Parameter Synthesis of Polynomial Dynamical Systems

Alberto Casagrande[a], Thao Dang[b], Luca Dorigo[c,d], Tommaso Dreossi[e], Carla Piazza[c], Eleonora Pippia[f]

[a] *University of Trieste, Italy*
[b] *University of Grenoble Alpes, France*
[c] *University of Udine, Italy*
[d] *Fondazione Bruno Kessler, Italy*
[e] *Insitro, California*
[f] *Electrolux, Italy*

Abstract

Parametric dynamical systems emerge as a natural formalism for modeling natural and engineered systems ranging from biology, epidemiology, and medicine to cyber-physics. Parameter tuning is a complex task which, in many cases, is performed exploiting heavy simulations that have high computational complexity and do not ensure the correctness of the synthesized systems. In this manuscript, we consider the problem of parameter synthesis for discrete-time polynomial systems. We propose a formal method based on Bernstein coefficients that allows refining the set of parameters according to a temporal specification defined as a Signal Temporal Logic formula. We prove that the synthesized system is correct with respect to the specification and we demonstrate the scalability of the approach by implementing it in the C++ library `Sapo`, also available within a stand-alone application and as a web application. Finally, we illustrate the tool usage and the interface through a simple yet realistic epidemiological model and consider an intriguing application enhancing the accuracy of the verification of neural networks.

Keywords: Parametric Dynamical Systems, Bernstein Coefficients, Signal Temporal Logic, Neural Network Verification.

Introduction

Dynamical systems are standard mathematical models used to describe the temporal evolution of systems (e.g., see [13, 36]). Parameters allow specifying and analysing the behaviour of a possibly infinite set of models at one time (e.g., see [69]). In the case of partial knowledge over the observed system parameters,

Email addresses: `acasagrande@units.it` (Alberto Casagrande), `thao.dang@univ-grenoble-alpes.fr` (Thao Dang), `dorigo.luca001@spes.uniud.it` (Luca Dorigo), `tommasodreossi@gmail.com` (Tommaso Dreossi), `carla.piazza@uniud.it` (Carla Piazza), `pippia.eleonora@gmail.com` (Eleonora Pippia)

they allow drawing general conclusions despite the lack of information. They are also helpful in abstracting stochastic noise when this has no dramatic effects on global behaviour. During the design phases of engineered systems, parameters have to be refined to guarantee the desired result. In the context of parametric dynamical systems, we are interested in defining algorithmic techniques for solving both reachability analysis and parameter synthesis. Reachability analysis aims to determine the values assumed by the system variables during an evolution that starts from an initial set of states. Instead, parameter synthesis refines the set of parameters to ensure that the reachable states satisfy a specification.

A dynamical system is either *discrete-time* or *continuous-time* depending on whether the changes occur in discrete time instants or in continuous time, respectively. The typical formalisms used to describe discrete-time and continuous-time dynamical systems are *difference* and *differential* equations, respectively. In this work, we consider discrete-time dynamical systems since, in most contexts, the system under analysis can be observed only at fixed time instants. In some cases, discrete numerical integration methods (e.g., Euler, Runge-Kutta) are applied in the study of continuous-time models, see, e.g. [1, 8, 2] where discretization is used to compute discrete step dynamics and, then, the continuous-time flowpipe is over-approximated.

We focus, in particular, on parametric discrete-time dynamical systems in which the dynamics are polynomials. Many works exist on the analysis of linear systems, whose properties, such as convexity preservation, allow to define efficient algorithms, e.g., for computing sets of reachable points. Some attempts for tackling the reachability problem in the case of nonlinearity used approximation and symbolic computation techniques. A recent and detailed review of the state-of-the-art in the context of reachability analysis can be found in [2].

Polynomial systems allow us to exploit the properties of Bernstein coefficients [34] to over-approximate the reachability problem. By imposing linear dependence on the parameters, we can rely on linear optimization algorithms to efficiently solve our problem.

As far as parameter synthesis is concerned, we use temporal logic for specifying the properties of interest. Temporal logics were introduced in formal verification of hardware and software systems for defining specifications involving runtime acceptable computations [61]. Their success in that field is not only due to their balance between expressive power and ease-of-use, but also to the development of efficient algorithms for checking whether a system satisfies a temporal logic specification (e.g., see [18]).

Temporal logic has found applications outside formal verification, for instance, in *monitoring*. Within this context, the system is treated as a black box whose observable behaviours can be analysed by evaluating the satisfaction of the desired temporal specification. *Signal Temporal Logic* (STL [55, 56]) has been introduced for specifying properties of dense-time real-valued signals. It is particularly suitable for monitoring both industrial case studies (e.g., see [47, 46, 23]) and biological systems (e.g., see [25, 67, 20]). It has also been used in the study of parametric systems (e.g., see [7, 43]) where parametric disturbance rejection properties are formalized in STL and then verified. One of its exciting 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 [33, 9, 40].

In our setting, we exploit Bernstein coefficients' properties again and we automatically infer new linear constraints on the parameters. Such constraints ensure that the specification is satisfied.

Recently several methods and tools have been developed for reachability analysis and parameter synthesis. Some examples are HyTech [41], d/dt [6], Breach [24], SpaceEx [35], Ariadne [10], Flow* [17], pyHybridAnalysis [15, 16], dReach [52], CORA [3], DynIbex [57], JuliaReach [11], Kaa [51], and HyPro [64]. Similarly to what we present here, Breach performs parameter synthesis. However, it exploits linearizations to compute the flow over a finite set of points. On the other hand, Flow* and HyPro deal with non-linear systems by exploiting Taylor model arithmetic techniques. They also handle hybrid systems, but not parametric ones. More detailed comparison can be found in [58] where a methodology that integrates different tools for the design of cyber-physical systems is described. ARCH-COMP (International Workshop on Applied Verification of Continuous and Hybrid Systems - Competition) reports the state of the art in the field, see, e.g., [37, 38].

This manuscript extends [22] and [28]. On the one hand, we present the algorithms described in [22] trying to keep the technicalities as simple as possible and give more intuitions through simple examples. On the other hand, we introduce some improvements to obtain more precise results. As far as the implementation is concerned, we extended the C++ library `Sapo` introducing the input language SIL and a web application, significantly improving usability. Finally, we briefly show the usefulness of `Sapo` in the contexts of epidemiological models and neural networks analysis.

The work is organized as follows. Section 1 formally introduces our setting and the problems we consider. Section 2 gives more details on the dynamics and on the set of points/parameters we manipulate. Such details are necessary for defining our reachability algorithm. Section 3 focus on the parameter synthesis problem. `Sapo` library, stand-alone tool, and the web application are described in Section 4. Section 5 presents applications in two different realms, while Section 6 concludes the work and presents some possible future developments.

1. Preliminaries

In this section, we describe the formal notions at the basis of our work. We are interested in dynamical systems, i.e., phenomena evolving in time, and we focus on models of such systems in which time evolves through discrete steps. Specifically, we consider *polynomial discrete-time dynamical systems* whose parameters are used to tune the model to match given observations. We also present a formal language for specifying the expected evolutions of the system called *Signal Time Temporal Logic* (STL) [55] that was originally introduced in the context of monitoring.

1.1. Parametric Dynamical Systems

Let us start with basic notions [48] necessary to define dynamical systems.

The *state* of a system is a description that is sufficient to predict the system's future. In this work, we deal with memoryless systems, i.e., systems for which at time t it is possible to predict the future states without recurring to states prior to t. The space of all the possible system states is called the *state space* of the dynamical system.

The nature of a dynamical system is related to the structure of time it relies on. If the time of a system ranges on non-negative real values, then the system is called *continuous*, while if the time is described by naturals, then the system is said to be *discrete*. The *evolution* of the system over time is a continuous trajectory (in the continuous-time case) or a sequence (in the discrete-time case) of states through the state space.

The rules that allow us to determine the state of the systems are called *dynamics* or *laws of evolution*. Typically the dynamics of dynamical systems are described by differential equations or difference equations depending on whether they are continuous-time or discrete-time, respectively. Finally, the *initial condition* is the state at an initial time from which an evolution starts.

Parametric discrete-time dynamical systems are dynamical systems that evolve in discrete time and include parameters in their definition.

Definition 1 (Parametric Discrete-Time Dynamical System)**.** *A* parametric discrete-time dynamical system *is a tuple* $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ *where:*

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is the state space;*
- $\mathcal{P} \subseteq \mathbb{R}^m$ *is the parameter space;*
- $\mathbf{f} : \mathcal{X} \times \mathcal{P} \to \mathcal{X}$ *is a function.*

The evolutions of parametric discrete-time dynamical systems are governed by difference equation of the form:

$$\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{p}) \tag{1}$$

where $\mathbf{x} \in \mathcal{X}$ is the state of the system, $\mathbf{p} \in \mathcal{P}$ are the parameters, and $t \in \mathbb{N}$ is a discrete-time variable. During the evolution of the dynamical system the state variables can change their values, while the parameters remain constant.

Definition 2 (Trajectory of Parametric Discrete-Time Dynamical System)**.** *A* trajectory *of a parametric discrete-time dynamical system* $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ *starting from an initial state* $\mathbf{x} \in \mathcal{X}$ *with parameters* $\mathbf{p} \in \mathcal{P}$ *is a function* $\xi_{\mathbf{x}}^{\mathbf{p}} : \mathbb{N} \to \mathcal{X}$ *such that* $\xi_{\mathbf{x}}^{\mathbf{p}}$ *is the solution of the difference equation* $\mathbf{x}_{t+1} = \mathbf{f}(\mathbf{x}_t, \mathbf{p})$ *with initial condition* $\mathbf{x}$*, that is:*

$$\xi_{\mathbf{x}}^{\mathbf{p}}(0) = \mathbf{x} \textit{ and } \forall t \in \mathbb{N}_{>0}, \xi_{\mathbf{x}}^{\mathbf{p}}(t+1) = \mathbf{f}(\xi_{\mathbf{x}}^{\mathbf{p}}(t), \mathbf{p}). \tag{2}$$

Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a parametric discrete-time dynamical system, $X_0 \subseteq \mathcal{X}$ be a set of initial conditions, and $P \subseteq \mathcal{P}$ be a set of parameters. The set of all possible trajectories of $\mathcal{D}$ with initial conditions in X_0 and parameters in P is defined as:

$$\Xi(X_0, P) = \{\xi^{\mathbf{p}}_{\mathbf{x}_0} \mid \mathbf{x}_0 \in X_0, \mathbf{p} \in P, \text{ and } \xi^{\mathbf{p}}_{\mathbf{x}_0} \text{ is a trajectory of } \mathcal{D}\}. \tag{3}$$

Given two states $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, we say that $\mathbf{x}'$ is *reachable* from $\mathbf{x}$ in time $0 \leq t < +\infty$ if there are a parameter $\mathbf{p} \in \mathcal{P}$ and a trajectory $\xi^{\mathbf{p}}_{\mathbf{x}}$ of $\mathcal{S}$ starting in $\mathbf{x}$ such that $\mathbf{x}' = \xi^{\mathbf{p}}_{\mathbf{x}}(t)$. The set of all the states reached by the system from $\mathbf{x}_0 \in \mathcal{X}$ with parameter $\mathbf{p} \in \mathcal{P}$ is defined as:

$$\mathit{Reach}^{\mathbf{p}}(\mathbf{x}_0) = \{\mathbf{x}' \mid \mathbf{x}' = \xi^{\mathbf{p}}_{\mathbf{x}_0}(t), t \in \mathbb{N}\}. \tag{4}$$

We can extend the notion of reachability to sets, that is, given a set of initial conditions $X_0 \subseteq \mathcal{X}$ and a parameter set $P \subseteq \mathcal{P}$, the *reachable set* is the set of all the states reachable by the system:

$$\mathit{Reach}^{P}(X_0) = \bigcup_{\mathbf{x}_0 \in X} \bigcup_{\mathbf{p} \in P} \mathit{Reach}^{\mathbf{p}}(\mathbf{x}_0). \tag{5}$$

The definition of reachable set reflects the behaviour of the considered dynamical system for an infinite amount of time. However, we might be interested in studying a model for a bounded time horizon. In particular, let $I = [a, b]$ be a closed bounded interval of $\mathbb{N}$. The set of states reachable within the time interval I is defined as:

$$\mathit{Reach}^{\mathbf{p}}_I(\mathbf{x}_0) = \{\mathbf{x}' \mid \mathbf{x}' = \xi^{\mathbf{p}}_{\mathbf{x}_0}(t), t \in I\} \tag{6}$$

$$\mathit{Reach}^{P}_I(X_0) = \bigcup_{\mathbf{x}_0 \in X_0} \bigcup_{\mathbf{p} \in P} \mathit{Reach}^{\mathbf{p}}_I(\mathbf{x}_0). \tag{7}$$

Example 1. *Let us introduce a simple yet significant example that will be used throughout the paper to help the reader. SIR is a classic 3D epidemic model [50] that describes the spread of an epidemic disease in the simplest possible form. In the SIR model the population is partitioned into 3 classes: S susceptible, I infected, and R removed. Two parameters characterize the disease: β represents the transmission rate, i.e., the probability for a susceptible to become infected once there is a contact with a sick person; α models the recovery rate, i.e., the inverse of the time required for moving from infected to removed. The susceptible individuals at time $t+1$ are the susceptible individuals at time t decremented by the individuals that at time $t+1$ become infected, i.e., decremented by $\beta * S_t * I_t$. Similarly, the infected at time $t+1$ gain the new infected and lose the removed, i.e., $\alpha * I_t$. Finally, the removed are incremented with the new removed. Every removed individual is considered immune to the disease in this simple model. In this work, we will refer to a normalized population of size* 1*, i.e., the system's variables will range from* 0 *to* 1*.*

With our notation an SIR model is defined as $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ where:

- $\mathcal{X} = [0, 1]^3$ *is the state space described by the variables S, I, and R;*

- $\mathcal{P} = [0,1]^2$ *is the parameter space described by the parameters* α *and* β*;*
- $\mathbf{f} : \mathcal{X} \times \mathcal{P} \to \mathcal{X}$ *is defined as*

$$\begin{aligned} S_{t+1} &= S_t - \beta S_t I_t \\ I_{t+1} &= I_t + \beta S_t I_t - \alpha I_t \\ R_{t+1} &= R_t + \alpha I_t \end{aligned}$$

Let us consider the case of $\beta = 0.5$ *and* $\alpha = 1/5$*. If the initial point* $\mathbf{x}_0$ *is* (S_0, I_0, R_0) *where* $S_0 = 0.99$*,* $I_0 = 0.01$*, and* $R_0 = 0$*, we have:*

$$\begin{aligned} Reach^{(0,5,1/5)}_{[0,6]}(\mathbf{x}_0) = \{&(0.99, 0.01, 0), (0.98505, 0.01295, 0.002), \\ &(0.978672, 0.0167382, 0.00459), \\ &(0.970481, 0.0215812, 0.00793764), \\ &(0.96000914, 0.02773698, 0.01225387), \\ &(0.94669527, 0.03550347, 0.01780127), \\ &(0.92988978, 0.04520825, 0.02490196)\} \end{aligned}$$

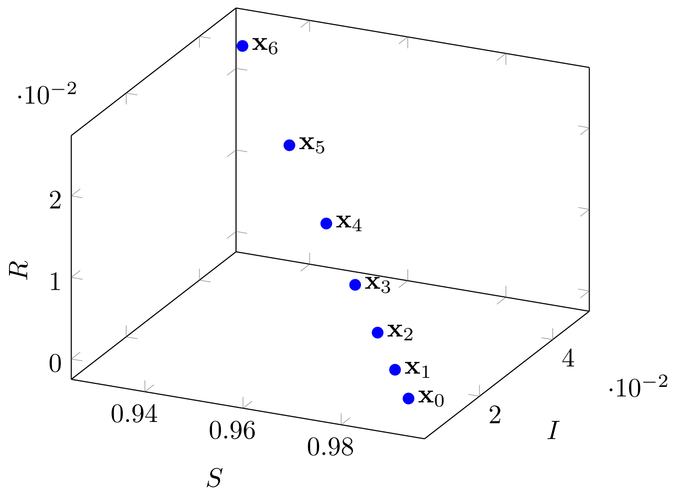


Figure 1: The set $Reach^{(0,5,1/5)}_{[0,6]}(\mathbf{x}_0)$, where $\mathbf{x}_0 = (0.99, 0.01, 0)$, described in Example 1. The points are labelled as $\mathbf{x}_i$ instead of $\xi^{\mathbf{p}}_{\mathbf{x}_0}(i)$ to simplify the plot.

As soon as one of the parameters or one of the variables is estimated through an interval, we obtain an infinite set of points.

1.2. Synthesis Language: Signal Temporal Logic

Temporal logics [61] are well-known formalisms for specifying desirable properties of systems under investigation. In formal verification, temporal logic properties are used to identify system's bugs. Model-checking algorithms take as input a model and a temporal formula and return a counter-example if the model does not satisfy it, i.e., a proof of the existence of unwanted behaviours. In our setting, we use temporal logic formulas to synthesize a set of parameters that guarantees the temporal property.

Example 2. *Let us consider the SIR model described in Example 1 again and imagine that we do not know the transmission rate β. We may try to estimate it by using some statistics. For instance, if we know that the number of infected ranged is in the interval* $[0.01, 0.02]$ *at time* 0*, while it roses to* $[0.015, 0.03]$ *after* 6 *days, and if we also know that the recovery time is* 4 *days, i.e.,* $\alpha = 1/4$*, then we can intuitively deduce that the transmission rate cannot be too close to* 1.

Signal Temporal Logic (STL [55, 56, 26]) formalizes properties of dense-time real-valued *signals*, i.e., functions defined on dense intervals. In our context, the trajectories of a dynamical system are the signals over which STL formulas are evaluated. We specifically focus on STL formulas in positive normal form over a bounded time horizon.

Let $\Sigma = \{\sigma_1, \sigma_2, \dots, \sigma_k\}$ be a finite set of predicates mapping $\mathbb{R}^n$ into Boolean values. For a given $j \in \{1, 2, \dots, k\}$, the predicate σ_j is of the form $\sigma_j = s_j(x_1, x_2, \dots, x_n) \leq 0$ where $s_j : \mathbb{R}^n \to \mathbb{R}$ is a function over the variables $x_1, x_2, \dots, x_n$.

A Signal Temporal Logic formula is generated by the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi U_I \varphi \tag{8}$$

where $\sigma \in \Sigma$ is a predicate and $I = [a, b]$ is a bounded closed interval of $\mathbb{N}$. For $t \in \mathbb{N}$, the shifted interval $t + I$ is the set $\{t + t' \mid t' \in I\}$. A STL formula is in positive normal form when does not use the negation operator $\neg$.

There are two elements that distinguish STL from other logics:

- the *predicates* σ are evaluated on real-values, that in our case are the states of the dynamical system;
- the *temporal operators* $\varphi U_I \varphi$ are decorated with intervals that determine the temporal windows on which the operators are defined.

Other standard temporal operators can be derived as follows:

- *true* $\top \equiv 0 \leq 0$;
- *false* $\bot \equiv 1 \leq 0$;
- $\neg\bot \equiv \top$ and $\neg\top \equiv \bot$;
- *release* $\varphi_1 R_{[a,b]} \varphi_2 \equiv\ (\varphi_2 U_{[a,b]} (\varphi_1 \wedge \varphi_2)) \vee (\varphi_2 U_{[b+1,b+1]} \top)$;
- *eventually in the future* $F_I \varphi \equiv \top U_I \varphi$;
- *always in the future* $G_I \varphi \equiv \top U_{[a,a]} (\bot R_{[0,b-a]} \varphi)$.

Any STL formula ϕ can be rewritten as an equivalent STL formula in positive normal form φ such that $\varphi \equiv \phi$ (e.g., see [63]). Because of this result, in the remaining part of this manuscript, we exclusively deal with STL formulas in positive normal form.

Example 3. *The property informally described in Example 2 can be stated as:*

$$F_{[6,6]}(0.015 \leq I \leq 0.03).$$

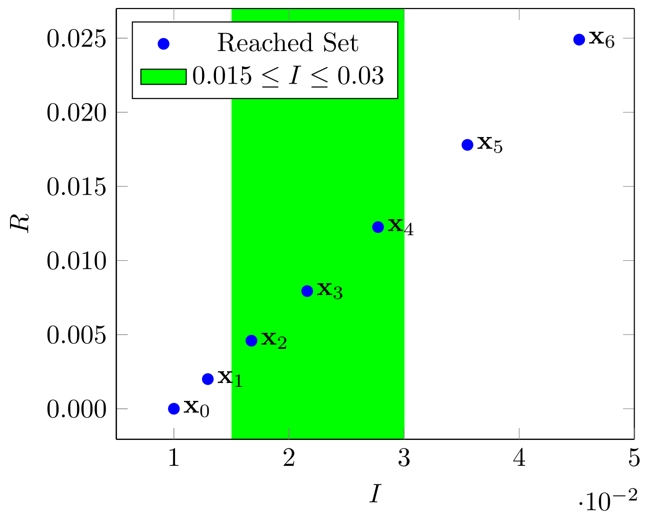


Figure 2: The evaluation of the STL formulas $F_{[6,6]}(0.015 \leq I \leq 0.03)$ on the trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ described in Example 1. The points are labelled as $\mathbf{x}_i$ instead of $\xi^{\mathbf{p}}_{\mathbf{x}_0}(i)$ to simplify the plot. The formula does not hold because $\xi^{\mathbf{p}}_{\mathbf{x}_0}(6)$ does not satisfy the formula $0.015 \leq I \leq 0.03$.

An interesting aspect of STL is its semantics. Two semantics for STL formulas can be defined: *qualitative* semantics, also known as Boolean semantics, and *quantitative* semantics. Intuitively, the former establishes the truth value of a formula over a trace, telling us whether a formula holds or not; the latter provides additional information on how robustly a trace satisfies (or not) a formula. This article refers to the qualitative semantics of STL which is defined over trajectories as follows.

Definition 3 (Qualitative Semantics [55])**.** *Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a parametric discrete-time dynamical system. Let $\xi^{\mathbf{p}}_{\mathbf{x}}$ be a trajectory of $\mathcal{D}$ starting from $\mathbf{x} \in \mathcal{X}$ with parameter $\mathbf{p} \in \mathcal{P}$. Let $t \in \mathbb{R}_{\geq 0}$ be a time instant, and φ be an STL formula. The* qualitative semantics *of φ at time t over $\xi^{\mathbf{p}}_{\mathbf{x}}$ is given by the following inductive definition:*

$$
\begin{array}{lll}
\xi_{\mathbf{x}}^{\mathbf{p}}, t \models \sigma & \textit{iff} & \sigma(\xi_{\mathbf{x}}^{\mathbf{p}}(t)) \textit{ is true} \\
\xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_1 \wedge \varphi_2 & \textit{iff} & \xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_1 \textit{ and } \xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_2 \\
\xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_1 \vee \varphi_2 & \textit{iff} & \xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_1 \textit{ or } \xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_2 \\
\xi_{\mathbf{x}}^{\mathbf{p}}, t \models \varphi_1 U_I \varphi_2 & \textit{iff} & \exists t' \in t + I \textit{ s.t. } \xi_{\mathbf{x}}^{\mathbf{p}}, t' \models \varphi_2 \textit{ and} \\
 & & \quad \textit{for all } t'' \in [t, t'], \xi_{\mathbf{x}}^{\mathbf{p}}, t'' \models \varphi_1
\end{array}
$$

We say that a trajectory $\xi_{\mathbf{x}}^{\mathbf{p}}$ *satisfies* φ, denoted by $\xi_{\mathbf{x}}^{\mathbf{p}} \models \varphi$, if and only if $\xi_{\mathbf{x}}^{\mathbf{p}}, 1 \models \varphi$. Notice that the semantics is given with respect to $t = 1$, while usually $t = 0$ is considered. In other terms is like if we are considering only formulas starting with a *next* operator. It will become clear soon that this is a reasonable requirement for talking about synthesis.

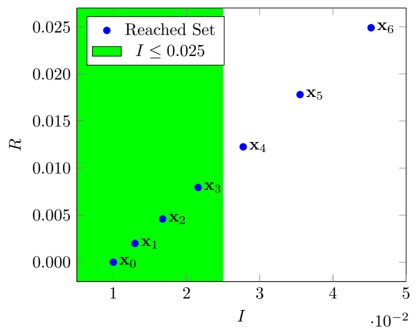
Example 4. *Let us consider again the bounded trajectory described in Example 1. It satisfies the formula:*

$$G_{[0,3]}(I \leq 0.025),$$

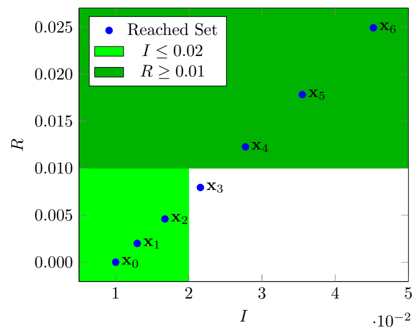
since the maximum value reached by I *among* $\xi_{\mathbf{x}_0}^{\mathbf{p}}(0)$, $\xi_{\mathbf{x}_0}^{\mathbf{p}}(1)$, $\xi_{\mathbf{x}_0}^{\mathbf{p}}(2)$, *and* $\xi_{\mathbf{x}_0}^{\mathbf{p}}(3)$ *is* 0.0215812. *Instead, it does not satisfy the formula:*

$$(I \leq 0.02) U_{[0,3]} (R \geq 0.01),$$

since $I \leq 0.02$ *becomes false in* $\xi_{\mathbf{x}_0}^{\mathbf{p}}(3)$, *while* $R \geq 0.01$ *is not yet true.*



(a) The trajectory $\xi_{\mathbf{x}_0}^{\mathbf{p}}$ satisfies the STL formula $G_{[0,3]} I \leq 0.025$ since $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ satisfy the formula $I \leq 0.025$.



(b) Since $\mathbf{x}_0$, $\mathbf{x}_1$, $\mathbf{x}_2$, and $\mathbf{x}_3$ do not satisfy $R \geq 0.01$ and $I \leq 0.02$ doesn't hold on $\mathbf{x}_3$, $\xi_{\mathbf{x}_0}^{\mathbf{p}}$ does not satisfy $(I \leq 0.02) U_{[0,3]} (R \geq 0.01)$.

Figure 3: The evaluation of the two STL formulas $G_{[0,3]} I \leq 0.025$ and $(I \leq 0.02) U_{[0,3]} (R \geq 0.01)$ on the trajectory $\xi_{\mathbf{x}_0}^{\mathbf{p}}$ described in Example 1. In order to simplify the plot, the points are labelled as $\mathbf{x}_i$ meaning $\xi_{\mathbf{x}_0}^{\mathbf{p}}(i)$.

1.3. Parametric Reachability and Synthesis Problems

Within the above described context, i.e., models defined in terms of parametric discrete-time dynamical systems and properties specified as STL formulas, we are ready to formalize two problems of interest: reachability computation and parameter synthesis.

Definition 4 ((Bounded) Parametric Reachability Computation)**.** *Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a dynamical system, $X_0 \subseteq \mathcal{X}$ be a set of initial conditions, $P \subseteq \mathcal{P}$ be a set of parameters, and I be a closed interval of $\mathbb{N}$.*

The Parametric Reachability Computation problem *asks to compute the set of points that can be reached with a trajectory of $\mathcal{D}$ starting from $\mathbf{x} \in X_0$ and $\mathbf{p} \in P$, i.e., it asks to compute the set $Reach^P(X_0)$.*

The Bounded Parametric Reachability Computation problem *asks to compute the set of points that can be reached with a trajectory of $\mathcal{D}$ starting from $\mathbf{x} \in X_0$ and $\mathbf{p} \in P$ within the time interval I, i.e., it asks to compute the set $Reach^P_I(X_0)$.*

As one can imagine in the general case both reachability and bounded reachability are not computable (see, e.g., [42, 12]). However, in Section 2 we will discuss how bounded reachability can be over-approximated in the case of polynomial systems.

Definition 5 (Parameter Synthesis)**.** *Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a dynamical system, $X_0 \subseteq \mathcal{X}$ be a set of initial conditions, $P \subseteq \mathcal{P}$ be a set of parameters, and φ be an STL specification. The* Parameter Synthesis Problem *asks to compute the largest subset $P_\varphi \subseteq P$ such that $\forall \mathbf{x}_0 \in X_0, \forall \mathbf{p} \in P_\varphi, \xi^{\mathbf{p}}_{\mathbf{x}_0} \models \varphi$, where $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ is a trajectory of $\mathcal{D}$.*

Now that we have the definition of the problem it is natural to observe that we had to refer the notion of satisfiability to time $t = 1$, since at time $t = 0$ the set X_0 is given. Hence, it makes no sense to consider parameter refinement at time $t = 0$, since they have not been used yet.

Again, while the problem is in general not computable, in Section 3 we will present a method for under-approximating the solution over polynomial systems.

2. Parametric Reachability Computation

2.1. Set Based Reachability

In the context of formal verification, the interest is not only in simulating one specific trajectory of a dynamical system starting from a given initial condition, but also in computing a *flowpipe* that contains all the states reachable by the system from a possibly infinite set of initial conditions.

A standard methodology for computing flowpipes in the case of parametric discrete-time dynamical systems consists of the following steps:

- Choice of a domain DX providing finite representations for the set of points to be manipulated. The domain should allow any set of points to be both over and under-approximated. The most used domains are ellipsoids, hyper-rectangles, zonotopes.

- Choice of a domain DP providing finite representations for the set of parameters to be considered.

- Choice of a family F of admissible functions to be used in the dynamical systems. For instance, linear systems or polynomial systems.
- Definition of an algorithm REACHSTEP that for each representable set of points $X \in DX$, each representable set of parameters $P \in DP$, and each admissible function $\mathbf{f} \in F$, approximates the set of points $\mathbf{f}(X, P) = \{\mathbf{f}(\mathbf{x}, \mathbf{p}) \mid \mathbf{x} \in X \text{ and } \mathbf{p} \in P\}$ with a representable set of points $X' \in DP$.

Notice that in the general case the set of points $\mathbf{f}(X, P)$ could be not exactly representable in the chosen domain of sets of points DX. For instance, if one considers hyper-rectangles and quadratic functions, the image of a hyper-rectangle is not a hyper-rectangle. Hence, the algorithm has to approximate such image with a new hyper-rectangle.

Given a set $X_0 \in DX$, a set of parameters $P \in DP$, a function $\mathbf{f} \in F$, and a time interval $I = [a, b]$, Algorithm 1 can be used to compute an approximated solution of the bounded reachability problem, i.e., to approximate $Reach_I^P(X_0)$, by iterating REACHSTEP computations.

Algorithm 1 Reachability

```
1:  function REACH($X_0, P, \mathbf{f}, a, b$)
2:      $X \leftarrow X_0$
3:      $Y \leftarrow \emptyset$
4:      for $i = 1, \ldots, b$ do
5:          $X \leftarrow$REACHSTEP$(X, P, \mathbf{f})$
6:          if $i \geq a$ then
7:              $Y \leftarrow Y \cup \{X\}$
8:          end if
9:      end for
10:     return $Y$
11: end function
```

In the next section, we will focus on our choice of the domains DX and DP and on the family F of polynomial functions and describe our REACHSTEP algorithm based on Bernstein coefficients.

2.2. Sets of Points and Sets of Parameters: Polytopes

The domain that we consider for sets of points is that of polytopes.

Definition 6. *A* polytope $Q \subset \mathbb{R}^n$ *is a closed, compact, bounded subset of* $\mathbb{R}^n$ *such that there is a finite set of half-spaces* $H = \{h_1, \ldots, h_m\}$ *whose intersection is* Q*, that is:*

$$Q = \bigcap_{i=1}^{m} h_i \tag{9}$$

where a half-space $h = \{\mathbf{x} \mid \mathbf{d}\mathbf{x}^T \leq c\}$ *is a set characterized by a non-null normal vector* $\mathbf{d} \in \mathbb{R}^n$ *and an offset* $c \in \mathbb{R}$.

The linear constraints that generate the half-spaces can be organized in a matrix $D \in \mathbb{R}^{m \times n}$ called *template* and a vector $\mathbf{c} \in \mathbb{R}^m$ called *offset vector*, in short *offsets*. The polytope generated by the template D and the offset vector $\mathbf{c}$ is denoted by $\langle D, \mathbf{c} \rangle$. Notice that not all the pairs $\langle D, \mathbf{c} \rangle$ define a nonempty polytope.

Example 5. *Let us consider the SIR model of Example 1 again. Imagine that we start observing the system after some days from the initial spread of the disease. It can be the case that we do not precisely know the number of susceptible, infected, and removed, but we only have some estimations saying that the number of removed is at most the* 2% *of the population. This means that we can consider the following polytope:*

$$\begin{cases} S \geq 0 \\ I \geq 0 \\ R \geq 0 \\ R \leq 0.02 \\ S + I \geq 0.98 \end{cases}$$

The corresponding template matrix and offset vector are

$$D = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 0.02 \\ 0.98 \end{pmatrix}$$

Our reachability algorithm fixes a template matrix and computes at each step a new offset. Boxes and parallelotopes are particular classes of polytopes and are helpful in our context to provide canonical representations. Boxes are n-dimensional generalizations of rectangles.

Definition 7 (Box)**.** *A set $B \subset \mathbb{R}^n$ is a* box *if and only if it can be expressed as the Cartesian product of n intervals, that is:*

$$B = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n] = \prod_{i=1}^{n} [\underline{x}_i, \overline{x}_i], \tag{10}$$

where $\underline{x}_i, \overline{x}_i \in \mathbb{R}$, for $i \in \{1, \ldots, n\}$.

It is easy to see that a box $B = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_n, \overline{x}_n]$ is a polytope since it can be represented by a template $D \in \mathbb{R}^{2n \times n}$ and offsets $\mathbf{c} \in \mathbb{R}^{2n}$ where:

$$D = \begin{pmatrix} 1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & 1 \\ -1 & \dots & 0 \\ \vdots & \ddots & \vdots \\ 0 & \dots & -1 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} \overline{x}_1 \\ \vdots \\ \overline{x}_n \\ -\underline{x}_1 \\ \vdots \\ -\underline{x}_n \end{pmatrix} \tag{11}$$

The unit box is the box $[0,1] \times [0,1] \times \cdots \times [0,1]$.

A parallelotope is a centrally symmetric convex polytope whose opposite facets are parallel. It can be represented as a collection of linear constraints.

Definition 8 (Parallelotope)**.** *Let $\Lambda \in \mathbb{R}^{2n \times n}$ be a template matrix such that, for each $i \in \{1, 2, \dots, n\}$, $\Lambda_i = -\Lambda_{i+n}$, and let $\mathbf{c} \in \mathbb{R}^{2n}$. The parallelotope generated by Λ and $\mathbf{c}$ is the polytope:*

$$\langle \Lambda, \mathbf{c} \rangle = \{\mathbf{x} \mid \Lambda \mathbf{x} \leq \mathbf{c}\}. \tag{12}$$

Example 6. *We consider again the SIR model. A parallelotope over the variables of the system is given by:*

$$\Lambda = \begin{pmatrix} -1 & 0 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} -0.8 \\ -0.95 \\ 0 \\ 0.85 \\ 1 \\ 0 \end{pmatrix}$$

According to the definition, each line in Λ and $\mathbf{c}$ defines a linear inequality on S, I, and R which corresponds to a half-space in $S \times I \times R$. For instance, the second line defines the inequality $-S - I \leq -0.95$; the associated half-space consists of all the points that do not lay below the green line in Figure 4. Since all the inequalities belong to the same system defined by $\langle \Lambda, \mathbf{c} \rangle$, all of them must hold at the same time. Thus, the parallelotope is the intersection of all the corresponding half-spaces.

In this example, $\langle \Lambda, \mathbf{c} \rangle$ constrains the variable S to lay in $[0.8, 0.85]$, the variable R to be 0, and the sum of $S + I$ to belong to the interval $[0.95, 1]$. Figure 4 depicts a projection of $\langle \Lambda, \mathbf{c} \rangle$ itself on the plane $R = 0$.

The above representation is also called *constraint representation.* Another way to characterize parallelotopes, similar to the one adopted for zonotopes [19], is to fix a point of origin and use vectors to define the parallelotope. From Equation (11), it is immediate that boxes are parallelotopes.

Definition 9 (Parallelotope Generator Representation)**.** *Let $U = \{\mathbf{u}^1, \mathbf{u}^2, \dots, \mathbf{u}^n\}$ be a set of linearly independent unit vectors in $\mathbb{R}^n$ and $\boldsymbol{\beta} = (\beta_1, \beta_2, \dots, \beta_n)$*

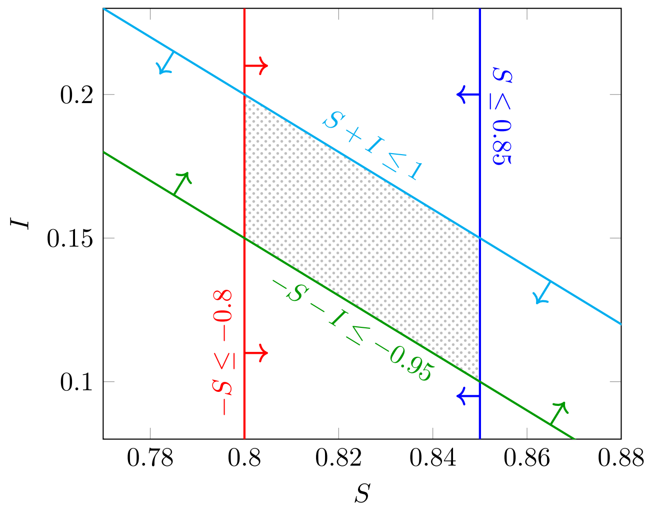


Figure 4: The projection on the plane $R = 0$ of the paralletope $\langle \Lambda, \mathbf{c} \rangle$ described in Example 6. Each line in Λ and $\mathbf{c}$ defines a linear inequality on S, I, and R which corresponds to a half-space in $S \times I \times R$. The paralletope is the intersection of all the half-spaces. The gray colored region depicted in this figure is the projection of $\langle \Lambda, \mathbf{c} \rangle$.

be a vector of coefficients. We consider the set G of generators defined as $G = \{\mathbf{g}^1 = \beta_1 \mathbf{u}^1, \mathbf{g}^2 = \beta_2 \mathbf{u}^2, \ldots, \mathbf{g}^n = \beta_n \mathbf{u}^n\}$. Moreover, let $\mathbf{q}$ be a point in $\mathbb{R}^n$. The parallelotope generated by G and $\mathbf{q}$ is:

$$\mathcal{P}_{gen}(G, \mathbf{q}) = \left\{ \mathbf{q} + \sum_{j=1}^{n} \lambda_j \mathbf{g}^j \mid (\lambda_1, \ldots, \lambda_n) \in [0,1]^n \right\}. \tag{13}$$

This representation is called *generator representation*. The vectors $\mathbf{g}^j$ are called *generators* of the parallelotope and $\mathbf{q}$ is called *base vertex*. Given a set of generators $G = \{\mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^n\}$ and a base vertex $\mathbf{q}$, we also represent the parallelotope generated by G and $\mathbf{q}$ with the notation:

$$\mathcal{P}_{gen}(G, \mathbf{q}) = \{\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n) \mid (\lambda_1, \ldots, \lambda_n) \in [0,1]^n\}, \tag{14}$$

where $\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n)$ is the linear function defined as:

$$\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n) \ = \mathbf{q} + \sum_{j=1}^{n} \lambda_j \mathbf{g}^j. \tag{15}$$

The generator representation of parallelotopes resembles the way in which zonotopes are usually defined, i.e., as the sets $\{\mathbf{c} + \sum_{j=1}^{m} \lambda_j \mathbf{g}^j \mid (\lambda_1, \ldots \lambda_m) \in [-1,1]^m\}$ where $\mathbf{c}$ is the center of the zonotope and $\mathbf{g}^1, \ldots, \mathbf{g}^m$ are its generators. This similarity is not surprising as the parallelotopes in $\mathbb{R}^n$ are special kinds of zonotopes having exactly n linearly independent generators. However, it is worth noticing that our representation of parallelotopes uses base vertices as opposed to the zonotope definition which refers to the zonotope centers. This choice affects the domain of the coefficient $\lambda_1, \dots \lambda_n$, which is reduced from $[-1, 1]^n$ to $[0, 1]^n$ in parallelotope representation, and emphasizes that any parallelotope can be seen as the affine transformation of the unit box $[0, 1]^n$. The reachability algorithm described in the remaining part of this section extensively uses this property.

Standard algebraic procedures can be used to switch from one representation to the other and back (e.g., see [27]).

Example 7. *Let us consider the parallelotope described in Example 6. In order to compute a parallelotope generator representation for it, we can proceed as follows. We consider as base vertex the point of the parallelotope obtained by minimizing all the variables, i.e.,* $\mathbf{q} = (0.8, 0.15, 0)$*. We compute the vectors that go from* $\mathbf{q}$ *to the other vertexes and normalize them. So we obtain* $U = \{(0.7071, -0.7071, 0), (0, 1, 0), (0, 0, 1)\}$ *and* $\boldsymbol{\beta} = (0.07071, 0.05, 0)$*. This means that the parallelotope is defined as the set of points whose coordinates satisfy:*

$$\mathbf{q} + \lambda_1 \mathbf{g}^1 + \lambda_2 \mathbf{g}^2$$

where $\mathbf{q} = (0.8, 0.15, 0)$*,* $\mathbf{g}^1 = (0.05, -0.05, 0)$*,* $\mathbf{g}^2 = (0, 0.05, 0)$*, and* $\lambda_1, \lambda_2 \in [0, 1]$*. In this case,* $\mathbf{g}^3$ *and* λ_3 *disappear since* $\beta_3 = 0$*. This is because* R *has a single value instead of ranging in a proper interval in Example 6.*

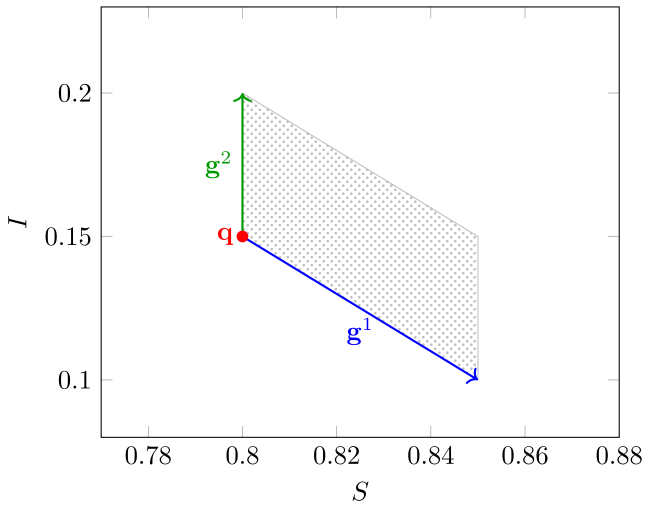


Figure 5: The projection on the plane $R = 0$ of the paralletope $\langle \Lambda, \mathbf{c} \rangle$ described in Example 6, its base vertex $\mathbf{q} = (0.8, 0.15, 0)$, and the generators $\mathbf{g}^1 = (0.05, -0.05, 0)$ and $\mathbf{g}^2 = (0, 0.05, 0)$.

Figure 5 depicts the projection of the plane $R = 0$ *of the parallelotope de-*

scribed in Example 6 together with the base vertex $\mathbf{q} = (0.8, 0.15, 0)$ *and its generators.*

Parallelotopes can be used to decompose polytopes through parallelotope bundles.

Definition 10 (Parallelotope Bundle)**.** *A* parallelotope bundle $\mathcal{B} = \{P_1, \dots, P_b\}$ *is a finite set of parallelotopes whose intersection, denoted by* $\mathcal{I}(\mathcal{B}) = \cap_{i=1}^{b} P_i$*, generates a polytope.*

The polytope $\mathcal{I}(\mathcal{B})$ represented by a parallelotope bundle $\mathcal{B}$ corresponds to the set of all the templates and offsets of the parallelotopes that constitute $\mathcal{B}$ itself. The following Lemma can be easily proved (see [30]) and establishes an upper bound on the number of parallelotopes necessary to represent a polytope.

Lemma 1 (Polytope Decomposition)**.** *Any polytope* $Q \subset \mathbb{R}^n$ *defined by* m *constraints can be represented by a bundle involving* $\lceil m/n \rceil$ *parallelotopes.*

Example 8. *Intuitively the polytope described in Example 5 can be decomposed in the two parallelotopes defined as follows:*

- *in the parallelotope* P_1 *the variable* S *ranges in* $[0, 0.98]$*, the variable* I *ranges in* $[0, 0.98]$*, and the variable* R *ranges in* $[0, 0.02]$*;*
- *in the parallelotope* P_2 *the sum* $S + I$ *ranges in* $[0, 0.98]$*, the variable* I *ranges in* $[0, 0.98]$*, and the variable* R *ranges in* $[0, 0.02]$*.*

As far as the domain for sets of parameters is concerned, we refer again to polytopes: $P \subseteq \mathcal{P}$ is going to be a polytope. While the described representations for sets of states in terms of boxes and parallelotopes are at the basis of our algorithm, the polytopes representing sets of parameters will be handled as systems of linear inequalities with variables in $\mathcal{P}$.

2.3. Admissible Functions: Polynomials

The family of functions that we consider for defining dynamical systems is based on polynomials.

Definition 11 (Parametric Polynomial System)**.** *A* parametric polynomial system $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ *is a parametric discrete-time dynamical system in which* $\mathbf{f} : \mathcal{X} \times \mathcal{P} \to \mathcal{X}$ *is such that* $\mathbf{f}(\mathbf{x}, \mathbf{p}) = (\mathbf{f}_1(\mathbf{x}, \mathbf{p}), \dots, \mathbf{f}_n(\mathbf{x}, \mathbf{p}))$ *and each of the* $\mathbf{f}_i$ *is a polynomial and it is linear over* $\mathbf{p}$*.*

Bernstein coefficients and their properties are valuable tools for bounding polynomials. In [30], we investigated the use of Bernstein coefficients for the over-approximation of bounded reachability in the case of polynomial systems without parameters. Since this work deals with parametric dynamical systems, we want to bound parametric polynomials of the kind $\mathbf{f}_i(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. Thus, we have to extend the definition of Bernstein basis to the parametric case.

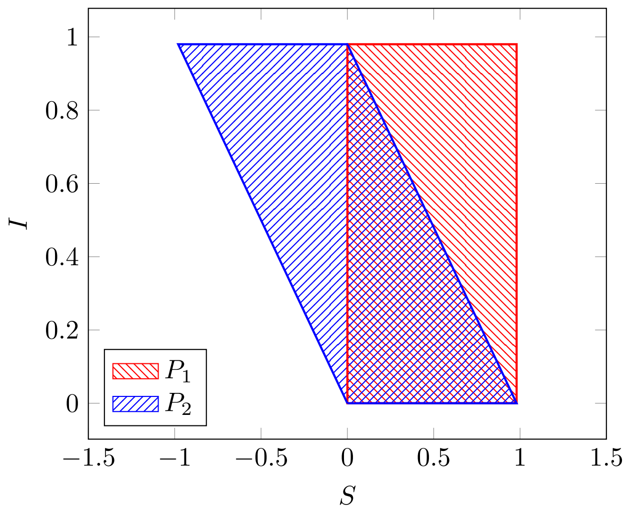


Figure 6: The projection on the plane $R = 0$ of the polytope $\langle D, \mathbf{c}\rangle$ described by Example 5. $\langle D, \mathbf{c}\rangle$ can be decomposed in the two parallelotope P_1 and P_2 and, as a consequence, in the bundle $\mathcal{B} = \{P_1, P_2\}$. The intersection of the bundle parallelotopes equals the polytope, i.e., $\mathcal{I}(\mathcal{B}) = P_1 \cap P_2 = \langle D, \mathbf{c}\rangle$

We will see that all the presented properties related to the Bernstein coefficients also hold when parameters are involved.

A *multi-index* is a vector $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \dots, \mathbf{i}_n)$ where each $\mathbf{i}_j \in \mathbb{N}$. Given two multi-indices $\mathbf{i}$ and $\mathbf{d}$ of the same dimension, we write $\mathbf{i} \leq \mathbf{d}$ ($\mathbf{d}$ dominates $\mathbf{i}$) if for all $j \in \{1, 2, \dots, n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. We also write $\mathbf{i}/\mathbf{d}$ for the multi-index $(\mathbf{i}_1/\mathbf{d}_1, \mathbf{i}_2/\mathbf{d}_2, \dots, \mathbf{i}_n/\mathbf{d}_n)$ and $\binom{\mathbf{d}}{\mathbf{i}}$ for the product of the binomial coefficients $\binom{\mathbf{d}_1}{\mathbf{i}_1}\binom{\mathbf{d}_2}{\mathbf{i}_2}\dots\binom{\mathbf{d}_n}{\mathbf{i}_n}$.

Let us now focus on each of the components of the function $\mathbf{f}$. To avoid using further indexes we denote it as $\pi(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$.

A *parametric polynomial* $\pi(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is usually represented in the power basis as:

$$\pi(\mathbf{x}, \mathbf{p}) = \sum_{\mathbf{i} \in I^\pi} \mathbf{a}_\mathbf{i}(\mathbf{p})\mathbf{x}^\mathbf{i}, \tag{16}$$

where I^π is the set of multi-indexes of the polynomial π over the variables $\mathbf{x}$ and the coefficients $\mathbf{a}_\mathbf{i}(\mathbf{p}) : \mathbb{R}^m \to \mathbb{R}$, for $\mathbf{i} \in I^\mathbf{f}$, are functions defined over the parameters $\mathbf{p} \in \mathbb{R}^m$. In the case of parametric polynomial systems we further required a linear dependence over $\mathcal{P}$, i.e., the coefficients $\mathbf{a}_\mathbf{i}(\mathbf{p})$ are linear in $\mathbf{p}$.

Example 9. *Consider the parametric polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1) = 1/3\,\mathbf{p}_1\,\mathbf{x}_1^2 - 1/2\,\mathbf{p}_1\,\mathbf{x}_2 + 1/4\,\mathbf{x}_1\mathbf{x}_2 + 1/2$*. Its non-null parametric coefficients are* $\mathbf{a}_{(2,0)}(\mathbf{p}_1) = \mathbf{p}_1/3$*,* $\mathbf{a}_{(0,1)}(\mathbf{p}_1) = -\mathbf{p}_1/2$*,* $\mathbf{a}_{(1,1)}(\mathbf{p}_1) = 1/4$*, and* $\mathbf{a}_{(0,0)}(\mathbf{p}_1) = 1/2$*.*

Bernstein basis polynomials of degree $\mathbf{d}$ are basis for the space of polynomials of degree at most $\mathbf{d}$ over $\mathbb{R}^n$. For $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, ..., \mathbf{x}_n) \in \mathbb{R}^n$, the $\mathbf{i}$-th Bernstein polynomial of degree d is defined as:

$$\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(\mathbf{x}_1)\beta_{\mathbf{d}_2,\mathbf{i}_2}(\mathbf{x}_2)\dots\beta_{\mathbf{d}_n,\mathbf{i}_n}(\mathbf{x}_n) \tag{17}$$

where, for any real number $x \in \mathbb{R}$ and for any $d, i \in \mathbb{N}$,

$$\beta_{d,i}(x) = \binom{d}{i} x^i (1-x)^{d-i}. \tag{18}$$

Example 10. *It is easy to verify that* $\beta_{0,0}(x) = 1$*,* $\beta_{1,0}(x) = 1-x$*,* $\beta_{1,1}(x) = x$*,* $\beta_{2,0}(x) = (1-x)^2$*,* $\beta_{2,1}(x) = 2x(1-x)$*, and* $\beta_{2,2}(x) = x^2$*. Thus,*

$$\mathcal{B}_{((2,1),(0,1))}(\mathbf{x}) = \beta_{2,0}(\mathbf{x}_1)\beta_{1,1}(\mathbf{x}_2) = (1-\mathbf{x}_1)^2\mathbf{x}_2.$$

Analogously, $\mathcal{B}_{((2,1),(0,0))}(\mathbf{x}) = (1-\mathbf{x}_1)^2(1-\mathbf{x}_2)$*,* $\mathcal{B}_{((2,1),(1,0))}(\mathbf{x}) = 2\mathbf{x}_1(1-\mathbf{x}_1)(1-\mathbf{x}_2)$*,* $\mathcal{B}_{((2,1),(1,1))}(\mathbf{x}) = 2\mathbf{x}_1(1-\mathbf{x}_1)\mathbf{x}_2$*,* $\mathcal{B}_{((2,1),(2,1))}(\mathbf{x}) = \mathbf{x}_1^2\mathbf{x}_2$*, and* $\mathcal{B}_{((2,1),(2,0))}(\mathbf{x}) = \mathbf{x}_1^2(1-\mathbf{x}_2)$*.*

Any parametric polynomial $\pi(\mathbf{x},\mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ can be represented using Bernstein basis as:

$$\pi(\mathbf{x},\mathbf{p}) = \sum_{\mathbf{i}\in I^\pi} \mathbf{b}_\mathbf{i}(\mathbf{p})\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) \tag{19}$$

where, for each $\mathbf{i} \in I^\pi$, the *parametric Bernstein coefficient*, is defined as:

$$\mathbf{b}_\mathbf{i}(\mathbf{p}) = \sum_{\mathbf{j}\leq\mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}}\mathbf{a}_\mathbf{j}(\mathbf{p}). \tag{20}$$

The parametric Bernstein coefficients are functions of the form $\mathbf{b}_\mathbf{i}(\mathbf{p}) : \mathbb{R}^m \to \mathbb{R}$. However, the $\mathbf{b}_\mathbf{i}(\mathbf{p})$ have a linear dependence on $\mathbf{p}$.

Example 11. *Consider again the parametric polynomial presented in Example 9, i.e.,* $\pi(\mathbf{x}_1,\mathbf{x}_2,\mathbf{p}_1) = 1/3\mathbf{p}_1\mathbf{x}_1^2 - 1/2\mathbf{p}_1\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$*. For illustrative purposes, we study only the multi-index* $(1,1)$*, the corresponding parametric*

Bernstein coefficient is:

$$\begin{aligned}
\mathbf{b}_{(1,1)}(\mathbf{p}_1) =& \frac{\binom{(1,1)}{(1,1)}}{\binom{(2,1)}{(1,1)}}\mathbf{a}_{(1,1)}(\mathbf{p}_1) + \frac{\binom{(1,1)}{(0,1)}}{\binom{(2,1)}{(0,1)}}\mathbf{a}_{(0,1)}(\mathbf{p}_1) \\
&\qquad + \frac{\binom{(1,1)}{(1,0)}}{\binom{(2,1)}{(1,0)}}\mathbf{a}_{(1,0)}(\mathbf{p}_1) + \frac{\binom{(1,1)}{(0,0)}}{\binom{(2,1)}{(0,0)}}\mathbf{a}_{(0,0)}(\mathbf{p}_1) \\
=& \frac{\binom{1}{1}\binom{1}{1}}{\binom{2}{1}\binom{1}{1}}\mathbf{a}_{(1,1)}(\mathbf{p}_1) + \frac{\binom{1}{0}\binom{1}{1}}{\binom{2}{0}\binom{1}{1}}\mathbf{a}_{(0,1)}(\mathbf{p}_1) \\
&\qquad + \frac{\binom{1}{1}\binom{1}{0}}{\binom{2}{1}\binom{1}{0}}\mathbf{a}_{(1,0)}(\mathbf{p}_1) + \frac{\binom{1}{0}\binom{1}{0}}{\binom{2}{0}\binom{1}{0}}\mathbf{a}_{(0,0)}(\mathbf{p}_1) \\
=& \frac{\binom{1}{1}}{\binom{2}{1}}\mathbf{a}_{(1,1)}(\mathbf{p}_1) + \frac{\binom{1}{0}}{\binom{2}{0}}\mathbf{a}_{(0,1)}(\mathbf{p}_1) \\
&\qquad + \frac{\binom{1}{1}}{\binom{2}{1}}\mathbf{a}_{(1,0)}(\mathbf{p}_1) + \frac{\binom{1}{0}}{\binom{2}{0}}\mathbf{a}_{(0,0)}(\mathbf{p}_1).
\end{aligned}$$

Since $\binom{n}{0} = 1$ *and* $\binom{n}{1} = n$ *for any* $n \in \mathbb{N}$,

$$\mathbf{b}_{(1,1)}(\mathbf{p}_1) = \frac{1}{2}\mathbf{a}_{(1,1)}(\mathbf{p}_1) + 1\mathbf{a}_{(0,1)}(\mathbf{p}_1) + \frac{1}{2}\mathbf{a}_{(1,0)}(\mathbf{p}_1) + 1\mathbf{a}_{(0,0)}(\mathbf{p}_1).$$

As noticed in Example 9, the non-null parametric coefficients of the investigated polynomial are $\mathbf{a}_{(2,0)}(\mathbf{p}_1) = \mathbf{p}_1/3$, $\mathbf{a}_{(0,1)}(\mathbf{p}_1) = -\mathbf{p}_1/2$, $\mathbf{a}_{(1,1)}(\mathbf{p}_1) = 1/4$, *and* $\mathbf{a}_{(0,0)}(\mathbf{p}_1) = 1/2$. *Thus,*

$$\begin{aligned}
\mathbf{b}_{(1,1)}(\mathbf{p}_1) =& \frac{1}{2}\frac{1}{4} + 1(-\mathbf{p}_1/2) + \frac{1}{2}0 + 1\frac{1}{2} \\
=& \frac{1}{8} + \frac{4}{8} - \mathbf{p}_1/2 = 5/8 - \mathbf{p}_1/2.
\end{aligned}$$

Analogously, we obtain the parametric Bernstein coefficients $\mathbf{b}_{(0,0)}(\mathbf{p}_1) = 1/2$, $\mathbf{b}_{(0,1)}(\mathbf{p}_1) = (1 - \mathbf{p}_1)/2$, $\mathbf{b}_{(1,0)}(\mathbf{p}_1) = 1/2$, $\mathbf{b}_{(2,0)}(\mathbf{p}_1) = \mathbf{p}_1/3 + 1/2$, *and* $\mathbf{b}_{(2,1)}(\mathbf{p}_1) = 3/4 - \mathbf{p}_1/6$.

Since the parametric polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1)$ *has the degree* $\mathbf{d} = (2, 1)$, *its*

Bernstein representation is:

$$
\begin{aligned}
\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1) =& \mathbf{b}_{(0,0)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(0,0))}(\mathbf{x}) + \mathbf{b}_{(0,1)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(0,1))}(\mathbf{x}) \\
&+ \mathbf{b}_{(1,0)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(1,0))}(\mathbf{x}) + \mathbf{b}_{(1,1)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(1,1))}(\mathbf{x}) \\
&+ \mathbf{b}_{(2,0)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(2,0))}(\mathbf{x}) + \mathbf{b}_{(2,1)}(\mathbf{p}_1)\mathcal{B}_{(\mathbf{d},(2,1))}(\mathbf{x}) \\
=& \frac{1}{2}\left((1-\mathbf{x}_1)^2(1-\mathbf{x}_2)\right) + \frac{1-\mathbf{p}_1}{2}\left((1-\mathbf{x}_1)^2\mathbf{x}_2\right) \\
&+ \frac{1}{2}(2\mathbf{x}_1(1-\mathbf{x}_1)(1-\mathbf{x}_2)) + \left(\frac{5}{8} - \frac{\mathbf{p}_1}{2}\right)(2\mathbf{x}_1(1-\mathbf{x}_1)\mathbf{x}_2) \\
&+ \left(\frac{\mathbf{p}_1}{3} + \frac{1}{2}\right)(\mathbf{x}_1^2(1-\mathbf{x}_2)) + \left(\frac{3}{4} - \frac{\mathbf{p}_1}{6}\right)(\mathbf{x}_1^2\mathbf{x}_2).
\end{aligned}
$$

We now extend the range enclosing property of Bernstein coefficients to the parametric case. Let $P \subset \mathbb{R}^m$ be a polytope.

Lemma 2 (Parametric Range Enclosing [27])**.**

$$\min_{\mathbf{i}\in I^\pi} \min_{\mathbf{p}\in P} \mathbf{b}_{\mathbf{i}}(\mathbf{p}) \leq \pi(\mathbf{x}, \mathbf{p}) \leq \max_{\mathbf{i}\in I^\pi} \max_{\mathbf{p}\in P} \mathbf{b}_{\mathbf{i}}(\mathbf{p}), \tag{21}$$

for all $\mathbf{x} \in [0,1]^n$ *and* $\mathbf{p} \in P$*, where* $\mathbf{b}_{\mathbf{i}}(\mathbf{p})$*, for* $\mathbf{i} \in I^\pi$*, are the parametric Bernstein coefficients of* π*.*

Proof. Since we let $\mathbf{p}$ linearly range over P, this is an immediate consequence of the range enclosing property of Bernstein coefficients that states that the minimum and maximum Bernstein coefficients of a non-parametric polynomial π are a lower bound and an upper bound of the image of π over the unit box domain, respectively [14]. □

Example 12. *Let us consider once more the parametric polynomial described by Example 9 and assume that* $\mathbf{p}_1$ *ranges in* $P = [0, 10]$*. Since* $\mathbf{p}_1$ *is linear in* $\mathbf{b}_{\mathbf{i}}(\mathbf{p}_1)$*, either* $\mathbf{b}_{\mathbf{i}}(0) = \min_{\mathbf{p}_1\in P}(\mathbf{b}_{\mathbf{i}}(\mathbf{p}_1))$ *and* $\mathbf{b}_{\mathbf{i}}(10) = \max_{\mathbf{p}_1\in P}(\mathbf{b}_{\mathbf{i}}(\mathbf{p}_1))$ *or* $\mathbf{b}_{\mathbf{i}}(10) = \min_{\mathbf{p}_1\in P}(\mathbf{b}_{\mathbf{i}}(\mathbf{p}_1))$ *and* $\mathbf{b}_{\mathbf{i}}(0) = \max_{\mathbf{p}_1\in P}(\mathbf{b}_{\mathbf{i}}(\mathbf{p}_1))$*. By Lemma 2, the polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1)$ *ranges in the interval between* $\min_{\mathbf{i}\in I^\pi}\{\mathbf{b}_{\mathbf{i}}(0), \mathbf{b}_{\mathbf{i}}(10)\}$ *and* $\max_{\mathbf{i}\in I^\pi}\{\mathbf{b}_{\mathbf{i}}(0), \mathbf{b}_{\mathbf{i}}(10)\}$ *when* $\mathbf{x} \in [0,1]^2$*. Moreover, because of the values of the Bernstein coefficients evaluated in Example 11,* $\min_{\mathbf{i}\in I^\pi}\{\mathbf{b}_{\mathbf{i}}(0), \mathbf{b}_{\mathbf{i}}(10)\} = \mathbf{b}_{(0,1)}(10) = -9/2$ *and* $\max_{\mathbf{i}\in I^\pi}\{\mathbf{b}_{\mathbf{i}}(0), \mathbf{b}_{\mathbf{i}}(10)\} = \mathbf{b}_{(2,0)}(10) = 23/6$*. Thus, the image of* $\mathbf{x} \in [0,1]^2$ *through the polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1)$ *is contained by* $[-9/2, 23/6]$ *(see Fig. 7).*

2.4. Single Step Reachability Algorithm

In [30], we presented an algorithm for single-step reachability in the case of polytopes and polynomial functions without parameters. In particular, we exploited Bernstein coefficients to compute a polytope that over-approximates the image of a given polytope produced by a polynomial transformation. The idea can be generalized to the case of parametric polynomial functions provided that one can determine the minimum and the maximum of a parametric Bernstein

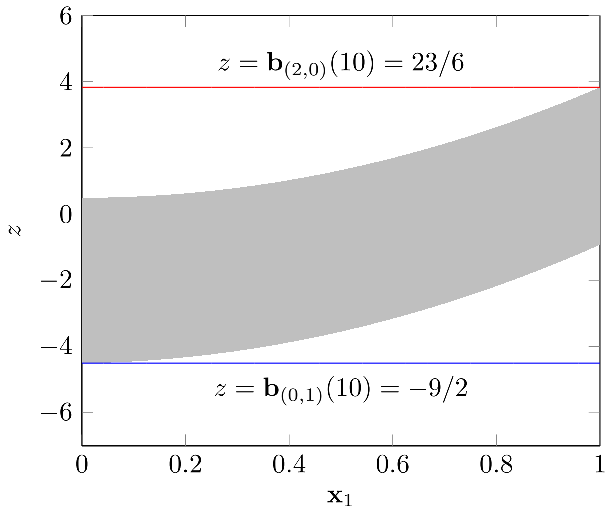


Figure 7: The projection of $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1) = \mathbf{p}_1 * \mathbf{x}_1^2/3 - \mathbf{p}_1 * \mathbf{x}_2/2 + \mathbf{x}_1 * \mathbf{x}_2/4 + 1/2$ on the plane $\mathbf{x}_2 = 0$ for $\mathbf{x} \in [0,1]^2$ and $\mathbf{p}_1 \in [0,10]$. Under these conditions, $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1)$ lays in the interval $[-9/2, 23/6]$ for all $\mathbf{x} \in [0,1]^2$. See Example 9, 10, 11, and 12 for all the details.

coefficient over the set P of parameters. We recall here the main ingredients of the procedure and refer the reader to [27] for all the details.

Let us start from the trivial case of the unit box $[0,1]^n$. So let $X = [0,1]^n$, P be a polytope, $\mathbf{f} = (\mathbf{f}_1, \dots, \mathbf{f}_n)$ be polynomial, and proceed as follows:

1. for each $j = 1, \dots, n$ compute the Bernstein coefficients $\mathbf{b}_\mathbf{i}(\mathbf{p})$ of the polynomial $\mathbf{f}_j$, with $\mathbf{i} \in I^{\mathbf{f}_j}$;
2. for each Bernstein coefficient $\mathbf{b}_\mathbf{i}(\mathbf{p})$ of $\mathbf{f}_j$ determine the minimum $min_j(\mathbf{i}) = \min_{\mathbf{p}\in P} \mathbf{b}_\mathbf{i}(\mathbf{p})$ and the maximum $max_j(\mathbf{i}) = \max_{\mathbf{p}\in P} \mathbf{b}_\mathbf{i}(\mathbf{p})$. This is possible since we are bounding a linear function over a polytope;
3. for each $j = 1, \dots, n$ consider the minimum $min_j = \min_{\mathbf{i}\in I^{\mathbf{f}_j}} min_j(\mathbf{i})$ and the maximum $max_j = \max_{\mathbf{i}\in I^{\mathbf{f}_j}} max_j(\mathbf{i})$.

The output box, which over-approximates $\mathbf{f}([0,1]^n, P)$, is $[min_1, max_1] \times \cdots \times [min_n, max_n]$. The reader should be aware that the parameter selection that minimizes or maximizes one of the Bernstein coefficients may differ from that that minimizes or maximizes another coefficient. So, while the returned box certainly over-approximates the set reachable in one step, the larger the parameter set, the coarser the over-approximation.

Example 13. *Let us consider the SIR model of Example 1. Let S, I, and R range in $[0,1]$, while $\beta \in [0.1, 0.2]$ and $\alpha \in [0.5, 0.6]$. The Bernstein coefficients for the dynamic law of S are in $\{0, 1, 1-\beta\}$, while those for I are in $\{0, 1-\alpha, 1+\beta-\alpha\}$, and those for R are in $\{0, 1, \alpha, 1+\alpha\}$. Notice that some coefficients are repeated more than once for each function, e.g., 0. However, since we are* *only interested in minimum and maximum values, the repetitions are irrelevant. The minimum coefficient for S is 0, while the maximum is 1. The minimum value for I is 0 and the maximum is 0.7 ($1 + \beta - \alpha = 0.7$ when $\beta = 0.2$ and $\alpha = 0.5$). Finally, the minimum value for R is 0 and the maximum is 1.6 ($1 + \alpha = 1.6$ when $\alpha = 0.6$). So, the set reached after one step is a subset of $[0, 1] \times [0, 0.7] \times [0, 1.6]$. It is worth noticing that the values for α that maximize I and R –i.e., 0.5 and 0.6, respectively– are different.*

If the parameters set is $\beta \in [0.1, 0.15]$ and $\alpha \in [0.5, 0.55]$, then the symbolic Bernstein coefficient sets are the same we computed above. Thus, the minima for S, I, and R and the maxima for S do not change, whereas the maxima for I and R are 0.65 ($1+\beta-\alpha = 0.65$ when $\beta = 0.15$ and $\alpha = 0.5$) and 1.55 ($1+\alpha = 1.55$ when $\alpha = 0.55$), respectively. So, when $\beta \in [0.1, 0.15]$ and $\alpha \in [0.5, 0.55]$, the parallelotope $[0, 1] \times [0, 0.65] \times [0, 1.55]$ is an over-approximation of the reachable set in one step from the unit box.

In both the cases, if we want to compute a further reachability step, we cannot reuse the above-described method because the initial set is no more a unit box. The next paragraph presents a generalization of the method that deals with any kind of parallelotopes.

If we are instead interested in bounding the image of a parallelotope $\langle \Lambda, \mathbf{c} \rangle$ we have to first rely on its generator representation, e.g., following these steps:

1. compute the generator representation of $\langle \Lambda, \mathbf{c} \rangle$. Let $\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n)$ be the linear function over the unit box that occurs in such representation;
2. for each $k = 1, \dots, 2n$ let Λ_k be k-th row of the template matrix Λ. Consider the function $\Lambda_k(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n), \mathbf{p}))$ over $[0, 1]^n \times P$;
3. determine the Bernstein coefficients of $\Lambda_k(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n), \mathbf{p}))$. Identify the maximum c'_k of such coefficients over P. Since both $\gamma_{(\mathbf{q},G)}$ and Λ_k are linear, in the function $\Lambda_k(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n), \mathbf{p}))$ the dependence on $\mathbf{p}$ is still linear and the maximum can be found exploiting linear optimization techniques. Moreover, the property of Bernstein coefficients ensures that $\Lambda_k(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n), \mathbf{p})) \leq \mathbf{c}'_k$ over $[0, 1]^n \times P$, and hence, $\Lambda_k(\mathbf{f}(\mathbf{x}, \mathbf{p})) \leq c'_k$ over $\langle \Lambda, \mathbf{c} \rangle \times P$.

The output parallelotope which over-approximates $\mathbf{f}(\langle \Lambda, \mathbf{c} \rangle, P)$ is $\langle \Lambda, \mathbf{c}' \rangle$, where $\mathbf{c}' = (\mathbf{c}'_1, \dots, \mathbf{c}'_{2n})$. The procedure is sketched in Algorithm 2.

Algorithm 2 Parallelotope-based reachability step

```
1: function REACHSTEP($X = \langle \Lambda, \mathbf{c} \rangle, P, \mathbf{f}$)
2:     for $k \in \{1, \dots, 2n\}$ do
3:         $\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n) \leftarrow$ CON2GEN($X$)    ▷ Compute generator function
4:         $\mathbf{c}'_k \leftarrow$ MAXBERNCOEFF($\Lambda_k(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \dots, \lambda_n), \mathbf{p})), P$)    ▷ Compute maximum coefficient
5:     end for
6:     return $X' = \langle \Lambda, \mathbf{c}' \rangle$
7: end function
```

Example 14. *Let us now compute another step from the set reached in Example 13. We are considering the set* $[0,1]\times[0,0.7]\times[0,1.6]$ *that can be represented by the parallelotope* $\{\mathbf{x} \mid \Lambda\mathbf{x}^T \leq \mathbf{c}\}$ *where*

$$\Lambda = \begin{pmatrix} -1 & 0 & 0 \\ 0 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \qquad \mathbf{c} = \begin{pmatrix} 0 \\ 0 \\ 0 \\ 1 \\ 0.7 \\ 1.6 \end{pmatrix}.$$

This means that $\mathbf{q} = (0,0,0)$ *and* $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3) = (\lambda_1, 0.7\lambda_2, 1.6\lambda_3)$*. The Bernstein coefficients of the polynomials in the variables* $\lambda_1,\lambda_2,\lambda_3$ *are:*

$$\begin{aligned} \rho_S &= \lambda_1 - 0.7\beta\lambda_1\lambda_2 \\ \rho_R &= 0.7\lambda_2 + 0.7\beta\lambda_1\lambda_2 - 0.7\alpha\lambda_2 \\ \rho_I &= 1.6\lambda_3 + 0.7\alpha\lambda_2 \end{aligned}$$

The Bernstein coefficients of ρ_S *are in* $\{0,1,1-0.7\beta\}$*, those of* ρ_R *are in* $\{0, 0.7(1-\alpha), 0.7(1+\beta-\alpha)\}$*, and those of* ρ_I *are in* $\{0, 1.6, 0.7\alpha, 1.6+0.7\alpha\}$*. By minimizing and maximizing them over* $\beta \in [0.1, 0.2]$ *and* $\alpha \in [0.5, 0.6]$ *we get the new intervals for* S*,* I*, and* R*, i.e.,* $S \in [0,1]$*,* $I \in [0, 0.49]$*, and* $R \in [0, 2.02]$*.*

If the initial set is $(S,I,R) \in [0,1]\times[0,0.65]\times[0,1.55]$*,* $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3)$ *is* $(\lambda_1, 0.65\lambda_2, 1.55\lambda_3)$ *and the Bernstein coefficients sets for the dynamic laws of* S*,* I*, and* R *are* $\{0,1,1-0.65\beta\}$*,* $\{0, 0.65(1-\alpha), 0.65(1+\beta-\alpha)\}$*, and* $\{0, 1.55, 0.65\alpha, 1.55+0.65\alpha\}$*, respectively. Thus, if* $\beta \in [0.1, 0.15]$ *and* $\alpha \in [0.5, 0.55]$*,* (S,I,R) *ranges in* $[0,1]\times[0,0.4225]\times[0,1.9075]$*.*

In the case of polytopes, the bundle decomposition allows us to efficiently solve the problem of defining a function from $[0,1]^n$ to a generic polytope Q. However, this is crucial only for transforming the input, while the directions for computing the new offset vector can be changed or combined differently in the bundles during the computation. Consequently, in [30], we described two different strategies for computing the new bundle. One of them is faster to compute, while the other is the more precise of the two. The above considerations, that allow us to generalize the methods presented in [30] to the parametric case, also apply to such strategies [27].

The presented algorithms correctly compute an over-approximation of the reachable set, as proved by Lemma 2. It is worth noticing that the proposed strategy is memory-less and the evaluated evolution from a point does not consider the parameter values that let that point be reached. While this approach simplifies the parametric reachability problem and, in some sense, makes it solvable, it produces an over-approximation of the solution that gets coarser and coarser as the evolution proceeds because, at each step, we admit any value in the parameter set as a valid choice for the parameter values.

Example 15. *Let us consider the classical SIR model (see Example 1) and its evolution from* $\mathbf{x}_0 = (0.99, 0.01, 0.0)$ *when* $\alpha \in \{1/5, 4/5\}$ *and* $\beta = 1/2$*.*

Since α and β are parameters, their values should be fixed once for all, thus, $\xi^{(1/5,1/2)}_{\mathbf{x}_0}$ and $\xi^{(4/5,1/2)}_{\mathbf{x}_0}$ are the only valid trajectories from $\mathbf{x}_0$ and, for instance, $\xi^{(1/5,1/2)}_{\mathbf{x}_0}(2) = (0.9786718, 0.0167382, 0.00459)$ and $\xi^{(4/5,1/2)}_{\mathbf{x}_0}(2) = (0.98162695, 0.00481305, 0.01356)$.

If we over-approximate the reachable set by avoiding to select α once for all and preserving the choice between $\alpha = 1/5$ and $\alpha = 2/5$ at each evolution step, we produce an infinite number of "spurious" trajectories and, in particular, in two steps we can reach $\xi^{(4/5,1/2)}_{\mathbf{x}_1}(1) = (0.9786718, 0.0089682, 0.01236)$ where $\mathbf{x}_1 = \xi^{(1/5,1/2)}_{\mathbf{x}_0}(1)$.

While the proposed algorithm exclusively deals with dense sets of parameters, it applies the just-mentioned approach to over-approximate the flowpipe.

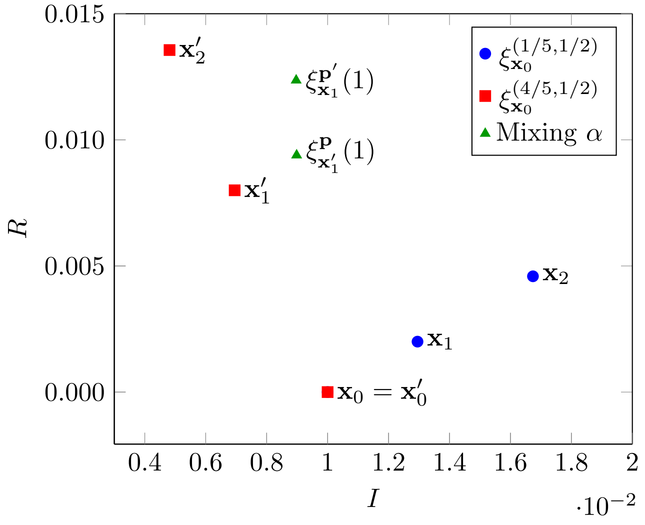


Figure 8: Each step of the reachability procedure admits any possibile value in the parameter set. Thus, when the parameter set is not a singleton, the algorithm follows also "spurious" trajectories along which the parameters changes. The trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ described in Example 1 for $\mathbf{p} = (1/5, 1/2)$ and $\mathbf{p}' = (4/5, 1/2)$. The points $\xi^{\mathbf{p}}_{\mathbf{x}_0}(i)$ and $\xi^{\mathbf{p}'}_{\mathbf{x}_0}(i)$ are labelled as $\mathbf{x}_i$ and $\mathbf{x}'_i$, respectively. The triangles depict two points that are included in the over-approximation of the reachable set exclusively because the algorithm selected two different values for α during the two-time step horizon.

As far as computational complexity is concerned, the most expensive part of the computation is the Bernstein coefficients' computation. However, since the template parallelotopes do not change during the evolution of the reach set, the coefficients can be symbolically computed once for all and instantiated at each step. This approach significantly reduces the complexity (see [27]).

3. Parameter Synthesis

The parameter synthesis problem requires handling flows of parametric traces that, in most cases, cannot be exactly computed, as we saw in the previous section. Therefore, we need to find a compromise between precision and tractability of the problem. For this reason, we introduce the notion of approximated sets of trajectories which allows us to deal with the synthesis problem.

Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a discrete-time dynamical system, $X_0 \subseteq \mathcal{X}$, and $P \subseteq \mathcal{P}$. The set of trajectories $\Xi(X_0, P)$ of $\mathcal{D}$ can be over-approximated by the flowpipe

$$\mathcal{W}_{(X_0,P)} = \{X_t \mid t \in \mathbb{N} \text{ and for } t > 0 \text{ it holds } X_t = \mathbf{f}(X_{t-1}, P)\}.$$

The over-approximating flowpipe $\mathcal{W}_{(X_0,P)}$ can be computed with the set-integration algorithms presented in Section 2. However, it is worth noticing that this is an over-approximation of $\Xi(X_0, P)$ since the relationship between a single trace and its corresponding parameter is not kept.

We can define a semantics on flowpipes that reflects the parameter synthesis problem. In particular, we define the following *synthesis semantics.*

Definition 12 (Synthesis Semantics). *Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a discrete-time dynamical system, $\mathcal{W}_{(X_0,P)}$ be a flowpipe with $X_0 \subseteq \mathcal{X}$ and $P \subseteq \mathcal{P}$, $t \in \mathbb{N}$ be a time instant, and φ be an STL formula in positive normal form. The* synthesis semantics *$\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$ of φ at time t over the flowpipe $\mathcal{W}_{(X_0,P)}$ is given by the following inductive definition:*

$$\begin{aligned}
\Phi(\sigma, \mathcal{W}_{(X_0,P)}, t) &= P_\sigma, \textit{ where } P_\sigma \subseteq P \textit{ is the largest subset s.t.} \\
&\qquad \forall \mathbf{x}_0 \in \mathcal{W}_{(X_0,P)}(t), \forall \mathbf{p} \in P_\sigma, \sigma(\mathbf{f}(\mathbf{x}_0, \mathbf{p})) \textit{ is true} \\
\Phi(\varphi_1 \wedge \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t) \cap \Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t) \\
\Phi(\varphi_1 \vee \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t) \cup \Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t) \\
\Phi(\varphi_1 U_I \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \bigcup_{t' \in t+I} (\Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t') \cap \\
&\qquad \bigcap_{t'' \in [t,t']} \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t''))
\end{aligned} \tag{22}$$

From an intuitive point of view, $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$ returns a subset $P_\varphi \subseteq P$ of parameters that ensures that φ is satisfied at time t starting from any point in X_0 and assigning to the parameters any value in P_φ. Again, the synthesis semantics at time t returns a set P_φ that steers the system from time $t + 1$ on. This is slightly counterintuitive since the semantics usually returns evaluations referring to the time instant in which they are applied. However, it is consistent with our choice on the semantics of STL.

We say that a flowpipe $\mathcal{W}_{(X_0,P)}$ *satisfies* a formula φ, denoted with $\mathcal{W}_{(X_0,P)} \models \varphi$ if and only if $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t) = P$. By using Definition 12 and structural induction on φ, it is easy to see that $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$ is idempotent on P, i.e., $\Phi(\varphi, \mathcal{W}_{(X_0,P_\varphi)}, t) = P_\varphi$ where $P_\varphi = \Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$. Thus, $\mathcal{W}_{(X_0,P_\varphi)} \models \varphi$. This statement, which is formally reported in the following theorem, asserts that the synthesis semantics provides a refined set of parameters that satisfies the formula φ.

Theorem 1. *If $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0) = P_\varphi$, then $\mathcal{W}_{(X_0,P_\varphi)} \models \varphi$.*

Since $\mathcal{W}_{(X_0,P)}$ over-approximates $\Xi(X_0, P_\varphi)$ with $P_\varphi \subseteq P$, if φ is satisfied by $\mathcal{W}_{(X_0,P)}$, then it is satisfied also by $\Xi(X_0, P_\varphi)$. The following lemma shows that the parameters generated by the synthesis semantics are also valid for the exact trajectories of a dynamical system.

Lemma 3. *If $\Phi(\varphi, \mathcal{W}_{(X,P)}, t) = P_\varphi$, then for each $\mathbf{x} \in X$ and for each $\mathbf{p} \in P_\varphi$ it holds that $\xi^{\mathbf{p}}_{\mathbf{x}}, t \models \varphi$.*

Proof. By structural induction on φ.

- (σ) Let $\mathbf{p} \in P_\sigma, \mathbf{x} \in X$, $\mathbf{x}_{t-1} = \xi^{\mathbf{p}}_{\mathbf{x}}(t-1)$, and $X_{t-1} = \mathcal{W}_{(X,P)}(t-1)$. Since $P_\sigma \subseteq P$ we have that $\mathbf{x}_{t-1} \in X_{t-1}$ and $\mathbf{x}_t = \mathbf{f}(\mathbf{x}_{t-1}, \mathbf{p}) \in \mathbf{f}(X_{t-1}, P_\sigma)$. Hence, by definition of P_σ we have that $\sigma(\mathbf{x}_t)$ is true, i.e., the thesis;

- ($\varphi_1 \wedge \varphi_2$) Let $\Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t) = P_{\varphi_1}$ and $\Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t) = P_{\varphi_2}$. We have that $P_\varphi = P_{\varphi_1} \cap P_{\varphi_2}$. Let $\mathbf{p} \in P_\varphi$ and $\mathbf{x} \in X$, since $\mathbf{p}$ belongs to both P_{φ_1} and P_{φ_2}, by inductive hypothesis we have that $\xi^{\mathbf{p}}_{\mathbf{x}}(t) \models \varphi_1$ and $\xi^{\mathbf{p}}_{\mathbf{x}}(t) \models \varphi_2$. Hence, we get the thesis;

- ($\varphi_1 \vee \varphi_2$) Similar to the conjunction;

- ($\varphi_1 U_I \varphi_2$) By definition, if $\mathbf{p} \in P_\varphi$, there exists $t' \in t + I$ such that $\mathbf{p} \in P^{t'}_{\varphi_2} \cap \bigcap_{t'' \in [t,t']} P^{t''}_{\varphi_1}$, where $P^{t'}_{\varphi_2} = \Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t')$ and $P^{t''}_{\varphi_1} = \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t'')$. Hence, by inductive hypothesis on φ_1 and φ_2, we get the thesis.

□

As a consequence, we are correctly under-approximating the original parameter synthesis problem, as stated by the following theorem.

Theorem 2. *If $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0) = P_\varphi$, then $\Xi(X_0, P_\varphi) \models \varphi$.*

Proof. This is an immediate consequence of Lemma 3. □

In summary, we proved that all the trajectories starting in X_0 and having parameters in P_φ, where P_φ is the result of the synthesis semantics of φ on the flowpipe $\mathcal{W}_{(X_0,P)}$, satisfy the formula φ. Thus, the synthesis semantics can be used to produce under-approximations of the parameter synthesis problem (see Definition 4).

In the rest of this section, we will see how to synthesize a set P' such that $\Xi(X_0, P') \models \varphi$ in the case of the domains and systems defined in Section 2. Our goal will be achieved by exploiting the synthesis semantics and, in particular, $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0)$. Even though this last set would be the largest subset of P satisfying the specification, it will be enough to under-approximate it, rather than exactly compute it, because, if $\Xi(X_0, P) \models \varphi$, then $\Xi(X_0, P') \models \varphi$ too for any subset P' of P.

We will proceed by induction on the structure of φ and rely on the main procedure described in Algorithm 3 which takes in input a set of states $X_0 \subseteq \mathcal{X}$, a set of parameters $P \subseteq \mathcal{P}$, a function $\mathbf{f}$ describing the evolution of the system, and an STL formula φ in positive normal form, and produces a refinement $P_\varphi \subseteq P$.

Algorithm 3 Parameter synthesis.

```
function ParaSynth($X_0, P, \mathbf{f}, \varphi$)
    if $\varphi = \sigma$ then                                              ▷ Predicate
        return RefPredicate($X_0, P, \mathbf{f}, \sigma$)
    end if
    if $\varphi = \varphi_1 \wedge \varphi_2$ then                                     ▷ Conjunction
        return ParaSynth($X_0, P, \mathbf{f}, \varphi_1$) $\cap$ ParaSynth($X_0, P, \mathbf{f}, \varphi_2$)
    end if
    if $\varphi = \varphi_1 \vee \varphi_2$ then                                      ▷ Disjunction
        return ParaSynth($X_0, P, \mathbf{f}, \varphi_1$) $\cup$ ParaSynth($X_0, P, \mathbf{f}, \varphi_2$)
    end if
    if $\varphi = \varphi_1 U_{\mathcal{I}} \varphi_2$ then                                          ▷ Until
        return UntilSynth($X_0, P, \mathbf{f}, \varphi_1 U_I \varphi_2$)
    end if
end function
```

3.1. Basic Refinement Step $\varphi \equiv \sigma$

Let us consider the case of sets of points in the domain of parallelotopes, set of parameters in the domain of polytopes and polynomial functions with a linear dependence on the parameters. Let us also assume that $\sigma \equiv s(\mathbf{x}) \leq 0$, where s is a linear function. In this case, similarly to what has been done in Section 2.4, we can compute the Bernstein coefficients that are going to be linear in the parameters. By imposing the parametric Bernstein coefficients to be nonpositive, we obtain a set of new linear constraints that we add to the definition of the set of parameters. Specifically, if $X_0 = \langle \Lambda, \mathbf{c} \rangle$, the RefPredicate procedure operates as follows:

1. compute the generator representation of $\langle \Lambda, \mathbf{c} \rangle$. Let $\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n)$ be the linear function over the unit box that occurs in such representation;
2. consider the function $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n), \mathbf{p}))$. The image of this function over $[0, 1]^n \times P$ coincides with the image of $s(\mathbf{f}(\mathbf{x}, \mathbf{p}))$ over $\langle \Lambda, \mathbf{c} \rangle \times P$;
3. determine the Bernstein coefficients $\mathbf{b_i}(\mathbf{p})$ of $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1, \ldots, \lambda_n), \mathbf{p}))$;
4. add to P all the constraints of the form $\mathbf{b_i}(\mathbf{p}) \leq 0$.

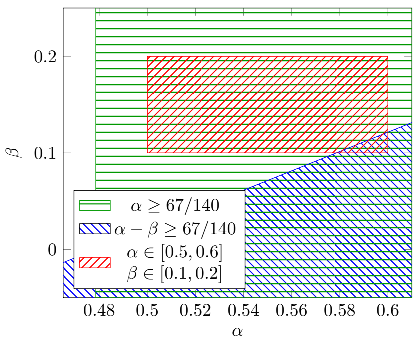
We now present two examples, dealing with different formulas and two sets of initial conditions, to clarify the RefPredicate procedure.

Example 16. *Let us consider the SIR model described by Example 1 and let us synthesize a set of parameters satisfying the STL formula*

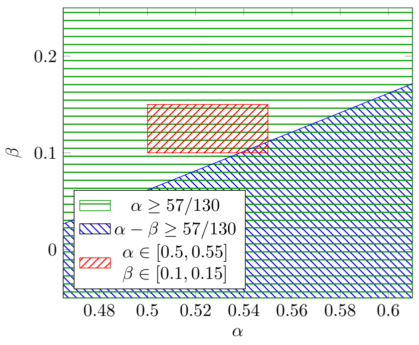
$$I \leq 0.365$$

when $S \in [0,1]$, $I \in [0,0.7]$, $R \in [0,1.6]$, $\beta \in [0.1,0.2]$ *and* $\alpha \in [0.5,0.6]$. *First of all, the generator for* $[0,1] \times [0,0.7] \times [0,1.6]$, *i.e.,* $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3) = (\lambda_1, 0.7\lambda_2, 1.6\lambda_3)$, *is found (see the first part of Example 14). Then, the algorithm computes* $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\ldots,\lambda_n),\mathbf{p}))$, *where* $s(\mathbf{x},\mathbf{p}) \leq 0$ *is the constraint to be satisfied. In this example,* $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\ldots,\lambda_n),\mathbf{p}))$ *is* $0.7(\lambda_2 + \beta\lambda_1\lambda_2 - \alpha\lambda_2) - 0.365$ *(see the first part of Example 14). Finally, the algorithm finds* $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\ldots,\lambda_n),\mathbf{p}))$ *Bernstein coefficients, i.e.,* $\{-0.365, 0.335 - 0.7\alpha, 0.335 + 0.7\beta - 0.7\alpha\}$, *and refines the parameter set by using the constraints* $\mathbf{b_i}(\mathbf{p}) \leq 0$, *for each coefficient* $\mathbf{b_i}(\mathbf{p})$. *Thus, the synthesized parameter set is such that* $\alpha \in [0.5,0.6]$, $\beta \in [0.1,0.2]$, $-0.365 \leq 0$, $0.335 - 0.7\alpha \leq 0$, *and* $0.335 + 0.7\beta - 0.7\alpha \leq 0$, *which is equivalent to* $\alpha \leq 0.6$, $\beta \geq 0.1$, *and* $\alpha - \beta \geq 67/140$.

If, instead, the initial conditions are $S \in [0,1]$, $I \in [0,0.65]$, $R \in [0,1.55]$, $\beta \in [0.1,0.15]$ *and* $\alpha \in [0.5,0.55]$, $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3) = (\lambda_1, 0.65\lambda_2, 1.55\lambda_3)$ *(see Example 14) and* $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\ldots,\lambda_n),\mathbf{p}))$ *is* $0.65(\lambda_2 + \beta\lambda_1\lambda_2 - \alpha\lambda_2) - 0.365$. *Thus, the corresponding Bernstein coefficients are* -0.365, $0.285 - 0.65\alpha$, *and* $0.285 + 0.65\beta - 0.65\alpha$ *and the algorithm returns the set such that* $\alpha \in [0.5,0.55]$, $\beta \in [0.1,0.15]$, $-0.365 \leq 0$, $0.285 - 0.65\alpha \leq 0$, *and* $0.285 + 0.65\beta - 0.65\alpha \leq 0$, *which are equivalent to* $\alpha \leq 0.55$, $\beta \geq 0.1$, *and* $\alpha - \beta \geq 57/130$.



(a) The synthesized set when $S \in [0,1]$, $I \in [0,\ 0.7]$, $R \in [0,1.6]$, $\beta \in [0.1,0.2]$ and $\alpha \in [0.5,0.6]$.



(b) The synthesized set when $S \in [0,1]$, $I \in [0,\ 0.65]$, $R \in [0,1.55]$, $\beta \in [0.1,0.15]$ and $\alpha \in [0.5,0.55]$.

Figure 9: The synthesized parameter set for the SIR model presented in Example 1 satisfying the formula $I \leq 0.365$. See Example 16 for the details.

Example 17. *Let us consider the SIR model described by Example 1 and let us synthesize a set of parameters satisfying the STL formula*

$$R \leq 2$$

when $S \in [0,1]$, $I \in [0,0.7]$, $R \in [0,1.6]$, $\beta \in [0.1,0.2]$ and $\alpha \in [0.5,0.6]$.

The generator representation $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3)$ equals $(\lambda_1, 0.7\lambda_2, 1.6\lambda_3)$, and $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\dots,\lambda_n),\mathbf{p}))$ is $1.6\lambda_3+0.7\alpha\lambda_2-2$ (see the first part of Example 14). The Bernstein coefficients of $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\dots,\lambda_n),\mathbf{p}))$ are -2,-0.4, $0.7\alpha-2$, and $0.7\alpha-0.4$ and the algorithm returns the set such that $\alpha \in [0.5,4/7]$ and $\beta \in [0.1,0.2]$.

If, instead, the initial conditions are $S \in [0,1]$, $I \in [0,0.65]$, $R \in [0,1.55]$, $\beta \in [0.1,0.15]$ and $\alpha \in [0.5,0.55]$, $\gamma_{(\mathbf{q},G)}(\lambda_1,\lambda_2,\lambda_3) = (\lambda_1, 0.65\lambda_2, 1.55\lambda_3)$ and $s(\mathbf{f}(\gamma_{(\mathbf{q},G)}(\lambda_1,\dots,\lambda_n),\mathbf{p}))$ is $1.55\lambda_3+0.65\alpha\lambda_2-2$ (see the last part of Example 14). Thus, the corresponding Bernstein coefficients are -2, -0.45, $0.65\alpha-2$, and $0.65\alpha-0.45$ and the original parameter set, $\alpha \in [0.5,0.55]$ and $\beta \in [0.1,0.15]$, is returned.

3.2. Conjunction and Disjunction Formulas

In the case of formulas of the form $\varphi_1 \wedge \varphi_2$, if $\textsc{ParaSynth}(X_0, P, \mathbf{f}, \varphi_1)$ and $\textsc{ParaSynth}(X_0, P, \mathbf{f}, \varphi_2)$ return two polytopes P_{φ_1} and P_{φ_2} defined as a set of linear constraints, it is sufficient to put together all the constraints and we obtain the set $P_{\varphi_1 \wedge \varphi_2}$.

On the other hand, in the case of formulas of the form $\varphi_1 \vee \varphi_2$, we have to accumulate the outputs of the two calls. A similar issue will occur in the case of the until operator described in the next subsection.

Example 18. *Let us synthesize the parameter set satisfying the STL formula*

$$I \leq 0.365 \wedge R \leq 2$$

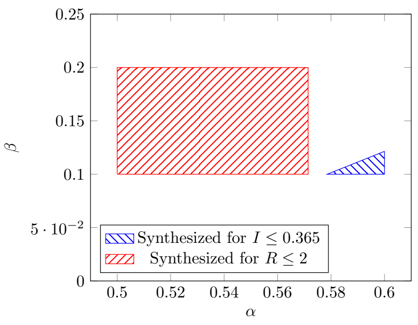
on the SIR model described by Example 1 when $S \in [0,1]$, $I \in [0,0.7]$, $R \in [0,1.6]$, $\beta \in [0.1,0.2]$ and $\alpha \in [0.5,0.6]$. Under these conditions, the synthesized set for $I \leq 0.365$ is such that $\alpha \leq 0.6$, $\beta \geq 0.1$, and $\alpha - \beta \geq 67/140$ (see the first part of Example 16), while that for $R \leq 2$ is such that $\alpha \in [0.5,4/7]$ and $\beta \in [0.1,0.2]$ (see the first part of Example 17). Their intersection is empty because $\alpha - \beta \geq 67/140$ is not even satisfied by the maximum of α, i.e., $4/7$, and the minimum for β, i.e., 0.1.

If, instead, the initial conditions are $S \in [0,1]$, $I \in [0,0.65]$, $R \in [0,1.55]$, $\beta \in [0.1,0.15]$ and $\alpha \in [0.5,0.55]$, then, the synthesized set for $I \leq 0.365$ is such that $\alpha \leq 0.55$, $\beta \geq 0.1$, and $\alpha - \beta \geq 57/130$ (see the last part of Example 16), while that for $R \leq 2$ is such that $\alpha \in [0.5,0.55]$ and $\beta \in [0.1,0.15]$ (see the last part of Example 17). Their intersection equals the former.

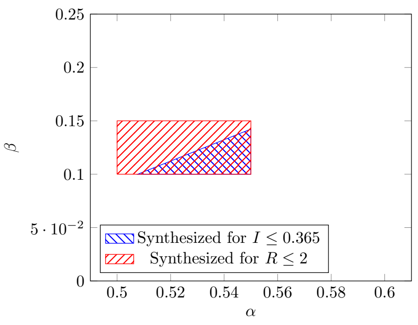
3.3. Until Formulas

The function $\textsc{UntilSynth}(X, P, \varphi_1 U_{[a,b]} \varphi_2)$ (Algorithm 4) refines the set P with respect to an until formula $\varphi_1 U_{[a,b]} \varphi_2$. It is structured in three main blocks, depending on the values a, b that define the interval of the until formula. The cases are the following:

1. $a > 0$ and $b > 0$: the interval is far from time 0;
2. $a = 0$ and $b > 0$: the interval starts at time 0 and ends somewhere else;



(a) The synthesized set when $S \in [0,1]$, $I \in [0,\ 0.7]$, $R \in [0,1.6]$, $\beta \in [0.1,0.2]$ and $\alpha \in [0.5,0.6]$.



(b) The synthesized set when $S \in [0,1]$, $I \in [0,\ 0.65]$, $R \in [0,1.55]$, $\beta \in [0.1,\ 0.15]$ and $\alpha \in [0.5,0.55]$.

Figure 10: The synthesized parameter set for the SIR model presented in Example 1 satisfying the formula $I \leq 0.365 \wedge R \leq 2$. See Example 18 for the details.

3. $a = 0$ and $b = 0$: the interval coincides with the single time instant 0.

Intuitively, the function UNTILSYNTH recursively transforms the cases 1 and 2 into the base case 3.

Before defining the algorithm, it is worth to point out that a single until formula $\varphi_1 U_{[a,b]} \varphi_2$ may require several basic refinements. Consider, for instance, the case where φ_1 always holds and φ_2 holds at several time instants inside $[a,b]$. Here, the number of basic refinements that $\varphi_1 U_{[a,b]} \varphi_2$ requires equals the number of time instants in which φ_2 holds. This means that an until can admit several valid refinements. Some of the refined parameter sets might be included in others, but since we do not know it in advance, we need to compute and accumulate all the possible solutions.

Let us now analyse the three cases constituting the structure of UNTILSYNTH (Algorithm 4):

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). Then the algorithm performs a reachability step using the valid parameter set P_{φ_1} to produce the new set X' (Line 4). Now the algorithm proceeds with the recursive call (Line 5). This can be seen as a step towards the interval $[a,b]$, except that instead of restoring the synthesis from time 1, we shift the interval backwards;
2. $a = 0$ and $b > 0$: there are two ways to satisfy the until formula:
   (a) φ_2 is satisfied right now at time 0;
   (b) φ_1 holds until φ_2 is satisfied before the time instant b.

   In the first case, we need to refine the parameter set with respect to φ_2. In the second case, the algorithm refines with respect to φ_1, then the procedure performs a reachability step under the refined parameter set P_{φ_1}, obtaining the new set X'. Similarly to the previous case, we execute

Algorithm 4 Until synthesis.

```
1: function UntilSynth($X_0, P, \mathbf{f}, \varphi_1 U_{[a,b]} \varphi_2$)
2:     if $a > 0$ and $b > 0$ then                                          ▷ Outside interval
3:         $P_{\varphi_1} \leftarrow$ ParaSynth($X_0, P, \mathbf{f}, \varphi_1$)               ▷ Check $\varphi_1$
4:         $X' \leftarrow$ ReachStep($X_0, P_{\varphi_1}, \mathbf{f}$)
5:         return UntilSynth($X', P_{\varphi_1}, \mathbf{f}, \varphi_1 U_{[a-1,b-1]} \varphi_2$)
6:     end if
7:     if $a = 0$ and $b > 0$ then                                          ▷ In interval
8:         $P_{\varphi_1} \leftarrow$ ParaSynth($X_0, P, \mathbf{f}, \varphi_1$)               ▷ Check $\varphi_1$
9:         $P_{\varphi_2} \leftarrow$ ParaSynth($X_0, P, \mathbf{f}, \varphi_2$)               ▷ Check $\varphi_2$
10:        $X' \leftarrow$ ReachStep($X_0, P_{\varphi_1}, \mathbf{f}$)
11:        return $P_{\varphi_2} \cup$ UntilSynth($X', P_{\varphi_1}, \mathbf{f}, \varphi_1 U_{[a,b-1]} \varphi_2$)
12:    end if
13:    if $a = 0$ and $b = 0$ then                                          ▷ Base
14:        return ParaSynth($X_0, P, \mathbf{f}, \varphi_2$)
15:    end if
16: end function
```

a step forward by shortening the interval by one. 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 P with respect to φ_2 and return P_{φ_2}.

Example 19. *Let us consider once more the original SIR model as presented by Example 1 when the initial values for S, I, and R range in $[0, 1]$, and the parameters α and β belong to $[0.1, 0.2]$ and $[0.5, 0.6]$, respectively. Let us synthesize the parameters α and β such that the STL formula*

$$F_{[1,1]} \left(I \leq 0.365 \wedge R \leq 2\right)$$

does hold. First of all, this formula is translated into the corresponding U-based formula, $\top U_{[1,1]} \left(I \leq 0.365 \wedge R \leq 2\right)$, as explained in Section 1.2. Then, Algorithm 4 performs the following steps:

1. *synthesize the parameters for the formula $\top$;*
2. *compute an over-approximation of the set reachable by using the evaluated parameters;*
3. *further restrict α and β by synthesizing their admissible values for the formula $\top U_{[0,0]} \left(I \leq 0.365 \wedge R \leq 2\right)$ by using as initial set that computed during step 2.*

The formula $\top$ always holds by definition, thus, step 1 does not change the parameter set. So, step 2 is equivalent to that reported in the first part of Example 13 and it computes the parallelotope $(S, I, R) \in [0, 1] \times [0, 0.7] \times [0, 1.6]$. Then, Algorithm 4 reduces step 3 to synthesizing a set for the formula $I \leq 0.365 \wedge R \leq 2$ by using the just-mentioned conditions as initial region. We know that the resulting set is empty from the first part of Example 18.

Instead if the initial parameter set is $\alpha \in [0.5, 0.55]$ and $\beta \in [0.1, 0.15]$, step 2 computes the set $(S, I, R) \in [0, 1] \times [0, 0.65] \times [0, 1.55]$ as detailed in the last part of Example 13. Thus, step 3 synthesizes a parameter set satisfying the formula $I \leq 0.365 \wedge R \leq 2$ when the initial conditions are those just-mentioned and $\beta \in [0.1, 0.15]$ and $\alpha \in [0.5, 0.55]$. As seeing in the last part of Example 18, the algorithm returns the non-empty set $\alpha \leq 0.55$, $\beta \geq 0.1$, and $\alpha - \beta \geq 57/130$.

The correctness of the above described algorithms in synthesizing a set of parameters that guarantees the STL specification is ensured by the correctness of the reachability algorithm that at each step over-approximates the set of reachable points together with Theorem 2.

As far as the computational complexity is concerned, in [22] it has been shown that the synthesis could require a number of reachability steps, basic refinements, unions and intersections which is exponential with respect to the dimension of the STL formula. The dimension of a formula includes both the number of operators and the size of the observed time horizon. Some scalability tests are available in [22, 27], where time intervals having length up to 120 and formulas with increasing nesting of until operators up to 20 have been considered on an Ebola system involving 5 variables and 6 parameters (see Section 6.2.4 and Tables 6.1 and 6.2 in [27]).

3.4. Parameter Splits

The described method guarantees the synthesis of a subset of parameters satisfying the specification over all the possible starting points. In many cases, we can obtain an empty set of parameters because of the over-approximations of the reached points accumulated during the computation. In particular, we do not store, for each point $\mathbf{x}$ in the reach set, the parameters that have allowed the model to flow to that very point and we propagate $\mathbf{x}$ in the future steps by using all the parameter values that are considered valid up to that step. This approach is hazardous when the initial set of parameters is large and produces an extremely conservative over-approximation of the real reach set.

Moreover, our procedure refines only the parameters directly involved in the dynamics of the variables occurring in the specification. Thus, when a parameter does not appear in the dynamic of a variable v occurring in the specification, but it rules the evolutions of other variables involved in the dynamic of v itself, we cannot refine the parameter set and we return an empty answer.

We take care of both issues by introducing an automatic split procedure over the parameters set. If we have not identified any parameter satisfying the specification, `sapo` resets the computation to time 0 and autonomously splits the set of parameters into two subsets over each of the hyperplanes defining it. For instance, we get four sub-rectangles in the case of two parameters ranging over a rectangle. Then, each subset is refined independently. The splitting step is iterated until either a non-empty parameter set is synthesized or the maximum number of iterations allowed by the user has been reached. Since each split exponentially increases the computational requirements, users should carefully choose a trade-off between efficiency and precision.

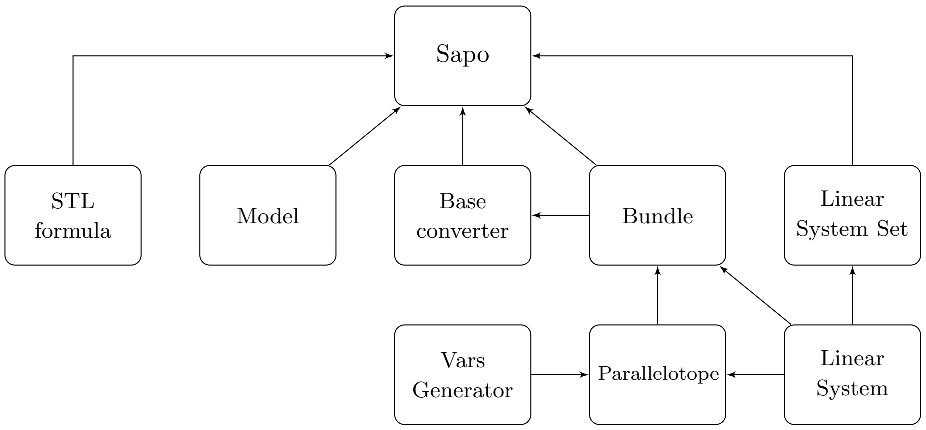


Figure 11: Tool architecture.

Example 20. *If we try to synthesize a parameter set starting from* $\alpha \in [0.5, 0.6]$ *and* $\beta \in [0.1, 0.2]$ *and satisfying the formula*

$$F_{[1,1]} I \leq 0.365 \wedge R \leq 2$$

on the evolution of the SIR model of Example 1 when initial conditions are $(S, I, R) \in [0, 1]^3$*, we obtain an empty set (see the first part of Example 19).*

However, if we split the initial parameter set into the four subsets $[0.5, 0.55] \times [0.1, 0.15]$*,* $[0.5, 0.55] \times [0.15, 0.2]$*,* $[0.55, 0.6] \times [0.1, 0.15]$*, and* $[0.55, 0.6] \times [0.15, 0.2]$ *and we consider the union of the synthesized sets for the investigated formula by using as initial parameter sets each of the mentioned subsets independently, we obtain a different result. As a matter of the facts, the set synthesized from the first of them is non-empty (see the last part of Example 19).*

4. Sapo: Set-based Analysis of Parametric Polynomial systems

4.1. Sapo Core

The reachability and parameter synthesis algorithms described in the previous sections have been implemented in Sapo [28, 27]: an open source tool for the formal analysis of discrete-time polynomial dynamical systems freely available at https://github.com/dreossi/sapo.

Figure 11 sketches Sapo architecture. The Model and STL modules implement dynamical systems and temporal logic properties, respectively. The initial set of points, X_0, and the parameter set, P, can be defined using the modules Bundles and LinearSystemSet. The main module Sapo performs the reachability and parameter synthesis computation. The BaseConverter module symbolically computes the Bernstein coefficients of polynomials. Thanks to the method described in [29, 21], Sapo symbolically evaluates these coefficients only once and, then, fetches and numerically instantiates them during the computation.

4.2. `sapo` *and* `Sapo` *Input Language (SIL)*

Originally, `Sapo` was meant to be a library rather than a stand-alone tool. Users were forced to represent models using the C++ programming language and write the code for calling the aimed analysis methods. Moreover, a new compilation was required every time either the model or the reachability/synthesis problem changed. In order to simplify the usage of `Sapo` and broaden its user base, we developed a stand-alone application, named `sapo`, that avoids recompilations by accepting in input problems described by the `Sapo` Input Language (SIL for short). This tool is an efficient and practical alternative to the `Sapo` library for non-expert users; it also embeds for free some nice features –such as multi-threading computations– that, during the writing of this article, were not fully integrated by the `Sapo` C++ API.

`sapo` takes in input SIL file which consist of four main sections:

Header which, among the other options, specifies whether a synthesis or a reachability analysis must be performed;

Symbols definition which declares the variables, the parameters, the constants, the dynamics, and, whenever needed, the STL specification;

Matrices definition which defines the template matrices and the initial set;

Footer which contains some `Sapo`-specific options to fine-tune its behaviour.

Pairs of rows in the parallelotope template matrices are linearly dependent and, in these pairs, each line can be obtained from the other one by multiplying the latter by -1 (e.g., see the template matrix Λ of Example 14).

In order to reduce specification redundancy and decrease user effort, SIL decouples the template matrices of all the bundles used during the computation into two components: the vector of the planes/directions and the template vector whose values specify the directions of each parallelotope. Under this convention, the initial set is declared by specifying an interval of admitted values for each direction.

Section 5.1 presents a case study together with some SIL examples. Please, refer to the SIL page of the project wiki for a complete overview of the syntax.

Thanks to SIL, the `sapo` application avoids the need for a recompilation at each new analysis. The class `Driver` interprets the input provided in SIL format and returns an `InputData` object to represent the model description and, possibly, its specification. As soon as it is executed, the `sapo` tool creates an instance of the `Driver` class and, through it, parses the input. The resulting `InputData` object is used to fill a `Model` object and create a `sapo_opt` object to represent the computation options reported in the input. Then, the application creates an instance of the class `Sapo` and performs the reachability or the synthesis analysis by calling either the `Sapo::reach` or the `Sapo::synthesize`, respectively. Finally, it prints the results: in the case of reachability, users obtain a temporal sequence of sets representing the flowpipe, while, when the synthesis analysis is requested, the tool also shows the set of synthesized parameters. In both cases, the sets are represented as the union of the solution sets of some linear inequalities.

4.3. Web Application

With the main aim of relieving the user from installing `Sapo` and from updating it any time a new version is released, we developed a web interface for the `sapo` tool which, among the other features, can plot flowpipes and sets of synthesized parameters.

The web interface named `webSapo`, allows the user to input variables, system dynamics, and parameters. According to the user's choices for the domains of variables and parameters, it establishes which matrices must be input by users and automatically creates the others. If the addressed problem is parameter synthesis, it also offers a section for the STL specification.

Once the investigated problem has been analysed, the results are locally saved and they can be depicted in both 2D or 3D plots. The step-by-step evolution of the reached set can be also presented as a sequence of images.

Among all its features, `webSapo` also supports off-line computations: any `sapo` output file in JSON format can be loaded by `webSapo` and the analysis results can be plotted as they were just computed by `webSapo` itself.

The `webSapo` source code can be downloaded from `https://github.com/LucaDorigo/webSapo` and a manual for it is available on the project wiki page. A free-to-use online `webSapo` service is offered at `http://encase.uniud.it:3001`.

5. Case Studies

In this section we consider two case studies drawn from real scenarios meant to provide a better insight on the `Sapo` potentials. Here, we do not present any scalability test because, on the one hand, designing adequate tests having increasing complexity and avoiding empty parameter set scenarios is not an easy task, on the other hand, completely factitious models may miss to provide realistic results. Some scalability tests performed on `Sapo` are instead presented in [27].

5.1. COVID-19 Simple Model

The SIR example allows us to keep the description simple, show the main functionalities of `Sapo`, and, at the same time, make some consideration on the current COVID-19 outbreak.

Many models of COVID-19 can be found in the literature (e.g., see [5, 39, 32]). Here we are not interested in depth investigating these models. Some of them are very complex, for instance, involving partial differential equations or stochasticity, and a full paper could be devoted to their implementation and analysis with `Sapo`. Instead, we aim to present our framework using fascinating yet simple examples.

Let us consider a more realistic SIR model with two extra parameters: Vaccination rate μ and loss of immunity rate γ. The system dynamics are:

$$\begin{array}{rcl} S_{t+1} & = & S_t - \beta S_t I_t - \mu S_t + \gamma R_t \\ I_{t+1} & = & I_t + \beta S_t I_t - \alpha I_t \\ R_{t+1} & = & R_t + \mu S_t - \gamma R_t + \alpha I_t \end{array}$$

To capture the evolution of COVID-19 pandemic, let us assume that:

- the transmission rate β ranges in $[0.055, 0.1]$. This is equivalent to an R_0 rate ranging in $[0, 77, 1.4]$;
- the recovery rate α ranges in $[0.05, 0.07]$, i.e., between 14 days and 20 days are necessary for recovery;
- the vaccination rate μ ranges in $[0.00001, 0.001]$, i.e., we assume that we can vaccinate at most 0.1% of the susceptible individuals each day. We consider that only a low fraction of the population can be vaccinated each day, i.e., we are assuming that we want to rely on the existing vaccination centers and not on the hub built during the emergency. Notice that to be more precise μ represents the rate of susceptible individuals that each day are vaccinated and over which the vaccine has efficacy;
- the loss of immunity rate γ ranges in $[0.0027, 0.0055]$, representing that the immunity duration lays between 180 and 360 days.

In order to model the situation in Italy in October 2021 at time 0, we also assume that:

- R ranges in $[0.7, 0.8]$. This accounts of all the individuals that are either vaccinated or recovered from the disease;
- I ranges in $[0.001, 0.1]$. This range is a pessimistic estimation in which we assume a very high number of asymptomatic individuals;
- S ranges in $[0.2, 0.3]$. This value is due to the other choices.

One could be initially interested in forecasting the pandemic situation within 100 days, i.e., computing the set of points reachable within 100 steps. The model together with the just-mentioned problem is described by the following SIL file.

```
problem: reachability;
iterations: 100;

var s;
var i;
var r;

param beta in [0.055, 0.1];
param mu in [0.00001, 0.001];
param gamma in [0.0027, 0.0055];
param alpha in [0.05, 0.07];

dynamic(s) = s - beta*s*i - mu*s + gamma*r;
dynamic(i) = i + beta*s*i - alpha*i;
dynamic(r) = r + mu*s - gamma*r + alpha*i;

direction s in [0.2, 0.3];
direction i in [0.001, 0.1];
direction r in [0.7, 0.8];
```

The last three lines in the SIL file define the directions vector and the intervals vector representing the initial parallelotope. The expression between the reserved words `direction` and `in` could be any expression linear in `s`, `i`, and `r`; for instance, $3 * s/4 + r$ or $s - 2 * i + r/3$.

Since our example uses the canonical directions, i.e., our parallelotopes are boxes, the last three lines can be omitted by specifying the box boundaries during the variable declarations as follows:

```
var s in [0.2, 0.3];
var i in [0.001, 0.1];
var r in [0.7, 0.8];
```

Figure 12, which was produced by `webSapo`, depicts the number of infected individuals over time as computed by `sapo`.

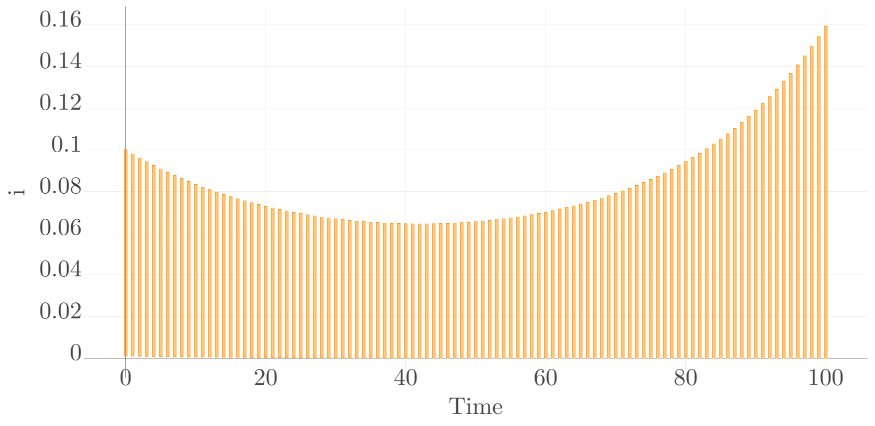


Figure 12: COVID-19 infected individuals over time. This image was produced by `webSapo` and then modified for editorial needs.

It is easy to see that this number surges after around 80 days. Thus, one could be interested in synthesizing the parameters that ensure that the number of infected stays below 0.1 within the next 100 days. Since this last property corresponds to the STL formula $G_{[0,100]}\, i < 0.1$, our goal can be achieved by replacing the word `reachability` with `synthesis` in the first line of the SIL file and, at the same time, by adding the line

```
spec: G[0,100] i < 0.1;
```

at the end of it.

Figure 13 depicts a graphical representation, produced by `webSapo`, of the synthesized values for α and β.

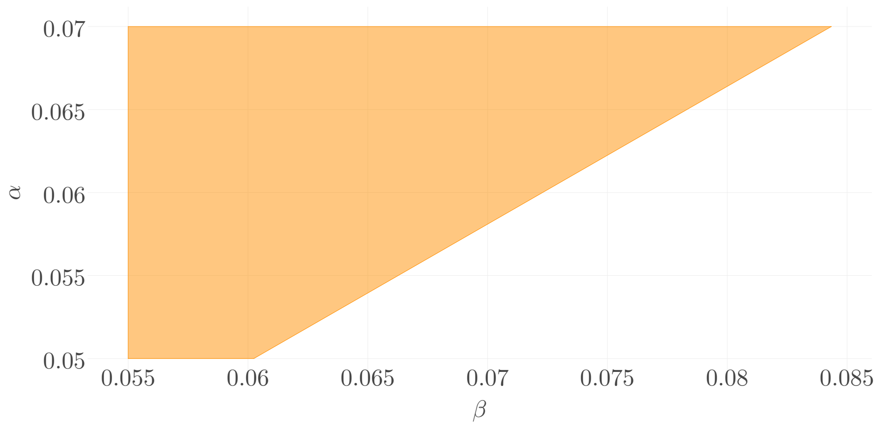


Figure 13: Synthesized values of β and α with respect to $G_{[0,100]}(i < 0.1)$. This image was produced by `webSapo` and then modified for editorial needs.

The meaning of the result is quite intuitive: since we cannot revaccinate a high fraction of the population, we have at least to introduce restrictions on the circulation that reduce the transmission rate β below 0.9. Moreover, β has to be further reduced by introducing more stringent measures if we cannot provide treatments, such as antiviral drugs, that increase the recovery rate α. As a matter of fact, a low recovery rate increases the probability for an infected individual to transmit the disease to a susceptible one. Figure 14 depicts the evolution of the infected with respect to the synthesized parameters.

Finally, one could adjust the vaccination rate μ and loss of immunity rate γ. Let us assume that we can allow the transmission rate β to grow up to 0.15 (i.e., $R_0 = 2.1$), i.e., we would prefer to avoid restrictions. On the other hand, we are prepared for investing in massive revaccination, i.e., μ can grow up to 0.1 of the susceptibles. This approach is not realistic when most of the population is susceptible, but it becomes feasible after the first round of vaccination when the susceptibles are a low fraction of the population. Moreover, this is one of the parameters we are interested in synthesizing, so we let it range in a large interval and check whether it gets reduced after the computation. We are now interested in ensuring that the infected are less than 15% until the susceptible are less than

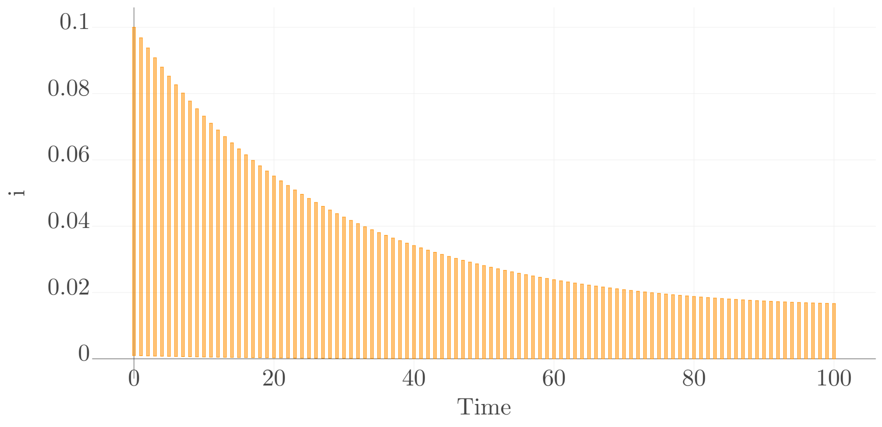


Figure 14: COVID-19 infected individuals over time when using parameters satisfying $G_{[0,100]}(i < 0.1)$. This image was produced by `webSapo` and then modified for editorial needs.

10% within the next 100 days, i.e., we are considering the specification

$$(I < 0.15)U_{[0,100]}(S < 0.1).$$

In this case, the parameter splitting procedure is necessary to obtain a non-empty result. Since `sapo` does not split parameter set during the synthesis by default, we added the line:

```
max_parameter_splits: 3;
```

in the header of the SIL file. This syntax sets the maximum number of admitted parameter splits; in this case, three. The first synthesis round considers the whole initial parameter set. If the synthesis produces a non-empty set, `sapo` returns the result. Otherwise, the parameter set is split in 2^n sub-sets and the process is repeated until either the maximum number of splits has been reached or `sapo` synthesizes a non-empty set of parameters. It is worth noticing that a non-empty set may be returned during the first round. However, in the worst case scenario, the synthesis takes exponential time with respect the maximal number of admitted splits. Because of this, we decided to let users tune this option according to their wishes.

Figure 15 represents the 2D projection of the synthesis over γ and μ and it proves that the parameter set was split during the computation. As expected, the picture shows that if the loss of immunity rate γ is high, i.e., the immunity period is short, we need to invest in revaccination drastically, i.e., μ has to be at least 0.075. Let us notice again that the splits are the key ingredient to getting a non-empty result.

The above described examples have been run on a MacBook Pro M1 2020, with 16GB of RAM, taking less than a second, a part from the last one that

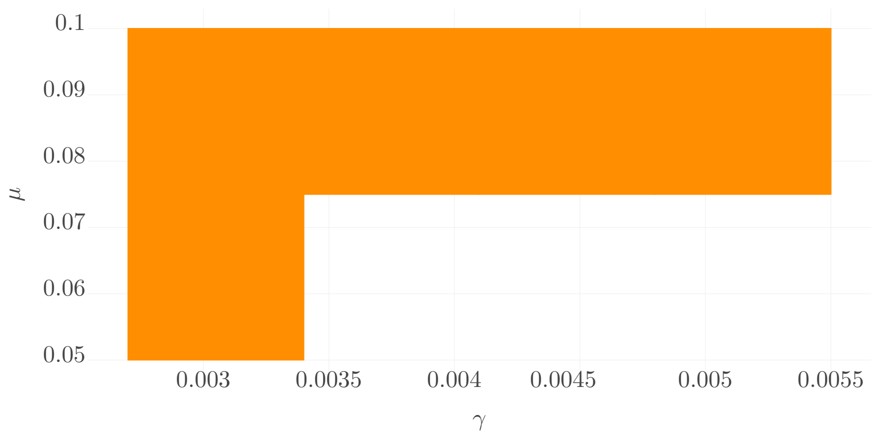


Figure 15: 2D projection of the synthesis over γ and μ. This image was produced by `webSapo` and then adapted to the editorial style of this journal.

took around 34 seconds with multi-threading and 118 with single-threaded computation.

5.2. Enhancing Accuracy of Neural Networks Analysis

In this section we illustrate another potential application of `Sapo`, which is formal analysis of neural networks (NN). This problem has become crucial in the recent development of autonomous applications such as robots, self-driving vehicles, and medical devices. Indeed, neural networks are increasingly used in such applications thanks to their approximation power and ability to “learn” from data, demonstrated by their high performance in image classification, natural language processing and speech recognition (see for example [53, 49]). However, assuring the safety and robustness of their usage, in particular under the impact of adversarial inputs or perturbations, is still challenging. Beyond classification purposes, NN can be used to approximate and implement complex control laws, and verifying a closed-loop system with such neural network controllers requires the ability to approximate the image of an input set by a neural network within a given error bound. Various set propagation and reachability computation from formal verification and abstract interpretation of dynamical systems have been applied to address this problem (see, e.g. [70, 31, 44, 45, 68]). While the application of these approaches does not pose serious theoretical problems, computational complexity is an issue mainly because of the non-linearities of activation functions and the large size of practical neural networks. Piecewise-linear approximations, which are commonly used in these approaches, provide good accuracy for ReLU activation functions, but may lead to overly conservative results for sigmoid activation functions. Polynomial/rational approximations and the Bernstein techniques can thus be combined to enhance the accuracy for parts of the network with high sensitivity. This is what we want to demonstrate in this section. In particular, we focus on the usage of `Sapo` for reachability

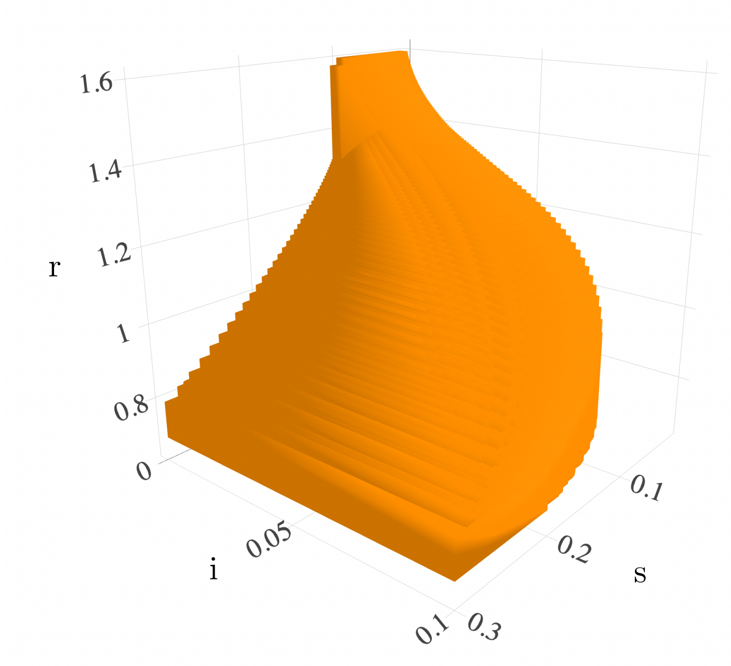


Figure 16: The reachable set after synthesis. This image was produced by `webSapo` and then modified for editorial needs.

computation in the non-parametric case, while some considerations on possible extensions to the parameter synthesis functionalities conclude this section. Therefore, in [60, 59], we propose an algorithm for accurately computing the image of activation functions, by first approximating them using polynomial and rational functions and then applying the Bernstein techniques on the resulting functions.

Let us now demonstrate the idea with feed-forward neural networks. Each layer l of such a network N has the form:

$$\mathbf{h}^l(\mathbf{x}) = \mathbf{s}(\ell^l(\mathbf{x})),$$

where ℓ^l is a linear function, while $\mathbf{s}$ is an activation function. Given an initial set of values for $\mathbf{x}$, the image of the neural networks is the result of the application of all the layers, denoted by $N(\mathbf{x})$. First, our algorithm [60, 59] approximates the functions $\mathbf{h}^l$ by polynomials. Then it over-approximates the outputs of the activation functions $\mathbf{h}^l$ using `Sapo`, parametric reachability computation. In [60], we show how hyperbolic tangent activation functions are approximated using polynomials. In particular, given a desired degree we can sample the hyperbolic tangent in the confident interval and we can tune the coefficients of the polynomial function in order to minimize the maximum absolute error, using, for instance, the `polyfit` function available in MATLAB®. Considering degree 5, we obtained the following polynomial:

$$\tanh(x) \approx 0.9568x - 0.2107x^3 + 0.02354x^5$$

with maximum error below 0.01.

An implementation of this algorithm is then integrated in the tool ERAN (ETH Robustness Analyzer for Neural Networks software) [66], which is a well-developed platform implemented in Python. It has been used for proving NN robustness against adversarial perturbations based on changes in pixel intensity and geometric transformations of images. What makes this tool also powerful is that it supports convolutional, feedforward, and residual networks with different types of activation functions. The tool deals with hyperbolic tangent activation functions via piecewise-linear approximations which, as mentioned earlier, can lead to imprecise results.Combining ERAN and the Bernstein techniques implemented in `Sapo` enables benefitting from the advantages of both, namely the accuracy of image approximations for activation functions of `Sapo`, and the optimized propagation of sets through NN layers of ERAN. Indeed, ERAN propagates the dependency between the neurons through the layers by grouping k activation functions [65]. Note that propagating the full dependency between the neurons is prohibitively expensive, and the group size k is a user-defined parameter for fine-tuning the tradeoff between accuracy and computation cost, by retaining the dependency only between the outputs of the activation functions of the same group. ERAN over-approximates the inputs to these groups by sets of fixed form (also a way to reduce the geometric complexity of the sets generated through the propagation) which can be represented using the parallelotope bundles of Sapo. While ERAN uses a piecewise-linear approximation to deal with hyperbolic tangent activation functions, our Sapo-based algorithm uses the Bernstein expansion.

We illustrate now the advantage of this combination with two examples. One involves two neural network controllers for a robotic arm, and the other involves the neural network benchmarks provided by ERAN. The second example aims at showing the accuracy advantage of combining Sapo with ERAN by a comparison between this combination and ERAN alone. Their implementations are available at https://github.com/PippiaEleonora/SapoForNN.

Neural Network Controllers. The neural networks here are trained using data generated by a nominal discrete-time feedback PID controller for a robotic arm provided by MathWorks®. The aim is to control the trajectory of this system while the value of the input reference signal $r(\cdot)$ randomly changes in $[-0.5, 0.5]$ every 10 seconds. The control u_k at time k is the result of the neural network on 10 inputs: the reference signals r_k, r_{k-1}, r_{k-2}, r_{k-3}; the outputs y_k, y_{k-1}, y_{k-2}, y_{k-3}; the controls u_{k-1}, u_{k-2}. Note that since PID controllers have memory and in order to capture it using a feedforward neural network, previous values of the variables should be fed as inputs to the network.

Two neural networks trained using different data sets are considered: one "bad" using data with diverging behaviour and one "good" with correct behaviour. We train two different neural nets with 2 hidden layers with 30 neurons each. We call the first neural network NN_{bad} since it has a diverging behaviour as shown in Figure 17 for the constant reference with value -0.2. The second is called NN_{ok} since it has a correct behaviour shown in Figure 18.

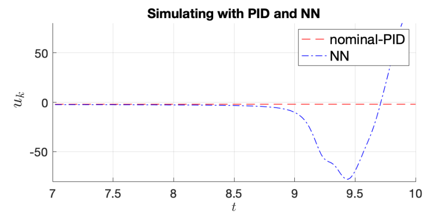


Figure 17: Comparison of the control output for NN_{bad} and a standard PID controller.

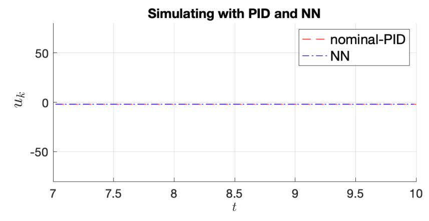


Figure 18: Comparison of the control output for NN_{ok} and a standard PID controller.

We define a noise factor $\epsilon = |r| \cdot p$ where p has three values of $p = \{0.01\%, 0.1\%, 0.5\%\}$. We run our algorithm where the first 4 inputs $r_k, \ldots, r_{k-3}$ take constant value -0.2, the next 4 inputs have the range $[-0.2 - \epsilon, -0.2 + \epsilon]$ and the last two have the range $u = 10y$. We obtain the approximations in Table 1.

Here we use the above-described algorithm combining Sapo and ERAN to compute the set of reachable states under uncertain inputs for the two networks. The computation using our algorithm confirms the expected results, *i.e.*, the "bad" network diverges, while the "good" one does not. For instance, when the perturbation p is at most 1% the "bad" network reaches the interval $[-7.812, 1.049]$, while the "good" one remains in $[-4.146, 0.528]$. We can see

p	$\mathrm{NN}_{\mathrm{bad}}$	$\mathrm{NN}_{\mathrm{ok}}$
0.01%	[-2.344, -1.684]	[-2.133, -1.862]
0.1%	[-5.342, 1.313]	[-3.368, -0.627]
0.5%	[-18.673, 14.600]	[-8.850, 4.858]

Table 1: Interval approximation of $\mathrm{NN}_{\mathrm{bad}}$ and $\mathrm{NN}_{\mathrm{ok}}$ adding the noise factor ϵ.

that the output interval of $\mathrm{NN}_{\mathrm{bad}}$ is 2.4 times the size of the output interval of $\mathrm{NN}_{\mathrm{ok}}$, and this result is coherent with the behaviour of the two nets.

`Sapo`-ERAN versus ERAN. Our goal here is to evaluate if our algorithm using `Sapo` is more precise compared to ERAN which uses piecewise-linear approximations. In the experiments, we use a neural network with hyperbolic tangent activation functions for image recognition provided by ERAN (named `mnist_tanh_3_50.tf`). We describe in the following the results obtained for different sizes of the input box. The group size parameter k is 3, meaning that the activation functions of each layer are considered in groups of size 3.

We run ERAN and `Sapo`-ERAN over different input boxes (e.g., $[-6, 6]$, $[0, 1]$, and $[3, 5]$) obtaining that `Sapo`-ERAN always returned a region included in the one returned by ERAN and, in particular, the rate between the volumes of the regions produced by the former and that of produced by the latter is always about 0.008. In Table 2, we report the outputs for the input box $[-1, 1]$.

Variable	ERAN	`Sapo`-ERAN
n. 1	[-0.4212406, 0.5770473]	[0.0163105, 0.5726539]
n. 2	[-0.3896767, 0.0377295]	[-0.3207002, -0.0517504]
n. 3	[-0.3349207, 0.2658570]	[-0.1952389, 0.2020677]
n. 4	[-0.3848480, 0.0126727]	[-0.3804094, -0.0099185]
n. 5	[-0.2343316, 0.3914449]	[0.0450511, 0.3863766]
n. 6	[0.0583854, 0.4344926]	[0.1958679, 0.3695503]
n. 7	[0.1373136, 0.4805078]	[0.1809298, 0.3411388]
n. 8	[0.2759886, 0.8369578]	[0.4734756, 0.8258935]
n. 9	[0.0675532, 0.8206135]	[0.1222727, 0.5918586]
n. 10	[-0.0174709, 0.5043359]	[-0.0100259, 0.4139967]

Table 2: ERAN and `Sapo`-ERAN outputs for the input box [-1,1].

In the examples on neural networks we exploited the reachability functionalities of `Sapo` over systems without parameters. In the current implementation of Sapo the dynamics can be parametric with a linear dependence on the parameters. Such linear dependence allows the synthesis algorithms to rely on linear optimization subroutines. It would be very interesting if we could keep the linear functions defining the neural network parametric and synthesize the parameters that guarantee the expected behaviours. Unfortunately, the dependence on the parameters does not remain linear when the activation function is applied. We can easily synthesize linear parameters added at the level of the activation function, while it is more natural to have them in the linear part. The extension of Sapo with non linear optimization is currently under investigation, since it is of interest also in other applicative domains.

6. Conclusion and Future Work

We described algorithms for reachability and synthesis of parametric polynomial dynamical systems based on Bernstein coefficients that are implemented in the Sapo C++ library. We also presented a stand-alone application sapo that integrates these algorithms and avoids the need for problem recompilations. This stand-alone tool implements all the features offered by the Sapo library, presents them in a simplified interface, and offers some free extra bonus such as multi-threading. We are aware that libraries are usually more flexible than stand-alone applications and ease integrations in other tools and, because of this reason, sapo is meant to be an alternative way of using the Sapo library rather than a replacement for it.

The webSapo application further extends the audience of Sapo to any possible user on the internet. We presented two examples: the first from epidemiological models using a simple yet realistic COVID setting; the second from neural networks reachability analysis.

Sapo is a well-established tool, but we still consider to add features and expand its applicability domain. In the near future, it will support continuous dynamic laws and hybrid models. In particular, it will be able to over-approximate a flowpipe within a time interval by explicitly adding a time variable ranging in the interval itself. We also aim to support disturbances, control inputs, and we intend to integrate a model checker.

As far as the implementation is concerned, while Sapo already partially supports multi-threading and parallelizes the computation after parameter splits, we plan to extensively use this technology in linear algebra and symbolic operations too. Moreover, we would like to implement status dump, so to save the status of an incomplete analysis at wish and restart it afterward, and MPI multi-processing to distribute the computation on a cluster.

Another intriguing technical development consists of computing Bernstein coefficients and, more in general, each bundle's single evolution step by using GPU. In this case, the aimed efficiency boost will be related to convex components' complexity rather than their number. As far as the synthesis is concerned, the latter approach is symmetrical to the thread-based one: the more Sapo can refine a set of parameters without nullifying it, the less parameter splittings are performed.

Finally, we plan to explore the use of Sapo in other applicative contexts such as security verification [62], performances analysis [4], and blockchain [54].

Acknowledgments Alberto Casagrande is partially supported by the INdAM-GNCS project "*InSANE: Investigating Sparse Algorithms in the post von Neumann Era*".

Thao Dang is partially supported by the ANR-JST CREST project CyPhAI funded by the French National Research Agency ANR and Japan Science and Technology Agency JST, and the project Modylam (Plan Cancer, INSERM).

Carla Piazza is partially supported by the Italian PRIN project "*Non-interference and Reversibility Analysis in Private Blockchains (NiRvAna)*" - 20202FCJMH and by the INdAM/GNCS project "*LESLIE: LogichE non-claSsiche per tooL Intelligenti ed Explainable*".

We thank Alessandro Grisafi and Gianluca Ermacora for their contribution on the first version of the GUI for sapo.

References

[1] Matthias Althoff and Goran Frehse. Combining zonotopes and support functions for efficient reachability analysis of linear systems. In *2016 IEEE 55th Conference on Decision and Control (CDC)*, pages 7439–7446. IEEE, 2016.

[2] Matthias Althoff, Goran Frehse, and Antoine Girard. Set propagation techniques for reachability analysis. *Annual Review of Control, Robotics, and Autonomous Systems*, 4:369–395, 2021.

[3] Matthias Althoff and Dmitry Grebenyuk. Implementation of interval arithmetic in CORA 2016. In *Proc. of the 3rd International Workshop on Applied Verification for Continuous and Hybrid Systems*, pages 91–105, 2016.

[4] Giacomo Alzetta, Andrea Marin, Carla Piazza, and Sabina Rossi. Lumping-based equivalences in markovian automata: Algorithms and applications to product-form analyses. *Information and Computation*, 260:99–125, 2018.

[5] Julien Arino and Stéphanie Portet. A simple model for COVID-19. *Infectious Disease Modelling*, 5:309–315, 2020.

[6] Eugene Asarin, Thao Dang, and Oded Maler. The d/dt tool for verification of hybrid systems. In *International Conference on Computer Aided Verification*, pages 365–370. Springer, 2002.

[7] Eugene Asarin, Alexandre Donzé, Oded Maler, and Dejan Nickovic. Parametric identification of temporal properties. In *Runtime Verification, RV*, pages 147–160. Springer-Verlag, 2012.

[8] Stanley Bak and Parasara Sridhar Duggirala. Simulation-equivalent reachability of large linear systems with inputs. In *International Conference on Computer Aided Verification*, pages 401–420. Springer, 2017.

[9] Ezio Bartocci, Luca Bortolussi, and Laura Nenzi. On the robustness of temporal properties for stochastic models. In *Hybrid Systems and Biology, HSB*, volume 125 of *EPTCS*, pages 3–19, 2013.

[10] Luca Benvenuti, Davide Bresolin, Pieter Collins, Alberto Ferrari, Luca Geretti, and Tiziano Villa. Ariadne: Dominance checking of nonlinear hybrid automata using reachability analysis. In *International Workshop on Reachability Problems*, pages 79–91. Springer, 2012.

[11] Sergiy Bogomolov, Marcelo Forets, Goran Frehse, Kostiantyn Potomkin, and Christian Schilling. Juliareach: a toolbox for set-based reachability. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 39–44, 2019.

[12] Thomas Brihaye, Laurent Doyen, Gilles Geeraerts, Joël Ouaknine, Jean-François Raskin, and James Worrell. On reachability for hybrid automata over bounded time. In *International Colloquium on Automata, Languages, and Programming*, pages 416–427. Springer, 2011.

[13] Michael Brin and Garrett Stuck. *Introduction to Dynamical Systems*. Cambridge University Press, 2002.

[14] G. Cargo and O. Shisha. The Bernstein form of a polynomial. *Journal of Research of the National Bureau of Standards*, 70, 01 1966.

[15] Alberto Casagrande and Tommaso Dreossi. `pyHybridAnalysis`: A Package for Semantics Analysis of Hybrid Systems. In *Digital System Design, DSD*, pages 815–818, 2013.

[16] Alberto Casagrande, Tommaso Dreossi, Jana Fabriková, and Carla Piazza. ϵ-semantics computations on biological systems. *Inf. Comput.*, 236:35–51, 2014.

[17] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification, CAV*, pages 258–263, 2013.

[18] Edmund M. Clarke Jr, Orna Grumberg, Daniel Kroening, Doron Peled, and Helmut Veith. *Model checking*. MIT press, 2018.

[19] Harold Scott Macdonald Coxeter. *Regular polytopes*. Courier Corporation, 1973.

[20] Thao Dang, Tommaso Dreossi, Eric Fanchon, Oded Maler, Carla Piazza, and Alexandre Rocca. Set-based analysis for biological modeling. In *Automated Reasoning for Systems Biology and Medicine*, pages 157–189. Springer, 2019.

[21] Thao Dang, Tommaso Dreossi, and Carla Piazza. Parameter synthesis using parallelotopic enclosure and applications to epidemic models. In *Hybrid Systems and Biology, HSB*, pages 67–82, 2014.

[22] Thao Dang, Tommaso Dreossi, and Carla Piazza. Parameter synthesis through temporal logic specifications. In *Formal Methods, FM*, pages 213–230, 2015.

[23] Jyotirmoy V. Deshmukh, Alexandre Donzé, Shromona Ghosh, Xiaoqing Jin, Garvit Juniwal, and Sanjit A. Seshia. Robust online monitoring of signal temporal logic. *Formal Methods in System Design*, 51(1):5–30, 2017.

[24] Alexandre Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification, CAV*, pages 167–170. Springer, 2010.

[25] Alexandre Donzé, Eric Fanchon, Lucie M. Gattepaille, Oded Maler, and Philippe Tracqui. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PLOS One*, 6(9):e24246, 2011.

[26] Alexandre Donzé, Thomas Ferrere, and Oded Maler. Efficient robust monitoring for STL. In *Computer Aided Verification, CAV*, pages 264–279. Springer, 2013.

[27] Tommaso Dreossi. *Reachability Computation and Parameter Synthesis for Polynomial Dynamical Systems*. PhD thesis, Université Grenoble Alpes; Università degli Studi di Udine, 2016.

[28] Tommaso Dreossi. Sapo: Reachability computation and parameter synthesis of polynomial dynamical systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*, pages 29–34, 2017.

[29] Tommaso Dreossi and Thao Dang. Parameter synthesis for polynomial biological models. In *Hybrid Systems: Computation and Control, HSCC*, pages 233–242, 2014.

[30] Tommaso Dreossi, Thao Dang, and Carla Piazza. Reachability computation for polynomial dynamical systems. *Formal Methods in System Design*, 50(1):1–38, 2017.

[31] S. Dutta, X. Chen, S. Jha, S. Sankaranarayanan, and A. Tiwari. Sherlock - a tool for verification of neural network feedback systems: Demo abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 262–263, 2019. `https://doi.org/10.1145/3302504.3313351`.

[32] Ralf Engbert, Maximilian M Rabe, Reinhold Kliegl, and Sebastian Reich. Sequential data assimilation of the stochastic SEIR epidemic model for regional COVID-19 dynamics. *Bulletin of mathematical biology*, 83(1):1–16, 2021.

[33] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[34] Rida T. Farouki. The Bernstein polynomial basis: A centennial retrospective. *Computer Aided Geometric Design*, 29(6):379–419, 2012.

[35] Goran Frehse, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In *Computer Aided Verification, CAV*, pages 379–395. Springer, 2011.

[36] Oded Galor. *Discrete dynamical systems.* Springer Science & Business Media, 2007.

[37] Luca Geretti, Julien Alexandre Dit Sandretto, Matthias Althoff, Luis Benet, Alexandre Chapoutot, Xin Chen, Pieter Collins, Marcelo Forets, Daniel Freire, Fabian Immler, Niklas Kochdumper, David P. Sanders, and Christian Schilling. ARCH-COMP20 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics. In Goran Frehse and Matthias Althoff, editors, *ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20)*, volume 74 of *EPiC Series in Computing*, pages 49–75. EasyChair, 2020.

[38] Luca Geretti, Julien Alexandre Dit Sandretto, Matthias Althoff, Luis Benet, Alexandre Chapoutot, Pieter Collins, Parasara Sridhar Duggirala, Marcelo Forets, Edward Kim, Uziel Linares, David P. Sanders, Christian Schilling, and Mark Wetzlinger. ARCH-COMP21 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics. In Goran Frehse and Matthias Althoff, editors, *8th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH21)*, volume 80 of *EPiC Series in Computing*, pages 32–54. EasyChair, 2021.

[39] Giulia Giordano, Franco Blanchini, Raffaele Bruno, Patrizio Colaneri, Alessandro Di Filippo, Angela Di Matteo, and Marta Colaneri. Modelling the COVID-19 epidemic and implementation of population-wide interventions in Italy. *Nature medicine*, 26(6):855–860, 2020.

[40] Iman Haghighi, Noushin Mehdipour, Ezio Bartocci, and Calin Belta. Control from signal temporal logic specifications with smooth cumulative quantitative semantics. In *2019 IEEE 58th Conference on Decision and Control (CDC)*, pages 4361–4366. IEEE, 2019.

[41] Thomas A. Henzinger, Pei-Hsin Ho, and Howard Wong-Toi. HyTech: A model checker for hybrid systems. In *Computer Aided Verification, CAV*, pages 460–463. Springer, 1997.

[42] Thomas A. Henzinger, Peter W. Kopke, Anuj Puri, and Pravin Varaiya. What's decidable about hybrid automata? In *Symposium on Theory of computing, STOC*, pages 373–382. ACM, 1995.

[43] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, 20(1):79–93, 2018.

[44] C. Huang, J. Fan, W. Li, X. Chen, and Q. Zhu. Reachnn: Reachability analysis of neural-network controlled systems. *ACM Trans. Embed. Comput. Syst.*, 2019. `https://doi.org/10.1145/3358228`.

[45] R. Ivanov, J. Weimer, R. Alur, G. J. Pappas, and I. Lee. Verisig: verifying safety properties of hybrid systems with neural network controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, HSCC '19, pages 169–178, 2019. `https://arxiv.org/abs/1811.01828`.

[46] Xiaoqing Jin, Alexandre Donzé, Jyotirmoy V. Deshmukh, and Sanjit A Seshia. Mining requirements from closed-loop control models. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 34(11):1704–1717, 2015.

[47] Kevin D. Jones, Victor Konrad, and Dejan Nickovic. Analog property checkers: a DDR2 case study. *Formal Methods in System Design*, 36(2):114–130, 2010.

[48] Rudolf Emil Kalman, Peter L. Falb, and Michael A. Arbib. *Topics in mathematical system theory*, volume 33. McGraw-Hill New York, 1969.

[49] U. Kamath, J. Liu, and J. Whitaker. *Deep Learning for NLP and Speech Recognition*. Springer International Publishing, 2019. `https://www.springer.com/gp/book/9783030145958`.

[50] William O. Kermack and Anderson G. McKendrick. A contribution to the mathematical theory of epidemics. In *Royal Society of London A: Mathematical, Physical and Engineering Sciences*, volume 115, pages 700–721. The Royal Society, 1927.

[51] Edward Kim and Parasara Sridhar Duggirala. Kaa: A Python implementation of reachable set computation using bernstein polynomials. *EPiC Series in Computing*, 74:184–196, 2020.

[52] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dReach: δ-Reachability Analysis for Hybrid Systems. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 200–205. Springer, 2015.

[53] A. Krizhevsky, I. Sutskever, and G. Hinton. Imagenet classification with deep convolutional neural networks. *Neural Information Processing Systems*, 25:1097–1105, 01 2012. `http://doi.org/10.1145/3065386`.

[54] Ivan Malakhov, Carlo Gaetan, Andrea Marin, and Sabina Rossi. Workload prediction in btc blockchain and application to the confirmation time estimation. In *Performance Engineering and Stochastic Modeling*, pages 3–21. Springer, 2021.

[55] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[56] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. *Pillars of computer science*, pages 475–505, 2008.

[57] Olivier Mullier, Alexandre Chapoutot, and Julien Alexandre dit Sandretto. Validated computation of the local truncation error of Runge–Kutta methods with automatic differentiation. *Optimization Methods and Software*, 33(4-6):718–728, 2018.

[58] Pierluigi Nuzzo, Alberto L Sangiovanni-Vincentelli, Davide Bresolin, Luca Geretti, and Tiziano Villa. A platform-based design methodology with contracts and related tools for the design of cyber-physical systems. *Proceedings of the IEEE*, 103(11):2104–2132, 2015.

[59] Eleonora Pippia. *Optimization and Modeling Techniques for Food Service Appliances*. PhD thesis, Università degli Studi di Udine, 2021.

[60] Eleonora Pippia, Thao Dang, and Alberto Policriti. Image approximation for feed forward neural nets. In *Verification of Neural Networks Workshop, VNN 2020*, 2020.

[61] Amir Pnueli. The temporal logic of programs. In *Symposium on Foundations of Computer Science*, SFCS, pages 46–57. IEEE, 1977.

[62] Sabina Rossi. Model checking adaptive multilevel service compositions. In *International Workshop on Formal Aspects of Component Software*, pages 106–124. Springer, 2010.

[63] Sadra Sadraddini and Calin Belta. Robust temporal logic model predictive control. In *2015 53rd Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 772–779, 2015.

[64] Stefan Schupp, Erika Abraham, Ibtissem Ben Makhlouf, and Stefan Kowalewski. HyPro: A C++ library of state set representations for hybrid systems reachability analysis. In *NASA Formal Methods Symposium*, pages 288–294. Springer, 2017.

[65] Gagandeep Singh, Rupanshu Ganvir, Markus Püschel, and Martin Vechev. Beyond the single neuron convex barrier for neural network certification. *Advances in Neural Information Processing Systems*, 32, 2019.

[66] Gagandeep Singh, Timon Gehr, Markus Püschel, and Martin Vechev. Boosting robustness certification of neural networks. In *International Conference on Learning Representations*, 2018.

[67] Szymon Stoma, Alexandre Donzé, François Bertaux, Oded Maler, and Gregory Batt. 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.

[68] H.-D. Tran, X. Yang, D. M. Lopez, P. Musau, L. V. Nguyen, W. Xiang, S. Bak, and T. T. Johnson. NNV: the neural network verification tool for deep neural networks and learning-enabled cyber-physical systems. *CoRR*, abs/2004.05519, 2020.

[69] Paul P. J. van den Bosch and Alexander C. van der Klauw. *Modeling, identification and simulation of dynamical systems.* crc Press, 2020.

[70] W. Xiang, H. Tran, and T. T. Johnson. Output reachable set estimation and verification for multilayer neural networks. *IEEE Transactions on Neural Networks and Learning Systems*, 29(11):5777–5783, 2018. `https://arxiv.org/abs/1708.03322`.