Model Checking Markov Chains with Actions and State Labels

Christel Baier, Lucia Cloth, Boudewijn R. Haverkort, *Fellow*, *IEEE*, Matthias Kuntz, and Markus Siegle

Abstract—In the past, logics of several kinds have been proposed for reasoning about discrete-time or continuous-time Markov chains. Most of these logics rely on either state labels (atomic propositions) or on transition labels (actions). However, in several applications it is useful to reason about both state properties and action sequences. For this purpose, we introduce the logic asCSL which provides a powerful means to characterize execution paths of Markov chains with actions and state labels. asCSL can be regarded as an extension of the purely state-based logic CSL (continuous stochastic logic). In asCSL, path properties are characterized by regular expressions over actions and state formulas. Thus, the truth value of path formulas depends not only on the available actions in a given time interval, but also on the validity of certain state formulas in intermediate states. We compare the expressive power of CSL and asCSL and show that even the state-based fragment of asCSL is strictly more expressive than CSL if time intervals starting at zero are employed. Using an automaton-based technique, an asCSL formula and a Markov chain with actions and state labels are combined into a product Markov chain. For time intervals starting at zero, we establish a reduction of the model checking problem for asCSL to CSL model checking on this product Markov chain. The usefulness of our approach is illustrated with an elaborate model of a scalable cellular communication system, for which several properties are formalized by means of asCSL formulas and checked using the new procedure.

Index Terms—Protocol verification, performance of systems, model checking, automata, Markov processes.

1 Introduction

Besides being functionally correct, an ever larger share of computer and communication systems, especially in the area of embedded systems, has to meet severe performance and dependability constraints. Such constraints are typically formalized in a stochastic framework. To reason about such stochastic phenomena, a variety of high-level models such as stochastic Petri nets, stochastic process algebras, queueing networks, etc., have been established; see [1]. Typically, the verification of quantitative properties relies on a transformation of these high-level models into a (finite-state) 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 [2]. Over the past 10 years, several researchers have adapted the temporal-logic approach to reason about probabilistic properties. One result of these efforts is the logic CSL (continuous stochastic logic), introduced in [3] and extended in [4], which is a continuous-time variant of PCTL (probabilistic computational tree logic) [5], that can be used as specification formalism for performance and dependability properties. For instance, the CSL formula $\mathcal{P}_{\geq 0.99}(\text{legal}\ \mathcal{U}^{\leq 5}\text{goal})$ specifies the state-property asserting that "there is at least a 99 percent probability to reach a goal state within the next five 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 reasoning about stationary probabilities. Formula $\mathcal{S}_{\geq 0.75}(\text{green})$ states that, in equilibrium, the accumulated probability mass for green states is at least 75 percent. An extension of CSL to reason about rewards has been introduced in [6]. Notice that the specification of these measures is completely state-oriented.

For action-oriented modeling formalisms, such as stochastic process algebra, CSL is not an adequate specification formalism, since it is not possible to characterize sequences of actions. In [7] an action-based variant of CSL, called aCSL, was proposed, and, in [8], it was shown how to employ this logic for performability modeling.

Although the state-labeled and action-labeled approaches are similar in their expressivity and transformations between them can be provided as in the nonstochastic case [9], reasoning with doubly labeled models is often more intuitive, or even efficient. A similar observation was made in [10] for nonprobabilistic systems. A first step toward the combination of state-oriented and action-oriented features in logics for Markov chains is the logic aCSL+ [11], which employs regular expressions for characterizing more general path-based properties.

In this paper, we introduce a new logic, called asCSL, (for CSL with actions and state labels), which combines aspects of all the above-mentioned logics. Preliminary work on logics similar to asCSL has been published in [12] and [13]. asCSL can be seen as being motivated either by the method of path-based reward variables as described in [14], or by the propositional dynamic logic [15] and extended linear

- *C. Baier is with the Institute for Theoretical Computer Science, Technische Universität Dresden, D-01062 Dresden, Germany. E-mail: baier@tcs.ino.tu-dresden.de.*
- *L. Cloth, B.R. Haverkort, and M. Kuntz are with the EWI/DACS, University of Twente, PO Box 217, 7500 AE Enschede, The Netherlands. E-mail: {lucia, brh, kuntzwgm}@cs.utwente.nl.*
- *M. Siegle is with the Universität der Bundeswehr München, Institute für Technische Informatik, Fakultät für Informatik, 85577 Neubiberg, Germany. E-mail: markus.siegle@unibw.de.*

Manuscript received 3 Mar. 2006; revised 18 July 2006; accepted 4 Aug. 2006; published online 1 Mar. 2007.
Recommended for acceptance by S. Donatelli.
For information on obtaining reprints of this article, please send e-mail to: tse@computer.org, and reference IEEECS Log Number TSE-0052-0306.

temporal logic [16]. With asCSL, paths are characterized 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 provides an elegant way to reason about state-oriented and action-oriented behaviors. Unlike extended linear temporal logic [16], we do not allow for ω-regular expressions (representing infinite behavior). 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 of PCTL, PCTL* [17], which is concerned with specifying complex properties of infinite computations, whereas asCSL focuses on complex properties of finite computations with real-time constraints (e.g., hard or soft deadlines).

We finally note that we published a short conference paper on asCSL [18] that resulted from a joint effort to combine the two earlier extended abstracts on pathCSL [12] and SPDL [13]. The current paper extends that 10-page paper in that definitions of syntax and semantics are given in more detail, with more examples. Also, the sections that compare the expressive power with other logics as well as bisimulation results have been extended. Furthermore, the model checking procedure is described in more detail and an elaborated example is presented. Finally, to ease the reading of the paper, a running example illustrating the complete model checking procedure has been added.

This paper is further organized as follows: In Section 2, we define Markov chains with actions and state labels. Section 3 presents syntax and semantics of the new logic asCSL. In Section 4, we relate asCSL to other logics, and, in Section 5, we present an important bisimulation equivalence property for asCSL. Section 6 is dedicated to the model checking procedure for asCSL. Throughout these sections, we use a "running example" to illustrate the new concepts. In Section 7, we then apply the new technique to a nontrivial example, in order to derive properties of the handover procedure in a cellular radio network. The paper ends with a short summary and conclusions.

2 Markov Chains with Actions and State Labels

In this section, we explain the notation for Markov chains with actions and state labels. The reader is supposed to be familiar with Markov chains; see [19]. Action names are used to label the transitions. The state labels are drawn from 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. A function L assigns to any state the set of atomic propositions that are assumed to hold in the given state.

Definition 1 (Markov chain with actions and state labels). *A continuous-time Markov chain with actions and state labels (ASMC) is a tuple $\mathcal{M} = (S, \mathrm{Act}, \mathrm{AP}, L, \mathbf{R})$, where S is a finite set of states, Act is a finite set of action labels, AP is a finite set of atomic propositions, $L : S \to 2^{\mathrm{AP}}$ a state labeling function, and $\mathbf{R} : S \times \mathrm{Act} \times S \to \boldsymbol{R}_{\geq 0}$ is a rate matrix.*

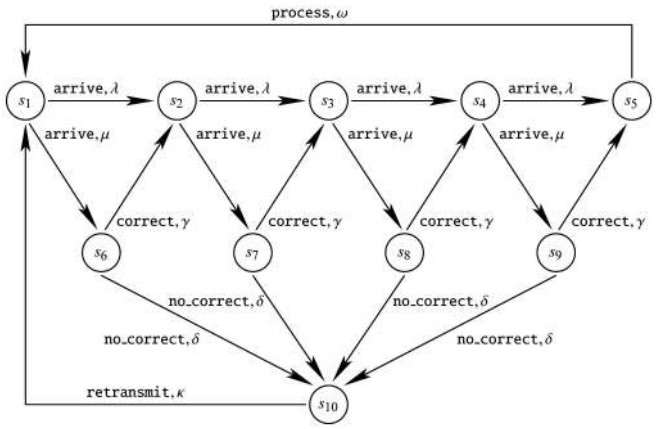

Fig. 1. ASMC for the simple data transmission system of Example 2.

Notice that we do not explicitly define an initial state or an initial state distribution as these are not relevant for our purposes.

Example 2. Fig. 1 represents an ASMC that serves as a running example throughout this paper. It models a data transmission system that receives four data packets in a row and then processes them jointly. The set of states is given by $S = \{s_1, \ldots, s_{10}\}$, the set of actions is

$$\mathrm{Act} = \{\texttt{arrive}, \texttt{correct}, \texttt{no\_correct}, \texttt{retransmit}, \texttt{process}\}.$$

The rate matrix $\mathbf{R}$ becomes clear from Fig. 1.

In more detail, an arrival is modeled by action arrive. The transmission of data packets can be error-free (rate λ) or erroneous (rate μ). An erroneous packet might be corrected $(\texttt{correct}, \gamma)$ or the system might be unable to correct it $(\texttt{no\_correct}, \delta)$. In case it cannot be corrected, the buffer is emptied and all data packets have to be retransmitted $(\texttt{retransmit}, \kappa)$. If all four data packets are error-free or corrected, the received data can be processed $(\texttt{process}, \omega)$ and the system awaits new data. The set of atomic propositions is $\mathrm{AP} = \{\mathrm{empty}, \mathrm{full}, \mathrm{error}\}$, with

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

Intuitively, if $\mathbf{R}(s, a, s') = \lambda > 0$, then there is an a-labeled 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'.

The underlying CTMC[1] of an ASMC is given as a tuple $(S, \mathrm{AP}, L, \mathbf{R}')$ that arises from $\mathcal{M}$ by removing the action-set and accumulating the rates of "parallel" transitions, that is, $\mathbf{R}'(s, s') = \sum_{a \in \mathrm{Act}} \mathbf{R}(s, a, s')$.

Definition 3 (Paths in $\mathcal{M}$). *A finite path in $\mathcal{M}$ is a finite word*

$$\sigma = (s_0, a_0, t_0), (s_1, a_1, t_1), \ldots, (s_{n-1}, a_{n-1}, t_{n-1}), s_n \in (S \times \mathrm{Act} \times \boldsymbol{R}_{>0})^* \times S,$$

1. Here and following, we use the abbreviation CTMC for a continuous-time Markov chain with just state labels.

where $\mathbf{R}(s_i, a_i, s_{i+1}) > 0$ *for* $i = 0, \ldots, n-1$. *An infinite path in* $\mathcal{M}$ *is an infinite word*

$$\varsigma = (s_0, a_0, t_0), (s_1, a_1, t_1), \ldots \in (S \times \mathrm{Act} \times \boldsymbol{R}_{>0})^{\omega}$$

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.,* $t_0 + t_1 + t_2 + \ldots = \infty$*), with* t_i *the time spent in* s_i. *A special case arises if* $\mathcal{M}$ *contains absorbing states, i.e., states without outgoing transitions. We then define a second type of infinite paths. Such an absorbing path is a word*

$$\varsigma = (s_0, a_0, t_0), (s_1, a_1, t_1), \ldots (s_{n-1}, a_{n-1}, t_{n-1}), (s_n, t_n) \in (S \times \mathrm{Act} \times \boldsymbol{R}_{>0})^* \times (A \times \{\infty\}),$$

where A *is the set of absorbing states of* $\mathcal{M}$ *and* $\mathbf{R}(s_i, a_i, s_{i+1}) > 0$ *for all* $0 \leq i \leq n-1$. *The sojourn time of an absorbing state is always* ∞.

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 or absorbing path as before, then $|\sigma| = n$ denotes the length of σ, i.e., the number of transitions in σ, and let $\sigma[i] = s_i$ the $(i+1)$st state 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 $(i \leq j)$. In particular, $\sigma(i, i) = s_i$ is a path of length 0.

If σ_1 and 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 if σ_2 is an infinite path. $\mathrm{Path}_{\mathrm{fin}}^{\mathcal{M}}$ denotes the set of all finite paths in $\mathcal{M}$, whereas $\mathrm{Path}_{\omega}^{\mathcal{M}}$ stands for the set of infinite paths in $\mathcal{M}$. By $\mathrm{Path}_{\mathrm{fin}}^{\mathcal{M}}(s)$, respectively, $\mathrm{Path}_{\omega}^{\mathcal{M}}(s)$, we denote the set of all finite, respectively, infinite, paths in $\mathcal{M}$ with initial state s.

Example 4. The following is a path in the ASMC of Example 2. It describes the successful transmission, correction, and processing of three related data packets:

$$\sigma = s_2 \xrightarrow{\text{arrive},1.1} s_3 \xrightarrow{\text{arrive},1.2} s_4 \xrightarrow{\text{arrive},0.8} s_9 \xrightarrow{\text{correct},0.5} s_5 \xrightarrow{\text{process},2.0} s_1 \xrightarrow{\text{arrive},2.1} s_2.$$

The second state of the path is $\sigma[1] = s_3$, and the state occupied at time 3 is $\sigma@3 = s_4$.

If we concatenate an infinite number of copies of σ, we obtain an infinite path where the data packets never need to be retransmitted.

In the following, we deal with the standard probability measure $\Pr_s^{\mathcal{M}}$ on $\mathrm{Path}_{\omega}^{\mathcal{M}}(s)$ (where the underlying σ-field can be defined with the help of basic cylinders as in [4]). For measurable $X \subseteq \mathrm{Path}_{\omega}^{\mathcal{M}}(s)$, we often omit the parameter $\mathcal{M}$ and/or s and simply write $\Pr(X)$ or $\Pr^{\mathcal{M}}(X)$. The transient state probabilities $\pi^{\mathcal{M}}(s, s', t)$ are given by

$$\pi^{\mathcal{M}}(s, s', t) = \Pr^{\mathcal{M}}\{\varsigma \in \mathrm{Path}_{\omega}^{\mathcal{M}}(s) \mid \varsigma@t = s'\},$$

and the steady state probability of being in state s' in the long run, provided that the system started in state s, is

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

For $S' \subseteq S$, we define $\pi^{\mathcal{M}}(s, S') = \sum_{s' \in S'} \pi^{\mathcal{M}}(s, s')$.

3 Syntax and Semantics of asCSL

The logic CSL (continuous stochastic logic) [3], [4] specifies state-based properties for CTMCs, built out of propositional logic (with atoms $\mathrm{q} \in \mathrm{AP}$), a steady-state operator that refers to the stationary probabilities, and a probabilistic operator for reasoning about transient state probabilities, but can also express other probabilistic properties with or without real-time constraints. We present here an extension of CSL, called asCSL, that allows one to specify probability bounds for action-sequences and state-sequences. Later, we will discuss the expressivity of asCSL in relation to that of other logics (Section 4), as well as the relation between the equivalence induced by asCSL and action-labeled and state-labeled bisimulation equivalence (Section 5).

3.1 Syntax of asCSL

The syntax of asCSL is defined according to Definition 5 (state formulas) and Definition 6 (programs or path formulas). Here, we assume that the sets Act of actions and AP of atomic propositions are fixed.

Definition 5 (State formulas of asCSL). *State formulas of* asCSL *are given by the following grammar:*

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

where $\mathrm{q} \in \mathrm{AP}$ *is an atomic proposition,* $p \in [0, 1]$ *denotes a probability value,* $\bowtie \in \{<, \leq, >, \geq\}$ *a comparison operator,* $I = [t, t'] \subseteq \boldsymbol{R}_{\geq 0}$ *a time interval, and* α *a program as defined in Definition 6. 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 $\mathcal{S}_{\bowtie p}(\phi)$ asserts that the probability of being in a ϕ-state in the long run obeys the bound $\bowtie p$. The operator $\mathcal{P}_{\bowtie p}(\alpha^I)$ 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, obeys the bound $\bowtie p$. We omit the time interval I if $I = [0, \infty]$, which does not impose any proper real-time constraints. If $I = [0, t]$, we write $\leq t$ instead.

The program α specifies a property for finite paths via a regular set of finite words whose atomic symbols are pairs (ϕ, b) (when no confusion can arise, we sometimes 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

$b \in \mathrm{Act}$ or $b = \surd$ (where $\surd \notin \mathrm{Act}$). The symbol $\surd$ can be viewed as a pseudoaction which is always immediately executable and does not change the current state in the ASMC. Formally, the programs are regular expressions over the alphabet

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

Definition 6 (Programs). asCSL *programs are defined by the following grammar:*

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

where $(\phi, b) \in \Sigma$. *The language* $\mathcal{L}(\alpha) \subseteq \Sigma^*$ *is defined in the standard way.*

The reader should notice the difference between the pseudoaction symbol $\surd$ and the empty word ε (viewed as an element of Σ^*) as the use of $\surd$ is only allowed in combination with a state formula.

In the context of asCSL, the meaning of a program (which will be formally defined in the next section) is a set of finite paths in the underlying ASMC. The intuitive meaning of ϕb is that the current state s fulfills ϕ (that is, the state formula ϕ can be viewed as a test for the current state s) and, if $b \in \mathrm{Act}$, state s has an outgoing b-transition. If $b = \surd$, no statement about outgoing transitions is made. 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).

Example 7. Let p, q, r, and s be atomic propositions and let a, b, and c be actions.

The language of the program $(\mathrm{p}, a)\,((\mathrm{q}, \surd) \cup (\mathrm{r}, b))$ consists of words of length 2 that

- start with (p, a) and
- end with either $(\mathrm{q}, \surd)$ or (r, b).

Thus,

$$\mathcal{L}((\mathrm{p}, a)\,;((\mathrm{q}, \surd) \cup (\mathrm{r}, b))) = \{(\mathrm{p}, a)(\mathrm{q}, \surd), (\mathrm{p}, a)(\mathrm{r}, b)\}$$

has exactly two members.

The language of the program

$$((\mathrm{p}, a); (\mathrm{q}, b); (\mathrm{r}, \surd))^*\,; (\mathrm{s}, c)$$

involving the Kleene star is infinite. It contains

- (s, c) (a word in Σ^* of length 1),
- $(\mathrm{p}, a)(\mathrm{q}, b)(\mathrm{r}, \surd)(\mathrm{s}, c)$ (a word in Σ^* of length 4),
- $(\mathrm{p}, a)(\mathrm{q}, b)(\mathrm{r}, \surd)(\mathrm{p}, a)(\mathrm{q}, b)(\mathrm{r}, \surd)(\mathrm{s}, c)$ (a word in Σ^* of length 7),

and so on.

Note that the test can be empty, i.e., $\phi = \mathrm{true}$. This gives the possibility to speak about action sequences of paths without any (further) constraints for the intermediate states.

Example 8. The asCSL formula

$$\mathcal{P}_{\geq 0.99}((\mathrm{true}, \texttt{arrive})^*; (\mathrm{true}, \texttt{process}))$$

denotes that the probability for an action sequence in `arrive`*; `process` is at least 0.99, whereas

$$\mathcal{P}_{\geq 0.99}\Big((\mathrm{true}, \texttt{arrive})^*; (\mathrm{true}, \texttt{process})^{\leq 5}\Big)$$

asserts the same probability bound for action-sequences `arrive`*; `process` to be performed within five time units. As another intuitive example, the asCSL formula

$$\mathcal{P}_{\geq 0.99}\Big((\mathrm{true}, \texttt{arrive})^*; (\mathrm{full}, \surd)^{\leq 5}\Big)$$

asserts at least a 99 percent chance to reach a state labeled with full via an `arrive`*-labeled path within five time units.

Example 9. In the context of the data transmission system of Example 2, the formula

$$\Psi = \mathcal{P}_{>0}\Big((\mathrm{true}, \texttt{arrive}); (\mathrm{full}, \surd)\Big)$$

characterizes exactly state s_4, because it is the only state from which a full-state can be reached with a single `arrive` transition.

In the sequel, we illustrate several concepts by means of the following formula:

$$\Phi = \mathcal{P}_{\leq 0.1}\big(\alpha^{\leq 7.3}\big),$$

where

$$\begin{aligned}\alpha = \big(&(\mathrm{true}, \texttt{arrive})\\ &\cup\ (\mathrm{true}, \texttt{arrive}); (\mathrm{error}, \texttt{correct})\big)^*;\\ &(\Psi, \texttt{arrive}); (\mathrm{error}, \texttt{correct}); (\mathrm{full}, \surd).\end{aligned}$$

Formula Φ states that the probability that the buffer is full after at most 7.3 time units, the last packet contains a correctable error, and that all other packets are either error-free or with correctable error, is at most 10 percent.

3.2 Semantics of asCSL

The formal semantics of asCSL is provided by means of a satisfaction relation $\models$ for the state and path formulas. In the following, we assume a fixed ASMC $\mathcal{M} = (S, \mathrm{Act}, \mathrm{AP}, L, \mathbf{R})$. For state s in $\mathcal{M}$ and state formula ϕ, $\mathcal{M}, s \models \phi$ means that the state property specified by ϕ holds for s. Similarly, for an infinite path ς, $\mathcal{M}, \varsigma \models \alpha^I$ denotes that the behavior 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 10 (Semantics of asCSL). *The satisfaction relation* $\models$ *for the state formulas is defined as follows:*

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

where $\mathrm{Sat}^{\mathcal{M}}(\phi) = \{s \in S | \mathcal{M}, s \models \phi\}$ *denotes the satisfaction set of* ϕ *in* $\mathcal{M}$ *and*

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

The meaning of the path formulas is formalized as follows: If ς *is an infinite path in* $\mathcal{M}$, *then*

$$\mathcal{M}, \varsigma \models \alpha^I$$

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

Fig. 2. Semantics of the programs.

iff there exists a finite prefix σ of ς with $\sigma \in \mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\alpha)$ and $\tau(\sigma) \in I$.[2] *Here, the set $\mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\alpha)$ consists of all finite paths σ in $\mathcal{M}$ that can be viewed as instances of α. Formally, the sets $\mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\alpha)$ are defined as shown in Fig. 2, where $\alpha^{i+1} = \alpha;\alpha^i$ and $\alpha^0 = \varepsilon$ (the empty word in Σ^*).*[3]

In the sequel, we often write $s \models \phi$ and $\varsigma \models \alpha^I$ rather than $\mathcal{M}, s \models \phi$ and $\mathcal{M}, \varsigma \models \alpha^I$, respectively. Moreover, (ϕ, B) stands short for $\bigcup_{a \in B}(\phi, a)$.

The reader should note the difference between the programs

$$\alpha_1 \;=\; (\mathrm{true}, \mathrm{Act})^*(\phi, \surd)$$

and

$$\alpha_2 \;=\; (\mathrm{true}, \mathrm{Act})^*(\phi, \mathrm{Act}).$$

Program α_1 defines all finite paths σ that end in a ϕ-state, whereas α_2 defines all finite paths whose prefinal state satisfies ϕ.

Example 11. Consider an infinite path ς which has the finite path

$$\sigma = s_2 \overset{\text{arrive},1.1}{\longrightarrow} s_3 \overset{\text{arrive},1.2}{\longrightarrow} s_4 \overset{\text{arrive},0.8}{\longrightarrow} s_9 \overset{\text{correct},0.5}{\longrightarrow} s_5 \overset{\text{process},2.0}{\longrightarrow} s_1 \overset{\text{arrive},2.1}{\longrightarrow} s_2$$

of Example 4 as prefix. Then, $\varsigma \models \alpha^{\leq 7.3}$ where α is the program of Example 9. Note that the last two transitions of σ are actually irrelevant for the validity of α^I.

4 Comparison of Expressive Power

In the following sections, we discuss the expressivity of asCSL in relation to that of CSL, aCSL, and aCSL+.

4.1 asCSL versus CSL

We now study the relation between asCSL and CSL [3], [4]. The syntax of CSL-state formulas is as in asCSL, except for the probabilistic operator which takes as input a probability bound $\bowtie p$ (as in asCSL) and a CSL-path formula of the form $\phi_1 \mathrm{U}^I \phi_2$ or $\mathrm{X}^I \phi$ (rather than a time-bounded asCSL program α^I). U^I is called time-bounded until operator and X^I a time-bounded next step operator.

For the formal definition of the syntax and semantics of CSL, we refer to [4]. Intuitively, $\phi_1 \mathrm{U}^I \phi_2$ asserts that there is some time point $t \in I$ such that the given path is in a ϕ_2-state at time point t and in ϕ_1-states at all earlier time points. Similarly, the CSL-path formula $\mathrm{X}^I \phi$ holds for a path ς if the second state in ς fulfills ϕ and the first transition in ς is taken at some time point $t \in I$.

A CSL formula ψ is said to be equivalent to an asCSL formula ϕ iff for any ASMC $\mathcal{M}$ and all states s in $\mathcal{M}$, ϕ holds for s iff ψ holds for s in the underlying CTMC of $\mathcal{M}$. We now discuss the possibilities to express the CSL modalities U^I and X^I in asCSL by providing asCSL formulas that are similar to the CSL formulas $\mathcal{P}_{\bowtie p}(\phi_1 \mathrm{U}^I \phi_2)$ and $\mathcal{P}_{\bowtie p}(\mathrm{X}^I \phi)$.

We first observe that state-based formulas that abstract from the action sequences and use CSL-like time-bounded operators can be derived from the syntax of asCSL path formulas. A CSL-like time-bounded until operator $\mathcal{U}^I$ is obtained in asCSL as follows:[4]

$$\phi_1 \mathcal{U}^I \phi_2 \overset{\text{def}}{=} ((\phi_1, \mathrm{Act})^*; (\phi_2, \surd))^I.$$

If $\varsigma = s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \ldots$ is an infinite path in an ASMC, then $\varsigma \models \phi_1 \mathcal{U}^I \phi_2$ iff there exists some index $i \geq 0$ such that $s_i \models \phi_2$, $\sum_{k=0}^{i-1} t_k \in I$, and $s_j \models \phi_1$ for all $j < i$. From this, we may derive the time-bounded eventually-operator

$$\diamond^I \phi \overset{\text{def}}{=} \mathrm{true}\, \mathcal{U}^I \phi.$$

We then have $\varsigma \models \diamond^I \phi$ (as opposed to the CSL formula $\diamond^I \phi$) iff there exists some index i with $s_i \models \phi$ and $\sum_{k=0}^{i-1} t_k \in I$. For instance, $\mathcal{P}_{\leq 0.05}(\diamond^{\leq 5}\mathrm{error})$ states that the probability to reach an error state within five time units is at most 0.05. Its dual, the time-bounded always-operator, is obtained (as in CSL) by using the duality of the temporal modalities "eventually" and "always" and the duality of lower and upper probability bounds. For instance, we may define

$$\mathcal{P}_{\geq p}(\boxdot^I \phi) \overset{\text{def}}{=} \mathcal{P}_{\leq 1-p}(\diamond^I \neg\phi).$$

Intuitively, the above asCSL formula states that ϕ continuously holds in the time interval I with probability at least p. In an analogous way, time-bounded always with other probability bounds can be defined.

A CSL-like time-bounded next operator can be expressed as

$$\mathcal{X}^I \phi \overset{\text{def}}{=} ((\mathrm{true}, \mathrm{Act}); (\phi\surd))^I.$$

We have $\varsigma \models \mathcal{X}^I \phi$ iff ς has the form $s_0 \xrightarrow{a_0,t_0} \varsigma'$, where $t_0 \in I$ and ϕ holds for the first state of ς'.

The asCSL semantics of the derived time-bounded until or next step operators $\mathcal{U}^I$ and $\mathcal{X}^I$ agrees with the corresponding CSL semantics of U^I and X^I if $\inf I = 0$ for

2. Recall that $\tau(\sigma)$ denotes the execution time of σ.
3. Note that all paths σ with $|\sigma| = 0$ belong to $\mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\alpha^*)$.
4. We will use calligraphic letters $\mathcal{U}$ and $\mathcal{X}$ for the until and next step operators in asCSL and the letters U and X for until and next step operators in CSL.

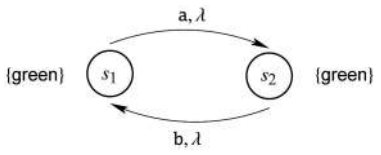

Fig. 3. A simple ASMC $\mathcal{M}$.

the time interval I. In fact, as long as we restrict our attention to time bounds of the form $\leq t$ or $< t$ where $t \in \boldsymbol{R} \cup \{\infty\}$, then CSL can be viewed as a sublogic of asCSL. More precisely, any CSL formula can be transformed into an equivalent asCSL formula by replacing $\mathrm{X}^{\trianglelefteq t}$ with $\mathcal{X}^{\trianglelefteq t}$ and $\mathrm{U}^{\trianglelefteq t}$ with $\mathcal{U}^{\trianglelefteq t}$, where $\trianglelefteq \in \{<, \leq\}$. (The proof can be provided by induction on the length of the given CSL formula.) Thus, we obtain:

Proposition 12. *CSL with lower time bound equal to zero is a sublogic of asCSL.*

For time bounds specified by intervals I with $\inf I > 0$, there is a slight difference between the semantics of the asCSL program $\varphi = \phi_1 \mathcal{U}^I \phi_2 = ((\phi_1, \mathrm{Act})^*; (\phi_2, \surd))^I$ and the CSL path formula $\psi = \phi_1 \mathrm{U}^I \phi_2$. Let us consider the case $I = [t, t']$ where $0 < t < t'$. The reason φ and ψ are not equivalent is that $\varsigma \models \psi$ but $\varsigma \not\models \varphi$ if ς is an infinite path that starts with a prefix

$$s_0 \xrightarrow{a_0,t_0} s_1 \xrightarrow{a_1,t_1} \cdots \xrightarrow{a_{n-2},t_{n-2}} s_{n-1} \xrightarrow{a_{n-1},t_{n-1}} s_n \xrightarrow{a_n,t_n} s_{n+1}.$$

such that

1. $s_i \models \phi_1$, $i = 0, 1, \ldots, n-1$,
2. $s_n \models \phi_1 \wedge \phi_2$,
3. $s_{n+1} \not\models \phi_1 \vee \phi_2$, and
4. $t_0 + \ldots + t_{n-1} < t < t_0 + \ldots + t_{n-1} + t_n$.

Note that the CSL semantics captures the possibility of time passage in the ϕ_2-state s_n, whereas the asCSL semantics requires the ϕ_2-state to be entered at some time instant $\delta \in [t, t']$. However, assuming ϕ_1 and ϕ_2 to be disjoint (i.e., $\phi_1 \wedge \phi_2 \equiv \mathrm{false}$) the CSL semantics and asCSL semantics of the until operator agree. In case ϕ_1 and ϕ_2 can hold in the same state, the CSL and the asCSL semantics of the until operator are different.

We now restrict our attention to the fragment of CSL and asCSL where $\inf I = 0$ for all time intervals I. It is not surprising that asCSL formulas that refer to the action labels in a nontrivial way cannot be expressed by CSL formulas. For example, there is no CSL formula ϕ which is equivalent to the asCSL formula

$$\phi = \mathcal{P}_{\geq 1}\big((\mathrm{true}, a); (\mathrm{true}, b)\big)$$

which holds exactly for those states where all outgoing paths start with action a followed by b.

For instance, in the ASMC shown in Fig. 3, states s_1 and s_2 differ only in the action-name of the outgoing transition, but have the same labeling $\{\mathrm{green}\}$ and exit rate. Hence, they fulfill the same CSL formulas,[5] but s_1 and s_2 can be distinguished by the asCSL formula ϕ as we have $s_1 \models \phi$ and $s_2 \not\models \phi$.

We finally restrict ourselves to the state-based fragment of asCSL, i.e., the fragment of asCSL in which the programs α are regular expressions using the atoms (ϕ, Act) and $(\phi, \surd)$. First, we observe that the asCSL formula

$$\phi = \mathcal{P}_{\bowtie p}((\mathrm{true}, \mathrm{Act}); (\mathrm{q}, \mathrm{Act}); (\mathrm{p}, \mathrm{Act})),$$

where $p \in]0, 1[$ and $\mathrm{q}, \mathrm{p} \in \mathrm{AP}$ cannot be described in CSL. We skip here a formal argument, but observe that ϕ states a nontrivial probability bound for reaching a q-state with the first transition and a p-state with the second transition. In CSL, however, we can formalize the possibility to reach a q-state followed by a p-state via two next step operators, but each of them has to be augmented with a probability bound.

A final example that illustrates that even the state-based fragment of asCSL is strictly more expressive than CSL is given by the asCSL formula

$$\phi' = \mathcal{P}_{>0}(((\mathrm{true}, \mathrm{Act}); (\mathrm{q}, \mathrm{Act}))^*; (\mathrm{p}, \surd)),$$

which states that a p-state is reachable via a finite path where any state at an odd position is labeled by q. Using a formal argument similar to [16], it can be shown that there is no CSL formula equivalent to ϕ'.

4.2 asCSL versus aCSL and aCSL+

The logic aCSL [7] is the action-based counterpart of CSL, just as aCTL [20] is the action-based counterpart of CTL. aCSL, which does not provide atomic propositions, is interpreted over a CTMC with action labels only, i.e., an ASMC without state labeling. Similar to CSL, aCSL offers a probability operator ($\mathcal{P}_{\bowtie p}$) and a steady-state operator ($\mathcal{S}_{\bowtie p}$). In aCSL, only time intervals with lower time bound equal to zero can be defined. Similar to aCTL, aCSL offers means to characterize satisfying paths via action-decorated versions of the next and until operators.

As we have shown for CSL, it can also be demonstrated that every aCSL formula can be translated into an equivalent asCSL formula. Using an argument similar to the one in Section 4.1, it can be proven that asCSL is strictly more expressive than aCSL. In [11], the logic aCSL+ was introduced; it extends aCSL in two ways:

- aCSL+ supports atomic propositions and its semantic model is an ASMC (as in the case of asCSL), and
- satisfying paths are characterized by regular expressions of actions, but not tests.

By means of elementary examples, it can be shown that, using the test feature of asCSL, it becomes possible to differentiate between paths which were indistinguishable in aCSL+.

5 asCSL-Equivalence and Bisimulation

It is well-known that bisimulation equivalence on CTMCs is the coarsest equivalence that identifies all states of a CTMC fulfilling the same CSL formulas [21], [4], [22]. We now establish a similar result for asCSL, which we need in the

5. Formally, this follows from the observation that s_1 and s_2 are bisimulation equivalent in the underlying CTMC (see Section 5).

correctness proof of the model checking algorithm presented in Section 6.

Definition 13 (Bisimulation equivalence). *Bisimulation equivalence $\sim$ for an ASMC:*
$\mathcal{M} = (S, \mathrm{Act}, \mathrm{AP}, L, \mathbf{R})$ *is the coarsest equivalence on S such that, for all states $s_1 \sim s_2$:*

1. $L(s_1) = L(s_2)$ *and*
2. $\mathbf{R}(s_1, a, C) = \mathbf{R}(s_2, a, C)$ *for all actions $a \in \mathrm{Act}$ and all equivalence classes $C \in S/\sim$.*

For CTMCs with just state labels, bisimulation equivalence is defined in the same way by ignoring the action labels in item 2.

Bisimulation equivalence can be viewed as a refinement of ordinary lumpability of Markov chains [23], [24], since bisimulation equivalence takes both the state labeling and the action labeling into account. It essentially agrees with Markovian bisimulation for action-labeled Markov chains as introduced in [25], [26] (which takes the action labeling, but not the state labeling, into account).

In the following, asCSL-equivalence denotes the equivalence relation which identifies exactly those states that cannot be distinguished by asCSL formulas. We now show that bisimulation equivalence on ASMCs agrees with asCSL-equivalence. Using structural induction on the syntax of state formulas and programs 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

$$s_1 \models \phi \text{ iff } s_2 \models \phi \text{ for all state formulas } \phi \text{ of asCSL.} \quad (1)$$

$$\mathrm{Prob}^{\mathcal{M}}(s_1, \alpha^I) = \mathrm{Prob}^{\mathcal{M}}(s_2, \alpha^I) \text{ for all path formulas } \alpha^I \text{ of asCSL.} \quad (2)$$

The argumentation for (1) is straightforward and omitted here. In order to prove (2), a more general result can be established by structural induction on the syntax of programs which states that, for each program α, we have

$$\mathrm{Prob}^{\mathcal{M}}(s_1, \alpha^I, C) = \mathrm{Prob}^{\mathcal{M}}(s_2, \alpha^I, C).$$

for all bisimulation equivalent states s_1, s_1, all time intervals I, and all bisimulation equivalence classes $C \in S/\sim$. Here, $\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I, C)$ denotes the probability to reach a C-state from s via a finite path $\sigma \in \mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\alpha)$. The fact that

$$\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I) \;=\; \mathrm{Prob}^{\mathcal{M}}(s, \alpha^I, S),$$

finally yields the claim. Thus, bisimulation-equivalent states are asCSL-equivalent. To show the converse, similar arguments as in [22] can be used, to obtain:

Proposition 14. *Bisimulation equivalence for ASMCs agrees with asCSL-equivalence.*

More precisely, using the arguments of [22], it can be shown that even the sublogic of asCSL consisting of formulas built by conjunctions of atomic propositions and probabilistic formulas with upper time bounds and programs consisting of the base symbols (ϕ, b) is sufficient to provide a characterization of bisimulation equivalence for ASMCs.

As Markovian testing equivalence [27] is weaker than bisimulation equivalence, states that fulfill the same asCSL formulas are also Markovian testing equivalent.

6 Model Checking asCSL

The model checking procedure for asCSL is similar to that for CTL [28]. Given the asCSL state formula ϕ and an ASMC $\mathcal{M}$, we successively consider the subformulas ψ of ϕ and calculate the satisfaction sets $\mathrm{Sat}^{\mathcal{M}}(\psi) = \{s \in S | \mathcal{M}, s \models \psi\}$. 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 $\mathcal{S}_{\bowtie p}(\phi)$ can be handled with the same procedure as for CSL; see [4]. The new and challenging case is the treatment of formulas of type $\phi = \mathcal{P}_{\bowtie p}(\alpha^I)$. For each state s we have to compute the probability

$$\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I) = \Pr^{\mathcal{M}}\{\varsigma \in \mathrm{Path}^{\mathcal{M}}_{\omega}(s) \mid \mathcal{M}, \varsigma \models \alpha^I\},$$

and check whether it lies within the specified bound $\bowtie p$. The approach to calculate the values $\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I)$ is to build the product of $\mathcal{M}$ and a finite automaton $\mathcal{A}_\alpha$ (representing the program α), which yields a new Markov chain, denoted $\mathcal{M} \times \mathcal{A}_\alpha$, and then to apply the CSL model checking procedure to calculate the probabilities in $\mathcal{M} \times \mathcal{A}_\alpha$ to reach a state $\langle s', q \rangle$, with s' a state in $\mathcal{M}$ and q an accepting state in $\mathcal{A}_\alpha$, within the time interval I.

Section 6.1 is devoted to the construction of the automaton $\mathcal{A}_\alpha$ from the program α, whereas Section 6.2 presents the construction of the product Markov chain $\mathcal{M} \times \mathcal{A}_\alpha$.

6.1 The Program Automaton $\mathcal{A}_\alpha$

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 15 (NPA). *An NPA is a quintuple*

$$\mathcal{A} = (Z, \Sigma', \delta, Z_0, F),$$

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

We now describe how a program automaton $\mathcal{A}$ can be used to describe the path set $\mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\alpha)$ for a program α. Thus, we consider NPA as *acceptors for finite paths* in $\mathcal{M}$ (rather than as acceptors for finite words over the alphabet Σ'). The intuitive behavior of an NPA $\mathcal{A}$ 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 automaton state is z, then $\mathcal{A}$ 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 $b = a$, $\mathcal{A}$ proceeds in the same way for state z' and the path σ'. In the

6. Σ is the alphabet of programs as defined in Definition 6.

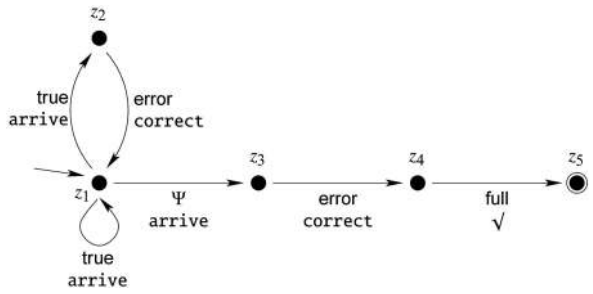

Fig. 4. NPA $\mathcal{A}_\alpha$ for the program α in Example 9.

former case, i.e., if $b = \surd$, no input symbol is consumed, i.e., the procedure is repeated with state z' and the path σ. If there is no outgoing transition from z which can be taken for the input path σ, then $\mathcal{A}$ rejects. As soon as $\mathcal{A}$ reaches a final state (a state in F) and the whole input path has been consumed, the automaton accepts.

The acceptance for paths is defined by means of runs, which are sequences of automaton states that can be generated by the operational behavior of $\mathcal{A}$ as sketched above, as defined formally in the following definition:

Definition 16 (Runs in NPA, accepted paths). *Let $\mathcal{A}$ be an NPA and $\mathcal{M}$ an ASMC as above, $z \in Z$, and σ a finite path in $\mathcal{M}$. Then, we define* $\mathrm{Runs}(z, \sigma)$ *as the greatest set of sequences $z, z_1, \ldots, z_n \in Z^+$ such that the following two conditions are fulfilled:*

1. $z \in \mathrm{Runs}(z, \sigma)$ *iff* $|\sigma| = 0$
2. *If* $z, z_1, \ldots, z_n \in \mathrm{Runs}(z, \sigma)$ *and* $n \geq 1$, *then there exists* $\phi b \in \Sigma'$ *such that*
   - $z_1 \in \delta(z, \phi b)$,
   - $\sigma[0] \models \phi$,
   - *if* $b \in \mathrm{Act}$, *then* $\sigma = s \xrightarrow{b,t} \sigma'$ *with* $z_1, \ldots, z_n \in \mathrm{Runs}(z_1, \sigma')$, *and*
   - *if* $b = \surd$, *then* $z_1, \ldots, z_n \in \mathrm{Runs}(z_1, \sigma)$.

The existence of such a greatest set follows by Tarski's fixed-point theorem for monotonic operators $2^{Z^*} \to 2^{Z^*}$.

Let $Z' \subseteq Z$. The elements of $\mathrm{Runs}(Z', \sigma) = \bigcup_{z \in Z'} \mathrm{Runs}(z, \sigma)$ *are called runs for σ in $\mathcal{A}$ 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,* $\mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\mathcal{A})$, *denotes the set of finite paths in $\mathcal{M}$ that have an accepting run in $\mathcal{A}$.*

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

Proposition 17. *If $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A})$, then*

$$\mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\alpha) = \mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\mathcal{A}).$$

Example 18. Fig. 4 shows an NPA $\mathcal{A}_\alpha$ for the language defined by the program α in Example 9.

State z_1 is the single initial state, that is, $Z_0 = \{z_1\}$. The set of final states F consists of the single state z_5.

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

Remark 19. An NPA $\mathcal{A} = (Z, \Sigma', \delta, Z_0, F)$ 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 $\mathcal{A}$, the "behavior" of $\mathcal{A}$ for an *input word* over Σ' is deterministic, i.e., there is at most one run, whereas the "behavior" of a deterministic NPA $\mathcal{A}$ for an *input path σ* can be *nondeterministic*, even if $\mathcal{A}$ does not contain $\surd$-transitions (i.e., transitions that are labeled 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 relation $\hat{\delta}^{\mathcal{M}}$. If σ is a path of length 0, i.e., $\sigma = s$ for some state s, then $\hat{\delta}^{\mathcal{M}}(Z', \sigma) = \hat{\delta}^{\mathcal{M}}(Z', s)$ consists of all automaton states $\tilde{z}$ that are reachable in $\mathcal{A}$ 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 20 ($\surd$-closure). *$\surd Closure(Z', s)$ denotes the least subset of Z such that*

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

The existence of this least set follows from Tarski's fixed-point theorem for monotonic operators $2^Z \to 2^Z$.

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

Definition 21 (Extended transition function). *The function $\hat{\delta}^{\mathcal{M}} : 2^Z \times \mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}} \to 2^Z$ is given by*

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

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

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

Note that Y stands for the set of all automaton states y that are reachable in $\mathcal{A}$ from a state $z' \in Z'$ via transitions labeled 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 $\hat{\delta}^{\mathcal{M}}(Z', \sigma)$ consists of all states that are reachable in $\mathcal{A}$ via a run starting in Z' for σ, i.e.,

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

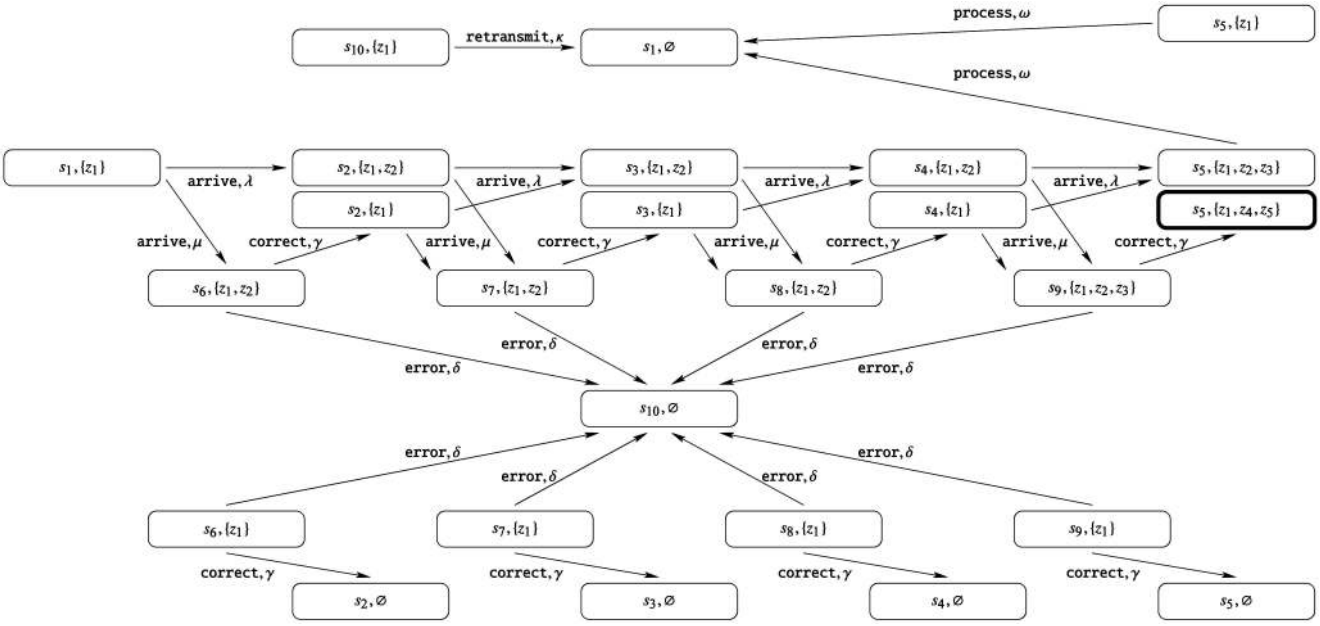

Fig. 5. Product Markov chain.

For $Z' = Z_0$, we obtain that $\hat{\delta}^{\mathcal{M}}(Z_0, \sigma)$ consists of all automaton states that can be reached via an initial run for σ. From this observation, we obtain:

Proposition 22. *If α is a program and $\mathcal{A}$ an NPA with $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A})$, then we have*

$$\mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\alpha) = \left\{\sigma \in \mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}} \mid \hat{\delta}^{\mathcal{M}}(Z_0, \sigma) \cap F \neq \emptyset\right\}.$$

Example 23. We consider the ASMC of the running example shown in Fig. 1 and the program α of Example 9. The corresponding program automaton $\mathcal{A} = \mathcal{A}_\alpha$ is shown in Fig. 4. Consider the finite path

$$\sigma_1 = s_2 \overset{\text{arrive},\cdot}{\longrightarrow} s_3 \overset{\text{arrive},\cdot}{\longrightarrow} s_4 \overset{\text{arrive},\cdot}{\longrightarrow} s_9 \overset{\text{correct},\cdot}{\longrightarrow} s_5.$$

with arbitrary sojourn times in the states. We then have

$$\hat{\delta}(\{z_1\}, \sigma_1) = \{z_1, z_4, z_5\}.$$

The path σ_1 belongs to $\mathrm{Path}^{\mathcal{M}}_{\mathrm{fin}}(\mathcal{A}_\alpha)$, because

$$z_5 \in \hat{\delta}(\{z_1\}, \sigma_1).$$

For the following path,

$$\sigma_2 = s_1 \overset{\text{arrive},\cdot}{\longrightarrow} s_6 \overset{\text{no\_correct},\cdot}{\longrightarrow} s_{10} \overset{\text{retransmit},\cdot}{\longrightarrow} s_1,$$

we have

$$\hat{\delta}(\{z_1\}, \sigma_2) = \emptyset.$$

This path is not contained in $\mathrm{Paths}^{\mathcal{M}}_{\mathrm{fin}}(\mathcal{A}_\alpha)$.

6.2 The Product Markov Chain $\mathcal{M} \times \mathcal{A}$

We now return to the question of how to calculate the satisfaction set $\mathrm{Sat}^{\mathcal{M}}(\phi)$ with $\phi = \mathcal{P}_{\bowtie p}(\alpha^I)$. First, we recursively apply an asCSL-model checking algorithm to the state formulas that occur in the program α. As soon as the satisfaction sets $\mathrm{Sat}^{\mathcal{M}}(\psi)$ 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 $\mathcal{A}$ for α (viewed as an ordinary regular expression over the alphabet Σ). We then consider $\mathcal{A}$ as an NPA and build the product of the ASMC $\mathcal{M}$ and $\mathcal{A}$ (which is defined below) and finally apply a CSL model checking algorithm to $\mathcal{M} \times \mathcal{A}$ to calculate the probability to reach a final automaton state within the given time interval I.

Definition 24 (Product Markov chain $\mathcal{M} \times \mathcal{A}$). *Let $\mathcal{M} = (S, \mathrm{Act}, \mathrm{AP}, L, \mathbf{R})$ be an ASMC and let $\mathcal{A} = (Z, \Sigma, \delta, Z_0, F)$ be an NPA. The product ASMC is defined as*

$$\mathcal{M} \times \mathcal{A} = (S^\times, \mathrm{Act}^\times, \mathrm{AP}^\times, L^\times, \mathbf{R}^\times)$$

with

- $S^\times = \{\langle s, Z'\rangle \mid s \in S \wedge Z' \in 2^Z\}$,
- $\mathrm{Act}^\times = \mathrm{Act}$, *and*
- $\mathrm{AP}^\times = \mathrm{AP} \cup \{\mathrm{accept}\}$ *(where* $\mathrm{accept} \notin \mathrm{AP}$*).*

The labeling function is defined by

$$L^\times(\langle s, Z'\rangle) = \begin{cases} L(s) \cup \{\mathrm{accept}\}, & \text{if } Z' \cap F \neq \emptyset \\ L(s), & \text{otherwise.} \end{cases}$$

The rate matrix is given 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 $Z_2 = \hat{\delta}^{\mathcal{M}}(Z_1, s_1 \overset{a,\cdot}{\longrightarrow} s_2)$, and $\mathbf{R}^\times(\cdot) = 0$ otherwise.

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

Example 25. Fig. 5 shows the product Markov chain resulting from the ASMC in Example 2 and the automaton in Example 18. Recall that s_4 is the only ASMC state satisfying Ψ. Only product states reachable from one of the "initial" states $\langle s, Z_0\rangle$ are shown. There is exactly one state labeled with accept where the automaton component contains the final state z_5; in Fig. 5, it is drawn in bold. Any transitions leaving the final state or

one of the sink states (the automaton component is $\emptyset$) are omitted.

Our goal is to show that the values $\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I)$ can be calculated using a model checking procedure for $\mathcal{M} \times \mathcal{A}$ and the simpler path formula $\diamond^I \mathrm{accept}$ (which means that a state labeled with the atomic proposition accept will be reached at some point in the time interval I). To establish this result, we first observe that $\mathcal{M}$ and $\mathcal{M} \times \mathcal{A}$ are statewise bisimulation equivalent when the set of atomic propositions in $\mathcal{M} \times \mathcal{A}$ is restricted to AP, i.e., we deal with the labeling function $L^{\times}_{\mathrm{AP}}$ which is given by $L^{\times}_{\mathrm{AP}}(\langle s, Z\rangle) = L(s)$ rather than $L^{\times}$. This follows by the fact that the coarsest equivalence $\mathcal{R}$ 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 $\mathcal{M}$ and all subsets Z' of Z. Using (2), we obtain:

Proposition 26. *For any state s of $\mathcal{M}$, we have*

$$\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I) = \mathrm{Prob}^{\mathcal{M}\times\mathcal{A}}(\langle s, Z_0\rangle, \alpha^I).$$

Next, we observe the one-to-one-correspondence between paths in $\mathcal{M}$ and paths in $\mathcal{M} \times \mathcal{A}$ (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 $\mathcal{M} \times \mathcal{A}$, one obtains a path in $\mathcal{M}$. Vice versa, each finite path

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

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

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

where, for $k = 1, \ldots, n$

$$Z_k = \hat{\delta}^{\mathcal{M}}\Big(Z_{k-1}, s_{k-1} \xrightarrow{a_{k-1},t} s_k\Big).$$

Hence, if $\mathcal{L}(\alpha) = (\mathcal{A})$ then (by Proposition 22):

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

Thus, for all infinite paths $\varsigma^{\times} \in \mathrm{Paths}^{\mathcal{M}\times\mathcal{A}}_{\omega}(\langle s, Z_0\rangle)$, we have

$$\mathcal{M} \times \mathcal{A}, \varsigma^{\times} \models \diamond^I \mathrm{accept} \text{ iff } \mathcal{M} \times \mathcal{A}, \varsigma^{\times} \models \alpha^I.$$

Hence, for all states s in $\mathcal{M}$ we have

$$\mathrm{Prob}^{\mathcal{M}\times\mathcal{A}}(\langle s, Z_0\rangle, \alpha^I) \;=\; \mathrm{Prob}^{\mathcal{M}\times\mathcal{A}}(\langle s, Z_0\rangle, \diamond^I \mathrm{accept}).$$

Using this observation and Proposition 26, we obtain the following theorem:

Theorem 27. *If α is an* asCSL *program, $\mathcal{A}$ an NPA with $\mathcal{L}(\alpha) = \mathcal{L}(\mathcal{A})$, and s a state in $\mathcal{M}$, then*

$$\mathrm{Prob}^{\mathcal{M}}(s, \alpha^I) \;= \mathrm{Prob}^{\mathcal{M}\times\mathcal{A}}(\langle s, Z_0\rangle, \diamond^I \mathrm{accept}).$$

In the case $I = [0, t]$, the eventually operators of asCSL and CSL agree (Proposition 12). Theorem 27 then states that the problem of computing the satisfaction set $\mathrm{Sat}^{\mathcal{M}}(\phi)$ for the asCSL-formula $\phi = \mathcal{P}_{\bowtie p}(\alpha^{\leq t})$ can be reduced to the problem of calculating the satisfaction set $\mathrm{Sat}^{\mathcal{M}\times\mathcal{A}}(\phi_{\mathrm{CSL}})$ for the CSL-state formula $\phi_{\mathrm{CSL}} = \mathcal{P}_{\bowtie p}(\diamond^{\leq t}\mathrm{accept})$, as illustrated in Fig. 6. In summary, to calculate $\mathrm{Sat}^{\mathcal{M}}(\phi)$ where ϕ is as above, we

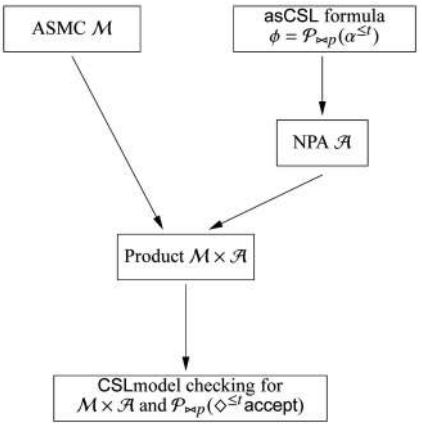

Fig. 6. Schema for the handling of the probabilistic path operator.

- apply standard techniques to generate a nondeterministic finite automaton $\mathcal{A}$ for α (viewed as an ordinary regular expression over the alphabet Σ),
- calculate the product ASMC $\mathcal{M} \times \mathcal{A}$, where it suffices to calculate the reachable part of $\mathcal{M} \times \mathcal{A}$ 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 rates of "parallel" transitions are accumulated,
- apply a CSL model checker to calculate the values $p_s = \mathrm{Prob}^{\mathcal{M}\times\mathcal{A}}(\langle s, Z_0\rangle, \diamond^{\leq t}\mathrm{accept})$ for all states s in $\mathcal{M}$, e.g., with the help of a transient analysis of the Markov chain, which is obtained from $\mathcal{M} \times \mathcal{A}$ when all states labeled with accept and all states from which one cannot reach a state labeled with accept (especially those that have an empty automaton part) are made absorbing [21], [4], and
- return the set $\{s \in S \,|\, p_s \bowtie p\}$.

Example 28. We want to check the formula $\Phi = \mathcal{P}_{\leq 0.1}(\alpha^{\leq 7.3})$ for α as defined in Example 9 on the running example ASMC $\mathcal{M}$ shown in Fig. 1. An automaton $\mathcal{A}_\alpha$ for the program α has been shown in Fig. 4. Fig. 5 presents the resulting product ASMC $\mathcal{M} \times \mathcal{A}_\alpha$. Let $\lambda = 9$, $\mu = 1$, $\gamma = 3$, $\delta = 1$, $\omega = 2$, and $\kappa = 20$. Applying a CSL model checker (which uses uniformization to compute the transient probabilities) results in the following probabilities:

$$p_{s_1} = 0.0695,\ p_{s_2} = 0.0713,\ p_{s_3} = 0.0731,\ p_{s_4} = 0.075$$

for states, where packets can arrive and

$$p_{s_5} = p_{s_6} = p_{s_7} = p_{s_8} = p_{s_9} = p_{s_{10}} = 0.$$

for states where the program cannot be followed for structural reasons. Since all these probabilities are ≤ 0.1 the satisfaction set $\mathrm{Sat}^{\mathcal{M}}(\Phi) = S$.

In the case $I = [t, t']$, where $t > 0$, the model checking procedure for $\mathcal{P}_{\bowtie p}(\alpha^I)$ has to be modified as follows: For

every state $x = (s, Z')$ of the product Markov chain with $\text{accept} \in L^{\times}(x)$, a duplicate state $\overline{x}$ with $L^{\times}(\overline{x}) = \{\overline{\text{accept}}\}$ is generated, where $\overline{\text{accept}}$ is a new atomic proposition. These duplicate states are made absorbing (note that x is not necessarily absorbing). The analysis now consists of two phases:

1. In the first phase, the duplicate states are not yet reachable. A transient analysis for time point t is carried out, yielding a probability vector $\vec{\pi}(t)$, where, for each duplicate state $\overline{x}$, the corresponding probability $\pi_{\overline{x}}(t)$ is zero (because these states are unreachable).
2. In the second phase, the product Markov chain is modified by redirecting all incoming arcs of a state $x = (s, Z')$ with $\text{accept} \in L^{\times}(x)$ to the corresponding duplicate state $\overline{x}$. On this modified Markov chain, a transient analysis for the time point $t' - t$ is carried out, taking the vector $\vec{\pi}(t)$ obtained in the first phase as the initial distribution. This yields the probability vector $\vec{\pi}(t' - t)$ from which the final result is computed as

$$\text{Prob}^{\mathcal{M}}(s, \alpha^I) = \sum_{\overline{x} \models \overline{\text{accept}}} \pi_{\overline{x}}(t' - t).$$

In [4], it was shown that the time complexity of the uniformization-based model checking algorithm for CSL formulas of type $\mathcal{P}_{\bowtie p}(\phi_1 \mathcal{U}^{[t,t']} \phi_2)$ is $\mathcal{O}(M \cdot q \cdot t')$, where M is the number of transitions in the CTMC and q is the uniformization rate (given by the largest exit rate of a state in the CTMC). In our approach, an asCSL formula of type $\mathcal{P}_{\bowtie p}(\alpha^I)$ is checked by first constructing an NPA $\mathcal{A}_\alpha$, which has $|Z| = \mathcal{O}(|\alpha|)$ states, and then constructing the product Markov chain, which has at most $M \cdot 2^{|Z|}$ transitions. The uniformization 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 $\mathcal{P}_{\bowtie p}(\alpha^{[t,t']})$ is bounded by $\mathcal{O}(M \cdot 2^{|\alpha|} \cdot q \cdot t')$.

7 Handover in a Cellular Mobile Communication Network

In this section, we present an elaborated example in order to illustrate the techniques we have developed. We consider a scalable cellular mobile communication network. Each cell is ruled by a base station subsystem (BSS). We are especially interested in the behavior of the system concerning a distinguished mobile radio station (MS) (also called the distinguished user) moving from one cell to another, thereby possibly triggering a so-called handover procedure. Handovers between the different cells are managed by the corresponding BSSs and the global mobile switching center (MSC). Depending on the load of the MSC and the availability of channels at the BSSs, a handover might succeed or fail. The model is inspired by the description of the GSM handover procedure in [29] and [30]. We describe the system as a set of synchronizing processes, namely, the switching center, the distinguished user's spatial movement and the user's functional behavior. The properties of interest are expressed with asCSL formulas involving programs. We show the corresponding NPA and relate the size of the resulting product Markov chains to the size of the original model. Finally we use a CSL-model checking tool to evaluate the formulas.

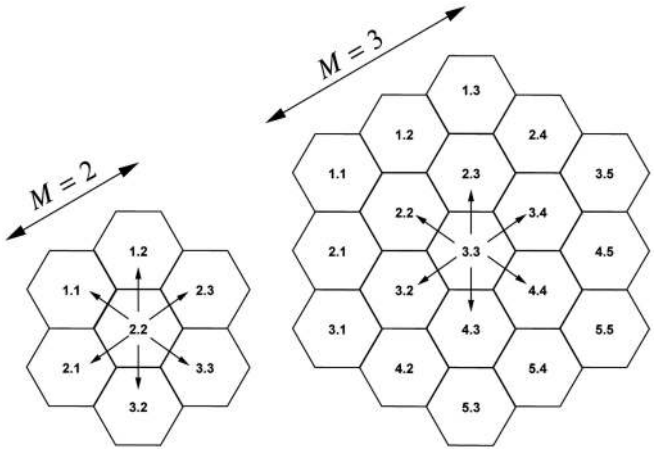

Fig. 7. Hexagon of cells for $M = 2$ and $M = 3$.

7.1 The Model

The distinguished MS is situated in one of several GSM cells and is allowed to move between neighboring cells. Each cell has a hexagonal shape. Together, cells are arranged in such a way that they again form a hexagon. The size of this hexagon is described by the number M of cells that constitute one edge of it. In Fig. 7, one can see the cell topologies for $M = 2$ and $M = 3$. It also illustrates the unique cell identifiers. The parameter M is used for scaling the model; the complete hexagon has $M^2 + M(M-1) + (M-1)^2 = 3M(M-1) + 1$ cells. We now describe the model of the MS functional behavior. When not active with a connection, the MS is idle. At any time, the MS can become active, meaning that it either accepts or establishes a (radio) connection. After a while, the connection can be 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 handover is eventually completed, the MS returns to the active state (note that the connection is continued during the entire handover procedure). If the handover procedure is not completed in time, the connection is lost. The connection is then terminated (assume that the distance to the former cell has become too large) and the MS returns to the idle state.

Fig. 8 shows a state-transition diagram for the distinguished MS. Transitions are labeled with action names. Note that, in ASMCs, we allow more than one transition between two states as long as they are labeled with different action names. Such "parallel" (or coexisting) transitions (receive and activate) can be found between states Idle and Active. The `move` transition synchronizes with the user's spatial movement whenever active.

The mobile services switching center (MSC) is modeled by its load. It has low, medium, or high load. The time needed for the handover procedure depends on the current load. Under high load, the MSC does not process any request for handover at all. Table 1 states the rates for

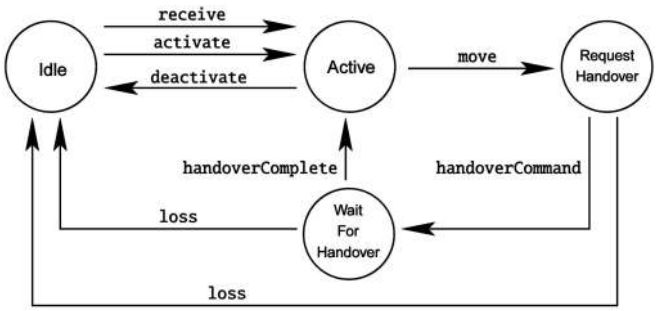

Fig. 8. State machine for the distinguished MS behavior.

transitions labeled with the given actions. Note that all numbers are educated guesses made on the basis of [31].

7.2 asCSL Properties

In the following section, we present several asCSL formulas, which are constructed with two goals in mind: On the one hand, they demonstrate the expressive power of asCSL. On the other hand, they formalize interesting properties of the handover procedure. For each formula, the model checking procedure involves the construction of a product ASMC. The results are interpreted in Section 7.4.

7.2.1 Move

The MS is always free to move from one cell to one of its neighboring cells. We ask whether the probability of moving within the next two minutes (120 seconds) is at least 98 percent. A program describing this behavior is

$$\alpha_a = (\text{true}, \text{Act}\backslash\{\texttt{move}\})^*; (\text{true}, \texttt{move}).$$

First, the ASMC is allowed to perform arbitrary transitions as long as they are not labeled `move`. If, then, a `move` transition occurs, the ASMC has shown the specified behavior. Fig. 9a shows an NPA for α_a.

The complete asCSL formula becomes

$$\phi_a = \mathcal{P}_{>0.98}(\alpha_a^{[0,120]}).$$

In this case, we could still state a CSL formula that has the same meaning. Moving is equivalent to being in one cell at one moment and in another cell at the next moment. So, the following CSL path formula describes moving out of a cell (i, j) within 120 seconds:

$$\varphi(i,j) = \text{InCell(i,j)}\ \text{U}^{[0,120]} \neg \text{InCell}(i,j).$$

A CSL formula equivalent to ϕ_a is then

$$\psi = \bigvee_{i,j} \mathcal{P}_{>0.98}(\varphi(i,j)).$$

It has to account for every cell the MS might be in. This makes the formula lengthy. We think that the asCSL version is much more readable and elegant. Note, however, that property ϕ_a can be expressed (in a straightforward manner) in aCSL by a single until operator, decorated with the action `move` as the final action.

7.2.2 Inbound Connection

In this paragraph, we describe an asCSL formula that relies on a special feature of ASMCs: the possibility of having more than one transition between two states. In our model, both transitions `activate` and `receive` lead from state Idle to state Active. We can never find out whether a connection is inbound or outbound just by looking at state properties, unless we split (duplicate) states. Only the transitions tell us what is the case. The following program has a similar structure to that of α_a but cannot be replaced by a CSL path formula:

$$\alpha_b = (\text{true}, \text{Act}\backslash\{\texttt{activate}, \texttt{receive}\})^*; (\text{Idle}, \texttt{receive}).$$

See an NPA for α_b in Fig. 9b. With $\phi_b = \mathcal{P}_{\geq 0.3}\left(\alpha_b^{[0,\ 2500]}\right)$, we check whether the probability of receiving an inbound connection within the next 2,500 seconds (without activating an outbound call) is at least 30 percent. Consider also the following asCSL formula: $\phi_{\mathcal{S}} = \mathcal{S}_{\geq 0.85}(\phi_b)$. It holds if the steady-state probability of states that satisfy ϕ_b is at least 85 percent.

7.2.3 Outdated 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 behavior 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

TABLE 1
Action Labels and Rates of the Transitions of the Cellular Network Model

process	action	rate	description
MS position	move	0.02	from cell (i, j) to up to six neighboring cells (equi-probable) (on average, 50 seconds residence per cell)
MS behavior	activate	0.0006250	average time between outbound connections is 1600 seconds
	receive	0.0003125	average time between inbound connections is 3200 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	connection has been transferred to new cell
	loss	0.1	might happen during handover procedure
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

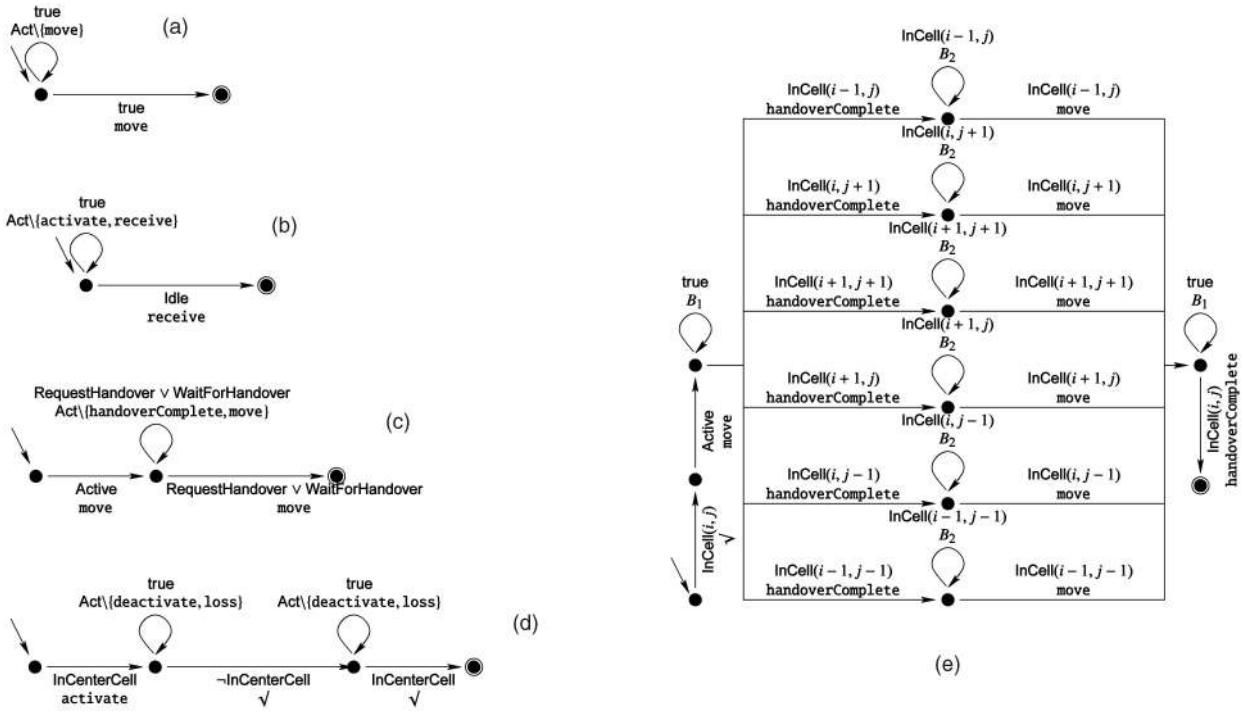

Fig. 9. NPA for the programs $\alpha_a, \ldots, \alpha_d$ and $\beta(i,j)$.

movement could cause a problem. So, we would like to know whether the probability of such an outdated handover is lower than, say, 3.5 percent. As an asCSL formula, this becomes:

$$\phi_c = \mathcal{P}_{\leq 0.035}(\alpha_c^{[0,\,\infty]}),$$

with

$$\alpha_c = (\text{Active}, \texttt{move}); \quad (3)$$
$$(\text{RequestHandover} \vee \text{WaitForHandover}, \quad (4)$$
$$\text{Act}\backslash\{\texttt{handoverComplete}, \texttt{move}\})^*; \quad (5)$$
$$(\text{RequestHandover} \vee \text{WaitForHandover}, \texttt{move}). \quad (6)$$

A move while the MS is active triggers a handover. Lines (4/3) describe the system inside the handover procedure. A move (6) leads to an outdated handover. An NPA for the program α_c is given in Fig. 9c.

7.2.4 Return without Interruption

Assume that the MS initiates a connection while in the center cell (M, M). It is free to move between cells. We would like it to leave the center cell and to return within 10 minutes (600 seconds) without terminating or losing the connection. Is the probability for this scenario at least 10 percent? Coded into an asCSL-formula this reads $\phi_d = \mathcal{P}_{>0.1}\left(\alpha_d^{[0,\,600]}\right)$, with

$$\alpha_d = (\text{InCenterCell}, \texttt{activate}); \quad (7)$$
$$(\text{true}, \text{Act}\backslash\{\texttt{deactivate}, \texttt{loss}\})^*; \quad (8)$$
$$(\neg\text{InCenterCell}, \surd); \quad (9)$$
$$\text{true}, \text{Act}\backslash\{\texttt{deactivate}, \texttt{loss}\})^*; \quad (10)$$
$$(\text{InCenterCell}, \surd). \quad (11)$$

The regular expression first ensures that the user activates a connection while being in the center cell (7). Then, the user can behave arbitrarily, as long as the connection is not ended via a `deactivate` or `loss` event (8). At some time, the user must have left the center cell (9) and can again behave arbitrarily, as long as the connection remains established (10). Finally, he should return to the center cell (11). Fig. 9d shows an NPA for the program α_d.

7.2.5 Ping-Pong

Sometimes there are handovers from a cell (i,j) to a neighboring cell (i',j') and back to cell (i,j) 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,j).

A ping-pong between cell (i,j) and its neighboring cells is described by the program for $\beta_{(i,j)}$:

$$(\text{InCell}(i,j), \surd); (\text{Active}, \texttt{move}); (\text{true}, B_1)^*;$$

$$\left(\bigcup_{(i',j') \text{ neighbor of } (i,j)} \begin{array}{l} ((\text{InCell}(i',j'), \texttt{handoverComplete}); \\ (\text{InCell}(i',j'), B_2)^*; \\ (\text{InCell}(i',j'), \texttt{move})) \end{array} \right);$$

$$(\text{true}, B_1)^*; (\text{InCell}(i,j), \texttt{handoverComplete}),$$

where

$$B_1 = \text{Act}\backslash\{\texttt{move}, \texttt{loss}, \texttt{handoverComplete}\}$$

and $B_2 = \text{Act}\backslash\{\texttt{deactivate}, \texttt{move}\}$. If (i,j) is an inner cell, that is, has all six neighbors, an NPA for the program $\beta_{(i,j)}$ is given in Fig. 9e. All possible ping-pong situations are described by $\alpha_e = \bigcup_{(i,j)} \beta_{(i,j)}$. The NPA for α_e consists of one replica of the automaton in Fig. 9e for each cell (i,j). It has

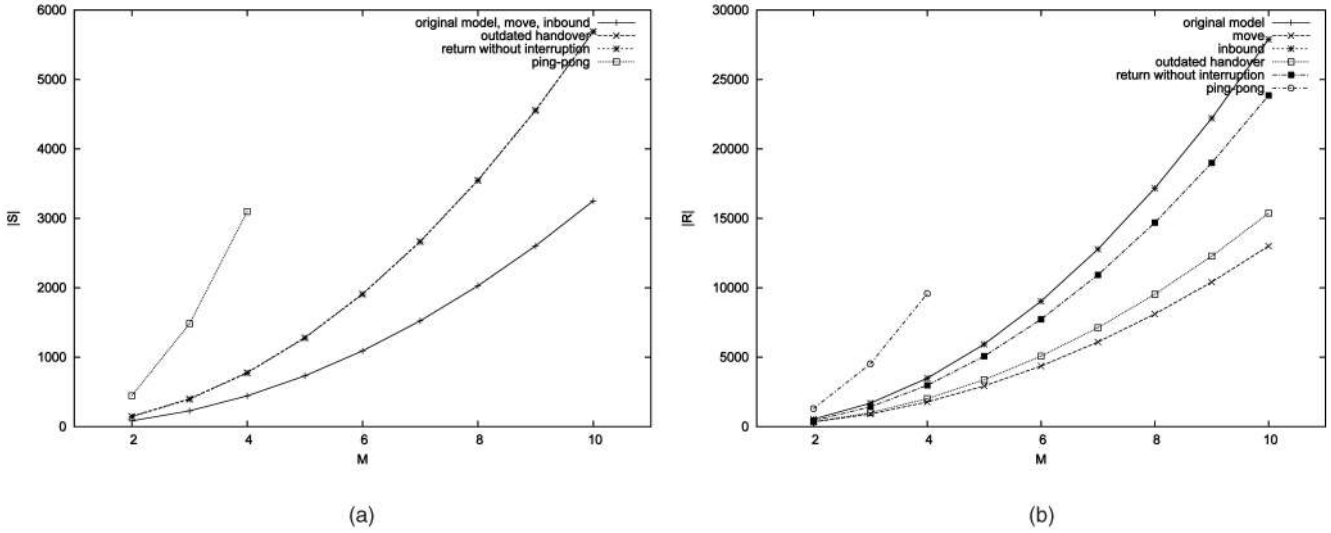

Fig. 10. Size of (a) state space and (b) number of transitions in the original ASMC and the product Markov chains.

an initial and a final state for each cell. The asCSL-formula $\phi_e = \mathcal{P}_{\leq 0.01}(\alpha_e^{[0,10]})$ formalizes the following question: "During an active connection, is the probability of such a ping-pong handover to occur within 10 seconds at most 1 percent?"

7.3 Tool Support/Implementation

A prototype that performs the construction of the product Markov chain given an ASMC and an NPA has been implemented in C++.

For modeling and evaluation, we employ a stochastic Petri net (SPN) model of the cellular system. All components of the systems are described by simple state machines; we therefore do not show their SPN representation here. The SPN is described in an extension of CSPL [32], 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 [33] has been extended in order to record these properties and the transition names (as action labels) and generates an ASMC, including any coexisting transitions. A state of the ASMC generated for this example is a triple consisting of the current cell the MS resides in, the state of the MS, and the load of the MSC. An example for such a state is ((2.2), Idle, low). The actions in Act are listed in Table 1. The set of atomic propositions AP is given by the possible states of the MS, that is, Idle, Active, RequestHandover, and WaitForHandover, and atomic propositions related to the position of the MS, that is, InCell(i, i) or InCenterCell.

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

The final computation of the satisfaction relation of the corresponding CSL formula is done using CSL model checking procedures. The size of the ASMCs was restricted by the runtime of the prototype implementation computing the reduced product Markov chain.

7.4 Results

7.4.1 Product Markov Chains.

Fig. 10a shows the number of states and (Fig. 10b shows the number of 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 given asCSL-formulas as a function of the number M of cells per hexagon edge that ranges from 2 to 10.

The original model has 3,252 states and 27,900 transitions for $M = 10$. For all presented programs, the number of states in the product Markov chain is equal to or larger than in the original ASMC. This could be expected, since the state space is a subset of the Cartesian product $S \times 2^{|Z|}$. For the "move" (Section 7.2.1) and "inbound connection" programs (Section 7.2.2), the state space is the original state space plus the two special merge states for rejecting and accepting states. No additional states are created because , after allowing arbitrary behavior, the automata go directly to their final states once the decisive action (`move` or `receive`) occurs. For the programs of "outdated handover" (Section 7.2.3) and "return without interruption" (Section 7.2.4), the size of the state space is roughly scaled by a factor of 1.75. This is the result of having more than one automaton state visited before reaching a final state.

The largest state space is the one of the ping-pong property (Section 7.2.5); it has more than 44,000 states for $M = 10$. Because 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 original state space. However, with the presented examples and formulas, this is not the case. The program for "inbound connection" leads to a product Markov chain in which there is exactly one transition for each transition in the original model. Transitions labeled `receive` now lead into the newly created accept state, transitions labeled `activate` lead into the reject state.

Even though the program for "move" has exactly the same structure as the program for "inbound connection," it generates fewer transitions. This is because of the merging of `accept` states into one state, which causes also all `move`

transitions (up to six) leaving a state to be aggregated into a single transition.

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 as well. 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 broad range of different combinations of states and transitions. Consequently, it shows the largest growth in state space, and the number of transitions is much larger than in the original model (147,175 for $M = 10$).

7.4.2 Model Checking

Applying a CSL model checker to the "move" product Markov chains reveals that all states of the ASMC satisfy the "move" formula ϕ_a for all parameters M. This is not surprising, since the `move` transition exists in every state and the mean time between two moves is $1/0.02 = 50$ seconds. This results in a sufficiently high probability of moving within two minutes.

The "inbound connection" formula ϕ_b is only satisfied by part of the states. That means that for some of the states the probability of having an inbound call within the next 2,500 seconds is less than 30 percent. To see the satisfaction of ϕ_b on the long run, we consider the formula $\phi_{\mathcal{S}} = \mathcal{S}_{\geq 0.85}(\phi_b)$. Since the ASMC is strongly connected, the satisfaction set of $\phi_{\mathcal{S}}$ is either empty or equals the complete state space. For $M = 2, \ldots, 6$, no state satisfies the steady-state formula. The accumulated steady-state probability for all ϕ_b-states is smaller than 85 percent. For $M = 7, \ldots, 10$, all states satisfy formula $\phi_{\mathcal{S}}$.

Not all states satisfy the "outdated handover" formula. For states where the MS is `active` and the MSC has low load, there are some states that do not fulfill ϕ_c. The cells are arranged in rings around the center cell (M, M), as can be seen in Fig. 7. If the MS resides in one of the $M - 2$ inner rings (and is active and the MSC load is low), the probability of following the behavior defined by α_c is above 3.5 percent and the state does not satisfy ϕ_c.

Formula ϕ_d ("return without interruption") is not valid in any state. For most of the states, the probability of following a path specified by α_d is 0 anyway because the MS is not in the center cell. But also for those states where the MS is in the center cell, the probability of returning with an ongoing call is too small to meet the bound.

Finally, in all states of the ASMC the "ping-pong" formula ϕ_e holds. This is not surprising when making a comparison with the result of checking ϕ_d: Already, the less restrictive specification of returning to the same cell yields very low probabilities and the probability of having a ping-pong handover is even always below 1 percent.

8 Conclusions

In this paper, we introduced the logic asCSL as a new temporal logical framework for reasoning about performance and dependability measures for Markov chains with both action labels and state labels. asCSL subsumes CSL (with time intervals $[0, t]$) as well as several other logics that have recently been suggested in the literature, such as aCSL, aCSL+ [7], [11], [12], [13]. Although the proposed logic asCSL is quite expressive, it still yields a simple and intuitive specification formalism, as illustrated by the example provided in Section 7, where complex properties referring to both state labels and actions have been formalized by means of rather simple asCSL formulas.

The model checking problem for asCSL can be solved by a procedure that combines well-known techniques for finite automata and for verifying continuous-time Markov chains. The calculation of the satisfaction set for formulas of type $\mathcal{P}_{\bowtie p}(\alpha^I)$ relies on a reduction to the CSL model checking problem via a product construction of the Markov chain $\mathcal{M}$ and an automaton for the path formula α^I, while the treatment of other formulas is exactly as in CSL. Thus, established techniques and tools for CSL model checking are still applicable for reasoning about complex properties specified by asCSL formulas.

Acknowledgments

The work in this paper was developed in the context of the joint DFG-NWO project "Validation of Stochastic Systems," through DFG grants BA 1679/2 (University of Bonn) and SI 710/2 (University of the Federal Armed Forces, Munich), and NWO grant DN 62-600 (University of Twente).

References

[1] *Lectures on Formal Methods and Performance Analysis: Proc. First EEF/Euro Summer School on Trends in Computer Science,* E. Brinksma, H. Hermanns, and J.-P. Katoen, eds., 2001.

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

[3] A. Aziz, K. Sanwal, V. Singhal, and R. Brayton, "Verifying Continuous Time Markov Chains," *Proc. Conf. Computer-Aided Verification,* pp. 269-276, 1996.

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

[5] 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.

[6] C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen, "On the Logical Characterisation of Performability Properties," *Proc. Int'l Colloquium Automata, Languages, and Programming (ICALP '00),* pp. 780-792, 2000.

[7] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "Towards Model Checking Stochastic Process Algebra," *Proc. Conf. Integrated Formal Methods (IFM '00),* pp. 420-439, 2000.

[8] H. Hermanns, J.-P. Katoen, J. Meyer-Kayser, and M. Siegle, "Implementing a Model Checker for Performability Behaviour," *Proc. Fifth Int'l Workshop Performability Modelling of Computer and Comm. Systems,* pp. 110-115, 2001.

[9] R.D. Nicola and F. Vaandrager, "Three Logics for Branching Bisimulation (Extended Abstract)," *Proc. Fifth Ann. IEEE Symp. Logic In Computer Science,* pp. 118-129, 1990.

[10] S. Chaki, E. Clarke, J. Ouaknine, N. Sharygina, and N. Sinha, "State/Event-Based Software Model Checking," *Proc. Conf. Integrated Formal Methods (IFM),* 2004.

[11] J. Meyer-Kayser, "Automatische Verifikation Stochastischer Systeme," PhD dissertation, Universität Erlangen-Nürnberg, Institut für Informatik (in German), 2004.

[12] C. Baier, L. Cloth, B. Haverkort, H. Hermanns, and J.-P. Katoen, "Model Checking pathCSL," *Proc. Sixth Int'l Workshop Performability Modeling of Computer and Comm. Systems,* pp. 19-22, 2003.

[13] M. Kuntz and M. Siegle, "A Stochastic Extension of the Logic PDL," *Proc. Sixth Int'l Workshop Performability Modeling of Computer and Comm. Systems,* pp. 58-61, 2003.

[14] W. Obal and W. Sanders, "State-Space Support for Path-Based Reward Variables," *Performance Evaluation,* vol. 35, nos. 3-4, pp. 233-251, 1999.
[15] M. Fischer and R. Ladner, "Propositional Dynamic Logic of Regular Programs," *J. Computer and System Sciences,* vol. 8, pp. 194-211, 1979.
[16] P. Wolper, "Specification and Synthesis of Communicating Processes Using an Extended Temporal Logic," *Proc. Ninth Symp. Principles of Programming Languages,* pp. 20-33, 1982.
[17] A. Aziz, V. Singhal, F. Balarin, and R.K. Brayton, "It Usually Works: The Temporal Logic of Stochastic Systems," *Proc. Conf. Computer-Aided Verification,* pp. 155-165, 1995.
[18] C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle, "Model Checking Action- and State-Labelled Markov Chains}," *Proc. Int'l Symp. Dependable Systems and Networks,* pp. 701-710, 2004.
[19] W. Stewart, *Introduction to the Numerical Solution of Markov Chains.* Princeton Univ. Press, 1994.
[20] A. Fantechi, S. Gnesi, and G. Ristori, "Model Checking for Action Based Logics," *Formal Methods in System Design,* vol. 4, pp. 187-203, 1994.
[21] C. Baier, B. Haverkort, J.-P. Katoen, and H. Hermanns, "Model Checking Continuous-Time Markov Chains by Transient Analysis," *Proc. Conf. Computer-Aided Verification (CAV '00),* pp. 358-372, 2000.
[22] J. Desharnais and P. Panangaden, "Continuous Stochastic Logic Characterizes Bisimulation of Continuous-Time Markov Processes," *J. Logic and Algebraic Programming,* vol. 56, nos. 1-2, pp. 99-115, 2003.
[23] J. Kemeny and J. Snell, *Finite Markov Chains.* Springer, 1976.
[24] P. Buchholz, "Exact and Ordinary Lumpability in Finite Markov Chains," *J. Applied Probability,* no. 31, pp. 59-75, 1994.
[25] J. Hillston, "A Compositional Approach to Performance Modelling," PhD dissertation, Univ. of Edinburgh, 1994.
[26] H. Hermanns and M. Rettelbach, "Syntax, Semantics, Equivalences, and Axioms for MTIPP," *Proc. Workshop Process Algebras and Performance Modeling (PAPMP '94),* pp. 71-88, 1994.
[27] M. Bernardo and R. Cleaveland, "A Theory of Testing for Markovian Processes," *Proc. Conf. Concurrency Theory (CONCUR '00),* pp. 305-319, 2000.
[28] E. Clarke, E. Emerson, and A. Sistla, "Automatic Verification of Finite-State Concurrent Systems Using Temporal Logic Specifications," *ACM Trans. Programming Languages and Systems,* vol. 8, no. 2, pp. 244-263, 1986.
[29] B. Walke, *Mobile Radio Networks.* Wiley, 1999.
[30] J.M. Thomsen and R. Manggaard, "Analysis of GSM Handover Using Coloured Petri Nets," master's thesis, Univ. of Aarhus, 2003.
[31] J. Ventura Agustina, P. Zhang, and R. Kantola, "Performance Evaluation of GSM Handover Traffic in a GPRS/GSM Network," *Proc. Eighth IEEE Int'l. Symp. Computers and Comm.,* pp. 137-142, 2003.
[32] G. Ciardo, J. Muppala, and K. Trivedi, "SPNP: Stochastic Petri Net Package," *Proc. Third Int'l Workshop Petri Nets and Performance Models,* pp. 142-151, 1989.
[33] B. Haverkort, H. Bohnenkamp, and A. Bell, "On the Efficient Sequential and Distributed Evaluation of Very Large Stochastic Petri Nets," *Proc. Int'l Workshop Petri Nets and Performance Models,* pp. 12-21, 1999.

Christel Baier received the diploma degree in mathematics in 1990 from the University of Mannheim in Germany. She received the PhD degree in 1994 and the venia legendi in 1999, both from the Department of Computer Science at the University of Mannheim. From 1999 to 2006, she was an associate professor of computer science at the Rheinische Friedrich-Wilhelms Universit at Bonn. Since October 2006, she has been a full professor of computer science at the Technical University Dresden. Her research interests are the theory of concurrent and probabilistic systems, verification, semantics of programming languages, and process calculi and mathematical logic.

Lucia Cloth received the diploma degree in computer science in 2000 from the RWTH Aachen University and the PhD degree from the University of Twente in 2006. Currently, she is a researcher in the Faculty for Electrical Engineering, Mathematics, and Computer Science at the University of Twente. Her research interests include model checking of stochastic systems and the abstract modeling of batteries.

Boudewijn R. Haverkort received the engineering degree and the PhD degree in computer science, both from the University of Twente, in 1986 and 1991, respectively. Since 2003, he is the chairholder for Design and Analysis of Communication Systems at the University of Twente, the Netherlands. Prior to that, he was, among other things, a professor of performance evaluation and distributed systems at the RWTH Aachen, Germany, for seven years, a lecturer in computer science at the University of Twente in the Netherlands for five years, and a visiting researcher in the Teletraffic Research Centre at the University of Adelaide. His research interests emcompass the design and performance and dependability evaluation of computer-communication systems, model checking, parallel and distributed computing, and fault-tolerant computer systems. He has published more than 75 papers in international journals and conference proceedings, edited several books and conference proceedings, and wrote a monograph on model-based performance evaluation of computer and communication systems. Since 2005, he has served on the editorial board of *Performance Evaluation*. He is a fellow of the IEEE and the IEEE Computer Society.

Matthias Kuntz received the diploma and PhD degrees from Friedrich-Alexander-University Erlangen-Nuernberg in 2001 and 2006, respectively. From 2001 to 2003, he was a research assistant in the Department of Computer Networks and Communication Systems at Friedrich-Alexander-Universitaet Erlangen-Nuernberg. From 2004 to 2005, he was with the Design of Computer and Communication Systems Research Group at the University of Federal Armed Forces in Munich. Since 2005, he has been a postdoctoral researcher in the Department of Informatics at the University of Twente, the Netherlands. His research interests include formal verification, especially model checking of systems with stochastic behavior.

Markus Siegle received the Dipl-Inf degree from the University of Stuttgart (1989), the master's degree from North Carolina State University (1990), the Dr-Ing degree from the University of Erlangen-Nürnberg (1995), and the habilitation degree from the University of Erlangen-Nürnberg (2002), all in computer science. Since 2003, he has been a professor with the Institute of Computer Engineering at the University of the Federal Armed Forces in München, Germany, heading a research group with a focus on the design of computer and communications systems. Previously, he was a researcher and lecturer at the University of Erlangen-Nürnberg. His research interests include performance and dependability evaluation, formal specification methods, representation, and solution of very large Markovian models, symbolic data structures, and the verification of nonfunctional requirements.