

Parameter and Controller Synthesis for Markov Chains with Actions and State Labels

Bharath Siva Kumar Tati and Markus Siegle

Universität der Bundeswehr München, Dept. of Computer Science, Germany
Bharath.Tati@unibw.de, Markus.Siegle@unibw.de

Abstract

This paper introduces a novel approach for synthesizing parameters and controllers for Markov Chains with Actions and State Labels (ASMC). Requirements which are to be met by the controlled system are specified as formulas of asCSL, which is a powerful temporal logic for characterizing both state properties and action sequences of a labeled Markov chain. The paper proposes two separate – but related – algorithms for untimed *until* type and untimed general asCSL formulas. In the former case, a set of transition rates and a common rate reduction factor are determined. In the latter case, a controller which is to be composed in parallel with the given ASMC is synthesized. Both algorithms are based on some rather simple heuristics.

1998 ACM Subject Classification C.4 Performance of Systems, D.2.4 Software/Program Verification (Model checking), G.3 Probability and Statistics (Markov processes)

Keywords and phrases Markov chains with actions and state labels, Parameter synthesis, Controller synthesis, Probabilistic model checking

Digital Object Identifier 10.4230/OASICS.xxx.yyy.p

1 Introduction

Markov chains are widely used to model systems with stochastic behavior and to analyze their quantitative properties such as performance (e.g. utilization, throughput or response time) or dependability (e.g. availability or mean time to failure). Models are usually obtained by transforming from high-level descriptions such as Petri nets or process algebraic description etc. into low-level Markov chains. ASMCS are continuous-time Markov chains extended with actions and state labels. To specify requirements of ASMCS, the temporal logic asCSL [3], which is an extension of CSL [1, 4], has been developed. In particular, asCSL makes it possible to specify complex path-based behavior with the help of regular expressions over state properties and action labels. The process of model checking ASMCS is explained in [3].

The topic of this paper is how to create a controller (also called a supervisor) that controls a given ASMC (also called plant) such that it will satisfy the given path-based requirements specified in asCSL. To this aim, we first study the special case of untimed until-type formulas and then proceed to general untimed path-based requirements. In the former case, we propose an algorithm for parameter synthesis which determines a subset of the ASMC's transitions and a common factor by which those transition rates are to be reduced. In the latter case, a controller is synthesized via a product automaton construction borrowed from the asCSL model checking algorithm where, again, rate reduction plays an important part in this construction. Composing the controller in parallel with the original plant will ensure that the requirement is satisfied. It is important to note that the controller will not only change a set of transition rates, but also potentially change the structural behavior of the plant. We have made the deliberate decision to work with rate reduction factors (as opposed to rate acceleration), since we advocate that it is in general possible to slow down a process



© Bharath Siva Kumar Tati and Markus Siegle;
licensed under Creative Commons License CC-BY

Conference/workshop/symposium title on which this volume is based on.

Editors: Billy Editor and Bill Editors; pp. 1–14

OpenAccess Series in Informatics



OASICS Schloss Dagstuhl – Leibniz-Zentrum für Informatik, Dagstuhl Publishing, Germany

(e.g. by reducing the speed of a machine or the capacity of a server), whereas speeding up a process may not be possible, since this would require additional resources. The present paper only considers the controller synthesis problem for untimed asCSL properties, i.e. here we do not consider time-bounded or time-interval-bounded requirements.

Related work: As early as 1993, Lawford and Wonham described an algorithm for synthesizing probabilistic supervisors for a class of probabilistic discrete event systems where a subset of the events is controllable [13], initiating a strand of research that is still active today (see e.g. [14]). The related area of model checking parametric Markov chains has been studied for more than a decade [6]. Some of the work is devoted to the problem of how to deal with the growing symbolic size of the rational functions obtained for the reachability probabilities of interest. This was addressed in [8] for Markovian models with rewards and nondeterminism, and associated tools have been provided [7]. Some approaches for rate parameter synthesis use a discretization of one-dimensional or multi-dimensional parameter ranges over a grid, together with refinement and/or sampling techniques [9]. The recent paper [15] synthesizes rate parameters such that either a given CSL time-bounded property should hold or that the probability of satisfaction is maximized. Their algorithms rely on uniformization combined with the computation of lower and upper bounds as described in [5], and also use parameter range refinement and sampling. Our approach described in this paper is different in that we do not work with parametric Markov chains, but with Markov chains whose rates are given as constant values. Our problem then is to determine a subset of the transition rates to be modified, and a common reduction factor for those rates, such that a given requirement will be satisfied. In this paper, we consider only untimed requirements, but we deal with the full generality of asCSL-type path properties (without nested probabilistic path operators). This requires the synthesis of a controller which is to be composed in parallel with the given plant, thereby adapting the plant's behavior according to the requirement, possibly also changing its structural behavior.

The rest of the paper is organized as follows: Sec. 2 introduces the fundamental concepts used in this paper. Sec. 3 explains the algorithm to synthesize parameters for untimed until-type formulas, Sec. 4 explains the algorithm to synthesize both parameters and a controller for general untimed asCSL formulas, and Sec. 5 concludes the paper.

2 Preliminaries

This section explains the fundamental concepts used in rest of the paper. A Markov chain with state labels and actions (ASMC) is defined as follows [3]:

► **Definition 2.1 (ASMC).** An ASMC \mathcal{M} is a tuple (S, Σ, R, L) where

- S is a finite set of states
- Σ is the set of action labels over transitions
- $R : S \times \Sigma \times S \mapsto \mathbb{R}_{\geq 0}$, is the transition function
- $L : S \rightarrow 2^{AP}$ is a state labeling function, where AP is a finite set of atomic propositions

A finite untimed *path* σ in an ASMC \mathcal{M} is a finite sequence $\sigma = [(s_0, a_0), (s_1, a_1), \dots, (s_{n-1}, a_{n-1}), s_n] \in (S \times \Sigma)^* \times S$ and with $Paths(s)$ we denote the set of all finite paths originating from state s . Probabilities are assigned to sets of finite paths by the usual cylinder set construction. An ASMC without action labels is called a state-labeled CTMC. So, the underlying CTMC for an ASMC is given by the tuple (S, R', L) , which is a result of removing the action labels and accumulating the rates of parallel transitions, i.e., $R'(s, s') = \sum_{a \in \Sigma} R(s, a, s')$. In order to specify user requirements and characterize execution paths

of ASMCs, we use the logic asCSL [3] (without time bounds and without the steady-state operator), which is an extension of the purely state-based logic CSL (continuous stochastic logic) [4].

► **Definition 2.2** (State formulas of asCSL). The grammar for untimed asCSL state formulas is given as:

$$\Phi ::= q \mid \neg\Phi \mid \Phi \vee \Phi \mid P_{\sim b}(\alpha)$$

where $q \in AP$ is an atomic proposition, \neg denotes negation, \vee denotes disjunction, $b \in (0, 1)$ denotes a probability value, $\sim \in \{<, \leq, >, \geq\}$ a comparison operator and α is a program as defined in Def 2.3. $P_{\sim b}(\alpha)$ asserts that the probability measure of the set of paths satisfying α meets the bound given by $\sim b$. The program α specifies the property for finite paths.

In contrast to [3], this paper considers probability bounds $b \in (0, 1)$ instead of $b \in [0, 1]$, since the approach presented here does not aim to turn a non-zero probability into zero, or to turn a probability smaller than one into one. Thus, we do not treat requirements of the form $P_{\leq 0}(\alpha)$ or $P_{\geq 1}(\alpha)$, and for similar reasons we also do not treat requirements of the form $P_{> 0}(\alpha)$ or $P_{< 1}(\alpha)$.

► **Definition 2.3** (Program). asCSL-programs are defined by the following grammar:

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

Formally, programs are regular expressions over the alphabet $\Omega = \Phi \times (\Sigma \cup \{\sqrt{\cdot}\}) = \{(\phi, b) \mid \phi \in \Phi \wedge b \in (\Sigma \cup \{\sqrt{\cdot}\})\}$. The operator $;$ denotes sequential composition, \cup denotes alternative choice, and $*$ denotes Kleene star. Intuitively, program (ϕ, b) means that the current state s should satisfy ϕ , and then the next action taken along the path should be b . If $b \in \Sigma$, an outgoing b -transition has to be taken, and if $b = \sqrt{\cdot}$ (pseudo-action $\sqrt{\cdot} \notin \Sigma$), no transition is taken. The full formal semantics of asCSL is given in [3].

Untimed asCSL is an extension of untimed CSL, so every CSL formula can be expressed in asCSL. The syntax and semantics of untimed *Until* formulas are as explained in [4]. For our purpose we consider only the *Until* operator, because the parameter synthesis for the *Next* operator follows trivially from the algorithm for the *Until* operator. For the sake of completeness we provide the semantics of CSL until-type path formulas.

► **Definition 2.4** (Untimed Until). The satisfaction relation \models for untimed *Until* path formulas is defined as:

$$\sigma \models \Phi_1 \mathcal{U} \Phi_2 \quad \text{iff } \exists k \geq 0 : \sigma[k] \models \Phi_2 \wedge \forall (0 \leq i < k) : \sigma[i] \models \Phi_1$$

where Φ_1, Φ_2 are state formulas, and $\sigma[k]$ denotes the k -th state on path σ .

From here on, untimed *Until* is simply called *Until*. Any CSL untimed *Until* property can be expressed in asCSL as $\Phi_1 \mathcal{U} \Phi_2 \stackrel{\text{def}}{=} (\Phi_1, \Sigma)^* ; (\Phi_2, \sqrt{\cdot})$, which follows from Prop. 12 in [3].

Let $Sat(\Phi)$ denote the set of states fulfilling state formula Φ . During the synthesis procedure, our attention will be on the set of so-called *functionally valid* states (or just *valid* for short), which are those states that possess at least one outgoing path which matches a given until-type requirement. This motivates the following definition:

► **Definition 2.5** (Valid states). Let $s \in S$ be a state in an ASMC \mathcal{M} and $\Phi = P_{\sim b}(\varphi)$ be a state formula, with $\varphi = \Phi_1 \mathcal{U} \Phi_2$ being an until-type path formula. Then a new relation \models_v characterizing *valid* states in \mathcal{M} , is defined as:

$$s \models_v \varphi \iff \exists \sigma \in Paths(s) : \sigma \models \varphi$$

We further define the set of valid states as $S^\varphi = \{s \in S \mid s \models_v \varphi\} \subseteq S$ and write

$$\mathcal{M} \models_v \Phi \iff \forall s \in S^\varphi \setminus \text{Sat}(\Phi_2) : s \models \Phi$$

Intuitively, *valid* states are those states in \mathcal{M} with an emanating path satisfying the path formula φ . Note that whether a state is valid or not is independent of the probability bound b . Even though the *target* states ($\text{Sat}(\Phi_2)$) belong to the *valid* states S^φ , we do not consider them for synthesis, because the probability of *target* states to satisfy φ is always 1 and cannot be modified. So, the given user requirement Φ is said to be satisfied by ASMC \mathcal{M} , iff all *valid* states, except the *target* states in \mathcal{M} , satisfy Φ .

After controller synthesis, in order to satisfy the user requirement, parallel composition of the plant and the controller is necessary (see Sec. 4). Therefore we now provide a definition for parallel composition of ASMCs. Note that different stochastic process algebras possess different semantics [10] for parallel composition. For our purpose, in case of synchronization the resulting rate of two actions with rates λ and μ shall be determined by their product $\lambda \cdot \mu$ (where, in practice, one of the two factors is either equal to one or a slowdown factor $0 < k \leq 1$).

► **Definition 2.6** (Parallel composition in ASMCs). The parallel composition of two ASMCs P and Q is defined by the following rules (analogous to [2, 12]):

$$\frac{P \xrightarrow{a, \lambda} P', Q \xrightarrow{a, \mu} Q'}{P \parallel_{\Sigma_{syn}} Q \xrightarrow{a, \lambda \cdot \mu} P' \parallel_{\Sigma_{syn}} Q'} \quad (a \in \Sigma_{syn})$$

and

$$\frac{P \xrightarrow{a, \lambda} P'}{P \parallel_{\Sigma_{syn}} Q \xrightarrow{a, \lambda} P' \parallel_{\Sigma_{syn}} Q} \quad (a \notin \Sigma_{syn})$$

and a third rule, symmetric to the second one, where Q makes a move while P remains stable. In these rules, $a \in \Sigma_{syn} \subseteq \Sigma$ is a synchronizing action, and λ, μ are the transition rates. The labeling of a state (p_i, q_j) in the product ASMC is defined to be the union of the labellings of p_i and q_j .

When we employ parallel composition in Sec. 4, one process will be the plant \mathcal{P} , the other process will be the controller \mathcal{C} , the set of synchronizing actions Σ_{syn} will be equal to the action set $\Sigma_{\mathcal{C}}$ of the controller, and all rates of the controller will be either equal to one or equal to a common reduction factor k , with $0 < k \leq 1$.

3 Parameter Synthesis for “Until”-type requirements

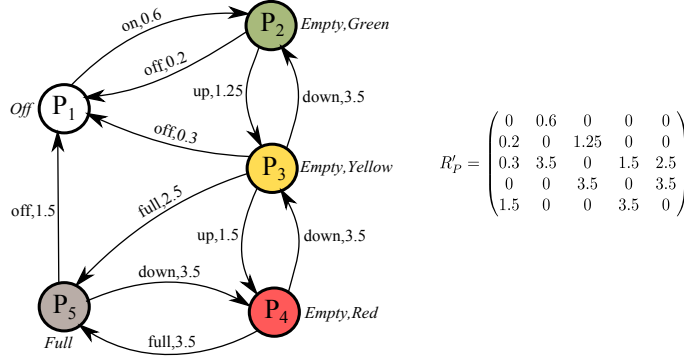
For until-type requirements, the ASMC parameter synthesis problem is intuitively explained as computing a reduction factor k for a subset of the transition rates in the original plant, so as to modify some reachability probabilities as needed. We will start with a simple example.

3.1 Example

This example considers a gas tank with an automatic filling pump, which can be turned off or on based on the levels of the tank.

3.1.1 Unrestricted plant \mathcal{P}

Fig. 1 shows the gas tank \mathcal{P} modeled as an ASMC along with the respective transition rates. Nodes represent different states of the tank and edges represent transitions between states. Initially, the gas tank can be in any state. *Empty* and *Full* represent different levels of the tank, and level *Empty* is further divided into *Green*, *Yellow*, *Red* for easy level reading. Note that the action labeling of transitions is irrelevant for this section.



■ **Figure 1** Tank \mathcal{P} shown as an ASMC with respective accumulated rate matrix

The user requirement on \mathcal{P} is given as an untimed *Until* formula Φ ,

$$\Phi = P_{\leq 0.7}(\varphi), \text{ where } \varphi = \text{Empty} \mathcal{U} \text{Full} \quad (1)$$

which checks whether the probability to reach a *Full* state from an *Empty* state, possibly via intermediate *Empty* states is at most 0.7. By using the PRISM tool [11], we computed the probabilities of the states in ASMC \mathcal{P} to be

$$Pr(P_1, \varphi) = p_{15} = 0 < 0.7 \quad (2)$$

$$Pr(P_2, \varphi) = p_{25} = 0.69473 < 0.7 \quad (3)$$

$$Pr(P_3, \varphi) = p_{35} = 0.80589 > 0.7 \quad (4)$$

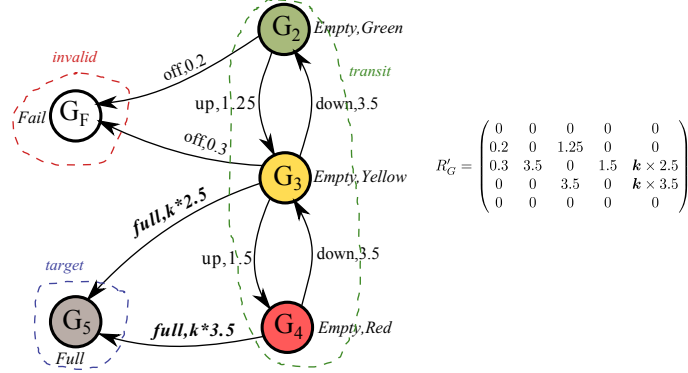
$$Pr(P_4, \varphi) = p_{45} = 0.90294 > 0.7 \quad (5)$$

$$Pr(P_5, \varphi) = p_{55} = 1 \quad (6)$$

where $p_{i5} = \text{Prob}(\text{to reach state } P_5 \text{ from state } P_i \text{ via a satisfying path})$. According to Definition 2.5, $Sat(\Phi) = \{P_1, P_2\}$, $Sat(Full) = \{P_5\}$ and $S_P^\varphi = \{P_2, P_3, P_4, P_5\}$. So, to make $\mathcal{P} \models_v \Phi$ the set $S_P^\varphi \setminus Sat(Full) = \{P_2, P_3, P_4\}$ should satisfy Φ . From the above equations state P_2 already satisfies Φ , whereas P_3 and P_4 do not. Hence, parameter synthesis is required on \mathcal{P} . This is done by reducing some of the transition rates in \mathcal{P} by a factor of k . To determine k , we create a reduced automaton \mathcal{G} .

3.1.2 Obtain the reduced automaton \mathcal{G} from \mathcal{P}

Fig. 2 shows the ASMC \mathcal{G} , which has all the valid states from \mathcal{P} . The remaining states from \mathcal{P} (in this case only the state P_1) are aggregated to a single absorbing fail state G_F , because visiting those states violates the *Until* formula (1). The target states are also made absorbing, i.e., state P_5 . To apply the heuristics, the states in \mathcal{G} are now partitioned into *invalid*, *transit* and *target* classes. According to the heuristics explained in Sec. 3.2.3, in R'_G the rates of the transitions leading from the *transit* class towards the *target* class should be



■ **Figure 2** Reduced automaton \mathcal{G} of \mathcal{P} , and its rate matrix R'_G

reduced by the factor k . The satisfying range of k between 0 and 1 should be obtained, such that the probabilities p_{35} and p_{45} fall below 0.7 as required by equation (1).

3.1.3 System of equations from \mathcal{G}

For each valid state G_i (except the target state G_5), we now construct a new equation (depending on k) for the probability p_{i5} . In our example, equations for p_{25} , p_{35} and p_{45} are obtained from the reduced automaton \mathcal{G} in Fig. 2 as follows:

$$p_{25} = \frac{1.25}{1.45} \times p_{35} \quad (7)$$

$$p_{35} = \frac{3.5}{k \times 2.5 + 5.3} \times p_{25} + \frac{1.5}{k \times 2.5 + 5.3} \times p_{45} + \frac{k \times 2.5}{k \times 2.5 + 5.3} \quad (8)$$

$$p_{45} = \frac{3.5}{k \times 3.5 + 3.5} \times p_{35} + \frac{k \times 3.5}{k \times 3.5 + 3.5} \quad (9)$$

According to equation (1), the following constraints need to be met:

$$0 < k \leq 1 \quad 0 \leq p_{25} \leq 0.7 \quad 0 \leq p_{35} \leq 0.7 \quad 0 \leq p_{45} \leq 0.7$$

Upon solving the system of equations in (7),(8),(9) along with the constraints, we obtain the satisfying range of k to be $0 < k \leq 0.326244$. Thus, the solution of the parameter synthesis problem consists of changing \mathcal{P} by multiplying the transition rates from P_3 to P_5 and from P_4 to P_5 by such a factor of k . We now proceed to the general algorithm and heuristics required to solve the parameter synthesis problem.

3.2 General Algorithm

We assume that the original plant is defined as an ASMC, $\mathcal{P}=(S_P, \Sigma_P, R'_P, L_P)$, and the user requirement is specified by $\Phi = P_{\sim b}(\Phi_1 \mathcal{U} \Phi_2)$. Algorithm 1 shows the procedure of how to determine the set of transition rates to be reduced and how to synthesize the reduction factor k . For the algorithm to deliver the correct result, the user-given *Until* formula should not contain nested probabilistic formulas. Note that this algorithm provides a simple way to control a plant according to the user requirement, but other, more distinguished, approaches would be also possible (see Sec. 5).

Algorithm 1 Parameter synthesis algorithm for untimed *Until* formulas

Input: Plant \mathcal{P} expressed as ASMC and requirement $\Phi = P_{\sim b}(\Phi_1 \mathcal{U} \Phi_2)$
Output: Set of transitions to be modified and reduction factor k and
if $S_P^\varphi = \emptyset$ **then** quit \triangleright No valid states in plant \mathcal{P} , and req. cannot be fulfilled
else
 if $\mathcal{P} \models_v \Phi$ **then** quit \triangleright No need of par-synthesis, since req. is already fulfilled
 else
 Construct $\mathcal{G} = (S_G, \Sigma_G, R'_G, L_G)$ \triangleright Refer Sec. 3.2.1
 Find set of trans. rates T to be reduced and reduction factor k \triangleright Refer Sec. 3.2.2
 Change \mathcal{P} to \mathcal{P}_{mod} by reducing the rates of T by factor k
 end if
end if

3.2.1 Generating the reduced automaton \mathcal{G}

The first step towards solving the parameter synthesis problem is to create a reduced automaton \mathcal{G} from \mathcal{P} with the help of Φ .

► **Definition 3.1** (Reduced automaton \mathcal{G}). The reduced automaton \mathcal{G} is defined as a tuple $(S_G, \Sigma_G, R'_G, L_G)$ where:

- $S_G \subseteq S_P$, such that $S_G = S_P^\varphi \cup \{G_F\}$, i.e. the *valid* states from \mathcal{P} , along with a new fail state which is absorbing and subsumes all states from $S_P \setminus S_P^\varphi$
- $\Sigma_G \subseteq \Sigma_P$ only those actions among the states in S_G
- add each transition $s_i \xrightarrow{a_{ij}, \lambda} s_j$ from \mathcal{P} , where $s_i, s_j \in S_P^\varphi$
- for each transition $s_i \xrightarrow{a_{ij}, \lambda} s_j$ in \mathcal{P} , where $s_i \in S_P^\varphi$ and $s_j \notin S_P^\varphi$ add transition $s_i \xrightarrow{a_{ij}, \lambda} G_F$
- target states must be made absorbing
- $L_G : S_G \mapsto 2^{AP} \cup \{\{Fail\}\}$ is a labeling function, where $\forall s \in S_G \setminus \{G_F\} : L_G(s) = L_P(s)$ and $L_G(G_F) = \{Fail\}$

As explained in Algorithm 1, \mathcal{G} is created only if $\mathcal{P} \not\models_v \Phi$ and $S_P^\varphi \neq \emptyset$. Once \mathcal{G} has been constructed according to Def. 3.1, some rates of R'_G need to be chosen for synthesis based upon some heuristics (sec 3.2.3) on the given property Φ .

3.2.2 Obtain k from \mathcal{G}

The states in \mathcal{G} are partitioned into *target*, *transit* and *invalid* classes, which are used while applying the heuristics. The *target* class consists of states which satisfy Φ_2 , the *transit* class contains all *valid* states (Def. 2.5) except the target states, and the *Fail* state G_F is placed into the *invalid* class. Parameter synthesis is based upon some rules as follows:

1. Transition rates in R'_G cannot exceed the original rates in R'_P and cannot be made zero.
2. Some transition rates in R'_G will be reduced by a common factor k (where $0 < k \leq 1$), to make sure that the probability during model checking will satisfy the bound given in Φ . The set of transition rates to be multiplied with k is determined according to the heuristics explained in Sec. 3.2.3.
3. Create a system of (rational polynomial) equations for model checking. There is one equation for each state from *transit* class, representing its probability to reach the *target* class via a satisfying path.
4. Impose the constraints on probabilities according to the given Φ .

5. Solve these equations and constraints for the probabilities, and the resultant k leads to a model that satisfies the user requirement.

3.2.3 Heuristics on \mathcal{G}

In R'_G , some of the rates need to be reduced by a common factor k , such that \mathcal{G} satisfies the requirement Φ . The following heuristics are applied to modify the rates. If in the given formula Φ

1. the probability bound is a lower bound, i.e. $P_{\geq b}(\varphi)$ or $P_{> b}(\varphi)$, then the rates of the transitions leading from class *transit* to class *invalid* should be multiplied by the reduction factor k ,
2. or, if the probability bound is an upper bound, i.e. $P_{\leq b}(\varphi)$ or $P_{< b}(\varphi)$, then the rates of the transitions leading from class *transit* to class *target* should be multiplied by the reduction factor k .

These heuristics determine the set of transition rates to be reduced, and the common reduction factor k ($0 < k \leq 1$) can then be found by solving the set of equations with the imposed constraints.

► **Theorem 3.2.** *Assume that in \mathcal{G} the invalid class is reachable from every state of the transit class. Then the set of constrained equations constructed according to the heuristics in Sec. 3.2.3 will always have a solution.*

Proof. For the reduction factor k it holds that $0 < k \leq 1$. This means that one can make the chosen set of transition rates arbitrarily close to zero (without completely disabling the transition). The heuristics distinguishes three classes *transit*, *target* and *invalid*, of which the latter two classes are absorbing. The *target* class is reachable from the *transit* class by definition, and the *invalid* class is reachable from the *transit* class by assumption. Hence, the uniform reduction of rates between two classes will make the transitions between the other two classes more likely and this will always lead to a solution. ◀

We end this section by characterizing the relation between the ASMC \mathcal{P} and its reduced automaton \mathcal{G} . If \mathcal{G} satisfies Φ , that implies that the modified version of \mathcal{P} will also satisfy Φ , because of the fact that \mathcal{G} focuses only on *valid* states of \mathcal{P} . This is stated by the following Theorem.

► **Theorem 3.3.** *For any ASMC \mathcal{P} and its reduced automaton \mathcal{G} , and for any untimed until-type formula Φ without nested probabilistic path operators, it holds that*

$$\mathcal{G} \models_v \Phi \implies \mathcal{P}_{mod} \models_v \Phi$$

where \mathcal{P}_{mod} is the modified version of \mathcal{P} by applying the reduction factor k to the selected transitions.

Proof. The proof follows directly from the construction of \mathcal{G} and from the semantics of until-type formulas. ◀

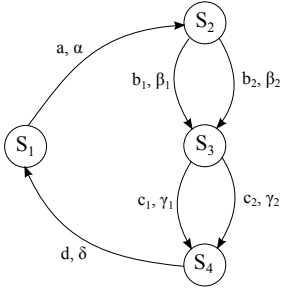
4 Controller synthesis for untimed asCSL requirements

Having seen the procedure to synthesize parameters for until-type requirements, we now propose an algorithm to synthesize a controller for general untimed asCSL formulas. Given a plant \mathcal{P} and a user requirement in the form of an asCSL formula, we propose a novel

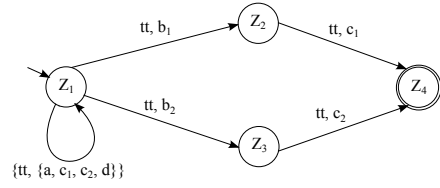
approach to synthesize a controller \mathcal{C} , which is another ASMC to be composed in parallel with the plant. Controller synthesis will also involve the synthesis of parameters as a subtask. To better understand the synthesis procedure, we again start this section with an example, followed by the general algorithm.

4.1 Example

Fig. 3 shows a plant modeled as an ASMC \mathcal{P} . The states can have atomic propositions, but they are not relevant for this example. The transitions shown among the states are labeled with the actions followed by their respective rates. The plant consists of 4 states and has the ability to start in any state.



■ **Figure 3** ASMC of plant \mathcal{P}



■ **Figure 4** NPA \mathcal{Z}_α for the given asCSL program α

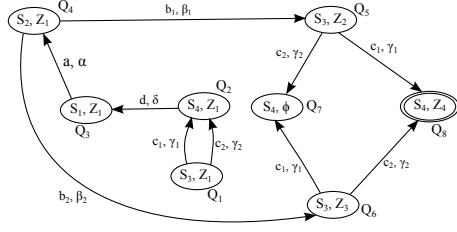
Assume that we wish to ensure that in \mathcal{P} , once action b_1 has taken place, it will be followed by action c_1 with high probability, and the same for actions b_2 and c_2 . For this criterion, the asCSL user requirement is given as follows,

$$\Phi = P_{>0.5}(\alpha) \quad (10)$$

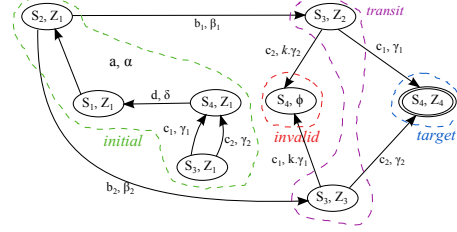
where, α is the following asCSL program:

$$\alpha = ((tt, a) \cup (tt, c_1) \cup (tt, c_2) \cup (tt, d))^*; \left(((tt, b_1); (tt, c_1)) \cup ((tt, b_2); (tt, c_2)) \right) \quad (11)$$

Formula Φ states that the probability to take action b_i followed by c_i , where $i \in \{1, 2\}$, should be greater than 0.5, and tt stands for *true*. The first part of α (covered by the Kleene star), states that the initial behavior before action b_1 or b_2 occur can be arbitrary. For model checking ASMCs against an asCSL requirement, a NPA \mathcal{Z}_α (non-deterministic program automaton) is constructed from the program α . The automaton \mathcal{Z}_α for the given α is shown in Fig. 4, and it happens to be deterministic for this example. According to the asCSL model checking algorithm [3], we now construct the product automaton \mathcal{Q} for \mathcal{P} and \mathcal{Z}_α . Fig. 5 shows this product automaton \mathcal{Q} , with the accepting state Q_8 and the absorbing fail state Q_7 . For model checking we considered the rates in \mathcal{P} to be $\alpha = 2$, $\beta_1 = 4$, $\beta_2 = 6$, $\gamma_1 = 6$, $\gamma_2 = 4$ and $\delta = 3$. Model checking the product automaton (with PRISM [11]) gave that the probability of \mathcal{P} satisfying the given Φ is 0.48 (the same for all states in \mathcal{P}), which is a violation of our requirement in equation (10). Hence, we need to synthesize a controller. A controller has the ability to modify both the transition rates and also the structural behavior of an ASMC, which is achieved by using parameter synthesis and controller synthesis respectively. For parameter synthesis, we need to construct a blueprint automaton \mathcal{B} (similar to the purpose of \mathcal{G} in Sec. 3) from the product automaton \mathcal{Q} , which is useful for applying heuristics and also to synthesize the controller. Hence, the name blueprint automaton.



■ **Figure 5** Product automaton \mathcal{Q}



■ **Figure 6** Blueprint automaton \mathcal{B} from \mathcal{Q}

4.1.1 Blueprint automaton \mathcal{B} for this example

In the product automaton \mathcal{Q} , we have to identify *target*, *invalid*, *initial* and *transit* classes. These are intuitively defined as

1. *target* class contains all states where Z_α is accepting, i.e. state Q_8 ,
2. *initial* class contains only those states which start from the initial states of Z_α , i.e. states Q_1, Q_2, Q_3 and Q_4 ,
3. *invalid* class contains those states which have no path to a target state, i.e. state Q_7 ,
4. *transit* class contains the states which are on the path from *initial* to *target* class, but do not start from initial states of Z_α , i.e. states Q_5 and Q_6 .

Note that these four classes are not necessarily disjoint (e.g. some initial states can be invalid). The reason for this classification is to identify those states which should satisfy the probability bound given in Φ (10). Fig. 6 shows the classes of \mathcal{B} obtained from \mathcal{Q} . From the automaton \mathcal{B} , some of the rates are then chosen for parameter synthesis using heuristics (Sec. 4.2.2). In order to increase the probability from 0.48 to above 0.5 as in the given Φ , the rates leaving from *transit* to *invalid* class are considered for parameter synthesis. The aim now is to find a suitable value k ($0 < k \leq 1$) as the common reduction factor for these rates. According to the model checking algorithm for asCSL [3], in the product automaton we only check for $\varphi = (tt \mathcal{U} target)$ formula. Also, with respect to Def. 2.5, the valid states are $S_Q^\varphi = \{Q_1, Q_2, Q_3, Q_4, Q_5, Q_6, Q_8\}$ and $Sat(target) = \{Q_8\}$. As we will see later, to make \mathcal{P} satisfy Φ , it is sufficient if only the states in the *initial* class of \mathcal{B} satisfy Φ , but the states Q_1, Q_2, Q_3 will not affect the probability as they have single outgoing transition. Hence, only state Q_4 needs to be considered. The probability is computed using the equations below.

$$Pr(Q_4, \varphi) = p_{48} = \frac{\beta_1}{\beta_1 + \beta_2} \times \frac{\gamma_1}{\gamma_1 + k \times \gamma_2} + \frac{\beta_2}{\beta_1 + \beta_2} \times \frac{\gamma_2}{k \times \gamma_1 + \gamma_2} \quad (12)$$

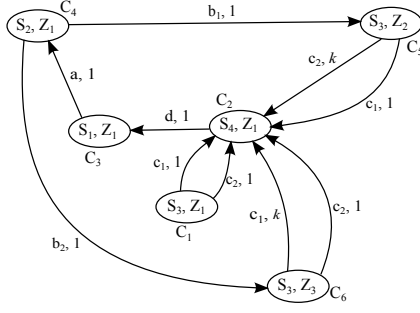
$$\text{with constraints:} \quad 0 < k \leq 1 \quad 0.5 < p_{48} \leq 1 \quad (13)$$

Solving this constrained quadratic inequality for p_{48} yields k to be ($0 < k < 0.92$). Hence any k value from this range satisfies the user given Φ .

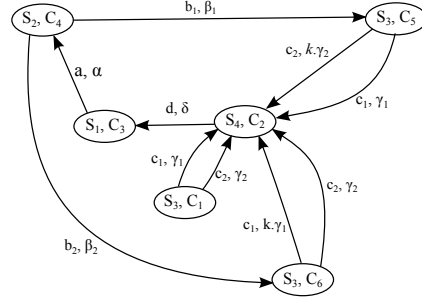
4.1.2 Creating controller \mathcal{C} for \mathcal{P}

The controller will restrict the behavior of the plant according to the user requirements. We create a controller \mathcal{C} , such that $(\mathcal{P} \parallel_{\Sigma_C} \mathcal{C}) \models_v \Phi$. We use \mathcal{B} as a blueprint for the controller, making all the absorbing states of \mathcal{B} non-absorbing as follows:

- The absorbing states (S_4, \emptyset) and (S_4, Z_4) are removed and all the transitions leading to them are diverted to the state (S_4, Z_1) , because when the plant \mathcal{P} has reached state S_4 and whether or not the previous trajectory has satisfied Φ , the plant need to start from the initial state of Z_α .



■ **Figure 7** Controller \mathcal{C} for the plant \mathcal{P}



■ **Figure 8** Result of Parallel composition of $\mathcal{P} \parallel_{\Sigma_C} \mathcal{C}$

- The rates which we obtained by parameter synthesis are replaced by the reduction factor k and the rest of the rates are intentionally set to 1.

Fig. 7 shows the controller \mathcal{C} created for the plant \mathcal{P} . The transition rates obtained via parameter synthesis from the states C_5 and C_6 to C_2 are multiplied by the reduction factor k , and the rest of the rates are unchanged, hence the multiplication factor is 1. We already obtained the value of k . When the parallel composition is performed between the original plant \mathcal{P} and the controller \mathcal{C} , synchronizing over all the actions in Σ_C as per the standard composition rules (Def. 2.6), the resultant product automaton satisfies the user requirement Φ . The result of parallel composition of plant \mathcal{P} and the controller \mathcal{C} is shown in Fig. 8.

4.2 General Algorithm for parameter synthesis for untimed asCSL

In Algorithm 1, we explained the general procedure to synthesize the parameters for *until* type requirements. Now we give an algorithm to synthesize a controller for general untimed asCSL requirements (which also includes parameter synthesis). A prerequisite for the algorithm to work correctly is that the asCSL program α should not contain any nested probabilistic formulas.

Algorithm 2 Controller synthesis algorithm for general untimed asCSL formulas

Input: Plant \mathcal{P} modeled as ASMC and an untimed asCSL specification $\Phi = P_{\sim p}(\alpha)$
Output: Controller \mathcal{C} such that $(\mathcal{P} \parallel_{\Sigma_C} \mathcal{C}) \models_v \Phi$
Construct product automaton \mathcal{Q} from \mathcal{P} and \mathcal{Z}_α as in [3] and classify its states
Let $\varphi = (tt \ \mathcal{U} \ target)$
if $S_{\mathcal{Q}}^\varphi = \emptyset$ **then** quit \triangleright No valid states in plant \mathcal{Q}
else
 if $\forall s \in S_{\mathcal{Q}}^\varphi \cap initial : s \models \Phi$ **then** quit \triangleright No need of contr. synthesis
 else
 Construct $\mathcal{B} = (S_{\mathcal{B}}, \Sigma_{\mathcal{B}}, R_{\mathcal{B}}, L_{\mathcal{B}})$ from \mathcal{Q} \triangleright Refer Sec. 4.2.1
 From \mathcal{B} , find set of trans. to be modified and the red. factor k \triangleright Refer Sec. 4.2.1
 Modify the absorbing states in \mathcal{B} and multiply the selected rates with k
 The result is controller \mathcal{C}
 end if
end if

4.2.1 Blueprint automaton \mathcal{B}

The blueprint automaton \mathcal{B} is based on the product automaton \mathcal{Q} of the plant \mathcal{P} and \mathcal{Z}_α .

► **Definition 4.1** (Blueprint automaton \mathcal{B}). The blueprint automaton \mathcal{B} is obtained from the product automaton \mathcal{Q} , along with the following modifications:

- In \mathcal{Q} , accepting states are put into class *target*, states starting from initial states of \mathcal{Z}_α are classified as *initial* and the remaining valid states which are not in *initial* and *target* are put into *transit* class. The states which are not valid are put into class *invalid*.
- Some transition rates of \mathcal{Q} are multiplied by the reduction factor k as per the heuristics in Sec. 4.2.2.

From the resultant automaton \mathcal{B} , a set of equations is created, taking into account Th. 27 in [3] which states that for an ASMC \mathcal{P} and an asCSL-program α it holds that

$$Prob^{\mathcal{P}}(s, \alpha) = Prob^{\mathcal{P} \times \mathcal{Z}_\alpha}(\langle s, Z_0 \rangle, tt \ \mathcal{U} \ target)$$

where s is a state in \mathcal{P} and Z_0 is the set of initial states of NPA \mathcal{Z}_α . From this theorem it follows that for an ASMC \mathcal{P} to satisfy the probability bound in the asCSL formula Φ , it suffices to make all the states of the form $\langle s, Z_0 \rangle$ in the product automaton \mathcal{Q} satisfy the probability bound. Therefore, we construct one equation for each *valid* state from class *initial*, representing its probability to reach the *target* class via a satisfying path. Solving these equations yields the value of k .

4.2.2 Heuristics on \mathcal{B}

Even though the heuristics on \mathcal{B} are similar in concept to those on \mathcal{G} (Sec. 3.2.3), refined conditions need to be taken into account due to the extra *initial* class in \mathcal{B} .

If in the given property Φ ,

1. the probability bound is a lower bound, i.e. $P_{\geq b}(\alpha)$ or $P_{> b}(\alpha)$, then the rates of the paths leading from *initial* to *invalid* classes should be reduced. If this does not yield a solution then transitions from *transit* to *invalid* should be reduced. If again no solution is found, then the rates from both the *initial* and *transit* classes to *invalid* should be reduced,
2. or, if the probability bound is an upper bound, i.e. $P_{\leq b}(\alpha)$ or $P_{< b}(\alpha)$, then the rates of the transitions should be reduced as follows until a solution is found:
 - a. all transitions from *initial* to *target*,
 - b. all transitions from *transit* to *target*,
 - c. combine steps **2a** and **2b**

Under the assumption that the *invalid* class is reachable from every valid state of the *initial* class, Th. 3.2 also applies to this situation. Thus, under this assumption these heuristics will always yield a solution.

4.2.3 Controller \mathcal{C}

The controller is derived from the blueprint automaton \mathcal{B} , but to make the controller non-blocking, all absorbing states in \mathcal{B} should be made non-absorbing. The controller \mathcal{C} is obtained by modifying \mathcal{B} as follows:

1. Replace all the states from the *invalid* class of the form (S_i, \emptyset) with states of the kind (S_i, Z_0) , where Z_0 is the set of starting states of the automaton \mathcal{Z}_α .
2. All the states from the *target* class like (S_i, Z_j) (Z_j is a set containing an accepting state of \mathcal{Z}_α) are replaced with (S_i, Z_0) .

5 Conclusion

For a given ASMC model (the plant) and an asCSL path-based requirement, we have studied the problem of controlling the plant in such a way that its states will satisfy the requirement, wherever possible at all. Satisfaction can be achieved by either a reduction of a subset of the plant's transition rates, or by parallel composition with a controller. We have presented two algorithms: Algorithm 1 performs parameter synthesis for untimed until-type requirements, and Algorithm 2 extends this concept to controller synthesis for general untimed asCSL formulas.

In this paper, we have restricted our attention to requirements without nested probabilistic path operators. Such nesting requires special care, since changing some parameters in order to satisfy an inner probabilistic path formula can have either a positive or an adverse effect on the satisfiability of the enclosing formula. As a simple example, the strategy to maximize the satisfaction set of $P_{\geq b_1}(\Phi_1 \mathcal{U} P_{\geq b_2}(\Phi_2 \mathcal{U} \Phi_3))$ will not be the same as for $P_{\leq b_1}(\Phi_1 \mathcal{U} P_{\geq b_2}(\Phi_2 \mathcal{U} \Phi_3))$, since in the latter case the set of states satisfying the overall formula will be larger if fewer states satisfy the inner formula. We intend to elaborate on this in a forthcoming paper.

The paper has presented feasible solutions, but did not address the question of optimality. Solutions which are “better” in some sense could be obtained by applying more complex heuristics than the ones described in this paper, for instance by allowing non-uniform reduction factors or by also reducing some rates within the class *transit*. However, how to characterize the optimal solution and how to obtain it is future work. Another important issue for future work is the control problem for *time-bounded* problems which will involve the computation of transient state probabilities with the method of uniformization.

Acknowledgments We would like to thank the anonymous reviewers for their valuable constructive feedback and also our colleague Alexander Gouberman for his suggestions.

References

- 1 A. Aziz, K. Sanwal, V. Singhal, and R. Brayton. Model Checking Continuous Time Markov chains. In *Computer-Aided Verification*, 1996.
- 2 J. Bachmann, M. Riedl, J. Schuster, and M. Siegle. An efficient symbolic elimination algorithm for the stochastic process algebra tool CASPA. In *35th Int. Conf. on Current Trends in Theory and Practice of Computer Science (SOFSEM'09)*, pages 485–496. Springer LNCS 5404, 2009.
- 3 C. Baier, L. Cloth, B. Haverkort, M. Kuntz, and M. Siegle. Model checking Markov chains with Actions and State labels. *IEEE Trans. on Software Engineering*, 33(4):209–224, 2007.
- 4 C. Baier, B. Haverkort, H. Hermanns, and J.-P. Katoen. Model-Checking Algorithms for Continuous-Time Markov Chains. In *IEEE Transactions on Software Engineering*, volume 29, July 2003.
- 5 L. Brim, M. Češka, S. Dražan, and D. Šafránek. Exploring parameter space of stochastic biochemical systems using quantitative model checking. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification*, volume 8044 of *LNCS*, pages 107–123. Springer Berlin Heidelberg, 2013.
- 6 C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *Theor. Aspects of Computing - ICTAC 2004*, volume 3407 of *LNCS*, pages 280–294. Springer Berlin Heidelberg, 2005.

- 7 E.M. Hahn, H. Hermanns, B. Wachter, and L. Zhang. Param: A model checker for parametric Markov models. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 660–664. Springer Berlin Heidelberg, 2010.
- 8 E.M. Hahn, H. Hermanns, and L. Zhang. Probabilistic reachability for parametric Markov models. *Int. Journal on Software Tools for Technology Transfer*, 13(1):3–19, 2011.
- 9 T. Han, J.-P. Katoen, and A. Mereacre. Approximate parameter synthesis for probabilistic time-bounded reachability. In *Real-Time Systems Symposium*, pages 173–182. IEEE, 2008.
- 10 H. Hermanns, U. Herzog, and J.-P. Katoen. Process algebra for performance evaluation. *Theoretical Computer Science*, 274:43–87, 2002.
- 11 A. Hinton, M. Kwiatkowska, G. Norman, and D. Parker. PRISM: A tool for automatic verification of probabilistic systems. In *Proc. 12th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS' 06)*, volume 3920 of *LNCS*, pages 441–444. Springer, 2006.
- 12 M. Kuntz and M. Siegle. Deriving symbolic representations from stochastic process algebras. In *Process Algebra and Probabilistic Methods, Proc. PAPM-PROBMIV'02*, pages 188–206. Springer, LNCS 2399, 2002.
- 13 M. Lawford and W.M. Wonham. Supervisory control of probabilistic discrete event systems. In *Proc. of the 36th Midwestern Symposium on Circuits and Systems, 1993.*, pages 327–331, vol.1, 1993.
- 14 V. Pantelic, M. Lawford, and S. Postma. A framework for supervisory control of probabilistic discrete event systems. In *12th Int. Workshop on Discrete Event Systems (WODES 2014)*, pages 477–484, vol.12, 2014.
- 15 M. Češka, F. Dannenberg, M. Kwiatkowska, and N. Paoletti. Precise parameter synthesis for stochastic biochemical systems. In *Computational Methods in Systems Biology*, volume 8859 of *LNCS*, pages 86–98. Springer International Publishing, 2014.