Symbolic Model Checking of Stochastic Systems: Theory and Implementation

Matthias Kuntz, Markus Siegle

Department of Computer Engineering
University of the Federal Armed Forces Munich, Germany

Abstract. This paper presents IM-SPDL, a stochastic extension of the modal logic PDL, which supports the specification of complex performance and dependability requirements. The logic is interpreted over extended stochastic labelled transition systems (ESLTS), i.e. transition systems containing both immediate and Markovian transitions. We define the syntax and semantics of the new logic and show that IM-SPDL provides powerful means to specify path-based properties with timing restrictions. In general, paths can be characterised by regular expressions, also called programs, where the executability of a program may depend on the validity of test formulae. For the model checking of IM-SPDL time-bounded path formulae, a deterministic program automaton is constructed from the requirement. Afterwards the product transition system between this automaton and the ESLTS is built and subsequently transformed into a continuous time Markov Chain (CTMC) on which numerical analysis is performed. Empirical results given in the paper show that model checking IM-SPDL can be realised efficiently in practice.

Keywords: Stochastic systems, performance and dependability analysis, symbolic model checking, model checking software.

1 Introduction

It is extremely important to develop techniques for constructing and analysing distributed, concurrent hard- and software systems, which have become part of our daily life. Such systems must work correctly and meet high performance and dependability requirements. Our approach to the combined analysis of functional, performance and dependability aspects (the latter two commonly known as performability) is based on the formal verification of a stochastic model which describes both, functional and temporal aspects of behaviour.

Such models can be constructed with the help of high-level formalisms, where stochastic Petri nets and stochastic process algebras are among the most popular ones. Generalised stochastic Petri nets (GSPNs) [1] offer two types of transitions: Timed transitions, associated with an exponentially distributed delay, and immediate transitions which, once enabled, fire without delay. Immediate transitions have shown to be very useful for the modelling of complex synchronisation or cooperation schemes, for representing probabilistic decisions or simply

for modelling bookkeeping activities which consume only negligible time. For this reason, immediate activities are also an integral part of the Stochastic Activity Network (SAN) modelling formalism [24] as implemented in the Möbius modelling framework [10] for the modelling and analysis of performability properties of distributed systems. For similar reasons, immediate transitions have also been included, in the form of immediate actions, into several stochastic process algebras, such as TIPP [13], EMPA [8] and IMC [12]. Overall, one may say that immediate transitions are a very valuable and often used modelling feature.

While there has been substantial work on the model checking of stochastic systems, the aspect of immediate transitions has not been considered in this context. In this paper, we present an extension of the modal logic PDL [11], called IM-SPDL (immediate and Markovian stochastic PDL), which can be used for specifying requirements for models that contain both immediate and Markovian transitions. Such a model we call extended stochastic labelled transition system (ESLTS), since it has two types of transitions and carries action labels as well as state labels. As the paper shall explain, IM-SPDL is a very powerful logic in that it allows its user to specify requirements which are based on the probability measure of sets of execution paths, where regular expressions of actions and so-called tests are used to characterise the set of satisfying paths. The model checking of IM-SPDL time-bounded path formulae follows an automaton-based approach: From the requirement, a deterministic program automaton is constructed, and subsequently the product transition system between this automaton and the ESLTS is built and thereafter transformed into a continuous time Markov Chain (CTMC) on which numerical analysis is performed.

Related Work In recent years, many efforts have been made to devise temporal logics for the specification of system properties in the area of performance analysis, where the underlying model is a labelled Markov chain. One result of these efforts is the logic CSL (continuous stochastic logic) [6], introduced by [3] and extended in [7] with an operator to reason about steady-state behaviour. CSL allows the specification of certain types of performability measures (cf. [5]), but the specification of these measures is completely state-oriented, i.e. based on atomic propositions. In [15] an action-based variant of CSL, called aCSL, was proposed, which is not based on atomic propositions but on sequences of named actions and therefore more suitable for action-oriented formalisms such as process algebras. In [14] it was shown how to employ the logic aCSL for performability modelling. A first combination of the state-oriented and action-oriented approach was the logic aCSL+ [21], where regular expressions of actions are used to characterise satisfying paths. In [17] we presented the first ideas of a stochastic extension of the logic PDL (SPDL) which also combines state-oriented and action-oriented features, and where paths can be specified via regular expressions of actions and so-called tests. The logic asCSL [4], inspired by the path-based reward variables of [22], follows a similar motivation. However, we emphasise the fact that the model to be checked by all logics mentioned in this paragraph is a labelled CTMC which is not allowed to contain immediate transitions.

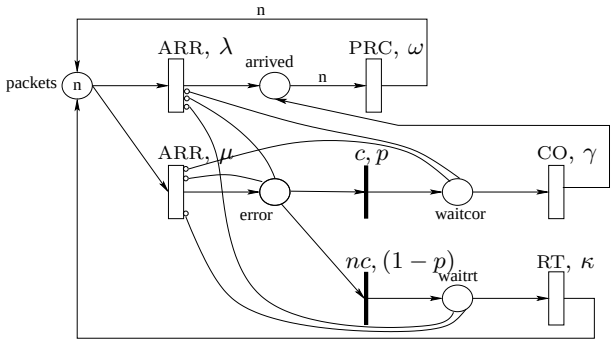

Fig. 1. GSPN-style model of a fault-tolerant packet collector

Organisation of the Paper This paper is organised as follows: In Sec. 2, we introduce ESLTS, the class of models which we consider. Sec. 3 defines the syntax and semantics of the new logics IM-SPDL. In Sec. 4, we show how model checking of IM-SPDL path formulae can be carried out, by constructing a product transition system on which numerical analysis is performed. Sec. 5 presents some empirical results obtained from a prototype implementation. Finally, in Sec. 6 we summarise the results and give pointers to future research.

Example 1 (Running example: Fault-tolerant packet collector). Throughout this paper, we use the example of a fault-tolerant packet collector. Fig. 1 shows the GSPN-style specification of this simple system which has the following repeating behaviour: n data packets arrive independently, are stored, and then all n data packets are jointly processed. Arrivals can either be error-free (upper transition ARR, rate λ) or erroneous (lower transition ARR, rate μ). Rather unusual for GSPNs, there are two timed transitions bearing the same name, ARR, which expresses the fact that these transitions are not distinguishable by an observer. If a data packet contains an error, this error can be correctable (immediate transition c) with a certain probability p, or non-correctable (immediate transition nc) with the complementary probability. In case of a correctable error, the error is corrected (transition CO) and more data packets can be received. If the error is non-correctable, the data packet has to be retransmitted (transition RT). During the processing of an erroneous packet, no new packet can arrive, which is modelled by the inhibitor arcs from places *error*, *waitcor*, and *waitrt* to the ARR transitions of the model. □

2 Model: Extended Stochastic Labelled Transition Systems

The model of the logic IM-SPDL is an extended stochastic labelled transition system (ESLTS). An ESLTS has two types of transitions, immediate and Markovian transitions. Immediate transitions are untimed transitions, whereas Markovian transitions are associated with an exponentially distributed delay.

Definition 1. (Extended Stochastic Labelled Transition System) An extended stochastic labelled transition system (ESLTS) is a quintuple $\mathcal{M} := (S, L, R_I, R_M, s)$, where:

- S is a finite set of states.
- $L : S \mapsto 2^{\mathsf{AP}}$ is the state labelling function that associates with every state $s \in S$ the set of atomic propositions which hold in that state. AP is the set of atomic propositions.
- $R_I : S \times \mathsf{Act}_I \times I\!P \times S$ is the immediate transition relation, where $I\!P = (0, 1]$. If $(s, a, p, s') \in R_I$, we write $s \overset{a,p}{\dashrightarrow} s'$. Act_I is a finite set of immediate action labels, i.e. actions, that are associated with immediate transitions, and $p \in I\!P$ is a probability. The probabilities associated with the immediate transitions leaving a particular state must sum up to 1 (provided that the state has at least one emanating immediate transition).
- $R_M : S \times \mathsf{Act}_M \times I\!R \times S$ is the Markovian transition relation. Act_M is a finite set of Markovian action labels, i.e. actions, that are associated with Markovian transitions. We require that $\mathsf{Act}_I \cap \mathsf{Act}_M = \emptyset$. If $(s, a, \lambda, s') \in R_M$, we write $s \xrightarrow{a,\lambda} s'$.
- $s \in S$ is the unique initial state of $\mathcal{M}$.

Example 2 (ESLTS of packet collector). In Fig. 2, the ESLTS $\mathcal{M}$ for the packet collector GSPN from Fig. 1 is shown, where we assume that the number n of data packets that are to be processed is equal to four.

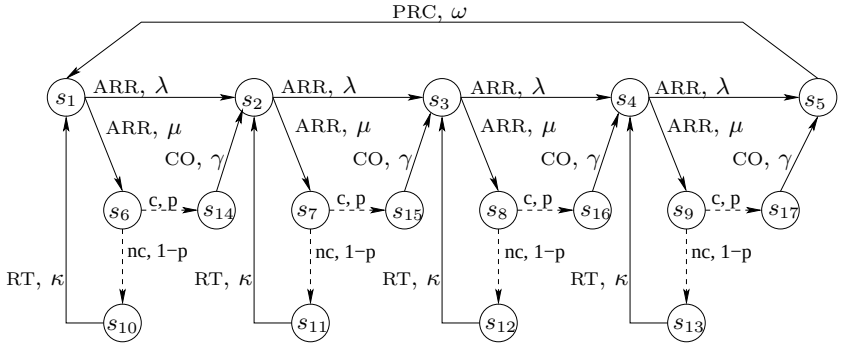

Fig. 2. ESLTS of the GSPN model for $n = 4$

The system has the following state labels:

$$L(s_5) = \{\mathsf{full}\}, \quad L(s_6) = ... = L(s_9) = \{\mathsf{error}\},$$
$$L(s_{10}) = ... = L(s_{13}) = \{\mathsf{waitrt}\}, \quad L(s_{14}) = ... = L(s_{17}) = \{\mathsf{waitcor}\}$$

The sets of immediate and Markovian actions are given as follows:

$$\mathsf{Act}_I := \{nc, c\}, \quad \mathsf{Act}_M := \{ARR, RT, CO, PRC\}$$

For example, transitions $s_6 \overset{nc,1-p}{\dashrightarrow} s_{10}$ and $s_6 \overset{c,p}{\dashrightarrow} s_{14}$ are immediate, whereas $s_1 \xrightarrow{ARR,\lambda} s_2$ and $s_{14} \xrightarrow{CO,\gamma} s_2$ are Markovian transitions. □

Since an ESLTS may have two types of transitions, there are two types of states, vanishing and tangible states.

Definition 2. (Vanishing and tangible states) A state of an ESLTS is called *vanishing (instable)* if it possesses at least one outgoing immediate transition. Otherwise the state is called *tangible (stable)*.

A vanishing state is left as soon as it is entered, i.e. its sojourn time is zero. A tangible state has at least one outgoing Markovian transition (unless it is absorbing), therefore its sojourn time is governed by an exponential distribution whose rate parameter λ equals the sum of all the rates of the Markovian transitions emanating from that state. In the context of compositional modelling formalisms, such as stochastic process algebras, a further refinement of the notions of tangible/vanishing states is possible [25]. However, as our model checking approach is independent of any high-level modelling formalism, as long as the model to be checked is a single ESLTS which is considered in isolation, it suffices to distinguish between vanishing and tangible states.

Example 3. In Fig. 2 states s_6, s_7, s_8, and s_9 are vanishing, the remaining states are tangible. □

For the semantics of the logic IM-SPDL, the following notion of a path is of great importance:

Definition 3. (Paths in $\mathcal{M}$) An infinite path σ of an ESLTS $\mathcal{M}$ is a sequence of transitions of the form $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \ldots$ where:

- $t_i = \tau(\sigma, i) \in \mathbb{R}_{\geq 0}$ is the real-valued sojourn time in s_i before passing to s_{i+1}.
- if $a_i \in \mathsf{Act}_M$, then $\exists \lambda : (s_i, a_i, \lambda, s_{i+1}) \in R_M$ and $t_i > 0$ is the sojourn time in state s_i (i.e. t_i is the value drawn from an exponential distribution).
- if $a_i \in \mathsf{Act}_I$, then $\exists p : (s_i, a_i, p, s_{i+1}) \in R_I$ and $t_i = 0$ is the sojourn time in state s_i.
- $\sigma[i]$ is the $(i+1)$st state on path σ.
- $\sigma@t$ is the state at time point t.
- $a[i]$ is the $(i+1)$st action on path σ.

A finite path σ is a finite sequence of transitions of the form: $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \ldots s_{n-1} \xrightarrow{a_{n-1},t_{n-1}} s_n$, where s_n is an absorbing state. For a finite path, $\tau(\sigma, i)$ is defined for $i < n$ as for infinite paths, and for $i = n$ we define $\tau(\sigma, i) = \infty$. The set $\mathsf{PATH}^{\mathcal{M}}(s) := \{\sigma \,|\, \sigma[0] = s\}$ is the set of all finite or infinite paths with initial state s.

3 Syntax and Semantics of IM-SPDL

The logic IM-SPDL is a stochastic extension of PDL [11], a multi-modal program logic. Beside the standard ingredients such as propositional logic and the

modal $\diamond$-operator ("possibly"), PDL enriches the $\diamond$-operator with so-called regular programs which are basically regular expressions of actions and tests (cf. Def. 5 below). If Φ and Ψ are PDL formulae and ρ is a program, then $\Phi \vee \Psi$, $\neg\Phi$ and $\langle\rho\rangle\Psi$ are formulae. $\langle\rho\rangle\Psi$ means that it is possible to execute program ρ, thereby ending up in a state that satisfies Ψ.

With respect to PDL we have added the following operators to obtain IM-SPDL: A path operator that extends the original PDL $\langle . \rangle$-operator by specifying time bounds within which the Ψ state has to be reached, a probabilistic path quantifier $\mathcal{P}_{\bowtie p}$ to reason about the transient probabilistic behaviour of a system, and a steady-state operator $\mathcal{S}_{\bowtie p}$ to reason about the behaviour of the system once stationarity of the underlying Markov chain is reached.

3.1 Syntax of IM-SPDL

The formulae of IM-SPDL are formally defined as follows:

Definition 4. (Syntax of IM-SPDL) Let $p \in [0,1]$ be a probability and $q \in \mathsf{AP}$ an atomic proposition and $\bowtie \in \{\leq, <, \geq, >\}$ a comparison operator.
The state formulae Φ of SPDL are defined as follows:

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

Path formulae ϕ are defined by:

$$\phi := \Phi[\rho]^I \Phi$$

where I is the closed time interval $[t, t']$ of the real axis. The symbol ρ represents a program as defined by Def. 5.

Definition 5. (Programs) Let $\mathsf{Act} = \mathsf{Act}_I \cup \mathsf{Act}_M$ be a set of actions, which are also called atomic programs, and TEST be a set of IM-SPDL state formulae. A program ρ is defined by the following grammar:

$$\rho := \epsilon \big| \Phi?; a \big| \rho; \rho \big| \rho \cup \rho \big| \rho^* \big| \Phi?; \rho \big| (\rho)$$

where $\epsilon \notin \mathsf{Act}$ is the empty program, $a \in \mathsf{Act}$ and $\Phi \in \mathsf{TEST}$.

The operators ; (sequential composition), $\cup$ (choice), and $*$ (Kleene star) have their usual meaning. The operator $\Phi?; \rho$ (resp. $\Phi?; a$) is the so-called test operator (also called guard operator). Its informal semantics is as follows: Test whether Φ holds in the current state of the model. If this is the case, then execute program ρ, otherwise ρ is not executable. Def. 5 requires that every atomic program is preceded by a test formula Φ, but this can be the trivial test (i.e. $\Phi = \mathsf{true}$). From standard automata theory it is known that regular expressions coincide with regular languages, i.e. sets of words that are generated according to the rules of regular expressions. Programs as defined in Def. 5 can be seen as regular expressions over the alphabet $\Sigma = \mathsf{TEST} \times (\mathsf{Act} \cup \epsilon)$. Words that are generated

from programs in IM-SPDL will be referred to as *program instances*. The set of these program instances is called, as before, a language.

The length of a program instance r, denoted by $\left|r\right|$, is the number of elements from Σ occuring in it. For $0 \leq i < \left|r\right|$, $r[i]$ is the $(i+1)$st element of r. $TF(r[i])$ denotes the test formula part of $r[i]$, and $Act(r[i])$ denotes the action part of $r[i]$.

Example 4 (Programs and program instances). Let $\mathsf{Act} = \mathsf{Act}_I \cup \mathsf{Act}_M$ as in Ex. 2 be the set of atomic programs, and $\mathsf{TEST} = \{\mathsf{error}, \mathsf{full}, ..., \neg\mathsf{error}, \neg\mathsf{full}, ...\}$ the set of test formulae. Using the grammar from Def. 5, possible programs are[1]

$\rho_1 = ARR; (\neg\mathsf{error}?; ARR)^*; c; CO; ARR^*$ and
$\rho_2 = (\neg\mathsf{full}?; ARR); c; CO; (\mathsf{full}?; \epsilon)$.

Some program instances of ρ_1 are:

$q = ARR; c; CO; ARR; ARR$,
$r = ARR; (\neg\mathsf{error}?; ARR); c; CO$ and
$s = ARR; (\neg\mathsf{error}?; ARR); (\neg\mathsf{error}?; ARR); c; CO; ARR$.

For r it holds that $\left|r\right| = 4$, $Act(r[1]) = ARR$ and $TF(r[1]) = \neg\mathsf{error}$. □

3.2 Semantics of IM-SPDL

Before we give the formal semantics of IM-SPDL, we provide an informal explanation. The meaning of negation ($\neg\Phi$) and disjunction ($\Phi \vee \Psi$) is as usual. $\mathcal{S}_{\bowtie p}(\Phi)$ asserts that the steady-state probability of the set of Φ-states, i.e. the probability to reside in a Φ-state once the system has reached stationarity, satisfies the boundary as given by $\bowtie p$. $\mathcal{P}_{\bowtie p}(\phi)$ asserts that the probability measure of the paths that satisfy ϕ is within the bounds as given by $\bowtie p$. Path formula $\Phi[\rho]^{[t,t']}\Psi$ means that a state that satisfies Ψ is reached within at least t but at most t' time units, and that all preceding states must satisfy Φ. Additionally, the action sequence of the path to the Ψ state must correspond to the action sequence of a word from the language $\mathcal{L}_\rho$ (the language induced by program ρ) and all test formulae that are part of program ρ must be satisfied by the corresponding states on the path.

Definition 6. (State probabilities) The probability to be in state s' at time point t, provided that the system is in state s at time 0, is given by

$$\pi^{\mathcal{M}}(s, s', t) = Pr(\sigma \in \mathsf{PATH}^{\mathcal{M}}(s) \big| \sigma@t = s')$$

The definition for steady-state probabilities is similar, taking into account that steady-state means 'on the long run':

$$\pi^{\mathcal{M}}(s, s') = lim_{t\to\infty}\pi^{\mathcal{M}}(s, s', t)$$

These definitions can be extended to sets of states: For $S' \subseteq S$:

$$\pi^{\mathcal{M}}(s, S', t) := \sum_{s'\in S'} \pi^{\mathcal{M}}(s, s', t) \quad \text{and} \quad \pi^{\mathcal{M}}(s, S') := \sum_{s'\in S'} \pi^{\mathcal{M}}(s, s').$$

[1] For better readability we often omit the trivial test formula, i.e. we write a instead of $(true?; a)$.

We are now ready to give the formal semantics of IM-SPDL.

Definition 7. (Semantics of IM-SPDL) The semantics of state formulae is defined as follows:

$$\begin{aligned}
\mathcal{M}, s \models q &\iff q \in L(s) \\
\mathcal{M}, s \models \neg\Phi &\iff \mathcal{M}, s \not\models \Phi \\
\mathcal{M}, s \models (\Phi \vee \Psi) &\iff \mathcal{M}, s \models \Phi \text{ or } \mathcal{M}, s \models \Psi \\
\mathcal{M}, s \models \mathcal{S}_{\bowtie p}(\Phi) &\iff \pi^{\mathcal{M}}(s, Sat(\Phi)) \bowtie p \\
\mathcal{M}, s \models \mathcal{P}_{\bowtie p}(\phi) &\iff Prob^{\mathcal{M}}(s, \phi) \bowtie p
\end{aligned}$$

$Sat(\Phi)$ is the set of states that satisfy Φ, and $Prob^{\mathcal{M}}(s, \phi)$ is the probability measure of all paths $\sigma \in \mathsf{PATH}(s)$ that satisfy ϕ:

$$Prob^{\mathcal{M}}(s, \phi) := Pr(\sigma \in \mathsf{PATH}^{\mathcal{M}}(s) \big| \mathcal{M}, \sigma \models \phi)$$

For the semantics of path formulae we have to relate the instances of the program ρ to words on paths in the ESLTS $\mathcal{M}$.

Definition 8. (Words on paths) The word $\mathcal{W}^k$ of length $k \geq 0$ on a path $\sigma \in \mathsf{PATH}^{\mathcal{M}}$ is defined as follows:

$$\begin{aligned}
& \mathcal{W}^0(\sigma) := \epsilon \\
& \mathcal{W}^k(\sigma) := \mathcal{W}^{k-1}(\sigma) \circ a[k-1] \\
\text{where} \quad & a[k-1] \in \mathsf{Act}_M \wedge \sigma[k-1] \xrightarrow{a[k-1], t_{k-1}} \sigma[k] \quad \text{or} \\
& a[k-1] \in \mathsf{Act}_I \wedge \sigma[k-1] \xrightarrow{a[k-1], 0} \sigma[k].
\end{aligned}$$

For $i = 0, 1, \ldots, k-1$, $\mathcal{W}^k(\sigma)[i]$ denotes the $i+1$st action on path σ.

Example 5. Consider a path $\sigma := s_1 \xrightarrow{ARR, t_1} s_6 \xrightarrow{c, 0} s_{14} \xrightarrow{CO, t_2} s_2 \xrightarrow{ARR, t_3} \ldots$ of the ESLTS from Fig. 2. The word of length 2 induced by σ is (ARR, c), the word of length 4 is (ARR, c, CO, ARR) and $\mathcal{W}^4(\sigma)[2] = CO$. □

Definition 9. (Semantics of path formulae) The semantics of path formulae is defined as follows:

$$\begin{aligned}
\mathcal{M}, \sigma \models \Phi[\rho]^{[t,t']}\Psi \iff \exists k \big(& \mathcal{M}, \sigma[k] \models \Psi \wedge \forall 0 \leq i < k (\mathcal{M}, \sigma[i] \models \Phi) \\
& \wedge \quad \mathit{time_restriction} \\
& \wedge \quad \mathit{program_matching} \big)
\end{aligned}$$

The first line states that there must be a state $\sigma[k]$ that satisfies Ψ and that all preceding states must satisfy Φ. The formula *time_restriction* is defined as follows:

$$\begin{aligned}
& \mathit{time_restriction} := \\
(1) \quad & \big((t = 0 \wedge \sum_{i=0}^{k-1} t_i \leq t') \vee \\
(2) \quad & \big(t \neq 0 \wedge ((t \leq \sum_{i=0}^{k-1} t_i \leq t') \vee (\sum_{i=0}^{k-1} t_i < t \wedge \sum_{i=0}^{k} t_i > t \wedge \sigma[k] \models \Phi)) \big)
\end{aligned}$$

It expresses the restrictions stemming from the time bounds that are imposed on paths. In line (1), if the lower time bound is zero, then the only requirement is to reach a Ψ-state before more than t' time units have passed. Line (2) covers the case where the lower time bound is greater than zero. In this case, either the entry time into state $\sigma[k]$ must lie within the interval $[t, t']$, or if the entry time is less than t, then the sojourn time in $\sigma[k]$ plus the sojourn times in the previous states must be greater than t. The formula *program_matching* is defined as follows:

$$
\begin{aligned}
& \mathit{program\_matching} := \\
(1)\quad & \big(\exists r \in \mathcal{L}(\rho) \wedge \big|r\big| = k \wedge \mathit{Act}(r[k-1]) \neq \epsilon \wedge \\
& \forall 0 \leq i \leq k-1 (\mathit{Act}(r[i]) = \mathcal{W}^{(k)}(\sigma)[i] \wedge \mathcal{M}, \sigma[i] \models TF(r[i]))\big) \vee \\
(2)\quad & \big(\exists r \in \mathcal{L}(\rho) \wedge \big|r\big| = k+1 \wedge \mathit{Act}(r[k]) = \epsilon \wedge \sigma[k] \models TF(r[k]) \wedge \\
& \forall 0 \leq i \leq k-1 (\mathit{Act}(r[i]) = \mathcal{W}^{(k)}(\sigma)[i] \wedge \mathcal{M}, \sigma[i] \models TF(r[i]))\big)
\end{aligned}
$$

This formula expresses that the word induced on path σ must be matched by the corresponding action parts of a program instance r and that the tests appearing in the program must be satisfied by the appropriate states on the path. There are two possibilities, as indicated in the formula: (1) If the last element of r is of the form $\Phi?; a$, where $a \neq \epsilon$, the corresponding state must satisfy the test formula and the last transition on the path must have a label identical to the action part of $r[k-1]$. (2) If the last element of r is of the form $\Phi?; \epsilon$, i.e. has an empty action part, then it only has to be checked whether the corresponding state on the path satisfies the test formula.

Example 6 (IM-SPDL formulae). With respect to the ESLTS $\mathcal{M}$ of Fig. 2 we specify four example requirements:

- Is the probability to receive four data packets with at most one packet containing a non-correctable error within 5 time units greater than 0.9?

$$\Phi_1 := \mathcal{P}_{>0.9}(\neg \mathsf{full}\ [ARR^*; nc; RT; ARR^* \cup ARR^*]^{[0,5]}\ \mathsf{full})$$

- Is the probability to reach a state in which the buffer is full with a single arrival greater than zero?

$$\Phi_2 := \mathcal{P}_{>0}(\neg \mathsf{full}\ [ARR]^{[0,\infty]}\ \mathsf{full})$$

  Requirement Φ_2 characterises state s_4.
- Is the probability that the buffer is full after at most 7.3 time units greater than 75 percent, if the following side conditions must be met: The only packet that contains an error is the fourth packet. This error must be correctable.

$$\Phi_3 := \mathcal{P}_{>0.75}(\mathsf{true}\ [ARR^*; (\Phi_2?; ARR); c; CO]^{[0,7.3]}\ \mathsf{true})$$

- In steady-state, is the probability that the system is currently processing either a correctable or a non-correctable error, less than 3%?

$$\Phi_4 := \mathcal{S}_{<0.03}(\mathsf{waitcor} \vee \mathsf{waitrt})$$

□

4 Model Checking IM-SPDL

In this section, we describe the model checking algorithm for the logic IM-SPDL. Central for this are the notions of program automata and product transition systems which we introduce in the sequel. Due to restricted space we will only describe the general idea of how to model check IM-SPDL path formulae, full details can be found in [18].

4.1 General Idea

The overall model checking algorithm for IM-SPDL is similar to that of CTL in the sense that we start by checking elementary subformulae and then proceed to the checking of more and more complex subformulae until the overall formula has been checked. Model checking propositional logic subformulae works as for CTL. Steady-state subformulae are checked in three steps as follows:

1. The ESLTS $\mathcal{M}$ is transformed into a state-labelled CTMC $\mathcal{M}'$, by eliminating the vanishing states, as described, for instance, in [2].
2. On $\mathcal{M}'$, model checking the steady-state operator works as for CSL [6]. Step 2 yields the verification results for the tangible states only.
3. During step 1, for each vanishing state the probability to reach a certain tangible state as the next tangible state is recorded. These probabilities are now combined with the results of step 2 in order to obtain the verification results for the vanishing states.

The basic model checking procedure for IM-SPDL path formulae with leading $\mathcal{P}_{\bowtie p}$ operator is more involved: We assume that we want to check whether state s of a given ESLTS $\mathcal{M}$ satisfies the formula $\mathcal{P}_{\bowtie p}(\phi)$, where $\phi = \Phi[\rho]^{[t,t']}\Psi$. The basic idea is to reduce the IM-SPDL model checking problem $\mathcal{M}, s \models \mathcal{P}_{\bowtie p}(\phi)$ to the CSL model checking problem of deciding whether $\mathcal{M}^*, s^* \models \mathcal{P}_{\bowtie p}(\mathsf{F}^{[t,t']}\mathsf{succ})$ for a CTMC $\mathcal{M}^*$ (to be constructed) and a state s^* of $\mathcal{M}^*$. A path satisfies the CSL path formula $\mathsf{F}^{[t,t']}\mathsf{succ}$, if within the time interval $[t, t']$ a state is reached that satisfies the new atomic proposition succ. We take the following steps:

1. From the program ρ we derive a deterministic program automaton $\mathcal{A}_\rho$, which is a variant of deterministic finite automata.
2. Using the given ESLTS $\mathcal{M}$ and the program automaton $\mathcal{A}_\rho$, we construct a product ESLTS (PESLTS) $\mathcal{M}^\times$. The state space of $\mathcal{M}^\times$ is the product between $\mathcal{M}$ and $\mathcal{A}_\rho$, i.e. states are of the form (s_i, z_i), where s_i is a state of $\mathcal{M}$ and z_i a state of $\mathcal{A}_\rho$. In addition, $\mathcal{M}^\times$ contains an absorbing error state with the new state label fail. The transitions in $\mathcal{M}^\times$ are labelled with rates in case of Markovian transitions and with probabilities in case of immediate transitions. The purpose of building this PESLTS is to check whether $\phi = \Phi[\rho]^{[t,t']}\Psi$ is functionally satisfiable in $\mathcal{M}$ or not.
3. In order to compute the probability measure of the paths satisfying ϕ we proceed as follows:

(a) All states (s_i, z_i) of $\mathcal{M}^\times$ for which s_i is a Ψ-state and z_i is an accepting state are replaced by a single absorbing goal state, with the special state label succ (for "success"). All transitions leading to a state (s_j, z_j) of the kind just described are redirected to this succ-state.
(b) The PESLTS $\mathcal{M}^\times$ is transformed into a CTMC $\mathcal{M}^*$ by eliminating vanishing states as in [2].

4. On $\mathcal{M}^*$ we can compute the probability measure of all paths satisfying the CSL formula $\mathcal{P}_{\bowtie p}(\mathsf{F}^{[t,t']}\mathsf{succ})$, which is equivalent to the probability measure of the paths satisfying the original formula $\mathcal{P}_{\bowtie p}(\phi)$ in the original model $\mathcal{M}$.

4.2 Program Automata

According to Sec. 4.1 we have to derive an automaton from a given program ρ. This is done by the following steps:

- At first, we construct from ρ a non-deterministic program automaton (NPA) $\mathcal{N}_\rho$. The definition of NPA is identical to that of non-deterministic finite automata as known from standard automata theory, albeit with special input alphabet Σ as introduced above in Sec. 3.1.
- Secondly, we turn $\mathcal{N}_\rho$ into a deterministic program automaton (DPA) $\mathcal{A}_\rho$. DPAs are formally defined in Def. 10. From this definition, it follows that the determinisation of an NPA is quite different from making a non-deterministic finite automaton deterministic. We will exemplify and justify our approach in example 7.

Definition 10 (Deterministic program automaton DPA). A DPA $\mathcal{A}$ is a quintuple $(Z_\mathcal{A}, \Sigma_\mathcal{A}, z^{Start}, E_\mathcal{A}, \delta_\mathcal{A})$ where

- $Z_\mathcal{A}$ is a finite set of states,
- $\Sigma_\mathcal{A} = \mathsf{TEST} \times (\mathsf{Act} \cup \epsilon)$ is the input alphabet,
- $z_\mathcal{A}^{Start} \in Z_\mathcal{A}$ is the initial state,
- $E_\mathcal{A} \subseteq Z_\mathcal{A}$ is the set of accepting states and
- $\delta_\mathcal{A} : Z_\mathcal{A} \times \Sigma_\mathcal{A} \to Z_\mathcal{A}$ is the state transition function which has to satisfy the following condition: If a state z possesses more than one outgoing transition then, either the action parts of the labellings of all outgoing transitions must be pairwise different, or if there are two or more transitions whose action parts are identical, then the test formula parts of them must not be true at the same time.

Our model checking approach relies on the following theorem:

Theorem 1. *For every NPA, an equivalent DPA can be constructed.*

Although Theorem 1 seems quite obvious, it should be noted that its proof is not the same as the equivalence proof of deterministic and non-deterministic finite automata from standard automata theory, since the input symbols have both a test part and an action part, and during determinisation the semantics of the test part must be taken into account. Instead of a formal proof of Theorem 1, we consider the following illustrative example:

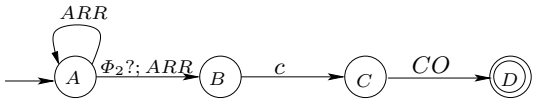

Fig. 3. Non-deterministic program automaton $\mathcal{N}_\rho$ for the program of Φ_3

Example 7 (NPA and DPA). Fig. 3 shows a non-deterministic program automaton $\mathcal{N}_\rho$ for the program $\rho = ARR^*; (\Phi_2?; ARR); c; CO$ (taken from Ex. 6, requirement Φ_3). The automaton is non-deterministic since the arcs emanating from state A, labelled with ARR (which is equivalent to $(true?; ARR)$) and $(\Phi_2?; ARR)$, have identical action label and the test parts are not disjoint. We cannot directly use such a non-deterministic automaton for our model checking algorithm, as the product construction explained in Sec. 4.3 could modify the stochastic behaviour of $\mathcal{M}$ and thus lead to wrong numerical results. Therefore we first construct a deterministic program automaton $\mathcal{A}_\rho$, which is shown in Fig 4. In $\mathcal{A}_\rho$, no two transitions are activated at the same time. This determinisation guarantees that the product automaton will preserve the branching structure and therefore the stochastic behaviour of $\mathcal{M}$. □

4.3 Product ESLTS Construction and Analysis

The central part of model checking probabilistic path formulae is the construction of the PESLTS of the model $\mathcal{M}$ and the DPA $\mathcal{A}_\rho$ for the program ρ of the path formula that is to be checked. In this section, we describe by means of an example how this PESLTS is generated.

Example 8 (Constructing the PESLTS). Let the ESLTS $\mathcal{M}$ from Fig. 2 and the DPA $\mathcal{A}_\rho$, shown in Fig. 4, be given. We now explain by example, how their PESLTS $\mathcal{M}^\times$, shown in Fig. 5, is constructed:

- The combinations of the transitions $s_1 \xrightarrow{ARR,\lambda} s_2$ in $\mathcal{M}$ and $A \xrightarrow{\neg\Phi_2?;ARR} A$ in $\mathcal{A}_\rho$ leads to the transition $(s_1, A) \xrightarrow{\lambda} (s_2, A)$ in $\mathcal{M}^\times$.
- In $\mathcal{M}$, transition $s_1 \xrightarrow{ARR,\mu} s_6$ is also possible, therefore $\mathcal{M}^\times$ also has the transition $(s_1, A) \xrightarrow{\mu} (s_6, A)$.
- Transition $(s_9, AB) \overset{p}{\dashrightarrow} (s_{17}, C)$ in $\mathcal{M}^\times$ stems from the transition $s_9 \overset{c,p}{\dashrightarrow} s_{17}$ in $\mathcal{M}$ and $AB \xrightarrow{c} C$ in $\mathcal{A}_\rho$.
- Transition $(s_6, A) \overset{1}{\dashrightarrow} \mathsf{fail}$ is composed of the transitions $s_6 \overset{c,p}{\dashrightarrow} s_{14}$ and $s_6 \overset{nc,1-p}{\dashrightarrow} s_{10}$ in $\mathcal{M}$, since in state A neither a c nor an nc transition is possible. Therefore in $\mathcal{M}^\times$ we obtain the transitions $(s_6, A) \overset{p}{\dashrightarrow} \mathsf{fail}$ and $(s_6, A) \overset{1-p}{\dashrightarrow} \mathsf{fail}$ which can be replaced by a single immediate transition that has probability one.

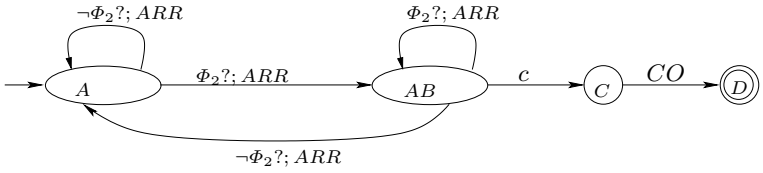

Fig. 4. Deterministic program automaton $\mathcal{A}_\rho$ for the program of Φ_3

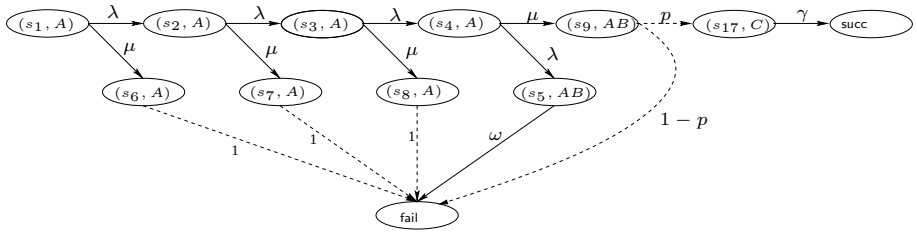

Fig. 5. Product ESLTS $\mathcal{M}^{\times}$

– In state C there is a transition $C \xrightarrow{CO} D$, where D is an accepting state, and in $\mathcal{M}$ there is a transition $s_{17} \xrightarrow{CO,\gamma} s_5$. In $\mathcal{M}^{\times}$ this leads to transition $(s_{17}, C) \xrightarrow{\gamma} \mathsf{succ}$, which stems from the fact that the automaton goal state D is accepting and that the goal state s_5 of the ESLTS satisfies $\Psi = \mathsf{true}$, i.e. state (s_5, D) satisfies the conditions of Sec. 4.1, item 3(a). □

After the product ESLTS $\mathcal{M}^{\times}$ has been constructed, its vanishing states are eliminated. We explain this elimination by means of the example:

Example 9 (Elimination of vanishing states). Let the PESLTS $\mathcal{M}^{\times}$ from Fig. 5 be given. The vanishing states (s_6, A), (s_7, A), (s_8, A) and (s_9, AB) are eliminated, thereby redirecting their incoming arcs to the respective successor states[2] and weighing them with the corresponding probabilities. This leads to the labelled CTMC $\mathcal{M}^*$ shown in Fig. 6. □

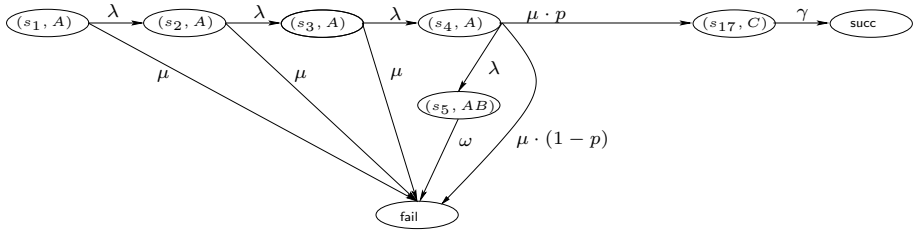

Fig. 6. $\mathcal{M}^*$: Result of the elimination of vanishing states

4.4 Complexity

It is known that the time complexity of model checking CSL is linear in the number of transitions of the model, the uniformisation rate (determined by the largest exit rate of any state of the model), and the involved time bound [6]. For model checking IM-SPDL probabilistic path formulae, a product transition system must be constructed first whose size, in the worst case, is the product of the original model and the program automaton at hand. However, in spite

[2] In general, sequences and even cycles of immediate transitions are possible, which situation can be handled by several published elimination algorithms.

of this potential blow-up of the state space, in most practical cases (like the ones in Sec. 5) the product transition system remains small (even smaller than the original model), since the program automaton typically restricts the possible behaviour of the original model and only the reachable portion of the product transition system needs to be constructed.

5 Empirical Results

This section presents empirical results of model checking IM-SPDL requirements, obtained with the help of the tool CASPA [19] which we have recently extended by model checking features.

5.1 The Tool CASPA

In CASPA the system to be checked is specified with the help of a stochastic process algebra (SPA) language, which is augmented by constructs for describing performability requirements as well as classical performance and dependability measures. Fig. 7 shows the building blocks of CASPA and their interaction. CASPA is a fully symbolic model checker, i.e. it relies completely on multi-terminal binary decision diagrams (MTBDDs), both for representing the transition system and for implementing the verification algorithms. In our experience, MTBDDs are superior to explicit representations in that they enable the compact storage of very large state spaces, where the symbolic representation of the ESLTS from the SPA specification can be generated very efficiently [16]. The use of MTBDDs thus enables the generation and storage of state spaces whose sizes are prohibitive in case of explicit storage schemes. Using extensions of MTBDDs and efficient algorithms for numerical analysis, as implemented in the tool PRISM [20], it is possible to analyse very large systems.

5.2 Case Studies

Fault-tolerant Packet Collector We check the path-based requirements Φ_1 and Φ_3 from Ex. 6. Table 1 shows the model checking times[3] for different values of the model parameter n. Columns "States $\mathcal{M}^*$" denote the number of states of the final transition system $\mathcal{M}^*$, obtained by eliminating the vanishing states from the respective product transition system $\mathcal{M}^\times$. For the model checking of Φ_1 and Φ_3, most of the time is needed for generating the product ESLTS, since the times for constructing the DPA and for eliminating the vanishing states were found to be negligible.

For Φ_1 the number of states for the PESLTS is about half the number of states of the original model because the requirement states that either all packets should be error-free or at most one packet may contain an incorrectable error. Likewise,

[3] All execution times are given in seconds, measured on a 3.0 GHz Pentium IV with 1GB of memory, running SuSe Linux 9.0.

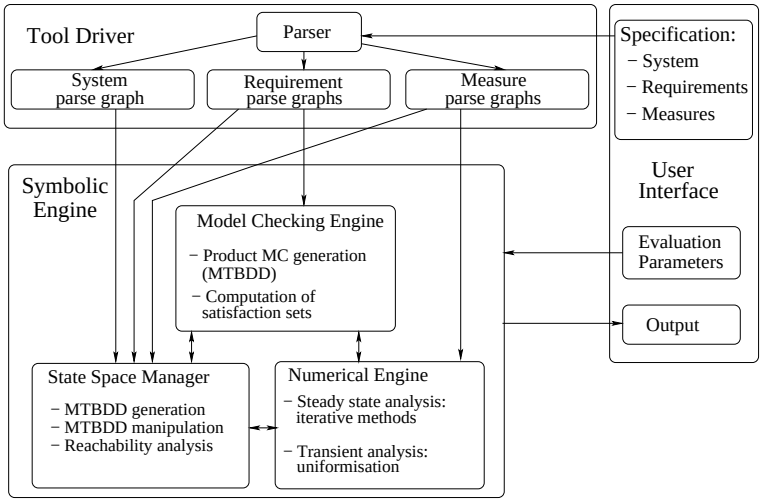

Fig. 7. Architecture of CASPA

n:	States:	Φ_1			Φ_3		
		States $\mathcal{M}^*$:	Gen. Time:	N.A. Time:	States $\mathcal{M}^*$:	Gen. Time:	N.A. Time:
5,000	20,001	10,002	1.69	0.32	5,003	1.80	0.11
15,000	60,001	30,002	6.09	1.08	15,003	6.70	0.32
30,000	120,001	60,002	13.72	2.20	30,003	14.59	0.68
50,000	200,001	100,002	25.88	3.57	50,003	28.12	1.16

Table 1. Model checking statistics for fault-tolerant packet collector for Φ_1 and Φ_3

we observe that for Φ_3 the number of states of the PESLTS is also smaller than for the original model. This is due to the fact that we are only interested in the paths that result from a single correctable error that occurs in the last packet to be received. This restricts the number of states, as all other transitions not being labelled with the appropriate actions in the respective states can be redirected to a single absorbing error state.

According to Table 1, the numerical analysis (N.A.) times are small compared to the generation times. However, model checking of Φ_1 consumes more time than that of Φ_3 because the number of states of the PESLTS is larger for Φ_1 (but the construction times for both PESLTSs are roughly the same).

Kanban System The Kanban manufacturing system was first described as a generalised stochastic Petri net in [9]. We consider a Kanban system with four cells, a single type of Kanban cards and the possibility that some workpieces may need to be reworked (i.e. moved back to the same cell).

The Kanban system demonstrates the usefulness of symbolic data structures for representing large state spaces. Table 2 gives the model statistics, where the scaling parameter n denotes the number of Kanban cards. For $n = 12$ and a reachable state space of more than 5.5 billion states only $32,324$ MTBDD nodes are required. Generating the state space from the given SPA specification and

n:	States:	Transitions:	Model Gen. Time:	MTBDD Nodes:
5	2,546,432	2.446e+07	0.42	5,392
6	11,261,376	1.15709e+08	0.94	8,086
7	41,644,800	4.45046e+09	1.59	10,389
10	1,005,927,208	1.20322e+10	8.37	23,245
11	2,435,541,472	2.98062e+10	12.90	27,425
12	5,519,907,575	6.88839e+10	17.22	32,324

Table 2. Model statistics for the Kanban system

restricting it to its reachable portion takes only about 17 seconds in case of $n = 12$. Consider the following requirements (only textual definitions are given):

- Φ_1: The probability, that within t time units a single workpiece needs exactly three reworks, should be below p.
- Φ_2: The requirement, that a job needs three reworks in the first cell and zero reworks in the second cell within a given time bound, should be satisfied with a probability of p?
- Φ_3 : The steady-state probability that cell 3 or 4 is blocked, i.e. the maximum number of Kanban cards is reached, should be below p.

From the results given in Table 3 we observe that for requirements Φ_1 and Φ_2 the state space of the PESLTS is dramatically smaller than that of the original system, which stems from the fact that in both cases only very specific paths in the system are of interest. The table also gives the generation time for the PESLTS for Φ_1 and Φ_2 with varying n. For Φ_3 the state space size is the same as for the original model, since in case of a steady-state requirement no PESLTS has to be generated. For Φ_1 and Φ_2 the numerical analysis time is small compared to the generation time (because the size of the PESLTS is very small). For Φ_3 numerical analysis for the cases $n \geq 10$ was not feasible (indicated by "–" in the table), since the allocation of a solution vector in main memory for more than one billion states is not possible on a common workstation, not to speak of the solution time.

n:	Φ_1			Φ_2			Φ_3	
	States $\mathcal{M}^*$:	Gen. Time :	N.A. Time:	States $\mathcal{M}^*$:	Gen. Time:	N.A. Time:	Gen. Time:	N.A. Time:
5	44	0.39	$< 10^{-6}$	42	0.19	$< 10^{-6}$	0.42	164
6	53	0.91	$< 10^{-6}$	53	0.93	$< 10^{-6}$	0.94	1093
7	62	1.71	$< 10^{-6}$	54	1.91	$< 10^{-6}$	1.59	75,600
10	89	7.76	0.01	108	7.71	0.03	8.37	–
11	98	12.88	0.03	119	11.46	0.04	12.90	–
12	107	16.03	0.05	130	17.89	0.09	17.22	–

Table 3. Model checking statistics for Kanban for Φ_1 to Φ_3

Fault-tolerant Multi Computer System (FTMCS) This computer system, originally described in [23], comes in different configurations, thereby achieving different degrees of fault-tolerance (due to the replication of certain components). For example, configuration $C1$ has three computers and three memory modules of which at least one must be operational for the entire system to be operational.

Config.:	States:	Φ		
		States $\mathcal{M}^*$:	Gen. Time:	N.A. Time:
C1	753,664	4,098	0.06	0.02
C2	2,152	33	0.02	$< 10^{-6}$
C3	889	11	0.01	$< 10^{-6}$
C4	123,760	134	0.03	0.01

Table 4. Model checking statistics for FTMCS for Φ

Requirement Φ which we check here, a time-bounded probabilistic path formula, describes a system failure which is only due to memory failures, no failures of other components shall contribute to this situation. Table 4 shows that, for all considered configurations, the state space size of $\mathcal{M}^*$ is very small compared to the size of the system, since only a very restricted number of paths is of interest, such that many transitions are redirected to the absorbing failure state.

6 Conclusions and Future Work

In this paper, we have introduced the logic IM-SPDL, a state- and action-oriented logic whose semantic model contains both Markovian and immediate transitions. We have shown how the model checking of IM-SPDL path formulae can be carried out with the help of a product transition system construction. The papers also presented some empirical results, obtained with our tool CASPA, which showed the feasibility and efficiency of the proposed method in spite of the theoretical worst-case complexity of the model checking algorithm.

As future work, it would be interesting to check whether the validity of IM-SPDL is invariant with respect to some notion of bisimulation, as is the case for other stochastic temporal logics. Such a result would enable reductions of the state space prior to model checking, which could be of great value, in particular in connection with compositional model checking approaches. We also plan to extend IM-SPDL with random time bounds, i.e. we intend to replace the fixed time bounds by time bounds whose value is drawn from a random variable.

References

1. M. Ajmone Marsan, G. Balbo, and G. Conte. A Class of Generalized Stochastic Petri Nets for the Performance Evaluation of Multiprocessor Systems. *ACM Transactions on Computer Systems*, 2(2):93–122, May 1984.
2. M. Ajmone Marsan, G. Balbo, G. Conte, S. Donatelli, and G. Franceschinis. *Modelling with Generalized Stochastic Petri Nets*. Wiley, 1995.
3. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Verifying continuous time Markov chains. In R. Alur and T.A. Henzinger, editors, *Computer-Aided Verification*, volume LNCS 1102, pages 146–162. Springer, 1996.
4. C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model Checking Action- and State-labelled Markov Chains. In *Int. Conf. on Dependable Systems and Networks: Performance and Dependability Symposium*, pages 701–710. IEEE Computer Society Press, 2004.
5. C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. On the logical characterisation of performability properties. In *ICALP*, pages 780–792. Springer, LNCS 1853, 2000.

6. C. Baier, B. Haverkort, H. Hermanns, and J.P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. *IEEE Trans. Software Eng.*, 29(7):1–18, July 2003.
7. C. Baier, J.-P. Katoen, and H. Hermanns. Approximate Symbolic Model Checking of Continuous-Time Markov Chains. In J.C.M. Baeten and S. Mauw, editors, *Conurrency Theory*, volume LNCS 1664, pages 146–162. Springer, 1999.
8. M. Bernardo and R. Gorrieri. A Tutorial on EMPA: A Theory of Concurrent Processes with Nondeterminism, Priorities, Probabilities and Time. *Theoretical Computer Science*, 202:1–54, 1998.
9. G. Ciardo and M. Tilgner. On the use of Kronecker operators for the solution of generalized stochastic Petri nets. Technical Report 96-35, ICASE, 1996.
10. D. Deavours, G. Clark, T. Courtney, D. Daly, S. Derisavi, J. Doyle, W.H. Sanders, and P. Webster. The Moebius Framework and its Implementation. *IEEE Transactions on Software Engineering*, 28(10):956–969, 2002.
11. M. Fischer and R. Ladner. Propositional dynamic logic of regular programs. *J. Comput. System Sci.*, 18:194–211, 1979.
12. H. Hermanns. *Interactive Markov Chains and the Quest for Quantified Quality.* Springer, LNCS 2428, 2002.
13. H. Hermanns, U. Herzog, U. Klehmet, V. Mertsiotakis, and M. Siegle. Compositional performance modelling with the TIPPtool. *Performance Evaluation*, 39(1-4):5–35, January 2000.
14. H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle. Implementing a Model Checker for Performability Behaviour. In *Proc. Fifth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS-5)*, pages 110–115, Erlangen, Germany, 2001.
15. H. Hermanns, J.P. Katoen, J. Meyer-Kayser, and M. Siegle. Towards model checking stochastic process algebra. In *Integrated Formal Methods*, volume LNCS 1945, pages 420–439. Springer, 2000.
16. M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. In *Process Algebra and Probabilistic Methods, Proc. PAPM-PROBMIV'02*, pages 188–206. Springer, LNCS 2399, 2002.
17. M. Kuntz and M. Siegle. A Stochastic Extension of the Logic PDL. In *Sixth Int. Workshop on Performability Modeling of Computer and Communication Systems (PMCCS-6)*, pages 58–61, 2003.
18. M. Kuntz and M. Siegle. A Stochastic Extension of the Logic PDL. Technical Report 2004-5, Universität der Bundeswehr München, Dept. of Computer Science, 2004. (http://fakinf.informatik.unibw-muenchen.de/~msiegle/PAPERS/TR_2004_5.ps.gz).
19. M. Kuntz, M. Siegle, and E. Werner. CASPA - A Tool for Symbolic Performance and Dependability Evaluation. In *Proceedings of EPEW'04 (FORTE satellite workshop)*, pages 293 – 307. Springer, LNCS 3236, 2004.
20. M. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: A hybrid approach. *International Journal on Software Tools for Technology Transfer (STTT)*, 6(2):128–142, 2004.
21. J. Meyer-Kayser. Verifikation stochastischer prozessalgebraischer Modelle mit aCSL+. Technical Report IB 01/03, Universität Erlangen-Nürnberg, Institut für Informatik 7, 2003 (in German).
22. W.D. Obal and W.H. Sanders. State-space support for path-based reward variables. *Performance Evaluation*, 35(3-4):233–251, 1999.

23. W. H. Sanders and L. M. Malhis. Dependability evaluation using composed SAN-based reward models. *Journal of Parallel and Distributed Computing*, 15(3):238–254, 1992.
24. W. H. Sanders and J. F. Meyer. Stochastic Activity Networks: Formal Definitions and Concepts. In *Lectures on Formal Methods and Performance Analysis*, pages 315–343. Springer, LNCS 2090, 2001.
25. M. Siegle. *Behaviour analysis of communication systems: Compositional modelling, compact representation and analysis of performability properties*. Shaker Verlag, Aachen, 2002.