Parameter Synthesis Through Temporal Logic Specifications

Thao Dang[1], Tommaso Dreossi[1,2(✉)], and Carla Piazza[2]

[1] VERIMAG, 2 avenue de Vignate, 38610, Gieres, France
{thao.dang,tommaso.dreossi}@imag.fr
[2] University of Udine, via delle Scienze, 206 33100, Udine, Italy
carla.piazza@uniud.it

Abstract. Parameters are often used to tune mathematical models and capture nondeterminism and uncertainty in physical and engineering systems. This paper is concerned with parametric nonlinear dynamical systems and the problem of determining the parameter values that are consistent with some expected properties. In our previous works, we proposed a parameter synthesis algorithm limited to safety properties and demonstrated its applications for biological systems. Here we consider more general properties specified by a fragment of STL (Signal Temporal Logic), which allows us to deal with complex behavioral patterns that biological processes exhibit. We propose an algorithm for parameter synthesis w.r.t. a property specified using the considered logic. It exploits reachable set computations and forward refinements. We instantiate our algorithm in the case of polynomial dynamical systems exploiting Bernstein coefficients and we illustrate it on an epidemic model.

Keywords: Parameter synthesis · STL · Biological systems · Reachability

1 Introduction

Temporal logic [1] is a formalism used to specify and reason on properties that involve time. It is typically adopted in the context of formal verification, where a temporal logic formula specifies the acceptable behaviors of a system and an algorithm is used to check whether all the behaviors of the system satisfy the formula. Such a procedure is commonly known as model checking [2]. Recently, temporal logic has found applications outside formal verification, for instance *monitoring*. In this case, a formal model is not necessary, since the system can be treated as a black box whose observable behaviors can be monitored by evaluating the satisfaction of the desired temporal property. *Signal Temporal Logic* (STL [3, 4]) is a recently developed logic that allows specifying properties of dense-time real-valued signals. It is particularly suitable for monitoring both industrial case studies (see, e.g., [5, 6]) and biological systems (see, e.g., [7, 8]). It has also been used in the study of parametric systems, (see, e.g., [9])

This work has been partially supported by GNCS-INDAM project "Algoritmica per il model checking e la sintesi di sistemi safety-critical".

where parametric disturbance rejection properties are formalized in STL and then verified. One of its interesting aspects is its semantics. In addition to the classical semantics, where the result of the evaluation of a formula is a truth value, STL offers a quantitative semantics that gives the idea of "how robustly" a property is satisfied [10, 11].

In this work we propose an application of STL in the context of parameter synthesis for dynamical systems. More concretely, given a parametric nonlinear dynamical system and an STL property, we want to find a set of parameter values that guarantees that all the possible runs of the model satisfy the property. The parameter synthesis is an important problem, since it allows the designer to fine-tune the model so that it captures and retains only the behaviors of interest.

Dealing with nonlinear dynamical systems is not easy. If reasoning on single trajectory can be efficiently done by standard techniques, the problem of verifying sets of trajectories remains difficult despite a number of existing methods (see, e.g., [12–14]). Adding parameter synthesis to such a context makes the problem even more challenging. Indeed, in addition to computing the trajectories of a parametric system, one needs to determine sets of parameter values such that the corresponding trajectories satisfy a given specification.

STL and its monitoring algorithms have been conceived to evaluate logic formulas on single continuous signals [3, 4]. In order to adapt STL to our synthesis problem, we need to introduce a new semantics defined on sets of traces rather than on a single one. This semantical definition requires then a new algorithm, since it is not easy to adapt the existing ones. In fact, available algorithms compute the truth values of a formula in a bottom-up approach, where atomic predicates are evaluated on the full-length signal and the final result is obtained by combining the logical operators. This approach does not suit our case, since the system traces are affected by the eventual dynamical parameter restriction in order to satisfy the property. This means that we cannot know precisely the complete system evolution until the valid parameter values are determined. For this reason we propose a new algorithm that operates in a forward way where, at each step, valid parameter sets are identified and the system evolves in the next steps under the on-the-fly synthesized parameters. We defer a discussion on related work in parameter synthesis to the final section, after our approach is described.

The paper begins with the preliminaries introducing the STL logic, the new semantics, and the parameter synthesis problem. In Section 3 we describe the abstract synthesis algorithm, we discuss its correctness and computational complexity.[1] Section 4 is dedicated to the concretization of our synthesis algorithm for nonlinear discrete-time polynomial dynamical systems. To show the effectiveness of the proposed approach, we apply our algorithm on an epidemic model describing the transmission of diseases through a population. We provide some experimental results and scalability evaluations obtained from a prototype C++

[1] All the proofs can be found at
http://www-verimag.imag.fr/~dreossi/docs/papers/parasynth.pdf

tool. These results are reported in Section 5. Finally, the paper ends with related works, a summary of our results, and possible future developments.

2 Preliminaries and Problem Statement

2.1 Parametric Dynamical Systems

Let $\mathbb{R}$ denote the set of reals. We consider a discrete-time parametric dynamical system

$$\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{p}) \qquad \mathbf{x}(0) \in X^{-}, \mathbf{p} \in P, \tag{1}$$

where $\mathbf{x} \in \mathbb{R}^n$ is the vector of state variables, $\mathbf{p} \in P \subseteq \mathbb{R}^m$ is the vector of parameters, $\mathbf{f}$ is a vector of functions $f_i : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ for $i = 1, \ldots, n$. The set $X^{-} \subseteq \mathbb{R}^n$ is called *pre-initial set*, and the set P is called *parameter set*. We use $X^0 = \mathbf{f}(X^{-}, P)$ to denote the initial set of starting states at time 0. The distinction between pre-initial and initial sets is introduced to overcome a technical issue, i.e., the pre-initial set may not satisfy the specification of interest, while the initial one does.

Given $\mathbf{x} \in X^{-}$ and $\mathbf{p} \in P$, let

$$tr^K_{\mathbf{p}}(\mathbf{x}) = \langle \mathbf{x}(0) = \mathbf{f}(\mathbf{x}, \mathbf{p}), \ldots, \mathbf{x}(j-1) = \mathbf{f}^j(\mathbf{x}, \mathbf{p}), \ldots, \mathbf{x}(K) = \mathbf{f}^{K+1}(\mathbf{x}, \mathbf{p}) \rangle$$

be the trace of length $K \in \mathbb{N}$ of the system originating from $\mathbf{x}$ with parameter values $\mathbf{p}$. The set of all possible traces of the dynamical system (1) can be denoted as $Tr^K_{\mathbf{p}}(X^{-}) = \{tr^K_{\mathbf{p}}(\mathbf{x}) \mid \mathbf{x} \in X^{-}\}$ and $Tr^K_P(X^{-}) = \{Tr^K_{\mathbf{p}}(X^{-}) \mid \mathbf{p} \in P\}$.

2.2 Logic

Let $\mathbb{B} = \{true, false\}$ be the set of Boolean values and $\Sigma = \{\sigma_1, \ldots, \sigma_k\}$ be a finite set of predicates mapping $\mathbb{R}^n$ to $\mathbb{B}$. For a given $j \in \{1, \ldots, k\}$, the predicate σ_j is of the form $\sigma_j \equiv s_j(x_1, \ldots, x_n) \sim 0$ where $\sim \in \{<, \geq\}$ and $s_j : \mathbb{R}^n \to \mathbb{R}$ is a function over the state variables.

We consider *Signal Temporal Logic* (STL) [3, 15] formulas in positive normal form, *i.e.*, formulas generated through the following grammar:

$$\varphi := \sigma \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi \, \mathcal{U}_{\mathcal{I}} \, \varphi \tag{2}$$

where $\sigma \in \Sigma$ and $\mathcal{I} \subset \mathbb{N}$ denotes the interval $\mathcal{I} = [a, b]$ with $a \leq b$ in $\mathbb{N}$. For $t \in \mathbb{N}$, $t + \mathcal{I}$ is the set $\{t + t' \mid t' \in \mathcal{I}\}$. We can define in the usual way other common operators, such as $\top, \bot, \mathcal{R}_{\mathcal{I}}, \Diamond_{\mathcal{I}}, \Box_{\mathcal{I}}$. Note that the negation operator is not included in the presented grammar. However, a given STL formula including some negations, can be rewritten in positive normal form by pushing the negations down to the predicates and reversing their inequalities. Finally, the horizon $h(\varphi)$ of a formula φ is the last time instant to which φ refers, i.e.:

$$h(\sigma) = 0 \qquad h(\varphi_1 \wedge \varphi_2) = h(\varphi_1 \vee \varphi_2) = max(h(\varphi_1), h(\varphi_2))$$
$$h(\varphi_1 \mathcal{U}_{[a,b]} \varphi_2) = max(h(\varphi_1) + b - 1, h(\varphi_2) + b).$$

Given $\mathbf{x} \in X^-$ and $\mathbf{p} \in P$ we can consider the standard Boolean semantics of STL formulas over the trace $tr_{\mathbf{p}}^K(\mathbf{x})$. Let φ be a formula such that $t + h(\varphi) \leq K$, we define:

$$\begin{array}{rl}
tr_{\mathbf{p}}^K(\mathbf{x}), t \models \sigma & \text{iff } \sigma(\mathbf{x}(t)) \text{ is true} \\
tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_1 \wedge \varphi_2 & \text{iff } tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_1 \text{ and } tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_2 \\
tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_1 \vee \varphi_2 & \text{iff } tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_1 \text{ or } tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_2 \\
tr_{\mathbf{p}}^K(\mathbf{x}), t \models \varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2 & \text{iff } \exists t' \in t + \mathcal{I}\ tr_{\mathbf{p}}^K(\mathbf{x}), t' \models \varphi_2 \text{ and } \forall t'' \in [t, t')\ tr_{\mathbf{p}}^K(\mathbf{x}), t'' \models \varphi_1
\end{array}$$

We use the notation $tr_{\mathbf{p}}^K(\mathbf{x}) \models \varphi$ for $tr_{\mathbf{p}}^K(\mathbf{x}), 0 \models \varphi$. We say that $Tr_P^K(X^-)$ *satisfies* a formula φ, denoted as $Tr_P^K(X^-) \models \varphi$ if and only if

$$\forall \mathbf{p} \in P\ \forall tr_{\mathbf{p}}^K(\mathbf{x}) \in Tr_{\mathbf{p}}^K(X^-) : tr_{\mathbf{p}}^K(\mathbf{x}) \models \varphi.$$

Note that we consider discrete-time systems over a finite horizon; our formulas can thus be encoded in LTL formulas involving Boolean and next operators interpreted over finite traces [16]. However, STL offers us some advantages. First, to express the bounded until operator using LTL, the corresponding formula may be long. Furthermore, STL has both a quantitative discrete-time and a continuous-time semantics. For some classes of systems, the quantitative analysis on a time-discretized system gives complete information also on its continuous-time version [10].

2.3 Parameter Synthesis Problem

Our parameter synthesis problem can now be stated as follows. Let X^- be a pre-initial set, P be a parameter set, and φ a logical formula, find the largest subset $P_\varphi \subseteq P$ such that starting from X^-, the behaviors of the system satisfy φ up to time K, that is $Tr_{P_\varphi}^K(X^-) \models \varphi$.

The above problem requires handling sets of parametric traces, which is hard especially when the solution of the dynamical system can only be approximated, as in the case of polynomial systems that we will specifically treat later. Therefore, we consider a variant of this problem for approximated sets of traces. The set $Tr_P^K(X^-)$ of traces can be over-approximated by considering the sets $X^{j+1} = \{\mathbf{f}(\mathbf{x}, \mathbf{p}) \mid \mathbf{x} \in X^j, \mathbf{p} \in P\}$, for $j = 0, \ldots, K$, where X^- acts as X^{-1}, and the *behaviors* defined as $\mathcal{W}_P^K(X^-) = \langle X^0, X^1, \ldots, X^K \rangle$. It is important to note that this over-approximation does not keep the relation between a trace and its corresponding parameter value.

On behaviors we can define a semantics that reflects the parameter refinement problem we are interested in. In particular, we consider the function $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), t)$ defined by structural induction on formulas as follows:

$$\begin{aligned}
\mathcal{X}(\sigma, \mathcal{W}_P^K(X^-), t) &= P_\sigma^t, \text{ where } P_\sigma^t \text{ is the largest subset } P_\sigma^t \subseteq P \text{ such that} \\
&\qquad \forall \mathbf{x} \in X^{t-1}, \forall \mathbf{p} \in P_\sigma^t, \sigma(\mathbf{f}(\mathbf{x}, \mathbf{p})) \text{ is true} \\
\mathcal{X}(\varphi_1 \wedge \varphi_2, \mathcal{W}_P^K(X^-), t) &= \mathcal{X}(\varphi_1, \mathcal{W}_P^K(X^-), t) \cap \mathcal{X}(\varphi_2, \mathcal{W}_P^K(X^-), t) \\
\mathcal{X}(\varphi_1 \vee \varphi_2, \mathcal{W}_P^K(X^-), t) &= \mathcal{X}(\varphi_1, \mathcal{W}_P^K(X^-), t) \cup \mathcal{X}(\varphi_2, \mathcal{W}_P^K(X^-), t) \\
\mathcal{X}(\varphi_1 \mathcal{U}_\mathcal{I} \varphi_2, \mathcal{W}_P^K(X^-), t) &= \bigcup_{t' \in t+\mathcal{I}} (\mathcal{X}(\varphi_2, \mathcal{W}_P^K(X^-), t') \cap \bigcap_{t'' \in [t,t')} \mathcal{X}(\varphi_1, \mathcal{W}_P^K(X^-), t''))
\end{aligned}$$

Intuitively, $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), t)$ returns a subset P_φ^t of parameters that ensures that φ is satisfied at time t starting from any point in X^{t-1} and assigning to the parameters any value in P_φ^t. We say that a behavior $\mathcal{W}_P^K(X^-)$ *satisfies* a formula φ, denoted with $\mathcal{W}_P^K(X^-) \models \varphi$, if and only if $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0) = P$.

We can prove that $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), t)$ reverse the order of inclusions between parameter sets and it is idempotent. This implies that the refined set of parameter values satisfies the formula φ, as stated by the following theorem.

Theorem 1. *If* $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0) = P_\varphi$*, then* $\mathcal{W}_{P_\varphi}^K(X^-) \models \varphi$.

The following results establish relationships between the two semantics. Since $\mathcal{W}_P^K(X^-)$ over-approximates $Tr_P^K(X^-)$, if a formula φ is satisfied by $\mathcal{W}_P^K(X^-)$, then it is satisfied also by $Tr_P^K(X^-)$.

Theorem 2. *If* $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0) = P_\varphi$*, then* $Tr_{P_\varphi}^K(X^-) \models \varphi$.

If we compute $P_\varphi = \mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0)$ then we have an under-approximation of the solution of the original parameter synthesis problem. Moreover, P_φ is nothing but the solution of a parameter synthesis problem over behaviors. In this approach we introduce three main sources of approximation error.

First, in the semantics of disjunction over behaviors we impose that for a value $\mathbf{p}$ either all the points $\mathbf{x}$ satisfy the first disjunct or they all satisfy the second one. In a more general setting, this would correspond to approximating a property of the form $\forall y(A(y) \vee B(y))$ with $\forall y(A(y)) \vee \forall y(B(y))$.

Second, $\mathcal{W}_P^K(X^-)$ represents a set of traces that is larger than $Tr_P^K(X^-)$, since from each point in X^j we can reach each point in X^{j+1}. This influences the semantics of the until operator. In fact, we need to require that there exists a time point t' at which φ_2 is satisfied from all the points.

Third, $\mathcal{W}_P^K(X^-)$ is a-priori computed using P, so we also propagate points of X^j that are not necessarily reachable if we replace P with a proper subset.

In the next sections we present an algorithm that computes an under-approximation of the solution to the original parameter synthesis problem. Our algorithm is inspired by $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0)$, but it produces a better approximation since the parameter set is dynamically refined and the above-mentioned third source of approximation is avoided as at each step only the refined parameter set is used to determine the next states.

3 Parameter Synthesis Algorithm

An intuitive way to solve our synthesis problem is to express the behavior set X^j at each time instant j, up to time K. Then by examining the sets from time K back to time 0 we can derive the conditions on the parameters for the satisfaction of the temporal property, as in the standard monitoring approaches [3]. In other terms, we could use such backward analysis for the semantics defined by (2.3). However, while in monitoring only a single trace is considered at a time and furthermore the trace is already given, in our parameter synthesis problem the behavior set needs to be approximated (since exact reachability computation for nonlinear systems is often impossible). When approximations are used, a major drawback of such a backward procedure is that the approximation error depends on the size of the parameter set and is accumulated step after step. The more spurious behaviors are included in the computed set, the more restricted the parameter set is. In order to gain more accuracy, it is thus important to be able to remove, as early as possible, the parameter values that make the system violate the property. This is the reason we opt for a forward procedure.

We describe our top-down algorithm ParaSynth(X, P, φ) (Algorithm 1), that takes as input a set of states X, a set of parameters P, and a formula φ, and refines P through a series of recursions driven by the structure of φ. At each step, we let the system evolve under the parameter set synthesized up to that step. It is structured in four main blocks, one for each type of subformulæ: predicate, conjunction, disjunction, and until. It uses the following two basic functions. Given a set X, a parameter set P and a predicate σ,

- ReachStep(X, P) computes the image $\mathbf{f}(X, P)$ of X under P;
- RefPredicate(X, P, σ) computes the largest subset $P_\sigma \subseteq P$ such that all states in $\mathbf{f}(X, P_\sigma)$ (computed by ReachStep) satisfy σ, that is $P_\sigma = \{\mathbf{p} \mid \mathbf{p} \in P \ \wedge \ \forall \mathbf{x} \in X \, \sigma(\mathbf{f}(\mathbf{x}, \mathbf{p})) = \mathit{true}\}$. We call the computation of RefPredicate a basic refinement, since it is a refinement of the parameter set w.r.t. a predicate.

The base case is when the formula φ is a predicate σ (Line 2). In this case, the algorithm simply calls the function RefPredicate(X, P, σ) that refines the parameter set P w.r.t. the predicate σ and returns the result.

If φ is the conjunction of two formulas $\varphi_1 \wedge \varphi_2$ (Line 5), from P the algorithm, with two recursive calls, produces two refined parameter sets P_{φ_1} and P_{φ_2}, w.r.t. the subformulas φ_1 and φ_2, respectively, and then returns the intersection $P_{\varphi_1} \cap P_{\varphi_2}$. Similarly, if φ is a disjunction $\varphi_1 \vee \varphi_2$ (Line 8), the algorithm returns the union $P_{\varphi_1} \cup P_{\varphi_2}$.

The case where φ is $\varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2$ (Line 11) is slightly more complicated and requires a specific function UntilSynth (Algorithm 2). The function UntilSynth$(X, P, \varphi_1 \mathcal{U}_{[a,b]} \varphi_2)$ is structured in three main blocks, depending on the values a, b: (1) $a > 0$ and $b > 0$; (2) $a = 0$ and $b > 0$; (3) $a = 0$ and $b = 0$. Intuitively, the function recursively transforms the cases (1) and (2) into the base case (3). Notice that a single until formula $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ may require several basic refinements. Consider for instance the case where φ_1 always holds and φ_2

Algorithm 1 Parameter synthesis.

```
function ParaSynth($X, P, \varphi$)
    if $\varphi = \sigma$ then                                        ▷ Predicate
        return RefPredicate($X, P, \sigma$)
    end if
    if $\varphi = \varphi_1 \wedge \varphi_2$ then                    ▷ Conjunction
        return ParaSynth($X, P, \varphi_1$) $\cap$ ParaSynth($X, P, \varphi_2$)
    end if
    if $\varphi = \varphi_1 \vee \varphi_2$ then                      ▷ Disjunction
        return ParaSynth($X, P, \varphi_1$) $\cup$ ParaSynth($X, P, \varphi_2$)
    end if
    if $\varphi = \varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2$ then  ▷ Until
        return UntilSynth($X, P, \varphi_1 \mathcal{U}_{\mathcal{I}} \varphi_2$)
    end if
end function
```

Algorithm 2 Until synthesis.

```
function UntilSynth($X, P, \varphi_1 \mathcal{U}_{[a,b]} \varphi_2$)
    if $a > 0$ and $b > 0$ then                                       ▷ Outside interval
        $P_{\varphi_1} \leftarrow$ ParaSynth($X, P, \varphi_1$)       ▷ Check $\varphi_1$
        if $P_{\varphi_1} = \emptyset$ then
            return $\emptyset$
        else
            $X' \leftarrow$ ReachStep($X, P_{\varphi_1}$)
            return UntilSynth($X', P_{\varphi_1}, \varphi_1 \mathcal{U}_{[a-1,b-1]} \varphi_2$)
        end if
    end if
    if $a = 0$ and $b > 0$ then                                       ▷ In interval
        $P_{\varphi_1} \leftarrow$ ParaSynth($X, P, \varphi_1$)       ▷ Check $\varphi_1$
        $P_{\varphi_2} \leftarrow$ ParaSynth($X, P, \varphi_2$)       ▷ Check $\varphi_2$
        if $P_{\varphi_1} = \emptyset$ then
            return $P_{\varphi_2}$                                    ▷ Until unsatisfied
        else
            $X' \leftarrow$ ReachStep($X, P_{\varphi_1}$)
            return $P_{\varphi_2} \cup$ UntilSynth($X', P_{\varphi_1}, \varphi_1 \mathcal{U}_{[a,b-1]} \varphi_2$)
        end if
    end if
    if $a = 0$ and $b = 0$ then                                       ▷ Base
        return ParaSynth($X, P, \varphi_2$)
    end if
end function
```

holds at several time points inside $[a, b]$. Here the number of basic refinements that $\varphi_1 \mathcal{U}_{[a,b]} \varphi_2$ requires is exactly the number of time points at which φ_2 holds. We now analyze the three cases in UntilSynth reported:

(1) $a > 0$ and $b > 0$: the until formula is satisfied if φ_1 holds until φ_2 is true inside the interval $[a, b]$. We first refine the parameters at time 0 over φ_1,

obtaining the subset P_{φ_1} (Line 3). If P_{φ_1} is empty, the until formula cannot be satisfied, and the algorithm returns the empty set. If P_{φ_1} is not empty, the algorithm performs a reachability step using the valid parameter set P_{φ_1} to produce the new set X' (Line 7). Now the algorithm proceeds with the recursive call $\text{UntilSynth}(X', P_{\varphi_1}, \varphi_1 \mathcal{U}_{[a,b]-1} \varphi_2)$ (Line 8). This can be seen as making a step towards the interval $[a, b]$, except that instead of restoring the synthesis from time 1, we shift the interval backwards by 1. Hence, the next refinement is computed always at time 0.

(2) $a = 0$ and $b > 0$: there are two ways to satisfy the until formula: (1) φ_2 is satisfied right now at time 0, or (2) φ_1 holds until φ_2 is satisfied before the time instant b. In the first case, we need to refine the parameter set w.r.t. φ_2. If the resulting P_{φ_2} is not empty, it is a valid parameter set that satisfies the until formula. In the second case, the algorithm refines w.r.t. φ_1 and checks whether the result P_{φ_1} is empty (Line 12). If so, the until formula cannot be satisfied in the future. Hence the algorithm returns the refined set P_{φ_2} previously computed. If P_{φ_1} is not empty, the procedure performs a reachability step under the refined parameters P_{φ_1}, obtaining the new set X'. Similarly to the previous case, we execute a step forward by shortening the interval by one (Line 18). The procedure then returns the union of P_{φ_2} and the result provided by the recursive call;

(3) $a = 0$ and $b = 0$: this is the base case of the recursive calls. It suffices to refine w.r.t. φ_2 and return P_{φ_2}.

Example 1. We illustrate ParaSynth in the case $\phi = (\phi_1 \vee \phi_2)\mathcal{U}_{[1,2]}(\phi_3 \wedge \phi_4)$.

With the call $\text{ParaSynth}(X^-, P, (\phi_1 \vee \phi_2)\mathcal{U}_{[1,2]}(\phi_3 \wedge \phi_4))$ the algorithm enters the until section and calls UntilSynth. The first synthesis is performed inside the ($a > 0$ and $b > 0$) case w.r.t. to the sub-formula $\phi_1 \vee \phi_2$. ParaSynth computes the refined sets $P^0_{\phi_1}$ and $P^0_{\phi_2}$ w.r.t. ϕ_1 and ϕ_2, and returns the union $P^0_{\phi_1 \vee \phi_2} = P^0_{\phi_1} \cup P^0_{\phi_2}$. Back to the until synthesis, supposing that $P^0_{\phi_1 \vee \phi_2}$ is not empty, the algorithm computes X^0 through a reachability step from X^- under the parameter set $P^0_{\phi_1 \vee \phi_2}$, and calling itself with the updated reachability set, the refined parameter set, and the shifted until interval, i.e., $\text{UntilSynth}(X^0, P^0_{\phi_1 \vee \phi_2}, (\phi_1 \vee \phi_2)\mathcal{U}_{[0,1]}(\phi_3 \wedge \phi_4))$.

At this point UntilSynth enters the ($a = 0$ and $b > 0$) section. It first refines w.r.t. $(\phi_3 \wedge \phi_4)$, trying to find the first final solution. To do so, it calls $\text{ParaSynth}(X^0, P^0_{\phi_1 \vee \phi_2}, \phi_3 \wedge \phi_4)$ that produces the parameter set $P^1_{\phi_3 \wedge \phi_4} = P^1_{\phi_3} \cap P^1_{\phi_4}$, result of the intersection of the two refinements of $P^0_{\phi_1 \vee \phi_2}$ w.r.t. ϕ_3 and ϕ_4. This set $P^1_{\phi_3 \wedge \phi_4}$, if not empty, represents the first valid parameter set.

Trying to find other possible solutions, the algorithm proceeds by computing the parameter set $P^1_{\phi_1 \vee \phi_2}$ through the refinement of $P^0_{\phi_1 \vee \phi_2}$ w.r.t. $\phi_1 \vee \phi_2$ and performing a reachability step to the new set $X^1_{\prime\prime}$. It then calls itself reducing the until interval to $[0, 0]$. This is the base case ($a = 0$ and $b = 0$): the algorithm refines w.r.t. $\phi_3 \wedge \phi_4$ and returns the refined parameter set $P^2_{\phi_3 \wedge \phi_4}$. The synthesis process is shown in Figure 1a. The figure depicts the series of refinements and reachable sets that lead to the final result $P^2_{\phi_3 \wedge \phi_4} \cup P^1_{\phi_3 \wedge \phi_4}$.

In the following, we prove the correctness of the presented algorithm and determine its computational complexity.

Theorem 3. *If* ParaSynth *(X^-, P, φ) returns P_φ, then $\mathcal{X}(\varphi, \mathcal{W}_P^K(X^-), 0) \subseteq P_\varphi$ and $Tr_{P_\varphi}^K(X^-) \models \varphi$.*

We remark that the above theorem has been proved under the assumption that the function ReachStep(X, P) computes exactly the image $\mathbf{f}(X, P)$ and the function RefPredicate(X, P, σ) the largest valid parameter set for the predicate σ. However, it is not hard to see that, for Theorem 3 to hold, it suffices to provide an over-approximation of the image $\mathbf{f}(X, P)$ and an under-approximation of the valid parameter set.

As far as the computational complexity of our algorithm is concerned, let us refer to RefPredicate, ReachStep, $\cup$, and $\cap$ as symbolic operations. If we have a formula without until operators our procedure performs a number of symbolic operations that is linear in the length of the formula. In the case of formulas with possibly nested until operators in the worst case we could perform an exponential number of symbolic operations w.r.t. the minimum between the length of the formula and its time horizon. Let us consider the case of formulas using only predicates and $\mathcal{U}_{[0,1]}$ operators. Let the length of a formulas φ be defined as the maximum number of nested until operators. For a formula $\varphi_1 \mathcal{U}_{[0,1]} \varphi_2$ having length m and horizon k our recursive procedure has a recursive complexity equation in term of symbolic operations of the form

$$T(m,k) = \begin{cases} \Theta(1) & \text{if } m = 1 \text{ or } k = 0 \\ T(m_2,k_2) + T(m_1,k_1) + T(m,k-1) + \Theta(1) & \text{otherwise} \end{cases}$$

where m_i and k_i are the length and horizon of φ_i. In the worst case we could have $m_2 = m - 2$ and $k_2 = k - 1$. In this case we obtain $T(m,k) \geq 2T(m-2, k-1) + \Theta(1)$, which tells us that in the worst case $T(m,k) = \Omega(2^{min(m,k)})$ (number of symbolic operations). If we were interested in monitoring a formula over a finite set of traces, we could have reduced such complexity to a polynomial one (see, e.g., [3, 15]). As we already pointed out, since we are interested in refining sets of parameters and we do not use a precomputed set of traces to avoid rough approximations, we do not see an easy way to reduce such complexity. Finally, it is important to notice that the worst case complexity occurs only in very pathological cases, which are not typical in real case studies.

4 The Case of Polynomial Systems

In Section 3 we presented an abstract algorithm that synthesizes a parameter set under which the behavior of a system satisfies a given formula. A concrete application depends on the ability to represent the parameter set, to implement the function ReachStep for computing the behavior set, and the function RefPredicate for refining the parameter set. We now propose a concretization of the algorithm for nonlinear polynomial dynamical systems with polytopic parameter sets. The implementation that we present extends the synthesis procedures developed in our previous works for safety specifications [17, 18]. Notice that the abstract algorithm exposed in Section 3 can be used for more general systems as long as an implementation of the required procedures are provided. An example might be continuous-time piecewise-linear dynamical systems, for which reachability techniques and tools have been developed [19, 20].

From now on, we work with polynomial discrete-time dynamical systems of the form $\mathbf{x}(k+1) = \mathbf{f}(\mathbf{x}(k), \mathbf{p})$, where $\mathbf{x} \in \mathbb{R}^n$ and $\mathbf{p} \in P$. The parameter set P is a bounded polytope in $\mathbb{R}^m$ and the polynomial function $\mathbf{f}$ is linear in $\mathbf{p}$.

To compute the functions $\textsc{RefPredicate}(X, P, \sigma)$ and $\textsc{ReachStep}(X, P)$, we extend our reachability computation method based on the Bernstein form of polynomials proposed in [17, 18] which we briefly recall in the following.

The sets of states are represented with template parallelotopes, that is the n-dimensional generalization of the parallelograms. In order to exploit linear programming, we require the predicates σ to be non-strict linear inequalities over the state variables $\mathbf{x}$[2]. In [17, 18] we developed a technique that, under these assumptions, converts the parameter synthesis problem into a linear program. In particular, if the predicate σ is of the from $s(\mathbf{x}) \leq 0$ and the parameter set $P \subset \mathbb{R}^m$ is a convex polytope, to find a subset $P_s \subseteq P$ such that the image $\mathbf{f}(\mathbf{x}, \mathbf{p})$ satisfies $\sigma(\mathbf{x})$ for all $\mathbf{x} \in X$ and $\mathbf{p} \in P_s$, it suffices to require $s(\mathbf{f}(\mathbf{x}, \mathbf{p})) \leq 0$ to hold for all $\mathbf{x} \in X$ and $\mathbf{p} \in P_s$. We call P_s the valid parameter set w.r.t. the predicate σ. Note that $s(\mathbf{f}(\mathbf{x}, \mathbf{p}))$ is a polynomial in $\mathbf{x}$ and is linear in $\mathbf{p}$. To find P_s, we take advantage of the geometric properties of the coefficients $\mathbf{b}_1(\mathbf{p}), \ldots, \mathbf{b}_n(\mathbf{p})$ of the polynomial $s(\mathbf{f}(\mathbf{x}, \mathbf{p}))$ expressed in Bernstein form. Intuitively, these coefficients provide an upper and a lower bound of the considered polynomial. Thus, if we can restrict the parameter set P such each control point is smaller than 0, then the resulting restricted set P_s contains all the valid parameter values w.r.t. the predicate σ. The behavior set can also be over-approximated by composing the template constraints of the parallelotope with $\mathbf{f}$ and bound the resulting function by exploiting its Bernstein coefficients. Since the parameters $\mathbf{p}$ appear linearly in the dynamics, the coefficients $\mathbf{b}_1(\mathbf{p}), \ldots, \mathbf{b}_n(\mathbf{p})$ are linear functions of $\mathbf{p}$. Hence, the valid parameter set can be determined by solving a linear system where all the coefficients are constrained to be non-positive.

4.1 Parameter Set Representation

A convex polytope is the simplest form that we use to represent a parameter set. With the notation $P \equiv A\mathbf{p} \leq b$ we mean that the parameter set P corresponds to the solution of the linear system $A\mathbf{p} \leq b$. More complex parameter sets can be obtained by the intersection and the union of several basic convex polytopes.

Let $P_1 \equiv A_1\mathbf{p} \leq b_1$ and $P_2 \equiv A_2\mathbf{p} \leq b_2$ be two convex polytopes. It is not difficult to see that the intersection $P_1 \cap P_2$ is the convex polytope that corresponds to $P_1 \cap P_2 \equiv A\mathbf{p} \leq b$, where $A = \left[\begin{smallmatrix} A_1 \\ A_2 \end{smallmatrix}\right]$ and $b = \left[\begin{smallmatrix} b_1 \\ b_2 \end{smallmatrix}\right]$. Less trivial is the union of two convex polytopes since it might not be convex and consequently the

[2] Note that using only non strict inequalities, truth values $\top$ and $\bot$ can be abbreviated as $\top := z(\mathbf{x}) \leq 0$ and $\bot := o(\mathbf{x}) \leq 0$, where $z(\mathbf{x}) = 0$ and $o(\mathbf{x}) = 1$ for all $\mathbf{x} \in \mathbb{R}^n$.

representation through a linear system may not be possible. For this reason we symbolically represent the union of two polytopes P_1 and P_2 by simply keeping the list of the corresponding linear systems. Formally, with an abuse of notation, $P_1 \cup P_2$ is represented as $P_1 \cup P_2 \equiv \{A_1\mathbf{p} \leq b_1, A_2\mathbf{p} \leq b_2\}$.

If a parameter set P is in the form $P = \bigcup_{i=1}^{n} \bigcap_{j=1}^{m_i} P_{i,j} = (P_{1,1} \cap \ldots \cap P_{1,m_1}) \cup \ldots \cup (P_{n,1} \cap \ldots \cap P_{n,m_n})$ then it is said to be in *union normal form*. This form is suitable for our set representation since the intersections of sets can be collapsed in a unique linear system while the unions can be stored in single list.

4.2 Parameter Synthesis

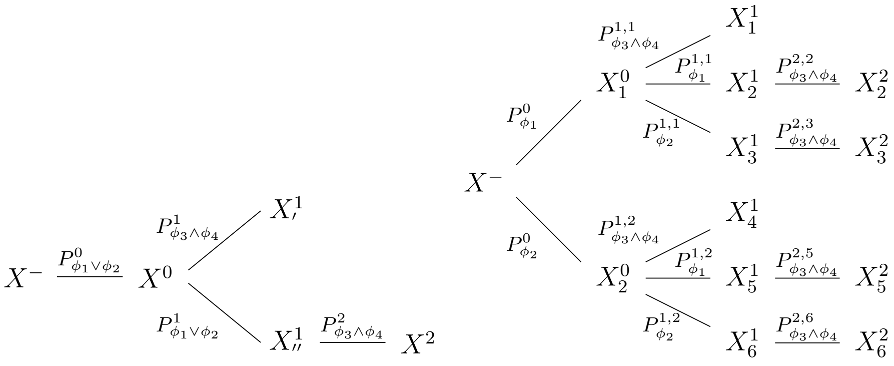
We now discuss the implementation using the above represention of parameter sets and the behavior computation based on the Bernstein form.

The refinement RefPredicate(X, P, σ) where $P \equiv \{P_1, \ldots, P_n\}$ (Algorithm 1, Line 2) can be done by refining each polytope P_i w.r.t. σ (using the procedure exposed in [17]). Each function returns a set $P_{i,\sigma} \subseteq P$. The final result $P_\sigma \subseteq P$ is the union of the basic polytopes, represented as $P_\sigma \equiv \{P_{1,\sigma}, \ldots, P_{n,\sigma}\}$.

The conjunction and disjunction cases (Lines 5 and 8) involve the intersection and union of P_{φ_1} and P_{φ_1} provided in union normal form. The intersection can be obtained by intersecting each basic polytope of P_{φ_1} with each basic polytope of P_{φ_2}. This operation can be carried out by just merging the linear systems representing the considered basic polytopes. The union can be easily obtained by concatenating the lists of basic polytopes that compose P_{φ_1} and P_{φ_2}.

We now focus on Algorithm 2, in particular on the calls of the functions ReachStep and UntilSynth (Lines 7 and 13). Here the main issue concerns the computation of ReachStep(X, P) whose result can be non-convex. To this end, we open several branches, one for each basic convex polytope of P. Hence, instead of computing a single behavior set, we split the computation in several reachability steps, one for each basic parameter set and refine the parameters from them w.r.t. the considered sub-formula.

Example 2. Let us consider the formula $(\phi_1 \vee \phi_2)\mathcal{U}_{[1,2]}(\phi_3 \wedge \phi_4)$ of Example 1.The algorithm starts with ParaSynth $(X^-, P, (\phi_1 \vee \phi_2)\mathcal{U}_{[1,2]}(\phi_3 \wedge \phi_4))$ that invokes UntilSynth that enters in the case ($a > 0$ and $b > 0$) and performs the first refinement of P w.r.t. the sub-formula $\phi_1 \vee \phi_2$ by calling ParaSynth. This function synthesizes P by refining w.r.t. both the predicates ϕ_1 and ϕ_2 and merging the partial results. The result is the set $P^0_{\phi_1 \vee \phi_2} \equiv \{P^0_{\phi_1}, P^0_{\phi_2}\}$. At this point, UntilSynth opens a branch from X^- for each computed refinement. We denote by X^0_1 the set reached from X^- under $P^0_{\phi_1}$ and by X^0_2 the set reached from X^- under $P^0_{\phi_2}$. UntilSynth proceeds with two recursive calls, UntilSynth$(X^0_1, P^0_{\phi_1}, (\phi_1 \vee \phi_2)\mathcal{U}_{[0,1]}(\phi_3 \wedge \phi_4))$ and UntilSynth$(X^0_2, P^0_{\phi_2}, (\phi_1 \vee \phi_2)\mathcal{U}_{[0,1]}(\phi_3 \wedge \phi_4))$. We now consider the first recursive call. In this phase, UntilSynth is in the case ($a = 0$ and $b > 0$). First, trying to satisfy the whole until, the algorithm refines the set $P^0_{\phi_1}$ w.r.t. ϕ_3 and ϕ_4. This is done by calling ParaSynth $(X^0, P^0_{\phi_1}, \phi_3 \wedge \phi_4)$. We denote the result with $P^{1,1}_{\phi_3 \wedge \phi_4} = P^1_{\phi_3} \cap P^1_{\phi_4}$. If not empty, $P^{1,1}_{\phi_3 \wedge \phi_4}$ is the first valid parameter set. Trying to find other solutions, UntilSynth refines also w.r.t. $\phi_1 \vee \phi_2$ opening two new branches, one for each disjunct. Each branch corresponds to a recursive call of the form ParaSynth $(X^1_2, P^{1,1}_{\phi_1}, (\phi_1 \vee \phi_2)\mathcal{U}_{[0,0]}(\phi_3 \wedge \phi_4))$. The synthesis process is shown in Figure 1b.


X^-
$P^0_{\phi_1 \vee \phi_2}$
X^0
$P^1_{\phi_3 \wedge \phi_4}$
$X^1_{\prime}$
$P^1_{\phi_1 \vee \phi_2}$
$X^1_{\prime\prime}$
$P^2_{\phi_3 \wedge \phi_4}$
X^2
X^-
$P^0_{\phi_1}$
X^0_1
$P^{1,1}_{\phi_3 \wedge \phi_4}$
X^1_1
$P^{1,1}_{\phi_1}$
X^1_2
$P^{2,2}_{\phi_3 \wedge \phi_4}$
X^2_2
$P^{1,1}_{\phi_2}$
X^1_3
$P^{2,3}_{\phi_3 \wedge \phi_4}$
X^2_3
$P^0_{\phi_2}$
X^0_2
$P^{1,2}_{\phi_3 \wedge \phi_4}$
X^1_4
$P^{1,2}_{\phi_1}$
X^1_5
$P^{2,5}_{\phi_3 \wedge \phi_4}$
X^2_5
$P^{1,2}_{\phi_2}$
X^1_6
$P^{2,6}_{\phi_3 \wedge \phi_4}$
X^2_6

(a) General case. **(b)** Polynomial system case.

Fig. 1. Parameter synthesis sequences

5 Experimental Results

We implemented a prototype tool[3] written in C++ that exploits the GiNaC library [21] to symbolically manipulate polynomials and GLPK (GNU Linear Programming Kit)[4] to solve linear programs. The experiments have been carried out on an Intel Core(TM)2 Duo (2.40 GHz, 4GB RAM) running Ubuntu 12.04.

In this final section we first apply our technique on a model of diseases transmission. The model is a variation of the system that describes the Ebola outbreak in Congo 1995 and Uganda 2000 presented in [22]. A population composed of N individuals, is classified in five compartments S, E, Q, I, and R. Each individual, at a certain time, belongs to a specific compartment accordingly with his/her relationship with the disease. All the individual displacements between compartments are regulated by the parameters $\beta, \kappa_1, \kappa_2, \gamma_1, \gamma_2$, and σ.

S contains the healthy individuals that are *susceptible* to the disease. A member of S who enters in contact with a sick person, moves to E, that is the class of individuals who have been *exposed* to the disease. The ratio I/N is the probability that a susceptible individual enters in contact with an infected one, while β is the transmission rate. An exposed individual is either moved in *quarantine* in Q, or directly in the *infected* I compartment, depending on whether the

[3] Available on line at https://github.com/tommasodreossi/parasynth
[4] http://www.gnu.org/software/glpk/glpk.html

malady was diagnosed. The controllable quarantine rate is κ_1, while $1/\kappa_2$ is the mean incubation period. A person in quarantine, if considered healthy after the isolation period, is moved back to the susceptible group. The unfortunate case is when the individual manifests symptoms and moves from the quarantine to the infected group. The reintegration with the susceptible people happens after a period of $1/\gamma_1$, while the incubation period is $1/\gamma_2$. Finally an individual is *removed* from the system by migrating in R at a recovering or death rate σ. The epidemic model is formalized through the following system:

$$\begin{aligned}
S_{n+1} &= S_n - S_n \beta I_n/N + \gamma_1 Q_n \\
E_{n+1} &= E_n + S_n \beta I_n/N - (\kappa_1 + \kappa_2) E_n \\
Q_{n+1} &= Q_n + \kappa_1 E_n - (\gamma_1 + \gamma_2) Q_n \\
I_{n+1} &= I_n + \gamma_2 Q_n + \kappa_2 E_n - \sigma I_n \\
R_{n+1} &= R_n + \sigma I_n
\end{aligned}$$

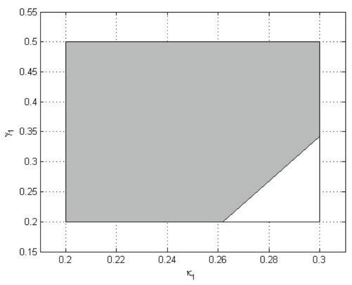
$$\begin{tikzcd}
 & & Q \arrow[lld, "\gamma_1"'] \arrow[dd, "\gamma_2"] & \\
S \arrow[r, "\beta I/N"'] & E \arrow[ru, "\kappa_1"] \arrow[rd, "\kappa_2"'] & & \\
 & & I \arrow[r, "\sigma"'] & R
\end{tikzcd}$$

The difference between our model and the one presented in [22] is that we introduce the quarantine compartment and consider the reintegration of individual in the susceptible population. Doing so, we enrich the original model by making it more realistic and interesting. Also, our model is defined on discrete time. Note that in the literature there are various works presenting epidemic models directly with discrete-time dynamics or difference equations (see, e.g., [23, 24]).

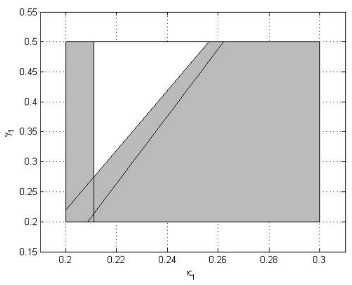
We first considered a population of $N = 1000$ individuals, of which $S = 800$ are susceptible and $I = 200$ are infected. We fixed the parameters values as specified in [22] in the case of the Ebola outbreak in Uganda during 2000. The uncontrollable parameter values are $\beta = 0.35, \kappa_2 = 0.3, \gamma_2 = 0.6$, and $\sigma = 0.28$, while the controllable parameters are $\kappa_1 \in [0.2, 0.3]$ and $\gamma_1 \in [0.2, 0.5]$ that represent the quarantine rate and mean isolation period, respectively. We considered the specification $\phi_1 \equiv (I(t) \leq 200)\mathcal{U}_{[6,10]}(Q(t) \leq 20)$ whose meaning is to avoid the saturation of the quarantine compartment especially in the time interval between 6 and 10 when a number of infected individuals higher than 200 is expected. Our tool found five feasible parameters sets in 0.10 seconds, one of which is shown in Figure 2a.

In a second experiment, we changed the uncontrollable parameter values to $\beta = 0.9$, $\kappa_2 = 0.5$, $\gamma_2 = 0.5$, and $\sigma = 0.28$, while the controllable parameters to $\kappa_1 \in [0.2, 0.3]$ and $\gamma_1 \in [0.2, 0.5]$. Instead of imposing directly a constraint on the system, we could imagine a scenario where we have a maximum number of 40 patients in quarantine unless the number of infected patients is below 270. This means that if there are less than 270 infected individuals, then we have free resources that can be devoted to the quarantine. This property can be formalized with the formula $\phi_2 \equiv (Q(t) \leq 40)\mathcal{U}_{[10,15]}(I(t) \leq 270)$. Our tool found a valid parameter set in 0.14 seconds.

Finally, on the same system configuration, we tested a more complex until formula that involves a disjunction, that is $\phi_3 \equiv (Q(t) \leq 50)\mathcal{U}_{[5,15]}(E(t) > 100 \vee Q(t) > 25)$. Our tool found four parameter refinements in 0.14 seconds. Figure 2b depicts the union of the refined parameter sets.



(a) $(I(t) \leq 200)\mathcal{U}_{[6,10]}(Q(t) \leq 20)$



(b) $(Q(t) \leq 50)\mathcal{U}_{[5,15]}(E(t) > 100 \vee Q(t) > 25)$

Fig. 2. Results of the refinements

Table 1. Scalability tests. Times are in seconds. Values in parenthesis are the computed polytopes per refinement. $\phi_1 \equiv (I(t) \leq 200)\mathcal{U}_{[a,b]}(Q(t) \leq 20)$, $\phi_2 \equiv (Q(t) \leq 40)\mathcal{U}_{[a,b]}(I(t) \leq 270)$, $\phi_3 \equiv (Q(t) \leq 50)\mathcal{U}_{[a,b]}(E(t) > 100 \vee Q(t) > 25)$.

a	b	ϕ_1	ϕ_2	ϕ_3
5	15	0.20 (11)	0.15 (7)	0.14 (4)
5	20	0.35 (16)	0.24 (11)	0.21 (4)
5	30	0.82 (26)	0.55 (21)	0.36 (4)
5	50	2.63 (46)	1.65 (41)	0.80 (4)
5	100	-	10.95 (91)	1.69 (4)
5	125	-	-	1.69 (4)
15	20	0.29 (6)	0.22 (4)	0.19 (0)
20	30	0.64 (11)	0.45 (11)	0.35 (0)
30	50	1.88 (21)	1.27 (21)	0.78 (0)
50	100	13.00 (51)	6.89 (51)	1.72 (0)
100	200	-	-	1.66 (0)

(a) Increasing until interval

N	ϕ_1^N	ϕ_2^N	ϕ_3^N
1	0.11 (5)	0.14 (2)	0.13 (4)
2	0.26 (9)	0.36 (7)	0.29 (3)
3	0.48 (13)	0.69 (12)	0.50 (3)
4	0.74 (17)	1.09 (17)	0.70 (3)
5	1.10 (21)	1.61 (22)	1.01 (3)
6	1.50 (25)	2.28 (27)	1.20 (3)
7	1.97 (29)	3.05 (32)	1.65 (3)
8	2.59 (33)	3.97 (37)	1.81 (3)
9	3.23 (37)	5.06 (42)	2.16 (3)
10	4.98 (42)	6.70 (47)	2.56 (3)
15	10.33 (61)	11.80 (62)	4.85 (3)
16	11.75 (65)	15.98 (67)	5.43 (3)
17	-	-	5.97 (3)

(b) Nesting until

In order to evaluate the scalability of our method, we now consider non-trivial formulas that we artificially created. First, we take the three until formulas ϕ_1, ϕ_2, and ϕ_3 previously defined and we stretch their intervals $[a, b]$. In the worst case, such growth exponentially increases the number of branches that our algorithm must open. Table 1a reports the running times for this test. For ϕ_1 we are able to stretch the interval up to $[50, 100]$, that deals to a parameter set composed by 51 convex polytopes. For ϕ_2 the maximum tractable interval was $[5, 100]$, for which a parameter set of cardinality 91 is found. Finally, we notice that enlargement of the window of ϕ_3 does not affect the parameter refinement, that is valid refinements are found only for the initial part of the interval. Not growing in the size of the results, the algorithm needs linear time in the until time horizon. As second evaluation, we nest several until on the (most critical) right hand side. For instance, the double nesting of ϕ_1 is $\phi_1^N \equiv (I(t) \leq 200)\mathcal{U}_{[6,10]}((I(t) \leq 200)\mathcal{U}_{[6,10]}(Q(t) \leq 20))$ with $N = 2$. Table 1b reports the running times for this evaluation. Our tool computes refinements of ϕ_1 and ϕ_2 nested 16 times, finding 65 and 67 convex parameter sets. As in the previous case, nesting ϕ_3 does not increase the size of the result, thus the computation times are restrained. It is interesting to notice that even if ϕ_1 and ϕ_2 are composed by less atomic formulas than ϕ_3, their running times and size of results, are sensibly larger than the latter. This suggests that it might be hard to estimate a priori the algorithm performance just by looking at the specification, since its execution time "numerically" depends on the system's behavior and mostly on the computed partial refinements. Finally, these experiments show that the weakness of our tool is memory. So far, parameter set are represented as lists of linear systems composed by collections of inequalities. To reduce memory consumption it might be interesting to introduce mechanisms to avoid the insertion of redundant constraints and polytopes included in larger ones.

6 Related Work and Conclusion

In this work we proposed a parameter synthesis algorithm for STL specifications. We extended standard STL defined on single traces to sets of traces generated by parametric dynamical systems. The whole procedure operates forwardly, that is refining the parameter sets and computing the behavior on-the-fly, in order to obtain less restrictive parameter sets. We proved the correctness of the algorithm and discussed its computational complexity. Moreover, we provided a concrete implementation of the algorithm for polynomial discrete-time dynamical systems, that produces valid parameter sets in form of sets of convex polytopes. The algorithm was implemented and illustrated on an epidemic model.

In the literature there are different approaches to the problem of finding good parameters. In [25] pure model checking is used on discrete finite structures. Parametric interaction graphs representing Genetic Regulatory Networks are analyzed through LTL in [26]. Discrete-time simulations are considered in [27], where parameters optimize the satisfaction degree of LTL formulas constrained over the reals. [28] describes stochastic approaches for parameter fitting over experimental data, and [29] parameter synthesis for CTMC w.r.t. CSL (Continuous Stochastic Logic) specifications. [30] uses simulation guided analysis to find values of the parameters that do not produce oscillating behaviors, while counterexample guided abstraction refinement over linear hybrid automata has been proposed in [31]. A parameter synthesis approach based on "good" parameters values was proposed in [32] over timed automata with respect to safety properties. Parametric temporal properties are considered in [9], where the parameters are identified to make a temporal property satisfied by experimental data. Many of the above mentioned works find applications in the study of biological systems.

The paper [33] is close to our work since it considers discrete-time systems and LTL properties; however the targeted systems are piecewise-affine (PWA) and the systems we address are polynomial. While our algorithm approximates the reachable sets of polynomial systems, the approach in this work involves computing first a discrete abstraction of the original PWA system. In the abstraction (that is a transition system), each transition is labeled with a set of parameter values that allow the transition to be feasible. LTL model checking is then used to remove some transitions so that the resulting system satisfies the property. The parameters are finally restricted to allow only the remaining transitions. Note that due to special properties of multi-affine functions, the discrete abstraction can be performed efficiently. Nevertheless, this is not possible for the polynomial functions in our problem. Another related work is [34] where STL properties and continuous-time nonlinear systems are considered; it computes the reachable set using the trajectories from a finite number of initial states and sensitivity analysis to determine the robustness of the trajectory set from the neighborhoods of those initial states. The parameter set is refined when the trajectory set violates the property. This approach requires a box discretization of the parameter set that is less compact than our polytopic set representation. A similar approach using robustness is applied to a biological system in [35].

The main novelty of our results is that our parameter synthesis method that can handle nonlinear polynomial systems and complex properties specified using STL. We are able to specify interesting temporal properties and reason on the behavior of a dynamical system and its parameters. Two directions are promising. First, we intend to extend our approach to quantitative semantics integrating a robustness metric such as in [34]. The second direction concerns the treatment of continuous-time dynamical systems and dense-time STL. This is straightforward for linear systems by using one of the available polytope-based algorithms for continuous-time reachability operations, such as [36]. For nonlinear systems, conservatively enclosing the reachable sets over time intervals could be done by exploting ideas from interval computation.

References

1. Pnueli, A.: The temporal logic of programs. In: Symposium on Foundations of Computer Science, SFCS, pp. 46–57. IEEE (1977)
2. Clarke, E.M., Grumberg, O., Peled, D.: Model checking. MIT press (1999)
3. Maler, O., Nickovic, D.: Monitoring temporal properties of continuous signals. In: Lakhnech, Y., Yovine, S. (eds.) FORMATS 2004 and FTRTFT 2004. LNCS, vol. 3253, pp. 152–166. Springer, Heidelberg (2004)
4. Maler, O., Nickovic, D., Pnueli, A.: Checking temporal properties of discrete, timed and continuous behaviors. Pillars of Computer Science, 475–505 (2008)
5. Jones, K.D., Konrad, V., Nickovic, D.: Analog property checkers: a DDR2 case study. Formal Methods in System Design 36(2), 114–130 (2010)
6. Jin, X., Donzé, A., Deshmukh, J.V., Seshia, S.A.: Mining requirements from closed-loop control models. In: Proc. of International Conference on Hybrid Systems: Computation and Control, HSCC, pp. 43–52. ACM (2013)
7. Donzé, A., Fanchon, E., Gattepaille, L.M., Maler, O., Tracqui, P.: Robustness analysis and behavior discrimination in enzymatic reaction networks. PLOS One 6(9), e24246 (2011)

8. Stoma, S., Donzé, A., Bertaux, F., Maler, O., Batt, G.: STL-based Analysis of TRAIL-induced Apoptosis Challenges the Notion of Type I/Type II Cell Line Classification. PLoS Computational Biology 9(5), e1003056 (2013)
9. Asarin, E., Donzé, A., Maler, O., Nickovic, D.: Parametric identification of temporal properties. In: Khurshid, S., Sen, K. (eds.) RV 2011. LNCS, vol. 7186, pp. 147–160. Springer, Heidelberg (2012)
10. Fainekos, G.E., Pappas, G.J.: Robustness of temporal logic specifications for continuous-time signals. Theoretical Computer Science 410(42), 4262–4291 (2009)
11. Bartocci, E., Bortolussi, L., Nenzi, L.: On the robustness of temporal properties for stochastic models. In: Hybrid Systems and Biology, HSB. EPTCS, vol. 125, pp. 3–19 (2013)
12. Chen, X., Ábrahám, E., Sankaranarayanan, S.: Flow*: An analyzer for non-linear hybrid systems. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 258–263. Springer, Heidelberg (2013)
13. Gao, S., Kong, S., Chen, W., Clarke, E.M.: Delta-complete analysis for bounded reachability of hybrid systems. CoRR abs/1404.7171 (2014)
14. Testylier, R., Dang, T.: NLTOOLBOX: A library for reachability computation of nonlinear dynamical systems. In: Van Hung, D., Ogawa, M. (eds.) ATVA 2013. LNCS, vol. 8172, pp. 469–473. Springer, Heidelberg (2013)
15. Donzé, A., Ferrère, T., Maler, O.: Efficient robust monitoring for STL. In: Sharygina, N., Veith, H. (eds.) CAV 2013. LNCS, vol. 8044, pp. 264–279. Springer, Heidelberg (2013)
16. De Giacomo, G., Vardi, M.Y.: Linear temporal logic and linear dynamic logic on finite traces. In: Proc. of Intenrational Joint Conference on Artificial Intelligence, IJCAI, IJCAI/AAAI (2013)
17. Dreossi, T., Dang, T.: Parameter synthesis for polynomial biological models. In: Proc. of International Conference on Hybrid Systems: Computation and Control, HSCC, pp. 233–242. ACM (2014)
18. Dang, T., Dreossi, T., Piazza, C.: Parameter synthesis using parallelotopic enclosure and applications to epidemic models (2014), http://www-verimag.imag.fr/~dreossi/docs/papers/hsb_2014.pdf
19. Asarin, E., Bournez, O., Dang, T., Maler, O.: Approximate reachability analysis of piecewise-linear dynamical systems. In: Lynch, N.A., Krogh, B.H. (eds.) HSCC 2000. LNCS, vol. 1790, pp. 20–31. Springer, Heidelberg (2000)
20. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)
21. Bauer, C., Frink, A., Kreckel, R.: Introduction to the GiNaC framework for symbolic computation within the C++ programming language. Journal of Symbolic Computation 33(1), 1–12 (2002)
22. Chowell, G., Hengartner, N.W., Castillo-Chavez, C., Fenimore, P.W., Hyman, J.: The basic reproductive number of Ebola and the effects of public health measures: the cases of Congo and Uganda. Journal of Theoretical Biology 229(1), 119–126 (2004)
23. Allen, L.J.: Some discrete-time SI, SIR, and SIS epidemic models. Mathematical Biosciences 124(1), 83–105 (1994)
24. Zhou, X., Li, X., Wang, W.-S.: Bifurcations for a deterministic sir epidemic model in discrete time. Advances in Difference Equations 2014(1), 1–16 (2014)

25. Barnat, J., Brim, L., Krejci, A., Streck, A., Safranek, D., Vejnar, M., Vejpustek, T.: On parameter synthesis by parallel model checking. IEEE/ACM Trans. Comput. Biol. Bioinformatics 9(3), 693–705 (2012)
26. Gallet, E., Manceny, M., Le Gall, P., Ballarini, P.: An LTL Model Checking Approach for Biological Parameter Inference. In: Merz, S., Pang, J. (eds.) ICFEM 2014. LNCS, vol. 8829, pp. 155–170. Springer, Heidelberg (2014)
27. Rizk, A., Batt, G., Fages, F., Soliman, S.: Continuous valuations of temporal logic specifications with applications to parameter optimization and robustness measures. Theoretical Computer Science 412(26), 2827–2839 (2011)
28. Gratie, D.-E., Iancu, B., Petre, I.: ODE Analysis of Biological Systems. In: Bernardo, M., de Vink, E., Di Pierro, A., Wiklicky, H. (eds.) SFM 2013. LNCS, vol. 7938, pp. 29–62. Springer, Heidelberg (2013)
29. Češka, M., Dannenberg, F., Kwiatkowska, M., Paoletti, N.: Precise parameter synthesis for stochastic biochemical systems. In: Mendes, P., Dada, J.O., Smallbone, K. (eds.) CMSB 2014. LNCS, vol. 8859, pp. 86–98. Springer, Heidelberg (2014)
30. Dreossi, T., Dang, T.: Falsifying oscillation properties of parametric biological models. In: Hybrid Systems and Biology, HSB. EPTCS, vol. 125, pp. 53–67 (2013)
31. Frehse, G., Jha, S.K., Krogh, B.H.: A counterexample-guided approach to parameter synthesis for linear hybrid automata. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 187–200. Springer, Heidelberg (2008)
32. André, É., Soulat, R.: Synthesis of timing parameters satisfying safety properties. In: Delzanno, G., Potapov, I. (eds.) RP 2011. LNCS, vol. 6945, pp. 31–44. Springer, Heidelberg (2011)
33. Yordanov, B., Belta, C.: Parameter synthesis for piecewise affine systems from temporal logic specifications. In: Egerstedt, M., Mishra, B. (eds.) HSCC 2008. LNCS, vol. 4981, pp. 542–555. Springer, Heidelberg (2008)
34. Donzé, A.: Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In: Touili, T., Cook, B., Jackson, P. (eds.) CAV 2010. LNCS, vol. 6174, pp. 167–170. Springer, Heidelberg (2010)
35. Sankaranarayanan, S., Miller, C., Raghunathan, R., Ravanbakhsh, H., Fainekos, G.: A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In: Proc. of Annual Allerton Conference on Communication, Control, and Computing. IEEE (2012)
36. Frehse, G., Le Guernic, C., Donzé, A., Cotton, S., Ray, R., Lebeltel, O., Ripado, R., Girard, A., Dang, T., Maler, O.: SpaceEx: Scalable verification of hybrid systems. In: Gopalakrishnan, G., Qadeer, S. (eds.) CAV 2011. LNCS, vol. 6806, pp. 379–395. Springer, Heidelberg (2011)