Approximate Parameter Synthesis for Probabilistic Time-Bounded Reachability

Tingting Han\textsuperscript{1,2}, Joost-Pieter Katoen\textsuperscript{1,2}, Alexandru Mereacre\textsuperscript{1}

\textsuperscript{1} Software Modeling and Verification, RWTH Aachen University, Germany
\textsuperscript{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
of state changes. They are expressed using the polynomial ring over reals, allowing rate expressions such as $3\alpha - \beta$ 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 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) $x_1, \ldots, x_m$, 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 $v : \mathcal{X} \to \mathbb{R}$ 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 $1 \leq i \leq m$. 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 $\alpha \propto c$, where $\alpha \in \mathbb{R}[\mathcal{X}]$, $\propto \in \{<, \leq, =, >, \geq\}$ and $c \in \mathbb{R}$. A constraint is a conjunction of atomic constraints. A valuation $v$ satisfies the constraint $\alpha \propto c$ if $\alpha[v] \propto c$. 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, $\mathbf{R} : \mathbb{R} \times S \to \mathbb{R}_{\geq 0}$ 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 $S$, $E = [E(s_1), \ldots, E(s_n)]$ is the vector of exit rates, where $E(s_i) = \sum_{s' \in S} \mathbf{R}(s_i, s')$. The vector $\pi(t) = [\pi_1(t), ..., \pi_n(t)]$ gives the transient probability of the CTMC, i.e., the probability of being in state $s_i$ ($1 \leq i \leq n$) at time $t$. The Chapman-Kolmogorov equations describe the evolution of the transient probability distribution over time:

$$\frac{d\pi(t)}{dt} = \pi(t)Q,$$

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

Definition 2 ($p$CTMCs) 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 $\mathbf{R}(\mathcal{X}) : S \times S \to \mathbb{R}[\mathcal{X}]$ is the parametric rate matrix.

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

Definition 3 (Instance CTMCs) For $p$CTMC $\mathcal{C}(\mathcal{X})$ 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 $v(x_i)$. 
In Fig. 1(a), we illustrate a pCTMC over \( \{x_1, x_2\} \) with the range constraint: \( 0 \leq x_1 \leq 2.5 \) and \( 0 \leq x_2 \leq 2 \). The rate constraint is as follows:

\[
2x_1 + 4 \geq 0 \land 2 - x_2 \geq 0 \land x_2 - x_1 + 1 \geq 0 \land x_1^2 - x_2 \geq 0.
\]

\( \zeta_{\text{range}} \) is the shaded area in Fig. 1(b), while \( \zeta_{\text{rate}} \) is the area between the two curves. The initial region \( \zeta_0 \) is the shaded area. Any point in \( \zeta_0 \) 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).

**Figure 1. pCTMCs and related concepts**

For a pCTMC \( C(X) = (S, R(X), s_0) \) over the set of parameters \( \mathcal{X} = \{x_1, ..., x_m\} \), the initial region \( \zeta_0 \) satisfies the following constraint:

\[
\bigwedge_{i=1}^m l_i \leq x_i \leq u_i \land \bigwedge_{s, s' \in S} R(X)(s, s') \geq 0, \quad (2)
\]

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

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

\[
2x_1 + 4 \geq 0 \land 2 - x_2 \geq 0 \land x_2 - x_1 + 1 \geq 0 \land x_1^2 - x_2 \geq 0.
\]

\( \zeta_{\text{range}} \) is the rectangular area in Fig. 1(b), while \( \zeta_{\text{rate}} \) is the shaded area. Any point in \( \zeta_0 \) 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 \([p_t, p_u]\)? In temporal logics, such as CSL [3], this is formalized as \( P_{[p_t, p_u]}(\diamond^{\leq t} s_g) \), where \( 0 \leq p_t \leq p_u \leq 1 \) and \( \diamond^{\leq t} s_g \) denotes the reachability of \( s_g \) within \( t \) time units.

Note that the interval \([p_t, p_u]\) 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 \( P_{[p_t, p_u]}(\diamond^{\leq t} s_g) \) holds. The computation of reachability probabilities boils down to computing transient probabilities. Formally,

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

\[
\text{Prob}(s_0, \diamond^{\leq t} s_g) = \pi_{s_g}(t), \quad (3)
\]

where \( \text{Prob}(s_0, \diamond^{\leq t} s_g) \) 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 \( \pi(t) \) 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 \( C = (S, R, s_0) \), let \( q \geq \max_{s \in S} \{E(s)\} \), and define the matrix \( P = I + \frac{Q}{q} \), where \( I \) is the identity matrix of cardinality \( |S| \). \( \pi(t) \) is then computed as:

\[
\pi(t) = \pi(0) \cdot \sum_{i=0}^{\infty} \frac{(qt)^i}{i!} P^i. \quad (4)
\]

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

\[
\sum_{i=0}^{k_\varepsilon} \frac{(qt)^i}{i!} \geq 1 - \varepsilon e^{-qt} = (1 - \varepsilon) e^{qt}. \quad (5)
\]

If \( qt \) is large, \( k_\varepsilon \) 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 a pCTMC \( C(X) = (S, R(X), s_0) \) with the set \( \mathcal{X} \) of parameters and initial region \( \zeta_0 \), the uniformization rate \( q \) is at least the largest
number that an exit rate can take in $\zeta_0$. Formally,

$$q \geq \max_{1 \leq i \leq n} \left\{ \max_{v \in \zeta_0} \{ E^{(X)}(s_i)[v] \} \right\}$$  \hspace{1cm} (6)

Note that $q$ is a constant. It suffices to first maximize each expression $E^{(X)}(s_i)$ (each is an objective function) within the closed region $\zeta_0$ (the inner $\max$), and then take the maximum out of $n$ candidates (the outer $\max$), where $n = |V|$. 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 $\zeta_0$ 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 $\zeta_0$ 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 $E^{(X)}(s_i)$ 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 $\pi(t)$. Recall that $\pi(t)$ is the $\varepsilon$-approximation of the transient probability vector $\pi(t)$. The reachability probability to $s_y$ within time $t$ equals $\pi_{s_y}(t)$, and thus can be $\varepsilon$-approximated by $\tilde{\pi}_{s_y}(t)$. Similarly, we truncate the infinite sum of $\pi(t)$ to obtain $\tilde{\pi}(t)$. Since the truncation point $k_\varepsilon$ in Eq. (5) is independent of the rates in the CTMC, it coincides with the non-parametric case. The transient probability vector $\tilde{\pi}(t) = \pi(0) \cdot \sum_{i=0}^{k_\varepsilon} e^{-\varepsilon t \frac{(qt)^i}{i!}} (P^{(X)})^i$ can be computed by vector-matrix multiplication. For given $q$ and $t$, $e^{-\varepsilon t \frac{(qt)^i}{i!}}$ is constant, while $(P^{(X)})^i$ contains parameters. Let $\deg_{x_i}(P^{(X)})$ denote the maximal degree of parameter $x_i$ in all expressions in $P^{(X)}$. For instance, $\deg_{x_1}(P^{(X)}) = 2$ and $\deg_{x_2}(P^{(X)}) = 1$ for the pCTMC in Fig. 1(a). Note that $\deg_{x_i}(P^{(X)}) = \deg_{x_j}(R^{(X)})$. The degree of a polynomial is the sum of the degrees of all its variables. $(P^{(X)})_{k_\varepsilon}$ has degree at most $k_\varepsilon$, where $k_\varepsilon = \sum_{i=1}^{m} k_i \cdot \deg_{x_i}(P^{(X)})$. Given $q$ and $t$, the transient probability $\tilde{\pi}_{s_y}(t)$ is a polynomial function over the parameters $x_1, \ldots, x_m$, i.e.,

$$f(x_1, \ldots, x_m) = \sum_{j=i(1),\ldots,i(m)} a_j x_1^{i_1} \cdots x_m^{i_m},$$  \hspace{1cm} (7)

where $i \leq k_\varepsilon \cdot \deg_{x_i}(P^{(X)})$ (1 $\leq \ell \leq m$), and $a_j \in \mathbb{R}$. The degree of $f$ is at most $k_\varepsilon$, which is of order $O(nm r t)$, where $r := \max_{x_i \in X} \deg_{x_i}(P^{(X)})$ is the maximal degree of the polynomial expressions appearing in the pCTMC. Note that $k_\varepsilon$ is usually much larger than $\deg_{x_i}(R^{(X)})$ and
4 Parameter synthesis

The parameter synthesis problem is to determine all the values that parameters can take such that the satisfaction of the property \( P_{l,p_i} \) (\( \langle \leq s_g \rangle \)) is ensured in the derived instance model. We define the synthesis region \( \zeta_{\text{syn}} \subseteq \zeta_0 \) to be the set of valuations, such that each valuation (or point) \( v = (x'_1,...,x'_m) \) therein induces an instance CTMC \( C[v] \), for which \( f(x'_1,...,x'_m) \in [p_i,p_u] \). The aim is to find the approximate synthesis region \( \zeta_{\text{syn}} \). To enable easy visualization, we restrict to \( p \)CTMCs with at most two parameters. Our techniques can be applied to multi-parameter cases, however, the computational complexity will grow drastically.

Synthesis regions. Given the \( p \)CTMC \( C(x_1,x_2) \) and property \( P_{[p_i,p_u]} (\langle \leq s_g \rangle) \), the transient probability \( z = f(x_1,x_2) \) defines a surface, see Fig. 3(a) for an (artificial) example. For \( z \leq p \) (\( \forall \in \{,\leq,>,\geq\} \), \( p \in [0,1] \) a constant), the projection of the surface on the \( x_1 x_2 \)-plane \( z = p \) (in particular in region \( \zeta_{\text{range}} \)) is a region \( \zeta_{\text{op}} \) with boundary curve \( \nabla_p \). \( \zeta_{\text{op}} \) is the set of points \( (x_1,x_2) \) such that \( f(x_1,x_2) \leq p \). The boundary curve \( \nabla_p \) is given by \( f(x_1,x_2) - p = 0 \). The region \( \zeta_{[p_i,p_u]} \) is the intersection of \( \zeta_{\geq p_i} \) and \( \zeta_{\leq p_u} \), where \( p_i \) and \( p_u \) are the probability bounds on the reachability property \( \langle \leq s_g \rangle \). As an example, the shaded areas in Fig. 3(b) and (c) depict the region \( \zeta_{\geq p_i} \) and \( \zeta_{\leq p_u} \) derived from the projection of the surface in Fig. 3(a) on \( z = p_i \) and \( z = p_u \). The intersection \( \zeta_{\geq p_i} \cap \zeta_{\leq p_u} \) (Fig. 3(d)) is the synthesis region \( \zeta_{[p_i,p_u]} \), given \( \zeta_{\text{range}} \) 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.

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 \( \zeta_{\text{syn}}^* \) of the synthesis region while ignoring the rate constraint \( \zeta_{\text{rate}} \). In Step 2, we compute \( \zeta_{\text{syn}} = \zeta_{\text{syn}}^* \cap \zeta_{\text{rate}} \) 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).

### Algorithm 1 Framework for obtaining approximate synthesis regions.

**Require:** \( p \)CTMC \( C(x_1,x_2) \), formula \( P_{[p_i,p_u]} (\langle \leq s_g \rangle) \)

**Ensure:** approximate synthesis region \( \zeta_{\text{syn}} \) (a set of polygons)

\begin{itemize}
  \item 1: compute \( \zeta_{\text{syn}}^* := \zeta_{\geq p_i} \cap \zeta_{\geq p_i} \cap \zeta_{\text{range}} \);
  \item 1.1: discretize \( \zeta_{\text{range}} \) on \( x_1x_2 \)-plane by grid steps \( \Delta(i=1,2) \);
  \item 1.2: compute all intersection points (short as int. pts.) of grid lines with boundary curves \( \nabla_{p_i}, \nabla_{p_u} \);
  \item 1.3: identify to which polygon each int. pt. belongs;
  \item 1.4: if necessary, refine the grid with \( \Delta(i=1,2) \); goto Step 1.2;
  \item 1.5: connect certain int. pts. to form approximate polygons;
  \item 2: compute \( \zeta_{\text{syn}} := \zeta_{\text{syn}}^* \cap \zeta_{\text{rate}} \);
  \item 2.1: discretize \( \zeta_{\text{rate}} \) on \( x_1x_2 \)-plane with the same grid;
  \item 2.2: intersect \( \zeta_{\text{syn}}^* \) and \( \zeta_{\text{rate}} \) grid by grid;
\end{itemize}
each point in the dense region $\zeta_{\text{range}}$, we only check the discrete grid points as “samples”.

**Example 3** Given the shaded range region $\zeta_{\text{range}}$ 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 $\Delta$ (i.e., $\Delta = \Delta_1 = \Delta_2$) 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 $\zeta_{\text{syn}}^*$ by approximating its boundary curves $\nabla_{p_1}$, $\nabla_{p_u}$ by a set of polygons. A grid point $g_p$ be positive if $\pi_{\text{pos}}^{\text{syn}}(g_p) \in [p_i, p_u]$ and negative otherwise. Two grid points $g_p = (i, j)$ and $g_p' = (i', j')$ are adjacent if $|i - i'| + |j - j'| = 1$. The main idea is to find all intersection points of the boundary curves $\nabla_{p_1}$, $\nabla_{p_u}$ with the grid lines. These intersection points are then connected to approximate the curves $\nabla_{p_1}$ and $\nabla_{p_u}$. 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 $\zeta_{\text{syn}}^*$, 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 $\zeta$ by a tree, such that one (positive) grid point $g_p$ inside $\zeta$ is picked as the root and all other (positive) grid points in $\zeta$ 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 $\zeta_{\text{syn}}^*$ 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:** $f(x_1, x_2), [l_i, u_i], \Delta_i (i = 1, 2), \Delta_{\text{min}}$

**Ensure:** polygon approximate synthesis region $\zeta_{\text{syn}}^*$

/* initialization */
1: find all int. pts. between $\nabla_{p_1}$, $\nabla_{p_u}$ with the grid lines;
/* label intersection and grid points with root tags */
2: while (there exists a positive orphan grid point $g_p$ do
3: make $g_p$ the root of a new tree;
4: $g_p$ is set as an unexplored tree node;
5: while (tree $g_p$ has unexplored node $curt$) do
6: $curt$ is set to be explored;
7: for each (curt’s adjacent node $adj$) do
8: if $(adj$ is positive $\land$ adj is orphan $\land$
9: $\#\text{IntPts}(curt, adj)=0) \; \text{then}$
10: let $adj$ have the same root as $curt$;
11: adj is set as an unexplored tree node;
12: elseif ((adj is pos. $\land$ IntPts(curt, adj) > 0) $\lor$
13: (adj is negative))
14: find the leaf node $lp$ closest to curt;
15: make lp’s root the same as curt’s root;
16: end if
17: end while
18: end while
/* refine the discretization steps, if necessary */
19: if $\min\{\Delta_1, \Delta_2\} \geq \Delta_{\text{min}}$, then
20: for each (gc with (#leaves(gc) $\geq$ 4) or
21: (leaves(gc)=2 $\land$ #PosVtx(gc)=0, 4))
22: refine the nine grids with gc in the middle;
/* connect the intersection points to form polygons */
23: for each (lp_1, lp_2 with the same root in the same grid)
24: add line segment lp_1-lp_2 as one side of a polygon $\zeta_{\text{syn}}^*$;
25: return $\zeta_{\text{syn}}^*$;

**Initialization.** Prior to running the main core of the algorithm, we determine all intersection points of the boundary curves $\nabla_{p_1}$ and $\nabla_{p_u}$ with the grid lines (cf. line 1). For each grid line $x_i = j\Delta_i (i = 1, 2$ and $0 \leq j \leq N_i)$, we solve the following system of equations:

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

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 $N_1 + 1$ grid lines in dimension $i$. Note that for $p$CTMCs 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 \(a, b, \ldots\), the intersection points (\(X\)) are denoted by \(1, 2, \ldots\) and grid cells are denoted by A,B,C.\ldots. We use \(#\text{IntPts}(gp_1, gp_2)\) to denote the number of intersection points between grid points \(gp_1\) and \(gp_2\). Starting from a positive grid point \(c\) 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 \(#\text{IntPts}(curt, adj) = 0\) (see Alg. 2, line 7), then \(adj\) should belong to the same polygon as \(curt\), thus share the same root tag.
  
- if \(adj\) is positive and \(#\text{IntPts}(curt, adj) > 0\) (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 \(#\text{IntPts}(curt, adj) > 0\) (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 \(#\text{leaves}(gc)\) 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 between a boundary curve and one of the grid sides, then it will be counted twice. Note that \(#\text{leaves}(gc)\) is always even. We also define the number of positive vertices of \(gc\), denoted by \(#\text{PosVtx}(gc)\). For instance, \(#\text{PosVtx}(H) = 2\). Note that \(#\text{PosVtx}(gc)\) can be at most 4.

**Figure 5. Synthesis Region Approximation**

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

<table>
<thead>
<tr>
<th>(#\text{leaves})</th>
<th>(#\text{PosVtx})</th>
<th>ex. grid cell</th>
<th>to refine?</th>
</tr>
</thead>
<tbody>
<tr>
<td>2</td>
<td>0</td>
<td>A (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>2</td>
<td>1</td>
<td>B (Fig. 4(b))</td>
<td>check convexity</td>
</tr>
<tr>
<td>2</td>
<td>2</td>
<td>C (Fig. 4(b))</td>
<td>check convexity</td>
</tr>
<tr>
<td>2</td>
<td>3</td>
<td>D (Fig. 4(b))</td>
<td>check convexity</td>
</tr>
<tr>
<td>2</td>
<td>4</td>
<td>E (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>0</td>
<td>F (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>1</td>
<td>G (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>2</td>
<td>H (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>3</td>
<td>I (Fig. 5(a))</td>
<td>yes</td>
</tr>
<tr>
<td>4</td>
<td>4</td>
<td>J (Fig. 5(a))</td>
<td>yes</td>
</tr>
</tbody>
</table>

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

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 \(#\text{leaves}(gc) = 2\) 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.

- \#leaves(gc) = 2 \land \#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.

- \#leaves(gc) \geq 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 \#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 \#leaves(gc) > 4, they will be refined due to the similar reasons.

- \#leaves(gc) = 0 \land \#PosVtx(gc) \in \{0, 4\}

The grid cell gc is either completely outside the polygon (\#PosVtx(gc) = 0) or completely inside (\#PosVtx(gc) = 4). Thus, there is no need to refine 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 \( \Delta_{\text{min}} \) or there are no grids to refine due to our criterion.

4.4 Efficiency and accuracy

Time complexity. In the worst case, the discretization step is \( \Delta_{\text{min}} \) and there are \( N_i = \frac{\Delta_{\text{min}} - \Delta_{\text{syn}}}{\Delta} \) (i = 1, 2) subintervals. Obtaining the closed-formed expression of \( f(x_1, x_2) \) (see Eq. (7)), takes \( O(n^2 q t) \) time, like computing the transient probabilities. For the initialization, \( 2(N_1 + N_2 + 2) \) polynomial equations have to be solved with precision \( 2^{-\beta} \) (\( \beta \) is the bit precision). Using the algorithm in [17], this has time complexity \( O\left(k^2 \log(\hat{k}_e) \log(1/\beta k_e)\right) \), where \( \hat{k}_e \) 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., \( O(N_1 N_2) \). Evaluating \( f(x_1, x_2) \) at each grid point takes \( O(\hat{k}_e) \) time. Gathering these complexity figures we have:

**Theorem 5** The worst case time complexity of the SRA is:
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 \( d_{\text{max}} = \max_i \{d_i\} \) be the largest such distance. It is, however, very costly to compute every \( d_i \) and thus \( d_{\text{max}} \). In practice, \( d_{\text{max}} \) is taken to be \( \sqrt{\Delta_f^2 + \Delta_s^2} \) 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 \( \zeta_{\text{syn}} \) (dashed polygon in Fig. 7(b)) and \( d_{\text{max}} \), we can construct polygons \( \zeta_{\text{max}} \) (the largest polygon in Fig. 7(b)) and \( \zeta_{\text{min}} \) (the smallest polygon), where distance \( d_{\text{max}} \) is added and subtracted from the boundary of \( \zeta_{\text{syn}} \), respectively. The points in \( \zeta_{\text{max}} - \zeta_{\text{syn}} \) may induce a valid CTMC, while the points in \( \zeta_{\text{min}} \) always induce a valid CTMC. \( \zeta_{\text{syn}} \) 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.

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 \( \lambda \) and are served with rate \( \mu \); the hardware/software will fail with rate \( \gamma \), while the error checking takes place with rate \( \sigma \). 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 \( \mathcal{P}_{\Phi_1} (\bigotimes_{j} s_F) \).

Typically, the probability \( r \) can be logically adjusted to guarantee some given specification. On the other hand, \( \mu \), \( \sigma \), and \( \gamma \) 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 \( \mu \) and \( \sigma \), 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 \( \Phi_1 = \mathcal{P}_{\leq 0.0075} (\bigotimes_{j} s_F) \), where \( t \in \{100, \ldots, 500\} \), 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 \( t = 200, \ldots, 500 \), \( r \) 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 \( \Phi_1 \).
Two parameters: $\mu$ and $\sigma$. 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 $\Theta_0$ is the same rectangular area as $\Theta_{\text{range}}$. For the specification $\Phi_2 = P_{0.0025}(\Theta_{\leq 0.2} \cap F)$, 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 $\Phi_2$ holds. If error checking takes longer (i.e., with a low error checking rate $\sigma$), 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 $\mu$), 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 $\mu$ and $\sigma$ 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