Approximate Parameter Synthesis for Probabilistic Time-Bounded Reachability [*]

Tingting Han[1,2] Joost-Pieter Katoen[1,2] Alexandru Mereacre[1]

[1] Software Modeling and Verification, RWTH Aachen University, Germany
[2] Formal Methods and Tools, University of Twente, The Netherlands

Abstract

This paper proposes a technique to synthesize parametric rate values in continuous-time Markov chains that ensure the validity of bounded reachability properties. Rate expressions over variables indicate the average speed of state changes and are expressed using the polynomials over reals. The key contribution is an algorithm that approximates the set of parameter values for which the stochastic real-time system guarantees the validity of bounded reachability properties. This algorithm is based on discretizing parameter ranges together with a refinement technique. This paper describes the algorithm, analyzes its time complexity, and shows its applicability by deriving parameter constraints for a real-time storage system with probabilistic error checking facilities.

1 Introduction

Model checking aims at checking a property, typically stated in some temporal logic, against a given concrete model. For real-time systems whose timing is subject to random influences, efficient model-checking algorithms [3] and accompanying tools such as PRISM [13] have been developed, and have been applied to case studies from a broad application area such as, e.g., performance and dependability analysis, and systems biology. A prominent model for stochastic real-time systems is continuous-time Markov chains (CTMCs, for short). In these stochastic transition systems, state delays are exponentially distributed, and successor states are picked randomly where the branching probabilities are determined by the residence time distributions. CTMC model checking has received considerable attention in the last decade and has been adopted by various classical performance analysis tools. The quest for correctness of stochastic real-time systems such as CTMCs mainly focuses on checking time-bounded reachability properties—is the probability to reach a fail state within the deadline at most 10^{-6}?

A disadvantage of the traditional approaches to model checking is that they can only check the validity of properties under the assumption that all parameter values are known. This means that concrete values of e.g., timing parameters, branching probabilities, costs, and so forth, need to be explicitly given. Although this might be appropriate for the a posteriori verification of concrete system realizations, for design models at a higher level of abstraction this is less adequate. In earlier design phases, such explicit information about model parameters is mostly absent, and instead, only the ranges of parameter values, or the relationship between parameters is known (if at all). For models that incorporate aspects of a random nature, the need for concrete parameter values is, in fact, a significant hurdle, as mostly precise information about the random variables is known after extensive experimentation and measurements only. This is, e.g., witnessed by the fact that fitting — roughly speaking, the attempt to find an appropriate and accurate distribution to actual measurements— is an active field of research in model-based performance analysis [22].

In practical system design, one is not interested in checking a concrete instance, but rather, often in deriving parameter constraints that ensure the validity of the property under consideration. Typical examples are failure-repair systems such as multi-processor systems and modern distributed storage systems, in which components (such as memories or processors) may fail and where only lower- and upper bounds on repair times are known. Rather than determining whether for a certain combination of failure and repair rates, a property holds, one would like to synthesize the set of pairs of rates for which the validity of the property is guaranteed.

This paper studies a parametric version of CTMCs, a novel variant of CTMCs in which rate expressions over variables (with bounded range) indicate the average speed

[*]This research is partially funded by the DFG research training group 1295 Algosyn, the NWO project QUPES and the EU FP7 project QUASIMODO.

of state changes. They are expressed using the polynomial ring over reals, allowing rate expressions such as $3\alpha{\cdot}\beta$ and $\alpha^3{-}\alpha{\cdot}\beta$. We show that checking whether time-bounded reachability probabilities meet certain thresholds amounts to solving a polynomial function over the rate parameters. The main contribution of this paper is an algorithm that approximates the synthesis region, i.e., the set of rate parameter values for which the validity of an a priori given time-bounded reachability property is guaranteed. Synthesizing these values is done by a (grid) discretization of the parameter ranges together with a refinement technique. This paper describes the details of this approach for the *two-parameter* setting. The time complexity of our algorithm is quadratic in the number of states in the parametric CTMC, linear in the grid discretization parameter, and polynomial in the expected number of discrete steps taken before reaching the deadline.

The feasibility of the proposed technique is shown by synthesizing the parameter ranges for a storage system that incorporates error checking, i.e., after each operation (read/write), there is a possibility to check whether an error occurred [4]. The first question in this case study is to determine the error check probability, i.e., at which frequency should errors be checked, such that the probability of a fatal system shutdown within a given deadline, is low. Secondly, we determine the sets of values for service time and error checking time (for given error check probabilities) such that a shutdown rarely happens in time.

Related work. Although some work has been done on (symbolic) parameter synthesis for timed systems [11, 23, 1], parameter synthesis of probabilistic models has received scant attention with the notable extensions of [6, 15]. Lanotte *et al.* [15] consider parametric discrete-time Markov chains (DTMCs), and establish minimal (and maximal) parameter values for simple reachability probabilities. For the two-parameter case, they show the decidability if parameter constraints are linear expressions. Our setting is rather different as we consider a continuous-time model, time-bounded properties, and rather than focussing on decidability issues we attempt to come up with an approximation algorithm. Daws [6] also considers DTMCs and uses regular expressions to do PCTL model checking over DTMCs where certain transition probabilities are unknown.

Roadmap of this paper. Section 2 introduces parametric CTMCs. Section 3 defines uniformization of such models, and shows that time-bounded reachability probabilities can be determined by solving a polynomial function over the rate parameters. Section 4 presents the core of the paper and describes the approximate parameter synthesis algorithm along with its time complexity. Section 5 presents the results for the above mentioned case study, and Section 6 concludes.

2 Parametric CTMCs

Parameters and constraints. For a set $\mathcal{X}$ of m variables (or parameters) $x_1, ..., x_m$, expressions in the polynomial ring $\mathbb{R}[\mathcal{X}]$ over the reals $\mathbb{R}$ are formed by the grammar:

$$\alpha ::= c \mid x \mid \alpha + \alpha \mid \alpha{\cdot}\alpha,$$

where $\alpha \in \mathbb{R}[\mathcal{X}]$, $c \in \mathbb{R}$ and $x \in \mathcal{X}$. The operations $+$ and $\cdot$ are addition and multiplication, respectively. A *valuation* is a function $v : \mathcal{X} \rightarrow \mathbb{R}$ assigning a real value to a variable. We assume that all variables have a closed interval with finite bounds as the range of possible values, i.e., $v(x_i) \in [l_i, u_i]$ and $l_i, u_i \in \mathbb{R}$, for $1 \leqslant i \leqslant m$. The set of all valuation functions is $\mathbb{R}^{\mathcal{X}}$. $\alpha[v]$ denotes the valuation of polynomial $\alpha \in \mathbb{R}[\mathcal{X}]$ by instantiating $x_i \in \mathcal{X}$ by $v(x_i)$. Note that we do not restrict to linear expressions as this will not simplify matters in our setting (as explained later).

An *atomic constraint* is of the form $\alpha \bowtie c$, where $\alpha \in \mathbb{R}[\mathcal{X}]$, $\bowtie \in \{<, \leqslant, >, \geqslant\}$ and $c \in \mathbb{R}$. A *constraint* is a conjunction of atomic constraints. A valuation v satisfies the constraint $\alpha \bowtie c$ if $\alpha[v] \bowtie c$. A *region* $\zeta \subseteq \mathbb{R}^{\mathcal{X}}$ is a set of valuations satisfying a constraint.

Definition 1 (CTMCs) *A* continuous-time Markov chain *$\mathcal{C}$ is a triple $(S, \mathbf{R}, s_0)$ with S a finite state space, $\mathbf{R} : S \times S \rightarrow \mathbb{R}_{\geqslant 0}$ the rate matrix and $s_0 \in S$ the initial state.*

State s is *absorbing* if $\mathbf{R}(s, s') = 0$ for all $s' \neq s$. Given n the cardinality of S, $\boldsymbol{E} = [E(s_1), ..., E(s_n)]$ is the vector of exit rates, where $E(s_i) = \sum_{s' \in S} \mathbf{R}(s_i, s')$. The vector $\boldsymbol{\pi}(t) = [\pi_1(t), ..., \pi_n(t)]$ gives the *transient probability* of the CTMC, i.e., the probability of being in state s_i ($1 {\leqslant} i {\leqslant} n$) at time t. The Chapman-Kolmogorov equations describe the evolution of the transient probability distribution over time:

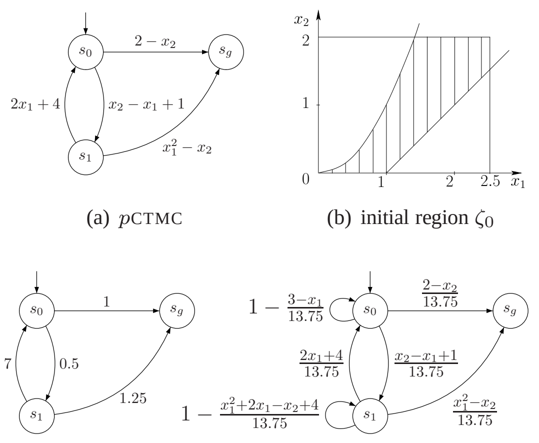
$$\frac{d\boldsymbol{\pi}(t)}{dt} = \boldsymbol{\pi}(t)\mathbf{Q}, \qquad \sum_{i=1}^{n} \pi_i(t_0) = 1, \tag{1}$$

where $\boldsymbol{\pi}(t_0)$ is the initial condition and $\mathbf{Q} = \mathbf{R} - diag(\boldsymbol{E})$ is the *infinitesimal generator* of CTMC $\mathcal{C}$ and $diag(\boldsymbol{E})$ is the diagonal matrix constructed from $\boldsymbol{E}$.

Definition 2 (pCTMCs) *A* parametric CTMC *over the set $\mathcal{X}$ of parameters is a triple $\mathcal{C}^{(\mathcal{X})} = (S, \mathbf{R}^{(\mathcal{X})}, s_0)$, where S and s_0 are as before, and $\mathbf{R}^{(\mathcal{X})} : S \times S \rightarrow \mathbb{R}[\mathcal{X}]$ is the parametric rate matrix.*

The parametric infinitesimal matrix $\mathbf{Q}^{(\mathcal{X})}$ and the exit rate vector $\boldsymbol{E}^{(\mathcal{X})}$ are defined in a similar way as $\mathbf{R}^{(\mathcal{X})}$.

Definition 3 (Instance CTMCs) *For pCTMC $\mathcal{C}^{(\mathcal{X})}$ and valuation v, $\mathcal{C}^{(\mathcal{X})}[v]$ (or simply $\mathcal{C}[v]$ when $\mathcal{X}$ is clear from the context) is the* instance CTMC *of $\mathcal{C}^{(\mathcal{X})}$ obtained by instantiating $x_i \in \mathcal{X}$ by $v(x_i)$.*



(a) pCTMC (b) initial region ζ_0

(c) instance CTMC (d) uniformized pCTMC

Figure 1. pCTMCs and related concepts

For a pCTMC $\mathcal{C}^{(\mathcal{X})} = (S, \mathbf{R}^{(\mathcal{X})}, s_0)$ over the set of parameters $\mathcal{X} = \{x_1, ..., x_m\}$, the *initial region* ζ_0 satisfies the following constraint:

$$\bigwedge_{i=1}^{m} l_i \leqslant x_i \leqslant u_i \wedge \bigwedge_{s,s' \in S} \mathbf{R}^{(\mathcal{X})}(s, s') \geqslant 0, \qquad (2)$$

where $\bigwedge_{i=1}^{m} l_i \leqslant x_i \leqslant u_i$ is the *range constraint* of the parameters and $\bigwedge_{s,s' \in S} \mathbf{R}^{(\mathcal{X})}(s, s') \geqslant 0$ the *rate constraint* ruling out the negative rates. The regions satisfying the range and rate constraints are called *range region* (ζ_{range}) and *rate region* (ζ_{rate}), respectively. Note that $\zeta_0 = \zeta_{range} \cap \zeta_{rate}$.

Example 1 *In Fig. 1(a), we illustrate a* pCTMC *over* $\{x_1, x_2\}$ *with the range constraint:* $0 \leqslant x_1 \leqslant 2.5$ *and* $0 \leqslant x_2 \leqslant 2$*. The rate constraint is as follows:*

$$2x_1 + 4 \geqslant 0 \wedge 2 - x_2 \geqslant 0 \wedge x_2 - x_1 + 1 \geqslant 0 \wedge x_1^2 - x_2 \geqslant 0.$$

ζ_{range} *is the rectangular area in Fig. 1(b), while* ζ_{rate} *is the area between the two curves. The initial region* ζ_0 *is the shaded area. Any point in* ζ_0 *will induce an instance* CTMC*, e.g., for the valuation* $v(x_1) = 1.5$*,* $v(x_2) = 1$*, the instance* CTMC *is shown in Fig. 1(c).*

3 Time-bounded reachability

We are mainly interested in *probabilistic time-bounded reachability* properties, i.e., given a goal state s_g, does the probability of reaching s_g within time t lie in the interval $[p_l, p_u]$? In temporal logics, such as CSL [3], this is formalized as $\mathcal{P}_{[p_l,p_u]}(\diamondsuit^{\leqslant t} s_g)$, where $0 \leqslant p_l \leqslant p_u \leqslant 1$ and $\diamondsuit^{\leqslant t} s_g$ denotes the reachability of s_g within t time units. Note that the interval $[p_l, p_u]$ may also be open, or half-open.

The aim of model checking is to check for a given state (typically the initial state s_0) of the CTMC whether $\mathcal{P}_{[p_l,p_u]}(\diamondsuit^{\leqslant t} s_g)$ holds. The computation of reachability probabilities boils down to computing transient probabilities. Formally,

Lemma 4 ([3]) *Given a* CTMC $\mathcal{C} = (S, \mathbf{R}, s_0)$*, and* $s_g \in S$ *an absorbing state:*

$$Prob(s_0, \diamondsuit^{\leqslant t} s_g) = \pi_{s_g}(t), \qquad (3)$$

where $Prob(s_0, \diamondsuit^{\leqslant t} s_g)$ *is the reachability probability of* s_g *from* s_0 *within* t *time units and* $\pi_{s_g}(t)$ *is the transient probability of state* s_g *at time* t*.*

This lemma also applies to pCTMCs in the sense that every valuation $v \in \zeta_0$ will induce an instance CTMC, for which this lemma applies. Thus in the rest of this section, we focus on deriving an expression for $\pi_{s_g}(t)$ with parameters in $\mathcal{X}$. This is done by uniformization. The expression for $\pi_{s_g}(t)$ (equals the reachability probability) is the basis of solving the synthesis problem in Section 4.

Uniformization in CTMCs. Uniformization (a.k.a. Jensen's method or randomization) [12] is a well-known method for computing the vector $\boldsymbol{\pi}(t)$ of transient probabilities. Uniformization is attractive because of its good numerical stability and the fact that the computational error is well-controlled and can be specified in advance.

For CTMC $\mathcal{C} = (S, \mathbf{R}, s_0)$, let $q \geqslant \max_i\{E(s_i)\}$, and define the matrix $\mathbf{P} = \mathbf{I} + \frac{\mathbf{Q}}{q}$, where $\mathbf{I}$ is the identity matrix of cardinality $|S|$. $\boldsymbol{\pi}(t)$ is then computed as:

$$\boldsymbol{\pi}(t) = \boldsymbol{\pi}(0) \cdot \sum_{i=0}^{\infty} e^{-qt} \frac{(qt)^i}{i!} \mathbf{P}^i. \qquad (4)$$

The infinite summation problem is solved by introducing a required accuracy ε, so that $\|\boldsymbol{\pi}(t) - \tilde{\boldsymbol{\pi}}(t)\| \leqslant \varepsilon$, where $\|\boldsymbol{v}\| = \sum_i |v_i|$ is the norm, $\tilde{\boldsymbol{\pi}}(t) = \boldsymbol{\pi}(0) \cdot \sum_{i=0}^{k_\varepsilon} e^{-qt} \frac{(qt)^i}{i!} \mathbf{P}^i$ is the ε-approximation of $\boldsymbol{\pi}(t)$, and k_ε is the number of terms to be taken in Eq. (4), which is the smallest value satisfying:

$$\sum_{i=0}^{k_\varepsilon} \frac{(qt)^i}{i!} \geqslant \frac{1 - \varepsilon}{e^{-qt}} = (1 - \varepsilon) \cdot e^{qt}. \qquad (5)$$

If qt is large, k_ε tends to be of the order $\mathcal{O}(qt)$.

In the rest of this section, we show how to apply this technique to pCTMCs.

Computing uniformization rate q. Given pCTMC $\mathcal{C}^{(\mathcal{X})} = (S, \mathbf{R}^{(\mathcal{X})}, s_0)$ with the set $\mathcal{X}$ of parameters and initial region ζ_0, the uniformization rate q is at least the largest

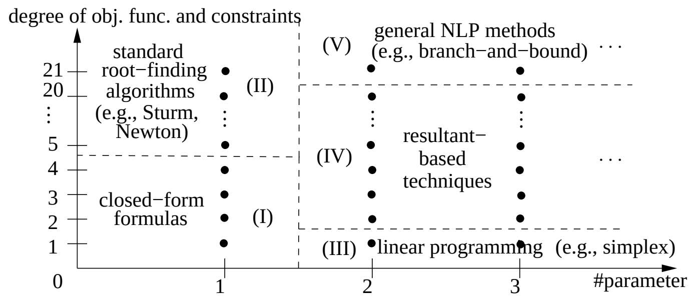

degree of obj. func. and constraints
standard root−finding algorithms (e.g., Sturm, Newton)
closed−form formulas
(I)
(II)
(III)
(IV)
(V)
general NLP methods (e.g., branch−and−bound)
resultant−based techniques
linear programming (e.g., simplex)
#parameter
0
1
2
3
4
5
20
21

Figure 2. Methods for maximizing polynomials under constraints

number that an exit rate can take in ζ_0. Formally,

$$\boxed{q \geqslant \max_{1\leqslant i\leqslant n}\left\{\max_{v\in\zeta_0}\left\{E^{(\mathcal{X})}(s_i)[v]\right\}\right\}} \qquad (6)$$

Note that q is a *constant*. It suffices to first maximize each expression $E^{(\mathcal{X})}(s_i)$ (each is an objective function) within the closed region ζ_0 (the inner $\max$), and then take the maximum out of n candidates (the outer $\max$), where $n = |S|$. Typically, we take the minimal value of q that fulfills Eq. (6). Let us now discuss how a solution to Eq. (6) can be obtained. First we give an example:

Example 2 *For the* pCTMC *in Fig. 1(a) and the initial region* ζ_0 *in Fig. 1(b), the exit rate of* s_0 *and* s_1 *is given by the expressions* $g_0(x_1) := 3 - x_1$ *and* $g_1(x_1, x_2) := x_1^2 + 2x_1 - x_2 + 4$*, respectively. The maximal value of* g_0 *in* ζ_0 *is* 3 *and* 13.75 *for* g_1*. Therefore, we take the uniformization rate* $q = 13.75$*. The uniformized* pCTMC *for this rate is shown in Fig. 1(d).*

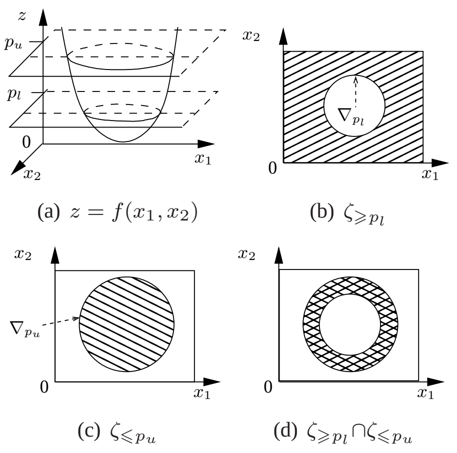
In general, determining the inner max in Eq. (6) boils down to solving a *nonlinear programming* (NLP) problem [2], where the constraints are provided by the initial region. Note that both the objective function $E^{(\mathcal{X})}(s_i)$ and the constraints (Eq. (2)) are polynomial expressions. For some special cases, e.g., one parameter or linear expression rates, the NLP problem can be simplified. Fig. 2 summarizes the techniques that can be applied. The highest degree of the objective function and constraints is indicated along the y-axis, whereas the number of variables x_i of the objective function and constraints is indicated along the x-axis. Each dot represents a combination. We partition the x-y plane into 5 areas (indicated (I) through (V)) by dashed lines, where the dots in the same area can be solved by the techniques specified in the graph. In details:

- For the case of one parameter and degree at most 4 (area (I) in Fig. 2), the maximal value can be obtained by first deriving a closed-form expression and then solving the expression. For polynomials of higher degree, this is *impossible* as proven by Galois [21].
- For the case of one parameter x and the degree at least 5 (area (II) in Fig. 2), standard *root-finding algorithms* can be applied. To be more precise, the roots of the first derivative of the polynomial are to be found as the extreme values, among which together with the boundary values of x we can obtain the maximal values of the polynomial. The prevailing techniques are *Newton's method* [8], *Sturm's method* [10], *Laguerre's method* [14], to name a few.
- For the case of more than one parameter and degree one (area (III) in Fig. 2), it boils down to solving a *linear programming problem* [18], where, amongst others, the *simplex algorithm* [5] and the *interior point method* [16] are well-known solution techniques and quite efficient.
- For the case of more than one parameter and degree from 2 to 20 (area (IV) in Fig. 2), the resultant-based techniques or Gröbner bases methods [7] perform better than branch-and-bound techniques (see below). Note that 20 is an estimation due to performance considerations.
- The remaining cases (area (V) in Fig. 2) are general NLP problems and can be solved numerically by, say, the *branch-and-bound techniques* [9].

Computing $\tilde{\boldsymbol{\pi}}(t)$. Recall that $\tilde{\boldsymbol{\pi}}(t)$ is the ε-approximation of the transient probability vector $\boldsymbol{\pi}(t)$. The reachability probability to s_g within time t equals $\pi_{s_g}(t)$, and thus can be ε-approximated by $\tilde{\pi}_{s_g}(t)$. Similarly, we truncate the infinite sum of $\boldsymbol{\pi}(t)$ to obtain $\tilde{\boldsymbol{\pi}}(t)$. Since the truncation point k_ε in Eq. (5) is *independent of the rates* in the CTMC, it coincides with the non-parametric case. The transient probability vector $\tilde{\boldsymbol{\pi}}(t) = \boldsymbol{\pi}(0)\cdot\sum_{i=0}^{k_\varepsilon} e^{-qt}\frac{(qt)^i}{i!}\left(\mathbf{P}^{(\mathcal{X})}\right)^i$ can be computed by vector-matrix multiplication. For given q and t, $e^{-qt}\frac{(qt)^i}{i!}$ is constant, while $\left(\mathbf{P}^{(\mathcal{X})}\right)^i$ contains parameters. Let $\deg_{x_i}(\mathbf{P}^{(\mathcal{X})})$ denote the maximal degree of parameter x_i in all expressions in $\mathbf{P}^{(\mathcal{X})}$. For instance, $\deg_{x_1}(\mathbf{P}^{(\mathcal{X})}) = 2$ and $\deg_{x_2}(\mathbf{P}^{(\mathcal{X})}) = 1$ for the pCTMC in Fig. 1(a). Note that $\deg_{x_i}(\mathbf{P}^{(\mathcal{X})}) = \deg_{x_i}(\mathbf{R}^{(\mathcal{X})})$. The degree of a polynomial is the sum of the degrees of all its variables. $\left(\mathbf{P}^{(\mathcal{X})}\right)^{k_\varepsilon}$ has degree at most $\hat{k_\varepsilon}$, where $\hat{k_\varepsilon} = \sum_{i=1}^{m} k_\varepsilon\cdot\deg_{x_i}(\mathbf{P}^{(\mathcal{X})})$. Given q and t, the transient probability $\tilde{\pi}_{s_g}(t)$ is a polynomial function over the parameters $x_1, ..., x_m$, i.e.,

$$\boxed{f(x_1, ..., x_m) = \sum_{j=(i_1,...,i_m)} a_j\cdot x_1^{i_1}\cdot...\cdot x_m^{i_m},} \qquad (7)$$

where $i_\ell \leqslant k_\varepsilon\cdot\deg_{x_\ell}(\mathbf{P}^{(\mathcal{X})})$ ($1 \leqslant \ell \leqslant m$), and $a_j \in \mathbb{R}$. The degree of f is at most $\hat{k_\varepsilon}$, which is of order $\mathcal{O}(mqtr)$, where $r := \max_{x_i\in\mathcal{X}}\deg_{x_i}(\mathbf{P}^{(\mathcal{X})})$ is the maximal degree of the polynomial expressions appearing in the pCTMC. Note that k_ε is usually much larger than $\deg_{x_i}(\mathbf{R}^{(\mathcal{X})})$ and



(a) $z = f(x_1, x_2)$ (b) $\zeta_{\geqslant p_l}$

(c) $\zeta_{\leqslant p_u}$ (d) $\zeta_{\geqslant p_l} \cap \zeta_{\leqslant p_u}$

Figure 3. An example synthesis region

thus the degree of the polynomial expression in Eq. (7) is not much affected if we would restrict rate expressions in pCTMCs to be linear.

4 Parameter synthesis

The parameter synthesis problem is to determine all the values that parameters can take such that the satisfaction of the property $\mathcal{P}_{[p_l,p_u]}(\Diamond^{\leqslant t} s_g)$ is ensured in the derived instance model. We define the *synthesis region* $\zeta_{syn} \subseteq \zeta_0$ to be the set of valuations, such that each valuation (or point) $v = (x'_1, ..., x'_m)$ therein induces an instance CTMC $\mathcal{C}[v]$, for which $f(x'_1, ..., x'_m) \in [p_l, p_u]$. The aim is to find the (approximate) synthesis region ζ_{syn}. To enable easy visualization, we restrict to pCTMCs with at most two parameters. Our techniques can be applied to more-parameter cases, however, the computational complexity will grow drastically.

Synthesis regions. Given the pCTMC $\mathcal{C}^{(x_1,x_2)}$ and property $\mathcal{P}_{[p_l,p_u]}(\Diamond^{\leqslant t} s_g)$, the transient probability $z = f(x_1, x_2)$ defines a surface, see Fig. 3(a) for an (artificial) example. For $z \bowtie p$ ($\bowtie \in \{<, \leqslant, >, \geqslant\}$, $p \in [0, 1]$ a constant), the projection of the surface on the x_1x_2-plane $z = p$ (in particular in region ζ_{range}) is a region $\zeta_{\bowtie p}$ with boundary curve ∇_p. $\zeta_{\bowtie p}$ is the set of points (x_1, x_2) such that $f(x_1, x_2) \bowtie p$. The boundary curve ∇_p is given by $f(x_1, x_2) - p = 0$. The region $\zeta_{[p_l,p_u]}$ is the intersection of $\zeta_{\geqslant p_l}$ and $\zeta_{\leqslant p_u}$, where p_l and p_u are the probability bounds on the reachability property $\Diamond^{\leqslant t} s_g$. As an example, the shaded areas in Fig. 3(b) and 3(c) depict the region $\zeta_{\geqslant p_l}$ and $\zeta_{\leqslant p_u}$ derived from the projection of the surface in Fig. 3(a) on $z = p_l$ and $z = p_u$. The intersection $\zeta_{\geqslant p_l} \cap \zeta_{\leqslant p_u}$ (Fig. 3(d)) is the synthesis region $\zeta_{[p_l,p_u]}$, given ζ_{range} the rectangular area.

Note that in general it is impossible to get the exact shape of the boundary curve $f(x_1, x_2) - p = 0$ (as $f(x_1, x_2)$ is a high-degree polynomial) as well as the exact synthesis region. As a result, we use *a set of linear line segments* to approximate the boundary curves, thus the approximate synthesis region is *a set of polygons*. This will be explained in more detail in Section 4.2.

Algorithm 1 Framework for obtaining approximate synthesis regions.

```
Require: pCTMC $\mathcal{C}^{(x_1,x_2)}$, formula $\mathcal{P}_{[p_l,p_u]}(\Diamond^{\leqslant t} s_g)$
Ensure: approximate synthesis region $\zeta_{syn}$ (a set of polygons)
 1: compute $\zeta^*_{syn} := \zeta_{\leqslant p_u} \cap \zeta_{\geqslant p_l} \cap \zeta_{range}$;
   1.1: discretize $\zeta_{range}$ on $x_1x_2$-plane by grid steps $\Delta_i$($i$=1,2);
   1.2: compute all intersection points (short as int. pts.)
          of grid lines with boundary curves $\nabla_{p_l}, \nabla_{p_u}$;
   1.3: identify to which polygon each int. pt. belongs;
   1.4: if necessary, refine the grid with $\Delta'_i(<\Delta_i)$; goto Step 1.2;
   1.5: connect certain int. pts. to form approximate polygons;
 2: compute $\zeta_{syn} := \zeta^*_{syn} \cap \zeta_{rate}$;
   2.1: discretize $\zeta_{rate}$ on $x_1x_2$-plane with the same grid;
   2.2: intersect $\zeta^*_{syn}$ and $\zeta_{rate}$ grid by grid;
```

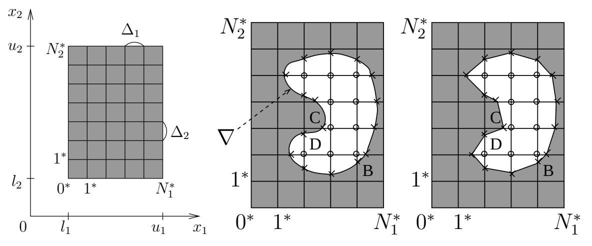
A high-level algorithm to obtain an approximate synthesis region is given in Alg. 1. The algorithm proceeds in two steps: in Step 1 we obtain a first approximation ζ^*_{syn} of the synthesis region while ignoring the rate constraint ζ_{rate}; in Step 2, we compute $\zeta_{syn} = \zeta^*_{syn} \cap \zeta_{rate}$ as the final synthesis region such that all the points that will induce a negative rate (thus not a CTMC at all) are removed. Steps 1.1 and 2 are based on the discretization techniques of Section 4.1. Steps 1.2-1.5 will be explained in more detail in Section 4.2 and 4.3 as the SRA algorithm (Alg. 2).

4.1 Discretization

Given the parameter set $\mathcal{X} = \{x_1, x_2\}$ and ζ_{range}, we specify a discretization step $\Delta_i \in \mathbb{R}_{>0}$ for each parameter x_i ($i = 1, 2$), such that $u_i - l_i = N_i\Delta_i$. Thus, the range $[l_i, u_i]$ of values that variable x_i can take is partitioned into N_i subintervals:

$$[l_i,l_i{+}\Delta_i]\,,(l_i{+}\Delta_i,l_i{+}2\Delta_i],...,(l_i{+}(N_i-1)\Delta_i,l_i{+}N_i\Delta_i].$$

The values $l_i + j\Delta_i$ ($0 \leqslant j \leqslant N_i$) are assigned indices $0, 1, ..., N_i$. We obtain a 2-dimensional grid, where the *grid points* are of the form $gp = (j_1, j_2)$ for $0 \leqslant j_i \leqslant N_i$ with the valuation $(l_1 + j_1\Delta_1, l_2 + j_2\Delta_2)$ and a *grid cell* is a smallest rectangle with grid points as its four vertices. The region ζ_{range} consists of at most $(N_1 + 1)(N_2 + 1)$ grid points and each point gp induces an instance CTMC $\mathcal{C}[gp]$ by the valuation of gp. The transient probability $\pi^{\mathcal{C}[gp]}_{s_g}(t)$ is computed by standard methods for computing transient probabilities in CTMCs. It is important to realize that this yields a discretization in the sense that instead of checking



(a) discretized ζ_{range} (b) accurate ζ^*_{syn} (c) apprx. ζ^*_{syn}

Figure 4. Discretization using grids

each point in the dense region ζ_{range}, we only check the discrete grid points as "samples".

Example 3 *Given the shaded range region* ζ_{range} *formed by* $[l_1, u_1]$ *and* $[l_2, u_2]$*, the grid discretization is as in Fig. 4(a), where* $\Delta_1 = \Delta_2$*. The grid indices are marked with a star to distinguish them from the ones on the axes.*

It is usually convenient to choose a global Δ (i.e., Δ=Δ_1=Δ_2) so that the approximation error is the same in each dimension. Since the grid points near the curves are much more important, it is possible to use a *non-uniform* grid, e.g., smaller grid steps near the curve and larger grid steps away from the curve.

4.2 Approximating the synthesis region

As a next step, we characterize the region ζ^*_{syn} by approximating its boundary curves $\nabla_{p_l}, \nabla_{p_u}$ by a set of polygons. A grid point gp be *positive* if $\pi^{\mathcal{C}[gp]}_{s_g}(t) \in [p_l, p_u]$ and *negative* otherwise. Two grid points $gp = (i, j)$ and $gp' = (i', j')$ are *adjacent* if $|i - i'| + |j - j'| = 1$. The main idea is to find all intersection points of the boundary curves $\nabla_{p_l}, \nabla_{p_u}$ with the grid lines. These intersection points are then connected to approximate the curves ∇_{p_l} and ∇_{p_u}. The resulting area bounded by the approximate curves is a set of polygons (polyhedra, in case of more than 2 parameters). This idea is illustrated by Fig. 4(b) and 4(c), where the white area in Fig. 4(b) is the accurate synthesis region ζ^*_{syn}, the circle-marked grid points are positive, and the intersection points are marked with X. The approximate synthesis region is the white polygon in Fig. 4(c).

Let us explain the algorithm in more detail. We represent the polygon ζ by a *tree*, such that one (positive) grid point gp inside ζ is picked as the *root* and all other (positive) grid points in ζ are *internal nodes*, while the intersection points are *leaf nodes*. This terminology will be used interchangeably in the rest of the section. Since the synthesis region may consist of more than one polygon (tree), we need a root tag to indicate to which tree a leaf or an internal node belongs. A leaf or an internal node without a root tag is called an *orphan*. The approximate synthesis region ζ^*_{syn} is represented by a set of polygons, i.e., sets of line segments.

The sketch of the synthesis region approximation (SRA) algorithm is shown in Alg. 2.

Algorithm 2 Synthesis Region Approximation (SRA)

```
Require: $f(x_1, x_2)$, $[l_i, u_i]$, $\Delta_i$ ($i = 1, 2$), $\Delta_{\min}$
Ensure: polygon approximate synthesis region $\zeta^*_{syn}$
    /* initialization */
 1: find all int. pts. between $\nabla_{p_l}, \nabla_{p_u}$ with the grid lines;
    /* label intersection and grid points with root tags */
 2: while (there exists a positive orphan grid point $gp$) do
 3:   make $gp$ the root of a new tree;
      $gp$ is set as an unexplored tree node;
 4:   while (tree $gp$ has unexplored node $curt$) do
 5:     $curt$ is set to be explored;
 6:     for each ($curt$'s adjacent node $adj$) do
 7:       if ($adj$ is positive $\wedge$ $adj$ is orphan $\wedge$
          #IntPts($curt$,$adj$)=0)   then
 8:         let $adj$ have the same root as $curt$;
 9:         $adj$ is set as an unexplored tree node;
10:       elseif ( ($adj$ is pos. $\wedge$#IntPts($curt$,$adj$) $> 0$) $\vee$
          ($adj$ is negative))
11:         find the leaf node $lp$ closest to $curt$;
            make $lp$'s root the same as $curt$'s root;
12:       end if
13:   end while
14: end while
    /* refine the discretization steps, if necessary */
15: if $\min\{\Delta_1, \Delta_2\} \geqslant \Delta_{\min}$ then
16:   for each ($gc$ with (#leaves($gc$) $\geqslant 4$) or
      (#leaves($gc$)=2 $\wedge$ #PosVtx($gc$)=0, 4))
17:       refine the nine grids with $gc$ in the middle;
    /* connect the intersection points to form polygons */
18: for each ($lp_1, lp_2$ with the same root in the same grid)
19:   add line segment $lp_1$-$lp_2$ as one side of a polygon $\zeta^*_{syn}$;
20: return $\zeta^*_{syn}$;
```

Initialization. Prior to running the main core of the algorithm, we determine all intersection points of the boundary curves ∇_{p_l} and ∇_{p_u} with the grid lines (cf. line 1). For each grid line $x_i = j\Delta_i$ ($i = 1, 2$ and $0 \leqslant j \leqslant N_i$), we solve the following system of equations:

$$\begin{cases} f(x_1, x_2) - p_l = 0 \\ x_i = j\Delta_i \end{cases} \qquad \begin{cases} f(x_1, x_2) - p_u = 0 \\ x_i = j\Delta_i \end{cases},$$

which boils down to solving a *single variable* polynomial function, which in general can be solved by standard root-finding algorithms [8, 10, 17]. Since the polynomial function is usually of (very) high-degree, the method in [19] is more applicable. In total, we need to solve $2(N_1 + N_2 + 2)$ polynomials, since we have 2 curves and $N_i + 1$ grid lines in dimension i. Note that for pCTMCs with more than 2 parameters, obtaining all the intersection points is more costly, as the number of grid points increases exponentially with the number of parameters.

Label grid and intersection points. We will explain the main part of the SRA algorithm with the aid of Fig. 5(a), where the grid points (circles for positive and squares for negative ones) are denoted by $a, b, ...$, the intersection points (X) are denoted by $1, 2, ...$ and grid cells are denoted by A,B,C.... We use #IntPts(gp_1, gp_2) to denote the number of intersection points between grid points gp_1 and gp_2. Starting from a positive grid point $curt$ as a root node (say c), we search its four adjacent grid points $adj \in \{w, n, e, s\}$. Some possible cases are:

- if adj is positive and orphan and #IntPts$(curt, adj) = 0$ (see Alg. 2, line 7), then adj should belong to the same polygon as $curt$, thus share the same root tag. This applies in Fig. 5(a) to grid points c and n.
- if adj is positive and #IntPts$(curt, adj) > 0$ (line 10, first disjunct), then adj and $curt$ belong to different polygons (trees), see e.g. grid points c and w and their polygons in Fig. 5(b) at the same position. In this case, we pick the intersection point lp which is closest to $curt$ and label it with the same root as $curt$ (line 11).
- If adj is negative (line 10, second disjunct), then #IntPts$(curt, adj) > 0$ (since $curt$ is positive), see e.g. grid points c and s or e in Fig. 5(a). We deal with this case the same as the previous one (line 11).

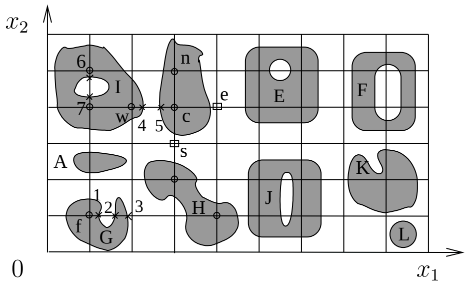
When each grid point in a polygon is explored (line 4) – its four adjacent grid points have been checked – we can choose another root node until there are no positive orphan grid points (line 2). The justification for taking the closest intersection point lp is that lp is for sure on the boundary of the current polygon (see point c and 5, not 4).

As is typical for polygon approximation algorithms, the algorithm is not guaranteed to find all regions correctly. We might exclude some intersection points (e.g., only point 1 is obtained for the tree f, 2 and 3 are regarded not in the same polygon). The main cause is that the grid is too coarse. This can be repaired by a grid refinement, as explained below.

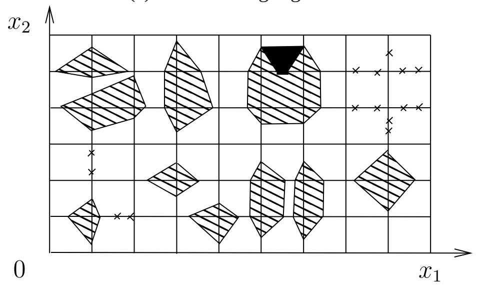
4.3 Refinement

The above root-labeling algorithm does not guarantee to find all regions. This, of course, depends on the granularity of the grid. For instance, in Fig. 5(a), the two positive grid points in grid cell H are in the same region, but since they are not adjacent, they will be identified as the roots of two different trees (and thus give rise to two polygons), cf. Fig. 5(b). The approximation of grid cell G is also too coarse, since intersection points 2 and 3 are orphans.

A solution to find the neglected regions is by refining the discretization steps. Let #leaves(gc) denote the number of intersection points on the four sides of grid cell gc. For instance, grid cell H in Fig. 5(a) has four leaves. If a leaf point lp coincides with one of gc's vertices or is the tangent point



(a) root-labeling algorithm



(b) polygon approximation

Figure 5. Synthesis Region Approximation

between a boundary curve and one of the grid sides, then it will be counted twice. Note that #leaves(gc) is always even. We also define the *number of positive vertices* of gc, denoted by #PosVtx(gc). For instance, #PosVtx(H) $= 2$. Note that #PosVtx(gc) can be at most 4.

Where to refine? Let us now explain when refinement is necessary. We list some combinations of #leaves(gc) and #PosVtx(gc) below, each with an example grid cell. For the "to refine?" column, *yes* is clear; whereas "*check convexity*" indicates a conditional refinement.

#leaves	#PosVtx	ex.grid cell	to refine?
2	0	A (Fig. 5(a))	*yes*
2	1	B (Fig. 4(b))	*check convexity*
2	2	C (Fig. 4(b))	*check convexity*
2	3	D (Fig. 4(b))	*check convexity*
2	4	E (Fig. 5(a))	*yes*
4	0	F (Fig. 5(a))	*yes*
4	1	G (Fig. 5(a))	*yes*
4	2	H (Fig. 5(a))	*yes*
4	3	I (Fig. 5(a))	*yes*
4	4	J (Fig. 5(a))	*yes*

- #leaves$(gc) = 2 \wedge$ #PosVtx$(gc) \in \{1, 2, 3\}$

  In general, we have to check the convexity of the curve in gc in order to decide whether or not to refine gc. In particular, if the curve in gc is *convex*, then it *does not* need to be refined; otherwise, it needs a refinement.

  The convexity as well as #leaves$(gc) = 2$ ensure that the line segment (between the two leaves) closely approximates the curve in gc. This applies to all the grid cells in Fig. 4(b).

The nonconvexity, on the other hand, indicates that the curve has protuberances that a line segment *cannot* sufficiently approximate, cf. grid cell K in Fig. 5(a).

However, it is quite costly to check the convexity in each grid cell. In practice, we choose *not* to refine in this case. If the grid step is sufficiently small, the protuberances inside one grid cell are negligible.

- $\#\text{leaves}(gc) = 2 \wedge \#\text{PosVtx}(gc) \in \{0, 4\}$

  A refinement is required in this case as some area is smaller than a grid cell, see grid cells A and E, where A has orphan leaves. For E, all the four vertices (intersection points) of the black trapezoid in Fig. 5(b) belong to the same polygon according to the algorithm, however, it is unknown how these four points are connected as sides of a (larger) polygon.

- $\#\text{leaves}(gc) \geqslant 4$

  Typically, the more intersection points gc has, the more possible that some locally abrupt behavior (or protuberances) of the curve occurs in gc. Since the area of interest is usually smaller than a grid cell, the discretization is too coarse and needs a refinement.

  This can be seen by all the $\#\text{leaves}(gc) = 4$ cases shown in the table, where grid cells H, I, J split one connected region into two separated polygons, while grid cells F and G have orphan leaves. None of those grid cells yield a close approximation.

  For $\#\text{leaves}(gc) > 4$, they will be refined due to the similar reasons.

- $\#\text{leaves}(gc) = 0 \wedge \#\text{PosVtx}(gc) \in \{0, 4\}$

  The grid cell gc is either completely outside the polygon ($\#\text{PosVtx}(gc) = 0$) or completely inside ($\#\text{PosVtx}(gc) = 4$). Thus, there is no need to refinement for this case.

How to refine? The table can be used as a criterion to check whether or not a grid cell needs to be refined. Once we have identified the suspicious grid cell gc, the following question is how to refine? There can be different strategies for refinement [20], e.g., global vs. local; with uniform or non-uniform steps; how to reduce the discretization steps, etc. The strategies highly depend on the application, i.e., the structure of the polygons.

For the sake of simplicity, we consider one strategy, namely, the local and bisectional refinement. To be more exact, we will refine locally the area of 9 grid cells with gc in the center. Note that it is also possible to refine more or fewer (than 9) grid cells as the "local area". A new discretization with step size $\frac{1}{2}\Delta_i$ will be performed on those grid cells. For the new grid points, redo the SRA algorithm until either the discretization step is less than the user-defined $\Delta_{\min}$ or there are no grids to refine due to our criterion.

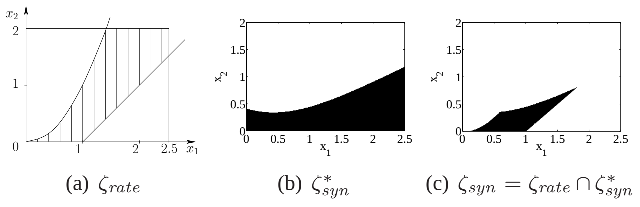


Figure 6. Example synthesis regions

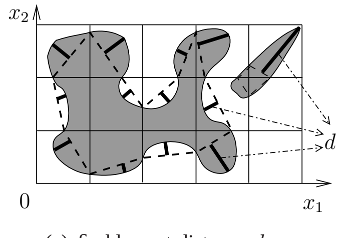
Construct the polygons. In case the algorithm terminates before $\Delta_{\min}$ is reached, it is guaranteed that there does not exist any grid cell with more than 2 intersection points. Hence, obtaining polygons amounts to connecting the intersection points which share the same root within the same grid cell, see Fig. 4(c). Otherwise ($\Delta_{\min}$ is reached), the intersection points can be connected according to the same rules, but certain regions might not be detected. For instance, the rightmost circle in grid cell L has no intersection points with any grid lines in Fig. 5(a), and thus cannot be detected. These regions are only bounded in one grid cell. Thus, given a small discretization step, the undetected areas can be safely neglected within the predefined error bound. Note that to obtain a more precise approximation, we can take other discretization techniques, say, adding diagonal lines as well. In this case, a cell is a triangle, where our algorithm can be adapted easily.

Example 4 *Consider the* pCTMC *in Fig. 1, and let the discretization steps* $\Delta_1 = \Delta_2 = 0.01$, *uniformization error bound* $\varepsilon = 10^{-6}$ *and property* $\mathcal{P}_{\geqslant 0.5}(\Diamond^{\leqslant 0.5}\, s_g)$. *Given the rate region* ζ_{rate} *as in Fig. 6(a),* ζ^*_{syn} *and* ζ_{syn} *are as shown in Fig. 6(b) and Fig. 6(c), respectively. We omit the grid lines so as to make the figure readable. Since no grid cell has to be refined according to our criteria, the final region remains as* ζ_{syn}.

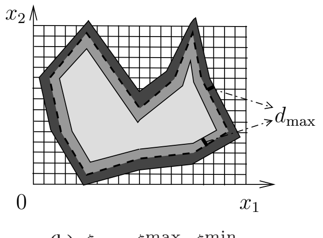
4.4 Efficiency and accuracy

Time complexity. In the worst case, the discretization step is $\Delta_{\min}$ and there are $N_i = \frac{u_i - l_i}{\Delta_{\min}}$ $(i = 1, 2)$ subintervals. Obtaining the closed-formed expression of $f(x_1, x_2)$ (see Eq. (7)), takes $\mathcal{O}(n^2 qt)$ time, like computing the transient probabilities. For the initialization, $2(N_1 + N_2 + 2)$ polynomial equations have to be solved with precision $2^{-\beta}$ (β is the bit precision). Using the algorithm in [17], this has time complexity $\mathcal{O}\left(\hat{k}^2_\varepsilon \log(\hat{k}_\varepsilon) \log(\beta \hat{k}_\varepsilon)\right)$, where $\hat{k}_\varepsilon$ is the degree of the polynomial $f(x_1, x_2)$ (see the end of Section 3). For the root tag labeling part, the time complexity is in the order of the number of grid points, i.e., $\mathcal{O}(N_1 N_2)$. Evaluating $f(x_1, x_2)$ at each grid point takes $\mathcal{O}(\hat{k}_\varepsilon)$ time. Gathering these complexity figures we have:

Theorem 5 *The worst case time complexity of the SRA is:*



(a) find largest distance $d_{\max}$



(b) ζ_{syn}, $\zeta_{syn}^{\max}$, $\zeta_{syn}^{\min}$

Figure 7. Error bound analysis

$$\mathcal{O}\left(n^2qt + \hat{k_\varepsilon}N_1N_2 + \hat{k}_\varepsilon^2\log(\hat{k}_\varepsilon)\log(\beta\hat{k}_\varepsilon)(N_1{+}N_2)\right),$$

where $\hat{k}_\varepsilon$ is of the order $\mathcal{O}(mqtr)$ with r the maximal degree of the polynomial expressions in the pCTMC.

Error bound. An important question is how accurate the derived synthesis region is. Let us explain this by using Fig. 7(a), where the accurate region is the gray area and its approximation is the dashed polygon. We assume that within any grid cell the boundary curve is convex. Let d_i be the distance between the line segment approximating the curve and the tangent (with the same slope) to the curve in the grid cell i. Let $d_{\max} = \max_i\{d_i\}$ be the largest such distance. It is, however, very costly to compute every d_i and thus $d_{\max}$. In practice, $d_{\max}$ is taken to be $\sqrt{\Delta_1^2 + \Delta_2^2}$ which is the maximal value it can take. The top-rightmost distance in Fig. 7(a) is very close to this upper bound.

Given the approximate polygon ζ_{syn} (dashed polygon in Fig. 7(b)) and $d_{\max}$, we can construct polygons $\zeta_{syn}^{\max}$ (the largest polygon in Fig. 7(b)) and $\zeta_{syn}^{\min}$ (the smallest polygon), where distance $d_{\max}$ is added and subtracted from the boundary of ζ_{syn}, respectively. The points in $\zeta_{syn}^{\max} - \zeta_{syn}^{\min}$ *may* induce a valid CTMC, while the points in $\zeta_{syn}^{\min}$ *always* induce a valid CTMC. $\zeta_{syn}^{\min}$ can be regarded as the "safe" synthesis region.

5 Case study

We apply our approximation algorithm to a concrete case study from the literature. A *storage system with error checking* incorporates redundant data for discovery of and recovery from errors caused by hardware or software faults [4]. The use of redundancy enhances the reliability of the storage system but unavoidably degrades the system's performance because of the extra processing time required for error checking. Typically, on every request it is checked whether an error occurred. *Probabilistic error checking* can be applied to reduce the error checking overhead. In particular, each access operation will be followed by an error checking with probability $r \in [0,1]$, instead of almost surely (i.e., $r = 1$). Such a storage system can be modeled by a pCTMC as indicated in Fig. 8.

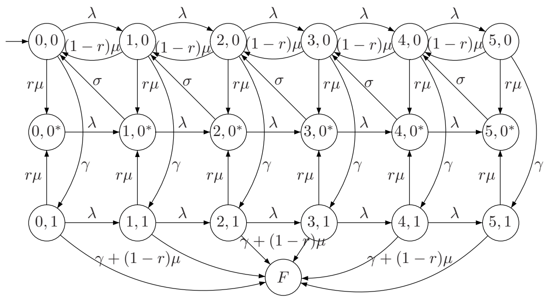
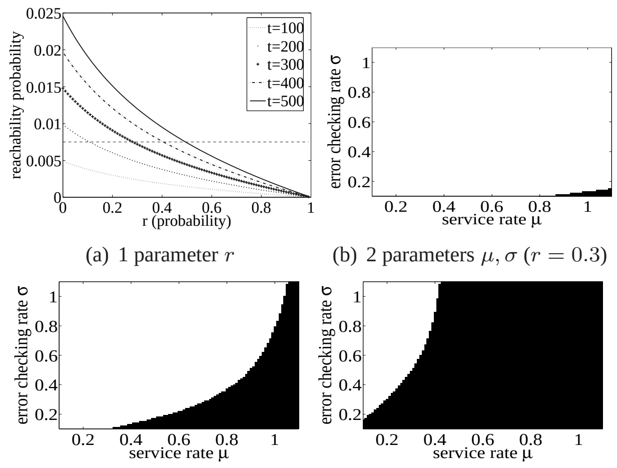


Figure 8. A storage system with probabilistic error checking, with queue capacity 5

The storage system is *1-correctable*, i.e., the system can recover from a single error, but fails (state F) if two or more errors occur. We suppose that all access operations (reads and writes) as well as the error checking are atomic and all delays involved (such as arrivals, checks, etc.) are exponentially distributed. Access operation requests *arrive* with rate λ and are *served* with rate μ; the hardware/software will *fail* with rate γ, while the *error checking* takes place with rate σ. The arrived but not yet served requests are queued. We assume a queue capacity of 5. The states of the pCTMC are of the form (i,j), where i is the number of queued access operation requests and j the number of errors (0 or 1); an asterisk indicates that an error check is being performed. The property of interest is $\mathcal{P}_{\leqslant p}(\Diamond^{\leqslant t} s_F)$.

Typically, the probability r can be *logically adjusted* to guarantee some given specification. On the other hand, μ, σ, and γ can be *physically adjusted* by changing the hardware/software. In the following, we show the experimental results for 1) one parameter r, i.e., for which error checking probability r can we guarantee that the probability to reach the fail state (within the deadline) is low, e.g., less than 0.0075? 2) two parameters μ and σ, i.e., how fast should read/write requests be handled and errors be checked in order to obtain a low failure probability? In all computations the error bound for uniformization is $\varepsilon = 10^{-6}$.

One parameter: r. Let $\lambda = 0.3$ (0.3 access operation requests per second), $\mu = 0.5$, $\sigma = 0.5$ and $\gamma = 5 \times 10^{-5}$ (an average time of two consecutive errors is approximately 5 days). The parameter r has initial range $[0,1]$ and the discretization step $\Delta = 0.01$. For the specification $\Phi_1 = \mathcal{P}_{\leqslant 0.0075}(\Diamond^{\leqslant t} s_F)$, where $t \in \{100, ..., 500\}$, the synthesis region is an interval as shown in Fig. 9(a), where the probability threshold $p = 0.0075$ is marked by a dashed line. For $t = 100$, the entire range $r \in [0,1]$ is safe; intuitively, it is less probable to fail given a small period of time. For $t = 200, ..., 500$, r approximately lies in the intervals $[0.1, 1]$, $[0.28, 1]$, $[0.41, 1]$, $[0.5, 1]$, respectively. The larger the time bound, the higher the error checking probability r should be to satisfy Φ_1.



(a) 1 parameter r (b) 2 parameters μ, σ ($r = 0.3$)

(c) 2 parameters μ, σ ($r = 0.5$) (d) 2 parameters μ, σ ($r = 0.7$)

Figure 9. Syn. regions for storage system

Two parameters: μ and σ. Let $\lambda = 0.3$, $\gamma = 5 \times 10^{-5}$ and $r = 0.3$, 0.5 or 0.7. The parameter ranges are $\mu \in [0.1, 1.1]$ and $\sigma \in [0.1, 1.1]$, with $\Delta_\mu = \Delta_\sigma = 0.01$. The initial region ζ_0 is the same rectangular area as ζ_{range}. For the specification $\Phi_2 = \mathcal{P}_{\leqslant 0.002}(\diamondsuit^{\leqslant 200}\, s_F)$, the synthesis regions are the black regions as shown in Fig. 9(b) through 9(d) for different values of r. Notice that the shape of the boundary curves is simple (i.e. without local protuberances), refinement is not performed. The higher the error checking probability r is, the larger the region for which Φ_2 holds. If error checking takes longer (i.e., with a low error checking rate σ), then it is less probable to fail. This is due to the assumption that during the error checking, no error will occur. On the other hand, if a request is served faster (i.e., with high service rate μ), then it is less probable to fail. This is because faster serving causes less errors. In practice, an error checking is preferred to be carried out fast for the sake of efficiency. We, therefore, can adjust the combination of μ and σ to meet the specification and enhance the efficiency.

6 Conclusion

The central question that we considered is: for a stochastic real-time system with parametric random delays, can we find sets of parameter values for which a given specification is satisfied? This paper presented an algorithm that yields an approximation of these values for CTMCs and time-bounded reachability specifications. To our knowledge, this is the first parameter synthesis approach in this setting. An example from the literature showed the feasibility of our approach. Note that most of the time we will have a very-high-degree polynomial function to solve, however there are existing algorithms [19] to factorize such polynomials efficiently. Future work will concentrate on improvements of our algorithm.

References

[1] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *STOC*, pages 592–601, 1993.

[2] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Publishing, 2003.

[3] C. Baier, B. R. Haverkort, H. Hermanns, and J.-P. Katoen. Model-checking algorithms for continuous-time Markov chains. *IEEE Trans. Software Eng.*, 29(6):524–541, 2003.

[4] I.-R. Chen and I.-L. Yen. Analysis of probabilistic error checking procedures on storage systems. *Comput. J.*, 38(5):348–354, 1995.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms*. The MIT Press and McGraw-Hill Book Company, 2001.

[6] C. Daws. Symbolic and parametric model checking of discrete-time Markov chains. In *ICTAC*, LNCS 3407, pages 280–294, 2004.

[7] R. Fröberg. *An Introduction to Gröbner Bases*. John Wiley & Sons, 1998.

[8] R. W. Hamming. *Numerical Methods for Scientists and Engineers*. McGraw-Hill, 1973.

[9] E. Hansen. *Global Optimization Using Interval Analysis*. Dekker, New York, 1992.

[10] P. Henrici and W. R. Kenan. *Applied & Computational Complex Analysis: Power Series Integration Conformal Mapping Location of Zero*. John Wiley & Sons, 1988.

[11] T. Hune, J. Romijn, M. Stoelinga, and F. W. Vaandrager. Linear parametric model checking of timed automata. *J. Log. Algebr. Program.*, 52-53:183–220, 2002.

[12] A. Jensen. Markoff chains as an aid in the study of Markoff processes. *Skand. Aktuarietidskrift*, 36:87–91, 1953.

[13] M. Z. Kwiatkowska, G. Norman, and D. Parker. Probabilistic symbolic model checking with PRISM: a hybrid approach. *STTT*, 6(2):128–142, 2004.

[14] E. Laguerre. Sur la théorie des équations numériques. *J. Math. Pures Appl. (3e série)*, 9:99–146, 1883.

[15] R. Lanotte, A. Maggiolo-Schettini, and A. Troina. Parametric probabilistic transition systems for system design and analysis. *Formal Asp. Comput.*, 19(1):93–109, 2007.

[16] J. Nocedal and S. Wright. *Numerical Optimization*. Springer, New York, 1999.

[17] V. Y. Pan. Approximating complex polynomial zeros: Modified Weyl's quadtree construction and improved Newton's iteration. *J. Complexity*, 16(1):213–264, 2000.

[18] A. Schrijver. *Theory of Linear and Integer Programming*. John Wiley & Sons, 1998.

[19] G. A. Sitton, C. S. Burrus, J. W. Fox, and S. Treitel. Factoring very-high-degree polynomials. *Signal Processing Magazine, IEEE*, 20(6):27–42, 2003.

[20] H. Stetter. *Analysis of Discretization Methods for Ordinary Differential Equations*. Springer-Verlag, 1973.

[21] I. Stewart. *Galois Theory*. Chapman and Hall, 1989.

[22] A. Thümmler, P. Buchholz, and M. Telek. A novel approach for phase-type fitting with the EM algorithm. *IEEE Trans. Dependable Sec. Comput.*, 3(3):245–258, 2006.

[23] D. Zhang and R. Cleaveland. Fast on-the-fly parametric real-time model checking. In *RTSS*, pages 157–166, 2005.