Model Checking Action- and State-Labelled Markov Chains

Christel Baier
Universität Bonn
Institut für Informatik
baier@cs.uni-bonn.de

Lucia Cloth, Boudewijn Haverkort
University of Twente
Department of Computer Science
[lucia,brh]@cs.utwente.nl

Matthias Kuntz, Markus Siegle
University of the Federal Armed Forces Munich
Department of Computer Science
[kuntz,siegle]@informatik.unibw-muenchen.de

Abstract

In this paper we introduce the logic asCSL*, an extension of continuous stochastic logic (*CSL*), which provides powerful means to characterise execution paths of action- and state-labelled Markov chains. In* asCSL*, path properties are characterised by regular expressions over actions and state-formulas. Thus, the executability of a path not only depends on the available actions but also on the validity of certain state formulas in intermediate states. Our main result is that the model checking problem for* asCSL *can be reduced to* CSL *model checking on a modified Markov chain, which is obtained through a product automaton construction. We provide a case study of a scalable cellular phone system which shows how the logic* asCSL *and the model checking procedure can be applied in practice.*

1. Introduction

It becomes increasingly important to assert that distributed hardware and software systems are working correctly and that they meet high performance and dependability constraints. To reason about stochastic phenomena, several high-level models such as stochastic Petri nets, stochastic process algebras, queueing networks, etc., have been established. Typically, the verification of quantitative properties relies on a transformation of these high-level models into a Markov chain, on which the actual analysis is carried out. For the model-based verification of functional properties, temporal logics provide powerful means to specify complex requirements that a system has to satisfy, see e.g. [1]. Over the past 10 years, several researchers have adapted the temporal-logical approach to reason about probabilistic properties. One result of these efforts is the logic CSL (continuous stochastic logic), introduced in [2] and extended in [3], which is a continuous-time variant of PCTL (probabilistic computation tree logic) [4] and can be used as specification formalism for performance and dependability properties. For instance, the CSL-formula $\mathscr{P}_{\geq 0.99}(\mathsf{legal}\,\mathscr{U}^{\leq 5}\mathsf{goal})$ specifies the state-property asserting that "there is at least a 99% chance to reach a goal state within the next 5 time units while passing only legal states before". The goal states and legal states can be formalized, e.g., by atomic propositions that are attached to the states or by complex CSL-formulas. A so-called steady-state operator allows to reason about stationary probabilities. Formula $\mathscr{S}_{\geq 0.75}(\mathsf{green})$ states that the probability mass accumulated in green states is at least 75%. An extension of CSL to reason about rewards has been introduced in [5]. However, the specification of these measures is completely state-oriented. For action-oriented modelling formalisms, such as stochastic process algebra, CSL is not expressive enough, since it is not possible to characterise sequences of actions. In [6] an action-based variant of CSL, aCSL, was proposed, and in [7] it was shown how to employ this logic for performability modelling. A further step is the logic aCSL+ [8], which combines state-oriented and action-oriented features, and employs regular expressions for characterising more general path-based properties.

In this paper we propose a new logic, called asCSL (for action- and state-oriented CSL), which subsumes all the above mentioned logics. (Preliminary work on logics similar to asCSL has been published in [9] and [10].) asCSL can be seen as being motivated either by the method of path-based reward variables as described in [11], or by the propositional dynamic logic [12] and extended linear temporal logic [13]. With asCSL, paths are characterised by regular expressions, also called programs, but in addition it is possible to express that a program is executable only if

the current state satisfies a given state property. This makes it possible to combine in an elegant way state- and action-oriented behavior. Unlike extended linear temporal logic [13], we do not allow for ω-regular expressions (representing infinite behaviors). Instead, in asCSL the regular expressions are used in combination with lower and upper time bounds. Thus, the switch from CSL to asCSL is orthogonal to the extension PCTL* [14] of PCTL which is concerned with specifying "complex" properties of infinite computations, whereas asCSL focusses on "complex" properties of finite computations with real-time constraints (e.g. deadlines).

This paper is further organised as follows: In Section 2 we define action- and state-labelled Markov chains. Section 3 presents syntax and semantics of the new logic asCSL. Section 4 is dedicated to the model checking procedure for asCSL. In Section 5 we apply the new technique to derive properties of the handover procedure in a cellular radio network. The paper ends with a short summary and conclusions.

2. Action and state labelled Markov chains

In this section we explain the notation for Markov chains with action and state labels. The reader is supposed to be familiar with Markov chains, c.f. [15]. Action names are used to label the transitions. The state labels are formalized by a set AP of atomic propositions, which e.g. can assert that the system (or one of its subprocesses) is at a certain control point or that a program variable has a certain value, and a function L that assigns to any state the set of atomic propositions that are assumed to hold in the given state.

Definition 1 (Action- and state-labelled Markov chain) *An action- and state-labelled continuous-time Markov chain (ASMC) is a tuple $\mathscr{M} = (S, \mathsf{Act}, \mathsf{AP}, L, \mathbf{R})$, where S is a finite set of states,* Act *is a finite set of action labels,* AP *is a finite set of atomic propositions, $L : S \to 2^{\mathsf{AP}}$ a state labelling function, and $\mathbf{R} : S \times \mathsf{Act} \times S \to \mathbb{R}_{\geq 0}$ a rate matrix. We require that for any state s there exists a pair $(a, s') \in \mathsf{Act} \times S$ with $\mathbf{R}(s,a,s') > 0$.* □

Intuitively, if $\mathbf{R}(s,a,s') = \lambda > 0$ then there is an a-labelled transition from state s to state s' whose delay is specified by an exponential distribution with rate λ. For $S' \subseteq S$ we write $\mathbf{R}(s,a,S') = \sum_{s' \in S'} \mathbf{R}(s,a,s')$ to denote the total rate to move from state s via action a to state-set S'.

Definition 2 (Paths in $\mathscr{M}$) *A finite path in $\mathscr{M}$ is a finite word $\sigma = (s_0,a_0,t_0),(s_1,a_1,t_1), \ldots, (s_{n-1},a_{n-1},t_{n-1}), s_n \in (S \times \mathsf{Act} \times \mathbb{R}_{>0})^* \times S$, where $\mathbf{R}(s_i,a_i,s_{i+1}) > 0$ for $i = 0,\ldots,n-1$. An infinite path in $\mathscr{M}$ is an infinite word $\varsigma = (s_0,a_0,t_0),\ (s_1,a_1,t_1),\ \ldots \in (S \times \mathsf{Act} \times \mathbb{R}_{>0})^\omega$ with $\mathbf{R}(s_i,a_i,s_{i+1}) > 0$ for all $i \geq 0$ and such that the infinite series $\sum_i t_i$ is divergent (i.e., $t_0 + t_1 + t_2 + \ldots = \infty$).* □

We write paths as sequences of transitions, e.g., for the finite path σ above, we use the notation

$$\sigma \;=\; s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} s_2 \xrightarrow{a_2,t_2} \cdots \xrightarrow{a_{n-2},t_{n-2}} s_{n-1} \xrightarrow{a_{n-1},t_{n-1}} s_n.$$

Let σ be a finite path as before, then $|\sigma| = n$ denotes the length of σ, i.e., number of transitions in σ, $\sigma[i] = s_i$ the $(i+1)$st state on σ. We refer to $\tau(\sigma) = \sum_{j=0}^{n-1} t_j$ as the execution time of σ. For $t \leq \tau(\sigma)$, $\sigma@t$ denotes the state that is occupied at time t on path σ, that is, $\sigma@t = \sigma[k]$, where k is the smallest index for which $t < \sum_{j=0}^{k} t_j$. We write $\sigma(i,j)$ to denote the fragment of path σ starting at the $(i+1)$-st state s_i and ending at the $(j+1)$-st state s_j $(i \leq j)$.

If σ_1, σ_2 are finite paths such that the first state of σ_2 agrees with the last state of σ_1 then the concatenation $\sigma_1\sigma_2$ is a path of length $|\sigma_1| + |\sigma_2|$ which is defined in the obvious way. Similar notation is used for infinite paths. $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}$ denotes the set of all finite paths in $\mathscr{M}$, whereas $\mathsf{Path}^{\mathscr{M}}_{\omega}$ stands for the set of infinite paths in $\mathscr{M}$. By $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(s)$, resp. $\mathsf{Path}^{\mathscr{M}}_{\omega}(s)$, we denote the set of all finite, resp. infinite, paths in $\mathscr{M}$ with initial state s.

In the sequel, we shall deal with the standard probability measure $\Pr^{\mathscr{M}}_s$ on $\mathsf{Path}^{\mathscr{M}}_{\omega}(s)$ (where the underlying σ-field can be defined with the help of basic cylinders as in [16]). For measurable $X \subseteq \mathsf{Path}^{\mathscr{M}}_{\omega}(s)$, we often skip the parameter $\mathscr{M}$ and/or s and simply write $\Pr(X)$ or $\Pr^{\mathscr{M}}(X)$. The transient state probabilities $\pi^{\mathscr{M}}(s,s',t)$ are given by

$$\pi^{\mathscr{M}}(s,s',t) \;=\; \Pr^{\mathscr{M}}\{\varsigma \in \mathsf{Path}^{\mathscr{M}}_{\omega}(s) : \varsigma@t = s'\},$$

and the steady state probability of being in state s' on the long run, provided that the system started in state s, is $\pi^{\mathscr{M}}(s,s') = \lim_{t\to\infty} \pi^{\mathscr{M}}(s,s',t)$. For $S' \subseteq S$, we put $\pi^{\mathscr{M}}(s,S') = \sum_{s' \in S'} \pi^{\mathscr{M}}(s,s')$.

3. Syntax and semantics of asCSL

We now extend CSL (continuous stochastic logic) as introduced in [2, 3] by path formulas that specify constraints for action- and state-sequences in combination with lower and/or upper time bounds.

3.1. Syntax of asCSL

The syntax of asCSL is defined according to Def. 3 (state formulas) and Def. 4 (programs or path formulas). Here, we assume that the sets Act of actions and AP of atomic propositions are fixed.

Definition 3 (State formulas of asCSL) *State formulas of* asCSL *are given by the following grammar:*

$$\phi \;::=\; \mathsf{q} \;\Big|\; \neg\phi \;\Big|\; \phi \vee \phi \;\Big|\; \mathscr{S}_{\bowtie p}(\phi) \;\Big|\; \mathscr{P}_{\bowtie p}(\alpha^I)\,,$$

where $\mathsf{q} \in \mathsf{AP}$ *is an atomic proposition,* $p \in [0,1]$ *denotes a probability value,* $\bowtie \in \{<, \leq, >, \geq\}$ *a comparison operator,* $I = [t,t'] \subseteq \mathbb{R}_{\geq 0}$ *a time interval and* α *a program as defined in Def. 4. We refer to* α^I *as a path-formula and use* Φ *to denote the set of state formulas of* asCSL. □

The logical connectives $\neg$ and $\vee$ have their usual meaning. Using negation $\neg$ and disjunction $\vee$, the constants true, false and all other boolean connectives such as conjunction $\wedge$, implication $\rightarrow$, etc. can be derived. The so-called steady-state operator $\mathscr{S}_{\bowtie p}(\phi)$ asserts that the probability of being in a ϕ-state on the long run is within the interval $\bowtie p$. The operator $\mathscr{P}_{\bowtie p}(\alpha^I)$ asserts that the probability measure of all infinite paths which have a prefix that satisfies α^I, where α is a program and I is a real interval specifying the time bound, is within the interval $\bowtie p$.
The program α specifies a property for finite paths via a regular set of finite words whose atomic symbols are pairs (ϕ, b) (in the sequel we simply write ϕb) consisting of an asCSL-state formula ϕ (which is viewed as a test for the current state of a path) and an action $b \in \mathsf{Act}$ or $b = \surd$ (where $\surd \notin \mathsf{Act}$). The symbol $\surd$ can be viewed as a pseudo-action which is always immediately executable and does not change the current state in the ASMC. Formally, the programs are regular expressions over the alphabet

$$\Sigma = \Phi \times (\mathsf{Act} \cup \{\surd\}) \;=\; \big\{\phi b \;\big|\; \phi \in \Phi \wedge b \in (\mathsf{Act} \cup \{\surd\})\big\}.$$

Definition 4 (Programs) asCSL*-programs are defined by the following grammar:*

$$\alpha \;::=\; \varepsilon \;\Big|\; \phi b \;\Big|\; \alpha;\alpha \;\Big|\; \alpha \cup \alpha \;\Big|\; \alpha^*,$$

where $\phi b \in \Sigma$ *(i.e.,* $\phi \in \Phi$*,* $b \in \mathsf{Act} \cup \{\surd\}$*). The language* $\mathscr{L}(\alpha) \subseteq \Sigma^*$ *is defined in the standard way.* □

The symbol ε stands for the empty word (as an element of Σ^*). The intuitive meaning of ϕb is that the current state s fulfills the test ϕ (that is, the formula ϕ holds in s) and, if $b \in \mathsf{Act}$, state s has an outgoing b-transition. If $b = \surd$, there is no statement about outgoing transitions. Note that the test can be empty, i.e., $\phi = \mathsf{true}$. The operator ; denotes sequential composition (concatenation), $\cup$ denotes alternative choice (union), and * denotes the n-fold sequential composition for arbitrary $n \geq 0$ (Kleene star). As examples for the corresponding languages, $\mathscr{L}\big(\phi_1 a; (\phi_2 \surd \cup \phi_3 b)\big)$ consists of the two words $\phi_1 a, \phi_2 \surd$ and $\phi_1 a, \phi_3 b$ whereas $\mathscr{L}\big((\phi_1 a; \phi_2 b; \phi_3 \surd)^*; \phi_4 c\big)$ is an infinite language that contains the words $\phi_4 c$ and $\phi_1 a,\ \phi_2 b,\ \phi_3 \surd,\ \phi_4 c$ and $\phi_1 a,\ \phi_2 b,\ \phi_3 \surd,\ \phi_1 a, \phi_2 b,\ \phi_3 \surd,\ \phi_4 c$, and so on.
In the context of asCSL, the meaning of a program (which will be formally defined in the next subsection) is a set of finite paths in the underlying ASMC.

3.2. Semantics of asCSL

The semantics of asCSL is provided by means of a satisfaction relation $\models$ for the state- and path formulas. In the sequel, we assume a fixed ASMC $\mathscr{M} = (S, \mathsf{Act}, \mathsf{AP}, L, \mathbf{R})$. For state s in $\mathscr{M}$ and state formula ϕ, $\mathscr{M}, s \models \phi$ means that the state-property specified by ϕ holds for s. Similarly, for an infinite path ς, $\mathscr{M}, \varsigma \models \alpha^I$ denotes that the behaviour specified by the path formula α^I is fulfilled by ς. The formal definition of the satisfaction relation $\models$ for the state and path formulas is by structural induction on the syntax of the formulas.

Definition 5 (Semantics of asCSL**)** *The satisfaction relation* $\models$ *for the state formulas is defined as follows:*

$$\begin{array}{lcl}
\mathscr{M}, s \models \mathsf{q} & \Leftrightarrow & \mathsf{q} \in L(s) \\
\mathscr{M}, s \models \neg\phi & \Leftrightarrow & \mathscr{M}, s \not\models \phi \\
\mathscr{M}, s \models \phi_1 \vee \phi_2 & \Leftrightarrow & \mathscr{M}, s \models \phi_1 \text{ or } \mathscr{M}, s \models \phi_2 \\
\mathscr{M}, s \models \mathscr{S}_{\bowtie p}(\phi) & \Leftrightarrow & \pi^{\mathscr{M}}(s, \mathsf{Sat}^{\mathscr{M}}(\phi)) \bowtie p \\
\mathscr{M}, s \models \mathscr{P}_{\bowtie p}(\alpha^I) & \Leftrightarrow & \mathsf{Prob}^{\mathscr{M}}(s, \alpha^I) \;\bowtie\; p
\end{array}$$

where $\mathsf{Sat}^{\mathscr{M}}(\phi) = \big\{s \in S : \mathscr{M}, s \models \phi\big\}$ *denotes the satisfaction set of* ϕ *in* $\mathscr{M}$ *and*

$$\mathsf{Prob}^{\mathscr{M}}(s, \alpha^I) \;= \mathsf{Pr}^{\mathscr{M}}\big\{\varsigma \in \mathsf{Path}^{\mathscr{M}}_{\omega}(s) \big| \mathscr{M}, \varsigma \models \alpha^I\big\}.$$

The meaning of the path formulas is formalized as follows. If ς *is an infinite path in* $\mathscr{M}$ *then* $\mathscr{M}, \varsigma \models \alpha^I$ *iff there exists a finite prefix* σ *of* ς *with* $\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha)$ *and* $\tau(\sigma) \in I$.[1] *Here, the set* $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha)$ *consists of all finite paths* σ *in* $\mathscr{M}$ *that can be viewed as an instance of* α*. Formally:*

$$\begin{array}{lll}
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\varepsilon) & \textit{iff} & |\sigma| = 0 \\
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\phi a) & \textit{iff} & \exists t > 0 \textit{ s.t. } \sigma = s \xrightarrow{a,t} s' \\
 & & \textit{and } \mathscr{M}, s \models \phi \\
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\phi\surd) & \textit{iff} & \sigma = s \textit{ and } \mathscr{M}, s \models \phi \\
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_1;\alpha_2) & \textit{iff} & \exists i \in \{0,1,\ldots,|\sigma|\} \textit{ s.t.} \\
 & & \sigma(0,i) \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_1) \textit{ and } \sigma(i,|\sigma|) \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_2) \\
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_1 \cup \alpha_2) & \textit{iff} & \sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_1) \cup \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha_2) \\
\sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha^*) & \textit{iff} & \exists i \geq 0 \textit{ s.t. } \sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha^i)
\end{array}$$

where $\alpha^{i+1} = \alpha;\alpha^i$ *and* $\alpha^0 = \varepsilon$ *(the empty word in* Σ^**).*[2] *In the sequel, we often write* $s \models \phi$ *and* $\varsigma \models \alpha^I$ *rather than* $\mathscr{M}, s \models \phi$ *and* $\mathscr{M}, \varsigma \models \alpha^I$ *respectively.* □

We simply write ϕA for $\bigcup_{a \in A} \phi a$. The reader should notice the difference between the programs $\alpha_1 = (\mathsf{true}\,\mathsf{Act})^*\phi\surd$ and $\alpha_2 = (\mathsf{true}\,\mathsf{Act})^*\phi\,\mathsf{Act}$. Program α_1 stands for all finite paths σ that end in a ϕ-state, whereas α_2 stands for all finite paths whose prefinal state satisfies ϕ.
A time-bounded until operator can be derived from the syntax of asCSL by $\phi_1 \mathscr{U}^I \phi_2 = ((\phi_1 \mathsf{Act})^* \phi_2 \surd)^I$. From

[1] Recall that $\tau(\sigma)$ denotes the execution time of σ.
[2] Note that all paths σ with $|\sigma| = 0$ belong to $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha^*)$.

this, we may derive the time-bounded eventually-operator $\Diamond^I\phi = \mathsf{true}\mathscr{U}^I\phi$. For instance, $\mathscr{P}_{\leq 0.05}(\Diamond^{\leq 5}\mathsf{error})$ states that the probability to reach an error state within 5 time units is bounded above by 0.05. Its dual, the time-bounded always-operator, is obtained (as in CSL) through $\mathscr{P}_{\geq p}(\Box^I\phi) = \mathscr{P}_{\leq 1-p}(\Diamond^I\neg\phi)$, stating that ϕ continuously holds in the time interval I with probability at least p.

3.3. asCSL-equivalence and bisimulation

The correctness proof of the model checking algorithm presented in Section 4 will make use of the observation that asCSL-equivalence (which identifies those states that cannot be distinguished by asCSL-formulas) agrees with bisimulation equivalence for ASMCs. The latter can be viewed as a variant of lumpability [17, 18] and essentially agrees with Markovian bisimulation as introduced in [19, 20] for action-labelled Markov chains. Formally, bisimulation equivalence $\sim$ for an ASMC $\mathscr{M} = (S, \mathsf{Act}, \mathsf{AP}, L, \mathbf{R})$ is the coarsest equivalence on S such that for all $s_1 \sim s_2$: (i) $L(s_1) = L(s_2)$ and (ii) $\mathbf{R}(s_1, a, C) = \mathbf{R}(s_2, a, C)$ for all actions $a \in \mathsf{Act}$ and all equivalence classes $C \in S/\sim$.
It is well-known that bisimulation equivalence $\sim$ agrees with CSL-equivalence [16, 21] (which means the equivalence that identifies exactly those states that satisfy the same CSL state formulas). In particular, bisimulation equivalent states have the same transient and steady state probabilities. Using structural induction on the syntax of state- and path-formulas of asCSL the preservation property for asCSL and bisimulation equivalence can be established in the following sense: If $s_1 \sim s_2$ then we have $s_1 \models \phi$ iff $s_2 \models \phi$ for all state-formulas ϕ of asCSL: and $\mathsf{Prob}^{\mathscr{M}}(s_1, \alpha^I) = \mathsf{Prob}^{\mathscr{M}}(s_2, \alpha^I)$ for all path-formulas α^I of asCSL. As asCSL subsumes the logic CSL we obtain by the completeness result established in [21]:

Proposition 1 *Bisimulation equivalence agrees with* asCSL*-equivalence.*

As Markovian testing equivalence [22] is weaker than bisimulation equivalence, states that fulfill the same asCSL-formulas are Markovian testing equivalent.

4. Model Checking asCSL

The model checking procedure for asCSL is similar to that for CTL [23]. Given the asCSL state formula ϕ and an ASMC $\mathscr{M}$, we successively consider the subformulas ψ of ϕ and calculate the satisfaction sets $\mathsf{Sat}^{\mathscr{M}}(\psi) = \{s \in S : \mathscr{M}, s \models \psi\}$. This technique allows us to treat subformulas as atomic propositions. The treatment of subformulas whose top-level operator is a boolean connective (negation or disjunction) is obvious. Subformulas of the form $\mathscr{S}_{\bowtie p}(\phi)$ can be handled with the same procedure as for CSL, see [24]. The new and challenging case is the treatment of formulas of type $\phi = \mathscr{P}_{\bowtie p}(\alpha^I)$. For each state s we have to compute the probability $\mathsf{Prob}^{\mathscr{M}}(s, \alpha^I) = \mathsf{Pr}^{\mathscr{M}}\{\varsigma \in \mathsf{Path}^{\mathscr{M}}_{\omega}(s) \mid \mathscr{M}, \varsigma \models \alpha^I\}$ and check whether it lies within the specified bound $\bowtie p$. The approach to calculate the values $\mathsf{Prob}^{\mathscr{M}}(s, \alpha^I)$ is to build the product of $\mathscr{M}$ and a finite automaton $\mathscr{A}_\alpha$ (representing the program α), which yields a new Markov chain, denoted $\mathscr{M} \times \mathscr{A}_\alpha$, and then to apply the CSL model checking procedure to calculate the probabilities in $\mathscr{M} \times \mathscr{A}_\alpha$ to reach a state $\langle s', q\rangle$, with s' a state in $\mathscr{M}$ and q a final state in $\mathscr{A}_\alpha$, within the time interval I.
Section 4.1 is devoted to the construction of the automaton $\mathscr{A}_\alpha$, whereas Section 4.2 presents the construction of the product Markov chain $\mathscr{M} \times \mathscr{A}_\alpha$.

4.1. Program automata

Since programs are regular expressions, we can apply standard techniques to construct a finite automaton for a given program. We call this a nondeterministic program automaton (NPA).

Definition 6 (NPA) *An NPA is a quintuple $\mathscr{A} = (Z, \Sigma', \delta, Z_0, F)$, where Z is a finite set of states, Σ' a finite subset of Σ (the input alphabet)*[3]*, $\delta : Z \times \Sigma' \to 2^Z$ is the transition function, $Z_0 \subseteq Z$ the set of initial states and $F \subseteq Z$ the set of accepting (final) states. $\mathscr{L}(\mathscr{A}) \subseteq (\Sigma')^*$ denotes the accepted language of $\mathscr{A}$ which is defined in the standard way.* □

We now describe how a program automaton $\mathscr{A}$ can be used to describe the path-set $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha)$ for a program α. Thus, we consider NPAs as *acceptors for finite paths* in $\mathscr{M}$ (rather than as acceptors for finite words over the alphabet Σ'). The intuitive behaviour of an NPA $\mathscr{A}$ for the input path $\sigma = s \xrightarrow{a,t} \sigma'$ is as follows. The automaton starts in one of its initial states $z_0 \in Z_0$. If the current automata-state is z, then $\mathscr{A}$ chooses nondeterministically between one of the outgoing transitions $z \xrightarrow{\phi b} z'$, where $s \models \phi$ and either $b = \surd$ or $b = a$, and then moves to state z'. In the latter case, i.e., if $a = b$, $\mathscr{A}$ proceeds in the same way for state z' and the input path σ'. In the former case, i.e., if $b = \surd$, no input symbol is consumed, i.e., the procedure is repeated with state z' and the input path σ. If there is no outgoing transition from z which can be taken for the input path σ then $\mathscr{A}$ rejects. As soon as $\mathscr{A}$ reaches a final state (a state in F) and the whole input path has been consumed, the automaton accepts.
The formal definition of acceptance for paths is provided by means of runs which are sequences of automaton-states that can be generated by the operational behaviour of $\mathscr{A}$ as sketched above.

[3] Σ is the alphabet of programs as defined in Def. 4.

Definition 7 (Runs in NPAs, accepted paths) *Let $\mathscr{A}$ be a NPA and $\mathscr{M}$ an ASMC as before, $z \in Z$ and σ a finite path in $\mathscr{M}$. Then, we define* $\mathsf{Runs}(z,\sigma)$ *as the greatest set of sequences $z, z_1, \ldots, z_n \in Z^+$ such that the following two conditions are fulfilled:*

(i) $z \in \mathsf{Runs}(z,\sigma)$ *iff* $|\sigma| = 0$

(ii) If $z, z_1, \ldots, z_n \in \mathsf{Runs}(z,\sigma)$ *and* $n \geq 1$ *then there exists* $\phi b \in \Sigma'$ *such that* $z_1 \in \delta(z, \phi b)$, $\sigma[0] \models \phi$ *and*

- *if* $b \in \mathsf{Act}$ *then* $\sigma = s \xrightarrow{b,t} \sigma'$ *with* $z_1, \ldots, z_n \in \mathsf{Runs}(z_1, \sigma')$
- *if* $b = \surd$ *then* $z_1, \ldots, z_n \in \mathsf{Runs}(z_1, \sigma)$.

Let $Z' \subseteq Z$. The elements of $\mathsf{Runs}(Z',\sigma) = \bigcup_{z \in Z'} \mathsf{Runs}(z,\sigma)$ *are called runs for σ in $\mathscr{A}$ with starting state in Z'. A run $z_0, z_1, \ldots, z_n$ for σ is called accepting iff it is initial (i.e., $z_0 \in Z_0$) and $z_n \in F$. The set of accepted paths,* $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A})$*, denotes the set of finite paths in $\mathscr{M}$ that have an accepting run in $\mathscr{A}$.* □

Acceptance of $\mathscr{A}$ as an ordinary finite automaton (acceptor for finite words over Σ') and acceptance of $\mathscr{A}$ as an NPA (acceptor for finite paths) are related in the same way as the language $\mathscr{L}(\alpha)$ of a program α and the induced path-set $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha)$. Hence, it is easy to verify the following proposition:

Proposition 2 *If* $\mathscr{L}(\alpha) = \mathscr{L}(\mathscr{A})$ *then* $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha) = \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A})$.

We now extend the transition function δ of $\mathscr{A}$ to a transition relation $\hat{\delta}^{\mathscr{M}}$ which associates with any pair (Z',σ) consisting of a set Z' of automaton-states and a finite path σ in the ASMC $\mathscr{M}$, the set of automaton states z such that z is the last state of a run for σ that starts in a Z'-state. The idea behind the definition of $\hat{\delta}^{\mathscr{M}}$ is similar to the definition of the transition relation of the deterministic finite automaton obtained from $\mathscr{A}$ (viewed as an acceptor for finite words) via the standard powerset construction. However, the following remark shows that the determinisation process of an NPA as an acceptor for finite paths in $\mathscr{M}$ has to be done "in conjunction" with $\mathscr{M}$.

Remark. An NPA $\mathscr{A} = (Z, \Sigma', \delta, Z_0, F)$ is called deterministic if Z_0 is a singleton set and $|\delta(z,\phi b)| \leq 1$ for all states $z \in Z$ and input symbols $\phi b \in \Sigma'$. Given a deterministic NPA $\mathscr{A}$, the "behaviour" of $\mathscr{A}$ for an *input word* over Σ' is deterministic, i.e., there is at most one run, whereas the "behaviour" of a deterministic NPA $\mathscr{A}$ for an *input path* σ can be *nondeterministic*, even if $\mathscr{A}$ does not contain $\surd$-transitions (i.e., transitions that are labelled with an input symbol $\phi\surd \in \Sigma'$). The reason is that the current automaton-state z might have two transitions $z \xrightarrow{\phi a} z'$ and $z \xrightarrow{\psi a} z''$, where a is the first action of the input path σ and where the first state of σ satisfies both ϕ and ψ. □

We now return to the formal definition of the extended transition function $\hat{\delta}^{\mathscr{M}}$. If σ is a path of length 0, i.e., $\sigma = s$ for some state s, then $\hat{\delta}^{\mathscr{M}}(Z',\sigma) = \hat{\delta}^{\mathscr{M}}(Z',s)$ consists of all automaton-states $\tilde{z}$ that are reachable in $\mathscr{A}$ from a Z'-state via $\surd$-transitions $z_1 \xrightarrow{\phi,\surd} z_2$ where state s fulfills the state formulas ϕ. This corresponds to the so-called $\surd$-closure of Z' for state s which is defined as follows.

Definition 8 ($\surd$-closure) $\surd Closure(Z',s)$ *denotes the least subset of Z such that*

$$Z' \cup \bigcup_{z \in \surd Closure(Z',s)} \bigcup_{\substack{\phi \in \Phi \, s.t. \\ s \models \phi}} \delta(z, \phi\surd) \;\subseteq\; \surd Closure(Z',s).$$

□

We now have all ingredients to define $\hat{\delta}^{\mathscr{M}}(Z',\sigma)$ by induction on the length of σ:

Definition 9 (Extended transition function) *The function $\hat{\delta}^{\mathscr{M}} : 2^Z \times \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}} \to 2^Z$ is given by:*

$$\hat{\delta}^{\mathscr{M}}(Z',s) = \surd Closure(Z',s)$$

and $\hat{\delta}^{\mathscr{M}}(Z', s \xrightarrow{a,t} \sigma') = \hat{\delta}^{\mathscr{M}}(Y,\sigma')$ *where*

$$Y \;=\; \bigcup_{z \in \surd Closure(Z',s)} \bigcup_{\substack{\phi \in \Phi \, s.t. \\ s \models \phi}} \delta(z, \phi a).$$ □

Note that Y stands for the set of all automaton states y that are reachable in $\mathscr{A}$ from a state $z' \in Z'$ via transitions labelled with elements $\psi\surd \in \Sigma$ such that $s \models \psi$ followed by a transition with a label $\phi a \in \Sigma$ such that $s \models \phi$.

It can be shown by induction on $|\sigma|$ that $\hat{\delta}^{\mathscr{M}}(Z',\sigma)$ consists of all states that are reachable in $\mathscr{A}$ via a run starting in Z' for σ, i.e.,

$$\hat{\delta}^{\mathscr{M}}(Z',\sigma) = \big\{ z \in Z \mid \exists z_0, z_1, \ldots, z_n \in \mathsf{Runs}(Z',\sigma) : z_n = z \big\}.$$

For $Z' = Z_0$, we obtain that $\hat{\delta}^{\mathscr{M}}(Z_0,\sigma)$ consists of all automaton-states that can be reached via an initial run for σ. Thus, $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A}) = \big\{ \sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}} : \hat{\delta}^{\mathscr{M}}(Z_0,\sigma) \cap F \neq \emptyset \big\}$. From this observation we obtain:

Proposition 3 *If α is a program and $\mathscr{A}$ an NPA with $\mathscr{L}(\alpha) = \mathscr{L}(\mathscr{A})$ then we have:*

$$\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha) \;=\; \big\{ \sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}} : \hat{\delta}^{\mathscr{M}}(Z_0,\sigma) \cap F \neq \emptyset \big\}.$$

Example 1 We consider the ASMC shown in Fig. 1(a) and the program

$$\alpha \;=\; (\phi\; a)^*; (\psi\; a); ((\mathsf{true}\; c) \cup (\xi\; b))$$

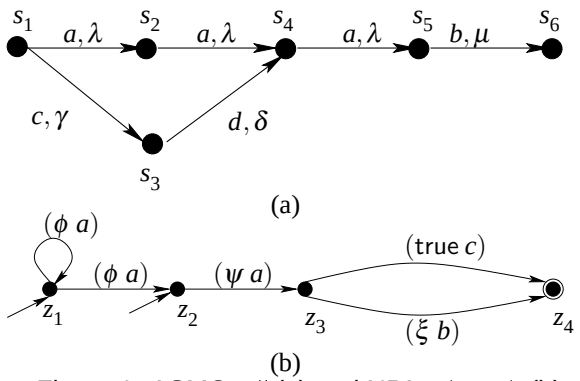


Figure 1. ASMC $\mathscr{M}$ (a) and NPA $\mathscr{A} = \mathscr{A}_\alpha$ (b)

with corresponding program automaton $\mathscr{A} = \mathscr{A}_\alpha$ as shown in Fig. 1(b). For the relevant state formulas we assume $\mathsf{Sat}^{\mathscr{M}}(\phi) = \{s_1, s_2\}$, $\mathsf{Sat}^{\mathscr{M}}(\psi) = \{s_4\}$ and $\mathsf{Sat}^{\mathscr{M}}(\xi) = \{s_5\}$. The finite paths that start in state s_1 and end in state s_6 have the form

$$\begin{aligned} \sigma_1 &= s_1 \xrightarrow{a,\cdot} s_2 \xrightarrow{a,\cdot} s_4 \xrightarrow{a,\cdot} s_5 \xrightarrow{b,\cdot} s_6 \\ \text{or}\quad \sigma_2 &= s_1 \xrightarrow{c,\cdot} s_3 \xrightarrow{d,\cdot} s_4 \xrightarrow{a,\cdot} s_5 \xrightarrow{b,\cdot} s_6, \end{aligned}$$

with arbitrary sojourn times. We then have $\hat{\delta}^{\mathscr{M}}(\{z_1,z_2\},\sigma_1) = \{z_4\}$ and $\hat{\delta}^{\mathscr{M}}(\{z_1,z_2\},\sigma_2) = \emptyset$. Therefore, all paths σ_1 belong to $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A})$ as we have $\hat{\delta}^{\mathscr{M}}(Z_0,\sigma_1) \cap F \neq \emptyset$, whereas none of the paths σ_2 belong to $\mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A})$ because $\hat{\delta}^{\mathscr{M}}(Z_0,\sigma_2) \cap F = \emptyset$. □

4.2. The product Markov chain

We now return to the question of how to calculate the satisfaction set $\mathsf{Sat}^{\mathscr{M}}(\phi)$ where $\phi = \mathscr{P}_{\bowtie p}(\alpha^I)$. We first apply recursively an asCSL-model checking algorithm to the state formulas that occur in the program α. As soon as the satisfaction sets $\mathsf{Sat}^{\mathscr{M}}(\psi)$ are known for all state formulas ψ in α we can treat them as atomic propositions. Then, we apply standard algorithms to construct a (nondeterministic) finite automaton $\mathscr{A}$ for α (viewed as an ordinary regular expression over the alphabet Σ). We then consider $\mathscr{A}$ as an NPA and build the product of the ASMC $\mathscr{M}$ and $\mathscr{A}$ (which is defined below) and finally apply a CSL model checking algorithm to $\mathscr{M} \times \mathscr{A}$ to calculate the probability to reach a final automaton state within the given time interval I.

Definition 10 (Product Markov chain $\mathscr{M}^\times$) *Let $\mathscr{M} = (S, \mathsf{Act}, \mathsf{AP}, L, \mathbf{R})$ be an ASMC and $\mathscr{A} = (Z, \Sigma, \delta, Z_0, F)$ an NPA. The product ASMC is defined as $\mathscr{M} \times \mathscr{A} = (S^\times, \mathsf{Act}^\times, \mathsf{AP}^\times, L^\times, \mathbf{R}^\times)$ where*

$$S^\times = \{\langle s, Z'\rangle \mid s \in S \wedge Z' \in 2^Z\},$$

$\mathsf{Act}^\times = \mathsf{Act}$, $\mathsf{AP}^\times = \mathsf{AP} \cup \{\mathsf{accept}\}$ *(where* $\mathsf{accept} \notin \mathsf{AP}$*) and* $L^\times(\langle s,Z'\rangle) = L(s) \cup \{\mathsf{accept}\}$ *if* $Z' \cap F \neq \emptyset$ *and* $L^\times(\langle s,Z'\rangle) = L(s)$ *otherwise. The rate matrix* $\mathbf{R}^\times$ *is defined by* $\mathbf{R}^\times(\langle s_1,Z_1\rangle, a, \langle s_2,Z_2\rangle) = \mathbf{R}(s_1,a,s_2)$*, if* $Z_2 = \hat{\delta}^{\mathscr{M}}(Z_1, s_1 \xrightarrow{a,\cdot} s_2)$*, and* $\mathbf{R}^\times(\cdot) = 0$ *otherwise.* □

The idea behind the definition of $\mathbf{R}^\times$ is to copy the transitions from $\mathscr{M}$, provided that the corresponding transition is possible in the current set of states of $\mathscr{A}$.

Our goal is to show that the values $\mathsf{Prob}^{\mathscr{M}}(s,\alpha^I)$ can be calculated using a CSL model checking procedure for $\mathscr{M} \times \mathscr{A}$ and the CSL-path formula $\Diamond^I \mathsf{accept}$ (which states that a state labelled with the atomic proposition accept will be reached at some point in the time interval I). To establish this result, we first observe that $\mathscr{M}$ and $\mathscr{M} \times \mathscr{A}$ are state-wise bisimulation equivalent when the set of atomic propositions in $\mathscr{M} \times \mathscr{A}$ is restricted to AP, i.e., we deal with the labeling function $L^\times_{\mathsf{AP}}$ which is given by $L^\times_{\mathsf{AP}}(\langle s,Z\rangle) = L(s)$ rather than $L^\times$. This follows by the fact that the coarsest equivalence $\mathscr{R}$ on $S \uplus (S \times 2^Z)$ which identifies any state s with any of its copies $\langle s,Z'\rangle$ where $Z' \subseteq Z$ is a bisimulation. Hence, $s \sim \langle s,Z'\rangle$ for all states s in $\mathscr{M}$ and all subsets Z' of Z. Using Prop. 1, we obtain:

Proposition 4 *For any state s of $\mathscr{M}$ we have:*

$$\mathsf{Prob}^{\mathscr{M}}(s,\alpha^I) \;=\; \mathsf{Prob}^{\mathscr{M}\times\mathscr{A}}(\langle s,Z_0\rangle, \alpha^I)$$

Next we observe the one-to-one-correspondence between paths in $\mathscr{M}$ and paths in $\mathscr{M} \times \mathscr{A}$ (when we fix the states $\langle s,Z_0\rangle$ as starting states). Clearly, by removing the automaton-component of any state in a path in $\mathscr{M} \times \mathscr{A}$ one obtains a path in $\mathscr{M}$. Vice versa, each finite path

$$\sigma = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{n-1},t_{n-1}} s_n \text{ in } \mathscr{M}$$

can be lifted to a path $\sigma^\times$ in $\mathscr{M} \times \mathscr{A}$ by extending the states by sets of automaton-states with the help of $\hat{\delta}^{\mathscr{M}}$:

$$\sigma^\times = \langle s_0,Z_0\rangle \xrightarrow{a_0,t_0} \langle s_1,Z_1\rangle \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{n-1},t_{n-1}} \langle s_n,Z_n\rangle$$

where $Z_k = \hat{\delta}^{\mathscr{M}}\left(Z_{k-1}, s_{k-1} \xrightarrow{a_{k-1},\cdot} s_k\right)$, $k = 0,1,\ldots,n$. Hence, if $\mathscr{L}(\alpha) = \mathscr{L}(\mathscr{A})$ then (by Prop. 3):

$$\begin{aligned} \langle s_n,Z_n\rangle \models \mathsf{accept} \quad &\text{iff} \quad \mathsf{accept} \in L^\times(\langle s_n,Z_n\rangle) \\ &\text{iff} \quad Z_n \cap F \neq \emptyset \\ &\text{iff} \quad \hat{\delta}^{\mathscr{M}}(Z_0,\sigma) \cap F \neq \emptyset \\ &\text{iff} \quad \sigma \in \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\mathscr{A}) = \mathsf{Path}^{\mathscr{M}}_{\mathsf{fin}}(\alpha). \end{aligned}$$

Thus, for all infinite paths $\varsigma^\times \in \mathsf{Path}^{\mathscr{M}\times\mathscr{A}}_{\omega}(\langle s,Z_0\rangle)$ we have: $\mathscr{M} \times \mathscr{A}, \varsigma^\times \models \Diamond^I \mathsf{accept}$ iff $\mathscr{M} \times \mathscr{A}, \varsigma^\times \models \alpha^I$. Hence, for all states s in $\mathscr{M}$ we have

$$\mathsf{Prob}^{\mathscr{M}\times\mathscr{A}}(\langle s,Z_0\rangle, \alpha^I) \;=\; \mathsf{Prob}^{\mathscr{M}\times\mathscr{A}}(\langle s,Z_0\rangle, \Diamond^I \mathsf{accept}).$$

Using this observation and Prop. 4 we obtain the following theorem.

Theorem 1 *If α is an* asCSL*-program, $\mathscr{A}$ an NPA with $\mathscr{L}(\alpha) = \mathscr{L}(\mathscr{A})$ and s a state in $\mathscr{M}$ then*

$$\mathsf{Prob}^{\mathscr{M}}(s, \alpha^I) \;=\; \mathsf{Prob}^{\mathscr{M}\times\mathscr{A}}(\langle s, Z_0\rangle, \Diamond^I \mathsf{accept}).$$

Theorem 1 shows that the problem of computing the satisfaction set $\mathsf{Sat}^{\mathscr{M}}(\phi)$ for the asCSL-formula $\phi = \mathscr{P}_{\bowtie p}(\alpha^I)$ is reducible to the problem of calculating the satisfaction set $\mathsf{Sat}^{\mathscr{M}\times\mathscr{A}}(\phi_{\mathsf{CSL}})$ for the CSL-state formula $\phi_{\mathsf{CSL}} = \mathscr{P}_{\bowtie p}(\Diamond^I \mathsf{accept})$. In summary, to calculate $\mathsf{Sat}^{\mathscr{M}}(\phi)$ where ϕ is as above we

- apply standard techniques to generate a nondeterministic finite automaton $\mathscr{A}$ for α (viewed as an ordinary regular expression over the alphabet Σ);
- calculate the product ASMC $\mathscr{M} \times \mathscr{A}$, where it suffices to calculate the reachable part of $\mathscr{M} \times \mathscr{A}$ with an on-the-fly construction that starts with the states $\langle s, Z_0\rangle$, and to ignore the action-labels in the sense that the rates of "parallel" transitions are cumulated;
- apply a CSL-model checker to calculate the values $p_s = \mathsf{Prob}^{\mathscr{M}\times\mathscr{A}}(\langle s, Z_0\rangle, \Diamond^I \mathsf{accept})$ for all states s in $\mathscr{M}$, e.g. with the help of a transient analysis of the Markov chain which is obtained from $\mathscr{M} \times \mathscr{A}$ when all states labelled by accept and all states from which one cannot reach a state labelled by accept (especially those that have an empty automaton part) are made absorbing [16, 24];
- return the set $\{s \in S : p_s \bowtie p\}$.

In [24] it was shown that the time complexity of the uniformisation-based model checking algorithm for CSL-formulas of type $\mathscr{P}_{\bowtie p}(\phi_1 \mathscr{U}^{[t,t']} \phi_2)$ is $\mathscr{O}(M \cdot q \cdot t')$, where M is the number of transitions of the model and q is the uniformisation rate (which is given by the largest exit-rate of a state of the model). In our approach, an asCSL-formula of type $\mathscr{P}_{\bowtie p}(\alpha^I)$ is checked by first constructing an NPA $\mathscr{A}_\alpha$ which has $|Z| = \mathscr{O}(|\alpha|)$ states, and then constructing the product Markov chain, which has at most $M \cdot 2^{|Z|}$ transitions. The uniformisation rate and the time bound t' are not affected by the product automaton construction. Therefore, the overall time complexity of our algorithm to calculate the satisfaction set for an asCSL-formula of type $\mathscr{P}_{\bowtie p}(\alpha^{[t,t']})$ is bounded by $\mathscr{O}(M \cdot 2^{|\alpha|} \cdot q \cdot t')$.

5. Handover in a cellular radio network

In this section we present a case study in order to illustrate the techniques we have developed. We consider a scalable cellular phone setting, where base stations and switching center operate at different load levels. We track a single distinguished mobile radio station (MS) moving from one cell to another, thereby possibly triggering a so-called handover procedure. We are especially interested in the behaviour of the system concerning this distinguished user. The model

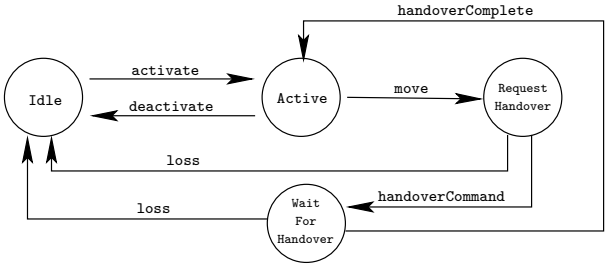


Figure 2. State machine for the MS behaviour

is inspired by the description of the GSM handover procedure in [25] and [26]. We describe the system as a set of synchronising processes, namely the cells, the switching center and the user. Each of these processes will be represented by a stochastic Petri net. The properties of interest are expressed with asCSL-formulas involving programs. We show the corresponding NPAs and relate the size of the resulting product Markov chains to the size of the original model.

5.1. The model

In the model, each cell corresponds to one base station subsystem (BSS). A BSS is modelled with only two states: either it has still free capacity or it is full and does not accept further connections. The number N of cells is a parameter of the model. The mobile services switching center (MSC) is modelled similarly: it has either low, medium or high load. The time needed for the handover command procedure depends on the current load. Under high load, the MSC does not process any request for handover.

Our distinguished MS has predetermined possibilities of moving between cells. For the time being, we assume that cells are positioned in a ring order, that is, if the MS is located in cell i it can only move to cells $(i+1) \bmod N$ and $(i-1) \bmod N$. We could easily change this to other topological cell orderings.

Finally, we have a model of the MS behaviour (beside its spatial position). When not being active with a connection, the MS is idle. At any time, the MS can become active, meaning that it has established a (radio) connection. After a while, the connection is terminated and the MS becomes idle again. If it moves from one cell to another while being active, the corresponding BSS commands a handover to the new cell from the MSC. If the new cell has free capacity, the handover is eventually completed and the MS returns to state active (note that the connection is continued during the entire handover procedure). If the handover procedure is not completed in time, the connection might also be lost. The connection is then terminated (assume, that the distance to the former cell has become too large) and the MS is back in idle state.

process	action	rate	description
BSS	block	0.002	cell will not accept further connections
	free	0.008	cell will accept further connections
MSC	lowtoMedium	0.5	from low load to medium load
	mediumToHigh	1.0	from medium load to high (blocking) load
	highToMedium	3.0	from high (blocking) load to medium load
	mediumToLow	1.0	from medium to low load
MS position	move	0.02	from cell i to $(i-1) \bmod N$ or $(i+1) \bmod N$
MS behaviour	activate	0.000625	average time between connections is 1600 seconds
	deactivate	0.008	connections last on average 125 seconds
	handoverCommand	1.0/0.5	for low/medium load of MSC, not available if MSC is blocking
	handoverComplete	1.0	only activated if new cell is not blocking
	loss	0.1	might happen during handover procedure

Table 1. Action labels and rates of the transitions of the cellular network model

Figure 2 shows a state machine for the user call behaviour. Transitions are labelled with action names. The `move` transition synchronises with the spatial movement of the user whenever he is active. Table 1 states the rates for transitions labelled with the given actions. Note that all numbers are educated guesses made on the basis of [27].

5.2. asCSL-properties

Out-dated handover. When the MS moves from one cell to the next, the BSS requests a handover to the new cell. However, the model does not prevent the MS from moving on to yet another cell. This behaviour is not explicitly visible in the model: here a handover is simply made to the cell the MS is in, no matter where it has been in between. In reality this type of movement could cause a problem. So, we would like to know whether the probability of such an outdated handover is lower than 2%. As asCSL-formula, this becomes: $\Phi_1 = \mathscr{P}_{\leq 0.02}(\alpha_1^{[0,\infty]})$, with

$$\begin{aligned} \alpha_1 = {} & (\texttt{Active}, \texttt{move}); && (1)\\ & (\texttt{RequestHandover} \vee \texttt{WaitForHandover}, && (2)\\ & \mathsf{Act}\backslash\{\texttt{handoverComplete}, \texttt{move}\})^*; && (3)\\ & (\texttt{RequestHandover} \vee \texttt{WaitForHandover}, \texttt{move}) && (4) \end{aligned}$$

A move while the MS is active triggers a handover. Lines (2/3) describe the system inside the handover procedure. A move (4) leads to an outdated handover. An NPA for the program α_1 is given in Fig. 3(a).

Return without interruption. Assume that the MS initiates a connection while in cell 1. It is free to move between cells. We would like it to leave cell 1 and to return within 10 minutes (600 seconds) without terminating or loosing the connection. Is the probability for this scenario at least 50%? Coded into an asCSL-formula this

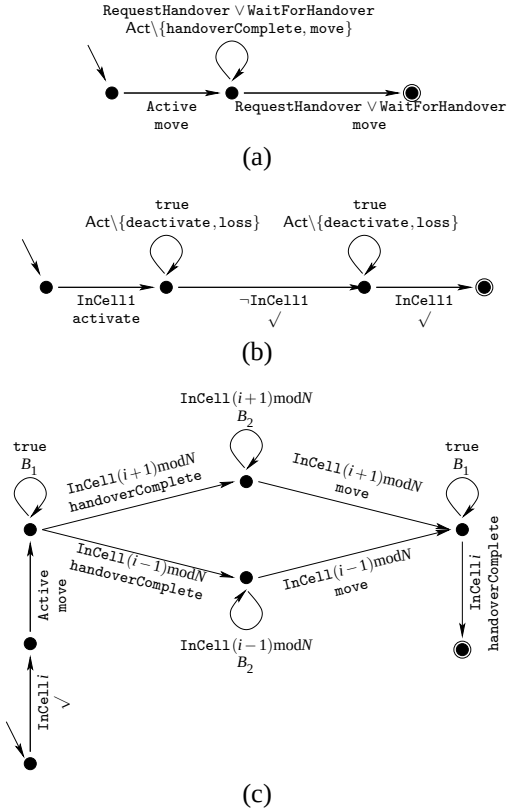


Figure 3. NPAs for the programs α_1, α_2 and β_i

reads $\Phi_2 = \mathscr{P}_{>0.5}(\alpha_2^{[0,600]})$, with

$$\begin{aligned} \alpha_2 = {} & (\texttt{InCell1}, \texttt{activate}); && (5)\\ & (\texttt{true}, \mathsf{Act}\backslash\{\texttt{deactivate}, \texttt{loss}\})^*; && (6)\\ & (\neg\texttt{InCell1}, \surd); && (7)\\ & (\texttt{true}, \mathsf{Act}\backslash\{\texttt{deactivate}, \texttt{loss}\})^*; && (8)\\ & (\texttt{InCell1}, \surd) && (9) \end{aligned}$$

The regular expression first ensures that the user activates a connection while being in cell 1 (5). Then the user can behave arbitrarily, as long as the connection is not ended via a `deactivate` or `loss` event (6). At some time, the user must have left cell 1 (7) and can again behave arbitrarily, as long the connection remains established (8). Finally, he should return to cell 1 (9). Figure 3(b) shows an NPA for the program α_2.

Ping-pong. Sometimes there are handovers from a cell i to a neighbouring cell and back to cell i within a short time interval. From a performance point of view this is not desirable since presumably the call could have remained in cell i. If having an active connection, is the probability of such a ping-pong handover to occur within 10 seconds at most 10 %? As asCSL-formula: $\Phi_3 = \mathscr{P}_{\leq 0.1}(\alpha_3^{[0,10]})$. A

ping-pong originating from cell i is described by the program

$$\begin{aligned}
\beta_i \quad &= (\texttt{InCell}i, \surd); (\texttt{Active}, \texttt{move}); (\texttt{true}, B_1)^*; && (10)\\
&((\texttt{InCell}(i+1) \bmod N, \texttt{handoverComplete}); && (11)\\
&(\texttt{InCell}(i+1) \bmod N, B_2)^*; && (12)\\
&(\texttt{InCell}(i+1) \bmod N, \texttt{move}) \quad \cup && (13)\\
&(\texttt{InCell}(i-1) \bmod N, \texttt{handoverComplete}); && (14)\\
&(\texttt{InCell}(i-1) \bmod N, B_2)^*; && (15)\\
&(\texttt{InCell}(i-1) \bmod N, \texttt{move})); && (16)\\
&(\texttt{true}, B_1)^*; (\texttt{InCell}i, \texttt{handoverComplete}) && (17)
\end{aligned}$$

with $B_1 = \mathsf{Act} \backslash \{\texttt{move}, \texttt{loss}, \texttt{handoverComplete}\}$ and $B_2 = \mathsf{Act} \backslash \{\texttt{deactivate}, \texttt{move}\}$. An NPA for this program is given in Fig. 3(c). If there are N cells, all possible ping-pong situations are described by $\alpha_3 = \beta_0 \cup \beta_1 \cup \ldots \cup \beta_{N-1}$. The NPA for α_3 consists of N replicas of the automaton in Fig. 3(c), instantiated with $i = 0, \ldots N-1$. It has N initial and N final states.

5.3. Tool support

A prototype implementation that performs the construction of the product Markov chain given an ASMC and an NPA was done in C++.

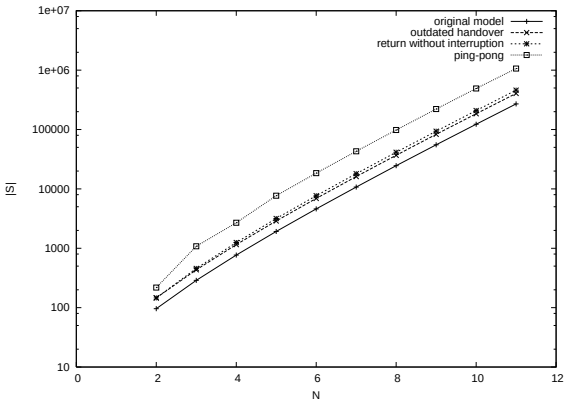
For the evaluation we used a stochastic Petri net (SPN) model of the cellular phone system. All components of the systems are described by simple state machines, we therefore omit their SPN representation. The SPN is described in a variant of CSPL, which also allows the specification of marking-dependent properties, which can be seen as atomic propositions in the underlying Markov chain. The state space generation code of [28] has been extended in order to record these properties and the transition names (as action labels) and generates an ASMC.

Programs have been described directly via their corresponding NPA. The prototype implementation takes the ASMC and the NPA as input and computes the reduced product Markov chain where only reachable states are generated and accept-states and states $\langle s, \emptyset \rangle$ with empty automaton part are merged into two special absorbing states.

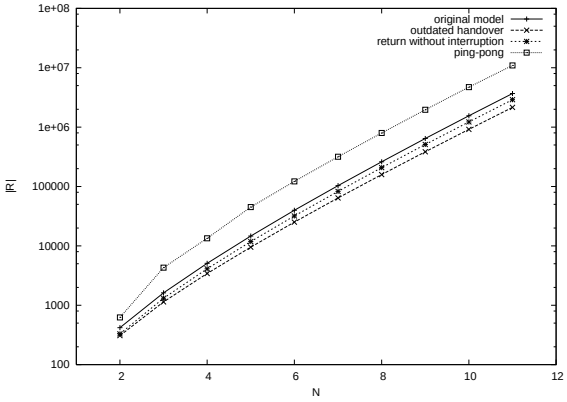
The final computation of the satisfaction relation of the corresponding CSL formula can then be done using an existing CSL model checker, for example ETMCC [7].

5.4. Results

Figure 4 shows the number of states and transitions of the original ASMC model of a cellular radio network and of the product Markov chains needed for the model checking procedure of the three given asCSL-formulas as a function of the number N of cells that ranges from 2 to 11.



(a) $|S^{\times}|$



(b) $|\mathbf{R}^{\times}|$

Figure 4. Size of state space (a) and number of transitions (b) in the original ASMC and the product Markov chains

The original model has about 270000 states and 3.6 million transitions for $N = 11$. For all three presented programs, the number of states in the product Markov chain is larger than in the original ASMC. This could be expected, since the state space is a subset of the Cartesian product of S and $2^{|Z|}$. The increase in size varies between a factor 1.5 and 4. The largest state space is the one of the ping-pong property, it has more than one million states for $N = 11$. If we keep only those states from which the accept-state is reachable and merge the others into one absorbing state, the number of states can also become smaller than the orginal state space.

For the properties "outdated handover" and "return without interruption", the number of transitions in the product Markov chain is smaller than in the original ASMC. The corresponding program automata are very restrictive, in the sense that in each state of the original ASMC only a subset of all outgoing transitions is allowed by the NPA. The NPA for "ping-pong" allows a broader range of different combinations of states and transitions. Consequently, it shows the largest growth in state space, and the number of transitions

is larger than in the original model.

6. Conclusions

We introduced the logic asCSL as a new specification formalism to reason about performability measures for Markov chains with both action- and state-labels. It subsumes CSL (with time intervals $[0,t]$) and several variants that have been suggested in the literature, such as aCSL, aCSL+, pathCSL, SPDL [6], [8], [9],[10]. Our model checking algorithm for formulas of type $\mathscr{P}_{\bowtie p}(\alpha^I)$ relies on a reduction to the CSL model checking problem via a product construction of the Markov chain $\mathscr{M}$ and an automaton for the path formula α^I. The case study in section 5 demonstrates how asCSL formulas can specify complex properties that refer to both action and state labels in a rather simple way.

References

[1] E. Clarke, O. Grumberg, and D. Peled, *Model Checking*. MIT Press, 1999.

[2] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying continuous time Markov chains," in *CAV'96, LNCS 1102*, 1996, pp. 269–276.

[3] C. Baier, J.-P. Katoen, and H. Hermanns, "Approximate symbolic model checking of continuous-time Markov chains," in *CONCUR'99, LNCS 1664*, 1999, pp. 146–161.

[4] H. Hansson and B. Jonsson, "A logic for reasoning about time and reliability," *Formal Aspects of Computing*, vol. 6, no. 5, pp. 512–535, 1994.

[5] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "On the logical characterisation of performability properties," in *ICALP'00, LNCS 1853*, 2000, pp. 780–792.

[6] H. Hermanns, J. Katoen, J. Meyer-Kayser, and M. Siegle, "Towards model checking stochastic process algebra," in *IFM'00, LNCS 1945*, 2000, pp. 420–439.

[7] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "Implementing a model checker for performability behaviour," in *Proc. 5th Int'l. Workshop on Performability Modelling of Computer and Communication Systems*, Erlangen, Germany, 2001, pp. 110–115.

[8] J. Meyer-Kayser, "Automatische Verifikation Stochastischer Systeme," Ph.D. dissertation, Universität Erlangen-Nürnberg, Institut für Informatik, 2004.

[9] C. Baier, L. Cloth, B. Haverkort, H. Hermanns, and J. Katoen, "Model checking pathCSL," in *Proc. 6th Int'l. Workshop on Performability Modeling of Computer and Communication Systems*, Monticello, Illinois, 2003, pp. 19–22.

[10] M. Kuntz and M. Siegle, "A stochastic extension of the logic PDL," in *Proc. 6th Int'l. Workshop on Performability Modeling of Computer and Communication Systems*, Monticello, Illinois, 2003, pp. 58–61.

[11] W. Obal and W. Sanders, "State-space support for path-based reward variables," *Perform. Eval.*, vol. 35, no. 3-4, pp. 233–251, 1999.

[12] M. Fischer and R. Ladner, "Propositional dynamic logic of regular programs," *J. Comput. Syst. Sci.*, vol. 8, pp. 194–211, 1979.

[13] P. Wolper, "Specification and synthesis of communicating processes using an extended temporal logic," in *Proc. 9th Symposium on Principles of Programming Languages*, 1982, pp. 20–33.

[14] A. Aziz, V. Singhal, F. Balarin, and R. K. Brayton, "It usually works: The temporal logic of stochastic systems," in *CAV'95, LNCS 939*, 1995, pp. 155–165.

[15] W. Stewart, *Introduction to the Numerical Solution of Markov Chains*. Princeton University Press, 1994.

[16] C. Baier, B. Haverkort, J.-P. Katoen, and H. Hermanns, "Model checking continuous-time Markov chains by transient analysis," in *CAV'00, LNCS 1855*, 2000, pp. 358–372.

[17] J. Kemeny and J. Snell, *Finite Markov Chains*. Springer, 1976.

[18] P. Buchholz, "Exact and ordinary lumpability in finite Markov chains," *J. Appl. Prob.*, no. 31, pp. 59–75, 1994.

[19] J. Hillston, "A Compositional Approach to Performance Modelling," Ph.D. dissertation, University of Edinburgh, 1994.

[20] H. Hermanns and M. Rettelbach, "Syntax, semantics, equivalences, and axioms for MTIPP," in *PAPM'94*. Universität Erlangen-Nürnberg, 1994, pp. 71–88.

[21] J. Desharnais and P. Panangaden, "Continuous stochastic logic characterizes bisimulation of continuous-time Markov processes," *J. Logic Algebr. Progr.*, vol. 56, no. 1-2, pp. 99–115, 2003.

[22] M. Bernardo and R. Cleaveland, "A theory of testing for Markovian processes," in *CONCUR 2000, LNCS 1877*, 2000, pp. 305–319.

[23] E. Clarke, E. Emerson, and A. Sistla, "Automatic verification of finite-state concurrent systems using temporal logic specifications," *ACM Trans. Program. Lang. Syst.*, vol. 8, no. 2, pp. 244–263, 1986.

[24] C. Baier, B. Haverkort, H. Hermanns, and J. Katoen, "Model-Checking Algorithms for Continuous-Time Markov Chains," *IEEE Trans. Softw. Eng.*, vol. 29, no. 7, pp. 1–18, 2003.

[25] B. H. Walke, *Mobile Radio Networks*. Wiley, 1999.

[26] J. M. Thomsen and R. Manggaard, "Analysis of GSM handover using coloured Petri nets," Master's thesis, University of Aarhus, 2003.

[27] J. Ventura Agustina, P. Zhang, and R. Kantola, "Performance evaluation of GSM handover traffic in a GPRS/GSM network," in *Proc. 8th IEEE Int'l. Symp. on Computers and Communications*. IEEE Press, 2003, pp. 137–142.

[28] B. Haverkort, H. Bohnenkamp, and A. Bell, "On the efficient sequential and distributed evaluation of very large stochastic Petri nets," in *PNPM'99*. IEEE Press, 1999, pp. 12–21.