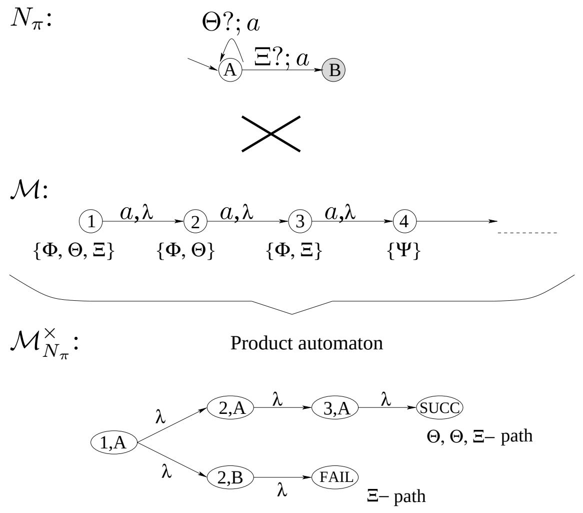

Fig. 6. Incorrect product automaton for φ_{im}

upper path, the (Θ,Θ,Ξ)-path leads to state 4 of $\mathcal{M}$ and to an accepting state in the automaton for π, therefore this path is a satisfying path. But, as both paths can be taken in state $(1,A)$ we doubled the rate λ, which modifies the stochastic behaviour of $\mathcal{M}$ and would therefore lead to wrong results during model checking. In Fig. 7, we find the correct product automaton by using the same kind of transition labelling procedure as in Fig. 4. We see that the product automaton in Fig. 7 preserves the branching and therefore the stochastic behaviour of $\mathcal{M}$. In contrast to N_π in A_π no two transitions are activated at the same time. The tests in A_π are disjoint, therefore when $\mathcal{M}$ is in state 1 and A_π is in state A only the transition with labelling $(\Theta\wedge\Xi)?;a$ to state AB can be taken, leading in the product automaton $\mathcal{M}^\times_{A_\pi}$ to the unique successor state $(2,AB)$.

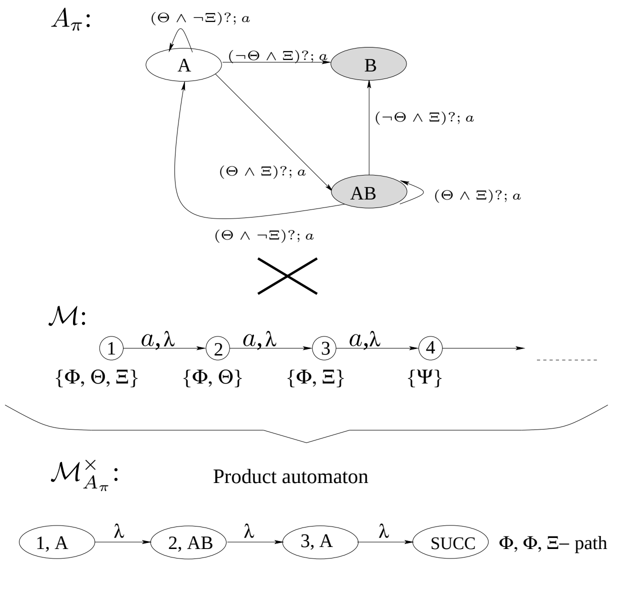

Fig. 7. Correct product automaton for φ_{im}

V. Conclusion

We have presented a stochastic extension of the logic PDL, SPDL, that allows the user to specify very complex performability requirements, including both state measures and path-based measures.

By a small example we have demonstrated how to check whether the model at hand meets the requirements. In the technical report version of this paper [7] we have also shown that bisimulation preserves validity of SPDL formulae. Furthermore we have shown that SPDL is strictly more expressive than CSL, aCSL and aCSL+.

In the near future we plan to implement SPDL and integrate it into our existing performance analysis tool CASPA [9]

Acknowledgments

The authors wish to thank Joachim Meyer-Kayser for many fruitful discussions.

References

[1] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, “Verifying continuous time Markov chains,” in *Computer-Aided Verification*, R. Alur and T. Henzinger, Eds., vol. LNCS 1102. Springer, 1996, pp. 146–162.

[2] C. Baier, J.-P. Katoen, and H. Hermanns, “Approximate Symbolic Model Checking of Continuous-Time Markov Chains,” in *Conurrency Theory*, J. Baeten and S. Mauw, Eds., vol. LNCS 1664. Springer, 1999, pp. 146–162.

[3] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, “On the logical characterisation of performability properties,” in *ICALP*, vol. LNCS 1853. Springer, 2000, pp. 780–792.

[4] H. Hermanns, J. Katoen, J. Meyer-Kayser, and M. Siegle, “Towards model checking stochastic process algebra,” in *Integrated Formal Methods*, vol. LNCS 1945. Springer, 2000, pp. 420–439.

[5] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, “Implementing a Model Checker for Performability Behaviour,” in *Fifth Int. Workshop on Performability Modelling of Computer and Communication Systems (PMCCS5)*, R. German, J. Luethi, and M. Telek, Eds. Universität Erlangen-Nürnberg, Arbeitsberichte des Instituts für Informatik, Band 34 Nummer 13, September 2001, 2001, pp. 110–115.

[6] J. Meyer-Kayser, “Verifikation stochastischer, prozessalgebraischer Modelle mit aCSL+ (in German),” Universität Erlangen-Nürnberg, Institut für Informatik 7, Tech. Rep. 01/03, 2003.

[7] M. Kuntz and M. Siegle, “A Stochastic Extension of the Logic PDL,” Friedrich-Alexander-Universität Erlangen-Nürnberg, Tech. Rep. 03/03, 2003.

[8] C. Baier, B. Haverkort, J.-P. Katoen, and H. Hermanns, “Model checking algorithms for continuous time Markov chains,” *to appear in IEEE Transactions on Software Engineering*, 2003.

[9] M. Kuntz, M. Siegle, and E. Werner, “CASPA: A performance evaluation tool based on stochastic process algebra and symbolic data structures,” in *to appear in tool proc. of the 2003 Illinois Int. Multiconference on Measurement, Modelling, and Evaluation of Computer-Communication Systems*, 2003.