A Stochastic Extension of the Logic PDL

Matthias Kuntz
Markus Siegle

Report 2004-5
October 2004

University of the Federal Armed Forces Munich

Department of

COMPUTER SCIENCE

Werner-Heisenberg-Weg 39 • D-85577 Neubiberg

Abstract. In this paper we present a stochastic extension of the modal logic PDL (propositional dynamic logic), SPDL, that is interpreted over labelled continuous time Markov chains (CTMC). We define the syntax and semantics of SPDL. SPDL provides, like PDL, powerful means to specify path properties. In general paths can be characterised by regular expressions, where the executability of a regular expression can depend on the validity of guard or test formulae. Such regular expressions enriched with test formulae are called programs. In order to model check SPDL path formulae it is necessary to derive from the programs a variant of deterministic finite automata and to build the product automaton between the labelled CTMC and this automaton. We describe two different ways to model check SPDL, at first via solving an integral equation system and secondly by transient analysis. We show that a variant of Markov bisimulation preserves the validity of SPDL formulae, finally we give the worst case complexity for model checking SPDL formulae by means of uniformisation.

1 Motivation and Introduction

It is commonplace that distributed, concurrent hard- and software systems have become part of our daily life. Because of our high dependency on these systems, it becomes more and more important to assert that they are working correctly and that they meet high performance requirements.

To do performance, dependability and reliability analysis it is necessary to have both a model and a number of measures of interest, like utilisation, mean number of jobs, mean time to failure and the like. The model is derived, roughly spoken in two steps: At first some specification method like (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. At second from this high level specification the low level representation is obtained. This low level representation is normally a continuous time Markov chain (CTMC). Now, to do performance, dependability or reliability analysis one has to specify the measures of interest. While for specification of models powerful means like the one mentioned above are available this is often not the case for measure specification.

In the realm of functional verification, temporal logics as CTL provide powerful means to specify complex properties of systems. In the recent years big efforts have been made to provide similar means for the specification of measures in the area of performance analysis. Thus, the logic CTL was extended to express complex measures.

In the sequel we will give a short account of the evolution of extensions of CTL.

At first we should mention the logic PCTL [10], a probabilistic logic, that is interpreted over discrete time Markov-chains. In PCTL the CTL path quantifiers A and E are replaced by a probabilistic operator $\mathcal{P}_{\bowtie p}(\varphi)$, that expresses that the probability that the path formula φ is satisfied meets the bounds expressed in $\bowtie p$.

Though, more important is the logic CSL, introduced in [1, 2] that is interpreted over a CTMC. This logic is extended in [3] by a steady-state operator $\mathcal{S}_{\bowtie p}(\Phi)$, that allows one to reason about steady-state probabilities, i.e. to reason about the probability that the system, considered on the long run is in a certain set of states. CSL provides also timed variants of the until- and the next-operator, U^I, X^I, allowing to make the validity of a formula also dependent, whether the time at which a satisfying state is reached at a time point t that is within the specified interval I.

A very important branch of modelling formalisms is that of stochastic process algebras (SPA). This formalism is action-oriented, i.e. the system behaviour is specified by actions. In this context, states constitute only an auxiliary mean within the semantic model of SPA-processes. The determination of the measures of interests in contrast is state oriented. To avoid this change of views, i.e. action- vs. state oriented, in [12] an action-based variant of CSL, aCSL, has been proposed. Similar to aCTL [7] the characterisation of satisfying paths is done by specifying sets of actions A that have to occur to satisfy a given path formula.

Though, aCSL has demonstrated its usefulness in several case studies, its weakness lies in the limited possibibilities to characterise satisfying paths. Paths satisfy an aCSL-requirement, if an arbitrary sequence of actions from A ocurred. It is not possible to state that a, b, c have to appear in this order and each of this actions exactly once. Therefore, aCSL has been extended to aCSL+ [16] to overcome this problem. In aCSL+, paths are characterised via regular expressions.

In this paper we give a stochastic extension of the logic PDL, where paths can not only be specified via regular expressions but also via properties of states that are visited during the execution of the regular expression specifying the paths. This is expressed via so called 'tests' that condition the execution of the subsequent regular expression. Subsequently, we will call regular expressions that may or may not contain tests 'programs'.

This paper is organised as follows: In the next section we give a short overview of PDL. In section 3 we introduce automata that represent PDL programs. In section 4 the syntax and semantics of our stochastic extension

of PDL, SPDL, is introduced. In section 5 we demonstrate how to construct automata for PDL. Section 6 is devoted to the introduction of model checking procedures for SPDL. In section 7 the method of uniformisation is explained in some detail. In section 8 apply by means of a small example the concepts introduced so far. In section 9 we show that the validity of SPDL formulae is preserved by a variant of Markov-AP-Bisimulation and we give a worst case approximation of the complexity of model checking SPDL probabilistic path formulae. In section 10 we extend SPDL by real time intervals, i.e. time intervals of the form $[t, t']$, where $t \neq 0$ is possible. Finally, we draw a conclusion and give a short outlook on future work.

2 Propositional Dynamic Logic PDL

PDL is a modal logic that is suited for reasoning about programs. It is interpreted over a Kripke structure $\mathcal{M}$, with $\mathcal{M} := \{S^{\mathcal{M}}, \mathcal{I}^{\mathcal{M}}\}$, where S is a set of states and $\mathcal{I}$ an interpretation function. The idea of modal logic, reasoning in situations where the truth value of a formula can vary over time, is perfectly suited for program execution. We can interprete the states of $\mathcal{M}$ as the set of all possible execution stages of a program. With any program ρ we can associate a binary relation R over $\mathcal{M}$ such that $(s, t) \in R$ iff there is an execution of ρ that, starting in state s terminates in state t. Now, the relation between modal logic and programs is that the programs ρ are written inside the modal operators $\Diamond$ and $\Box$: $\langle\rho\rangle$ and $[\rho]$, such that PDL is a multi-modal logic, where each program ρ is a modality. The intuitive meaning of $\langle\rho\rangle$ and $[\rho]$ is as follows:

- $\langle\rho\rangle\Phi$: It is possible to execute ρ and thereby ending up in a state satisfying Φ.
- $[\rho]\Phi$: Every terminating execution of ρ ends in a state satisfying Φ, where it is not necessary that there is a terminating execution at all.

2.1 Syntax and Semantics of PDL

In general, one can say that PDL contains elements that stem from propositional logic, modal logic and the algebra of regular expressions. The three constituents are composed as described by the following syntax definition.

Syntax of PDL: Basically a PDL-expression may consist of expressions of the following types:

- programs: $\rho, ...$
- formulae: $\Phi, \Psi, ...$

To build complex expressions out of simpler ones, i.e. atomic propositions and atomic programs, PDL provides a number of operators that can be categorised as follows:

- logical operators: $\neg, \vee$
- program operators: $\cup, ;, ^*$
- mixed operators: $\langle\ \rangle, ?$

PDL-expressions are mutually recursively defined:

Let Φ and Ψ be formulae and ρ_1 and ρ_2 programs, then: $\Phi \vee \Psi$, $\neg\Phi$, $\langle\rho\rangle\Phi$ are formulae, where formulae of this kind are associated with the states of the Kripke structure, i.e. state formulae, and $\rho_1;\rho_2$, $\rho_1 \cup \rho_2$, ρ_1^* and $\Phi?$ are programs. Programs are defined as follows:

Definition 1 (Programs). Let Act be a set of atomic programs, which we may also call actions and TEST be a set of state formulae. Together they form the alphabet Σ_ρ for the program ρ, i.e.

$$\Sigma_\rho := \mathsf{Act} \cup \mathsf{TEST}$$

A program ρ over an alphabet Σ_ρ is defined by the following grammar:

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

Where $a \in \mathsf{Act}$ and $\Xi \in \mathsf{TEST}$.

Semantics of PDL: Instead of giving the formal semantics of PDL, we will describe the intuitive meaning of some of the PDL-constructs. A more thorough account of the formal semantics of PDL can be found in [8, 11, 14].

- $\rho_1;\rho_2$: Execute ρ_1 and ρ_2 sequentially
- $\rho_1 \cup \rho_2$: Choose nondeterministically ρ_1 or ρ_2 and execute the chosen program
- ρ_1^*: Execute ρ_1 a non-deterministically chosen number of times, including zero times

– $\langle\rho_1\rangle\Phi$: It is possible to execute ρ_1 and halt in state satisfying Φ.
– $[\rho_1]\Phi$: Although not explicitly present in PDL, the box is the dual of the diamond and can be expressed as follows:

$$[\rho_1]\Phi \equiv \neg\langle\rho_1\rangle\neg\Phi$$

This means it is not possible to execute ρ_1 and end up in a state that does not satisfy Φ.
Equivalently, for every terminating computation of ρ_1 it holds that ρ_1 halts or stops in a state satisfying Φ. Note, that for the satisfaction of a formula of this kind no terminating computation at all must exist.
– $\Phi?;\rho_1$: Test, if Φ holds in the current state, if so, execute ρ_1 otherwise fail.

3 Automata for PDL

We will now introduce how to relate finite executions of programs that are definable by the syntactical means provided in section 2 with automata.

3.1 Alphabets, Programs and Automata

In this section we give a definition of programs for PDL that is adapted to our needs. We forbid programs of the form $(\Phi?;\epsilon)^*$, this will be justified in the sequel:

In the way we do model checking of SPDL formulae, i.e. by constructing a product Markov Chain between the system's original Markov chain and the automaton of the program defining the satisfying paths, it is not necessary to have (sub-)programs of the kind $(\Phi?;\epsilon)^*$, as with a test no transition in the Markov chain is associated and the program can be executed also zero times, the validity or non-validity of Φ in the actual state of the Markov chain is without significance for the model checking procedure.

Definition 2 (Programs). Let Act be a set of atomic programs, which we may also call actions and TEST be a set of state formulae. Together they form the alphabet Σ_ρ for the program ρ, i.e.

$$\Sigma_\rho := \mathsf{Act} \cup \mathsf{TEST}$$

A program ρ over an alphabet Σ_ρ is defined by the following grammar:

$$\rho := \epsilon \big| \rho;\rho \big| \rho \cup \rho \big| \Xi?;\rho \big| \rho_1 \big| (\rho)$$
$$\rho_1 := a \big| \rho_1;\rho_1 \big| \rho_1 \cup \rho_1 \big| \rho_1^* \big| \Xi?;\rho_1 \big| (\rho_1)$$

Where $a \in \mathsf{Act}$ and $\Xi \in \mathsf{TEST}$.

Definition 3 (Program transformation). Let a program ρ derived by the grammar from definition 2 be given, we apply to it the following transformation rules, such that the resulting program ρ' is equivalent to ρ.

(T1) Sequences of test formulae with no atomic programs, i.e. elements from Act, interspersed, i.e. sequences of the kind $\Xi_1?;\Xi_2?;...\Xi_n?;$ are transformed into a conjunction of the involved test formulae:

$$\Xi_1?;\Xi_2?;...\Xi_n?;\equiv\bigwedge_{i=1}^{n}\Xi_i?;$$

This transformation is correct, since it can be shown that the above equivalence relation holds: We show the correctness for $i=2$, the general case is an easy induction on the number of conjuncts.

$$\begin{aligned}\mathcal{I}((\Xi\wedge\Theta)?;\rho)&=\mathcal{I}((\Xi\wedge\Theta)?)\circ\mathcal{I}(\rho)\\&=\{(u,u)\big|u\in\mathcal{I}((\Xi\wedge\Theta))\}\circ\mathcal{I}(\rho)\\&=\{(u,u)\big|u\in(\mathcal{I}(\Xi)\cap\mathcal{I}(\Theta))\}\circ\mathcal{I}(\rho)\\&=\{(u,u)\big|u\in(Sat(\Xi)\cap Sat(\Theta))\}\circ\mathcal{I}(\rho)\end{aligned}$$

We have to show that the semantic definition for $\Xi?;\Theta?;\rho$ is identical to the one above:

$$\begin{aligned}\mathcal{I}(\Xi?;\Theta?;\rho)&=\mathcal{I}(\Xi?)\circ\mathcal{I}(\Theta?)\circ\mathcal{I}(\rho)\\&=\{(u,v)\big|\exists w(u,w)\in\mathcal{I}(\Xi?)\wedge(w,v)\in\mathcal{I}(\Theta?)\}\circ\mathcal{I}(\rho)\end{aligned}$$

For the remaining derivations we need the following equivalences:

$$\begin{aligned}(u,w)\in\mathcal{I}(\Xi?)&\Longleftrightarrow u=w\wedge u\in Sat(\Xi)\\(w,v)\in\mathcal{I}(\Theta?)&\Longleftrightarrow w=v\wedge w\in Sat(\Theta)\end{aligned}$$

Thus, it holds: $u=w\wedge w=v\rightarrow u=v$.
Furthermore, we have:

$$(u,u)\in\mathcal{I}(\Xi?)\Longleftrightarrow u\in Sat(\Xi)$$

Using the equivalences above, we can deduce:

$$\begin{aligned}&\{(u,v)\big|\exists w(u,w)\in\mathcal{I}(\Xi?)\wedge(w,v)\in\mathcal{I}(\Theta?)\}\circ\mathcal{I}(\rho)\\=&\{(u,u)\big|u\in Sat(\Xi)\wedge u\in Sat(\Theta)\}\circ\mathcal{I}(\rho)\\=&\{(u,u)\big|u\in(Sat(\Xi)\cap Sat(\Theta)\}\circ\mathcal{I}(\rho)\end{aligned}$$

(T2) As a second transformation, we turn each test formula that has no directly succeeding atomic program, i.e. test formulae in front of a choice operator $(\Xi?;(\rho_1 \cup \rho_2))$ or a star $(\Xi?;(\rho_1)^*)$ or at the end of a program $(\rho_1;\Xi?)$, into $\Xi?;\epsilon$, i.e. a test followed by the empty word. This is correct, since $\forall a \in \Sigma_\rho(a;\epsilon \equiv a)$.

(T3) Atomic programs a, not preceeded by a test formula are transformed into expressions of the kind $\mathsf{true}?;a$. This is correct, since test formulae are state formulae and true is assumed to hold in every state this does not affect the executability of a.

This transformation rules, from here on referred to as $(T1)$ to $(T3)$ are needed for the definition of the semantics of SPDL. In the sequel we will assume that each program ρ has been transformed according to $(T1)$ to $(T3)$. The imaginary alphabet Σ of such transformed programs is:

$$\Sigma := \mathsf{TEST} \times \mathsf{Act}$$
$$\text{Note: } \mathsf{true} \in \mathsf{TEST} \text{ and } \epsilon \in \mathsf{Act}.$$

I.e. each element of the alphabet is a tuple of test and atomic program.

Definition 4 (Program instances, length of program instances). A finite sequence of elements from the alphabet $\Sigma := \mathsf{TEST} \times \mathsf{Act}$ is called program instance. I.e. each element of this alphabet is a tuple consisting of a test formula succeeded by an atomic program.

The length of a program instance p denoted by $\left|p\right|$, is the number of elements from Σ occuring in it. For example:

$$p = (\mathsf{true}?;a);(\Xi?;b);(\mathsf{true}?;c) \Rightarrow \left|p\right| = 3$$

For $0 \leq i \leq \left|p\right|$ $p[i]$ is the $(i+1)$st element of p. For example:

$$p = (\mathsf{true}?;a);(\Xi?;b);(\mathsf{true}?;c) \Rightarrow p[1] = \Xi?;b$$

$Act(p[i])$ is the function that returns the atomic program part of $p[i]$. $TeF(p[i])$ is the function that returns the test formula part of $p[i]$. For example:

$$p = (\mathsf{true}?;a);(\Xi?;b);(\mathsf{true}?;c) \Rightarrow Act(p[1]) = b \wedge TeF(p[1]) = \Xi$$

Definition 5 (Equivalent program instances). Two program instances p_1 and p_2 are equivalent, $p_1 \equiv p_2$, iff either

– $p_1 = p_2$, i.e. they are syntactically equal

or

– $\left|p_1\right| = \left|p_2\right|$
– $\forall i((Act(p_1[i]) = Act(p_2[i])) \wedge (TeF(p_1[i]) = \mathsf{true} \Longleftrightarrow TeF(p_2[i]) = \mathsf{true}))$

Two programs ρ_1 and ρ_2 are equivalent, iff all their program instances are pairwise equivalent.

Definition 6 (Non-deterministic program automaton NPA). An NPA N is defined by the quintuple $(Z_N, \Sigma_N, Z_N^{Start}, E_N, \delta_N)$:

– Z_N: a finite set of states
– $\Sigma_N := \mathsf{TEST} \times \mathsf{Act}$: input alphabet
– Z_N^{Start}: a set of initial states, $Z_N^{Start} \subseteq Z_N$
– E_N: a set of accepting states $E_N \subseteq Z_N$
– δ_N: transition function: $\delta_N : Z_N \times \Sigma_N \rightarrow 2^{Z_N}$.

Definition 7 (Language of an NPA). The language of N, $\mathcal{L}(N)$ is defined as the set of all finite sequences of elements of its input alphabet Σ_N such that each sequence leads from an initial state to an accepting state:

$$\mathcal{L}(N) := \{p \in \Sigma_N^* \big| (z_0, p[1], z_1), ..., (z_{n-1}, p[n], z_n) \in \delta_N \wedge z_0 \in Z_N^{Start} \wedge z_n \in E_N\}$$

In this definition we have used the fact, that each n-ary function can be interpreted as $(n+1)$-ary relation. We have applied it to the transition function which is binary and the interpretation as transition relation is ternary.[1]

Definition 8 (Language of a program ρ). The set of all possible program instances of a program ρ is called its language, $\mathcal{L}(\rho)$[2].

For example, let the following program ρ be given:

$$\rho = (\Xi?; a); ((\mathsf{true}?; b); (\mathsf{true}?; c))^*; (\Theta?; d)^*$$

Then some instances of ρ are:

$$\Xi?; a \quad \big| \quad (\Xi?; a); (\mathsf{true}?; b); (\mathsf{true}?; c) \quad \big| \quad (\Xi?; a); (\mathsf{true}?; b); (\mathsf{true}?; c); (\Theta?; d)...$$

As another example let ρ be the following program:

$$\rho = (\mathsf{true}?; a); ((\Xi?; b) \cup (\mathsf{true}?; c))$$

[1] We will use this interpretation at many places without explicitly stating it
[2] Note, that we assume that ρ has been transformed according to $(T1)$ to $(T3)$

Then the language of ρ is the following set of program instances:

$$\mathcal{L}(\rho) = \{(\mathsf{true}?; a); (\Xi?; b), \quad (\mathsf{true}?; a); (\mathsf{true}?; c)\}$$

Theorem 1. *For each language $\mathcal{L}(\rho)$ there exists an NPA N_ρ, such that $\mathcal{L}(\rho) = \mathcal{L}(N_\rho)$.*

The proof follows the same lines as the proof for the common non-deterministic finite automata.

Definition 9 (Deterministic program automaton DPA). A DPA A is defined by the quintuple $(Z_A, \Sigma_A, z^{Start}, E_A, \delta_A)$:

- Z_A: a finite set of states
- $\Sigma_A : \mathsf{TEST} \times \mathsf{Act}$: input alphabet
- z_A^{Start}: a single initial state, $z_A^{Start} \in Z_A$
- E_A: a set of accepting states $E_A \subseteq Z_A$
- δ_A: state transition function: $\delta_A : Z_A \times \Sigma_A \to Z_A$: If a state z possesses more than one outgoing transitions, then it must hold, that either the action parts of the labellings of all outgoing transitions are different, or if there are at least two transitions which action parts are identical, then the test formula parts of them must fulfill the property that they can't be true at the same time.

Theorem 2. *For each NPA N an equivalent DPA A can be constructed*

This proof can be found in section 5

4 Stochastic PDL

This section presents the syntax and semantics of the stochastic propositional dynamic logic (SPDL).

4.1 Action- and State-Labelled Continuous-Time-Markov Chains

In this subsection the model that underlies SPDL is introduced.

Definition 10 (Action- and state-labelled continuous-time-Markov chains, ASMC). An $ASMC$ $\mathcal{M}$ is a quadruple (S, A, L, R), where

- S: finite set of states
- A: set of action names: $A = \mathsf{Act}$

– L: state labelling function: $S \to 2^{AP}$
– R: state transition relation : $R \subseteq S \times (A \times I\!R_{>0}) \times S$

AP is the set of atomic propositions.

Definition 11 (Rates and probabilities).

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

$\mathbf{R}_A(s, s')$: sum of all rates λ leading with action a from s to s'.

$$\mathbf{E}(s) := \sum_{s' \in S} \mathbf{R}_A(s, s')$$

$\mathbf{E}(s)$: sum of all rates of transitions emanating from state s.

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

$\mathbf{P}_A(s, s')$: probability to reach s' via s by performing an action a.
It holds:

$$\mathbf{P}_\emptyset(s, s') = \mathbf{R}_\emptyset(s, s') = 0 \text{ for all } s, s' \in S$$

For absorbing states:

$$\mathbf{P}_A(s, s') = \mathbf{R}_A(s, s') = \mathbf{E}(s) = 0 \text{ for arbitrary } s' \in S$$

Definition 12 (Paths in $\mathcal{M}$). An infinite path σ is a sequence of transitions of the form $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 ...$

– $s_i \in S$, $a_i \in A$, $(s_i, a, \lambda, s_{i+1}) \in R$
– $t_i = \tau(\sigma, i) \in I\!R_{>0}$: real sojourn time in s_i before passing to s_{i+1}.
– $\sigma[i]$: $(i+1)$st state on path σ
– $a[i]$: $(i+1)$st action on path σ
– $\sigma@t = \sigma[i]$: state that is reached at time instant t on path σ, it holds that i is the smallest index for which $t \leq \sum_{j=0}^{i} t_j$.

A finite path σ is a finite sequence of transitions of the form: $s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 ... s_{n-1} \xrightarrow{a_{n-1},t_{n-1}} s_n$, where $\mathbf{R}(s_i, s_{i+1}) > 0$ for all $i < n$ and $\mathbf{R}(s_n, s') = 0$ for

all $s' \in S$.
For finite paths σ, $\sigma[i]$ and $\tau(\sigma, i)$ are defined only for $i \leq n$, for $i < n$ as for infinite paths, for $i = n$ it holds $\tau(\sigma, i) = \infty$. For $t < \sum_{j=0}^{l-1} t_j$ let $\sigma@t = s_n$ for all other cases, $\sigma@t$ is defined as in the case of infinite paths.
The set of all paths with initial state s is called $\mathsf{PATH}(s)$

$$\mathsf{PATH}(s) := \{\sigma \big| \sigma[0] = s\}$$

Action sequences that characterise the set of fulfilling paths are defined in SPDL over programs. Programs are defined as in section 3.1. We need the following definition:

4.2 Syntax of SPDL

In this section we present the syntax of the stochastic extension of PDL. SPDL extends PDL with two probabilistic operators that allow to express steady state and transient measures. Like in the logic CSL [5, 4] SPDL provides two types of formulae: state formulae that are interpreted over the states of an $ASMC$ $\mathcal{M}$ and path formulae that are interpreted over paths in an $ASMC$.

Definition 13 (Syntax of SPDL). Let $p \in [0, 1]$, and $q \in \mathsf{AP}$ an atomic proposition, where AP is the set of atomic propositions and let $\bowtie \in \{\leq, <, \geq, >\}$.
The state formulae Φ of SPDL are defined as follows:

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

Path formulae are defined by:

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

where I is the closed interval $[t, t']$.[3]
Expressions of the form ρ are described by the grammar given in section 3.1.

4.3 Semantics of SPDL

Before we give the formal semantics of SPDL, we provide an informal explanation of the SPDL-formulae.

[3] In the sequel it is assumed $t = 0$

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

Formal semantics: For the semantics of path formulae we have to define the notion of words on paths. We need this, because we have to relate the paths of the DPA of the program π and the paths in the ASMC $\mathcal{M}$.

Definition 14 (Words on paths). The word $\mathcal{W}^k$ of length k, $k \geq 0$, over a path $\sigma \in \mathsf{PATH}$ is defined as follows:

$$\mathcal{W}^0(\sigma) := \epsilon$$
$$\mathcal{W}^k(\sigma) := \mathcal{W}^{k-1}(\sigma) \circ a[k-1]$$

where:

$$a[k-1] \in A \wedge \sigma[k-1] \overset{a[k-1],\lambda}{\longrightarrow} \sigma[k]$$

Where $\mathcal{W}^k(\sigma)[i] = p[i]$ is the i-th action on path σ.

We need some notation from probability theory, to define the semantics of :

Definition 15 (Probability vectors, state probabilities, etc.). If an initial probability distribution α is given, then the probability to be in state s' at time point t is given by

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

The length of α equals the cardinality of the state space of $\mathcal{M}$. The definition for steady state probabilities is similar, we only have to take into account that steady state means 'on the long run':

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

Often it occurs that a unique initial state s exists, i.e. $\alpha = \{1, 0, ..., 0\}$, we simply write Pr instead of Pr_{α} and $\pi^{\mathcal{M}}(s, s', t)$ instead of $\pi^{\mathcal{M}}(\alpha, s', t)$ in case of transient probabilities and analogously $\pi^{\mathcal{M}}(s, s')$ for steady state probabilities. The definitions can be extended to sets of states: For $S' \subseteq S$:

$$\pi^{\mathcal{M}}(\alpha, S') := \sum_{s' \in S'} \pi^{\mathcal{M}}(s, s') \text{ i.e.}$$
$$\pi^{\mathcal{M}}(\alpha, S') := lim_{t\to\infty} Pr_{\alpha}(\sigma \in \mathsf{PATH}^{\mathcal{M}} \big| \sigma@t \in S')$$

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

Definition 16 (Semantics of SPDL). The semantics of state formulae is defined as follows:

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

$Prob^{\mathcal{M}}(s, \varphi)$ is the probability measure of all paths $\sigma \in \mathsf{PATH}(s)$, starting in s to satisfy φ:

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

$\pi^{\mathcal{M}}(s, S')$ is the stationary state probability to be at time instant $\to \infty$ in a state from the set $S' \subseteq S$, provided that s is the state at time instant zero:

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

α denotes the given state probability distribution at time instant zero.

The semantics of path formulae is defined as follows:

$$\begin{aligned}
\mathcal{M}, \sigma \models \Phi[\rho]^{[0,t]}\Psi \Longleftrightarrow \exists k((\mathcal{M}, \sigma[k] \models \Psi \wedge \sum_{i=0}^{k} t_i \leq t) \wedge \\
(\forall 0 \leq i < k(\mathcal{M}, \sigma[i] \models \Phi)) \wedge (\exists p \in \mathcal{L}(\rho)((\big|p\big| = k) \wedge \\
\forall 0 \leq i < k(Act(p[i]) = \mathcal{W}^k(\sigma)[i] \wedge \mathcal{M}, \sigma[i] \models TeF(p[i]))))
\end{aligned}$$

4.4 Derived Operators

Temporal Operators The only temporal operator presented so far is $[\rho]^I$. We will show, how the operators 'U', 'X' ('next') and 'F' ('finally') can be derived:
The U-operator can be expressed as follows by $[\rho]^I$:

$$\begin{aligned}
\Phi \mathsf{U}^I \Psi &:= \Phi[\Sigma_\rho^*]^I \Psi \\
\Phi \mathsf{U} \Psi &:= \Phi[\Sigma_\rho^*]^{<\infty} \Psi
\end{aligned}$$

The F-operator kann is expressible by the following means:

$$\begin{aligned}
&\mathsf{F}[\rho]^I\Psi := \mathsf{true}[\rho]^I\Psi \\
&\mathsf{F}[\rho]\Psi := \mathsf{true}[\rho]^{<\infty}\Psi \\
&\mathsf{F}^I\Psi := \mathsf{true}[\Sigma^*]^I\Psi \\
&\mathsf{F}\Psi := \mathsf{true}[\Sigma^*]^{<\infty}\Psi
\end{aligned}$$

whereas X can be derived as follows:

$$\begin{aligned}
&\mathsf{X}[\Xi?;a]^I\Psi := \mathsf{true}[\Xi?;a]^I\Psi \\
&\mathsf{X}[\Xi?;a]\Psi := \mathsf{true}[\Xi?;a]^{<\infty}\Psi \\
&\mathsf{X}^I\Psi := \mathsf{true}[\Sigma\backslash\mathsf{TEST}]^I\Psi \\
&\mathsf{X}\Psi := \mathsf{true}[\Sigma\backslash\mathsf{TEST}]^{<\infty}\Psi
\end{aligned}$$

Modal Operators The modal operators $\big[\rho\big]$ ('necessarily') and $\big\langle\rho\big\rangle$ ('possibly') can be derived using the probabilistic path operater $\mathcal{P}_{\bowtie p}$ and the derived temporal operator F as follows:

$$\begin{aligned}
&\big\langle\rho\big\rangle\Psi := \mathcal{P}_{>0}(\mathsf{F}[\rho]^{\infty}\Psi) \\
&\big[\rho\big]\Psi := \neg\big\langle\rho\big\rangle\neg\Psi
\end{aligned}$$

5 Automata Construction

For model checking SPDL-path formulae it is necessary to derive a deterministic program automaton from the program ρ. This construction procedure will be covered in greater detail in this section.

5.1 Constructing the NPA

As we treat atomic programs and tests the same way when construction an NPA, and again treat them the same way as actions are treated when deriving a non-deterministic finite automaton NFA from a regular expression the construction process for an NPA N will be the same as for an NFA. Details are omitted.

5.2 Note on the notation

In sections 3 and 4 we have spent some effort on defining the semantics of SPDL and introduced input alphabets for automata and alphabets for programs that slightly differed in the way what they regarded as character or letter. Then we showed, that programs ρ that are derived by the grammar from 2 can be equivalently transformed into programs ρ' that serve as inputs for program automata thereby relating programs and program automata. Furthermore, the transformed program ρ' made it easier to define the semantics of SPDL-path formulae, because the relation between words on pahts of the ASMC and paths in the program automaton derived from ρ' was easier to establish. In this section we will somehow relax this strict notational rules and use a more sloppy way to handle programs. In the sequel we will use a as an abbreviation for $\mathsf{true}?;a$ and $\Xi?;$ as an abbreviation for $\Xi?;\epsilon$. This sloppiness eases a lot the presentation of the subsequent material. Using the strict notation from sections 3 and 4 would make it necessary to copiously describe, how elements of the input to an automaton that stem from a program instance p that either possess trivial tests as test formula part or the empty word as atomic program part can be equivalently transformed when applying rules for constructing automata from program instances. Using a sloppy notation we circumvent this difficulties.

5.3 Tests and Transitions

In this subsection we will describe how test transitions, i.e. transitions consisting either only of a test formula or a test formula succeeded by an atomic program, are treated on automata construction.

For reasons that lie in the model-checking procedure of SPDL it is in most cases, i.e. for internal transitions $(z, p[i], z')$, where $z' \notin E$, necessary to require that, if $p[i]$ is directly preceeded by a test with the empty word as its atomic program part, $p[i-1]$, then we want that $p[i-1]?;p[i]$ is a single transition, i.e. in the automaton the transitions $z_{i-2} \overset{p[i-1]?}{\longrightarrow} z_{i-1} \overset{p[i]}{\longrightarrow} z_i$ are replaced by $z_{i-2} \overset{p[i-1]?;p[i]}{\longrightarrow} z_i$. The exact way to obtain the last transitions from the two before is topic of the remainder of this subsection.

Program division: We present a basic, stepwise construction procedure to obtain a deterministic program automaton A_ρ from a given program ρ.

Let program ρ be given, to derive A_ρ, in a first step ρ is divided into i, $1 \leq i \leq n$ subprograms, ρ_i, such that each ρ_i contains at most one test, not equal to true. This eases the description of the treatment of test transitions while automata construction. The division of ρ proceeds as follows:

- As long as no tests are encountered, ρ is divided according to the syntactic structure of the expression.
- As soon as a test is found, the expression, governed by that test becomes a ρ_i.
  - According to ρ_i internal structure it might be necessary to further divide ρ_i.
- This division is continued until each sub-program contains at most one test

Let ρ_1, ρ_2 be subprograms without tests, then the test $\Xi?;$ governs $\Xi?;\rho_1$ and the test $\Xi?;$ governs $\Xi?;(\rho_1 \cup \rho_2)$.

Treatment of test transitions: On construction of the automaton A_ρ it might happen that transition are generated that are labelled with tests having an empty atomic program suffix, i.e. are of the form $\Xi?;\epsilon$. If the target state of such transitions is not an absorbing and accepting state, such transitions have to be treated in a special way. In the sequel we will write shortly $\Xi?;$ for $\Xi?;\epsilon$ for all involved test formulae. In the sequel we will use the following shorthands:

- X is either of the form a or $\Theta?;a$.
- Y is either of the form $\Lambda?;$ or b or $\Lambda?;b$
- Let z_Q be the source state of $\Xi?;$-transitions and Z_Y the (set of) target states of Y-transitions.

The following 'rules' can be applied to remove internal pure test transitions, i.e. transitions with a labelling that consists only of a test.

1. Let z_j be a non-accepting state, possessing loops of the form X, incoming transitions of the kind $\Xi?;$ and outgoing transitions Y. Replace the $\Xi?;$-transition from z_Q to z_j by $\Xi?;X$ and add to z_Q $\Xi?;Y$-transitions with target states from Z_Y.
2. Let z_j be a non-accepting state, with no loops, but with incoming transitions $\Xi?;$ and outgoing transitions Y. Then, replace in z_Q each $\Xi?;$-transition by $\Xi?;Y$-transitions with target states from Z_Y. The $\Xi?;$-transition can be deleted.

3. Let z_j be an accepting state, possessing loops of the kind X, no outgoing transitions, but incoming transitions of the form $\Xi?;$. Replace the $\Xi?;$-transition by $\Xi?;X$ and add to z_Q a new $\Xi?;$-transition that leads to an absorbing and accepting state. This state has possibly to be newly introduced.
4. Let z_j be an accepting state, possessing loops of the form X, incoming transitions of the kind $\Xi?;$ and outgoing transitions Y. Replace the ingoing $\Xi?;$-transition by $\Xi?;X$ and add to z_Q $\Xi?;Y$-transitions with target states from Z_Y. Add to z_Q a new $\Xi?;$-transition to an absorbing accepting state. This state has possibly to be newly introduced.

Example: Given the program $\rho = (c; a \cup d; \Xi?;); \Lambda?; b$. Program ρ consists of the following parts:

- $\rho_1 = c; a$
- $\rho_2 = d; \Xi?$
- $\rho_3 = \Lambda?; b$

For $\rho_1 \cup \rho_2$ and ρ_3 we obtain the automata shown in figure 1.

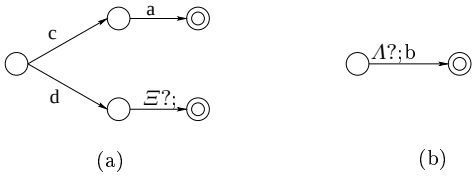

Fig. 1. Automata for $\rho_1 \cup \rho_2$, (a) and ρ_3, (b)

Putting the automata from figure 1 yield the nondeterministic automaton as shown on top of figure 2. Determinising and application of the transformation rules for internal test transitions yields the automaton shown on bottom of figure 2.

Correctness of the Rules: We will now show that the transformation rules given in this subsection are correct in the sense that the automata that are generated this way are equivalent to the original ones.

- **Correctness of rule 1:** An automaton having the form as described in 1 is derived from a program of the form $\Xi?; X^*; Y$. This yields the following

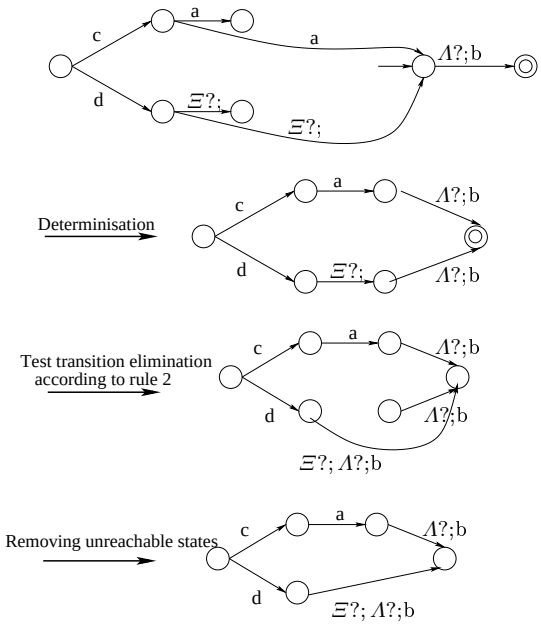

Fig. 2. Stepwise construction of A_ρ from N_ρ

syntactical derivations:

$$\begin{aligned}
\Xi?; X^*; Y &\equiv \Xi?; (\epsilon \cup X; X^*); Y && \text{Semantics of Kleene star}\\
\Xi?; (\epsilon \cup X; X^*); Y &\equiv (\Xi?; \epsilon \cup \Xi?; X; X^*); Y && \text{Distributivity of ';'}\\
(\Xi?; \epsilon \cup \Xi?; X; X^*); Y &\equiv (\Xi? \cup \Xi?; X; X^*); Y && a; \epsilon \equiv a\\
(\Xi? \cup \Xi?; X; X^*); Y &\equiv (\Xi?; Y \cup \Xi?; X; X^*; Y) && \text{Distributivity of ';'}
\end{aligned}$$

- **Correctness of rule 2:** An automaton having the form as described in rule 2 is derived from a program of the form $\Xi?; Y$. In this case, nothing has to be proven.
- **Correctness of rule 3:** An automaton having the form as described in rule 3 is derived from a program of the form $\Xi?; X^*$. We have the following syntactic conversions:

$$\begin{aligned}
\Xi?; X^* &\equiv \Xi?; (\epsilon \cup X; X^*) && \text{Semantics of Kleene star}\\
\Xi?; (\epsilon \cup X; X^*) &\equiv \Xi?; \epsilon \cup \Xi?; X; X^* && \text{Distributivity of ';'}\\
\Xi?; \epsilon \cup \Xi?; X; X^* &\equiv \Xi?; \cup \Xi?; X; X^* && a; \epsilon \equiv a
\end{aligned}$$

– **Correctness of rule 4:** An automaton having the form as described in rule 4 is derived from a program of the form $\Xi?; X^* \cup \Xi?; X^*; Y$.

$$\Xi?; X^* \cup \Xi?; X^*; Y \equiv \Xi?; (\epsilon \cup X; X^*) \cup \Xi?; (\epsilon \cup X; X^*); Y$$
$$\Xi?; (\epsilon \cup X; X^*) \cup \Xi?; (\epsilon \cup X; X^*); Y \equiv \Xi? \cup \Xi?; X; X^* \cup \Xi?; Y \cup \Xi?; X; X^*; Y$$
$$\Xi? \cup \Xi?; X; X^* \cup \Xi?; Y \cup \Xi?; X; X^*; Y \equiv \Xi?; \cup \Xi?; X; X^* \cup \Xi?; X; X^*; Y \cup \Xi?; Y$$
$$\Xi?; \cup \Xi?; X; X^* \cup \Xi?; X; X^*; Y \cup \Xi?; Y \equiv \Xi?; \cup \Xi?; X; X^*; (\epsilon \cup Y) \cup \Xi?; Y$$

5.4 Determinisation in Case of Ambiguous Tests

The automata constructed by the procedure as described so far, call them N might be non-deterministic. For model-checking purposes it is necessary to derive from N its deterministic version, A. Non-determinism here might stem from two sources and is purged in the following manner:

– Determine N by treating all labellings as action labellings as in the case of finite automata. This automaton is called N'.
– In N' ambiguous tests might occur, i.e. for the same state z several outgoing transitions might exist having the same atomic program a but different test formulae Ξ_i, $1 \leq i \leq m$. In the model $\mathcal{M}$ in which the test formulae are interpreted it is not necessarily the case that only one of the Ξ_i is true while all others are false. In such cases where several test formulae are satisfied the successor state in the model that is to be model-checked is not uniquely defined, therefore we have to provide means to combat this problem. The automaton obtained by applying this procedure will be called A.

Elimination of ambiguous tests: Let Ξ_i be the tests that emanate from z such that the succeeding atomic programs are identical, i.e. $Act(\Xi_i?; a) = Act(\Xi_j?; a)$. The algorithm in figure 3 removes ambiguous transitions.

We will now prove the following theorem from section 3:

Theorem 3. *For each NPA N an equivalent DPA A can be constructed*

Before we begin with the proof we should state what is meant by "equivalence" in the context of program automata. By equivalence we do not longer mean that both automata accept the same language, if we consider tests as being a part of the action name. By saying for any NPA an equivalent DPA can be

(1) $\mathcal{Z} := 2^{Z_{N'}}$
(2) **forall** $Z \in \mathcal{Z}$
(3) **forall** $a \in L(Z)$
/*$L(Z)$ is the set of transition labellings, emanating from Z*/
(4) $F := \{\Xi \big| a \in L(Z) \wedge TeF(a) = \Xi\}$
(5) $negF := \{\neg\Xi \big| \Xi \in F\}$
(6) $\big|F\big| := n$, $Con := \emptyset$
(7) $\mathcal{F}'[n] := 2^{F \cup negF}$ /*$\mathcal{F}'[n]$ is the powerset of $F \cup negF$, where each element has cardinality n */
(8) $\mathcal{F} := \mathcal{F}'[n] \backslash \{F' \in \mathcal{F}'[n] \big| \Xi \in F' \wedge \neg\Xi \in F'\}$
(9) **forall** $F'' \in \mathcal{F}$
(10) $Con' := Conj(F'')$ /*Conjunction of elements of F''. */
(11) $Con'' := Con' \backslash \{\bigwedge_{i=1}^{n} \neg\Xi_i\}$
(12) $Con := Con \cup Con''$
(13) **endforall**
(14) **forall** $\Xi \in Con$
(15) $\delta_A(Z, \Xi?; a) := \bigcup_{z \in Z} \{z' \big| \delta_{N'}(z, \Xi_i?; a) = z'$, for all subformulae $\Xi_i \in F\}$
(16) **endforall**
(17) **endforall**
(18) **endforall**

Fig. 3. Ambiguous test elimination algorithm

found, we mean that both are equivalent in a logical sense, i.e. by interpreting the tests. This means we say that two automata are equivalent, under any model the same action sequences can occur. At the end of the section we will explain this in more detail.

5.5 Motivation for our Notion of Equivalence

The special needs of model checking require that the (sub-)automaton as shown in 4 is nondeterministic, although from a purely syntactical point of view it can be considered to be deterministic.

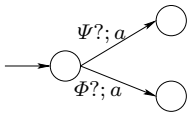

Fig. 4. Non-deterministic (sub-)automaton

As already motivated in great detail we have to perform the following transformation on the test formula part of the action labelling of the original NPA to obtain a DPA. [4] The desired DPA is shown in figure 5

[4] This transformation process mixes syntax and semantics of PDL

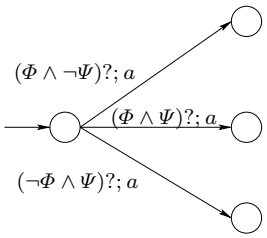

Fig. 5. Deterministic (sub-)automaton

We therefore have to show that both automata are equivalent, although the languages $_{DPA}$ and $_{NPA}$ recognised by them are different. It suffices to prove this claim for two transitions $\Phi?; a$ and $\Psi?; a$ emanating from a single source state.

5.6 Proof

1. We apply the semantics of PDL for the given expression, an expression that generates an NPA of the given form stems from the following PDL term:

$$\Phi?; a \cup \Psi?; a$$

Given the following semantic definition:

$$\begin{aligned}
[\![\Phi?; a \cup \Psi?; a]\!] &= [\![\Phi?; a]\!] \cup [\![\Psi?; a]\!] \\
&= ([\![\Phi?]\!] \circ [\![a]\!]) \cup ([\![\Psi]\!] \circ [\![a]\!]) \\
&= (\{(u,u)\big|u \in [\![\Phi]\!]\} \circ [\![a]\!]) \cup (\{(u',u')\big|u' \in [\![\Psi]\!]\} \circ [\![a]\!]) \\
&= (\{(u,u)\big|u \in [\![\Phi]\!]\} \cup \{(u',u')\big|u' \in [\![\Psi]\!]\}) \circ [\![a]\!] \\
&= \{(v,v)\big|v \in [\![\Phi]\!] \cup [\![\Psi]\!]\}) \circ [\![a]\!] \\
&= \{(v,v)\big|v \in [\![\Phi \vee \Psi]\!]\}) \circ [\![a]\!]
\end{aligned}$$

2. Now we observe that a DPA as the one shown in figure 5 stems from a PDL program that has the following appearance:

$$(\Phi \wedge \Psi)?; a \cup (\neg\Phi \wedge \Psi)?; a \cup (\Phi \wedge \neg\Psi)?; a$$

Applying to this program the semantic definitions of PDL yields:

$$\begin{aligned}
[\![(\Phi \wedge \Psi)?; a \cup (\neg\Phi \wedge \Psi)?; a \cup (\Phi \wedge \neg\Psi)?; a]\!] = \\
[\![(\Phi \wedge \Psi)?; a]\!] \cup [\![(\neg\Phi \wedge \Psi)?; a]\!] \cup [\![(\Phi \wedge \neg\Psi)?; a]\!] \\
= ... = (\{(w,w)\big|w \in ([\![(\Phi \wedge \Psi)]\!] \cup [\![(\neg\Phi \wedge \Psi)]\!] \cup [\![(\Phi \wedge \neg\Psi)]\!])\}) \circ [\![a]\!] \\
= (\{(w,w)\big|w \in [\![(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) \vee (\Phi \wedge \neg\Psi)]\!]\}) \circ [\![a]\!]
\end{aligned}$$

So, we have to show:

$$[\![\Phi \vee \Psi]\!] \equiv [\![(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) \vee (\Phi \wedge \neg\Psi)]\!]$$

This can be accomplished in two different ways:

1. Truth table: Comparing the truth tables of the two respective formulae yields the desired equivalence result:

Φ	Ψ	$\Phi \vee \Psi$	$\Phi \wedge \Psi$	$\neg\Phi \wedge \Psi$	$\Phi \wedge \neg\Psi$	$(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) \vee (\Phi \wedge \neg\Psi)$
0	0	0	0	0	0	0
0	1	1	0	1	0	1
1	0	1	0	0	1	1
1	1	1	1	0	0	1

2. Syntactic transformations:

   $$(\Phi \wedge \Psi) \vee (\neg\Phi \wedge \Psi) = (\Psi \wedge \Phi) \vee (\Psi \wedge \neg\Phi) = \Psi \wedge (\Phi \vee \neg\Phi) = \Psi$$

   Now applying this result to the third disjunct:

   $$\Psi \vee (\Phi \wedge \neg\Psi) = (\Psi \vee \Phi) \wedge (\Psi \vee \neg\Psi) = \Psi \vee \Phi$$

Thus, we could prove the claim that both automata are in fact equivalent. A few remarks on the meaning of *equivalence* in this context are in order.

Meaning of Equivalence in the Context of SPDL Programs Equivalence in the context of PDL deterministic and non-deterministic program automata cannot be considered to be language equivalence in the sense of finite automata as known from language theory. If the test formulae are interpreted as part of the action the languages of both automata types are clearly different.

In our context equivalence has to be interpreted as equivalence with respect to executability of programs. We have shown in the previous subsection that DPAs and NPAs are equivalent in this sense. Given a model $\mathcal{M}$ over which the test are interpreted we could show that the programs ρ_{DPA} and ρ_{NPA} are equivalent, i.e. if ρ_{DPA} led to an accepting state in $\mathcal{A}_{\rho_{DPA}}$ then ρ_{NPA} also led to an accepting state in $\mathcal{A}_{\rho_{NPA}}$.

Example: We will now illustrate the functionality of the algorithm from figure 3 by means of a small example.

Example 1. Let $\rho := (\Xi?; a)^*; \Theta?; a$. Construct A_ρ. The construction of N'_ρ is straightforward, only the final result is displayed in figure 6:

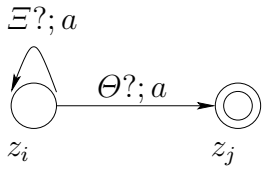

Fig. 6. Automaton N' for ρ

The transitions $\Xi?; a$ and $\Theta?; a$ will be replaced in A by: $(\Xi \wedge \Theta)?; a$, $(\neg\Xi \wedge \Theta)?; a$ and $(\Xi \wedge \neg\Theta)?; a$, all emanating from z_i. The successor states for the respective transitions are as follows:

$$\delta_A(z_i, (\Xi \wedge \Theta)?; a) = \{z_i, z_j\}$$
$$\delta_A(z_i, (\neg\Xi \wedge \Theta)?; a) = \{z_j\}$$
$$\delta_A(z_i, (\Xi \wedge \neg\Theta)?; a) = \{z_i\}$$

This yields the following automata graph for A_ρ: The construction of N'_ρ is straightforward, only the final result is displayed in figure 7:

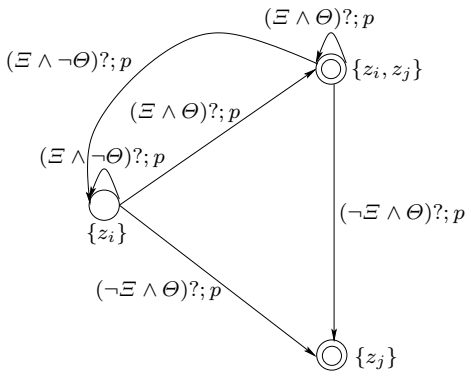

Fig. 7. Automaton A for ρ

6 Model Checking SPDL

The model checking procedure we present is an adaption of that of CSL [3] and aCSL+ [16]. This again is an adaption of that of CTL.

In SPDL the model checking procedure for non-probabilistic formulae is the same as for CTL. In the sequel we will provide means to model check probabilistic SPDL-state and -path formulae. Model checking path formulae as described in this section assumes that $I = [0, t']$.

6.1 Computing Stationary State Measures

The model checking procedure for computing stationary state measures is roughly the same as for CSL, which was described in [3]. The labelling function of the ASMC is extended by the notion of a 'characteristic state formula':

Definition 17 (Characteristic state formula). A characteristic state formula q_s is an atomic, propositional formula, only valid in state s of a specific ASMC $\mathcal{M}$.

The definition of characteristic state formula can be extended to the notion of a characteristic state set formula:

Let $S' \subseteq S$:

$$q_{S'} := \bigvee_{i=1}^{|S'|} q_{s'_i}$$

This formula is valid in each $s' \in S'$:

$$\begin{aligned} s'_i &\not\models q'_{s'_j}, \text{ iff } i \neq j \\ s'_i &\models q'_{s'_j}, \text{ iff } i = j \\ \Rightarrow s'_i &\models \bigvee_{i=1}^{|S'|} q_{s'_i}, \text{ iff } s_i \in S' \end{aligned}$$

For the computation of steady state measures no programs are needed. For **P** and **R** one obtains the following generalisations:

Definition 18 (Generalisations of P and R).

- $\mathbf{R}_A(s, s')$: total rate to come from s to s' by executing an arbitrary action from A. For an $a \in A$ for which holds $(s, a, \lambda, s') \notin R$, this rate is zero.

– $\mathbf{P}_A(s, s')$: total probability to reach s via s' by execution of an arbitrary action $a \in A$. For $a \in A$ for which holds $(s, a, \lambda, s') \notin R$, this probability is zero.

Let $BSCC(\mathcal{M})$ the set of bottom strongly connected components (BSCCs) of $\mathcal{M}$. For the computation of steady state measures in SPDL we obtain the following pseudo-algorithm:

The formula $\Psi := \mathcal{S}_{\bowtie p}(\Phi)$ has to be checked, i.e., to satisfy Φ it must hold:

$$\pi^{\mathcal{M}}(s, Sat(\Phi)) \bowtie p$$

1. Compute $BSCC(\mathcal{M}) = \{B_1, B_2, ..., B_m\}$.
2. Compute the set of states that satisfy Φ: $Sat(\Phi)$
3. $B_{Sat(\Phi)}(\mathcal{M}) := \{B_i \in BSCC(\mathcal{M}) \big| B_i \cap Sat(\Phi) \neq \emptyset\}$.
4. $\pi^{\mathcal{M}}(s, Sat(\Phi)) = \sum_{B \in B_{Sat(\Phi)}(\mathcal{M})} \left(Prob(s, \mathsf{F}q_B) \cdot \sum_{s' \in B \cap Sat(\Phi)} \pi^B(s') \right)$

q_B is the characteristic state set formula from definition 17. To fulfill $\mathsf{F}q_B$ eventually a state $s \in B$ must be reached. $\pi^B(s')$ is the stationary state probability of s' to be in B. $\pi^B(s')$ is computed as follows:

$$\pi^B(s') = \begin{cases} 1 & \text{if } B = \{s'\} \\ \sum_{s \in B, s \neq s'} \pi^B(s) \cdot R_A(s, s') = & \\ \pi^B(s') \cdot \sum_{s \in B, s \neq s'} R_A(s, s'), \text{ with } \sum_{s \in B, s \neq s'} \pi^B(s) = 1 & \text{otherwise} \end{cases}$$

$Prob(s, \mathsf{F}q_B)$ denotes the probability to finally reach B and is computed as follows:

$$Prob(s, \mathsf{F}q_B) = \begin{cases} 1 & \text{if } s \models q_B \\ \sum_{s'} \mathbf{P}_{\mathsf{A}}(s, s') \cdot Prob(s', \mathsf{F}q_B) & \text{otherwise} \end{cases}$$

6.2 Model Checking Probabilistic Path Formulae by Solving Integral Equations

At first we recall from subsection 4.2 that probabilistic path formulae are of the form: $\mathcal{P}_{\bowtie p}(\Phi[\rho]^I \Psi)$, where ρ is a program. I is the closed interval from 0 to t. From $\mathcal{M}$ only those paths are relevant for the measure that generate program instances on paths that are instances of ρ, i.e. those instances that lead in A_ρ from the initial state to an accepting state.

For a state $z \in Z_{A_\rho}$ we define its activation set:

Definition 19 (Activation set). For an arbitrary state z of Z we define

$$L(z) := \{a \in \Sigma_\rho \big| \exists z' \in Z_{A_\rho}(\delta_{A_\rho}(z,a) = z')\}$$

i.e. $L(z)$ is the set of all elements from Σ_ρ that emanate from z.

Furthermore, $Prob(s, \Phi[\rho]^I\Psi) = W(s, \Phi[\rho]^I\Psi, z_\rho^{Start})$, which will be characterised as follows:

$$W(s,\Phi[\rho]^I\Psi,z_\rho) = \begin{cases} 1 & \Longleftrightarrow (\mathcal{M}, s \models \Psi \wedge z_\rho \in E_\rho) \text{ or} \\ & \quad (\mathcal{M}, s \models \Psi \wedge (\exists \Xi(\mathcal{M}, s \models \Xi)) \wedge \\ & \quad \delta_\rho(z_\rho, \Xi?;) \in E_\rho \\ 0 & \Longleftrightarrow (\mathcal{M}, s \models \neg\Phi \wedge \neg\Psi) \text{ or} \\ & \quad (\mathcal{M}, s \models \neg\Phi \wedge \Psi \wedge z_\rho \notin E_\rho \wedge \\ & \quad \neg\exists \Xi \in Sat(s)(\delta_\rho(z, \Xi?;) \in \delta_\rho \wedge \\ & \quad \delta_\rho(z, \Xi?;) \in E_\rho)) \text{ or} \\ & \quad (\mathcal{M}, s \models (\Phi \wedge \Psi) \wedge \mathcal{M}, s \models \neg\Xi \wedge \\ & \quad \delta_\rho(z_\rho, \Xi?;) \in E_\rho \\ & \quad \wedge L(z_\rho) = \{\Xi?\}) \\ \int_0^t e^{-E(s)\cdot x} \cdot \sum_{a \in L(z_\rho)} \cdot \sum_{s' \in S} \mathbf{R}_a(s,s') \cdot & \\ W(s', \Phi[\rho']^{<I \ominus x}\Psi, \delta_\rho(z_\rho, a))dx & \Longleftrightarrow \mathcal{M}, s \models \Phi \wedge \neg\Psi \end{cases} \quad (1)$$

This deserves some words of explanation:

- We denote by $I \ominus x$ the following difference: $\{t - x \big| t \in I \wedge t \geq x\}$.
- Case 1: If the current state s in $\mathcal{M}$ is a state in which Ψ holds and z in A_ρ is an accepting state then the probability that formula φ is satisfied is equal to one. Alike the probability to satisfy φ is equal to one, iff s in $\mathcal{M}$ is a state satisfying Ψ and Ξ and the only transition from the current state in A_ρ is labelled $\Xi?$; and leads to an accepting state.
- Case 2: If s in $\mathcal{M}$ is a state that satisfies neither Φ nor Ψ, then the probality to satisfy φ is equal to zero. The same holds for a state s that satisfies $\neg\Phi \wedge \Psi$, but A_ρ is not in an accepting state. Alike, the probability to satisfy φ is zero, iff s is a state satisfying Ψ, but not Ξ and the current A_ρ-state is not accepting and the only transition leaving A_ρ is labelled with $\Xi?;$.
- Case 3: If s in $\mathcal{M}$ is a Φ-state then the probability to satisfy φ is equal to the probability to leave s in x time units and reach a state s'. This probability is taken over all atomic programs for which δ_A is defined.

This probability is multiplied with the probability to reach within $I \ominus x$, or equivalently $t - x$, time units a successor state s' in $\mathcal{M}$. As it might be the case that A_ρ offers several, differently labelled, outgoing transitions from its current state to some successor states the probabilities have to be summed up over all these different labellings. ρ' is the program that remains to be executed, after the execution of atomic program a.

When characterising the probabilities via systems of integral equations, a numerical, approximate procedure to solve them can be used. But the convergency of such methods is not satisfactorily, therefore, like in [3] we propose the approach to compute path probabilities via transient analysis.

6.3 Model Checking Probabilistic Path Formulae by Transient Analysis

To be able to do model checking of probabilistic path formulae by transient analysis, it is necessary to construct a product automaton $\mathcal{M}^\times$, from the $ASMC$ $\mathcal{M}$ and the deterministic program automaton A_ρ, i.e. $\mathcal{M} := \mathcal{M} \times A_\rho$. The construction process roughly proceeds as follows:
The transition labellings $a \in \mathsf{Act}$ are omitted. Rate informations and state labellings in $\mathcal{M}^\times$ are taken from $\mathcal{M}$. Let s be the current state in $\mathcal{M}$ and z the current state in A_ρ, then transitions from s having labellings that do not correspond to any of the labellings of transitions emanating from z are directed in $\mathcal{M}^\times$ to an absorbing error state FAIL. Transitions in $\mathcal{M}^\times$ are not directed to FAIL if the current $\mathcal{M}$-state s satisfies Φ and offers a transition whose labelling corresponds to one of the labellings of the current state z in the DPA or if state s satisfies Φ and Ξ and offers a transition whith labelling a, and in A_ρ z possesses a transition labelled $\Xi?; a$. If in $\mathcal{M}^\times$ an accepting state is reached, i.e. a state which components s and z are states satisfying Ψ respective are an accepting state in the DPA, the procedure stops.

The general idea behind our method is to reduce the model checking problem for probabilistic path formulae in SPDL to the model checking problem of CSL. I.e. we transform the SPDL formula $\Phi[\rho]^I\Psi$ into the CSL formula $\mathsf{F}^I\chi_G$, where χ_G is a characteristic formula which is attributed to those states in the product automaton, which ASMC components satisfy Ψ and which automaton components are accepting states of the program automaton.

The product automaton $\mathcal{M}^\times$ is called 'state-labelled Markov-chain' which is defined as follows:

Definition 20 (state-labelled Markov-chain, SMC). Let the $ASMC$ $\mathcal{M} = (S, A, L, R)$ and the DPA $A_\rho = (Z_\rho, \Sigma_\rho, z_\rho^{start}, E_\rho, \delta_\rho)$ given. The SMC $\mathcal{M}^\times = (S^\times, R^\times, L^\times)$ is defined as follows:

- $S^\times_{Start} \subseteq S^\times$
- $S^\times_{Acc} \subseteq S^\times$
- $R^\times \subseteq S^\times \times \mathbb{R}^+ \times S^\times$
- state space: $S^\times := \{(s_i, z_\rho^j) \big| s_i \in S \wedge z_\rho^j \in Z_\rho\} \cup \{\mathsf{FAIL}\}$
- initial states: $S^\times_{Start} := \{(s_i, z_\rho^{Start} \big| s_i \in S\}$
- accepting states: $S^\times_{Acc} := \{(s_i, z_\rho^j) \in S^\times \big| s_i \in Sat(\Psi) \wedge z_\rho^j \in E_\rho\}$
- labelling:
  1. $\forall (s_i, z_\rho^j) \in S^\times \backslash S^\times_{Acc} (L^\times(s_i, z_\rho^j) = L(s_i))$
  2. $\forall (s_i, z_\rho^j) \in S^\times_{Acc} (L^\times(s_i, z_\rho^j) = \{\chi_\mathsf{G}\})$
- transition function: $R^\times \subseteq (S \times Z) \times \mathbb{R}_{>0} \times (S \times Z)$

χ_G is a state formula that characterises exactly those states which automaton part is an accepting state and which Markov chain part is a state in which the formula Ψ of the path formula $\Phi[\rho]^{\leq t}\Psi$ holds.

Definition 21. For $A, B \in 2^{S^\times \times \mathbb{R}^+ \times S^\times}$ with $B = \emptyset$ or $|B| = 1$, $A \uplus B$ is defined as follows:

- $B = \emptyset$: $A \uplus B = A$
- $|B| = 1 \wedge B = \{(s, \lambda, s')\}$:

$$A \uplus B = \begin{cases} A \cup B & iff \ \nexists \gamma \in \mathbb{R}^+((s, \gamma, s') \in A) \\ (A \backslash \{(s, \gamma, s')\} \cup \{(s, \gamma + \lambda, s')\} & otherwise \end{cases}$$

$R^\times$ is succesively defined as follows:

1. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. $\mathcal{M}$ offers transitions with labelling a, so does A_ρ. The target state s' in $\mathcal{M}$ satisfies Φ or Ψ and at the same time the target state of A_ρ, z' must be accepting.

$$R^\times \uplus \{(s, z_\rho), \lambda, (s', z'_\rho) \big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{a} z'_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ [s' \in Sat(\Phi) \vee (s', z'_\rho) \in S^\times_{Acc}]\} \end{array}$$

2. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. A_ρ offers a test transition with test $\Xi?$; and atomic program a. $\mathcal{M}$ offers transitions with labelling a and satisfies the test formula of the corresponding transition in the DPA. The target state of $\mathcal{M}$ satisfies Φ or Ψ and at the same time the target state of A_ρ, z' must be accepting.

$$R^\times \uplus \{(s, z_\rho), \lambda, (s', z'_\rho) \big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{\Xi?;a} z'_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ s \in Sat(\Xi) \wedge \\ [s' \in Sat(\Phi) \vee (s', z'_\rho) \in S^\times_{Acc}]\} \end{array}$$

3. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. A_ρ offers in z a transition with labellings from Act_ρ the target state of this transition offers a transition with a labelling from TEST, say Θ. $\mathcal{M}$ satisfies in s the test formula of the corresponding z-transition. The target state of $\mathcal{M}$ satisfies Ψ and Θ.

$$R^\times \uplus \{(s, z_\rho), \lambda, (s', z''_\rho) \big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{\Xi?;a} z'_\rho \xrightarrow{\Theta?} z''_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ s \in Sat(\Xi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ s' \in Sat(\Theta) \wedge \\ [(s', z''_\rho) \in S^\times_{Acc}]\} \end{array}$$

4. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. $\mathcal{M}$ offers in s a transition labelled with a. A_ρ does not offer a transition bearing such a labelling.

$$R^\times \uplus \{(s, z_\rho), \lambda, \mathsf{FAIL} \big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ (z_\rho \xrightarrow{a} z'_\rho) \notin \delta_\rho\} \end{array}$$

5. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. Both $\mathcal{M}$ and A_ρ offer in s resp. z a transition labelled with

a. The target state of $\mathcal{M}$ does not satisfy Φ and the target state of $\mathcal{M}^\times$ is not accepting.

$$R^\times \uplus \{(s, z_\rho), \lambda, \mathsf{FAIL}\big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{a} z'_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ [s' \notin Sat(\Phi) \wedge (s', z'_\rho) \notin S^\times_{Acc}]\} \end{array}$$

6. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. Both $\mathcal{M}$ and A_ρ offer in s resp. z a transition labelled with a, where in A_ρ a is preceeded by a test. $\mathcal{M}$ does not satisfy the test formula in its current state s.

$$R^\times \uplus \{(s, z_\rho), \lambda, \mathsf{FAIL}\big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{\Xi?;a} z'_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ s \notin Sat(\Xi)\} \end{array}$$

7. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. Both $\mathcal{M}$ and A_ρ offer in s resp. z a transition labelled with a, where in A_ρ a is preceeded by a test. The target state of $\mathcal{M}$ does not satisfy Φ and the target state of $\mathcal{M}^\times$ is not accepting.

$$R^\times \uplus \{(s, z_\rho), \lambda, \mathsf{FAIL}\big| \begin{array}{l} s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{\Xi?;a} z'_\rho \wedge \\ s \in Sat(\Phi) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ s \in Sat(\Xi) \wedge \\ [s' \notin Sat(\Phi) \wedge (s', z'_\rho) \notin S^\times_{Acc}]\} \end{array}$$

8. In the SMC no accepting state has been reached. The original state s in $\mathcal{M}$ satisfies Φ. Both $\mathcal{M}$ and A_ρ offer in s resp. z a transition labelled with a, where in A_ρ a is preceeded by a test. The target state z'_ρ is not accepting and offers a transiton with labelling from TEST to an accepting state z''_ρ. The target state s' of $\mathcal{M}$ does not satisfy Θ.

$$R^\times \uplus \{(s, z_\rho), \lambda, \mathsf{FAIL}\big| \begin{array}{l} (s \xrightarrow{a,\lambda} s' \wedge z_\rho \xrightarrow{\Xi?;a} z'_\rho \xrightarrow{\Theta?} z''_\rho) \wedge \\ (s, z_\rho) \notin S^\times_{Acc} \wedge \\ s' \notin Sat(\Theta) \wedge \\ z''_\rho \in E_\rho\} \end{array}$$

6.4 Correctness of Model Transformation

In this subsection we will show that our transformation is correct, i.e. we show that the probability mass of the CSL formula that is checked in model $\mathcal{M}^\times$ is equal to the probability mass of the original formula $\Phi[\rho]^I\Psi$ in the original model $\mathcal{M}$.

To summarise the idea of section 6.3 we have done the following to perform transient analysis to check probabilistic SPDL path formulae:

- Transforming $\mathcal{M}$ to $\mathcal{M}^\times$
- thereby transforming $\Phi[\rho]^I\Psi$ to $\mathsf{F}^I\chi_G$

The following theorem states that the transformation steps are correct:

Theorem 4 (Correctness of model transformation). *The transformation of $\mathcal{M}$ into $\mathcal{M}^\times$ is correct. I.e. the probability of satisfying $\Phi[\rho]^I\Psi$ in $\mathcal{M}$ is equal to the probability of reaching χ_G within time $t \in I$ in $\mathcal{M}^\times$:*

$$Pr\{\sigma \in Path_s^{\mathcal{M}} \mid \mathcal{M}, \sigma \models \Phi[\rho]^I\Psi\} = Pr\{\sigma^\times \in Path_{(s,z_0)}^{\mathcal{M}^\times} \mid \exists t \in I : \mathcal{M}^\times, \sigma^\times@t \models \mathsf{F}^I\chi_G\}$$

Before we can prove theorem 4 we need the following definitions:

Definition 22 (Indicator function). The function $Ind(\mathcal{M}, s, \phi)$ indicates, whether an arbitrary SPDL state formula ϕ is satisfied in a given state s of a fixed model $\mathcal{M}$:

$$Ind(\mathcal{M}, s, \phi) = \begin{cases} 1 & \text{iff } \mathcal{M}, s \models \phi \\ 0 & \text{else} \end{cases}$$

For the reader's convenience we repeat definition 19:

Definition 23 (Activation set). For an arbitrary state z of Z we define

$$L(z) := \{a \in \Sigma_\rho \big| \exists z' \in Z_{A_\rho}(\delta_{A_\rho}(z, a) = z')\}$$

i.e. $L(z)$ is the set of all elements from Σ_ρ that emanate from z.

Definition 24 (End condition of a program). Let ρ be a program and $\mathcal{A}$ its corresponding program automaton. The end conditions of a program ρ are those suffixes of form $\Phi?;\epsilon$, where $\Phi = true$ is possible.

$$Fin_z(\mathcal{A}) = \begin{cases} true & \text{iff } z \in E \\ false & \text{iff } z \notin E \wedge \forall a \in L(z) : (\delta(z, a) \notin E) \\ \Phi_1 \vee ... \vee \Phi_n & \text{iff } z \notin E \wedge \forall i : (\Phi_i?;\epsilon \in L(z)) \wedge \exists \Phi_i : (\delta(z, \Phi_i?;\epsilon) \in E) \end{cases}$$

Proof (Theorem 4). We will prove theorem 4 by induction on the length of paths.
Induction start: $|\sigma| = |\sigma^{\times}| = 1$: Using the standard semantics of CSL (cf. [3]) we obtain:

$$Pr\{\sigma^{\times} \in Path^{\mathcal{M}^{\times}}_{(s,z_0)} \mid \exists t \in I : (\mathcal{M}^{\times}, \sigma^{\times}@t \models \chi_G)\} =$$
$$\int_0^t \sum_{(s',z') \in S^{\times}} \mathbf{R}((s,z_0),(s',z')) \cdot e^{-\mathbf{E}((s,z_0))\cdot x} \cdot Ind(\mathcal{M}^{\times},(s',z'),\chi_G)dx$$

As the length of the path is one, $Ind(\mathcal{M}^{\times},(s',z'),\chi_G)$ is either 1 or 0, i.e. χ_G either holds in (s',z') or does not.

For the original formula, the probability measure can be characterised as follows:

$$Pr\{\sigma \in Path^{\mathcal{M}}_{s} \mid \mathcal{M}, \sigma \models \Phi[\rho]^I \Psi\} =$$
$$\int_0^t \sum_{\{\Phi?;a \mid \Phi?;a \in L(z) \wedge \mathcal{M}, s \models \Phi\}} \sum_{s' \in S} \mathbf{R}_a(s,s') \cdot e^{-\mathbf{E}(s)\cdot x} \cdot Ind(\mathcal{M}, s', \Psi \wedge Fin_{z'}(\mathcal{A}))dx$$

Therefore we will now show that:

$$\int_0^t \sum_{\{\Phi?;a \mid \Phi?;a \in L(z) \wedge \mathcal{M}, s \models \Phi\}} \sum_{s' \in S} \mathbf{R}_a(s,s') \cdot e^{-\mathbf{E}(s)\cdot x} \cdot Ind(\mathcal{M}, s', \Psi \wedge Fin_{z'}(\mathcal{A}))dx =$$
$$\int_0^t \sum_{s' \in S} \sum_{\{\Phi?;a \mid \Phi?;a \in L(z) \wedge \mathcal{M}, s \models \Phi\}} \mathbf{R}_a(s,s') \cdot e^{-\mathbf{E}(s)\cdot x} \cdot Ind(\mathcal{M}, s', \Psi \wedge Fin_{z'}(\mathcal{A}))dx =$$
$$\int_0^t \sum_{(s',Z') \in S^{*}} \mathbf{R}((s,Z_0),(s',Z')) \cdot e^{-\mathbf{E}((s,Z))\cdot x} \cdot Ind(\mathcal{M}^{\times},(s',Z'),\chi_G)dx$$

The last equation holds, since by construction of $\mathcal{M}^{\times}$ we can conclude that $\sum_{\{\Phi?;a \mid \Phi?;a \in L(z) \wedge \mathcal{M}, s \models \Phi\}} \mathbf{R}(s,s') = \mathbf{R}((s,z_0),(s',z'))$. Therefore and by construction it holds that the two outer sums are equal. By construction of $\mathcal{M}^{\times}$ from $\mathcal{M}$ we conclude $\mathbf{E}(s) = \mathbf{E}((s,z_0))$. $Ind(\mathcal{M}^{\times},(s',z'),\chi_G) = Ind(\mathcal{M}, s', \Psi \wedge Fin_{z'}(\mathcal{A}))$ by construction, as those states are labelled with χ_G in which $Fin_{z'}(\mathcal{A}))$ and Ψ hold and in $\mathcal{A}$ an accepting state has been reached.
Induction step: We assume that for paths of length n the assumption holds, now we consider paths $\sigma^{\times}$ resp. σ of lenght $n+1$:

Let $\sigma^{\times'}$ resp. σ' be paths of length n, where $\sigma^{\times'}$ is suffix of $\sigma^{\times}$ and σ' is suffix of σ, then

$$Pr\{\sigma^{\times} \in Path^{\mathcal{M}^{\times}}_{(s,Z_0)} \big| \mathcal{M}^{\times}, \sigma^{\times}@t \models \chi_G\} =$$

$$\int_0^t \sum_{(s',Z') \in S^{\times}} \mathbf{R}((s,Z_0),(s',Z')) \cdot e^{-\mathbf{E}((s,Z_0))\cdot x} \cdot Pr\{\sigma^{\times'} \in Path^{\mathcal{M}^{\times}}_{(s,Z_0)} \big| \mathcal{M}^{\times}, \sigma^{\times'}@(t-x) \models \chi_G\}$$

Analogously:

$$Pr\{\sigma \in Path^{\mathcal{M}}_{s\in S} \big| \mathcal{M}, \sigma \models \Phi[\rho]^{\leq t}\Psi\} =$$

$$\int_0^t \sum_{\{\Phi?;a|\Phi?;a\in L(z)\wedge\mathcal{M},s\models\Phi\}} \mathbf{R}_a(s,s') \cdot e^{-\mathbf{E}(s)\cdot x} \cdot Pr\{\sigma' \in Path^{\mathcal{M}}_{s\in S} \big| \mathcal{M}, \sigma' \models \Phi[\rho']^{\leq t-x}\Psi\}$$

where ρ' is the suffix of ρ. Using **I.H.** and the induction start we conclude that the theorem holds, i.e.

$$Pr\{\sigma^{\times} \in Path^{\mathcal{M}^{\times}}_{(s,Z_0)} \big| \mathcal{M}^{\times}, \sigma^{\times}@t \models \chi_G\} = Pr\{\sigma \in Path^{\mathcal{M}}_{s\in S} \big| \mathcal{M}, \sigma \models \Phi[\rho]^{\leq t}\Psi\}$$

7 Details: Numerical Analysis Methods

This section aims to give a quick overview of mathematical methods we referred to and that are used for verification of path formulae.[5] In section 6.2 we gave an integral equation characterisation of the probability measure for time bounded path formulae. As mentioned earlier, model checking by directly solving integral equations is not satisfactory, because the numerical properties are not satisfactory. Therefore, more efficient means have been devised to solve the model checking problem. As the integral characterisation is only needed in 6.2 we will not go into details and refer to any book on numerical analysis, e.g. [6].

7.1 Transient Analysis and Uniformisation

Transient Analysis: The numerical properties of direct approaches to solve the Volterra integral equation system directly are not satisfactory. The transformation of the original ASMC $\mathcal{M}$ into SMC $\mathcal{M}^{\times}$ however, makes it possible to reduce the model checking problem of time bounded path formulae to transient analysis of the CTMC at hand.

[5] This overview does not claim to be exhaustive!

In general, to do transient analysis on CTMCs it is necessary to solve the Chapman-Kolmogorov differential equation system:

$$\frac{d}{dt}\vec{\rho}^{\mathcal{M}^\times}(\alpha,t) = \vec{\rho}^{\mathcal{M}^\times}(\alpha,t)\cdot Q.$$

$\vec{\rho}^{\mathcal{M}^\times}(\alpha,t)$ is a vector of length $\left|S^\times\right|$ and its elements are the probability to be at time instant t in state $s \in S^\times$, given an initial distribution α.

Q is the infinitesimal generator matrix and is derived from the rate matrix R, by setting $Q(s,s') = R(s,s')$, if $s \neq s'$. The rate matrix R characterises the transitions between the states of a CTMC. If $R(s,s') = \lambda$, $\lambda > 0$, then it is possible to move from s to s' with rate λ. The diagonal elements of R are replaced by $Q(s,s) = -E(s,s) = -\sum_{s\neq s'} R(s,s')$.

The unique solution of the Chapman-Kolmogorov differential equation system is given as follows:

$$\vec{\pi}^{\mathcal{M}^\times}(\alpha,t) = \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{Q\cdot t}$$

The matrix exponential can be rewritten as follows (Taylor series expansion):

$$e^{Q\cdot t} = \sum_{k=0}^{\infty}\frac{(Q\cdot t)^k}{k!}$$

The attempt to solve the matrix exponential using the Taylor expansion is not satisfactory, because ([17]):

- the truncation point of the series can not be computed efficiently
- the round-off errors are note negligeable, because Q contains both negative and non-negative entries.
- where Q is sparse, it is the case that $(Q\cdot t)^i$ becomes non-sparse.

Therefore more appropriate means have to be used to solve the equation.

Uniformisation: For uniformisation we define a stochastic matrix P, i.e. a matrix having entries that range from 0 to 1. P is derived from Q:

$$P := I + \frac{Q}{\lambda}$$

I is the identity matrix. λ is chosen as the maximum absolute value of the diagonal entries of the generator matrix Q, i.e. $\lambda \geq max(\left|Q(i,i)\right|)$. Therefore it is obvious, that P is a stochastic matrix. P is a DTMC. We rewrite Q:

$$Q = \lambda\cdot(P - I)$$

We obtain:

$$\begin{aligned}\vec{\pi}^{\mathcal{M}^\times}(\alpha,t) &= \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{Q\cdot t} = \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{(\lambda\cdot(P-I))\cdot t} \\ &= \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{-\lambda\cdot I\cdot t}\cdot e^{-\lambda\cdot P\cdot t} = \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{-\lambda\cdot t}\cdot e^{-\lambda\cdot P\cdot t}\end{aligned}$$

Using a series expansion we have

$$\vec{\pi}^{\mathcal{M}^\times}(\alpha,t) = \vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot e^{-\lambda\cdot t}\cdot\sum_{k=0}^{\infty}\frac{(\lambda\cdot t)^k\cdot P^k}{k!}$$

$e^{-\lambda\cdot t}\cdot((\lambda\cdot t)^k/(k!))$ are Poisson probabilities.

This Taylor-series now can be solved more efficiently. We write the equation above as follows:

$$\begin{aligned}\vec{\pi}^{\mathcal{M}^\times}(\alpha,t) &= \sum_{k=0}^{\infty}e^{-\lambda\cdot t}\frac{(\lambda\cdot t)^k\cdot P^k}{k!} = \sum_{k=0}^{\infty}e^{-\lambda\cdot t}\frac{(\lambda\cdot t)^k}{k!}\cdot(\vec{\pi}^{\mathcal{M}^\times}(\alpha,0)\cdot P^n) \\ &= \sum_{k=0}^{\infty}e^{-\lambda\cdot t}\frac{(\lambda\cdot t)^k}{k!}\cdot\vec{\pi}_k\end{aligned}$$

$\vec{\pi}_k$ is the distribution of state probabilities in the DTMC P after k steps and can be computed recursively:

$$\vec{\pi}_0 = \vec{\pi}(\alpha,0) \qquad\qquad \vec{\pi}_k = \vec{\pi}_{k-1}\cdot P$$

Now, we have reduced the problem to a number of vector-matrix multiplications. The question is, how large this 'number' is, i.e. we have to determine the truncation point of the series. We compute $\vec{\pi}_{approx}$ instead of $\vec{\pi}$, because the series looks like this:

$$\vec{\pi}^{\mathcal{M}^\times}_{approx}(\alpha,t) = \sum_{k=0}^{n_{approx}}e^{-\lambda\cdot t}\frac{(\lambda\cdot t)^k}{k!}\cdot\vec{\pi}_k$$

This trucation point n_{approx} can be computed efficiently. It has to be the least value for n_{approx} that satisfies the following condition:

$$\sum_{k=0}^{n_{approx}}\frac{(\lambda\cdot t)^k}{n!} \geq (1-\epsilon)\cdot e^{-\lambda\cdot t}$$

Where ϵ is the maximum round-off error we allow. The Poisson probabilities are computed using the Fox-Glynn-algorithm [9]. To check the validity of the path formula a method like the one described in [4, 13] can be employed.

The CTMC $\mathcal{M}^{\times}$, on which we check the variant of the original path formula, $\Pi = \mathcal{P}_{\bowtie p}(\Phi\ [\rho]^I\ \Psi)$, transformed to $\Pi' = \mathcal{P}_{\bowtie p}(\mathsf{true}\ \mathsf{U}^I\ \mathsf{SUCC})$, has to be uniformised, $\mathsf{unif}(\mathcal{M}^{\times}) := \mathcal{U}$. On $\mathcal{U}$ we check whether the probability bound p holds for Π resp. Π'.

8 Example: System Model and Measures

To illustrate our approach, specifying and checking performability measures using the logic SPDL, we consider an example, see figure 8.

8.1 The System Model

The model in figure 8 represents a system that receives four data packets and processes them, this behaviour is repeated indefinitely.

In more detail, an arrival is modelled by action a, each data packet can be error-free, arrival rate λ, or erroneous, arrival rate μ. An erroneous data packet can be corrected (co, γ), or can not be corrected, (e, δ). If it can not 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 date can be processed (prc, ω) and the system awaits new data.

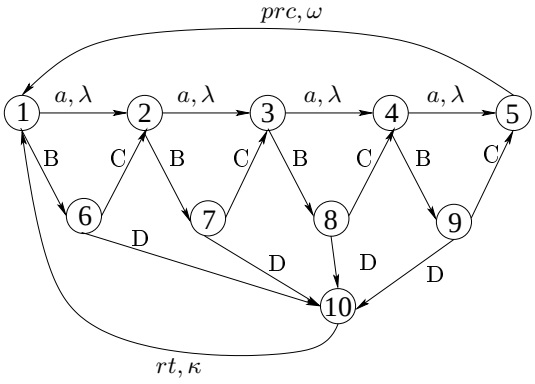

Fig. 8. System model - A 4-place buffer with erroneous arrivals

8.2 Performability Measures

To fully exploit the power of SPDL in defining performability measures we have to provide some details about the state labellings, i.e. formulae that are valid in the states of the system model. The example system has 10 states, enumerated 1 to 10.

- State 1: {empty}
- State 5: {full}
- State 6-10: {error}

Now, we will give some example measures:

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 4 data packets without error or with at most one non-correctable error within 5 time units greater than 0.9?
2. $\Phi_2 := \mathcal{P}_{>0}(\neg\mathsf{full}[a]^{[0,\infty)}\mathsf{full})$: 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 the only state, state 5, for which it is true that the buffer is full.
3. $\Phi_3 := \mathcal{P}_{\leq 0.1}(\mathsf{true}[a^*; (\Phi_2?; a; (co \cup e))]^{[0,\infty)}\mathsf{true})$: Is the probability that the fourth packet contains an error, correctable or incorrectable, at most ten percent, given that all preceedings packets were error-free?
4. $\Phi_4 := \mathcal{P}_{\leq 0.85}((\neg\mathsf{full})[(a \cup a; co)^*]^{[0,10]}(\mathsf{full}))$: Is the probability to reach state 5 within 10 time units, provided no packet contains incorrectable errors, at least 85 %?
5. $\Phi_5 := \mathcal{P}_{\leq 0.75}(\mathsf{true}[a^*; \Phi_2?; a; (e \cup co); (\mathsf{error}?; rt; a^* \cup \mathsf{full}?)]^{[0,25]}\mathsf{true})$: Is the probability to reach state 5 within 25 time units, given the only erroneous packet arrived was the 4th one and either the packet contains a correctable or incorrectable error, at most 75 percent?
6. $\Phi_6 := \mathcal{P}_{\leq 0.01}(\mathsf{true}[a^*; \Phi_2?; a; co]^{[0,7.3]}\mathsf{true})$: Is the probability that the buffer is full after at most 3 time units and that the 4th packet contains a correctable error, given that all preceeding packets were error free, at most one percent?

8.3 Building the Product Automaton

Consider the example system $\mathcal{M}$, from figure 8 and the requirement $\varphi := \mathcal{P}_{\leq 0.01}(\mathsf{true}[a^*; \Phi_2?; a; co]^{[0,7.3]}\mathsf{true})$

We want to check whether $\mathcal{M}$ satisfies Φ_3, provided the system starts in state 1. At first, we derive from $a^*; (\Phi_2?; a; co$ a non-deterministic automaton [6] N_ρ (cf. figure 9).

The test Φ_2 forms together with a a single transition. Now, we have to transform N_ρ into a determininistic automaton A_ρ (cf. figure 10). In figure 10

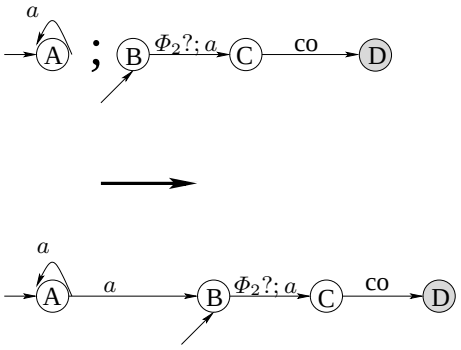

Fig. 9. Non-deterministic automaton N_ρ for $a^*; \Phi_2?; a; co$

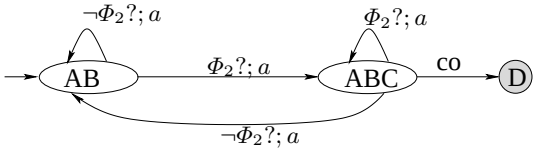

Fig. 10. Deterministic automaton A_ρ for $a^*; \Phi_2?; a; co$

we see that the labels of the transitions emanating from state AB are labelled with $\neg\Phi_2?; a$ resp. $\Phi_2?; a$. (cf. figure 11). 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.

We assign the following numerical values to the rates:

$$\lambda := 0.4 : \mu := 0.4 : \gamma := 0.2 : \omega := 0.2 : \delta := 0.001;$$

Assuming a precision of $\epsilon = 10^{-6}$ we obtain after 7 computation steps that this property is violated, since after 7 computation step the probability to be in state SUCC equals 0.011786.

[6] Grey-shaded states indicate the accepting end states

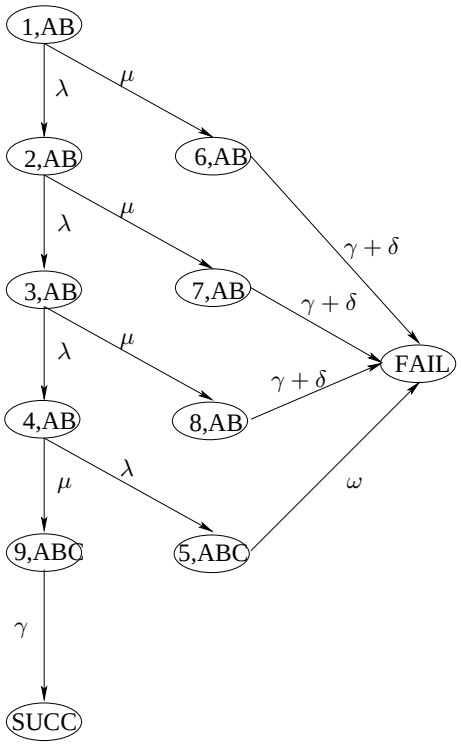

Fig. 11. Product automaton $\mathcal{M}^{\times} := \mathcal{M} \times A_{\rho}$

9 Bisimulation and Worst Case Complexity

In this short section we will prove that a variant of the well-known Markov bisimulation preserves the validity of SPDL formulae and give a worst case approximation of model checking probabilistic SPDL path formulae.

9.1 SPDL and Bisimulation

The chosen variant of Markov bisimulation accounts for the fact that beside rate information also action labels and state labels, i.e. state formulae have to be taken into account to identify states as being equivalent or not equivalent, we call this bisimulation relation Markov-AP-bisimulation. We will show by induction the claimed property, i.e. the preservation of the validity of SPDL formulae under Markov-AP-bisimulation.

Definition 25 (Markov-AP-Bisimulation). Let $\mathcal{M} = (S, A, L, R)$ be an action- and state-labelled CTMC An equivalence relation $\mathcal{B}$ on S is a Markov-AP-bisumulation over $\mathcal{M}$, if for all $(s, s') \in \mathcal{B}$ it holds that:

1. $L(s) = L(s')$
2. $\forall C \in S/\mathcal{B}\ \forall a \in A(\mathbf{R}_a(s, C) = \mathbf{R}_a(s', C))$

Where:

- $S/\mathcal{B} = \{C_1, ..., C_n\}$ is the partition of S into equivalence classes C_i induced by $\mathcal{B}$.
- Two states s and s' are called Markov-AP-bisimilar, if there is a Markov-AP-Bisimulation that contains both states.

Theorem 5. *Let $\mathcal{B}$ be a Markov-AP-Bisimulation, $s \in \mathcal{M}$, then we have:*

1. $\forall \Phi(\mathcal{M}, s \models \Phi \iff \mathcal{M}/\mathcal{B}, [s] \models \Phi$
2. $\forall \phi(Prob^{\mathcal{M}}(s, \phi) = Prob^{\mathcal{M}/\mathcal{B}}([s], \phi)$

Let $\phi = \Phi[\pi]^{\leq t}\Psi$ and ρ be an SPDL-program.

The proof is a structural induction over the length k of formula Φ

Proof (Theorem 5). We start with formulae of length one, i.e. with atomic formulae. Let the states s and t be in $\mathcal{B}$:

1. Let $\Phi \in \mathsf{AP}$, d.h. $\Phi = q$: Using the preriquisites it holds: $(s, t) \in \mathcal{B}$. Following the definition of $\mathcal{B}$ we can conclude: $\mathcal{M}, s \models q \iff \mathcal{M}, t \models q$.

This case serves as induction start. As induction hypothesis we assume that the proposition holds for formulae of length $< k$.

2. Let $\Phi = \neg\Psi$. Following **I.H.** it is true that $\mathcal{M}, s \models \Psi \iff \mathcal{M}, t \models \Psi$, we are able to prove:

$$\mathcal{M}, s \models \neg\Psi \overset{Sem.}{\iff} \mathcal{M}, s \not\models \Psi \overset{I.H.}{\iff} \mathcal{M}, t \not\models \Psi \overset{Sem.}{\iff} \mathcal{M}, t \models \neg\Psi$$

3. Let $\Phi = \Psi \vee \Xi$: Following **I.H.** gilt $\mathcal{M}, s \models \Psi \iff \mathcal{M}, t \models \Psi$ and $\mathcal{M}, s \models \Xi \iff \mathcal{M}, t \models \Xi$. Thus, we have as well:

$$\mathcal{M}, s \models \Psi \vee \Xi \overset{Sem.}{\iff} \mathcal{M}, s \models \Psi \text{ oder } \mathcal{M}, s \models \Xi$$
$$\overset{I.H.}{\iff} \mathcal{M}, t \models \Psi \text{ oder } \mathcal{M}, t \models \Xi \overset{Sem.}{\iff} \mathcal{M}, t \models \Psi \vee \Phi$$

4. Let $\Phi = \mathcal{P}_{\bowtie p}(\Delta[\rho]^{\leq t}\Psi)$. For the proof of this case we have to demonstrate that the probability measures of the paths satisfying path formula ϕ and emanating from s and t are identical. This requires an induction over the length n of the paths.

   (a) Let the length $n = 0$: The probability measure of a path satisfying ϕ is either zero or one.

      i. Let the measure be one: This is the case iff $\mathcal{M}, s \models \Psi \wedge z_\rho \in E$ or $\mathcal{M}, s \models \Psi \wedge \mathcal{M}, s \models \Xi \wedge \rho = \Xi? \wedge \delta(z_\rho, \Xi?) \in E$. Using the outer **I.H.** these assumptions are also valid for state t.

      ii. Let the measure be zero: This is the case iff $\mathcal{M}, s \not\models \Psi$ or $\mathcal{M}, s \models \Psi \wedge \mathcal{M}, s \not\models \Xi \wedge \rho = \Xi? \wedge z_\rho \not\in E$. Also in this case we can apply the outer induction hypothesis to exchange s and t.

   (b) Let $n \neq 0$: We assume that the claim holds for paths of length $< n$. We also assume that the length of the paths is minimal, i.e. for no path which is shorter than the considered length path formula ϕ is satisfiable. Let $n \neq 0$: We assume the assumption holds for paths of length $< n$.

$$Prob(\sigma \in PFAD(s) \big| \sigma \models \phi) =$$

$$\int_0^t e^{\mathbf{E}(s)\cdot x} \sum_{a \in L(z_\rho)} \sum_{s \in S'} \mathbf{R}_a(s, s') \cdot Prob(\sigma \in PFAD(s') \big| \sigma \models \phi) =$$

$$\int_0^t e^{\mathbf{E}(s)\cdot x} \sum_{a \in L(z_\rho)} \sum_{C \in \mathcal{M}/\mathcal{B}} \sum_{s \in C} \mathbf{R}_a(s, s') \cdot Prob(\sigma \in PFAD(s') \big| \sigma \models \phi) \stackrel{*}{=}$$

$$\int_0^t e^{\mathbf{E}(t)\cdot x} \sum_{a \in L(z_\rho)} \sum_{C \in \mathcal{M}/\mathcal{B}} \sum_{s \in C} \mathbf{R}_a(t, s') \cdot Prob(\sigma \in PFAD(s') \big| \sigma \models \phi) =$$

$$\int_0^t e^{\mathbf{E}(t)\cdot x} \sum_{a \in L(z_\rho)} \sum_{s \in S'} \mathbf{R}_a(t, s') \cdot Prob(\sigma \in PFAD(s') \big| \sigma \models \phi) =$$

$$Prob(\sigma \in PFAD(t) \big| \sigma \models \phi)$$

$\stackrel{*}{=}$ this is true due to the prerequisite that $(s, t) \in \mathcal{B}$ and the resulting fact that if s and t are in $\mathcal{B}$ they have the same cumulative rates for every target equivalence class C.

5. Let $\Phi = \mathcal{S}_{\bowtie p}(\Psi)$. It remains to show that $\pi(s, Sat(\Psi)) = \pi(t, Sat(\Psi))$:

$$\pi(s, Sa(\Psi)) =$$
$$\sum_{s' \in S} \pi(s, \{s'\}) = \sum_{\substack{C \models \Psi \\ C \in \mathcal{M}/\mathcal{B}}} \sum_{s' \in C} \lim_{t' \to \infty} Pr(\sigma \in PFAD(s) \big| \sigma @ t' = s') =$$
$$\sum_{\substack{C \models \Psi \\ C \in \mathcal{M}/\mathcal{B}}} \sum_{s' \in C} \lim_{t' \to \infty} Pr(\sigma \in PFAD(t) \big| \sigma @ t' = s') = \sum_{s' \in S} \pi(t, \{s'\}) =$$
$$\pi(t, Sa(\Psi))$$

9.2 Complexity Analysis

Theorem 6 (Worst Case Complexity of SPDL Model Checking). *For an action- and state labelled Markov chain $\mathcal{M}$ and an SPDL formula Φ, the time and memory complexity of the model checking procedure lies in:*

$$O(|\Phi| \cdot \left((2^n - n) \cdot 2^Z \cdot N\right) \cdot q \cdot t_{max} + (((2^n - n) \cdot 2^Z \cdot N)^{2.81})$$

where $N = |S|$ is the number of states, q the largest transition rate, t_{max} the maximum time bound of $\mathcal{M}$. $(2^n - n) \cdot 2^Z$ is the number of states of the DPA, where this numbers depend on both the length of the longest program appearing in any of the subformulae of Φ, and the maximum number of ambiguous tests that emanate from a state in the NPA.

Proof. Let ρ be the longest program in any of the subformulae of Φ, and $|\rho| = \rho_m$, then an NPA with at most $Z = 2 \cdot \rho_M$ states can be constructed. The corresponding DPA has at most $(2^n - n) \cdot (2^Z)$ states, where $Z = 2^{2 \cdot \rho_M}$. $2^n - n$ stems from the fact that we have to determinise the automaton with respect to ambiguous tests, n is the maximum number of ambiguous tests emanating from a state of the NPA, thus $2^n - n$ new states have to be added. $(2^n - n) \cdot (2^Z) \cdot N$ is the maximum number of states the product Markov chain $\mathcal{M}^\times$ thus may have. The rest follows from [3].

10 Extending SPDL with Real Time Intervals

For the sake of completeness we define the syntax of $SPDL^I$.

Definition 26 (Syntax of $SPDL^I$). Let $p \in [0, 1]$, $q \in AP$ and $\bowtie \in \{<, >, \leq, \geq\}$. SPDL state formulae can be defined by the following grammar:

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

Path formulae ϕ are defined as follows:

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

where $I = [t, t']$, $t \in I\!R_0 \wedge t' \in I\!R_{>0}$. Programs are defined by the following grammar:

$$\rho := \epsilon \big| \rho; \rho \big| \rho \cup \rho \big| \Phi?; \rho \big| \rho_1 \big| (\rho)$$
$$\rho_1 := a \big| \rho_1; \rho_1 \big| \rho_1 \cup \rho_1 \big| \rho_1^* \big| \Phi?; \rho_1 \big| (\rho_1)$$

We have to adapt the original definition of words over paths to the needs of $[t, t']$ time intervals.

Definition 27 (Words on paths). The word $\mathcal{W}^k$ of length k, $k \geq 0$ over a path $\sigma \in PATH$ of length k, $k \geq 0$ over a path $\sigma \in PATH$ is inductively defined as follows:

$$\mathcal{W}^0(\sigma) = \epsilon$$
$$\dots$$
$$\mathcal{W}^k(\sigma) = \mathcal{W}^{k-1}(\sigma) \circ a[k-1]$$

with:

$$a[k-1] \in A \wedge \sigma[k-1] \xrightarrow{a[k-1], t_{k-1}} \sigma[k]$$

$$\mathcal{W}^k(\sigma)@t_i = p@t_i$$

is the action on path σ belonging to the last transition, being terminated on time point t_i.

Definition 28 (Semantics of path formulae).

$\mathcal{M}, \sigma \models \Phi_{SPDL^I}[\rho]^I\Psi_{SPDL^I} \iff$
$\exists t_2 \in I((\mathcal{M}, \sigma@t_2 \models \Psi_{SPDL^I}) \wedge \forall t_1 \in [t, t_2)(\mathcal{M}, \sigma@t_1 \models \Phi_{SPDL^I}) \wedge$
$\exists p \in {}_\rho(\forall t_1 \in [t, t_2](|p| = |\sigma@t_2| \wedge \mathcal{M}, \sigma@t_1 \models TeF(p@t_1) \wedge \mathcal{W}^{|\sigma|}(\sigma)@t_1 = Act(p@t_1))))$

where $|\sigma@t_2|$ is defined as the length of the path at time point t_2.

10.1 Characterisation of Path Formulae by Integral Equations

We now have four cases, the notation is the same as in section 6.2.

$$W(s, \Phi[\rho]^{[t,t']}\Psi, z_\rho) = \begin{cases} 1 & \Longleftrightarrow t = 0 \wedge (\mathcal{M}, s \models \Psi \wedge z_\rho \in E_\rho) \text{ or} \\ & \quad (\mathcal{M}, s \models \Psi \wedge \mathcal{M}, s \models \Xi \wedge \\ & \quad \delta_\rho(z_\rho, \Xi?;) \in E_\rho) \\ e^{-\mathbf{E}(s)\cdot t} + \int_0^t e^{-E(s)\cdot x} \cdot \sum_{a \in A(z_\rho)} \cdot \sum_{s' \in S} \mathbf{R}_a(s, s') \cdot \\ W(s, \Phi[\rho']^{<I \ominus x}\Psi, \delta_\rho(z_\rho, a))dx & \Longleftrightarrow \mathcal{M}, s \models \Phi \wedge \Psi, t > 0 \\ \int_0^{t'} e^{-E(s)\cdot x} \cdot \sum_{a \in A(z_\rho)} \cdot \sum_{s' \in S} \mathbf{R}_a(s, s') \cdot \\ W(s, \Phi[\rho']^{<I \ominus x}\Psi, \delta_\rho(z_\rho, a))dx & \Longleftrightarrow \mathcal{M}, s \models \Phi \wedge \neg\Psi \\ 0 & \Longleftrightarrow (\mathcal{M}, s \models (\neg\Phi \wedge \neg\Psi)) \text{ or} \\ & \quad (\mathcal{M}, s \models (\neg\Phi \wedge \Psi) \wedge \\ & \quad \neg\exists \Xi \in Sat(s)(\delta_\rho(z_\rho, \Xi?) \in \delta_\rho) \wedge \\ & \quad \delta_\rho(z_\rho, \Xi?) \in E_\rho) \text{ or} \\ & \quad (\mathcal{M}, s \models (\Phi \wedge \Psi) \wedge \mathcal{M} \models \neg\Xi \wedge \\ & \quad \delta_\rho(z_\rho, \Xi?) \wedge L(z_\rho) = \{\Xi?\}) \end{cases}$$

Except for the second case this characterisation is identical to the one in section 6.2. The second case has the following explanation:

> The probability to fulfill φ is the probability to stay for more than $t > 0$ time units in state s plus the probability to reach s' from s within x time units, $x \leq t$ and to satisfy $\Phi[\rho']^{I \ominus x}\Psi$ along a path starting in s'.

11 Conclusions

In this technical report, we have presented the definition of a powerful stochastic logic that is capable of concisely and precisely expressing a rich variant of performability measures. We have also devised an algorithm for model

checking probabilistic SPDL path formulae by transforming the underlying model and thereby reducing it to the model checking problem of CSL, for which solution efficient algorithms exist. Furthermore we have proven some theoretical results on SPDL.

For the future we plan to integrate a stochastic model checking engine into our symbolic performance evaluation tool CASPA [15]. On the theoretical side we plan to extend SPDL by random time intervalls, such that the upper bound is not necessarily longer a fixed value but can be drawn from an arbitrary probability dsitribution. Further, we want to allow immediate transitions, here, we expect various changes in the model checking procedure, as now untimed transitions in the model can be matched by transitions in the automata. Additionally we want to compare the expressive powers of various stochastic logics with that of SPDL.

References

1. 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.
2. A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model checking continous time Markov chains. *ACM Transactions on Computational Logic*, 1(1):167–170, 2000.
3. 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.
4. C. Baier, B.R. Haverkort, J.-P. Katoen, and H. Hermanns. Model Checking Continuous-Time Markov Chains by Transient Analysis. In E.A. Emerson and A.P. Sistla, editors, *Computer Aided Verification*, volume LNCS 1855, pages 358–372. Springer, 2000.
5. 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.
6. G. Evans. *Practical Numerical Analysis.* Wiley, 1995.
7. A. Fantechi, S. Gnesi, and G. Ristori. Model checking for action based logics. *Formal Methods in System Design*, 4:187–203, 1994.
8. M. Fischer and R. Ladner. Propositional Dynamic Logic of Regular Programs. *Journal of Computer and System Sciences*, 1979.
9. B.L. Fox and P. W. Glynn. Computing Poisson probabilities. *Communications of the ACM*, 31(4):440–445, 1988.
10. H. Hansson and B. Jonsson. A logic for reasoning about time and reliability. *Formal Aspects of Computing*, 6:512–535, 1994.
11. D. Harel, D. Kozen, and J.Tiuryn. *Dynamic Logic.* Cambridge University Press, 2001.
12. 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.
13. Joost-Pieter Katoen, Marta Kwiatkowska, Gethin Norman, and David Parker. Faster and symbolic CTMC model checking. In *Process Algebra and Probabilistic Methods*, volume LNCS 2165, pages 23–38, 2001.

14. D. Kozen and J. Tiuryn. *Handbook of Theoretical Computer Science, Volume B: Formal Models and Semantics*, chapter Logic of Programs. Elsevier, 1990.
15. M. Kuntz, M. Siegle, and E. Werner. CASPA a performance evaluation tool based on stochastic process algebra and symbolic data structures. In *to appear*, 2003.
16. J. Meyer-Kayser. Verifikation stochastischer, prozessalgebraischer Modelle mit aCSL+. Technical Report IB 01/03, Universität Erlangen-Nürnberg, Institut für Informatik 7, 2003.
17. C. Moler and C.F. van Loan. Nineteen dubious ways to compute the exponential of a matrix. *SIAM Review*, 20(4):801–835, 1978.