# Reachability Computation and Parameter Synthesis for Polynomial Dynamical Systems

Tommaso Dreossi

Communauté
UNIVERSITÉ Grenoble Alpes

# THÈSE

Pour obtenir le grade de

**DOCTEUR DE LA COMMUNAUTÉ UNIVERSITÉ GRENOBLE ALPES**
**préparée dans le cadre d'une cotutelle entre la Communauté Université Grenoble Alpes et l'Università degli Studi di Udine**

Spécialité : **Informatique**

Arrêté ministériel : 25 mai 2016

Présentée par


## Tommasso DREOSSI


Thèse dirigée par **Thao Dang**, Chargée de recherche et codirigée par **Carla Piazza**

préparée au sein du **Laboratoire VERIMAG,** dans **l'École Doctorale Mathématiques, Sciences et technologies de l'information, Informatique**

# Calcul d'atteignabilité et synthèse de paramètres pour systèmes dynamiques polynomiaux

Thèse soutenue publiquement le **04 avril 2016,**
devant le jury composé de :

**Madame THAO DANG**
Directrice de recherche, CNRS DELEGATION ALPES,
Directrice de thèse

**Madame CARLA PIAZZA**
Professeur associé, UNIVERSITE D'UDINE - ITALIE, Présidente

**Monsieur SRIRAM SANKARANARAYANAN**
Professeur associé, UNIVERSITE DU COLORADO A BOULDER - USA,
Rapporteur

**Monsieur RADU GROSU**
Professeur, UNIVERSITE DE VIENNE - AUTRICHE, Rapporteur

**Monsieur SEBASTIANO BATTIATO**
Professeur associé, UNIVERSITE DE CATANE - ITALIE, Examinateur

**Madame SABINA ROSSI**
Professeur associé, UNIVERSITE CA' FOSCARI DE VENISE ITALIE,
Examinatrice

University of Udine
Department of Mathematics and Computer Science

University Joseph Fourier - Grenoble I
MSTII Doctoral School

# Reachability Computation and Parameter Synthesis for Polynomial Dynamical Systems

Tommaso Dreossi

Jury

| | |
|---|---|
| Carla Piazza | President, director |
| Thao Dang | Examiner, director |
| Sebastiano Battiato | Examiner |
| Sabina Rossi | Examiner |
| Sriram Sankaranarayanan | Reviewer |
| Radu Grosu | Reviewer |



UNIVERSITÉ FRANCO ITALIENNE | UNIVERSITÀ ITALO FRANCESE

# Acknowledgements

I would like to acknowledge some people who contributed to the development of this work.

First, I would like to thank my advisors. I thank Carla Piazza for introducing me to the research world and advising me. Thanks to her teachings and approach to science, I learned how to explore topics more deeply and to be more precise in what I do. Her guidance and fruitful suggestions helped me a lot, especially with the development of this work. I am grateful to Thao Dang; I thank her for conveying to me the passion for research and investigation. Her endless positivity and optimism have given me a serene approach to research. Thanks to her, I learned to enjoy every aspect of this work and understood how lucky I am to be a researcher. I really thank both of you for being constant, present, and patient mentors who have always supported me along this path.

I am grateful to Sriram Sankaranarayanan and Radu Grosu who reviewed this work and provided me with useful comments, improvements, and ideas for future developments.

During my Ph.D., I met people who influenced and inspired me. Among these people is Oded Maler; his creativity, passion for investigation, and irony made me understand, more deeply, the beauty of science and delights derived from it. I thank him for the stimulating discussions we had and his constant flow of reading suggestions. A great learning experience during my studies has been being an intern at Toyota. I want to thank Jim Kapinski, Jyo Deshmukh, and Xiaoqing Jin for giving me the opportunity of working with them and introducing me to real-world industrial problems.

Finally, I thank all the officemates, colleagues, and friends with whom I have shared this journey. And of course, I would like to thank my family, without whose support and love none of this would have been possible.

Tommaso Dreossi

# Abstract

English

Dynamical systems are important mathematical models used to describe the temporal evolution of systems. Often dynamical systems are equipped with parameters that allow the models to better capture the characteristics of the abstracted phenomena. An important question around dynamical systems is to formally determine whether a model (biased by its parameters) behaves well.

In this thesis we deal with two main questions concerning discrete-time polynomial dynamical systems: 1) the *reachability computation* problem, i.e, given a set of initial conditions and a set of parameters, compute the set of states reachable by the system in a bounded time horizon; 2) the *parameter synthesis* problem, i.e., given a set of initial conditions, a set of parameters, and a specification, find the largest set of parameters such that all the behaviors of the system staring from the set of initial conditions satisfy the specification.

The reachability computation problem for nonlinear dynamical systems is well known for being nontrivial. Difficulties arise in handling and representing sets generated by nonlinear transformations. In this thesis we adopt a common technique that consists in over-approximating the complex reachable sets with sets that are easy to manipulate. The challenge is to determine accurate over-approximations. We propose methods to finely over-approximate the images of sets using boxes, parallelotopes, and a new data structure called parallelotope bundles (that are collections of parallelotopes whose intersections symbolically represent polytopes). These approximation techniques are the basic steps of our reachability algorithm.

The synthesis of parameters aims at determining the values of the parameters such that the system behaves as expected. This feature can be used, for instance, to tune a model so that it imitates the modeled phenomenon with a sufficient level of precision. The contributions of this thesis concerning the parameter synthesis problem are twofold. Firstly, we define a new semantics for the Signal Temporal Logic (STL) that allows one to formalize a specification and reason on sets of parameters and flows of behaviors. Secondly, we define an algorithm to compute the synthesis semantics of a formula against a discrete-time dynamical system. The result of the algorithm constitutes a conservative solution of the parameter synthesis problem. The developed methods for both reachability computation and parameter synthesis exploit and improve Bernstein coefficients computation.

The techniques defined in this thesis have been implemented in a tool called *Sapo*. The effectiveness of our methods is validated by the application of our tool to several polynomial dynamical systems.

# Abstract

French

Les systèmes dynamiques sont des modèles mathématiques importants utilisés pour décrire l'évolution temporelle d'un processus physique. Souvent, les systèmes dynamiques sont équipés des paramètres qui permettent aux modèles de mieux saisir les caractéristiques observées des phénomènes. Une question importante est celle de déterminer formellement si un modèle paramétrique peut reproduire des comportement observés ou satisfait une propriété.

Dans cette thèse, nous traitons deux questions concernant les systèmes dynamiques polynômiaux à temps discret: 1) *Calcul d'atteignabilité*, i.e, étant donné un ensemble de conditions initiales et un ensemble de paramètres, calculer l'ensemble d'états atteignables par le système dans un horizon de temps borné; 2) *Synthèse de paramètres*, i.e., étant donné un ensemble de conditions initiales, un ensemble de paramètres, et une spécification, trouver le plus grand ensemble de paramètres tel que tous les comportements du système à partir de l'ensemble de conditions initiales satisfont la spécification.

Le calcul d'atteignabilité pour les systèmes dynamiques non-linéaires est bien connu pour être un problème difficile. Des difficultés surgissent dans le traitement et la représentation des ensembles générés par des transformations non-linéaires. Dans cette thèse, nous adoptons une technique qui consiste à approximer des ensembles d'états atteignables complexes par des ensembles qui sont plus faciles à manipuler. Le défi est de garantir une bonne précision d'approximation. Nous proposons des méthodes pour approximer les images des ensembles par des polynômes en utilisant des boîtes, des parallélotopes, et une nouvelle structure appelées "parallelotope bundle" (qui sont des collections de parallelotopes dont les intersections représentent symboliquement des polytopes). Ces techniques d'approximation sont les étapes de base de notre algorithme de calcul d'atteignabilité.

La synthèse de paramètres vise à déterminer les valeurs des paramètres telles que le système se comporte comme prévu. Cette fonctionnalité peut être utilisée, par exemple, pour ajuster un modèle de sorte qu'il imite le phénomène avec un degré de précision suffisant. Les contributions de cette thèse concernant le problème de la synthèse de paramètres comprennent deux volets. Premièrement, nous définissons une nouvelle sémantique pour la logique STL "Signal Temporal Logic" permettant de formaliser une spécification et de raisonner sur des ensembles de paramètres et les flux de trajectoires. Deuxièmement, nous définissons un algorithme pour calculer la sémantique de synthèse d'une formule par rapport à un système dynamique. Le résultat de l'algorithme constitue une solution du problème de la synthèse de paramètres. Les méthodes mises au point pour le calcul d'atteignabilité et la synthèse de paramètres exploitent et améliorent le calcul des coefficients de Bernstein des polynômes.

Les techniques définies dans cette thèse ont été implantées dans un outil appelé *Sapo*. L'efficacité de nos méthodes est validée par l'application de notre outil à plusieurs cas.

# Contents

# 1

# Introduction

## 1.1 Motivations

### 1.1.1 Models

A *model* is a simplified representation of something that is real, that is not the same as the modeled thing, but hopefully it is enough precise to be useful. A *mathematical model* is a model described by a mathematical formalism.

Since ancient times, humans used mathematical models to represent and simplify the real world aiming to understand the complexity of the surrounding events. For instance, numbers, appeared for the first time around 30.000 BC, were probably the first mathematical model used to abstract a quantity observable in real life. Since then, many areas benefited of mathematical abstraction and several milestones in human knowledge were achieved with the help of models. Some examples [173] are given by Eratosthenes of Cyrene (around 250 BC), who approximated the circumference of the Earth with a geometrical model, Ptolemy (around 150 AD) who used circles to predict the movement of planets, or Giotto di Bondone and Filippo Brunelleschi (around 1300 AD) who exploited geometry to abstract and picture the reality giving birth to the perspective.

Nowadays numerous domains benefit of mathematical models. Besides the classic natural and engineering disciplines, such as physics, mechanical engineering, and biology, models find application also in relatively recent fields such as political sciences, economics, or sociology. This wide range of applications requires the models to have a certain level of versatility and, no wonder, several formalisms have been developed, including but not limited to stochastic models, game theoretic models, discrete automata, and dynamical systems.

The function of a model varies depending on its use. There are several contexts in which models can be useful. Some examples are:

- *Disclose phenomena*: models can be used to better understand phenomena, investigating the relationships between various elements and formalizing the acting dynamics;

- *Make predictions*: once that a model has been constructed, it can be used to predict the behaviors of the modeled system or understand the causes that brought the system to a particular configuration;
- *Make decisions*: the ability of the models to make predictions can be used to simulate different future scenarios and then provide assistance in decision making.

All these features can be extremely useful, but they are effectively exploitable only if we are able to *simulate* the model, i.e., we use the mathematical model to imitate the abstracted phenomenon by carrying out a sequence of calculations. Moreover, the simulations provided by the model should be reliable, meaning that they should capture the characteristics of the modeled phenomenon with a sufficient level of precision.

There are several ways to construct a model from a set of observations. Two common techniques are *interpolation* and *model fitting*. In interpolation the construction of the model is completely driven by data, without the exploitation of any mathematical knowledge about the modeled phenomenon. In model fitting, the modeler posses mathematical hypothesis on the observed phenomenon and tries to calibrate this knowledge, often abstracted through a preexisting model, with the observations. If the simulations of the built model match the collected data and correctly predict future observations, then the model is *validated*. If this is not the case, i.e., the simulations provided by the model are inadequate, then model either needs to be redesigned, or it needs to be recalibrated with the experimental data. The verification of the correctness and the calibration of a model are the central topics of this thesis. In particular, we will focus on one of the most important and exploited class of mathematical models called dynamical systems.

### 1.1.2 Dynamical Systems

*Dynamical systems* are models that describe the relationship between elements in a sequence. This relationship captures the change of the terms of the sequence from one period to another one. If the change takes place over discrete time instants, the dynamical system is said to be *discrete-time*. Otherwise, if the change happens continuously, the dynamical system is called *continuous-time*. The mathematical tools used to formalize discrete-time and continuous-time dynamical systems are *difference* and *differential equations*, respectively. In this work we will mainly focus on discrete-time dynamical systems and hence on difference equations.

In the following we give an intuition of the definition of dynamical systems and we introduce the problems we are interested in. Chapter 2 will be devoted to the formalization of dynamical systems.

**Difference Equations** A discrete-time dynamical system can be represented by a model of difference equations of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k)$$

where $\mathbf{f} : \mathbb{R}^n \to \mathbb{R}^n$. The difference equations of a dynamical system represent an infinite set of functions through which it is possible to generate a sequence that constitutes the numerical solution of the model and, at a higher level, the model prediction.

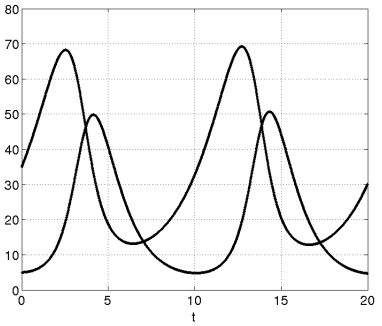



Figure 1.1: Evolution of the Lotka-Volterra discrete-time dynamical system representing the interactions between snowshoe hares $x$ and lynxes $y$ with initial conditions $x_0 = 35$, $y_0 = 5$.

From a given initial condition $\mathbf{x}_0$, it is possible to obtain the next term $\mathbf{x}_1$ applying the function $\mathbf{f}$ to $\mathbf{x}_0$, i.e., $\mathbf{x}_1 = \mathbf{f}(\mathbf{x}_0)$. Iterating this scheme, one can compute the sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ that represents the model prediction (or simulation). This iterative approach is adopted whenever it is not possible to analytically solve the difference equations for a given initial condition, meaning that there is not an explicit formula to describe $\mathbf{x}_k$ in function of $k$ and $\mathbf{x}_0$. This is often the case for *nonlinear dynamical systems*, that are systems whose difference equations right sides involve nonlinear functions. In this thesis we will focus on nonlinear dynamical systems, specifically on the polynomial case.

**Example 1.** *A popular nonlinear dynamical system is the predator-prey model, also known as the Lotka-Volterra model [142, 187]. A particular instance of discrete-time predator-pray model used to describe the dynamics between lynxes and snowshoe hares [94] consists in the following difference equations:*

$$\begin{aligned} x_{k+1} &= x_k + (0.48x_k - 0.02x_k y_k)0.01 \\ y_{k+1} &= y_k + (0.02x_k y_k - 0.92y_k)0.01 \end{aligned} \tag{1.1}$$

*In this model the variables $x_k$ and $y_k$ represent the number of snowshoe hares and lynxes, respectively, at time $k$. The constants terms appearing in the difference equations regulate the interactions between species.*

*Figure 1.1 depicts the sequence generated from the initial conditions $x_0 = 35$ and $y_0 = 5$. The data have been obtained iterating the difference equations* 2000 *times.*

**Parameters** A general mathematical model can often be used to characterize similar situations that differ from specific assumptions. For instance, in Example 1 we used the predator-pray model to describe the dynamics between lynxes and snowshoe hares. However, Volterra originally developed the predator-pray model taking inspiration from observations of fishes in the Adriatic sea [125].

What makes the same set of Lotka-Volterra equations applicable in different contexts, like sea or mountains, are *parameters*, that are constant terms that determine

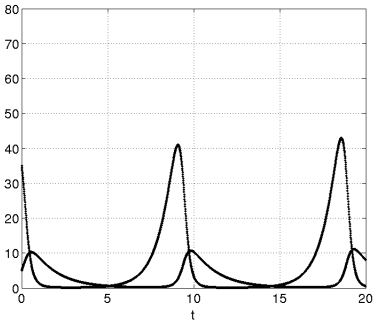



Figure 1.2: Evolution of the parametric discrete-time Lotka-Volterra dynamical system representing the interactions between prays $x$ and predators $y$ with initial conditions $x_0 = 35$, $y_0 = 5$, and parameters $\alpha = 1.3, \beta = 0.5, \gamma = 0.7$, and $\delta = 0.1$.

a specific form of a model without compromising its general nature. This means that with one model (such as the Lotka-Volterra equations) and different parameter values, it is possible to characterize several phenomena with similar dynamics (such as fishes and fishermen, or lynxes and snowshoe hares). Hence, parameters are a useful tool for tuning and adapting models to observations and diverse phenomena.

A parametric discrete-time dynamical system can be represented by a collection of difference equations of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p})$$

where $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. To obtain a sequence $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ from the initial condition $\mathbf{x}_0$ and parameters $\mathbf{p}$, it is sufficient to iterate the difference equations keeping the parameter $\mathbf{p}$ constant.

**Example 2.** *We now present the parametric version of the Lotka-Volterra model, already encountered in Example 1, introducing parameters* $\mathbf{p} = (\alpha, \beta, \gamma, \delta)$. *The model is composed by the following parametric difference equations:*

$$\begin{aligned} x_{k+1} &= x_k + (\alpha x_k - \beta x_k y_k)0.01 \\ y_{k+1} &= y_k + (\delta x_k y_k - \gamma y_k)0.01 \end{aligned} \tag{1.2}$$

*Figure 1.1 depicts the sequence generated from the initial conditions* $x_0 = 35$ *and* $y_0 = 5$, *and parameter instantiated with values* $\alpha = 1.3, \beta = 0.5, \gamma = 0.7$, *and* $\delta = 0.1$. *The sequence has been obtained iterating the parametric difference equations* 2000 *times.*

*Note that the model of Example 1 is a special case of the parametric version of the Lotka-Volterra model with parameters set to* $\alpha = 0.48$, $\beta = 0.02$, $\gamma = 0.92$, *and* $\delta = 0.02$. *Comparing Figure 1.1 with Figure 1.2 we can observe how different parameter values affect the progress of the dynamical system.*

Example 2 gives us the intuition of how parameters give flexibility to dynamical systems. Indeed, they can be used to adapt the general structure of a model to specific cases.

There is a famous quote attributed to John von Neumann by Enrico Fermi [76] that summarizes the importance of parameters and the flexibility that their variations can give to a model: "With four parameters I can fit an elephant, and with five I can make him wiggle his trunk".[1]

### 1.1.3 Reachability

An interesting problem involving dynamical systems is the verification of their soundness with respect to a given specification. Let us imagine that a dynamical system has been constructed to study the behavior of a device involved in a safety-critical scenario, i.e, a situation in which a failure of the system may cause serious consequences. In this case, we may want to verify that the modeled device always behaves well and there is no risk in using it.

Several techniques have been developed to study and analyze dynamical systems. Among these, there is *formal verification*, where it is required to formally establish whether a dynamical system satisfies a given specification.

An approach to formal verification consists of computing all its possible behaviors and testing them against the specification. However, it is of usual interest to verify the model for a uncertain set of initial conditions and parameters (possibly infinite), rather than for single ones. This means that an exhaustive verification procedure may deal with an infinity of simulations. The problem of computing all the states visited by a dynamical system is often call *reachability problem.* Chapter 3 is dedicated to the definition and analysis of this problem.

One might suspect that a finite number of simulations is sufficient to verify a dynamical system, but often there are situations in which small changes in the initial conditions, or in the parameters, cause wild variations in the system behaviors.

**Example 3.** *Let us consider the following discrete-time dynamical system:*

$$x_{k+1} = (1 - x_k)x_k$$

*Figure 1.3 shows two evolutions of the system with different initial conditions. In one case* $x_0 = 0.05$*, in the other* $x_0 = -0.05$*. From the figure we can observe how a small change in the initial condition can sensibly affect the evolution of the dynamical system. For* $x_0 = 0.05$*, the system tends to approach zero, while for* $x_0 = -0.05$ *it diverges towards minus infinity.*

Example 3 gives us the intuition that with a finite number of simulations, we might miss some sequences whose behaviors completely differ from the others. Hence, in general, a sampling-based approach may not be sufficient to verify and compute the reachable set of a dynamical system.

The reachability and formal verification problems attracted a lot of attention in the last two decades. Several techniques to handle infinite sets of behaviors as unique objects have been developed. However, their attentions have been mainly posed on linear dynamical systems, i.e., systems whose equations are linear functions. As a matter of fact, if the verification of linear dynamical systems has found numerous solutions, the

[1] In literature there are attempts of fitting elephants with parameters (see [188, 145]).

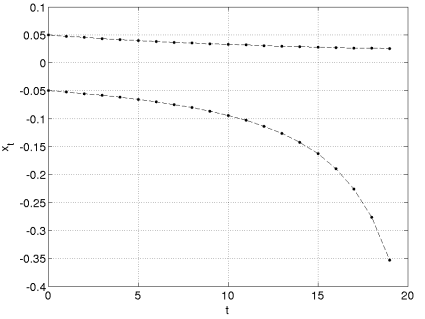



Figure 1.3: Evolutions with different initial conditions ($x_0 = 0.05$ and $x_0 = -0.05$). A small change in the initial condition can affect the evolution of the dynamical system.

analysis of nonlinear dynamical systems remains an open problem that has not yet found general and efficient solutions. Hence, methods to deal with nonlinear systems, that are particularly hard to handle, are needed.

Chapter 4 is dedicated to the development of techniques for the computation of reachable sets of nonlinear (specifically, polynomial) discrete-time dynamical systems. These techniques are useful for the verification of dynamical systems, but they will also play a fundamental role in the identification of valid sets of parameters.

### 1.1.4 Parameter Synthesis

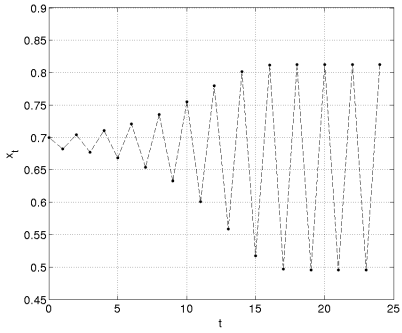
Parameters play an important role in the versatility of models. The closeness of a model to the abstracted phenomenon is sensibly influenced by the values of its parameters. It is therefore important to understand how to find good parameter values in order to obtain reliable models. Finding parameters that relate a model with experimental data is a fundamental step in model construction that takes the name of *parameter estimation.*

One major difficulty in parameter estimation is that models may require many parameters, and most of them are neither measurable nor available in literature. Moreover, since often there are many parameter values that can match the observations, parameter estimation is based not only on the error between the model simulations output and the observations, but also on the model robustness with respect to parameter variation. From a modeling point of view, robust parameters allow the model to fit new data without compromising the fit to the previous ones. This suggests us the importance of working with sets of parameters, rather than with single values.

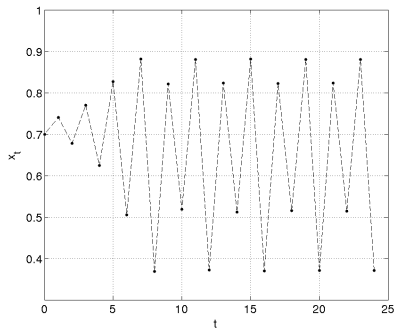
**Example 4.** *Let us consider the following parametric discrete-time dynamical system [95]:*

$$x_{k+1} = p(1 - x_k)x_k$$

*Figure 1.4 shows its evolutions from the initial condition* $x_0 = 0.7$ *and the two different parameter values* $p = 3.250$ *(Figure 1.4a) and* $p = 3.525$ *(Figure 1.4b). From the figure we can observe how a slight change in the parameter sensibly affects the evolution of the dynamical system.*




(a) $p = 3.250$.




(b) $p = 3.525$.

Figure 1.4: Evolutions under different parameter values. A small change in the parameters can affect the evolution of the dynamical system.

Example 4 shows how systems can be sensitive to small parameter changes. This means that whenever working with possibly infinite sets of parameters, a finite number of simulations may not be exhaustive, since we may miss some interesting parameters.

In literature it is possible to find several criteria for parameters estimation. One of the most common methods consists in finding parameters that minimize a cost function describing the difference between the model simulations and the experimental observations. Some examples of cost functions are, for instance, least-squares, Chebyshev criterion, or sums of absolute deviations. However, these approaches can typically handle finite sets of parameters, while in this work we aim to consider the parameter estimation problem from a formal verification perspective working with possibly infinite sets.

Lifting the parameter estimation problem to the formal verification field, we can recast the question of finding valid parameter values as determining all the parameters so that all the executions of threated dynamical system, starting from a set of initial conditions, satisfy a given specification. This problem is often referred as the *parameter synthesis problem* [109] since usually, from a raw set of parameters, it required to synthesize a subset of valid ones.

In Chapter 5 we study the parameter synthesis problem for discrete-time dynamical system and infinite sets of initial conditions and parameters, with a special focus to the polynomial case. We will formalize the problem with the aid of a particular temporal logic and we will propose an algorithm to compute valid sets of parameters.

## 1.2 Related Works

We now give an overview on the main existing techniques for reachability analysis and parameter synthesis.

### 1.2.1 Reachability

In recent years, reachability analysis has attracted a lot of interest especially in the context of formal verification of dynamical and hybrid systems. The large amount of techniques that have arisen can be cataloged by the complexity of system dynamics and the nature of sets. We now summarize the existing techniques for reachability and analysis of dynamical and hybrid system (see Table 1.1).

**Linear Systems** The computation of reachable sets for linear dynamical systems has been one of the most studied problems by the hybrid system verification community. This class of systems has the nice property that the convexity of sets is preserved under transformation. For this reason, convex geometric objects represent a valid tool for flowpipe constructions.

Convex polyhedra have been successfully used in various works. Some basic convex polyhedra are hyper-rectangles [178, 38] and parallelotopes [127, 128], that offer a good trade-off between the system dimension and precision. Zonotopes [96, 97, 3] are polyhedra whose facets are centrally symmetric. Their representation scales very efficiently in dimension but their intersection is difficult to compute. Techniques based on general convex polyhedra in combination with optimization are [107, 41, 42, 43, 186, 85, 169]. Support functions [137, 138], that are symbolic representations of convex polyhedra, have been successfully applied to systems with hundreds of state variables.

Other techniques are ellipsoidal-based methods [132, 27, 131] and exact symbolic methods with semialgebraic sets [4, 135, 159].

Tools based on these ideas are CheckMate [43], HyTech [107], d/dt [6], MPT [134], PHAVer [85], SpaceEx [87], and Ellipsoidal Toolbox (ET) [133].

**Nonlinear Systems** If the reachability problem for linear systems has been widely studied and many efficient solutions have been proposed, the problem of computing reachable sets of nonlinear systems remains rather open and the application of the techniques developed so far is limited to systems with few variables.

A way to deal with the complexity of nonlinear systems, is to consider a subclass of nonlinear dynamics. For instance, multi-affine functions are polynomials in which each variable appears with degree at most one. These functions, even if they are nonlinear (a term can be the product of several linear variables), have nice convexity properties. In this case, reachability analysis on methods have been proposed in [18, 20].

General polynomial systems have been threated in [52, 56, 164, 57, 172, 171], where different polynomial representations (such as Bézier simplex or Bernstein basis) are used to reduce the complexity of the original nonlinear system.

One of the issues related to nonlinear systems is the loss of convexity of a transformed set. One way to deal with this problem is to over-approximate flowpipes with convex sets, as done in [9], or to work directly with nonconvex sets, like for instance orthogonal polyhedra [28, 58]. Other approaches are based on the projections of sets to lower-dimensional spaces [101], interval sets and symbolic computations [165, 77, 88, 36], differential algebraic logic [160, 161], and Taylor models [22, 37].

Some of the tools born from the exposed ideas are d/dt [6], Ariadne [9], Coho [101], KeYmaera [162], pyHybrid Analysis [35], dReach [126], and Flow* [39].

| | | Linear | Nonlinear |
|---|---|---|---|
| Polytopes | Box/Parallelotopes | [178, 38, 127, 128] | [22, 37, 56, 164, 57] |
| | Zonotopes | [96, 97, 3] | |
| | Templates | [107, 41, 42, 43, 186, 85, 169] | |
| | Support Functions | [137, 138] | |
| Ellipsoids | | [132, 27, 131] | |
| Symbolic | | [4, 135, 159] | [161, 165, 77, 88, 35] |

Table 1.1: Summary of reachability analysis techniques.

### 1.2.2 Parameter Synthesis

The classification of the developed methods for the synthesis of valid parameters is non-trivial since the problem can be posed in several forms. Indeed, the problem of finding valid parameters has been studied under different perspectives from several communities, each of which differs in the formulation of the problem.

Table 1.2 summarizes and attempt of classification of the existing parameter synthesis techniques. In the well developed theory of bifurcation for dynamical systems, one aims to find a robust set of parameters that do not undergo a bifurcation under any variation of the parameters within that set. Usually, methods for studying bifurcations are analytical, in the sense that they qualitatively study the dynamics that define the model. Some examples are methods based on the analysis of the eigenvalues of the Jacobian matrix, the Routh-Hurwitz stability criteria [92, 93], or the Floquet multipliers [65].

Different techniques based on numerical calculations are, for instance, [139, 86, 109] in which the parameter synthesis problem is recast into a problem of state estimation by considering parameters as additional state variables of the system with zero derivatives. In [24, 25] a combination of abstraction and numerical reachability analysis is proposed for the estimation of parameters of multiaffine hybrid automata. The numerical parameter estimation for ODE biological systems using cost function optimization has been considered, for instance, by [146, 150, 100].

A different family of approaches, closer to computer science symbolic methods, has been inspired by model checking. Some of these methods are purely symbolical, others involve combinations of symbolic abstraction and model checking. Examples of pure symbolic methods are [10, 30] where the parameter synthesis problem is faced adapting standard model checking techniques. The drawback of this rigorous approach, typically intrinsic to model checking, is the state-space explosion problem. Some attempts to tackle this complexity issue have been done by proposing a parallel model checker that can also be used to synthesize parameters [29]. A similar approach based on the satisfaction of logical formulas is given in [141]. Here the synthesis is preformed using $\delta$-satisfiability, i.e., the $\delta$-approximation of the satisfaction level of the involved formulas. Also differential dynamic logic has been used to verify parametric hybrid systems [160, 161].

A special focus on the parameter synthesis for genetic regulatory networks have been posed by [17, 13, 14, 15, 16]. In this case, the parameter and state spaces of the models are first abstracted and then analyzed by techniques inspired by model checking. This

| Analytic | Numeric | Symbolic | Stochastic |
|---|---|---|---|
| [124] | [24] [25] | [34], [66] [161] | [45] [129] |
| [92] [93] | [70] [69] | [17] [14] [13] [15] [16] | [154] [12] [113] |
| [65] | [146][150][100] | [10] [30] [29] [141] | [23] |

Table 1.2: Summary of parameter synthesis techniques.

technique has been successfully applied in various cases, but its drawback is that discrete abstractions are generally hard to compute.

Parameter synthesis has been considered also for stochastic systems. A randomized method for detecting good parameters has been proposed in [129], while [12] develops a method, inspired by model checking, for monitoring stochastic models and their parameters. Other examples of methods based on random optimization for parameter identification are [154, 113, 23].

The works on parameter synthesis that are the closest to this thesis are [70, 69], where the parameter and initial sets are represented as a union of boxes and systematic simulations and sensitivity analysis [72, 53] are employed to approximate the reachable set. This method can be applied to ordinary differential equations (ODEs) as well as to black-box systems. Our work differs from this in the parameter set representation: we will use polytopes and refine the parameter sets using linear constraints, which allows us to obtain more compact representations and discovering dependencies between parameters. Moreover, in our methods the refinement of the initial parameter set will be dynamically guided by the information on the property violation.

Some examples of tools that can be used to synthesize and analyze parameters are HyTech [109], RoVerGeNe [15], KeYmaera [162], Breach [69], dReach [126], and SpaceRover [25].

## 1.3 Contributions

In the following sections we summarize the main contributions of this thesis concerning reachability analysis and parameter synthesis for dynamical systems.

### 1.3.1 Reachability Analysis

**Set Representation and Image Computation** In this work we develop three techniques to over-approximate reachable sets and compute the transformation of sets under polynomial functions. These methods are essential for the construction of flowpipes that over-approximate the trajectories of a dynamical system starting from a set of initial conditions and being influenced by a set of parameters. We develop techniques of polynomial set image over-approximation for:

- *Boxes* (or hyperrectangles) [73], i.e., the generalization of rectangles on higher dimensions;
- *Parallelotopes* [54], i.e., the generalization of parallelograms on higher dimensions;

- *Parallelotope bundles* [75], i.e., finite sets of parallelotopes whose intersections generate polytopes.

The designed techniques share at their cores a property of Bernstein coefficients of polynomials. This approach was originally developed in [181, 57] for the reachability analysis of polynomial dynamical systems and boxes. In this work we first extend the original technique to parametric dynamical systems with boxes. Then, we define a new way of approximating and transforming sets using parallelotopes always in combination with Bernstein coefficients. This new feature allows one to adopt more flexible sets and obtain finer over-approximating flowpipes. Finally, we further improve the parallelotope-based approximation technique by defining parallelotope bundles, that are sets of parallelotopes whose intersections symbolically represent polytopes. We define this new data structure to represent polytopes and, exploiting the ability of over-approximating the images of single parallelotopes, we define a family of operations for the over-approximation of the images of polytopes. We will exploit parallelotope bundles to define a new reachability algorithm for parametric polynomial dynamical systems that produces flowpipes that are finer than the box-based and parallelotopes-based ones.

**Bernstein Coefficients Computation** Bernstein coefficients are necessary to express polynomials in Bernstein form [21]. They own several interesting properties [83] including the ability of providing upper and lower bounds of polynomial over the unit box domain [174]. The techniques developed in this work heavily exploit Bernstein coefficients. Hence, their computation affects the efficiency and precision of our methods. In this work we contribute to the computation of Bernstein coefficients in several ways:

- We define a new *improved matrix method* [73] to efficiently compute the Bernstein coefficients of a given polynomial;
- We introduce the *symbolic parametric computation* [54] of Bernstein coefficients to avoid redundant computations;
- We propose a heuristic for *subdividing* [73] Bernstein coefficients and obtaining tighter bounds of polynomials.

The matrix method [168] is a technique for computing Bernstein coefficients based on operations on multidimensional matrices that avoids redundant computations. In this work we advance the original matrix method defining a more efficient way of transposing multidimensional matrices. Speeding-up the multidimensional transposition, we boost the computation of Bernstein coefficients and consequently the flowpipe construction for polynomial dynamical systems.

Studying our first reachability algorithm, we noted that we computed similar Bernstein coefficients for different sets. From certain prospectives, we were wasting computations in redundant calculations. For this reason, we developed a new method for symbolically computing the Bernstein coefficients associated with a particular set that allows the reachability algorithm to precompute the coefficients only once and then instantiate them runtime. This strategy drastically reduces the computational times of our reachability and parameter synthesis algorithms.

Our final contribution in the context of Bernstein coefficients concerns the precision of the provided bounds. We developed a subdivision technique to tighten the bounds

and generate finer set image over-approximations based on partial derivatives of the threated polynomials and the spatial positions of Bernstein coefficients.

### 1.3.2 Parameter Synthesis

**Parameter Synthesis and Signal Temporal Logic** Signal Temporal Logic (in short STL) [143, 144] is a logic suitable for specifying properties of dense-time real-valued signals. In this thesis we adopt STL to formalize the properties that a dynamical system must meet. The contributions of this work that involve parameter synthesis and STL are:

- The definition of the *parameter synthesis problem* [55] for dynamical systems through STL specifications;
- The definition of the *synthesis semantics* [55] for STL formulas;
- The realization of a *parameter synthesis algorithm* [55] for discrete-time dynamical systems and STL properties.

In this thesis we formalize the parameter synthesis problem for dynamical systems with respect to STL formulas, i.e., given a dynamical system, a set of initial conditions, a set of parameters, and an STL formula, we want to find the largest subset of parameters such that all the trajectories of the system starting from the set of initial conditions satisfy the formula. Since there might be a infinity of trajectories to analyze, we group them in a unique flowpipe that over-approximates all the states that a system can reach. However, STL formulas are usually evaluated on single signals, that in this case are single trajectories generated by our dynamical system. In order to evaluate a flow of trajectories, we need to adapt the usual semantics of STL. For this reason, we define a new semantics, called synthesis semantics, that allows us to reason on flows of trajectories and whose application to STL formulas produces sets of parameters such that the given formulas are satisfied.

We also define a synthesis algorithm that computes the synthesis semantics of an STL formula for a given discrete-time dynamical system. In particular, our algorithm receives in input a dynamical system, a set of initial conditions, a set of parameters, and an STL formula. It produces in output a subset of parameters such that all the trajectories starting from the set of initial conditions satisfy the STL formula. We propose an instance of our algorithm for polynomial discrete-time dynamical systems whose reachable sets can be over-approximated by boxes or parallelotopes, and parameter sets can be represented by polytopes. We prove the correctness and study the computational complexity of our synthesis algorithm.

### 1.3.3 Tool Implementation

The reachability analysis and parameter synthesis techniques developed in this thesis have been implemented in a C++ tool called *Sapo*. The main features of our tool are the following:

- Efficient computation of Bernstein coefficients of polynomials in power basis;

- Construction of flowpipes that over-approximate reachable sets of polynomial (possibly parametric) discrete-time dynamical systems;

- Synthesis of valid parameter sets with respect to STL specifications for polynomial discrete-time dynamical systems.

The computation of Bernstein coefficients is based on our improved matrix method. The flowpipe construction can be carried out using boxes, parallelotopes, and parallelotope bundles. For the parametric dynamical system, the sets of parameters can be represented as polytopes. The parameter synthesis algorithms supports boxes and parallelotopes to represent sets of states reached by the system and polytopes to represent sets of parameters.

## 1.4 Structure of the Thesis

The thesis is structured in the following chapters:

2. *Dynamical Systems and Parameters*: we begin with the definitions of parametric dynamical system and trajectories. Some illustrative examples are shown to exhibit how parameters influence the evolution of systems. We will give an intuition of what the parameter synthesis problem is and conclude the chapter posing two important questions that are the core of this thesis: how to compute all the states visited by a parametric dynamical system in a finite time horizon, and how to find sets of valid parameter values so that a dynamical system behaves well?

3. *Parametric Reachability*: in this chapter we define and become familiar with the parametric reachability problem, i.e., the problem of determining all the states visited by a parametric dynamical system. We will briefly describe the classical numerical integration schemes and see how it can be used to compute the trajectories generated by dynamical systems. Then, we will focus on the computation of reachable sets and we will see that the already developed methods can be grouped in two large categories: trajectory-based and set-based techniques. In this work, we will focus on the second one. The chapter ends with a brief discussion about the decidability of the reachability problem;

4. *Set Image Computation*: in this chapter, with the aim of developing a set-based reachability algorithm for polynomial dynamical systems, we focus on the problem of computing the polynomial image of a set. As we will discover later, this will be a fundamental task also for the parameter synthesis problem. The chapter starts with the problem formulation and an hypothetical solution based on the polytopic over-approximation of the set. This approach requires the optimization of the transforming function. Hence, we will introduce Bernstein polynomials whose coefficients can be used to bound polynomials. We will first adapt the standard Bernstein coefficients and their properties to the parametric case, and then we will present some new techniques to over-approximate images of sets through boxes, parallelotopes, and parallelotope bundles. We conclude the chapter proposing new methods to efficiently compute Bernstein coefficients;

5. *Parameter Synthesis*: this chapter is dedicated to the parameter synthesis problem. In particular, we will give the definition of STL logic and define the new synthesis semantics that allows us to reason on flows of trajectories and sets of parameters. Thus, we will formalize the parameter synthesis problem for dynamical system through STL specifications. After briefly discussing the decidability of the problem, we will present our synthesis algorithm, describing its structure and analyzing its correctness and computational complexity;

6. *Tool and Experimental Results*: the developed techniques have been implemented in a tool that is described and evaluated in this chapter. Two main parts compose it: in the first, we will expose the structure of the implemented tool presenting its main modules and discussing some implementation choices; in the second, we will apply our tool to some polynomial dynamical systems and evaluate the developed techniques;

7. *Conclusion and Future Works*: the thesis ends with some conclusive thoughts and discussions on the possible future directions that this work can follow.

# 2

# Dynamical Systems and Parameters

In this chapter we introduce the notion of *dynamical systems*, important mathematical objects widely used to model phenomena evolving in time. The modern theory of dynamical system dates back to end of the 19th century in the study of the evolution of the solar system [31].[1] Since then, dynamical systems have found numerous applications in important research fields, such as astronomy, biology, physics, and economics.

A dynamical system is often designed to model an observed phenomenon. In order to tune the model with the observations, one generally recurs to *parameters*, i.e., constant terms of the dynamical system that determine a specific form of the system, but not its general nature. Hence, parameters can be used to capture different evolutions of the modeled phenomenon without distorting the dynamical system. However, an important question is: "*How to find the values of the parameters in such a way that the dynamical system evolves as expected?*".

This chapter begins with the formalization of dynamical systems, introducing two fundamental classes: the *continuous-time* dynamical systems (Section 2.1.1) and the *discrete-time* dynamical systems (Section 2.1.2). In both cases, we will emphasize the role of parameters. Later, we will introduce the questions that are at the core of this work, that are the *reachability* and the *parameter synthesis* problems for dynamical systems (Section 2.2.1).

A dynamical system is said to be *parametric* if its dynamics involve parameters, i.e., constant terms whose values are fixed a priori.

## 2.1 Dynamical Systems

We begin with some basic notions (some taken from [118]) necessary to define dynamical systems.

[1] It is no surprise that the evolution of a state of a dynamical system is often called *orbit*.

The *state* of a system is a description that is sufficient to predict its 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 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 *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 also *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 the evolution starts.

### 2.1.1 Parametric Continuous-Time Dynamical Systems

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

**Definition 1** (Parametric Continuous-Time Dynamical System)**.** *A* parametric continuous-time dynamical system *is a tuple* $\mathcal{C} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ *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 well-behaving vector field.*[2]

The evolutions of a parametric continuous-time dynamical system are governed by differential equations of the form:

$$\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p}) \tag{2.1}$$

where $\mathbf{x} \in \mathcal{X}$ are the state variables of the system and $\mathbf{p} \in \mathcal{P}$ are the parameters. The fundamental difference between state variables and parameters is that during the evolution of the dynamical system the values of the state variables can change, while the values of the parameters are constant.

For every parameter $\mathbf{p} \in \mathcal{P}$, if we assume $\mathbf{f}(\mathbf{x}, \mathbf{p})$ globally Lipschitz continuous in $\mathbf{x}$ [140], we guarantee the existence and uniqueness of a solution of the differential equation $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p})$ for every initial condition in $\mathcal{X}$ and parameter $\mathbf{p} \in \mathcal{P}$. The uniqueness of the solutions ensures that the system is *deterministic*, i.e., from equal initial conditions and time lengths, the system evolves identically.

We now formalize the concept of evolution of a dynamical system giving the definition of trajectory.

**Definition 2** (Trajectory of Parametric Continuous-Time Dynamical System)**.** *A* trajectory *of a parametric continuous-time dynamical system* $\mathcal{C} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ *starting from*

[2] By *well-behaving* we mean that the function $\mathbf{f}$ has finite derivatives (of all orders) at all points.

*a state* $\mathbf{x} \in \mathcal{X}$ *with parameter* $\mathbf{p} \in \mathcal{P}$ *is a function* $\xi^{\mathbf{p}}_{\mathbf{x}} : \mathbb{R}_{\geq 0} \to \mathcal{X}$ *such that* $\xi^{\mathbf{p}}_{\mathbf{x}}$ *is the solution of the differential equation* $\dot{\mathbf{x}} = \mathbf{f}(\mathbf{x}, \mathbf{p})$ *with initial condition* $\mathbf{x}$ *and parameter* $\mathbf{p}$*, that is:*

$$\xi^{\mathbf{p}}_{\mathbf{x}}(0) = \mathbf{x} \text{ and } \forall t \in \mathbb{R}_{\geq 0}, \dot{\xi}^{\mathbf{p}}_{\mathbf{x}}(t) = \mathbf{f}(\xi^{\mathbf{p}}_{\mathbf{x}}(t), \mathbf{p}). \tag{2.2}$$

Let $\mathcal{C} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a parametric continuous-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 continuous trajectories of $\mathcal{C}$ having initial conditions in $X_0$ and parameters in $P$ is:

$$\Xi(X_0, P) = \{\xi^{\mathbf{p}}_{\mathbf{x}_0} \mid \mathbf{x}_0 \in X_0, \mathbf{p} \in P, \text{ and } \xi^{\mathbf{p}}_{\mathbf{x}_0} \text{ is a trajectory of } \mathcal{C}\}. \tag{2.3}$$

**Example 5.** *An example of continuous-time dynamical system is the* SIR *model [123], often used in biology to model the spread of an epidemic disease.*

*The SIR model is a dynamical system defined as* $\mathcal{C} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$*, with* $\mathcal{X} = \mathbb{R}^3$*,* $\mathcal{P} = \mathbb{R}^2$*, and* $\mathbf{f} = (\mathbf{f}_s, \mathbf{f}_i, \mathbf{f}_r)$ *such that:*

$$\begin{aligned} \dot{s} = \mathbf{f}_s(s, i, r) &= -\beta si/N \\ \dot{i} = \mathbf{f}_i(s, i, r) &= \beta si/N - \gamma i \\ \dot{r} = \mathbf{f}_r(s, i, r) &= \gamma i \end{aligned} \tag{2.4}$$

*The system describes a population of* $N \in \mathbb{R}_{\geq 0}$ *individuals partitioned in three compartments:* $s$ *is the group of* susceptible *individuals who have not been exposed to the disease,* $i$ *is the class of* infected *individuals, and* $r$ *are the* removed *individuals who recovered from the disease. The migration of individuals from one compartment to another is regulated by two parameters:* $\beta$ *is the probability for a susceptible individual to become infected once there is a contact with an sick person;* $1/\gamma$ *is the mean infection period, that is the time necessary for an infected individual to migrate to the removed compartment.*

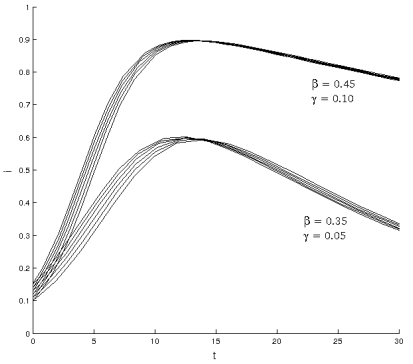
*We now compute some evolutions of the SIR model showing that different parameter values for the same set of initial conditions generate different trajectories. In our simulations, we normalized the population, i.e.,* $N = 1$*. Figure 2.1 shows two sets of trajectories generated by the same initial conditions picked in* $s_0 = 0.80, i_0 \in [0.15, 0.2], r_0 = 0.00$ *up to time* $t = 30$*. In first case, the parameter values are* $\beta = 0.35$ *and* $\gamma = 0.05$*, while in the second* $\beta = 0.40$ *and* $\gamma = 0.01$*. Note how a change in the parameters affects the course of the dynamical system.*

### 2.1.2 Discrete-Time Dynamical Systems

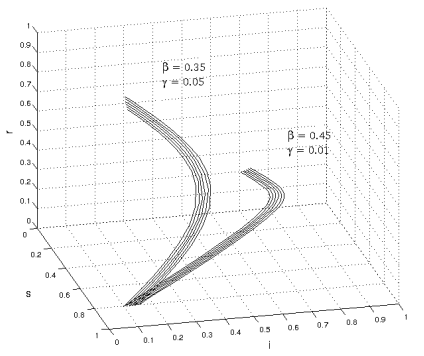
We now move to parametric discrete-time dynamical systems, i.e., dynamical systems that evolve in discrete time and include parameters in their definition.

**Definition 3** (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.*




(a) Evolution of infected individuals $i$ in time.




(b) Evolution of susceptible $s$, infected $i$, and removed $r$ individuals in space.

Figure 2.1: Trajectories of continuous-time SIR system with different parameters but same initial conditions.

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

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p}) \tag{2.5}$$

where $\mathbf{x} \in \mathcal{X}$ is the state of the system, $\mathbf{p} \in \mathcal{P}$ are the parameters, and $k \in \mathbb{N}$ is a discrete time variable. During the evolution of the dynamical system the state variables can change their values, while the parameters remain constant. Let us formalize the concept of evolution through the definition of trajectory.

**Definition 4** (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 parameter $\mathbf{p} \in \mathcal{P}$ is a function $\xi^{\mathbf{p}}_{\mathbf{x}} : \mathbb{N} \to \mathcal{X}$ such that $\xi^{\mathbf{p}}_{\mathbf{x}}$ is the solution of the difference equation $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p})$ with initial condition $\mathbf{x}$, that is:*

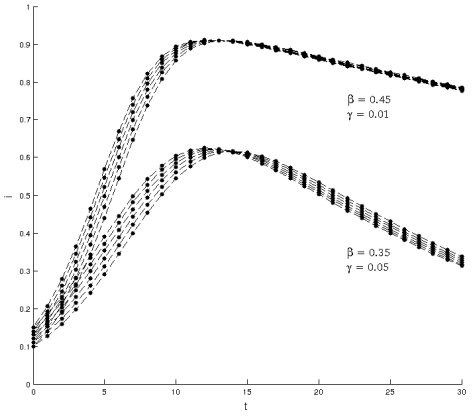
$$\xi^{\mathbf{p}}_{\mathbf{x}}(0) = \mathbf{x} \text{ and } \forall t \in \mathbb{N}_{>0}, \xi^{\mathbf{p}}_{\mathbf{x}}(t+1) = \mathbf{f}(\xi^{\mathbf{p}}_{\mathbf{x}}(t), \mathbf{p}). \tag{2.6}$$

Note that differently from the continuous-time systems, a trajectory of a discrete-time system consists in a sequence of states obtainable by iterating the function $\mathbf{f}$.

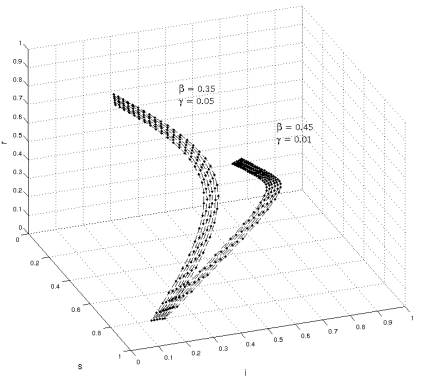
Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a parametric discrete-time dynamical system, $X_0 \subseteq \mathcal{X}$ be a set of initial conditions, and $P \subseteq \mathcal{P}$ be a set of parameters. The set of all possible trajectories of $\mathcal{D}$ with initial conditions in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

$$\Xi(X_0, P) = \{\xi^{\mathbf{p}}_{\mathbf{x}_0} \mid \mathbf{x}_0 \in X_0, \mathbf{p} \in P, \text{ and } \xi^{\mathbf{p}}_{\mathbf{x}_0} \text{ is a trajectory of } \mathcal{D}\}. \tag{2.7}$$

**Example 6.** *This example presents the discrete-time variant of the SIR epidemic model defined in Example 5. Also in this case the population is partitioned in the three compartments of* susceptible $s$*,* infected $i$*, and* removed $r$ *individuals. The interactions*




(a) Evolution of infected individuals $i$ in time.




(b) Evolution of susceptible $s$, infected $i$, and removed $r$ individuals in space.

Figure 2.2: Trajectories of discrete-time SIR systems with different parameters but same initial conditions.

*between different compartments are regulated by the parameters $\beta$ and $\gamma$. The parametric discrete-time SIR model is defined as $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ with $\mathcal{X} = \mathbb{R}^3$, $\mathcal{P} = \mathbb{R}^2$, and $\mathbf{f} = (\mathbf{f}_s, \mathbf{f}_i, \mathbf{f}_r)$ where:*

$$\begin{aligned} s_{k+1} &= \mathbf{f}_s(s_k, i_k, r_k) = s_k - \beta s_k i_k / N \\ i_{k+1} &= \mathbf{f}_i(s_k, i_k, r_k) = i_k + \beta s_k i_k / N - \gamma i_k \\ r_{k+1} &= \mathbf{f}_r(s_k, i_k, r_k) = r_k + \gamma i_k \end{aligned} \tag{2.8}$$

*Figure 2.2 shows two sets of trajectories generated from the normalized initial conditions picked inside the set $s_0 = 0.80, i_0 \in [0.15, 0.2], r_0 = 0.00$ and different parameters up to time $t = 30$. In first case the parameter values are $\beta = 0.35$ and $\gamma = 0.05$, in the second $\beta = 0.40$ and $\gamma = 0.01$. From the figure we observe that different parameters lead to different sets of trajectories.*

## 2.2 Parameter Synthesis Problem

From now on, we will assume that the state space $\mathcal{X}$ and the parameter space $\mathcal{P}$ of a generic dynamical system (continuous-time or discrete-time) $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ are Cartesian products of $\mathbb{R}$ whose exponents depend on the number of variables and parameters appearing in the dynamics $\mathbf{f}$. For instance, for $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, $\mathcal{S} = (\mathbb{R}^n, \mathbb{R}^m, \mathbf{f})$. For brevity, with a slight abuse of terminology, by "dynamical system" or just "system" we mean the dynamics $\mathbf{f}$ of a generic dynamical systems $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$. These shortcuts will allow us to define dynamical systems by giving directly the dynamics, without specifying

each time the state and parameter spaces.

### An Illustrative Example

Consider the parametric discrete-time SIR model presented in Example 6, with the same set of initial conditions $s_0 = 0.80, i_0 \in [0.15, 0.2], r_0 = 0.00$, and the set of parameters $\beta \in [0.35, 0.40]$ and $\gamma \in [0.01, 0.05]$. Differently from the previous examples, now we consider a set of parameters, rather than single values.

Suppose we are asked to find the largest subset of parameter values such that: *"Always between time* 0 *and* 30*, the number of infected individuals i is below* 0.70*"*. We call this requirement the *specification*, or *property*, to be satisfied.

From Figure 2.2 we can see that there are some parameters such that the system satisfies the specification and others that do not. For instance, the values $\beta = 0.35$ and $\gamma = 0.05$ seem to be good candidates, since the plotted trajectories are always below 0.70 between time 0 and 30. On the contrary, the trajectories generated with values $\beta = 0.45$ and $\gamma = 0.01$ assume values larger than 0.70, thus these parameters do not satisfy the specification.

### Formalize Requirements Using Temporal Logic

Observing the property *"Always between time* 0 *and* 30*, the number of infected individuals i is below* 0.70*"*, we notice the combination of two distinct aspects:

- a *temporal* requirement: "*Always between time* 0 *and* 30 *...*" that predicates on the evolution of the system over time;
- a *state-space* requirement: "*... the number of infected individuals i is below* 0.70" that constraints the values that the system variables can assume.

This kind of requirements can be suitably formalized using *temporal logics* [163], formalisms that allow the specification and reasoning on properties involving time.

Temporal logics are typically adopted in the context of formal verification, where a formula specifies the acceptable behaviors of a system and an algorithm is used to check whether all the behaviors of the system satisfy the formula. This procedure is commonly known as *model checking* [46]. Recently, temporal logic has found applications outside formal verification, for instance in *monitoring* [143, 59, 78, 106]. In this case, a formal model is not necessary, since the system can be treated as a black box whose observable behaviors can be monitored by evaluating the satisfaction level of the desired temporal property.

In this work, to specify the behaviors that a dynamical system must satisfy, we will use a recent temporal logic, called *Signal Temporal Logic* (STL [143, 144]). Its peculiarity is that it allows one to formalize properties on dense-time real-valued *signals*, that are functions defined on dense intervals. In our context, a trajectory of a dynamical system and a requirement can act as a signal and an STL formula.

In Chapter 5 we will define in detail STL and its semantics on signals and flowpipes. However, we informally introduce its syntax with the purpose of giving the intuition of what can be expressed by this logic. A Signal Temporal Logic [143] formula is generated

by the following grammar:

$$\varphi := s(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \sim 0 \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi U_I \varphi \tag{2.9}$$

where $s : \mathbb{R}^n \to \mathbb{R}$, $\sim \in \{<, \leq\}$, and $I$ is a closed non-singular interval of $\mathbb{R}_{\geq 0}$. There are two elements that distinguish STL from other logics:

- the *predicates* $s(\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \sim 0$ 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.

From these basic operators, other classical temporal operators can be defined in the usual way, such as *true* $\top$, *false* $\bot$, *eventually/future* $F_I\varphi \equiv \top U_I \varphi$, or *always/globally* $G_I\varphi \equiv \neg F_I \neg\varphi$.

With these elements, we can formalize our requirement expressed in human language *"Always between time* 0 *and* 30*, the number of infected individuals $i$ is below* 0.70*"* using the STL formula $G_{[0,30]}(i < 0.70)$.

### 2.2.1 The Parameter Synthesis Problem

Now that we have defined Signal Temporal Logic formulas, we are ready to formalize the *parameter synthesis problem* for a generic dynamical system.

**Definition 5** (Parameter Synthesis Problem)**.** *Let $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ 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 $\varphi$ be an STL specification. Find the largest subset $P^*_\varphi \subseteq P$ such that:*

$$\forall \mathbf{x}_0 \in X_0, \forall \mathbf{p} \in P^*_\varphi, \xi^{\mathbf{p}}_{\mathbf{x}_0} \text{ satisfies } \varphi \tag{2.10}$$

*where $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ is a trajectory of $\mathcal{S}$.*

The notion of formula satisfaction and the parameter synthesis problem will be formalized in Section 5.1.

## 2.3 Two Important Questions

An intuitive way to generate a valid parameter set is to check the parameters one by one and populate the set $P_\varphi$. In general, this naive algorithm is incomplete and incorrect.

Incomplete, because the parameter set might be infinite and uncountable, hence we will never be able to consider all the possible parameter values.

Incorrect, because when we have established the validity of a parameter value, we have done it considering a finite number of initial conditions and trajectories. If the set of initial conditions is infinite, there might be a point that we have missed such that the correspondent trajectory does not satisfy the specification.

These two observations suggest us that in order to solve the parameter synthesis problem, in the worst case we would need to compute *all* the trajectories starting from *all* the initial conditions with *all* the parameters. Moreover, even once we have all the

trajectories, we would need to produce a set that contains an *infinite* number of valid parameters. Then, the crucial questions are:

1. How to compute all the parametric trajectories generated from infinite sets of initial conditions and parameters?

2. How to compute and represent a valid refinement of the parameter set dealing with infinite sets?

The objective of this work is to give a possible solution to these questions. In Chapter 3 we will clarify the problem of computing all the trajectories generated from an infinite set of initial conditions, while in Chapter 4 we will define some techniques to over-approximate such computation. Later, in Chapter 5, we will define a method to synthesize valid parameter sets.

# 3

# Parametric Reachability

In this chapter we define and discuss the *parametric reachability problem* for parametric dynamical systems, i.e., the problem of computing all the states visited by the trajectories of a dynamical system starting from a set of initial conditions and being biased by a set of parameters. This problem plays a central role in the parameter synthesis problem, since we will be able to determine valid parameter sets only once we are able to compute the evolution of the system under the influence of the treated parameter set.

The chapter begins with the definition of the reachability problem (Section 3.1), then it presents the technique of the numerical integration (Section 3.2) and two different approaches for the computation of reachable sets (Section 3.3). Finally, there will be some considerations on the decidability of the reachability problem (Section 3.4).

## 3.1 Parametric Reachability Problem

The problem of computing the states visited by the trajectories of a dynamical system starting from an initial set and having a particular parameter set is called the *parametric reachability problem*.

Let $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a dynamical system. 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 $\xi_{\mathbf{x}}^{\mathbf{p}}$ of $\mathcal{S}$ starting in $\mathbf{x}$ such that $\mathbf{x}' = \xi_{\mathbf{x}}^{\mathbf{p}}(t)$. 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:

$$Reach^{\mathbf{p}}(\mathbf{x}_0) = \{\mathbf{x}' \mid \mathbf{x}' = \xi_{\mathbf{x}_0}^{\mathbf{p}}(t), t \in \mathbb{T}\} \tag{3.1}$$

where $\xi_{\mathbf{x}_0}^{\mathbf{p}}$ is a trajectory of $\mathcal{S}$ and $\mathbb{T}$ is the set of non-negative reals $\mathbb{R}_{\geq 0}$ or the set of naturals $\mathbb{N}$, depending on whether $\mathcal{S}$ is a continuous-time or discrete-time dynamical system, respectively.

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:

$$Reach^{P}(X_0) = \bigcup_{\mathbf{x}_0 \in X} \bigcup_{\mathbf{p} \in P} Reach^{\mathbf{p}}(\mathbf{x}_0). \tag{3.2}$$

The definition of reachable set reflects the behavior of the dynamical system for an infinite amount of time. However, we might be interested in studying a model for a bounded time horizon. Thus, the set of states reachable in a bounded amount of time $T \in \mathbb{T}$ is defined as:

$$Reach_{T}^{\mathbf{p}}(\mathbf{x}_0) = \{\mathbf{x}' \mid \mathbf{x}' = \xi_{\mathbf{x}_0}^{\mathbf{p}}(t), 0 \leq t \leq T\} \tag{3.3}$$

$$Reach_{T}^{P}(X_0) = \bigcup_{\mathbf{x}_0 \in X_0} \bigcup_{\mathbf{p} \in P} Reach_{T}^{\mathbf{p}}(\mathbf{x}_0). \tag{3.4}$$

### Reachable Set Computation

The computation of the reachable set of a dynamical system, in both its bounded or unbounded time versions, might be problematic. The first issue concerns the numerical computation of the states visited by a trajectory. With the exception of the cases where the trajectories can be characterized by explicit solutions (e.g., $\{\mathbf{x}_0 e^{At} \mid t \in \mathbb{R}_{\geq 0}\}$ for linear systems $\dot{\mathbf{x}} = A\mathbf{x}$), the usual way to compute the reachable states is to use *numerical integration*. The second issue interests the possible infinite number of trajectories we have to deal with, since we might consider infinite sets of initial conditions and parameters. There are several techniques that try to cope with these problems. They can be grouped in two classes:

- *Trajectory-Based Reachability*: a finite number of initial conditions and parameters, called *nominal values*, are chosen. Usually, the nominal values are the result of a discretization or some statistical assumptions on the state-parameter space. In general, the number of nominal values necessary to reach a certain level of coverage of the state-parameter space grows drastically in the dimension of the system.

- *Set-Based Reachability*: considering all the given initial conditions and parameters at once, an exhaustive set of trajectories, called *flowpipe*, is generated. This approach is strongly related to formal verification and set-based computation. In this case it is necessary to deal with image computation and manipulation of sets, problems that are mathematically and computationally nontrivial.

In this work we focus exclusively on set-based reachability and on the computation of valid flowpipes for dynamical systems. Before going into the details of our techniques, we become familiar with the notions of numerical integration, trajectory-based and set-based reachability techniques, providing an overview on the existing methods for the reachability problem.

## 3.2 Numerical Integration

*Numerical integration* is a common technique used to compute the set of states reachable by a dynamical system. The computation of the reachable states (or an approximation of them) is done by simulating[1] incrementally the system using discrete-time steps.

The aim of the numerical simulation is to obtain a *simulation trace*, that is sequence of states $\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots$, where $t_0, t_1, \ldots$ is a monotonic sequence of time steps and $\mathbf{x}_{t_i} \in \mathcal{X}$, for every $t_i \in \mathbb{N}$. In order to produce a simulation trace, an integrator needs [110]:

1. an initial value for $\mathbf{x}_{t_0}$;
2. a procedure to compute $\mathbf{x}_{t_{i+1}}$ from $\mathbf{x}_{t_i}$.

A precise integrator will produce a simulation trace $\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots$ that is close to the original trajectory generated by the dynamical systems.

### Numerical Integration of Continuous-Time Systems

Numerical integration of continuous-time dynamical systems is a well-known and widely studied mathematical problem for which large collections of techniques have been proposed (see, e.g., [61, 130, 110] for surveys on numerical integration or [105, 112] for integration of ordinary differential equations). The common element among these techniques is the *discretization scheme* that we briefly recall.

Let $\mathcal{C} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a parametric continuous-time dynamical system. We recall that a valid trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ of $\mathcal{C}$ staring in $\mathbf{x}_0 \in \mathcal{X}$ with $\mathbf{p} \in \mathcal{P}$ is such that:

$$\frac{d\xi^{\mathbf{p}}_{\mathbf{x}_0}(t)}{dt} = \mathbf{f}(\xi^{\mathbf{p}}_{\mathbf{x}_0}(t), \mathbf{p}) \tag{3.5}$$

condition that can be equivalently rewritten as:

$$\xi^{\mathbf{p}}_{\mathbf{x}_0}(t) = \mathbf{x}_0 + \int_0^t \mathbf{f}(\xi^{\mathbf{p}}_{\mathbf{x}_0}(\tau), \mathbf{p}) d\tau. \tag{3.6}$$

This suggests us that an approximation $\mathbf{x}_{t_{i+1}}$ of the state traversed by the trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ at time $t_{i+1}$ can be obtained by applying the iterative scheme:

$$\mathbf{x}_{t_{i+1}} = \mathbf{x}_{t_i} + g^{\mathbf{p}}(\mathbf{x}_{t_i}) \tag{3.7}$$

where $g^{\mathbf{p}}$ is approximation of the integral appearing in Equation 3.6.

Some well known examples of numerical integrations are the *Euler's method* where:

$$g^{\mathbf{p}}(\mathbf{x}) = \Delta \mathbf{f}(\mathbf{x}, \mathbf{p}) \tag{3.8}$$

or the *Runge-Kutta's method*:

$$g^{\mathbf{p}}(\mathbf{x}) = \Delta \mathbf{f}(\mathbf{x} + \frac{\Delta}{2}\mathbf{f}(\mathbf{x}, \mathbf{p}), \mathbf{p}) \tag{3.9}$$

[1] In this work the term *simulation* is used in the numerical/analytical sense [84] rather than in the algebraic one [147, 148, 155].

where $\Delta \in \mathbb{R}$ is a fixed discretization step. The problem of finding good discretization functions has been widely studied in mathematics and it goes outside the scope of this work. For more discretization techniques the reader may refer, e.g., to [61, 130, 110, 105, 112].

**Numerical Integration of Discrete-Time Systems**

The computation of the trajectories of discrete-time systems requires less mechanisms than the continuous-time case. In fact, here, to obtain a simulation trace $\mathbf{x}_{t_0}, \mathbf{x}_{t_1}, \ldots$, it is sufficient to apply iteratively the system dynamics to an initial condition and a parameter.

Given a parametric discrete-time dynamical systems $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$, the state $\mathbf{x}_{t_{i+1}}$ traversed by the trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ at time $t_{i+1}$ can be obtained by the iterative scheme dictated by the system dynamics:

$$\mathbf{x}_{t_{i+1}} = \mathbf{f}(\mathbf{x}_{t_i}, \mathbf{p}). \tag{3.10}$$

At the $t_i$-th iteration, the integrator generates a state $\mathbf{x}_{t_i}$ that corresponds exactly to the state traversed by the trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ at time $t_i$. Note that here the simulation trace matches exactly the states of the discrete-time trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$, hence no approximation is introduced by the numerical integration.

In the next sections we will see how numerical integration can be used in different ways to compute the reachable set of a dynamical system. We begin with trajectory-based analysis, where the reachable set is computed in a depth-first fashion. Then, we introduce set-based analysis, where the reachable set is determined in a breath-first way. We will discuss the benefits and the complications of both the methods, and later we will focus exclusively on set-based analysis, proposing new techniques to approximate the reachable set of dynamical systems.

## 3.3 Reachability Methods

The existing techniques to compute or estimate the reachable sets of dynamical systems can be grouped in two main categories: *trajectory-based* and *set-based* methods.

### 3.3.1 Trajectory-Based Reachability

Trajectory-based reachability methods are characterized by the *depth-first* computation of the reachable set (see Figure 3.1). The key steps of these methods are:

1. Selection of an initial condition and parameter;
2. Simulation of the dynamical system up to a maximum time instant (e.g., with some integration technique like those exposed in Section 3.2);
3. Repetition of Step 1 and 2 until a condition is met.

The halting condition of Step 3 can involve different criteria such as the achieving of a fix-point in the reachable set computation, the coverage level of the state-parameter space, or the satisfaction or violation of a specification.

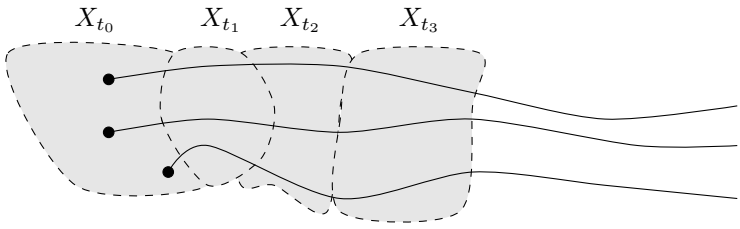



Figure 3.1: Trajectory-based (black lines) and set-based (gray sets) reachability.

The challenge of this kind of approach resides in the ability of finding conditions under which a finite number of simulation is sufficient to deduce all the possible trajectories and establish the validity of the system. Such conditions are usually inferred from continuity [67, 72] or statistical assumptions of the system under study [190, 26].

The selection of proper initial conditions is usually driven by an *abstraction* process (often coinciding with partitioning or discretization of the state-parameter space) that produces a finite number of quotients of the state-parameter space where each quotient is equivalent with respect to some property. The finite partition is then used to select some representative initial conditions and parameters, and construct a flowpipe that contains the reachable set [120, 67, 72, 117].

Unluckily there are many cases in which it is not possible to construct a finite abstraction and it is necessary to recur to approximation techniques. One way is to halt the abstraction process once it has been reached the level of tolerance expressed by the user. The precision of the result depends on the refinement of the partitions: the finer the partitions, the more accurate the over-approximation of the reachable set. A different approach consists in the relaxation of the definition of equivalence between states [98, 99].

Trajectory-based simulations have found application also outside the scope of reachability analysis. Some examples are the stability analysis of dynamical systems [119, 182], invariant set computation [104, 183] inspired by program analysis [49, 48], i.e., sets from which the system will not escape, and the validation ot falsification of systems against specifications [81, 184, 5, 74].

Some tools that exploit trajectory-based methods are Breach [69], S-Taliro [5], and RRT-Rex [74].

### 3.3.2 Set-Based Reachability

Differently from trajectory-based analysis, set-based reachability methods are characterized by the incremental computation of the reachable set. The goal here is to compute a sequence of sets $X_{t_0}, X_{t_1}, X_{t_2}, \ldots$ that constitutes a flowpipe containing all the trajectories starting from $X_{t_0}$. These methods can be seen as a *breath-first* computation of the reachable set (see Figure 3.1).

The usual way to compute the reachable set of dynamical systems is to use *numerical set-integration*. Similarly to single trace integrators (see Section 3.2), the key steps a of a set-based integrator are:

1. Fix a set of initial conditions $X_{t_0}$;

2. Compute $X_{t_{i+1}}$ using the set $X_{t_i}$ and the parameter set $P$;

3. Repeat Step 2 until a condition is met.

Usually the halting condition is defined on a maximum number of steps of the algorithm or the achieving of a fix-point in the reachable set computation, checkable by the inclusion $X_{t_{i+1}} \subseteq X_{t_i}$.

The key element of a set-based method is the computation of the image of a set (see Step 2). This task is at the core of the exhaustive computation of the transit behavior of a dynamical system. Nevertheless, the computation of the image of a set can be problematic and its difficulty depends on the system dynamics and the considered sets. For a survey on the existing reachability techniques see Sections 1.2.1.

## 3.4 (Un)Decidability

To conclude this overview on the reachability problem, we pose ourselves a fundamental question: *Is the reachable set computable?* The answer in general is *no*, since it has been proved [108] that the reachability problem can be reduced to the halting problem of a 2-counter machine (that is undecidable [149]). However, the question as posed, is in its most general form. Indeed, there are many variants depending on the considered system dynamics (linear, nonlinear), sets (polyhedra, ellipsoid, etc.), and time (continuous, discrete, bounded, unbounded).

In this work we focus on *polynomial parametric discrete-time dynamical systems*, i.e., parametric discrete-time dynamical systems whose dynamics are polynomials. These systems, in combination with semialgebraic sets (i.e., subsets of $\mathbb{R}^n$ defined by a finite sequence of polynomial equations and inequalities, or any finite union of such sets) show interesting properties. In the next section we will encode the bounded time reachability problem for polynomial discrete-time dynamical systems into the satisfiability of a first-order formula whose decidability is known.

### Semialgebraic Reachability

Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a polynomial discrete-time dynamical system and $X_0 \subseteq \mathcal{X}, P \subseteq \mathcal{P}$ be two semialgebraic sets (note that polytopes fall in this class). We can write two formulas $X_0[\mathbf{x}]$ and $P[\mathbf{p}]$ that characterize $X_0$ and $P$, respectively, whose conjunction $XP_0[\mathbf{x}, \mathbf{p}] \equiv X_0[\mathbf{x}] \wedge P[\mathbf{p}]$ represents the state-parameter space at time zero.

The formula that describes a single step of the system is:

$$XP_{i+1}[\mathbf{x}_{i+1}, \mathbf{p}] \equiv \exists \mathbf{x}_i (XP_i[\mathbf{x}_i, \mathbf{p}] \wedge \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{p})) \tag{3.11}$$

that in words means: take $\mathbf{x}_i$ and $\mathbf{p}$ from the state-parameter set $XP_i[\mathbf{x}_i, \mathbf{p}]$ , compute $\mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{p})$, and return $\mathbf{x}_{i+1}$ with its correspondent parameter $\mathbf{p}$. Hence, the set reachable in $T \in \mathbb{N}_{>0}$ steps can be captured with the formula:

$$XP_T[\mathbf{x}_T, \mathbf{p}] \equiv \bigwedge_{i=0}^{T} \exists \mathbf{x}_i (XP_i[\mathbf{x}_i, \mathbf{p}] \wedge \mathbf{x}_{i+1} = \mathbf{f}(\mathbf{x}_i, \mathbf{p})). \tag{3.12}$$

The set reachable at time $T$ is the set of points that substituted to $\mathbf{x}_T$ and $\mathbf{p}$ make the formula $XP_T[\mathbf{x}_T, \mathbf{p}]$ true. Finding this set is decidable [179] and it can be done by any quantifier elimination technique [47] (some quantifier elimination tools are QEPCAD B [33], Redlog [64], or Z3 [63]).

**Example 7.** *Consider a simple nonlinear dynamical system whose dynamics are* $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p}) = \mathbf{p}\mathbf{x}_k^2$*, with set of initial conditions* $X_0 = [0.1, 0.2]$ *and parameter set* $P = [-1.0, 1.0]$*. The state-parameter set at time zero can be described by the formula:*

$$XP_0[\mathbf{x}_0, \mathbf{p}] \equiv 0.1 \leq \mathbf{x}_0 \leq 0.2 \wedge -1.0 \leq \mathbf{p} \leq 1.0. \tag{3.13}$$

*The set reachable a time* 1 *can be described by the formula:*

$$\begin{aligned} XP_1[\mathbf{x}_1, \mathbf{p}] &\equiv \exists \mathbf{x}_0 (XP_0[\mathbf{x}_0, \mathbf{p}] \wedge \mathbf{x}_1 = \mathbf{f}(\mathbf{x}_0, \mathbf{p})) \\ &\equiv \exists \mathbf{x}_0 (0.1 \leq \mathbf{x}_0 \leq 0.2 \wedge -1.0 \leq \mathbf{p} \leq 1.0 \wedge \mathbf{x}_1 = \mathbf{p}\mathbf{x}_0^2) \end{aligned} \tag{3.14}$$

*whose equivalent version produced by the quantifier eliminator is:*

$$\begin{aligned} XP_1[\mathbf{x}_1, \mathbf{p}] \equiv\ & (-1.0 \leq \mathbf{p} \leq 1.0 \wedge \mathbf{p} + 25\mathbf{x}_1 = -1) \vee \\ & (\mathbf{p} \neq 0 \wedge \mathbf{p}\mathbf{x}_1 \leq 0 \wedge ((\mathbf{p} \leq 0 \wedge \mathbf{p} \geq -1 \wedge \mathbf{p}^2 + 25\mathbf{p}\mathbf{x}_1 \geq 0 \wedge \\ & \mathbf{p}^2 + 100\mathbf{p}\mathbf{x}_1 \leq 0) \vee (\mathbf{p} \geq 0 \wedge \mathbf{p} \leq 1 \wedge \mathbf{p}^2 + 25\mathbf{p}\mathbf{x}_1 \geq 0 \wedge \\ & \mathbf{p}^2 + 100\mathbf{p}\mathbf{x}_1 \leq 0))). \end{aligned} \tag{3.15}$$

The drawback of this approach is its computational complexity that is doubly exponential in the degree of the formula [102] (or in the number of quantifier alternations, depending on the algorithm). Notice from Example 7 how the degree of the formula duplicates in a single step. It is not hard to imagine that this approach tends to blowup in time. However, this semialgebraic approach establishes that the exact bounded-time reachability computation for polynomial discrete-time dynamical systems with semialgebraic sets is decidable.

The next chapter is dedicated to the research of a trade-off between precision and efficiency in the set-image computation problem.

# 4

# Set Image Computation

The operation at the core of the reachable set computation is the *set image* calculation, that is the transformation of a set. This task can be nontrivial, especially when the transforming function is nonlinear. In this chapter we focus on the problem of computing the image of sets with respect to polynomial functions. In Section 4.1.2 we will see how the image set computation can be connected to the problem of optimizing polynomials, issue that will be faced in Section 4.2 through Bernstein coefficients, i.e., coefficients used to express a polynomial with the Bernstein basis. Bernstein coefficients will be involved in techniques for over-approximating the transformation of sets using boxes (Section 4.3), parallelotopes (Section 4.4), and parallelotope bundles (i.e, sets of parallelotopes) (Section 4.5). At the end of the chapter (Section 4.6), we will focus on the precision and efficiency of the proposed methods, providing some techniques to obtain tighter over-approximation and faster algorithms.

## 4.1 Single Step Reachable Set Approximation

The construction of a flowpipe containing the reachable set of a dynamical system is based on the computation of the image of a set. As previously mentioned, this task can be difficult, especially when considering nonlinear systems. For instance, such kind of systems do not preserve convexity and the generated sets can be hard to handle. It is common practice to deal with this issue by over-approximating reachable sets through convex sets. The idea is to construct a series of convex sets $X_0, X_1, X_2, \dots$ whose union leads to a flowpipe that includes all the behaviors of the threated system. The challenge is to efficiently compute an over-approximation that is as tight as possible.

Since we approximate non-convex sets with convex ones, it is very unlikely that we will obtain the exact reachable set. However, we can try our best, choosing appropriate approximation sets and squeezing them around the reachable set.

The goal of the next sections is to focus on the computation of a single step reachability trying to determine tight sets that over-approximate the states reached by a system. Formally, given the discrete-time dynamical $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ and the sets $X \subseteq \mathcal{X}$

and $P \subseteq \mathcal{P}$, we want to find a manipulable tight set $X' \subseteq \mathcal{X}$ such that:

$$X' \supseteq \mathbf{f}(X, P), \tag{4.1}$$

where $\mathbf{f}(X, P)$ denotes the set $\{\mathbf{x}' \mid \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{p}), \mathbf{x} \in X, \mathbf{p} \in P\}$. Note that we do not have an explicit representation of the set $\mathbf{f}(\mathbf{x}, \mathbf{p})$ which means that we have to deduce the approximation set $X'$ from the previously reached set $X$, the parameter set $P$ (that we assume to be both in an explicit form), and the system dynamics $\mathbf{f}$.

### 4.1.1 Polytopes

In this section we introduce *polytopes*, a class of convex sets commonly used in the reachability problem.

*Polytopes* are the $n$-dimensional generalization of convex polygons and polyhedra. They are a natural tool to represent sets of states since their facets can be arbitrarily chosen so that precise over-approximations can be obtained.

The term "polytope" is often confusing. Originally, it was introduced by the twelve years old Alicia Boole Stott (daughter of the logician George Boole) to denote a four-dimensional convex solid [50]. During the years the term has evolved and has been used to mean related but different mathematical objects.

In geometry a polytope is a geometric object with flat sides definable in any dimension while the terms polygon and polyhedron denote a two-dimensional and three-dimensional polytope, respectively. However, the terms convex polytope and convex polyhedra are often used interchangeably. In addition, sometimes polytopes are required to be bounded while others they are allowed to be unbounded.

In this work we use the definition of polytope adopted by the hybrid system community, that is a convex, bounded, with flat sides set of $n$-dimensional points.

**Definition 6** (Polytope)**.** *A* polytope $Q \subset \mathbb{R}^n$ *is a bounded subset of* $\mathbb{R}^n$ *such that there is a finite set* $H = \{h_1, h_2, \ldots, h_m\}$ *of half-spaces whose intersection is* $Q$*, i.e.:*

$$Q = \bigcap_{i=1}^{m} h_i, \tag{4.2}$$

*where an half-space* $h = \{\mathbf{x} \mid \mathbf{dx} \leq c\}$ *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 *direction matrix* (or *template*) and a vector $\mathbf{c} \in \mathbb{R}^k$, called *offset vector*. The $i$-th row $D_i$ of $D$ together with the $i$-th component $\mathbf{c}_i$ of $\mathbf{c}$ defines the half-space $h_i \in H$, being its normal vector and offset, respectively. With a slight abuse of notation, we denote with $Q = \langle D, \mathbf{c} \rangle$ the polytope generated by the direction matrix $D$ and the offset vector $\mathbf{c}$. Notice that polytopes are bounded subsets of $\mathbb{R}^n$, hence not all the pairs $\langle D, \mathbf{c} \rangle$ define a polytope.

A polytope $Q$ can be represented as the intersection of different sets of half-spaces. For instance, adding to $Q$ new half-spaces that do not affect the intersection, we get a new representation of $Q$. Moreover, even without adding new half-spaces we can get a new representation by multiplying the $i$-th row of $D$ and the $i$-th component of $\mathbf{c}$ by

a constant $k_i > 0$. However, if needed, one can refer to the canonical representation in which all the half-spaces are necessary and the direction vectors are versors (vectors of norm 1). Since we use $\langle D, \mathbf{c}\rangle$ to denote a polytope, we may write $\langle D, \mathbf{c}\rangle = \langle D', \mathbf{c}'\rangle$ meaning that the generated polytopes are the same.

In literature, a polytope is often said a *$\mathcal{H}$-polytope* if it is represented as a set of half-spaces. However, a polytope $Q$ can also be seen as the convex hull of a finite set of points $V = \{\mathbf{v}_1, \mathbf{v}_2, \ldots, \mathbf{v}_p, \}$, with $\mathbf{v}_i \in \mathbb{R}^n$ for $i \in \{1, \ldots, p\}$, that is:

$$Q = \{\alpha_1\mathbf{v}_1 + \alpha_2\mathbf{v}_2 + \cdots + \alpha_p\mathbf{v}_p \mid \alpha_i \geq 0, \sum_{i=1}^{p} \alpha_i = 1\}. \tag{4.3}$$

A polytope represented by its vertices is called a *$\mathcal{V}$-polytope*.

These two representations are equivalent, in the sense that they can denote the same object, but their strengths and weaknesses depend on the operations that one wants to apply to the polytope. For instance, $\mathcal{H}$-polytopes are suitable for intersections, while $\mathcal{V}$-polytopes are convenient for convex hull of unions of polytopes. However, many algorithms that manipulate polytopes have a worst-case exponential complexity since a $n$-dimensional polytope with $m$ constraints has a worst-case exponential number of $\Theta(m^{\lfloor n/2 \rfloor})$ vertices. For our purposes, we will deal exclusively with $\mathcal{H}$-polytopes that, for brevity, we will call polytopes.

At last, it is important to mention *template polyhedra*[1] [170, 169], that are a subclass of polytopes where the directions of the constrains are fixed (or assumed as input) and the offsets can vary. In doing so, it is possible to symbolically denote an infinity of polytopes and balance efficiency with precision.

### 4.1.2 Polytope-Based Set Image

Polytopes can be used to over-approximate compact sets. In particular, given a compact set $S \subset \mathbb{R}^n$ we want to find a polytope $Q \subset \mathbb{R}^n$ that over-approximates $S$. In order to obtain a precise approximation, it is reasonable to push the constraints that define $Q$ as close as possible to $S$. A polytope $Q$ whose constraints are tangent to a set $S$ is said to be an *enclosing polytope* of $S$ (see Figure 4.1).

**Definition 7** (Set Enclosure)**.** *Let $S \subset \mathbb{R}^n$ be a compact set and $Q = \langle D, \mathbf{c}\rangle \subset \mathbb{R}^n$ be a polytope. The enclosure of $S$ with respect to $Q$ is defined as the polytope $\odot(S, Q) = \langle D, \mathbf{c}'\rangle$, where:*

$$\mathbf{c}'_i = \max_{\mathbf{x}\in S} D_i\mathbf{x}, \text{ for } i \in \{1, \ldots, m\}. \tag{4.4}$$

The enclosure of $S$ with respect to $Q$ can be seen as a tight over-approximation of $S$ obtained using the template of $Q$.

In our specific flow-pipe construction problem, we can think of over-approximating a reached set $X_{i+1} = \mathbf{f}(X_i, P)$ with an enclosing polytope $\overline{X}_{i+1}$. By definition, in order for $\overline{X}_{i+1} = \langle D, \mathbf{c}\rangle$ to be an enclosing polytope of $X_{i+1}$ it must hold that:

$$\mathbf{c}_j = \max_{\mathbf{x}\in X_{i+1}} D_j\mathbf{x}, \text{ for all } j \in \{1, 2, \ldots, m\}. \tag{4.5}$$

[1] In this section polyhedra are polytopes.

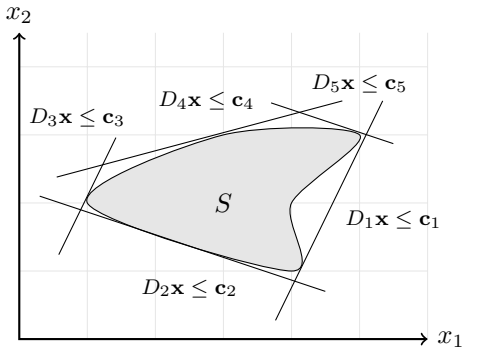



Figure 4.1: A compact set $S$ (in gray) and an enclosing polytope $Q = \langle D, \mathbf{c} \rangle$ (in white).

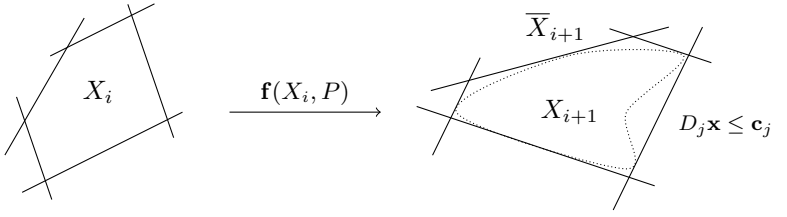



Figure 4.2: Over-approximation of a single step reachability with polytopes.

Since we might not have an explicit representation of $X_{i+1}$, we can think of symbolically replacing $X_{i+1}$ with $\mathbf{f}(X_i, P)$. Thus, the enclosing condition of Equation 4.5 becomes:

$$\mathbf{c}_j = \max_{\mathbf{x} \in \mathbf{f}(X_i, P)} D_j \mathbf{x}, \text{ for all } j \in \{1, 2, \ldots, m\}, \tag{4.6}$$

or equivalently:

$$\mathbf{c}_j = \max_{\mathbf{x} \in X_i, \mathbf{p} \in P} D_j \mathbf{f}(\mathbf{x}, \mathbf{p}), \text{ for all } j \in \{1, 2, \ldots, m\}. \tag{4.7}$$

This condition tells us that we can compute the offsets $\mathbf{c}_1, \ldots, \mathbf{c}_m$ of the polytope $\overline{X}_{i+1}$ by maximizing the product of its directions with the system dynamics over the sets $X_i$ and $P$ (see Figure 4.2).

If the template of $\overline{X}_{i+1}$ is known, as in the case of template polyhedra, finding the offsets $\mathbf{c}_1, \ldots, \mathbf{c}_m$ corresponds to solve $m$ optimization problems. If $X_i$ and $P$ are polytopes and $\mathbf{f}(\mathbf{x}, \mathbf{p})$ is a linear system, then the optimizations can be efficiently solved using linear programming (LP) [44, 111]. Nevertheless, we are considering nonlinear dynamical systems, hence the problem requires nonlinear/nonconvex optimization techniques.

Solving a nonlinear optimization problem is computationally expensive. Some methods to solve or approximate nonlinear optimization problems are gradient descent [8],

interior point method [60, 121], or sum of squares techniques [158, 157]. Unluckily, their scalability in the number of variables, degree of the function to optimize, and number of constraints, is not sufficiently efficient for our verification purposes.

A way to face the non linearity issue is to relax the enclosing constraints of the facets of $\overline{X}_{i+1}$, namely, instead or requiring the facets to be tangent to reached set, we try to push them as close as possible without falling in the complexity trap. Mathematically speaking, instead of finding the optimum $\mathbf{c}_j$ of Equation 4.7, we try to efficiently look for upper bounds $\overline{\mathbf{c}}_j \geq \mathbf{c}_j$ such that:

$$\overline{\mathbf{c}}_j \geq \mathbf{c}_j = \max_{\mathbf{x} \in X_i, \mathbf{p} \in P} D_j \mathbf{f}(\mathbf{x}, \mathbf{p}), \text{ for all } j \in \{1, 2, \ldots, m\}. \tag{4.8}$$

The result will be a polytope $\overline{X}_{i+1} = \langle D, \overline{\mathbf{c}} \rangle$ which might not be an enclosing for $X_{i+1}$, but will over-approximate the reached set. The quality of the approximation depends on how close the upper bounds $\overline{\mathbf{c}}_j$ are to the optimums $\mathbf{c}_j$.

## 4.2 Bounding Polynomials

In the previous section we have seen how the optimization of a function can be used to over-approximate the image of a set and thus compute the flowpipe of a dynamical system. In this section we focus on the problem of optimizing a polynomial over some subclasses of polytopes. The presented technique is based on the Bernstein expansion of polynomials and their properties, a mathematical tool that we will often use in this work.

A Bernstein polynomial is a polynomial expressed in the Bernstein form, that is a linear combination of the Bernstein basis polynomials. As we will see in the following, the coefficients associated with these basis own interesting properties that can be exploited to bound the image of a polynomial over a unit box domain.

This section is organized in two parts. First, we introduce Bernstein basis and their properties in their original form considering polynomials of the form $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$. Second, we extend the definition of Bernstein expansion to parametric polynomials of the kind $\pi(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. We will demonstrate that important properties of standard Bernstein basis also hold in the parametric case. These proofs will be the key element for bounding parametric polynomials, computing parametric flowpipes, and later, synthesizing valid parameter sets.

### 4.2.1 Bernstein Basis and Coefficients

Before defining the Bernstein basis, we introduce some notations useful to work with polynomials.

A *multi-index* is a vector $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n)$ 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 $j \in \{1, 2, \ldots, n\}$, $\mathbf{i}_j \leq \mathbf{d}_j$. Also, we write $\mathbf{i}/\mathbf{d}$ for the multi-index $(\mathbf{i}_1/\mathbf{d}_1, \mathbf{i}_2/\mathbf{d}_2, \ldots, \mathbf{i}_n/\mathbf{d}_n)$ and $\binom{\mathbf{d}}{\mathbf{i}}$ for the product of the binomial coefficients $\binom{\mathbf{d}_1}{\mathbf{i}_1}\binom{\mathbf{d}_2}{\mathbf{i}_2}\ldots\binom{\mathbf{d}_n}{\mathbf{i}_n}$.

A *polynomial* $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ can be represented using the power basis as follows:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I^\pi} \mathbf{a_i} \mathbf{x}^{\mathbf{i}} \tag{4.9}$$

where $\mathbf{i} = (\mathbf{i}_1, \mathbf{i}_2, \ldots, \mathbf{i}_n)$ is a multi-index of size $n \in \mathbb{N}$ and $\mathbf{x}^{\mathbf{i}}$ denotes the monomial $\mathbf{x}_1^{\mathbf{i}_1} \mathbf{x}_2^{\mathbf{i}_2} \ldots \mathbf{x}_n^{\mathbf{i}_n}$. The set $I^\pi$ is called the *multi-index set* of $\pi$. The *degree* $\mathbf{d}$ of $\pi$ is the smallest multi-index that dominates all the multi-indices of $I^\pi$, i.e., for all $\mathbf{i} \in I^\pi, \mathbf{i} \leq \mathbf{d}$. The coefficients $\mathbf{a}_i \in \mathbb{R}$ assume real values.

**Example 8.** *Consider the polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$. *The multi-index set of* $\pi$ *is* $I^\pi = \{(2,0), (0,1), (1,1), (0,0)\}$ *and the associated coefficients are* $\mathbf{a}_{(2,0)} = 1/3, \mathbf{a}_{(0,1)} = -1/2, \mathbf{a}_{(1,1)} = 1/4$, *and* $\mathbf{a}_{(0,0)} = 1/2$.

*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 $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n) \in \mathbb{R}^n$, the $\mathbf{i}$-th Bernstein polynomial of degree $\mathbf{d}$ is defined as:

$$\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) = \beta_{\mathbf{d}_1,\mathbf{i}_1}(\mathbf{x}_1)\beta_{\mathbf{d}_2,\mathbf{i}_2}(\mathbf{x}_2)\ldots\beta_{\mathbf{d}_n,\mathbf{i}_n}(\mathbf{x}_n) \tag{4.10}$$

where, for a real number $x \in \mathbb{R}$,

$$\beta_{\mathbf{d}_j,\mathbf{i}_j}(x) = \binom{\mathbf{d}_j}{\mathbf{i}_j} x^{\mathbf{i}_j}(1-x)^{\mathbf{d}_j - \mathbf{i}_j}. \tag{4.11}$$

A polynomial $\pi(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}$ can be represented using Bernstein basis as:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I^\pi} \mathbf{b_i} \mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) \tag{4.12}$$

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

$$\mathbf{b_i} = \sum_{\mathbf{j} \leq \mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}} \mathbf{a_j}. \tag{4.13}$$

Notice how Bernstein coefficients can be calculated from the coefficients of the monomials of the threated polynomial in power basis. The $(n+1)$-dimensional points $(\mathbf{i}/\mathbf{d}, \mathbf{b_i}) \in \mathbb{R}^{n+1}$ are called *Bernstein control points*.

**Example 9.** *Consider the polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ *from Example 8. For illustrative purposes, we examine only the multi-indices* $(1,1)$ *whose correspondent Bernstein coefficient is:*

$$\mathbf{b}_{(1,1)} = \frac{\binom{(1,1)}{(1,1)}}{\binom{(2,1)}{(1,1)}} 1/4 - \frac{\binom{(1,1)}{(0,1)}}{\binom{(2,1)}{(0,1)}} 1/2 + \frac{\binom{(1,1)}{(1,0)}}{\binom{(2,1)}{(1,0)}} 0 + \frac{\binom{(1,1)}{(0,0)}}{\binom{(2,1)}{(0,0)}} 1/2 = 0.125. \tag{4.14}$$

*Applying the same scheme to the other multi-indices, we obtain the Bernstein coefficients* $\mathbf{b}_{(0,0)} = 0.5$, $\mathbf{b}_{(0,1)} = 0.0$, $\mathbf{b}_{(1,0)} = 0.5$, $\mathbf{b}_{(1,1)} = 0.125$, $\mathbf{b}_{(2,0)} = 0.834$, *and* $\mathbf{b}_{(2,1)} = 0.584$.

### Properties of Bernstein Coefficients

Bernstein coefficients present several interesting properties. Here we expose two properties that will be exploited in our techniques of set image computation and parameter synthesis. For further properties see, for instance, [174, 83].

The properties of Bernstein coefficients we are interested are following.

**Lemma 1** (Range Enclosing)**.**

$$\min_{\mathbf{i}\in I^{\pi}} \mathbf{b}_{\mathbf{i}} \leq \pi(\mathbf{x}) \leq \max_{\mathbf{i}\in I^{\pi}} \mathbf{b}_{\mathbf{i}}, \tag{4.15}$$

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

**Lemma 2** (Sharpness)**.**

$$\text{for all } \mathbf{i} \in \mathcal{V}_{\mathbf{d}}, \mathbf{b}_{\mathbf{i}} = \pi(\mathbf{i}/\mathbf{d}), \tag{4.16}$$

*where* $\mathbf{b}_{\mathbf{i}}$*, for* $\mathbf{i} \in I^{\pi}$*, are the Bernstein coefficients of* $\pi$ *and* $\mathcal{V}_{\mathbf{d}}$ *is the set of vertices of the box* $[0, \mathbf{d}_1] \times [0, \mathbf{d}_2] \times \ldots [0, \mathbf{d}_n]$*.*

These two properties provide us some useful informations about the image of the polynomial $\pi$ over the unit box:

1. The range enclosing property states that the minimum and maximum Bernstein coefficients are a lower bound and an upper bound of the image of $\pi$ over the unit box domain, respectively;

2. The sharpness property implies that the Bernstein coefficients at the vertices of the box domain, match exactly the values of the polynomial at some points.

**Example 10.** *Consider the polynomial* $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ *and its Bernstein coefficients (from Example 9). Figure 4.3 shows the image of* $\pi$ *over the unit box (gray area) and its control points (black dots).*

*The coefficients* $\mathbf{b}_{(1,1)} = 0.125$ *and* $\mathbf{b}_{(2,0)} = 0.834$ *are a lower bound and upper bound of* $\pi([0,1]^2)$ *(range enclosing property) and the control points that fall on the vertices of the unit box match exactly the values of* $\pi([0,1]^2)$ *(sharpness property).*

Concerning the precision of the bounds provided by Bernstein coefficients, the following lemma [57] bounds the distance between a polynomial and its Bernstein control points, or in other words, the error between the maximum and minimum of a polynomial and the bounds provided by its Bernstein coefficients.

**Lemma 3.** *Let* $C_{\pi} : \mathbb{R}^n \to \mathbb{R}$ *be the piecewise linear function defined by the Bernstein control points of the polynomial* $\pi : \mathbb{R}^n \to \mathbb{R}$*, with respect to the box* $[0,1]^n$*. For all* $\mathbf{x} \in [0,1]^n$

$$| \pi(\mathbf{x}) - C_{\pi}(\mathbf{x}) | \leq \max_{\mathbf{x}\in[0,1]^n; i,j\in\{1,\ldots,n\}} | \partial_i\partial_j\pi(\mathbf{x}) | \tag{4.17}$$

*where* $|\cdot|$ *is the infinity norm on* $\mathbb{R}^n$*.*

Several convergent subdivision procedures for reducing the gap between bounds and optimums have been proposed [90, 152, 151]. In Section 4.6.3 we will also define a method to improve the precision of the bounds provided by Bernstein coefficients.

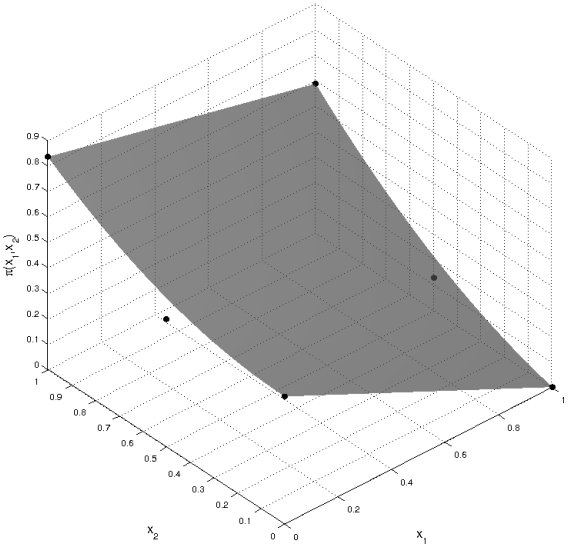



Figure 4.3: The polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2) = 1/3\mathbf{x}_1^2 - 1/2\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ over the unit box (in gray) and its control points (in black).

### 4.2.2 Parametric Bernstein Basis and Coefficients

Bernstein coefficients and their properties are useful tools for bounding polynomials. However, since we work with parametric dynamical systems, we want to bound parametric polynomials of the kind $\pi(\mathbf{x},\mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$. Thus, we have to extend the definition of Bernstein basis to the parametric case. We will see that all the presented properties related to the Bernstein coefficients also hold when parameters are involved.

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

$$\pi(\mathbf{x},\mathbf{p}) = \sum_{\mathbf{i}\in I^\pi} \mathbf{a}_\mathbf{i}(\mathbf{p})\mathbf{x}^\mathbf{i}, \tag{4.18}$$

where the coefficients $\mathbf{a}_\mathbf{i}(\mathbf{p}) : \mathbb{R}^m \to \mathbb{R}$, for $\mathbf{i} \in I^\pi$, are functions defined over the parameters $\mathbf{p} \in \mathbb{R}^m$.

**Example 11.** *Consider the parametric polynomial* $\pi(\mathbf{x}_1,\mathbf{x}_2,\mathbf{p}_1) = 1/3\mathbf{p}_1\mathbf{x}_1^2 - 1/2\mathbf{p}_1\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$. *The parametric coefficients are* $\mathbf{a}_{(2,0)}(\mathbf{p}_1) = 1/3\mathbf{p}_1$, $\mathbf{a}_{(0,1)}(\mathbf{p}_1) = -1/2\mathbf{p}_1$, $\mathbf{a}_{(1,1)}(\mathbf{p}_1) = 1/4$, *and* $\mathbf{a}_{(0,0)}(\mathbf{p}_1) = 1/2$.

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

$$\pi(\mathbf{x},\mathbf{p}) = \sum_{\mathbf{i}\in I^\pi} \mathbf{b}_\mathbf{i}(\mathbf{p})\mathcal{B}_{(\mathbf{d},\mathbf{i})}(\mathbf{x}) \tag{4.19}$$

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

$$\mathbf{b}_\mathbf{i}(\mathbf{p}) = \sum_{\mathbf{j}\le\mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{d}}{\mathbf{j}}}\mathbf{a}_\mathbf{j}(\mathbf{p}). \tag{4.20}$$

Differently from the nonparametric case, the parametric Bernstein coefficients are functions of the form $\mathbf{b}_\mathbf{i}(\mathbf{p}) : \mathbb{R}^m \to \mathbb{R}$.

**Example 12.** *Consider the polynomial* $\pi(\mathbf{x}_1,\mathbf{x}_2,\mathbf{p}_1) = 1/3\mathbf{p}_1\mathbf{x}_1^2 - 1/2\mathbf{p}_1\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ *from Example 11. For illustrative purposes, we study only the multi-index* $(1,1)$ *whose parametric Bernstein coefficient is:*

$$\mathbf{b}_{(1,1)}(\mathbf{p}_1) = \frac{\binom{(1,1)}{(1,1)}}{\binom{(2,1)}{(1,1)}}1/4 - \frac{\binom{(1,1)}{(0,1)}}{\binom{(2,1)}{(0,1)}}1/2\mathbf{p}_1 + \frac{\binom{(1,1)}{(1,0)}}{\binom{(2,1)}{(1,0)}}0 + \frac{\binom{(1,1)}{(0,0)}}{\binom{(2,1)}{(0,0)}}1/2 = 0.625 - 0.5\mathbf{p}_1. \tag{4.21}$$

*Applying the same scheme to the other multi-indices, we obtain the parametric Bernstein coefficients* $\mathbf{b}_{(0,0)}(\mathbf{p}_1) = 0.5$, $\mathbf{b}_{(0,1)}(\mathbf{p}_1) = 0.5 - 0.5\mathbf{p}_1$, $\mathbf{b}_{(1,0)}(\mathbf{p}_1) = 0.5$, $\mathbf{b}_{(1,1)}(\mathbf{p}_1) = 0.625 - 0.5\mathbf{p}_1$, $\mathbf{b}_{(2,0)}(\mathbf{p}_1) = 0.334\mathbf{p}_1 + 0.5$, *and* $\mathbf{b}_{(2,1)}(\mathbf{p}_1) = 0.75 - 0.1667\mathbf{p}_1$.

#### Properties of Parametric Bernstein Coefficients

We now extend range enclosing and sharpness properties defined on polynomials (see Section 4.2.1) to the parametric case. Let $P \subset \mathbb{R}^m$ be a polytope.

**Lemma 4** (Parametric Range Enclosing)**.**

$$\min_{\mathbf{i}\in I^\pi}\min_{\mathbf{p}\in P}\mathbf{b_i}(\mathbf{p}) \leq \pi(\mathbf{x},\mathbf{p}) \leq \max_{\mathbf{i}\in I^\pi}\max_{\mathbf{p}\in P}\mathbf{b_i}(\mathbf{p}), \tag{4.22}$$

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

**Lemma 5** (Parametric Sharpness)**.**

$$\text{for all } \mathbf{i} \in \mathcal{V}_\mathbf{d} \text{ and } \mathbf{p} \in P, \mathbf{b_i}(\mathbf{p}) = \pi(\mathbf{i}/\mathbf{d},\mathbf{p}), \tag{4.23}$$

*where* $\mathbf{b_i}(\mathbf{p})$*, for* $\mathbf{i} \in I^\pi$*, are the parametric Bernstein coefficients of* $\pi$ *and* $\mathcal{V}_\mathbf{d}$ *is the set of vertices of the box* $[0,\mathbf{d}_1] \times [0,\mathbf{d}_2] \times \ldots [0,\mathbf{d}_n]$*.*

*Proof.* Both Lemma 4 and Lemma 5 directly follow from Lemma 1 and 2, respectively. In fact any instantiation of the parametric case holds by the standard properties of Bernstein coefficients (see Lemmas 1 and 2). This is true for every parameter value $\mathbf{p} \in P$, which implies the validity of Lemma 4 and Lemma 5. □

Similarly to the standard Bernstein coefficients properties, the parametric coefficients can be used to get information about the evaluation of a parametric polynomial $\pi(\mathbf{x},\mathbf{p})$ over the unit box under the influence of the parameters $\mathbf{p} \in P$. Precisely:

1. The parametric range enclosing property can be used to find lower bound and upper bound;
2. The parametric sharpness property tells us where a parametric control point matches exactly the value of the parametric polynomial $\pi(\mathbf{x},\mathbf{p})$.

**Example 13.** *Consider the 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 Bernstein coefficients (from Example 12), and the parameter set* $P = [-0.1, 0.1]$*. Figure 4.4 shows two images of* $\pi$ *over the unit box and parameter* $\mathbf{p}_1$ *with values* $-0.1$ *and* $0.1$ *(gray areas), and the parametric control points ranging over the parameter set* $P = [-0.1, 0.1]$ *(in black). Also in this case, it is possible to see how the parametric range enclosing and sharpness properties hold.*

### 4.2.3 Computation of upper bound and lower bound

The parametric range enclosing property can be used to determine an upper and lower bound of a parametric polynomial $\pi(\mathbf{x},\mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ with $\mathbf{x} \in [0,1]^n$ and $\mathbf{p} \in P \subset \mathbb{R}^m$ polytope. We now define the algorithm maxBernCoeff (see Algorithm 1) that, exploiting Bernstein coefficients, determines an upper bound of a polynomial over the unit box and a polytopic parameter set.

maxBernCoeff begins by computing the set $B_\pi$ of all the parametric Bernstein coefficients of the polynomial $\pi(\mathbf{x},\mathbf{p})$. This can be done using Equation 4.20. Thus, for each coefficient $\mathbf{b_i}(\mathbf{p}) \in B_\pi$, the algorithm maximizes $\mathbf{b_i}(\mathbf{p})$ over the parameter set $P$ and updates the current maximum $\bar{b}$. Note that if $\mathbf{b_i}(\mathbf{p})$ is linear in $\mathbf{p}$, the optimization over $P$ can be carried out using linear programming. At the end of the loop, $\bar{b}$ will be an upper bound on the polynomial $\pi(\mathbf{x},\mathbf{p})$ over the unit box and parameter set $P$.

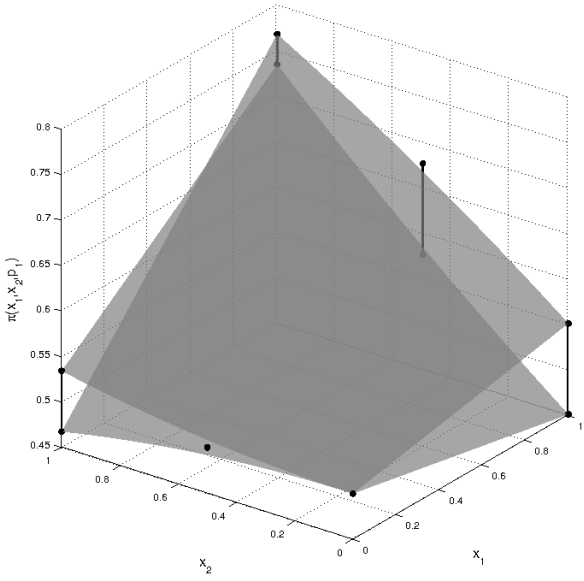



Figure 4.4: The polynomial $\pi(\mathbf{x}_1, \mathbf{x}_2, \mathbf{p}_1) = 1/3\mathbf{x}_1^2\mathbf{p}_1 - 1/2\mathbf{p}_1\mathbf{x}_2 + 1/4\mathbf{x}_1\mathbf{x}_2 + 1/2$ over the unit box with parameter values $-0.1$ and $0.1$ (in gray), and its parametric control points control points with $\mathbf{p}_1 \in [-0.1, 0.1]$ (in black).

**Algorithm 1** Compute maximum parametric Bernstein Coefficient

1: **function** MAXBERNCOEFF($\pi, P$) ▷ $\pi(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}, P \subset \mathbb{R}^m$ polytope
2: $B_\pi \leftarrow$BERNCOEFFS($\pi$) ▷ Compute Bernstein coefficients
3: $\overline{b} \leftarrow -\infty$ ▷ Initialize current maximum
4: **for** $\mathbf{b_i}(\mathbf{p}) \in B_\pi$ **do**
5: $b \leftarrow \max_{\mathbf{p} \in P} \mathbf{b_i}(\mathbf{p})$ ▷ Maximize current coefficient
6: $\overline{b} \leftarrow \max\{\overline{b}, b\}$ ▷ Update maximum
7: **end for**
8: **return** $\overline{b}$
9: **end function**

Similarly, we can define the algorithm MINBERNCOEFF that computes a lower bound of $\pi(\mathbf{x}, \mathbf{p})$. MINBERNCOEFF can be easily obtained from MAXBERNCOEFF by initializing the bound $\overline{b}$ with $+\infty$ (Line 3) and replacing maximum operators with minimums.

### 4.2.4 Summary

Bernstein coefficients can be used to bound the image of a polynomial, eventually parametric, over the unit box. However, it is very unlikely that the trajectories generated by a dynamical system will only span over unit boxes. The question is then how to exploit Bernstein coefficients outside unit boxes and perhaps outside box domains.

In the next chapters, we will define some techniques of set image approximation that involve generic boxes, parallelotopes, and sets generated by the intersection of several parallelotopes. These techniques will be used to compute over-approximations of sets of states reached by polynomial dynamical systems and will allow us to synthesize sets of parameters so that all the trajectories of the treated dynamical system satisfy a given STL property.

Moreover, at the end of the chapter, we will present some techniques to improve the efficiency with which Bernstein coefficients are computed and to obtain tighter bounds. These improvements will lead to faster and preciser reachability and parameter synthesis algorithms.

## 4.3 Boxes

We now go into the problem of the image computation. We begin by considering one of simplest class of sets called *boxes*, also known as *hyper-rectangles*.

**Definition 8** (Box)**.** *A set $B \subset \mathbb{R}^n$ is a* box *if and only if it can be expressed as the product of $n$ intervals, that is:*

$$B = [\underline{x}_1, \overline{x}_1] \times \ldots [\underline{x}_n, \overline{x}_n] = \prod_{i=1}^{n} [\underline{x}_i, \overline{x}_i], \tag{4.24}$$

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

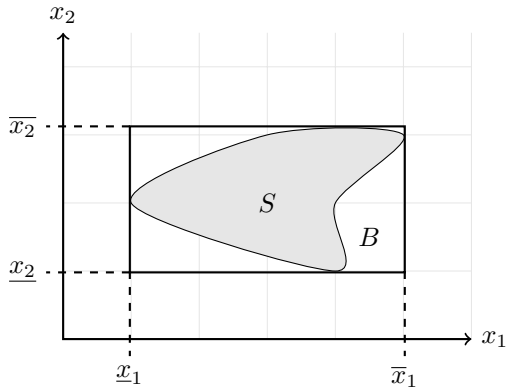



Figure 4.5: A set $S$ and its enclosing box $B$.

It is easy to see that a box is a polytope. In fact, a box $B = [\underline{x}_1, \overline{x}_1] \times \dots [\underline{x}_n, \overline{x}_n]$ can be represented as a polytope $\langle D, \mathbf{c} \rangle$ where:

$$D = \begin{pmatrix} 1 & 0 & \dots & 0 \\ -1 & 0 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & 0 & 1 \\ 0 & \dots & 0 & -1 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} \overline{x}_1 \\ -\underline{x}_1 \\ \vdots \\ \overline{x}_n \\ -\underline{x}_n \end{pmatrix} \tag{4.25}$$

Since a box is a polytope, it can be also represented by its vertices (see Section 4.1.1). However, the interval and constraint representations require only $2n$ elements against $2^n$ vertices.

### 4.3.1 Box-Based Set Image

We now restrict the set image problem to boxes. Indeed, we define a method to determine a box $X' \subset \mathbb{R}^n$ such that $X' \supseteq \mathbf{f}(X, P)$ with $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ polynomial, $X \subset \mathbb{R}^n$ box, and $P \subset \mathbb{R}^m$ polytope. The final target is to define the algorithm ReachStep that realizes a single reachability step of a polynomial dynamical system using boxes to represent the sets of states reached by the system. The technique we present is an extension of the approach developed in [57, 181] (were boxes and standard Bernstein coefficients are also used) to parametric dynamical systems.

We recall that the properties of Bernstein coefficients are valid only for the unit box domain. However, we can try to exploit Bernstein coefficients on a generic box $X = \langle D, \mathbf{c} \rangle = [\underline{x}_1, \overline{x}_1] \times \dots \times [\underline{x}_n, \overline{x}_n]$ defining a linear transformation $\mathbf{v}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$

that maps the unit box to $X$. The map $\mathbf{v}(\mathbf{x})$ is defined as follows:

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} \overline{x}_1 - \underline{x}_1 & 0 & \dots & 0 \\ 0 & \overline{x}_2 - \underline{x}_2 & \dots & 0 \\ \vdots & & & \vdots \\ 0 & \dots & 0 & \overline{x}_n - \underline{x}_n \end{pmatrix} \begin{pmatrix} \mathbf{x}_1 \\ \mathbf{x}_2 \\ \vdots \\ \mathbf{x}_n \end{pmatrix} + \begin{pmatrix} \underline{x}_1 \\ \underline{x}_2 \\ \vdots \\ \underline{x}_n \end{pmatrix}. \tag{4.26}$$

Now, suppose we want to bound a parametric polynomial $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ over a box $X \subset \mathbb{R}^n$ and a polytopic parameter set $P \subset \mathbb{R}^m$. We can compute the map $\mathbf{v}(\mathbf{x})$ for $X$ and then consider the composition $\pi(\mathbf{v}(\mathbf{x}), \mathbf{p})$. It holds that $\pi(X, P) = \pi(\mathbf{v}([0,1]^n), P)$. Note that the domain of $\pi(\mathbf{v}(\mathbf{x}), \mathbf{p})$ in $\mathbf{x}$ is the unit box. This means that we can use the Bernstein coefficients of $\pi(\mathbf{v}(\mathbf{x}), \mathbf{p})$, and in particular the algorithm maxBernCoeff (see Algorithm 1), to indirectly bound $\pi(\mathbf{x}, \mathbf{p})$ over $X$ and $P$.

We can define the function bound (Algorithm 2) that receives in input a parametric polynomial $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, a box $X \subset \mathbb{R}^n$, and a polytopic parameter set $P \subset \mathbb{R}^m$. Exploiting the map $\mathbf{v}(\mathbf{x})$ and Bernstein coefficients, it returns an upper bound $\overline{b} \in \mathbb{R}$ of $\pi(X, P)$ such that:

$$\overline{b} \geq \max_{\mathbf{x} \in [0,1]^n, \mathbf{p} \in P} \pi(\mathbf{v}(\mathbf{x}), \mathbf{p}) = \max_{\mathbf{x} \in X, \mathbf{p} \in P} \pi(\mathbf{x}, \mathbf{p}). \tag{4.27}$$

**Algorithm 2** Bound polynomial over box and polytopic parameter set

```
1: function BOUND(π, X, P)                          ▷ X ⊂ ℝ^n box, P ⊂ ℝ^m polytope
2:     v(x) ←MAPUNITBOXTO(X)                        ▷ Map [0,1]^n to X
3:     b̄ ←MAXBERNCOEFF(π(v(x)), P)                  ▷ Compute maximum coefficient
4:     return b̄
5: end function
```

The function bound, using the function mapUnitBoxTo based on Equation 4.26, computes the transformation $\mathbf{v}(\mathbf{x})$ (Line 2) that maps the unit box to the given box $X$. Then, through the function maxBernCoeff, it computes the Bernstein coefficients of $\pi(\mathbf{v}(\mathbf{x}), \mathbf{p})$ and returns their maximum over the parameter set $P$ (Line 3).

Now that we know how to bound a polynomial over a generic box, we can focus on the image set computation. Let $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ be a parametric polynomial, $X, X' \subset \mathbb{R}^n$ be boxes, and $P \subset \mathbb{R}^m$ be a polytopic parameter set. In order for $X' = \langle D, \mathbf{c}' \rangle$ to be an over-approximation box of the set $\mathbf{f}(X, P)$ it must hold that:

$$\mathbf{c}'_j \geq \max_{\mathbf{x} \in X, \mathbf{p} \in P} D_j \mathbf{f}(\mathbf{x}, \mathbf{p}) \text{ for all } j \in \{1, \dots, 2n\}. \tag{4.28}$$

Since the functions $D_j \mathbf{f}(\mathbf{x}, \mathbf{p})$ are polynomials and $X \subset \mathbb{R}^n$ and $P \subset \mathbb{R}^m$ are a box and a polytope, respectively, the upper bounds $\mathbf{c}'_j$, for $j \in \{1, \dots, 2n\}$, can obtained exploiting the algorithm bound. The optimizations of the functions $D_j \mathbf{f}(\mathbf{x}, \mathbf{p})$ lead to the offsets $\mathbf{c}'_j$ that associated with the template $D$ generate the over-approximation box $X' = \langle D, \mathbf{c}' \rangle \supseteq \mathbf{f}(X, P)$.

We collect these operations in the algorithm reachStep (Algorithm 3) that receives

in input a box $X \subset \mathbb{R}^n$ and a polytope $P \subset \mathbb{R}^m$, and, exploiting the algorithm BOUND, computes a box $X'$ that over-approximates the image $\mathbf{f}(X, P)$.

**Algorithm 3** Box-based reachability step

```
1: function REACHSTEP(X, P)                    ▷ X = ⟨D, c⟩ ⊂ ℝ^n box, P ⊂ ℝ^m polytope
2:     for i ∈ {1, ..., 2n} do
3:         c'_i ← BOUND(D_i f(x, p), X, P)      ▷ Bound box directions
4:     end for
5:     return X' = ⟨D, c'⟩                     ▷ Construct and return the new box
6: end function
```

**Example 14.** *Consider the dynamics of the discrete-time SIR epidemic system of Example 6 with state variables and parameters grouped in the vectors* $\mathbf{x} = (s, i, r)$ *and* $\mathbf{p} = (\beta, \gamma)$*:*

$$\mathbf{f}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f}_s(\mathbf{x}, \mathbf{p}) = s - \beta si \\ \mathbf{f}_i(\mathbf{x}, \mathbf{p}) = i + \beta si - \gamma i \\ \mathbf{f}_r(\mathbf{x}, \mathbf{p}) = r + \gamma i \end{pmatrix} \tag{4.29}$$

*Let* $s \in [0.80, 0.85]$*,* $i \in [0.15, 0.20]$*,* $r \in [0, 0]$*, and* $\beta \in [0.35, 0.36], \gamma \in [0.05, 0.06]$ *whose box-set representations are* $X = \langle D, \mathbf{c} \rangle = [0.80, 0.85] \times [0.15, 0.20] \times [0, 0]$ *and* $P = [0.35, 0.36] \times [0.05, 0.06]$*.*

*The map* $\mathbf{v}$ *that transforms the unit box to* $X$ *is:*

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} 0.85 - 0.80 & 0 & 0 \\ 0 & 0.20 - 0.15 & 0 \\ 0 & 0 & 0 - 0 \end{pmatrix} \begin{pmatrix} s \\ i \\ r \end{pmatrix} + \begin{pmatrix} 0.80 \\ 0.15 \\ 0 \end{pmatrix} \tag{4.30}$$

*whose composition with the system dynamics leads to the function:*

$$\mathbf{h}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f_s}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = (0.05s + 0.80) - \beta(0.05s + 0.80)(0.05i + 0.15) \\ \mathbf{f_i}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = (1 - \gamma)(0.05i + 0.15) + \beta(0.05s + 0.80)(0.05i + 0.15) \\ \mathbf{f_r}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = \gamma(0.05i + 0.15) \end{pmatrix} \tag{4.31}$$

*Composing the function* $\mathbf{h}(\mathbf{x}, \mathbf{p})$ *with the box template* $D$*, we obtain the functions:*

$$\begin{aligned} D_1\mathbf{h}(\mathbf{x}, \mathbf{p}) &= (0.05s + 0.80) - \beta(0.05s + 0.80)(0.05i + 0.15) \\ D_2\mathbf{h}(\mathbf{x}, \mathbf{p}) &= -(0.05s + 0.80) + \beta(0.05s + 0.80)(0.05i + 0.15) \\ D_3\mathbf{h}(\mathbf{x}, \mathbf{p}) &= (1 - \gamma)(0.05i + 0.15) + \beta(0.05s + 0.80)(0.05i + 0.15) \\ D_4\mathbf{h}(\mathbf{x}, \mathbf{p}) &= -(1 - \gamma)(0.05i + 0.15) - \beta(0.05s + 0.80)(0.05i + 0.15) \\ D_5\mathbf{h}(\mathbf{x}, \mathbf{p}) &= \gamma(0.05i + 0.15) \\ D_6\mathbf{h}(\mathbf{x}, \mathbf{p}) &= -\gamma(0.05i + 0.15) \end{aligned} \tag{4.32}$$

*whose Bernstein coefficients are:*

$$\begin{aligned}
B_{D_1\mathbf{h}}(\mathbf{p}) &= \{0.80 - 0.12\beta, 0.80 - 0.16\beta, 0.85 - 0.13\beta, 0.85 - 0.17\beta\} \\
B_{D_2\mathbf{h}}(\mathbf{p}) &= \{0.12\beta - 0.80, 0.16\beta - 0.80, 0.13\beta - 0.85, 0.17\beta - 0.85\} \\
B_{D_3\mathbf{h}}(\mathbf{p}) &= \{0.12\beta - 0.15\gamma + 0.15, 0.16\beta - 0.20\gamma + 0.20, \\
&\quad\ 0.13\beta - 0.15\gamma + 0.15, 0.17\beta - 0.20\gamma + 0.20\} \\
B_{D_4\mathbf{h}}(\mathbf{p}) &= \{0.15\gamma - 0.12\beta - 0.15, 0.20\gamma - 0.16\beta - 0.20, \\
&\quad\ 0.15\gamma - 0.13\beta - 0.15, 0.20\gamma - 0.17\beta - 0.20\} \\
B_{D_5\mathbf{h}}(\mathbf{p}) &= \{0.15\gamma, 0.2\gamma\} \\
B_{D_6\mathbf{h}}(\mathbf{p}) &= \{-0.15\gamma, -0.2\gamma\}
\end{aligned} \tag{4.33}$$

*that optimized on the parameter set $P$ lead to the upper bounds:*

$$\begin{aligned}
0.8054 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_1\mathbf{h}}(\mathbf{p})\} & -0.7424 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_2\mathbf{h}}(\mathbf{p})\} \\
0.2512 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_3\mathbf{h}}(\mathbf{p})\} & -0.1830 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_4\mathbf{h}}(\mathbf{p})\} \\
0.0120 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_5\mathbf{h}}(\mathbf{p})\} & -0.0075 &= \max_{\mathbf{p}\in P}\{\mathbf{b_i}(\mathbf{p}) \in B_{D_6\mathbf{h}}(\mathbf{p})\}
\end{aligned} \tag{4.34}$$

*Note that the signs of the coefficients for the directions $D_2$, $D_4$, and $D_6$ are swapped in the interval representation of $X'$. The over-approximation of the reachable set is $X' = [0.7424, 0.8054] \times [0.1830, 0.2512] \times [0.0075, 0.0120]$. Box $X'$ is shown in Figure 4.6 (in white) together with some reachable points (in black) computed with a sampling-based method (see Section 3.3.1). Note how $X'$ includes all the computed reachable points.*

We have developed a method based on boxes for over-approximating the image of parametric polynomials. However, the approximation introduced by a box can be quite rough due to the differences between the approximated set and the enclosing box. In the next section we will introduce a new technique based on parallelotopes, a more flexible class of sets that represents a good trade-off between computational complexity and approximation accuracy.

## 4.4 Parallelotopes

In this section we define a set image approximation method based on *parallelotopes*, i.e., the $n$-dimensional generalization of parallelepipeds. The use of parallelotopes makes the method more flexible as far as the choice of the initial set is concerned and it allows one to obtain better approximations.

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

**Definition 9** (Parallelotope Constraint Representation)**.** *Let $\Lambda \in \mathbb{R}^{2n\times n}$ be a template matrix such that, for each $i \in \{1, 2, \ldots, n\}$, $\Lambda_i = -\Lambda_{i+n}$, and let $\mathbf{c} \in \mathbb{R}^{2n}$. The parallelotope $\mathcal{P}$ generated by $\Lambda$ and $\mathbf{c}$ is:*

$$\mathcal{P} = \mathcal{P}_{con}(\Lambda, \mathbf{c}) = \langle \Lambda, \mathbf{c} \rangle = \{\mathbf{x} \mid \Lambda\mathbf{x} \leq \mathbf{c}\}. \tag{4.35}$$

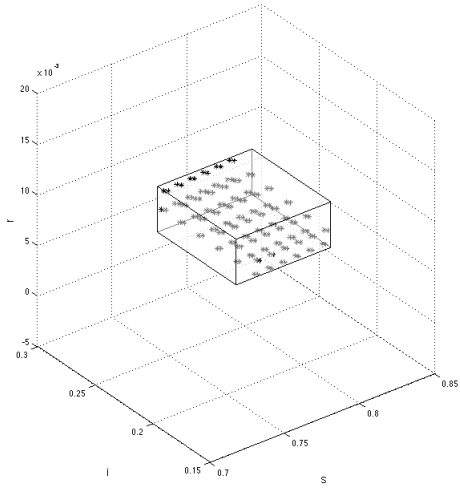



Figure 4.6: Box-based set image computation of SIR dynamical system. The figure shows the constructed box (in white) and some reachable points computed with a sampling-based method (in black).

The above representation is called *constraint representation* (see Figure 4.7a). Another way to characterize parallelotopes, similar to the one adopted for zonotopes [50], is to fix a point of origin and use vectors to define the parallelotope.

**Definition 10** (Parallelotope Generator Representation)**.** *Let* $G = \{\mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^n\}$ *be a set of* $n$ *linearly independent vectors in* $\mathbb{R}^n$ *and* $\mathbf{q}$ *be a point in* $\mathbb{R}^n$*. The parallelotope* $\mathcal{P}$ *generated by* $G$ *and* $\mathbf{q}$ *is:*

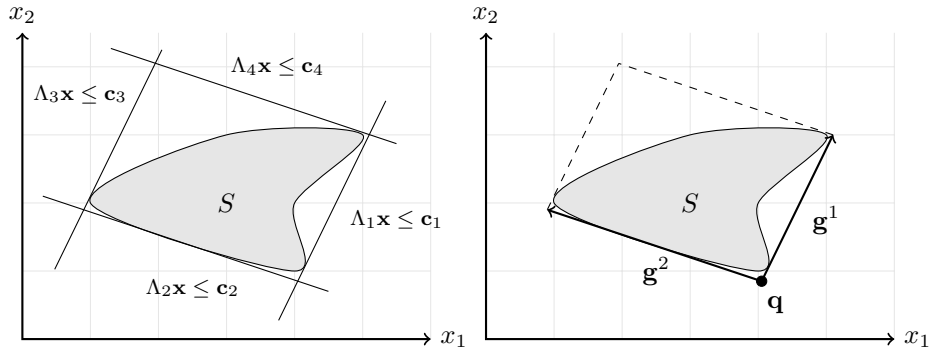
$$\mathcal{P} = \mathcal{P}_{gen}(G, \mathbf{q}) = \{\mathbf{q} + \sum_{j=1}^{n} \mathbf{x}_j \mathbf{g}^j \mid (\mathbf{x}_1, \ldots, \mathbf{x}_n) \in [0,1]^n\}. \tag{4.36}$$

This representation is said *generator representation* (see Figure 4.7b). The vectors $\mathbf{g}^1, \mathbf{g}^2, \ldots, \mathbf{g}^n$ 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:

$$\mathcal{P} = \mathcal{P}_{gen}(G, \mathbf{q}) = \{\gamma_{(\mathbf{q},G)}(\mathbf{x}) \mid \mathbf{x} \in [0,1]^n\}, \tag{4.37}$$

where $\mathbf{x} = (\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n)$ and $\gamma_{(\mathbf{q},G)}(\mathbf{x})$ is the linear function defined as:

$$\gamma_{(\mathbf{q},G)}(\mathbf{x}) = \mathbf{q} + \sum_{j=1}^{n} \mathbf{x}_j \mathbf{g}^j. \tag{4.38}$$




(a) Constraint representation. (b) Generator representation.

Figure 4.7: A set $S$ and two enclosing parallelotopes.

This notation emphasizes the aspect that a parallelotope can be seen as the affine transformation of the unit box. This suggests us that there might be a way to combine Bernstein properties and parallelotopes.

### 4.4.1 Representation Conversion

A parallelotope $X$ can be equivalently represented using constraints and generators. The choice of the representation depends on the operation that we want to apply to $X$. Later, in our image approximation algorithm, we will need a representation conversion, that is, given $X = \mathcal{P}_{con}(\Lambda, \mathbf{c})$, we want to compute $\mathcal{P}_{gen}(G, \mathbf{q}) = X$, and vice versa.

#### From Constraints to Generators

Given the constrain representation $\mathcal{P}_{con}(\Lambda, \mathbf{c})$ we want to find a generator set $G$ and a base vertex $\mathbf{q}$ such that $\mathcal{P}_{gen}(G, \mathbf{q}) = \mathcal{P}_{con}(\Lambda, \mathbf{c})$. First, we rewrite the inequalities given by the template $\Lambda$ and offsets $\mathbf{c}$ in form:

$$-\mathbf{c}_{n+i} \leq \Lambda_i \mathbf{x} \leq \mathbf{c}_i, \tag{4.39}$$

for all $i \in \{1, \ldots, n\}$. The based vertex $\mathbf{q}$ and the coordinates of the vertex $\mathbf{i}_i$ that lies on the straight line passing through the $i$-th generator vector applied to the vertex $\mathbf{q}$, are the solutions of the linear systems:

$$\begin{pmatrix} \Lambda_1 \\ \vdots \\ \Lambda_n \end{pmatrix} \mathbf{x} = \begin{pmatrix} -\mathbf{c}_{n+1} \\ \vdots \\ -\mathbf{c}_{2n} \end{pmatrix} \qquad \begin{pmatrix} \Lambda_1 \\ \vdots \\ \Lambda_i \\ \vdots \\ \Lambda_n \end{pmatrix} \mathbf{x} = \begin{pmatrix} -\mathbf{c}_{n+1} \\ \vdots \\ \mathbf{c}_i \\ \vdots \\ -\mathbf{c}_{2n} \end{pmatrix}. \tag{4.40}$$

Hence, the $i$-th generator $\mathbf{g}^i$ is the difference between the vertex $\mathbf{v}_i$ and the base vertex $\mathbf{q}$, i.e., $\mathbf{g}^i = \mathbf{v}_i - \mathbf{q}$. Finally, the versor $\mathbf{u}_i$ and the generator norm $\boldsymbol{\beta}_i$ such that $\mathbf{g}^i = \boldsymbol{\beta}_i \mathbf{u}_i$ are given by $\boldsymbol{\beta}_i = \|\mathbf{g}^i\|$ and $\mathbf{u}_i = \dfrac{\mathbf{g}^i}{\|\mathbf{g}^i\|}$.

**From Generators to Constraints**

We now consider the inverse conversion: given a generator representation $\mathcal{P}_{gen}(G, \mathbf{q})$ we want to find a template $\Lambda$ and a vector $\mathbf{c}$ such that $\mathcal{P}_{con}(\Lambda, \mathbf{c}) = \mathcal{P}_{gen}(G, \mathbf{q})$. Let $G = \{\mathbf{g}^1, \ldots, \mathbf{g}^n\}$ be the generator set and $\mathbf{q}$ be the base vertex.

As first step we calculate the points $\mathbf{p}_1, \ldots, \mathbf{p}_n$ that are traversed by the hyperplanes correspondent to the constrains. Each $\mathbf{p}_i$ is obtained by adding the generator $\mathbf{g}^i$ to the base vertex $\mathbf{q}$, i.e., $\mathbf{p}_i = \mathbf{q} + \mathbf{g}^i$. The $i$-th constraint of the parallelotope lies on the hyperplane (whose equation is $\pi_i = \mathbf{a}_i\mathbf{x} + \mathbf{c}_i$) passing through the points $\mathbf{q}, \mathbf{p}_1, \ldots, \mathbf{p}_{i-1}, \mathbf{p}_{i+1}, \ldots, \mathbf{p}_n$. The equation $\pi_{i+n} = \mathbf{a}_{i+n}\mathbf{x} + \mathbf{c}_{i+n}$ of the hyperplane parallel to $\pi_i$ can be found by translating the vertices used to compute $\pi_i$ by the vector $\mathbf{g}^i$, i.e., $\pi_{i+n}$ is the hyperplane passing through the points $\mathbf{q}+\mathbf{g}^i, \mathbf{p}_1+\mathbf{g}^i, \ldots, \mathbf{p}_{i-1}+\mathbf{g}^i, \mathbf{p}_{i+1}+\mathbf{g}^i, \ldots, \mathbf{p}_n + \mathbf{g}^i$. Let $\underline{\mathbf{c}}_i$ and $\overline{\mathbf{c}}^i$ be defined as $\underline{\mathbf{c}}_i = \min\{d_i, d_{i+n}\}$ and $\overline{\mathbf{c}}_i = \max\{d_i, d_{i+n}\}$.

Since $\pi_i$ and $\pi_{i+n}$ are parallel it must hold $\mathbf{a}_i = \mathbf{a}_{i+n}$. Hence, the portion of the parallelotope included between $\pi_i$ and $\pi_{i+n}$ is the solution of the inequalities $\underline{\mathbf{c}}_i \leq \mathbf{a}_i\mathbf{x} \leq \overline{\mathbf{c}}_i$, which means that the $i$-th and $(i+n)$-th rows of the template matrix are $\Lambda_i = \mathbf{a}_i$ and $\Lambda_{i+n} = -\mathbf{a}_i$, while the $i$-th and $(i+n)$-th offset elements are $\mathbf{c}_i = \overline{\mathbf{c}}_i$ and $\mathbf{c}_{i+n} = -\underline{\mathbf{c}}_i$.

### 4.4.2 Parallelotope-Based Set Image

We now focus on the set image computation problem. Let $X \subset \mathbb{R}^n$ be a parallelotope whose constraint representation is $X = \mathcal{P}_{con}(\Lambda, \mathbf{c})$ and generator representation is $X = \mathcal{P}_{gen}(G, \mathbf{q})$ (recall that $\gamma_{(\mathbf{q},G)}(\mathbf{x})$ is the linear function that defines the generator representation, see Equation 4.38). We are interested in computing a parallelotope $X' \subset \mathbb{R}^n$ such that $\mathbf{f}(X, P) \subseteq X'$. Adopting the template $\Lambda$ of $X$, we can obtain $X'$ by determining the offsets $\mathbf{c}' \in \mathbb{R}^{2n}$ such that $\mathbf{f}(X, P) \subseteq \mathcal{P}_{con}(\Lambda, \mathbf{c}') = X'$.

In order for a parallelotope $X' = \langle \Lambda, \mathbf{c}' \rangle$ in constraint representation to be an over-approximation of the set $\mathbf{f}(X, P)$ it must hold that:

$$\mathbf{c}'_j \geq \max_{\mathbf{x} \in X, \mathbf{p} \in P} \Lambda_j \mathbf{f}(\mathbf{x}, \mathbf{p}) \text{ for all } j \in \{1, \ldots, 2n\}. \tag{4.41}$$

Then the inclusion $\mathbf{f}(X, P) \subseteq \mathcal{P}_{con}(\Lambda, \mathbf{c})$ is guaranteed.

This condition can be rewritten using the generator representation as:

$$\mathbf{c}'_j \geq \max_{\mathbf{x} \in [0,1]^n, \mathbf{p} \in P} \mathbf{h}_j(\mathbf{x}, \mathbf{p}) \text{ for all } j \in \{1, \ldots, 2n\}, \tag{4.42}$$

where $\mathbf{h}_j(\mathbf{x}, \mathbf{p}) = \Lambda_j \mathbf{f}(\gamma_{(\mathbf{q},G)}(\mathbf{x}), \mathbf{p})$. Note that $\mathbf{h}_j(\mathbf{x}, \mathbf{p})$ is a polynomial function of $\mathbf{x}$ whose domain is the unit box. Therefore, we can use Bernstein coefficients to compute an upper bound $\mathbf{c}'_j \in \mathbb{R}$ of the function $\mathbf{h}_j(\mathbf{x}, \mathbf{p})$.

Let $B_{\mathbf{h}_j}(\mathbf{p}) = \{\mathbf{b}_\mathbf{i}(\mathbf{p}) \mid \mathbf{i} \in I^{\mathbf{h}_j}\}$ be the set of Bernstein coefficients of the function $\mathbf{h}_j(\mathbf{x}, \mathbf{p})$.

**Theorem 1.** *Let* $\mathbf{c}' = (\mathbf{c}'_1, \ldots, \mathbf{c}'_{2n})$ *be such that for each* $j \in \{1, \ldots, 2n\}$ *the component* $\mathbf{c}'_j$ *is defined as:*

$$\mathbf{c}'_j = \max\{\mathbf{b}_{\mathbf{i}}(\mathbf{p}) \mid \mathbf{i} \in I^{\mathbf{h}_j}, \mathbf{p} \in P\}. \tag{4.43}$$

*The vector* $\mathbf{c}'$ *satisfies the inclusion* $\mathbf{f}(X, P) \subseteq \mathcal{P}_{con}(\Lambda, \mathbf{c}')$.

*Proof.* It follows directly from Lemma 4 (range enclosing property of parametric Bernstein coefficients) and Equation 4.42. □

Before defining the algorithm for bounding the reachable set, we rewrite the generator representation explicitly distinguishing between the directions of the generators and their lengths.

Let $G = \{\mathbf{g}^1, \ldots, \mathbf{g}^n\}$ be a set of generators. Let $\boldsymbol{\beta}_i \in \mathbb{R}$ be the Euclidean norm of $\mathbf{g}^i$ and $\mathbf{u}^i$ be the versor of $\mathbf{g}^i$, i.e., $\mathbf{g}^i = \boldsymbol{\beta}_i \mathbf{u}^i$. Let $\boldsymbol{\beta} = (\boldsymbol{\beta}_1, \ldots, \boldsymbol{\beta}_n)$ and $U = \{\mathbf{u}^1, \ldots, \mathbf{u}^n\}$. With a slight abuse of notation, we rewrite the generator representation of a parallelotope $X$ as:

$$X = \mathcal{P}_{gen}(\mathbf{q}, \boldsymbol{\beta}, U) = \{\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) \mid \mathbf{x} \in [0, 1]^n\} \tag{4.44}$$

where $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ is the linear function of $\mathbf{x}$ defined as:

$$\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) = \mathbf{q} + \sum_{j=1}^{n} \mathbf{x}_j \boldsymbol{\beta}_j \mathbf{u}^j. \tag{4.45}$$

When working on parallelotopes using the constraint representation, we can fix a template $\Lambda$ and let the offset $\mathbf{c}$ free. In this way we symbolically denote an infinite set of parallelotopes. On the generator representation, this corresponds to fixing a set $U$ of $n$ versors and letting the base vertex $\mathbf{q}$ and vector lengths $\boldsymbol{\beta}$ free.

Let us now focus on a single reachability step: given a parallelotope $X = \mathcal{P}_{con}(\Lambda, \mathbf{c}) \subset \mathbb{R}^n$ and a polytopic parameter set $P \subset \mathbb{R}^m$, we want to compute $\mathbf{c}' \in \mathbb{R}^{2n}$ such that $\mathcal{P}_{con}(\Lambda, \mathbf{c}') \subset \mathbb{R}^n$ over-approximates the set $\mathbf{f}(X, P)$. The set $\mathbf{f}(X, P)$ can be characterized as:

$$\mathbf{f}(X, P) = \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, [0, 1]^n), P) = \{\mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \mid \mathbf{x} \in [0, 1]^n, \mathbf{p} \in P\}, \tag{4.46}$$

where $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ is the generator function of $X$. Finding the base vertex $\mathbf{q}'$ and the vector lengths $\boldsymbol{\beta}'$ such that $\mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, [0, 1]^n), P)) \subseteq \gamma_U(\mathbf{q}', \boldsymbol{\beta}', [0, 1]^n)$ means obtaining an over-approximation in generator representation of $\mathbf{f}(X, P)$. These $\mathbf{q}'$ and $\boldsymbol{\beta}'$ can be found thanks to the constraint representation.

Let $\Lambda$ be the template matrix associated with the generator versors $U$. The offset $\mathbf{c}'$ such that $\mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, [0, 1]^n), P) \subseteq \mathcal{P}_{con}(\Lambda, \mathbf{c}')$ can be calculated using Theorem 1. That is, an upper bound of $\Lambda_j \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, [0, 1]^n), P)$ can be found using the algorithm maxBernCoeff (see Algorithm 1) based on Bernstein coefficients.

Let us algorithmically formalize these ideas. First, we define the algorithm bound (Algorithm 4) that receives in input a parametric polynomial $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, a parallelotope $X = \langle \Lambda, \mathbf{c} \rangle$ in constraint representation, and a polytopic parameter set $P \subset \mathbb{R}^m$, and returns an upper bound $\overline{b} \in \mathbb{R}$ of $\pi(X, P)$.

**Algorithm 4** Bound polynomial over parallelotope and polytopic parameter set

1: **function** BOUND($\pi, X, P$) ▷ $X = \langle \Lambda, \mathbf{c} \rangle \subset \mathbb{R}^n$ parallelotope, $P \subset \mathbb{R}^m$ polytope
2: $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) \leftarrow$CON2GEN($X$) ▷ Compute generator function
3: $\overline{b} \leftarrow$MAXBERNCOEFF($\pi(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), P)$ ▷ Compute maximum coefficient
4: **return** $\overline{b}$
5: **end function**

The algorithm BOUND, using the function CON2GEN, converts the parallelotope $X$ from the constraint to the generator representation (Line 2), obtaining the generator function $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$. Then, it composes the polynomial $\pi(\mathbf{x}, \mathbf{p})$ with the generator function $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ and, using the function MAXBERNCOEFF, it determines an upper bound $\overline{b}$ of $\pi(X, P)$.

Thanks to the algorithm BOUND, we can now develop the parallelotope-based set image algorithm REACHSTEP (Algorithm 5) that, given a parametric polynomial function $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$, receives in input a parallelotope $X = \langle \Lambda, \mathbf{c} \rangle \subset \mathbb{R}^n$ in constraint representation and a polytope $P \subset \mathbb{R}^m$, and returns a parallelotope $X = \langle \Lambda, \mathbf{c}' \rangle \subset \mathbb{R}^n$ that over-approximates the set $\mathbf{f}(X, P)$.

**Algorithm 5** Parallelotope-based reachability step

1: **function** REACHSTEP($X, P$) ▷ $X = \langle \Lambda, \mathbf{c} \rangle \subseteq \mathbb{R}^n$ parallelotope, $P \subseteq \mathbb{R}^m$ polytope
2: **for** $i \in \{1, \ldots, 2n\}$ **do**
3: $\mathbf{c}'_i \leftarrow$BOUND($\Lambda_i \mathbf{f}(\mathbf{x}, \mathbf{p}), X, P$)
4: **end for**
5: **return** $X' = \langle \Lambda, \mathbf{c}' \rangle$
6: **end function**

For each constraint of the parallelotope, the bounding functions $\Lambda_i \mathbf{f}(\mathbf{x}, \mathbf{p})$ are computed by composing the template directions $\Lambda_i$ with the polynomial $\mathbf{f}(\mathbf{x}, \mathbf{p})$. The upper bounds $\mathbf{c}'_i$ are determined with the function BOUND, where Bernstein coefficients are computed and the values $\mathbf{c}'_i$ are obtained by maximizing the parametric coefficients over the parameter set $P$. The vector $\mathbf{c}'$ is the offset that associated with the template $\Lambda$ constitutes the searched approximation set $X' = \langle \Lambda, \mathbf{c}' \rangle$ of $\mathbf{f}(X, P)$.

**Example 15.** *Consider the dynamics of the discrete-time SIR epidemic model of Example 6 with state variables and parameters grouped in the vectors* $\mathbf{x} = (s, i, r)$ *and* $\mathbf{p} = (\beta, \gamma)$*:*

$$\mathbf{f}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f}_s(\mathbf{x}, \mathbf{p}) = s - \beta si \\ \mathbf{f}_i(\mathbf{x}, \mathbf{p}) = i + \beta si - \gamma i \\ \mathbf{f}_r(\mathbf{x}, \mathbf{p}) = r + \gamma i \end{pmatrix} \tag{4.47}$$

*Let* $X = \mathcal{P}_{con}(\Lambda, \mathbf{c})$ *be the parallelotope in constraint representation defined as:*

$$X = \Lambda\mathbf{x} \leq \mathbf{c} = \begin{pmatrix} -1 & 0 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \mathbf{x} \leq \begin{pmatrix} -0.80 \\ -0.95 \\ 0.00 \\ 0.85 \\ 1.00 \\ 0.00 \end{pmatrix} \tag{4.48}$$

*and* $P = [0.35, 0.36] \times [0.05, 0.06]$ *be the parameter set with* $\beta \in [0.35, 0.36]$ *and* $\gamma \in [0.05, 0.06]$.

*The equivalent generator representation of* $X$ *is* $X = \mathcal{P}_{gen}(\mathbf{q}, \boldsymbol{\beta}, U)$*, with* $\mathbf{q} = (0.80, 0.15, 0)$*,* $\boldsymbol{\beta} = (0.0707, 0.0500, 0.0000)$*, and* $U = \{(0.7071, -0.7071, 0), (0, 1, 0), (0, 0, 1)\}$*, whose generator function is:*

$$\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) = \begin{pmatrix} \mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s \\ \mathbf{q}_2 - 0.7070\boldsymbol{\beta}_1 s + \boldsymbol{\beta}_2 i \\ \mathbf{q}_3 + r\boldsymbol{\beta}_3 \end{pmatrix} \tag{4.49}$$

*We now show how to bound the direction* $\Lambda_1 = (-1, 0, 0)$*. The composition* $\mathbf{h}(\mathbf{x}, \mathbf{p}) = \Lambda_1 \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$ *is:*

$$\begin{aligned} \mathbf{h}(\mathbf{x}, \mathbf{p}) &= (-1, 0, 0) \begin{pmatrix} \mathbf{f}_s(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ \mathbf{f}_i(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ \mathbf{f}_r(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \end{pmatrix} = -\mathbf{f}_s(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ &= -((\mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s) - \beta(\mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s)(\mathbf{q}_2 - 0.7070\boldsymbol{\beta}_1 s + \boldsymbol{\beta}_2 i)) \end{aligned} \tag{4.50}$$

*Instantiating the base vertex* $\mathbf{q}$ *and lengths* $\boldsymbol{\beta}$*, we obtain the set of coefficients:*

$$\begin{aligned} B_{\mathbf{h}} = \{&-0.8000 + 0.1200\beta, -0.8000 + 0.1600\beta, -0.8250 + 0.1038\beta, \\ &-0.8250 + 0.1450\beta, -0.8500 + 0.0850\beta, -0.8500 + 0.1275\beta\}\}, \end{aligned} \tag{4.51}$$

*whose maximum over the parameter set* $P = [0.35, 0.36] \times [0.05, 0.06]$*, obtained by solving six linear programs, is* $\mathbf{c}'_1 = -0.7942$.

*Repeating the procedure for all the directions of the template* $\Lambda$ *we obtain the vector of offsets* $\mathbf{c}' = (-0.7440, -0.9425, -0.0050, 0.8203, 0.9925, 0.0100)$ *that, associated with the template* $\Lambda$*, can be used to construct the over-approximating parallelotope* $X' = \langle \Lambda, \mathbf{c}' \rangle = \mathcal{P}_{con}(\Lambda, \mathbf{c}')$.

*Figure 4.8 shows the constructed parallelotope* $X'$ *(in white) with some reachable points (in black) computed with a sampling-based method. Note how* $X'$ *contains all the computed points.*

In this section we developed some methods to bound parametric polynomials over parallelotopes and compute parallelotopic over-approximations of set images. Parallelotopes allow us to obtain approximations that are finer than the ones provided by boxes. However, in the next section, we will further improve in precision, exploiting sets of parallelotopes and their intersections for the representation of sets of states reached by a dynamical system.

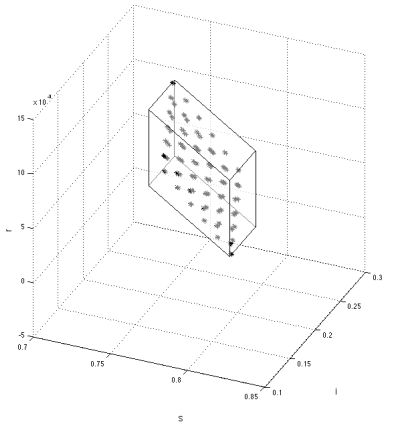



Figure 4.8: Parallelotope-based set image of SIR dynamical system. The constructed parallelotope $X'$ (in white) and some reachable points computed with a sampling-based method (in black).

## 4.5 Parallelotope Bundles

In this section we introduce *parallelotope bundles*, i.e., sets of parallelotopes whose intersections symbolically represents polytopes. Our definition and notation are inspired by [2].

**Definition 11** (Parallelotope Bundle)**.** *A* parallelotope bundle *is a finite set of parallelotopes* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *whose intersection, denoted by:*

$$\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap} = \bigcap_{i=1}^{b} \mathcal{P}_i, \tag{4.52}$$

*is the polytope generated by* $\langle D, \mathbf{c} \rangle$*, where* $D$ *and* $\mathbf{c}$ *are the union of the templates and offsets of* $\mathcal{P}_i$*, for* $i \in \{1, \ldots, b\}$*.*

Two parallelotope bundles $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ and $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\}$ are *equivalent* if they denote the same polytope. A bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ allows us to symbolically represent a polytope $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}$ without requiring the explicit computation of the intersection of the parallelotopes $\mathcal{P}_i$. Since, we are interested in bundles as symbolical representations of polytopes, we can always replace a bundle with an equivalent one, whenever this is convenient.

**Lemma 6** (Polytope Decomposition)**.** *Let* $Q$ *be a polytope. There exists a finite set of parallelotopes* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *such that* $Q = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}$*.*

*Proof.* Any set of parallelotopes $\mathcal{P}_1, \ldots, \mathcal{P}_b$ whose facets union is a cover of the facets of $Q$, generates a decomposing bundle such that $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap} = Q$. Let $\{h_1, \ldots, h_k\}$ be

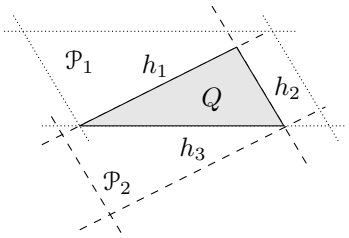



Figure 4.9: A polytope $Q$ and a decomposing bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$, i.e., $\{\mathcal{P}_1, \mathcal{P}_2\}^{\cap} = Q$.

the set of hyperplanes defining $Q$ and $\{h'_1, \ldots, h'_{k'}\} = \cup_{i=1}^{b}\{h_1^i, \ldots, h_{2n}^i\}$ be the union of all the constraints appearing in the bundle, for some $k' \in \mathbb{N}, k \leq k' \leq b2n$. The polytope generated by the bundle is then $\cap_{i=1}^{k'} h'_i$. Since $H$ is a cover of the constraints of $Q$, it holds that $\{h'_1, \ldots, h'_{k'}\} = \{h_1, \ldots, h_k\}$ and then $\cap_{i=1}^{k'} h'_i = \cap_{i=1}^{k} h_i = Q$. □

**Lemma 7** (Decomposition Cardinality)**.** *$\lceil m/n \rceil$ parallelotopes are sufficient to decompose a polytope $Q$ defined by $m$ constraints into a bundle.*

*Proof.* A single parallelotope can match at least $m$ constraints of $Q$. Then, the total number of sufficient parallelotopes to decompose $Q$ is the number of facets of $Q$ divided by the worst case maximum number of constrains matchable by a single parallelotope, i.e., $\lceil m/n \rceil$. □

**Example 16.** *Figure 4.9 shows a polytope $Q$ together with a possible decomposition, i.e., a bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$ such that $\{\mathcal{P}_1, \mathcal{P}_2\}^{\cap} = Q$. Here $m = 3$ and $n = 2$, so $\lceil 3/2 \rceil = 2$ parallelotopes are sufficient to decompose $Q$ (in our case $\mathcal{P}_1$ and $\mathcal{P}_2$).*

In Section 4.5.2 we will describe an algorithm for decomposing a polytope into a bundle in view of accurate image approximation.

In Section 4.1.2 we defined the enclosure of a compact set $S \subset \mathbb{R}^n$ with respect to a polytope $Q = \langle D, \mathbf{c} \rangle \subset \mathbb{R}^n$, that is the determination of a polytope $Q' = \langle \Lambda, \mathbf{c}' \rangle = \odot(S, Q)$ such that the constraints of the polytope $Q'$ are tangent to the set $S$. We now establish some properties of set enclosure that can be easily proved.

**Lemma 8.** *Let $S, S' \subset \mathbb{R}^n$ be compact sets with $S \subseteq S'$ and $Q = \langle D, \mathbf{c} \rangle$, $Q' = \langle D', \mathbf{c}' \rangle$ be two polytopes such that $D' \subseteq D$. It holds that:*

*(1) $S \subseteq \odot(S, Q)$;*

*(2) $\odot(S, Q) = \odot(S, \odot(S, Q)) = \odot(\odot(S, Q), Q)$;*

*(3) $\odot(S, Q) \subseteq \odot(S', Q)$;*

*(4) $\odot(S, Q) \subseteq \odot(S, Q')$.*

*Proof.* We now demonstrate each item of the Lemma.

(1) It follows directly from the definition of set enclosure (Definition 7);

(2) Let $Q_1 = \odot(S, Q)$, $Q_2 = \odot(S, \odot(S, Q))$, and $Q_3 = \odot(\odot(S, Q), Q)$. By definition, $Q_1 = \langle D, \mathbf{c}^1 \rangle$ is an enclosing polytope such that $\mathbf{c}_i^1 = \max_{\mathbf{x} \in S} D_i \mathbf{x}$, for $i \in \{1, \ldots, k\}$. $Q_2 = \odot(S, \odot(S, Q)) = \odot(S, Q_1)$, hence $Q_2$ has the same template $D$ of $Q_1$. Moreover, $Q_2 = \langle D, \mathbf{c}^2 \rangle$ is such that $\mathbf{c}_i^2 = \max_{\mathbf{x} \in S} D_i \mathbf{x} = \mathbf{c}_1^1$, for all $i \in \{1, \ldots, k\}$. Thus, $\mathbf{c}^2 = \mathbf{c}^1$ and then $Q_2 = \langle D, \mathbf{c}^2 \rangle = \langle D, \mathbf{c}^1 \rangle = Q_1$. Finally, $Q_3 = \odot(\odot(S, Q), Q) = \odot(Q_1, Q)$ has the same template $D$ of $Q$. Moreover, $Q_3 = \langle D, \mathbf{c}^3 \rangle$ is such that $\mathbf{c}_i^3 = \max_{\mathbf{x} \in Q_1 = \langle D, \mathbf{c}^1 \rangle} D_i \mathbf{x}$, for all $i \in \{1, \ldots, k\}$ that implies that $\mathbf{c}_i^3 = \mathbf{c}_i^1$ and then $Q_3 = \langle D, \mathbf{c}^3 \rangle = \langle D, \mathbf{c}^1 \rangle = Q_1$.

(3) Let $D \in \mathbb{R}^{k \times n}$ be a template. Since by hypothesis $S \subseteq S'$, it holds that $\mathbf{c}_i = \max_{\mathbf{x} \in S} D_i \mathbf{x} \leq \max_{\mathbf{x} \in S'} D_i \mathbf{x} = \mathbf{c}'_i$, for all $i \in \{1, \ldots, k\}$. Thus, $\odot(S, Q) = \langle D, \mathbf{c} \rangle \subseteq \langle D, \mathbf{c}' \rangle = \odot(S', Q)$.

(4) Let $\langle D', \mathbf{c}' \rangle = \odot(S, Q')$. By hypothesis $D' \subseteq D$ hence each directions $D'_i$ that appears also in $D$ as $D_j$ leads to an offsets $\mathbf{c}'_i = \max_{\mathbf{x} \in S} D'_i \mathbf{x} = \mathbf{c}_j$. This means that a point $\mathbf{p} \in \odot(S, Q) = \langle D, \mathbf{c} \rangle$ that satisfies the constraints $D\mathbf{p} \leq \mathbf{c}$, satisfies also the constraints $D'\mathbf{p} \leq \mathbf{c}'$ and then $\odot(S, Q) \subseteq \odot(S, Q')$.

□

Note that $S \subseteq \odot(S, Q)$. Moreover, if $Q = \langle D, \mathbf{c} \rangle$ and $Q' = \langle D', \mathbf{c}' \rangle$, and each row of $D'$ appears in $D$, then $\odot(S, Q) \subset \odot(S, Q')$, i.e., $Q$ provides a better over-approximation of $S$ then $Q'$.

The notion of set enclosure can be extended to bundles.

**Definition 12** (Set Bundle Enclosure)**.** *Let $S \subset \mathbb{R}^n$ be a compact set and $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ be a bundle. The enclosure of $S$ with respect to the bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ is defined as $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}) = \{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\}$, where $\mathcal{P}'_i = \odot(S, \mathcal{P}_i)$, for $i = 1, \ldots, b$.*

The set bundle enclosure shrinks the parallelotopes $\mathcal{P}_i$ around $S$, producing a bundle whose parallelotopes $\mathcal{P}'_i$ surround $S$. The two operators are related by the following equality.

**Lemma 9.** $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} = \odot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap})$.

*Proof.*

$$\boxdot(S, \{P_1, \ldots, P_b\})^{\cap} = \cap_{i=1}^{b} \odot (S, P_i) = \odot(S, \cap_{i=1}^{b} P_i) = \odot(S, \{P_1, \ldots, P_b\}^{\cap}) \quad (4.53)$$

□

The set bundle enclosure of $S$ with respect to a polytope $Q$ coincides with the decomposition of the polytope given by the enclosure $\odot(S, Q)$. As a consequence, we get that $\odot(\cdot, \cdot)$ and $\boxdot(\cdot, \cdot)$ are equivalent, with the difference that $\odot(\cdot, \cdot)$ returns a parallelotope $\mathcal{P}'$, while $\boxdot(\cdot, \cdot)$ returns the bundle $\{\mathcal{P}'\}$.

**Example 17.** *Figure 4.10 shows the set bundle enclosure of the polytope $Q$ with respect to the bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$ of Figure 4.9. The result of $\boxdot(S, \{\mathcal{P}_1, \mathcal{P}_2\})$ is the new bundle $\{\mathcal{P}'_1, \mathcal{P}'_2\}$.*

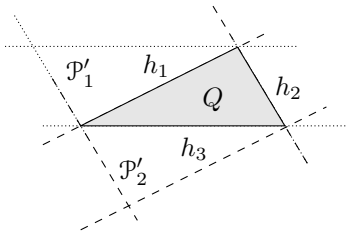



Figure 4.10: A polytope $Q$ and a its set bundle enclosure with respect to $\{\mathcal{P}_1, \mathcal{P}_2\}$ of Figure 4.9: $\boxdot(Q, \{\mathcal{P}_1, \mathcal{P}_2\}) = \{\mathcal{P}'_1, \mathcal{P}'_2\}$.

We say that $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\}$ is a *sub-bundle* of the bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ if $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\} \subseteq \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ and that two bundles are *strongly similar* if the set of normal vectors defining a parallelotope in one bundle is equal to the set of normal vectors defining a parallelotope in the other bundle. The following properties of the bundle enclosure operator immediately follow by definitions, Lemma 8, and Lemma 9.

**Lemma 10.** *Let $S, S' \subset \mathbb{R}^n$ be compact sets with $S \subseteq S'$, $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ be a bundle, $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\}$ be one of its sub-bundles, and $\{\mathcal{P}''_1, \ldots, \mathcal{P}''_b\}$ be a bundle strongly similar to $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$. It holds that:*

*(1)* $S \subseteq \boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap}$;

*(2)* $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}) = \boxdot(S, \boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})) = \boxdot\left(\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}\right)$;

*(3)* $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq \boxdot(S', \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap}$;

*(4)* $\boxdot(S, \{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\}) \subseteq \boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$ *and* $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq \boxdot(S, \{\mathcal{P}'_1, \ldots, \mathcal{P}'_{b'}\})^{\cap}$;

*(5)* $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$ *is strongly similar to* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$;

*(6)* $\boxdot(S, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}) = \boxdot(S, \{\mathcal{P}''_1, \ldots, \mathcal{P}''_b\})$.

*Proof.* We now demonstrate each item of the Lemma.

(1) By Lemma 8 item (1) it holds that $S \subseteq \odot(S, P_i) \subseteq \cap_{i=1}^{b} \odot(S, P_i) = \boxdot(S, \{P_1, \ldots, P_b\})^{\cap}$.

(2) By Lemma 8 item (2) we know that for each $P_i$, with $i \in \{1, \ldots, b\}$ it holds that $\odot(S, P_1) = \odot(S, \odot(S, P_1)) = \odot(\odot(S, P_1), P_1)$, from which it follows the thesis.

(3) It follows directly from Lemma 8 item (3).

(4) Similar to the previous cases but following from the item (4) of Lemma 8.

(5) By definition of set enclosure each parallelotope $\odot(S, P_i)$ of $\boxdot(S, \{P_1, \ldots, P_b\})$ has the same template of $P_i$, for $i \in \{1, \ldots, b\}$.

(6) By hypothesis $P_i$ and $P''_i$ have the same templates, for $i \in \{1, \ldots, b\}$. Thus $\odot(S, P_i) = \odot(S, P''_i)$ from which it follows the thesis.

□

A bundle representing a polytope may not be "minimal" in the sense that one or more paralleloptopes can be shrunk while the resulting bundle still represents the same polytope. The shrinking process removes subsets of parallelotopes that are not in the polytope. Such shrinking is thus useful for many operations, in particular image over-approximation. As we will see later, the shrinking reduces the error when the image over-approximation is performed on shrunk parallelotopes. We thus introduce the notion of *canonical form* for a bundle as follows.

**Definition 13** (Bundle Canonical Form)**.** *A bundle* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *is in* canonical form *if and only if:*

$$\boxdot(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}) = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}.$$

Intuitively, a bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ is in canonical form if the enclosure of its symbolic polytope $\mathcal{P}^{\cap} = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}$ with respect to $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ does not affect the parallelotopes $\mathcal{P}_i$, for $i \in \{1, \ldots, b\}$. The canonical form of a bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ can be obtained by enclosing its polytope $\mathcal{P}^{\cap}$ with respect to its parallelotopes $\mathcal{P}_i$. The bundle $\{\mathcal{P}'_1, \mathcal{P}'_2\}$ of Figure 4.10 is in canonical form, since it is the result of the bundle enclosure $\boxdot(Q, \{\mathcal{P}_1, \mathcal{P}_2\}) = \{\mathcal{P}'_1, \mathcal{P}'_2\}$ where $Q = \{\mathcal{P}_1, \mathcal{P}_2\}^{\cap}$. In virtue of Lemma 10 item *(2)*, the result of a bundle enclosure is always in canonical form. In other terms, the operator $\boxdot(\cdot, \cdot)$ can be exploited for canonizing bundles, as stated by the following result.

**Lemma 11** (Canonization)**.** *Let* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *be a bundle. The bundle:*

$$\boxdot(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$$

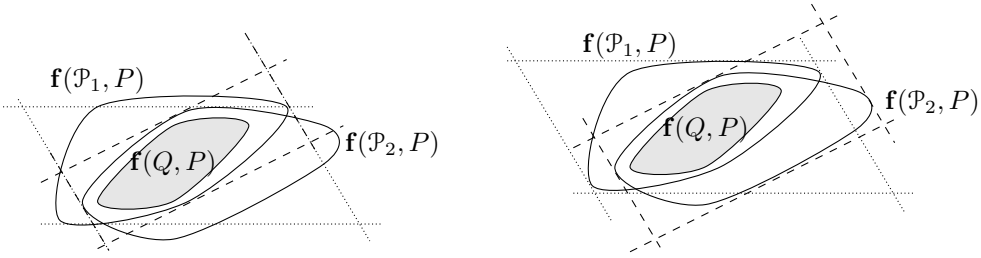
*is in canonical form and it is equivalent to* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$*.*

*Proof.* It follows directly by the definitions of set bundle enclosure and bundle in canonical form. □

Intuitively, a bundle in canonical form is a "minimal" representation of the polytope, with respect to a given set of parallelotope directions, since all the offsets are shifted towards the constraints of the polytope. The advantage of dealing with bundles in canonical form will become clearer on images approximation.

In the following we show the advantage of bundles in image approximation. We start by proving some inclusions that hold on the images of bundles by a continuous function. Note that these properties hold for all continuous functions, and in the case of polynomials, they are particularly useful for our image approximation problem, because we can indeed effectively enclose the image of a parallelotope.

**Lemma 12** (Bundle Image)**.** *Let* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *be a bundle with* $\mathcal{P}^{\cap} = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^{\cap}$*,* $P \subset \mathbb{R}^m$ *be a compact parameter set, and* $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ *be a continuous function.*




(a) $\cap_{i=1}^{2} \boxdot (\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \mathcal{P}_2\})^{\cap}$. (b) $\odot(\mathbf{f}(\mathcal{P}_1, P), \mathcal{P}_1) \cap \odot(\mathbf{f}(\mathcal{P}_2, P), \mathcal{P}_2)$.

Figure 4.11: Different bundle image techniques.

*The following inclusions hold:*

$$\mathbf{f}(\mathcal{P}^{\cap}, P) \subseteq \boxdot(\mathbf{f}(\mathcal{P}^{\cap}, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq \quad (12.1)$$

$$\subseteq \bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq \quad (12.2)$$

$$\subseteq \bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_i\})^{\cap} \quad (12.3)$$

*Proof.* Since $\mathbf{f}$ is continuous and $\mathcal{P}^{\cap}$ and $P$ are compact, we have that $\mathbf{f}(\mathcal{P}^{\cap}, P)$ is compact.

$\mathbf{f}(\mathcal{P}^{\cap}, P) \subseteq$ By Lemma 10 item *(1)*

$\boxdot(\mathbf{f}(\mathcal{P}^{\cap}, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq$ By Lemma 10 item *(3)*

$\boxdot(\bigcap_{i=1}^{b} \mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq$ By Lemma 10 item *(3)*

$\bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^{\cap} \subseteq$ By Lemma 10 item *(4)*

$\bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_i\})^{\cap}$

□

**Example 18.** *Figures 4.11 shows two bundle images. Figure 4.11a shows the intersection of the enclosures of* $\mathbf{f}(\mathcal{P}_1, P)$ *and* $\mathbf{f}(\mathcal{P}_2, P)$ *with respect to* $\{\mathcal{P}_1, \mathcal{P}_2\}$*; Figure 4.11b shows the enclosure of* $\mathbf{f}(\mathcal{P}_1, P)$ *with respect to* $\mathcal{P}_1$ *intersected with the enclosure of* $\mathbf{f}(\mathcal{P}_2, P)$ *with respect to* $\mathcal{P}_2$*. Note how the over-approximation of the first method is tighter than the second one.*

We observe that the polytope defined by item *(7.3)* coincides with the polytope symbolically represented by the bundle $\{\boxdot(\mathbf{f}(\mathcal{P}_1, P), \{\mathcal{P}_1\})^{\cap}, \ldots, \boxdot(\mathbf{f}(\mathcal{P}_b, P), \{\mathcal{P}_b\})^{\cap}\}$. Intuitively Lemma 12 suggests us two possible ways of approximating the image of a polytope. In both cases we first have to symbolically represent the polytope thought a

bundle, then either we consider the images of all the parallelotopes and we enclose each image in a new bundle or we consider the bundle obtained by enclosing each parallelotope image with respect to its original directions. In both cases it is only necessary to be able to compute images of parallelotopes.

Notice that the bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ could be not in canonical form, while the bundle $\boxdot(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^\cap, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$ is. The following result shows that if we approximate the image of $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^\cap$ exploiting the above lemma on the canonical bundle $\boxdot(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^\cap, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$ we get tighter approximations with respect to that obtained using the initial bundle.

**Theorem 2** (Canonical Bundle Image)**.** *Let us consider a bundle* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$*, with* $\mathcal{P}^\cap = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^\cap$*, a compact parameter set* $P \subset \mathbb{R}^m$*, and a function* $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$*. Let also:*

$$\{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\} = \boxdot(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}^\cap, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}).$$

*The following relations hold among the over-approximations of* $\mathbf{f}(\mathcal{P}^\cap, P)$*:*

$$\boxdot(\mathbf{f}(\mathcal{P}^\cap, P), \{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\})^\cap = \boxdot(\mathbf{f}(\mathcal{P}^\cap, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^\cap \subseteq$$

$$\bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}'_i, P), \{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\})^\cap \subseteq \bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})^\cap$$

$$\cap \qquad\qquad\qquad\qquad \cap$$

$$\bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}'_i, P), \{\mathcal{P}'_i\})^\cap \subseteq \bigcap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_i\})^\cap$$

*Proof.* The thesis is a consequence of Lemma 9 together with Lemma 12 and Lemma 10. □

As a consequence, having to compute the image of a generic compact set $S$, one can first over approximate $S$ through the bundle enclosure operator, which returns a bundle in canonical form, and then exploit the above results to over-approximate the image of $S$.

### 4.5.1 Bundle Data Structure

In this section we will provide more details on compact representation and basic transformations of canonical bundles.

A parallelotope bundle in canonical form can be compactly represented through the tuple $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$ where:

- $L \in \mathbb{R}^{k \times n}$ is the *directions matrix* that contains the directions used to build the parallelotopes. The $i$-th row $L_i$ of $L$ represents a direction;
- $\overline{\mathbf{d}} \in \mathbb{R}^k$ is the *upper offsets vector*. The $i$-th element of $\overline{\mathbf{d}}$, associated with the $i$-th direction $L_i$, constitutes the half-space $L_i\mathbf{x} \leq \overline{\mathbf{d}}_i$;
- $\underline{\mathbf{d}} \in \mathbb{R}^k$ is the *lower offsets vector*. The $i$-th element of $\underline{\mathbf{d}}$, associated with the $i$-th direction $L_i$, constitutes the half-space $-L_i\mathbf{x} \leq \underline{\mathbf{d}}_i$;

- $T \in \{1, \ldots, k\}^{b \times n}$ is the *templates* matrix that represents the set of the parallelotope templates. Each element in $T$ is a reference to a direction in $L$ and offsets in $\overline{\mathbf{d}}$ and $\underline{\mathbf{d}}$. A row in $T$ constitutes a set of half-spaces that generates a parallelotopes.

**Example 19.** *Consider for instance the bundle* $\{\mathcal{P}'_1, \mathcal{P}'_2\}$ *in canonical form of Figure 4.10 where* $\mathcal{P}'_1 = \langle \Lambda'_1, \mathbf{c}'_1 \rangle$ *and* $\mathcal{P}'_2 = \langle \Lambda'_2, \mathbf{c}'_2 \rangle$ *with:*

$$\Lambda'_1 = \begin{pmatrix} 1.6 & 1 \\ 0 & 1 \\ -1.6 & -1 \\ 0 & -1 \end{pmatrix} \mathbf{c}'_1 = \begin{pmatrix} 10 \\ 3.1 \\ -1 \\ -1 \end{pmatrix} \Lambda'_2 = \begin{pmatrix} 1.6 & 1 \\ -0.5 & 1 \\ -1.6 & -1 \\ 0.5 & -1 \end{pmatrix} \mathbf{c}'_2 = \begin{pmatrix} 10 \\ 1 \\ -1 \\ 1.7 \end{pmatrix}$$

*This bundle can be represented by the tuple* $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$ *where:*

$$L = \begin{pmatrix} 1.6 & 1 \\ 0 & 1 \\ -0.5 & 1 \end{pmatrix} \overline{\mathbf{d}} = \begin{pmatrix} 10 \\ 3.1 \\ 1 \end{pmatrix} \underline{\mathbf{d}} = \begin{pmatrix} -1 \\ -1 \\ 1.7 \end{pmatrix} T = \begin{pmatrix} 1 & 2 \\ 1 & 3 \end{pmatrix}.$$

With this representation we avoid the storage of redundant directions shared by different parallelotopes. Doing so, a single operation on an entry in the tuple, indirectly affects several parallelotopes in the bundle. Moreover, for each parallelotope we store only $n$ directions against $2n$ constraints, since we know that parallel constraints can be obtained by reversing the signs of the normal vectors. Note that each direction $L_i$ is associated with a unique upper and lower offset $\overline{\mathbf{d}}_i$ and $\underline{\mathbf{d}}_i$. This means that if two parallelotopes share a direction, the constraints defined by such direction coincide in the two parallelotopes. Hence, this data structure does not allow us to represent all the possible bundles (for instance the one shown in Figure 4.9), but it is expressive enough to capture all the canonical bundles (like the one of Figure 4.10).

We now show how the operations presented in Section 4.5 can be defined on our data structure $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$. We begin with the decomposition of a polytope (see Definition 11).

**Method 1** (Polytope Decomposition)**.** *Let* $Q \subset \mathbb{R}^n$ *be a polytope defined by* $m$ *constraints. Let* $L \in \mathbb{R}^{k \times n}$ *be a matrix containing all the normal versors of* $Q$ *without repetitions, i.e., the elements of* $L$ *are pairwise linearly independent. To generate the* $i$*-th decomposing parallelotope, it is sufficient to pick* $n$ *directions* $L_{j_1}, \ldots, L_{j_n}$ *from* $L$ *and store their indices in the template matrix* $T_i = (j_1, \ldots, j_n)$*. By Lemmas 6 and 7, we have to generate at most* $\lceil m/n \rceil$ *parallelotopes such that the union of the picked directions is a cover of the constraints of* $Q$*. Finally, the offset vectors* $\overline{\mathbf{d}}, \underline{\mathbf{d}} \in \mathbb{R}^k$ *can be obtained by enclosing* $Q$ *with respect to the constructed parallelotopes as described in Method 2.*

We now show how to compute the set bundle enclosure.

**Method 2** (Set Bundle Enclosure $\boxdot(\cdot, \cdot)$)**.** *The enclosure of a bounded set* $S \subset \mathbb{R}^n$ *with respect to a canonical bundle* $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ *stored as* $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$ *can be obtained by updating the upper and lower offset vector as* $\overline{\mathbf{d}}_i = \max_{\mathbf{x} \in S} L_i \mathbf{x}$ *and* $\underline{\mathbf{d}}_i = \max_{\mathbf{x} \in S} -L_i \mathbf{x}$*, for* $i = 1, \ldots, k$.

The described methods work only on canonical bundles and return a compact representation of a canonical bundle. The enclosure of a polytope with respect to a bundle

requires the resolution of $2k$ linear programs. Thus, the canonization of a bundle can be done by solving a series of linear programs where only the offsets of the constraints that do not participate to the intersection are modified.

The transformation of a bundle through a continuous function can be rather difficult, depending on the transforming function. If the function is linear, it is possible to exactly compute the image of each parallelotope and then obtain the exact bundle transformation. Things are more complex when the function is nonlinear and the geometric properties of the parallelotopes are not preserved. We will now describe two methods, based on Theorem 2, that only require to be able to compute the image of a parallelotope, besides being able to implement Method 2. As stated by Lemma 12 item *(3)*, an over-approximation of a bundle transformation $\mathbf{f}(\mathcal{P}^{\cap}, P)$, with $\mathcal{P}^{\cap} = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$, can be obtained by enclosing each image $\mathbf{f}(\mathcal{P}_i, P)$ with $\mathcal{P}'_i = \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_i\})^{\cap}$ and then considering the intersection $\cap_{i=1}^{b} \mathcal{P}'_i$ (see (12.3)). We call such approximation *one-for-one (OFO)*, since each parallelotope in the bundle is independently approximated.

**Method 3** (One-for-One Image (OFO))**.** *The* one-for-one approximation *of the bundle* $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$ *can be obtained by retrieving each parallelotope* $\mathcal{P}_i$*, computing the enclosures* $\mathcal{P}'_i = \odot(\mathbf{f}(\mathcal{P}_i, P), \mathcal{P}_i)$*, and then computing the canonization of* $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\}^{\cap} = \mathcal{P}'^{\cap}$*, that is* $\boxdot(\mathcal{P}'^{\cap}, \{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\})$.

The polytope provided by the OFO method corresponds to the set $\cap_{i=1}^{b} \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_i\})^{\cap}$ of Theorem 2.

In order to obtain a finer over-approximation, it is possible to change the template in the approximation process, i.e., we can fix a new template to enclose $\mathbf{f}(\mathcal{P}_i, P)$. As suggested by Lemma 12 item *(2)*, we can exploit all the directions of the bundle, i.e., instead of looking for a new template for each parallelotope, we can bound each set $\mathbf{f}(\mathcal{P}_i, P)$ with all the directions of the starting bundle. We call such approximation *all-for-one (AFO)* since all the directions of the bundle are used to approximate the image of a single parallelotope.

**Method 4** (All-for-One Image (AFO))**.** *The all-for-one approximation of the bundle* $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$ *can be obtained by retrieving each parallelotope* $\mathcal{P}_i$*, computing the set bundle enclosure* $\{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\} = \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$*, and then decomposing the polytope* $\cap_{i=1}^{b} \{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\}^{\cap}$*, i.e., computing* $\boxdot(\cap_{i=1}^{b} \{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\}^{\cap}, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$.

The AFO transformation produces a bundle whose symbolic polytope corresponds to the polytope of the left-hand side term of the last line in Theorem 2. By Theorem 2, the AFO approximation is finer than the one produced by the OFO method. Obviously the precision has a cost: the OFO method requires $b(2n) + k$ optimizations against the $b(2k) + k$ optimizations of the AFO approach (recall that $k \geq n$).

Both the approximation methods are based on a series of enclosures. The offsets of the constraints necessary to obtain the enclosures, can be attained by solving optimization problems of the form $\overline{\mathbf{d}}_j = \max_{\mathbf{x} \in \mathbf{f}(\mathcal{P}_i, P)} L_j \mathbf{x}$. If the transformation function $\mathbf{f}$ is nonlinear, these optimization problems might be computationally expensive. However, in the next section we expose a method, based on the Bernstein coefficients, to efficiently deal with images of polynomial functions.

### 4.5.2 Bundle-Based Set Image

In this section we define the image computation of a parallelotope bundle with respect to a parametric polynomial.

We define an algorithm (Algorithm 6) based on parallelotope bundles and their operations that computes the states reached by a single step of a dynamical system. For brevity, the bundle $\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$ is abbreviated by $X$.

**Algorithm 6** Bundle-based reachable set integration

```
1: function REACHSTEP(X, P)                    ▷ X bundle, P ⊆ ℝ^m polytope
2:     X' ←TRANSFORM(f, X, P)
3:     X' ←DECOMPOSE(X')                       ▷ Optional
4:     return X'
5: end function
```

The algorithm receives in input a bundle $X$ and a parameter set $P$ and over-approximates the set of states reachable through the transformation of the bundle $X$ with respect to the dynamics $\mathbf{f}$ and parameter set $P$. The transformation performed by the function TRANSFORM (Line 2) can be either the OFO (see Method 3) or the AFO (see Method 4). In both cases, the transformation produces a bundle $X'$ in canonical form that over-approximates the states reachable by the dynamical system from $X$. Finally, the symbolic polytope of the computed bundle $X'$ can be decomposed (Line 3), obtaining a new bundle whose parallelotopes combine the directions differently from $X$. The decomposition is optional, but it might improve the precision in the over-approximation of the future transformations, since the over-approximating parallelotopes might be smaller than the ones produced by the transformation. In the following we will discuss in detail the functions TRANSFORM and DECOMPOSE. For polynomial dynamical systems we begin with the transformation, since the decomposition is strictly related to the way we transform the bundles.

#### Transformation

The transformation of a bundle is strictly related to the transformation of a single parallelotope $X = \langle \Lambda, \mathbf{c} \rangle$ and in particular to the resolution of optimization problems of the form:

$$\mathbf{c}'_i = \max_{\mathbf{x} \in X, \mathbf{p} \in P} \Lambda_i \mathbf{f}(\mathbf{x}, \mathbf{p}).$$

In Section 4.2.3 we defined the function BOUND (Algorithm 4) that can be used to find an upper bound $\overline{b} \in \mathbb{R}$ of the polynomial $\Lambda_i \mathbf{f}(\mathbf{x}, \mathbf{p})$ over the parallelotope $X$ and the polytopic parameter set $P$ such that:

$$\overline{b} \geq \max_{\mathbf{x} \in X, \mathbf{p} \in P} \pi(\mathbf{x}, \mathbf{p}).$$

We briefly recall the procedure: given a parallelotope $X = \langle \Lambda, \mathbf{c} \rangle$ and a parameter set $P$, the function BOUND calls the procedure $\mathbf{c}'_i = \text{MAXBERNCOEFF}(\mathbf{h}(\mathbf{x}, \mathbf{p}), P)$, where $\mathbf{h}(\mathbf{x}, \mathbf{p}) = \Lambda_i \mathbf{f}(\gamma_{(\mathbf{q},\boldsymbol{\beta})}(\mathbf{x}), \mathbf{p})$ and $\gamma_{(\mathbf{q},\boldsymbol{\beta})}(\mathbf{x})$ is the generator function of the parallelotope $X = \langle \Lambda, \mathbf{c} \rangle$ computed by the procedure CON2GEN($X$).

We now see how the function BOUND can be used to define our bundle transformation methods.

The OFO transformation of a bundle $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T\rangle$ with parameter set $P \subset \mathbb{R}^n$, as exposed in Method 3, can be obtained by retrieving each parallelotope $\mathcal{P}_i = \langle \Lambda, \mathbf{c}\rangle$, for $i = 1, \ldots, b$, computing the new offsets $\mathbf{c}'_j = \text{BOUND}(\Lambda_j \mathbf{f}(\mathbf{x}, \mathbf{p}), \mathcal{P}_i, P)$, for $j = 1, \ldots, 2n$, and defining the over-approximating parallelotope $\mathcal{P}'_i = \langle \Lambda, \mathbf{c}'\rangle \supseteq \mathbf{f}(\mathcal{P}_i, P)$. Finally, the canonization of the transformed bundle $\{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\}$ can be obtained by solving a family of linear programs of the form $\max_{\mathbf{x} \in \mathcal{P}'^{\cap}} \Lambda_i \mathbf{x}$, where $\Lambda_i$ belongs to the template matrices of $\mathcal{P}'_j$ and $\mathcal{P}'^{\cap} = \{\mathcal{P}'_1, \ldots, \mathcal{P}'_b\}^{\cap}$ is the polytope of the computed bundle.

The AFO transformation of a bundle $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T\rangle$ with parameter set $P$, as defined in Method 4, can be done as follows. For each parallelotope of the bundle $\mathcal{P}_i$, for $i = 1, \ldots, b$, we have to compute the enclosure $\{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\} = \boxdot(\mathbf{f}(\mathcal{P}_i, P), \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$. An over-approximation of $\mathcal{P}'_{i_m}$ is the parallelotope $\langle \Lambda, \mathbf{c}'\rangle$ where $\Lambda$ is the template of $\mathcal{P}_{i_m}$ and $\mathbf{c}'_j = \text{BOUND}(\Lambda_j \mathbf{f}(\mathbf{x}, \mathbf{p}), \mathcal{P}_{i_m}, P)$, for all $j = 1, \ldots, 2n$. Finally, the canonization enclosure $\boxdot(\cap_{i=1}^{b}\{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\}^{\cap}, \{\mathcal{P}_1, \ldots, \mathcal{P}_b\})$ can be computed by solving a group of linear programs of the form $\max_{\mathbf{x} \in \mathcal{P}'^{\cap}} \Lambda_j \mathbf{x}$, where $\Lambda_j$ belongs to the template matrices of $\mathcal{P}'_{i_m}$ and $\mathcal{P}'^{\cap} = \cap_{i=1}^{b}\{\mathcal{P}'_{i1}, \ldots, \mathcal{P}'_{ib}\}^{\cap}$ is the polytope obtained by the intersection of the polytopes of the computed bundles.

### Decomposition

Since in our reachability algorithm (see Algorithm 6) we are interested in decomposing a polytope described by a bundle, we define a function DECOMPOSE that receives in input a bundle in canonical form $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T\rangle$ (whose polytope $\mathcal{P}^{\cap}$ has to be decomposed) and reorganizes the templates matrix $T$ creating a new collection of parallelotopes around the polytope $\mathcal{P}^{\cap}$. The goal of the decomposition is to create a set of small parallelotopes whose intersection is $\mathcal{P}^{\cap}$. There are two reasons why we want small parallelotopes:

1. Smaller parallelotopes $\mathcal{P}_i$ lead to a smaller bundle image $\{\mathbf{f}(\mathcal{P}_1, P), \ldots, \mathbf{f}(\mathcal{P}_d, P)\}$ and then to a more accurate over-approximation $\mathbf{f}(\mathcal{P}^{\cap}, P)$ (see, e.g., Theorem 2);

2. The shorter the largest side length of $\mathcal{P}_i$, the more accurate the over-approximation introduced by the Bernstein coefficients (see Lemma 3).

The aspects to take into account in the construction of the parallelotopes are: the volume and the maximum side length. Moreover, we do not have to forget that the set of the parallelotope directions must cover the directions of the decomposed symbolic polytope (see Definition 11). Finding the best decomposition in terms of volume and maximum length minimization is computationally expensive and might not be possible (recall that the set cover problem is NP-hard).

In order to efficiently find a good decomposition, we propose a heuristic technique that constructs the parallelotopes while trying to minimize the volumes and maximum side lengths. The proposed heuristic starts from a decomposition, applies a series of random changes to the templates matrix, and keeps only the best one accordingly to an evaluation function that we will soon define. The procedure is repeated until a fixed number of iterations is reached.

Given a bundle $\mathcal{P}^{\cap} = \{\mathcal{P}_1, \ldots, \mathcal{P}_b\}$, the evaluation function should take into account the volumes and side lengths of the parallelotopes $\mathcal{P}_i$, for $i \in \{1, \ldots, b\}$. The exact computation of the volume of a parallelotope is rather expensive, since it is equal to the determinant of a $n \times n$ matrix. To lighten the computation, we approximate the volume of $\mathcal{P} = \langle \Lambda, \mathbf{c} \rangle$ with the product of the distances of its constraints:

$$\tilde{v}(\mathcal{P}) = \prod_{i=1}^{n} \delta(\Lambda_i \mathbf{x} \leq \mathbf{d}_i, \Lambda_{i+n} \mathbf{x} \leq \mathbf{d}_{i+n}) \tag{4.54}$$

where $\delta(\Lambda_i \mathbf{x} \leq \mathbf{d}_i, \Lambda_{i+n} \mathbf{x} \leq \mathbf{d}_{i+n}) = |\mathbf{d}_i - \mathbf{d}_{i+n}| / \|\Lambda_i\|$ and $\|\cdot\|$ is the Euclidean norm.

The computation of the side lengths of a parallelotope passes inevitably through the determination of its vertices, an operation that can be computationally expensive. Instead of calculating the exact lengths, we opt for a faster heuristic that guesses the lengths of a parallelotope from the angles of the directions of its constraints. Intuitively, in the two-dimensional case, having fixed two parallel lines, the lengths of the edges not lying on the two fixed lines are minimal when the added directions and the fixed ones are orthogonal. Thus, we define the notion of *orthogonal proximity* $\theta(\Lambda_i, \Lambda_j) = \widehat{\Lambda_i, \Lambda_j} \pmod{\pi/2}$, where $\widehat{\Lambda_i, \Lambda_j}$ is the angle between $\Lambda_i$ and $\Lambda_j$, i.e., $\widehat{\Lambda_i, \Lambda_j} = \arccos \dfrac{\Lambda_i \Lambda_j}{\|\Lambda_i\| \|\Lambda_j\|}$. The orthogonal proximity of a parallelotope $\mathcal{P} = \langle \Lambda, \mathbf{c} \rangle$ is defined as

$$\Theta(\mathcal{P}) = \max_{i,j \in \{1,\ldots,2n\}} \theta(\Lambda_i, \Lambda_j). \tag{4.55}$$

Exploiting the notions of approximated volume $\tilde{v}$ and orthogonal proximity $\Theta$, we define the evaluation function $w$ for a bundle as:

$$w(\{\mathcal{P}_1, \ldots, \mathcal{P}_b\}) = \max_{i \in \{1,\ldots,b\}} \alpha \tilde{v}(\mathcal{P}_i) + (1 - \alpha) \Theta(\mathcal{P}_i) \tag{4.56}$$

with $\alpha \in [0, 1]$ tunable parameter.

**Example 20.** *Let us consider the dynamics of the discrete-time SIR epidemic model of Example 6 with state variables and parameters grouped in the vectors* $\mathbf{x} = (s, i, r)$ *and* $\mathbf{p} = (\beta, \gamma)$*:*

$$\mathbf{f}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f}_s(\mathbf{x}, \mathbf{p}) = s - \beta s i \\ \mathbf{f}_i(\mathbf{x}, \mathbf{p}) = i + \beta s i - \gamma i \\ \mathbf{f}_r(\mathbf{x}, \mathbf{p}) = r + \gamma i \end{pmatrix}. \tag{4.57}$$

We also consider the box and parallelotope of Examples 14 and 15, that here we denote with $\mathcal{P}_1$ and $\mathcal{P}_2$, respectively. Grouping $\mathcal{P}_1$ and $\mathcal{P}_2$, we define the bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$, whose intersection $\{\mathcal{P}_1, \mathcal{P}_2\}^{\cap}$ leads to the polytope depicted in Figure 4.12 (in gray). It is not difficult to see that $\{\mathcal{P}_1, \mathcal{P}_2\}^{\cap}$ is already in canonical form, hence we can exploit our data structure to represent it.

The bundle $\{\mathcal{P}_1, \mathcal{P}_2\}^{\cap}$ can be compactly represented with the tuple $\langle L, \overline{\mathbf{d}}, \underline{\mathbf{d}}, T \rangle$,

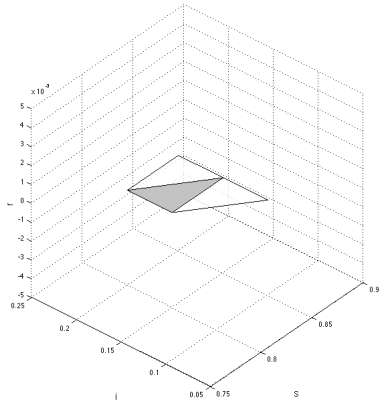


Figure 4.12: Parallelotope bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$. Parallelotopes $\mathcal{P}_1$ and $\mathcal{P}_2$ (in white) and polytope $\{\mathcal{P}_1, \mathcal{P}_2\}^\cap$ (in gray).

where:

$$L = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \\ 1 & 1 & 0 \end{pmatrix} \quad \overline{\mathbf{d}} = \begin{pmatrix} 0.85 \\ 0.20 \\ 0.00 \\ 1.00 \end{pmatrix} \quad \underline{\mathbf{d}} = \begin{pmatrix} -0.80 \\ -0.15 \\ -0.00 \\ -0.95 \end{pmatrix} \quad T = \begin{pmatrix} 1 & 2 & 3 \\ 1 & 3 & 4 \end{pmatrix}. \tag{4.58}$$

Let us now analyze the transformation of the bundle with respect to the dynamics of the discrete-time SIR epidemic model under the influence of the parameter set $P = [0.35, 0.36] \times [0.05, 0.06]$ where $\beta \in [0.35, 0.36]$ and $\gamma \in [0.05, 0.06]$.

We begin with the independent transformation of parallelotopes that compose the bundle. The images of the parallelotopes can be over-approximated by the enclosures $\mathcal{P}'_1 = \odot(\mathbf{f}(\mathcal{P}_1, P), \mathcal{P}_1)$ and $\mathcal{P}'_2 = \odot(\mathbf{f}(\mathcal{P}_2, P), \mathcal{P}_2)$, that we already encountered in Examples 14 and 15. The enclosing parallelotopes $\mathcal{P}'_1$ and $\mathcal{P}'_2$ can be used to define the bundle $\{\mathcal{P}'_1, \mathcal{P}'_2\}$ represented in Figure 4.13. The gray area represents the polytope $\{\mathcal{P}'_1, \mathcal{P}'_2\}^\cap$ that over-approximates the image $\mathbf{f}(\{\mathcal{P}_1, \mathcal{P}_2\}^\cap, P)$. In black there are some reachable points compute with a sampling-based technique. Note how the gray area contains all the computed reachable points. From the figure we can also note that the bundle is not in canonical form, since, for instance, the rightmost facets on the $s$-axis are not aligned.

To canonize the computed bundle and obtain its OFO transformation, we can push the directions of $L$ towards the polytope $\{\mathcal{P}'_1, \mathcal{P}'_2\}^\cap$ solving a family of linear programs. The canonization leads to the bundle that symbolically represents the polytope (depicted in Figure 4.14a) having offsets:

$$\overline{\mathbf{d}} = \begin{pmatrix} 0.8054 \\ 0.2501 \\ 0.0120 \\ 0.9925 \end{pmatrix} \quad \underline{\mathbf{d}} = \begin{pmatrix} -0.7424 \\ -0.1830 \\ -0.0075 \\ -0.9410 \end{pmatrix} \tag{4.59}$$

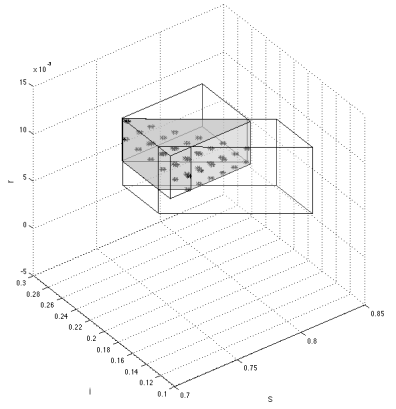


Figure 4.13: Parallelotope bundle $\{\mathcal{P}'_1, \mathcal{P}'_2\}$ (not in canonical form) obtained by the enclosures $\mathcal{P}'_i = \odot(\mathbf{f}(\mathcal{P}_i, P), \mathcal{P}_i)$, with $i \in \{1, 2\}$. The constructed parallelotopes $\mathcal{P}'_1, \mathcal{P}'_2$ (in white), the symbolic polytope $\{\mathcal{P}'_1, \mathcal{P}'_2\}^{\cap}$ (in gray), and some reachable points computed with a sampling-based method (in black).

The AFO transformation can be obtained by maximizing each direction of $L$ over the images $\mathbf{f}(\mathcal{P}_1, P)$ and $\mathbf{f}(\mathcal{P}_2, P)$ and then keeping the tightest bounds. These maximums can be over-approximated using the Bernstein technique. The offsets computed on the image $\mathbf{f}(\mathcal{P}_1, P)$ are:

$$\overline{\mathbf{d}} = \begin{pmatrix} 0.8250 \\ 0.2527 \\ 0.0120 \\ 1.0601 \end{pmatrix} \quad \underline{\mathbf{d}} = \begin{pmatrix} -0.7424 \\ -0.1830 \\ -0.0075 \\ -0.9410 \end{pmatrix} \tag{4.60}$$

while the offsets obtained on the image $\mathbf{f}(\mathcal{P}_2, P)$ are:

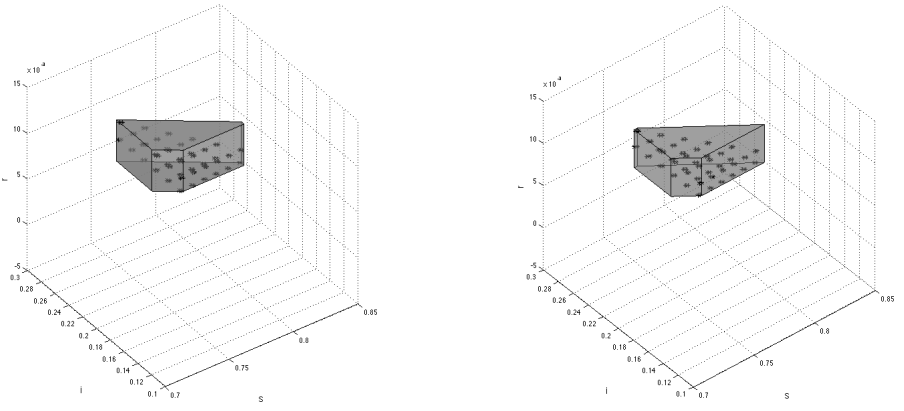
$$\overline{\mathbf{d}} = \begin{pmatrix} 0.8202 \\ 0.2476 \\ 0.0120 \\ 0.9925 \end{pmatrix} \quad \underline{\mathbf{d}} = \begin{pmatrix} -0.7424 \\ -0.1238 \\ -0.0050 \\ -0.9410 \end{pmatrix}. \tag{4.61}$$

Taking the minimums between the computed bounds we obtain the offsets:

$$\overline{\mathbf{d}} = \begin{pmatrix} 0.8202 \\ 0.2476 \\ 0.0120 \\ 0.9925 \end{pmatrix} \quad \underline{\mathbf{d}} = \begin{pmatrix} -0.7424 \\ -0.1238 \\ -0.0050 \\ -0.9410 \end{pmatrix} \tag{4.62}$$

that lead to the AFO transformation of the bundle $\{\mathcal{P}_1, \mathcal{P}_2\}$ whose symbolic polytope is represented in Figure 4.14b.

The AFO transformation generates an over-approximation that is finer than the



(a) OFO transformation. (b) AFO transformation.

Figure 4.14: Parallelotope bundle transformations. The symbolic polytopes generated by the transformations (in gray) and some reachable points computed with a sampling-based method (in black).

OFO one (the offsets coefficients $\overline{\mathbf{d}}_1$ and $\underline{\mathbf{d}}_1$ are tighter in the AFO case).

## 4.6 Bernstein Coefficients Computation

All the algorithms exposed in the previous sections share at their cores Bernstein coefficients. The computational complexity and precision of our algorithms are strictly related to the computation and bound precision of the coefficients. In the spirit of improving both performance and accuracy, we ask ourselves two questions:

1. Can we speed-up the computation of the coefficients?

2. Can we improve the precision of the bounds provided by the coefficients?

In the next sections we will try to answer to these questions defining a method to accelerate the computation of Bernstein coefficients (Section 4.6.1), a technique that symbolically manipulates them and avoids redundant computations (Section 4.6.2), and a method to improve the provided bounds (Section 4.6.3).

### 4.6.1 Improving Efficiency

The direct computation of Bernstein coefficients from the power base representation is exponential in the number of variables, precisely, the computational complexity of the standard iterative method (see Equation 4.20) is $O(d^{2n})$ with $d$ maximum degree of the polynomial and $n$ number of variables.

The *Matrix method* [168] is a technique based on operations on matrices that avoids redundant computations and reduces the computational complexity by a multiplicative factor to $O(d^{n+1})$. In [176] an efficient symbolic method for restricting the set of coefficients that contains the upper bound is proposed. Unluckily, this technique can not be applied to parametric polynomials since it requires the numerical knowledge of the coefficients of the treated polynomial in power base. However, we now propose an improvement of the matrix method that can reduce further the complexity of this technique. The gain in efficiency relies in a fast transposition of multidimensional matrices.

### The Matrix Method

The main idea of the Matrix method is to express Bernstein coefficients as a series of matrix products. For instance, the univariate polynomial[2] $\pi(x, \mathbf{p}) : \mathbb{R} \times \mathbb{R}^m \to \mathbb{R}$, that can be expressed in both power and Bernstein basis as (see Section 4.2):

$$\pi(x, \mathbf{p}) = \sum_{i=1}^{d} \mathbf{a}_i(\mathbf{p}) x^i = \sum_{i=1}^{d} \mathbf{b}_i(\mathbf{p}) \mathcal{B}_{(d,i)}(x), \tag{4.63}$$

can also be rewritten as:

$$\pi(x, \mathbf{p}) = (1, x, x^2, \ldots, x^d) \begin{pmatrix} \mathbf{a}_0(\mathbf{p}) \\ \vdots \\ \mathbf{a}_d(\mathbf{p}) \end{pmatrix} = \mathcal{X}\mathbf{a}(\mathbf{p}), \tag{4.64}$$

or as:

$$\pi(x, \mathbf{p}) = (\mathcal{B}_{(d,0)}(x), \mathcal{B}_{(d,1)}(x), \ldots, \mathcal{B}_{(d,d)}(x)) \begin{pmatrix} \mathbf{b}_0(\mathbf{p}) \\ \vdots \\ \mathbf{b}_d(\mathbf{p}) \end{pmatrix} = \mathcal{B}\mathbf{b}(\mathbf{p}), \tag{4.65}$$

where $\mathcal{B}_{(d,i)}(x)$ is the $i$-th Bernstein polynomial of degree $d$ and $\mathbf{b}_i(\mathbf{p})$ is a parametric Bernstein coefficient, with $i \in I^{\pi}$.

Defining the matrix:

$$U_x = \begin{pmatrix} 1 & 0 & 0 & \ldots & 0 \\ 1 & \binom{1}{0}\binom{d}{1}^{-1} & 0 & \ldots & 0 \\ 1 & \binom{2}{1}\binom{d}{1}^{-1} & \binom{2}{0}\binom{d}{2}^{-1} & \ldots & 0 \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ 1 & 1 & 1 & 1 & 1 \end{pmatrix} \tag{4.66}$$

for the unit box domain, it holds that $\pi(\mathbf{x}, \mathbf{p}) = \mathcal{B}\mathbf{b}(\mathbf{p}) = \mathcal{X}U_x^{-1}\mathbf{b}(\mathbf{p})$ from which follows $\mathbf{b}(\mathbf{p}) = U_x\mathbf{a}(\mathbf{p})$ [168]. Thus, to obtain the vector of Bernstein coefficients $\mathbf{b}(\mathbf{p})$

[2] We use the notation $\pi(x, \mathbf{p})$ instead of $\pi(\mathbf{x}, \mathbf{p})$ to stress the fact that the vector $\mathbf{x}$ has only one element and the indices $i$ and $d$ are scalars.

it is sufficient to multiply the lower triangular matrix $U_x$ by the polynomial coefficient vector $\mathbf{a}(\mathbf{p})$.

For multivariate polynomials, things are slightly more complicated, since the coefficients of the considered polynomial must be collected in a multidimensional matrix. Given a multivariate polynomial $\pi(\mathbf{x}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ of degree $\mathbf{d} = (\mathbf{d}_1, \ldots, \mathbf{d}_n) \in \mathbb{N}^n$, each coefficient $\mathbf{a_i}(\mathbf{p})$, with $\mathbf{i} \in I^\pi$, is placed at the $\mathbf{i}$-coordinates position in the matrix of functions $A(\mathbf{p}) \in (\mathbb{R}^m \to \mathbb{R})^{\mathbf{d}_1 \times \cdots \times \mathbf{d}_n}$. Then, Bernstein coefficients can be computed using the formula:

$$\mathbf{b}(\mathbf{p}) = (U_{\mathbf{x}_n}(\ldots(U_{\mathbf{x}_2}(U_{\mathbf{x}_1}A(\mathbf{p}))^T)^T\ldots)^T)^T \tag{4.67}$$

where the multidimensional transposition consists in a left-shift rotation of the dimensions of the matrix. For instance, if $M \in \mathbb{R}^{\mathbf{d}_1 \times \cdots \times \mathbf{d}_n}$, then $M^T \in \mathbb{R}^{\mathbf{d}_2 \times \mathbf{d}_3 \times \cdots \times \mathbf{d}_n \times \mathbf{d}_1}$.

The question is now how to compute multidimensional products and transpositions. In general, operations on multidimensional matrices can be reduced to bi-dimensional cases. In fact, a matrix $A \in \mathbb{R}^{\mathbf{d}_1 \times \cdots \times \mathbf{d}_n}$ can be represented as a matrix $C \in \mathbb{R}^{\mathbf{d}_1 \times \prod_{i=2}^n \mathbf{d}_i}$. In particular, we can define the functions $\overrightarrow{\eta} : \mathbb{N}^n \to \mathbb{N}^2$ and $\overleftarrow{\eta} : \mathbb{N}^2 \to \mathbb{N}^n$ that map an $n$-dimensional index vector $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ (specifying a position in a $n$-dimensional matrix) to a bi-dimensional index vector $(\mathbf{c}_1, \mathbf{c}_2)$ (specifying a position in a bi-dimensional matrix), and vice versa (see Figure 4.15), as:

$$\overrightarrow{\eta}(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n) = (\mathbf{a}_1, \sum_{i=2}^{n}(\mathbf{a}_i \prod_{j=2}^{i-1} \mathbf{d}_j)) \tag{4.68}$$

and $\overleftarrow{\eta}(\mathbf{c}_1, \mathbf{c}_2) = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$ where:

$$\mathbf{a}_i = \begin{cases} \mathbf{c}_1 & \text{if } i = 1; \\ \lfloor (\mathbf{c}_2 \bmod \mathbf{a}_{i+1}) / \prod_{j=2}^{i-1} \mathbf{d}_j \rfloor & \text{if } i \in \{2, n-1\}; \\ \lfloor \mathbf{c}_2 / \prod_{i=2}^{n-1} \mathbf{d}_i \rfloor & \text{if } i = n. \end{cases} \tag{4.69}$$

Hence, the product of two multidimensional matrices can be reduced to the product of their bi-dimensional versions. However, the transposition of the bi-dimensional representation of a multidimensional matrix remains to be defined.

Starting from an index vector $\mathbf{c}$, the transposition can be done in three steps:

1. Applying $\overleftarrow{\eta}(\mathbf{c}) = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$;
2. Performing the shift/transposition of the multidimensional coordinate $(\mathbf{a}_2, \mathbf{a}_3, \ldots, \mathbf{a}_n, \mathbf{a}_1)$;
3. Going back to $\overrightarrow{\eta}(\mathbf{a}_2, \mathbf{a}_3, \ldots, \mathbf{a}_n, \mathbf{a}_1) = \mathbf{c}' = \mathbf{c}^T$.

The map $\overrightarrow{\eta}$ requires $n$ sums and $n(n+1)/2$ multiplications, thus the computation of $\overrightarrow{\eta}$ is in $O(n^2)$. The whole procedures requires $\prod_{i=1}^{n} \mathbf{d}_i(O(n^2) + O(n^2)) = O(n^2 \prod_{i=1}^{n} \mathbf{d}_i)$ steps.

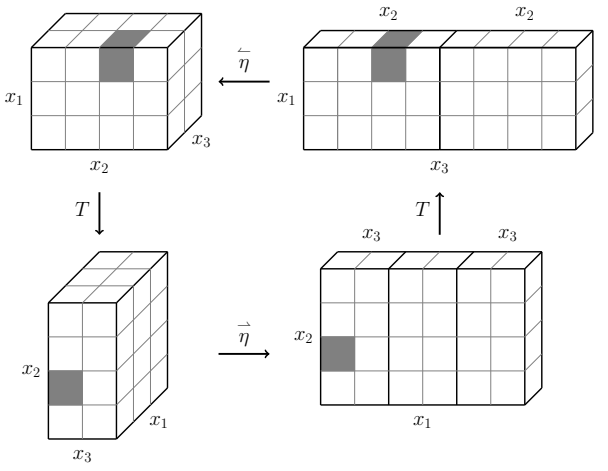



Figure 4.15: $n$-dimensional and bi-dimension transposition.

### Fast Multidimensional Transposition

To improve the efficiency of the coordinate dimension change, we propose a new method for computing the multidimensional matrix transposition operation.

Unfolding a bidimensional coordinate $\mathbf{c} = (\mathbf{c}_1, \mathbf{c}_2)$ we observe that:

$$\begin{aligned} \mathbf{c} &= (\mathbf{c}_1, \mathbf{c}_2) \\ &= (\mathbf{a}_1, \mathbf{a}_2 + \mathbf{a}_3\mathbf{d}_2 + \mathbf{a}_4\mathbf{d}_3\mathbf{d}_2 + \cdots + \mathbf{a}_n\mathbf{d}_{n-1}\ldots\mathbf{d}_2) \end{aligned} \tag{4.70}$$

and if $\overleftarrow{\eta}(\mathbf{c}_1, \mathbf{c}_2) = (\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)$, $(\mathbf{a}_1, \mathbf{a}_2, \ldots, \mathbf{a}_n)^T = (\mathbf{a}_2, \mathbf{a}_3, \ldots, \mathbf{a}_n, \mathbf{a}_1)$. Let us denote $\mathbf{c}^T = (\mathbf{c}'_1, \mathbf{c}'_2)$, then $(\mathbf{c}'_1, \mathbf{c}'_2) = (\mathbf{a}_2, \mathbf{a}_3 + \mathbf{a}_4\mathbf{d}_3 + \mathbf{a}_5\mathbf{d}_4\mathbf{d}_3 + \cdots + \mathbf{a}_n\mathbf{d}_{n-1}\ldots\mathbf{d}_3)$. From Equation 4.70 it follows that:

$$\begin{aligned} \mathbf{c}'_1 &= \mathbf{a}_2 \\ \mathbf{c}'_2 &= \frac{(\mathbf{c}_2 - \mathbf{a}_2)}{\mathbf{d}_2} + \mathbf{a}_1 \prod_{i=3}^{n} \mathbf{d}_i. \end{aligned} \tag{4.71}$$

This means that retrieving efficiently $\mathbf{a}_1$ and $\mathbf{a}_2$ from $\mathbf{c}$, we can quickly obtain $(\mathbf{c}'_1, \mathbf{c}'_2) = \mathbf{c}^T$. By Equation 4.70, the first element $\mathbf{a}_1$ is $\mathbf{c}_1$, hence $\mathbf{a}_1 = \mathbf{c}_1$. Rearranging Equation 4.70 we observe that:

$$\begin{aligned} \mathbf{c} &= (\mathbf{c}_1, \mathbf{c}_2) \\ &= (\mathbf{a}_1, \mathbf{a}_2 + \mathbf{a}_3\mathbf{d}_2 + \mathbf{a}_4\mathbf{d}_3\mathbf{d}_2 + \cdots + \mathbf{a}_n\mathbf{d}_{n-1}\ldots\mathbf{d}_2) \\ &= (\mathbf{a}_1, \mathbf{a}_2 + \mathbf{d}_2(\mathbf{a}_3 + \mathbf{d}_3(\mathbf{a}_4 + \mathbf{d}_4\ldots(\mathbf{a}_n)\ldots))) \\ &= (\mathbf{a}_1, \mathbf{a}_2 + \mathbf{d}_2 q) \end{aligned} \tag{4.72}$$

from which we deduce that $\mathbf{a}_2 = (\mathbf{c}_2 \bmod \mathbf{d}_2)$.

| | 2 | 3 | 5 | 7 | 10 |
|---|---|---|---|---|---|
| 1 | 0.00 | 0.00 | 0.00 | 0.02 | 0.69 |
| | 0.00 | 0.00 | 0.00 | 0.01 | 0.12 |
| | 0.00 | 0.00 | 0.00 | 0.00 | 0.05 |
| 3 | 0.00 | 0.01 | 0.81 | 166.35 | to |
| | 0.00 | 0.00 | 0.05 | 1.36 | to |
| | 0.00 | 0.00 | 0.04 | 0.74 | 76.52 |
| 5 | 0.00 | 0.06 | 42.84 | to | to |
| | 0.00 | 0.00 | 0.48 | 28.46 | to |
| | 0.00 | 0.00 | 0.32 | 17.77 | to |
| 10 | 0.02 | 1.88 | to | to | to |
| | 0.00 | 0.06 | 13.87 | to | to |
| | 0.00 | 0.05 | 10.54 | to | to |

Table 4.1: Computation of Bernstein coefficients (x-axis: number of variables, y-axis: maximum degree). Each line reports the computation times (in seconds) of standard method, matrix method, and improved matrix method (to: time-out 180.00s).

By substitution, from Equation 4.71, we finally obtain:

$$\begin{aligned} \mathbf{c}'_1 &= (\mathbf{c}_2 \bmod \mathbf{d}_2) \\ \mathbf{c}'_2 &= \frac{(\mathbf{c}_2 - (\mathbf{c}_2 \bmod \mathbf{d}_2))}{\mathbf{d}_2} + \mathbf{c}_1 \prod_{i=3}^{n} \mathbf{d}_i. \end{aligned} \tag{4.73}$$

With this final equation we can directly compute $\mathbf{c}^T = (\mathbf{c}'_1, \mathbf{c}'_2)$ without passing through the functions $\overleftarrow{\eta}$ and $\overrightarrow{\eta}$.

With this method, the transposition of a single element requires $n$ multiplications, hence, the whole procedure involves $O(n \prod_{i=1}^{n} \mathbf{d}_i)$ steps, which decreases the complexity of the standard multidimensional transposition method.

We implemented the standard iterative method, the matrix method, and our improved matrix method in a C++ tool (that will be presented in Section 6.1). Table 4.1 shows a comparison between the exposed methods. As a benchmark we generated polynomials of increasing number of variables (x-axis) and degree (y-axis). The polynomials are generated in such a way that each multi-index $\mathbf{i} \in I^\pi$ is associated with non-null coefficient $\mathbf{a_i} \neq 0$. In doing so, for a fixed number of variables $n$ and degrees $\mathbf{d}$, we create an extreme case where a polynomial is composed by all the possible terms from degree $(0, \ldots, 0)$ up to $(\mathbf{d}_1, \mathbf{d}_2, \ldots, \mathbf{d}_n)$. For the evaluations we impose a time-out threshold $to = 180.00$ seconds.

From the collected data we can see how matrix methods outperform the standard approach, especially from $n \geq 5$ and $\mathbf{d}_i \geq 5$. Our improved matrix method is always the fastest one and in the case $n = 10, \mathbf{d}_i = 3$ it is the only one that provides a result before the time-out.

### 4.6.2 Symbolic Coefficients

Bernstein coefficients have been used in all the presented set image algorithms. In particular, they have been exploited to bound the maximum of polynomials describing the directions of approximating sets.

To recall their role, in the following we discuss the parallelotope-based set image algorithm; however, the discussion is also valid for the box-based algorithm, since boxes are also parallelotopes.

Let us consider as reference algorithm the parallelotope-based set image algorithm reachStep (Algorithm 5). For a given parallelotope $\langle \Lambda, \mathbf{c} \rangle$, the algorithm, passing through the intermediate calls to bound (Algorithm 4) and maxBernCoeff (Algorithm 1), performs the following main steps:

1. For all the $2n$ direction $\Lambda_i$, the composition $\Lambda_i \mathbf{f}(\mathbf{x}, \mathbf{p})$ is computed;
2. The parallelotope in constraint representation $\langle \Lambda, \mathbf{c} \rangle$ is converted to the generator representation consisting in the linear transformation $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$;
3. With the function maxBernCoeff, Bernstein control points of the polynomial $\Lambda_i \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$ are computed and maximized over the parameter space.

Suppose that reachStep is used to compute the reachability set up to $k \in \mathbb{N}$ steps. With this structure, Bernstein control points are computed $k \cdot 2n$ times.

However, observing the reachStep algorithm, we notice three important aspects:

1. By definition of generator representation, once that the base vertex $\mathbf{q}$, generator lengths $\boldsymbol{\beta}$, and parameters $\mathbf{p}$ have been chosen, the domain of $\Lambda_i \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$ is the unit box, that is exactly the domain on which the range enclosing property of Bernstein coefficients holds (see Lemma 4);
2. Bernstein coefficients of the functions $\Lambda_i \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$, with $\mathbf{q}, \boldsymbol{\beta}$, and $\mathbf{p}$ seen as symbolic constants, are functions of the form $\mathbf{b_i}(\mathbf{q}, \boldsymbol{\beta}, \mathbf{p})$;
3. Both the template matrix $\Lambda$ and generator set $U$ are fixed, i.e., at each reachability step the directions of the facets of the parallelotope are the same.

This means that, since the template matrix $\Lambda$ and the generator set $U$ are fixed, we do not need to recompute Bernstein coefficients of $\Lambda_i \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$ at each reachStep call but, keeping symbolically the variables and parameters $\mathbf{q}, \boldsymbol{\beta}$, and $\mathbf{p}$, we can compute them only once obtaining a template of symbolic Bernstein coefficients that can be numerically instantiated at each reachability step. In the following we formalize this idea.

Given a template matrix $\Lambda \in \mathbb{R}^{2n \times n}$ and the correspondent set of generator versors $U = \{\mathbf{u}^1, \mathbf{u}^2, \ldots, \mathbf{u}^n\} \subset \mathbb{R}^n$ (obtainable with a representation conversion, see Section 4.4), we can produce a collection of symbolic Bernstein coefficients $\Upsilon$, that is a $2n$-dimensional vector of vectors of parametrizes coefficients of the form $\mathbf{b_i}(\mathbf{q}, \boldsymbol{\beta}, \mathbf{p}) : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$, with $\mathbf{i} \in I^{\mathbf{h}_j}$ and $\mathbf{h}_j = \Lambda_i \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$.

getSymbolicCofficients (Algorithm 7) implements the realization of a collection of symbolic coefficients building each function $\mathbf{h}_j$ and considering $\mathbf{q}, \boldsymbol{\beta}$, and $\mathbf{p}$ as symbolic constants. For each direction $\Lambda_i$ of the given template $\Lambda$ a set of symbolic Bernstein coefficients of the form $\mathbf{b_i}(\mathbf{q}, \boldsymbol{\beta}, \mathbf{p})$ is computed and stored in the collection $\Upsilon$.

**Algorithm 7** Build collection of symbolic Bernstein coefficients

```
function getSymbolicCofficients(Λ)
    for j = 1, ..., 2n do
        h_j = Λ_i f(γ_U(q, β, x), p)          ▷ Symbolic composition
        ϒ_j ←BernCoeffs(h_j)                  ▷ Compute Bernstein coefficients
    end for
    return ϒ
end function
```

**Example 21.** *Recall the dynamics of the SIR epidemic system, template matrix* $\Lambda$*, and liner transformation* $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ *from Example 15:*

$$\mathbf{f}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f}_s(\mathbf{x}, \mathbf{p}) = s - \beta si \\ \mathbf{f}_i(\mathbf{x}, \mathbf{p}) = i + \beta si - \gamma i \\ \mathbf{f}_r(\mathbf{x}, \mathbf{p}) = r + \gamma i \end{pmatrix} \tag{4.74}$$

$$\Lambda = \begin{pmatrix} -1 & 0 & 0 \\ -1 & -1 & 0 \\ 0 & 0 & -1 \\ 1 & 0 & 0 \\ 1 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{4.75}$$

$$\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) = \begin{pmatrix} \mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s \\ \mathbf{q}_2 - 0.7070\boldsymbol{\beta}_1 s + \boldsymbol{\beta}_2 i \\ \mathbf{q}_3 + \boldsymbol{\beta}_3 r \end{pmatrix}. \tag{4.76}$$

*From these elements, we build the compositions* $\mathbf{h}_j$ *for* $j \in \{1, \ldots, 6\}$*, where, for instance,* $\mathbf{h}_1(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}, \mathbf{p}) = \Lambda_1 \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p})$ *is:*

$$\begin{aligned} \mathbf{h}_1(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}, \mathbf{p}) &= (-1, 0, 0) \begin{pmatrix} \mathbf{f}_s(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ \mathbf{f}_i(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ \mathbf{f}_r(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \end{pmatrix} = -\mathbf{f}_s(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}), \mathbf{p}) \\ &= -((\mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s) - \beta(\mathbf{q}_1 + 0.7070\boldsymbol{\beta}_1 s)(\mathbf{q}_2 - 0.7070\boldsymbol{\beta}_1 s + \boldsymbol{\beta}_2 i)) \end{aligned} \tag{4.77}$$

*The collection* $\Upsilon$ *is populated computing the Bernstein coefficients of the various* $\mathbf{h}_j$ *keeping* $\mathbf{q}$*,*$\boldsymbol{\beta}$*, and* $\mathbf{p}$ *as symbolic constants. For brevity we report some examples:*

$$\begin{aligned} \Upsilon_{1,1} =& -\mathbf{q}_1 + \mathbf{q}_1\mathbf{q}_2\beta \\ \Upsilon_{1,2} =& -\mathbf{q}_1\mathbf{q}_1\boldsymbol{\beta}_2\beta + \mathbf{q}_1\mathbf{q}_2\beta \\ \Upsilon_{1,3} =& -0.3535\beta_1 - \mathbf{q}_1 + \mathbf{q}_1\mathbf{q}_2\beta - 0.3535\boldsymbol{\beta}_1\mathbf{q}_1\beta + 0.3535\boldsymbol{\beta}_1\mathbf{q}_2\beta \end{aligned} \tag{4.78}$$

*Note how the base vertex* $\mathbf{q}$ *and generator lengths* $\boldsymbol{\beta}$ *symbolically appear together with the system parameter* $\beta$ *in the symbolic coefficients.*

Suppose we computed a collection $\Upsilon$ of coefficients for a template $\Lambda$ and a parallelo-

tope $X = \langle \Lambda, \mathbf{c} \rangle$ whose generator function is $\mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ is given. In order to compute an over-approximation $X' \supseteq \mathbf{f}(X, P) = \mathbf{f}(\gamma_U(\mathbf{q}, \boldsymbol{\beta}, [0,1]^n), P)$, it is sufficient to:

1. Numerically instantiate $\mathbf{q}$ and $\boldsymbol{\beta}$ in the symbolic coefficient collection $\Upsilon$;
2. Find the maximum coefficient $\mathbf{c}'_j$ of each row $\Upsilon_j$, for $j \in \{1, \ldots, 2n\}$ over the parameter set $P$;
3. Return the parallelotope $\langle \Lambda, \mathbf{c}' \rangle$.

Note that without recomputing the Bernstein coefficients for $X$ we are able to determine $X'$. This means that, with this technique, it is sufficient to compute the coefficients once to obtain a series of parallelotope-base image over-approximations.

The function maxBernCoeff (Algorithm 8) is a variation of Algorithm 1 where, instead of a polynomial, a collection $\mathcal{B}$ of Bernstein coefficients of the form $\mathbf{b} : \mathbb{R}^n \times \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ is given, together with a base vertex $\mathbf{q} \in \mathbb{R}^n$, a vector of lengths $\boldsymbol{\beta} \in \mathbb{R}^n$, and a polytopic parameter set $P \subset \mathbb{R}^m$. The algorithm considers each coefficient $\mathbf{b}(\mathbf{q}, \boldsymbol{\beta}, \mathbf{p})$ from the collection $\mathcal{B}$, instantiates the symbolic coefficient $\mathbf{b}(\mathbf{q}, \boldsymbol{\beta}, \mathbf{p})$ with $\mathbf{q}$ and $\boldsymbol{\beta}$, maximizes the coefficient over $P$, and updates the current maximum $\overline{b}$. At the end, it returns the computed maximum $\overline{b}$.

**Algorithm 8** Compute maximum from collection of symbolic Bernstein coefficients

```
1: function maxBernCoeff(B, q, β, P)   ▷ B set of sym. coeff., q, β ∈ R^n, P ⊂ R^m
2:     b̄ ← −∞                                          ▷ Initialize current maximum
3:     for b(q, β, p) ∈ B do
4:         b ← max_{p∈P} b(q, β, p)                    ▷ Maximize current coefficient
5:         b̄ ← max{b̄, b}                               ▷ Update maximum
6:     end for
7:     return b̄
8: end function
```

Now, let us assume that a collection $\Upsilon$ of symbolic Bernstein coefficients for a template $\Lambda \in \mathbb{R}^{2n \times n}$ has been computed using getSymbolicCoefficients (Algorithm 7). Such a collection can be exploited in a parallelotope-based set image algorithm. The algorithm reachStep based on parametric symbolic coefficients is shown in Algorithm 9. It receives in input a parallelotope $X = \langle \Lambda, \mathbf{c} \rangle \subset \mathbb{R}^n$ in template representation, a parameter set $P \subset \mathbb{R}^m$, and a precomputed collection of symbolic Bernstein coefficients $\Upsilon$. The algorithm begins by converting the parallelotope from constraints to generators, extracting the current numerical base vertex $\mathbf{q}$ and generator lengths $\boldsymbol{\beta}$. Then, each offset $\mathbf{c}'_j$ with $j \in \{1, \ldots, n\}$ is obtained with the call to the function maxCoeff$(\Upsilon_j, \mathbf{q}, \boldsymbol{\beta}, P)$ that numerically instantiates each element of $\Upsilon$ with $\mathbf{q}$ and $\boldsymbol{\beta}$, maximizes each coefficient over the parameter set $P$, and finally returns the maximum numerical offset $\mathbf{c}'_j$. Finally, the algorithm combines the computed offsets $\mathbf{c}'$ with the template $\Lambda$ and returns the over-approximating parallelotope $\langle \Lambda, \mathbf{c}' \rangle$.

The benefit of this approach is that Bernstein coefficients are precomputed only once and evaluated on the fly at each set image step. When calculating the reachable set up to $k \in \mathbb{N}$ steps, we compute Bernstein coefficients only $2n$ times at the very beginning of

**Algorithm 9** Parallelotope-based reachable set integration with symbolic coefficients

**function** REACHSTEP($X, P, \Upsilon$) ▷ $X = \langle \Lambda, \mathbf{c} \rangle$ parallelotope, $\Upsilon$ symbolic collection
  $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x}) \leftarrow$CON2GEN($X$) ▷ Get numerical $\mathbf{q}$ and $\boldsymbol{\beta}$
  **for** $j = 1, \ldots, 2n$ **do**
    $\mathbf{c}'_j \leftarrow$MAXCOEFF($\Upsilon_j, \mathbf{q}, \boldsymbol{\beta}, P$) ▷ Determine maximum coefficient
  **end for**
  **return** $\langle \Lambda, \mathbf{c}' \rangle$
**end function**

the reachability algorithm, against the $k2n$ times of the standard approach. Thanks to this precomputation, the reduction of the computational load and time is remarkable. A similar technique will be later exploited in the refinement of parameter sets.

### 4.6.3 Improving Precision

We conclude the chapter providing a method to obtain from Bernstein coefficients tighter bounds and consequently generate finer over-approximations of reachable sets.

#### Subdivision Procedure

The *subdivision procedure* is a technique that improves the bounds provided by Bernstein coefficients. A subdivision of a box $X \subset \mathbb{R}^n$ in the $r$-th direction, with $1 \leq r \leq n$, is a bisection perpendicular to this direction. Let $X = [\underline{x}_1, \overline{x}_1] \times \ldots [\underline{x}_n, \overline{x}_n] \subseteq [0,1]^n$. If $X$ is subdivided along the $r$-th component direction at some point $\lambda_r \in [0,1]$, the resulting two sub-boxes $X_A$ and $X_B$ are $X_A = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\underline{x}_r, \hat{x}_r] \times \cdots \times [\underline{x}_n, \overline{x}_n]$ and $X_B = [\underline{x}_1, \overline{x}_1] \times \cdots \times [\hat{x}_r, \overline{x}_r] \times \cdots \times [\underline{x}_n, \overline{x}_n]$ where $\hat{x}_r = \underline{x}_r + \lambda_r(\overline{x}_r - \underline{x}_r)$.

A trivial way to improve bounds is to directly recompute the coefficients for $X_A$ and $X_B$, mapping the unit box to $X_A$ and $X_B$, and then using the standard procedure (see Equation 4.20). However, there exists a more efficient method that allows one to obtain the coefficients for $X_A$ and $X_B$ exploiting the already computed coefficients of $X$ [82, 89, 175].

Let $\pi : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}$ be a parametric polynomial, $1 \leq \mathbf{i}_r \leq n$ be a subdivision direction, and $\lambda \in [0,1]$ be a subdivision point. The subdivision procedure begins with the computation of $B^{(0)}_\pi = B_\pi = \{\mathbf{b}^{(0)}_{\mathbf{i}}(\mathbf{p}) \mid \mathbf{i} \in I^\pi\}$ that is the standard set of Bernstein coefficients for $\pi$. We then update $\mathbf{b}^{(k)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r,\ldots,\mathbf{i}_n)}(\mathbf{p})$ as follows:

- If $\mathbf{i}_r < k$, $\mathbf{b}^{(k)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r,\ldots,\mathbf{i}_n)}(\mathbf{p}) = \mathbf{b}^{(k-1)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r,\ldots,\mathbf{i}_n)}(\mathbf{p})$;
- If $\mathbf{i}_r \geq k$, $\mathbf{b}^{(k)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r,\ldots,\mathbf{i}_n)}(\mathbf{p}) = (1-\lambda_r)\mathbf{b}^{(k-1)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r-1,\ldots,\mathbf{i}_n)}(\mathbf{p}) + \lambda_r \mathbf{b}^{(k-1)}_{(\mathbf{i}_1,\ldots,\mathbf{i}_r,\ldots,\mathbf{i}_n)}(\mathbf{p})$.

To obtain the new coefficients of $X_A$ after the subdivision of $X$, we apply the above update rules for $\mathbf{i}_j = 0, \ldots, \mathbf{d}_j$, $j = 1, \ldots, r-1, r+1, \ldots, n$. Then, $B_\pi(X_A) = B^{(\mathbf{d}_r)}_\pi(X)$. Finally, we obtain directly the coefficients $B_\pi(X_B)$ since, for $k = 0, 1, \ldots, \mathbf{d}_r$, it holds that:

$$\mathbf{b}_{(\mathbf{i}_1,\ldots,\mathbf{d}_r-k,\ldots,\mathbf{i}_n)}(\mathbf{p})(X_2) = \mathbf{b}^{(n)}_{(\mathbf{i}_1,\ldots,\mathbf{d}_r,\ldots,\mathbf{i}_n)}(\mathbf{p})(X_1). \tag{4.79}$$

The complexity of the subdivision is $O(\mathbf{d}_r^{n+1})$.

The main property of the subdivision is that the minimum (maximum) coefficient of the union $B_\pi(X_A) \cup B_\pi(X_B)$ is not smaller (greater) than the minimum one of $B_\pi(X)$. This means that the subdivision can improve and always preserves the accuracy of the range enclosure property.

### Selection of Subdivision Direction

The subdivision method requires the identification of a direction $\mathbf{i}_r$ and a subdivision point $\lambda$ through which perform the splitting. In this section we consider the problem of finding an appropriate component direction for the subdivision to achieve better approximations of the polynomial values by Bernstein coefficients. Heuristics for the selection of a subdivision direction have been considered in [166, 189, 91, 152, 167].

Here we present a technique based on the largest first derivative of the studied polynomial [91, 167]. We use the following notation to express neighboring of multi-indices. Let $\mathbf{i} = (\mathbf{i}_1, \ldots, \mathbf{i}_n)$ be a multi-index, then $\mathbf{i}_{(r,k)} = (\mathbf{i}_1, \ldots, \mathbf{i}_r + k, \ldots, \mathbf{i}_n)$ with $0 \leq \mathbf{i}_r + k \leq \mathbf{d}_r$.

The selection is preformed by exploiting again Bernstein expansion. More concretely, we estimate:

$$\max_{\mathbf{x} \in X, \mathbf{p} \in P} |\frac{\partial \pi}{\partial \mathbf{x}_r}(\mathbf{x}, \mathbf{p})| \tag{4.80}$$

for each variable. Then, we select $r \in \{1, \ldots, n\}$ with the largest derivative value of $X$ and $P$. Exploiting Bernstein properties, the first partial derivative of the polynomial with respect to $\mathbf{x}_r$ is given by:

$$\frac{\partial \pi}{\partial \mathbf{x}_r}(\mathbf{x}, \mathbf{p}) = \mathbf{d}_r \sum_{\mathbf{i} \leq \mathbf{d}_{(r,-1)}} (\mathbf{b}_{\mathbf{i}_{(r,1)}}(\mathbf{p}) - \mathbf{b}_{\mathbf{i}}(\mathbf{p})) \mathcal{B}_{(\mathbf{d}_{(r,-1)}, \mathbf{i})}(\mathbf{x}) \tag{4.81}$$

where $\mathcal{B}_{(\mathbf{d}_{(r,-1)}, \mathbf{i})}(\mathbf{x})$ is the $\mathbf{i}$-th Bernstein basis of degree $\mathbf{d}_{(r,-1)}$ (see Section 4.2), that implies that we do not need to explicitly compute the various derivatives, but we can rely on Bernstein coefficients.

### Finding a Subdivision Point

Once the subdivision direction $r \in \{1, \ldots, n\}$ has been chosen accordingly with the first derivative, the second task is to find a subdivision point $\lambda_r \in [0, 1]$ such that the subdivision improves as much as possible the bounds. Since we are interest in decreasing the upper bound, we should subdivide the domain in order to retain as much as possible the coefficient:

$$\overline{\mathbf{b}}_{\mathbf{i}}(\mathbf{p}) = \max_{\mathbf{i} \in I^\pi, \mathbf{p} \in P} \mathbf{b}_{\mathbf{i}}(\mathbf{p}) \tag{4.82}$$

that determines the bound. We chose as division point $\lambda_r \in [0, 1]$ the projection of $\overline{\mathbf{b}}_{\mathbf{i}}(\mathbf{p})$ with the $\mathbf{x}_r$ plane, since a good subdivision has to separate this critical intersection point from the rest.

Note that Bernstein coefficients $\mathbf{b}_{\mathbf{i}}(\mathbf{p})$ are function of the parameters $\mathbf{p}$; to compute this projection, we can determine around each control point a box that accounts the uncertainty in $\mathbf{p}$, and then project the vertices of the resulting boxes (see Figure 4.16).

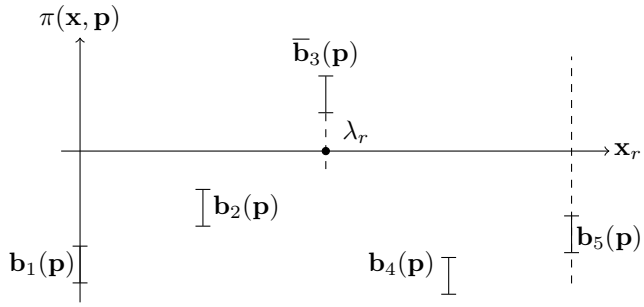



Figure 4.16: Finding a division point $\lambda_r$ as the intersection of the maximum control point with the $\mathbf{x}_r$-axis.

The whole subdivision procedure can be repeated until the real maximum is determined, condition that can be verified by using the sharpness property (see Lemma 5), or until the sizes of the sub-boxes reach a given threshold.

# 5

# Parameter Synthesis

In this chapter we focus on the *parameter synthesis problem*, that is, given a dynamical system $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$, a set of initial conditions $X_0 \subseteq \mathcal{X}$, a parameter set $P \subseteq \mathcal{P}$, and a specification $\varphi$, find the largest subset $P_\varphi \subseteq P$ such that for all $\mathbf{x}_0 \in X_0$ and $\mathbf{p} \in P$, the trajectories $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ of $\mathcal{S}$ satisfy the specification $\varphi$. In this work we assume that the specifications are given as formulas of the *Signal Temporal Logic* (STL), a logic that allows one to specify properties on dense-time real-valued functions.

STL and its standard semantics will be presented in Section 5.1. In order to adapt STL to our parameter synthesis context, in Section 5.2 we will define a new semantics, called *synthesis semantics*, to reason on flowpipes and sets of parameters rather than on single trajectories. In Section 5.3 we will present a synthesis algorithm that can be used to solve the parameter synthesis problem for discrete-time dynamical systems and STL specifications. Its correctness and complexity will be discussed in Section 5.3.4. Finally, in Section 5.4, also exploiting the reachability techniques developed in Chapter 4, we will focus on the implementation of our synthesis algorithm in the case of discrete-time polynomial dynamical systems.

## 5.1 Signal Temporal Logic

Temporal logic [163] is a formalism used to specify and reason on properties that involve time. It is typically adopted in the context of formal verification, where a temporal logic formula specifies the acceptable behaviors of a system and an algorithm is used to check whether all the behaviors of the system satisfy the formula. This procedure is commonly known as *model checking* [46]. Recently, temporal logic has found applications outside formal verification, for instance in *monitoring*. In this case, a formal model is not necessary, since the system can be treated as a black box whose observable behaviors can be monitored by evaluating the satisfaction level of the desired temporal property.

*Signal Temporal Logic* (STL [143, 144]) is a recent logic that allows one to specify properties of dense-time real-valued signals. It is particularly suitable for monitoring blackbox systems, such as industrial models (see, e.g., [116, 115]) and biological systems (see, e.g., [68, 177]). It has also been used in the study of parametric systems, (see,

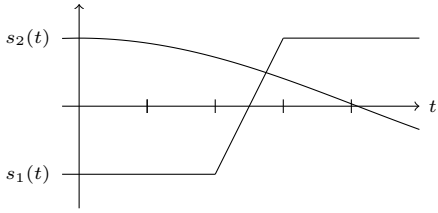



Figure 5.1: Trace $w = \{s_1, s_2\}$ and its signals $s_1$ and $s_2$. Time horizon $h(w) = 5$.

e.g., [7]) where parametric disturbance rejection properties are formalized in STL and then verified. An interesting aspect of STL 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 or violated [79, 11].

We now take advantage of STL in the context of the parameter synthesis problem for dynamical systems. More concretely, we exploit its ability to reason over signals to specify desirable executions of the dynamical system under study. Our final goal will be the determination of sets of parameter values such that a dynamical system, under the computed parameter set, satisfies the given STL specification.

### Syntax

Let us begin with the definition of STL introducing some basic concepts. A *signal* is a function $s : D \to S$, with $D \subseteq \mathbb{R}_{\geq 0}$ an interval of $\mathbb{R}_{\geq 0}$ and $S \subseteq \mathbb{R} \cup \mathbb{B}$, where $\mathbb{R}$ and $\mathbb{B}$ are the set of reals and booleans, respectively. Signals defined on $S = \mathbb{B}$ are called *boolean* signals, while those with $S = \mathbb{R}$ are called *real-valued* signals. A *trace* $w = \{s_1, s_2, \ldots, s_n\}$ is a set of real-valued signals of the form $s_i : D_i \to S$ with $D_i \subset \mathbb{R}_{\geq 0}$ an interval of $\mathbb{R}_{\geq 0}$, for $i \in \{1, \ldots, n\}$. We denote with $h(w)$ the *time horizon* of $w$ that is the smallest last instant on which the signals $s_i$ of $w$ are defined, i.e.:

$$h(w) = \min_{i \in \{1,\ldots,n\}} \sup D_i. \tag{5.1}$$

**Example 22.** *Figure 5.1 shows a trace $w = \{s_1, s_2\}$ composed by two signals $s_1$ and $s_2$. The time horizon of $w$ is the smallest last time instant on which $s_1$ and $s_2$ are defined, that is $h(w) = 5$.*

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

A Signal Temporal Logic [143, 71] formula is generated by the following grammar:

$$\varphi := \sigma \mid \neg\varphi \mid \varphi \wedge \varphi \mid \varphi U_I \varphi \tag{5.2}$$

where $\sigma \in \Sigma$ is a predicate and $I$ is a closed non-singular interval of $\mathbb{R}_{\geq 0}$, which includes bounded $[a, b]$ and unbounded $[a, +\infty)$ intervals for any $0 \leq a < b$. For $t \in \mathbb{R}$, the shifted interval $t + I$ is the set $\{t + t' \mid t' \in I\}$. The *time horizon* $h(\varphi)$ of a formula $\varphi$ is

the last time instant to which $\varphi$ refers, i.e.:

$$\begin{aligned} h(\sigma) &= 0 \\ h(\neg\varphi) &= h(\varphi) \\ h(\varphi_1 \wedge \varphi_2) &= \max\{h(\varphi_1), h(\varphi_2)\} \\ h(\varphi_1 U_{[a,b]} \varphi_2) &= \max\{h(\varphi_1) + b, h(\varphi_2) + b\} \end{aligned} \tag{5.3}$$

In the following, given a trace $w$ and a formula $\varphi$, we assume that the time horizon of $w$ is greater than the time horizon of $\varphi$, i.e., $h(w) > h(\varphi)$. This ensures that the trace $w$ is long enough to evaluate the formula $\varphi$.

### Qualitative Semantics

An interesting aspect of STL is its semantics. We define two semantics for STL formulas: a *qualitative* semantics, also known as Boolean semantics, and a *quantitative* semantics. Intuitively, the first establishes the truth value of a formula over a trace, telling us whether a formula holds or not; the second provides additional information on how robustly a trace satisfies (or not) a formula. As we will see later, there is a strong relationship between these two semantics. Later, for the synthesis of parameters, we will take advantage only of the qualitative one. However, for completeness and future insights, we present both of them.

**Definition 14** (Qualitative Semantics)**.** *Let $w$ be a trace, $t \in \mathbb{R}_{\geq 0}$ be a time instant, and $\varphi$ be an STL formula. The* qualitative semantics *of $\varphi$ at time $t$ over the trace $w$ is given by the following inductive definition:*

$$\begin{aligned} w, t &\models \sigma \quad \textit{iff } \sigma(w(t)) \textit{ is true} \\ w, t &\models \neg\varphi \quad \textit{iff } w, t \not\models \varphi \\ w, t &\models \varphi_1 \wedge \varphi_2 \quad \textit{iff } w, t \models \varphi_1 \textit{ and } w, t \models \varphi_2 \\ w, t &\models \varphi_1 U_I \varphi_2 \quad \textit{iff } \exists t' \in t + I \textit{ s.t. } w, t' \models \varphi_2 \textit{ and for all } t'' \in [t, t'], w, t'' \models \varphi_1 \end{aligned} \tag{5.4}$$

We say that a trace $w$ *satisfies* $\varphi$ if and only if $w, 0 \models \varphi$. For brevity, we write $w \models \varphi$ meaning $w, 0 \models \varphi$.

For a given formula $\varphi$, a trace $w$, and a time instant $t \in \mathbb{R}_{\geq 0}$, the *satisfaction signal* $\mathcal{X}(\varphi, w, t)$ is defined as:

$$\mathcal{X}(\varphi, w, t) = \begin{cases} \top \text{ if } w, t \models \varphi, \\ \bot \text{ otherwise.} \end{cases} \tag{5.5}$$

The satisfaction signal $\mathcal{X}(\varphi, w, t)$ tells us whether at the time $t$ the monitored trace $w$ satisfies or not the STL formula $\varphi$. The construction of the satisfaction signal can be recursively done on the structure of the formula, starting from the predicates and proceeding bottom-up [143].

**Example 23.** *Figure 5.2 shows a trace $w = \{s_1, s_2\}$, composed by two signals $s_1$ and $s_2$ (solid), and the satisfaction signals of some STL formulas (dashed). Figure 5.2a depicts the signal $s_1$ and the satisfaction signal of the formula $\varphi_1 = x_1 > -0.3$. Note how the*

*signal jumps to true at the moment in which $s_1$ becomes larger than $-0.3$. Figure 5.2b shows the satisfaction signal of the formula $\neg\varphi_1 = \neg(x_1 > -0.3)$ evaluated on the same signal $s_1$. Note how, due to the negation, the satisfaction signal is symmetric (with respect to the t-axis) to the previous case. Figure 5.2c shows a different STL atomic formula $\varphi_2 = x_2 > 0.5$ evaluated on a different signal $s_2$. In this case, the satisfaction signal, starting from the true value, switches to false in the moment in which $s_2$ becomes smaller than $0.5$. Figure 5.2d considers both the signals $s_1$ and $s_2$ and the formula $\neg\varphi_1 \wedge \varphi_2 = \neg(x_1 > -0.3) \wedge (x_2 > 0.5)$. Here the satisfaction signal of $\neg\varphi_1 \wedge \varphi_2$ starts from the true value, since both $s_1$ and $s_2$ satisfies both the conjuncts $\neg\varphi_1 = \neg(x_1 > -0.3)$ and $\varphi_2 = (x_2 > 0.5)$, but jumps to false in the moment in which the signal $s_1$ is no longer smaller than $-0.3$ and consequently the conjunct $\neg\varphi_1 = \neg(x_1 > -0.3)$ is not satisfied. Finally, Figure 5.2e shows both the signals $s_1$ and $s_2$ and the formula $\neg\varphi_1 U_{[1,2]}\varphi_2 = \neg(x_1 > -0.3)U_{[1,2]}(x_2 > 0.5)$. The satisfaction signal of $\neg(x_1 > -0.3)U_{[1,2]}(x_2 > 0.5)$ starts from the true value, since there exists a time instant between $1$ and $2$ (like for instance $1.1$) where $\varphi_2$ holds and in the intermediate interval (like $[0, 1.1]$) $\neg\varphi_1$ always holds. The signal jumps to false in $t = 1.35$ since in $1.35 + [1, 2] = [2.35, 3.35]$ there is not an instant $t'$ in which $\varphi_2$ holds and $\neg\varphi_1$ holds in $[1.35, t']$.*

### Quantitative Semantics

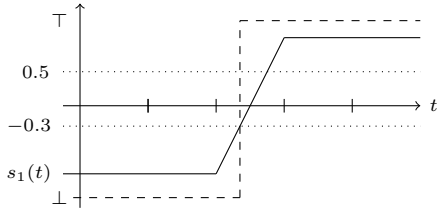
We now define an alternative semantics for STL formulas called *quantitative semantics*. The particularity of the quantitative semantics is that, in addition of knowing whether a signal satisfies or not a formula, it establishes how robustly the formula is satisfied or not. Intuitively, the quantitative evaluation of a formula provides a real value representing the distance to satisfaction or violation.

**Definition 15** (Quantitative Semantics)**.** *Let $w$ be a trace, $t \in \mathbb{R}_{\geq 0}$ be a time instant, and $\varphi$ be an STL formula. The* quantitative semantics *$\rho(\varphi, w, t)$ of $\varphi$ at time $t$ over the trace $w$ is given by the following inductive definition:*

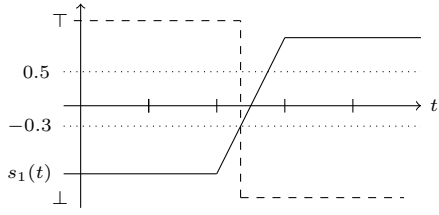
$$\begin{aligned} \rho(p(x_1, \ldots, x_n) \sim 0, w, t) &= p(w(t)) \text{ with } \sim \in \{<, \leq\} \\ \rho(\neg\varphi, w, t) &= -\rho(\varphi, w, t) \\ \rho(\varphi_1 \wedge \varphi_2, w, t) &= \min(\rho(\varphi_1, w, t), \rho(\varphi_2, w, t)) \\ \rho(\varphi_1 U_I \varphi_2, w, t) &= \sup_{t' \in t+I} \min(\rho(\varphi_2, w, t'), \inf_{t'' \in [t, t']} \rho(\varphi_1, w, t'')) \end{aligned} \tag{5.6}$$

The *robustness signal* of a formula $\varphi$ over a trace $w$ is the signal $\rho(\varphi, w, \cdot)$.

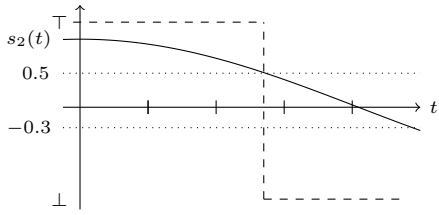
**Example 24.** *In this example we consider the same trace $w = \{s_1, s_2\}$ and formulas of Example 23, but we evaluate them using the qualitative semantics. We begin with the predicate $\varphi_1 = x_1 > -0.3$ and the signal $s_1$ depicted in Figure 5.3a. Applying the function $p$ that defines the predicate $\varphi_1$ to the signal $s_1$, we obtain the robustness signal $p(s_1(t)) = s_1(t) + 0.3$ of $\varphi_1$ with respect to $w$, that in this case consists in a shift of the signal $s_1$. Note how, in the moment in which $s_1$ becomes larger than $-0.3$, the robustness signal of $\varphi_1$ becomes positive. Figure 5.3b shows the same signal $s_1$ and the robustness signal of the formula $\neg\varphi = \neg(s_1(t) > -0.3)$ obtained by changing the sign of the robustness signal of $\varphi_1$. Figure 5.3c shows the robustness signal of a different predicate $\varphi_2 = x_2 > 0.5$ and signal $s_2$. Figure 5.3d shows the qualitative semantics of the*



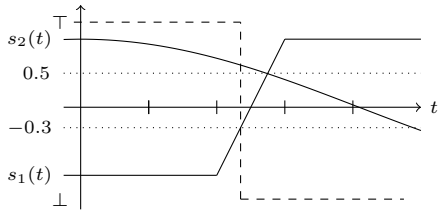

(a) $s_1(t)$ and $\mathcal{X}(\varphi_1, w, t)$



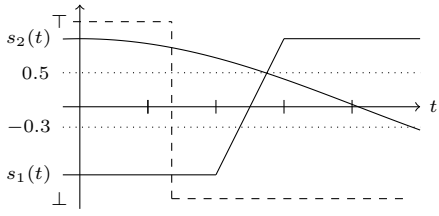

(b) $s_1(t)$ and $\mathcal{X}(\neg\varphi_1, w, t)$




(c) $s_2(t)$ and $\mathcal{X}(\varphi_2, w, t)$

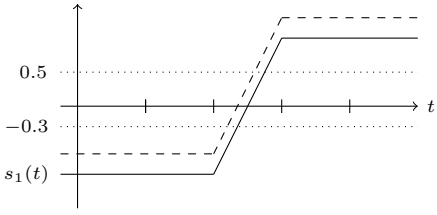



(d) $w(t)$ and $\mathcal{X}(\neg\varphi_1 \wedge \varphi_2, w, t)$.

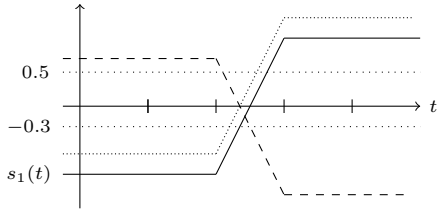



(e) $w(t)$ and $\mathcal{X}(\neg\varphi_1 U_{[1,2]}\varphi_2, w, t)$.

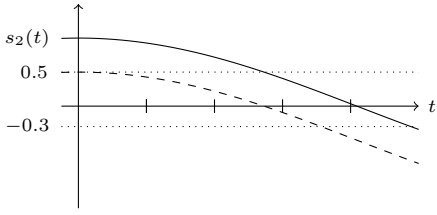
Figure 5.2: Trace $w = \{s_1, s_2\}$ (solid) and the satisfaction signals of formulas obtained from the atomic predicates $\varphi_1 = x_1 > -0.3$ and $\varphi_2 = x_2 > 0.5$ (dashed).



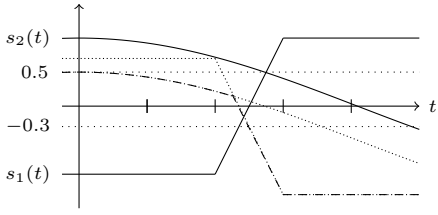

(a) $s_1(t)$ and $\rho(\varphi_1, w, t)$



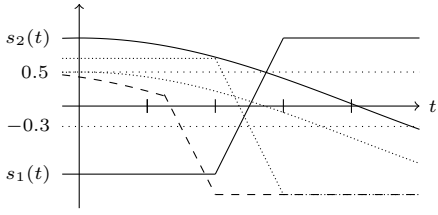

(b) $s_1(t)$ and $\rho(\neg\varphi_1, w, t)$




(c) $s_2(t)$ and $\rho(\varphi_2, w, t)$




(d) $w(t)$ and $\rho(\neg\varphi_1 \wedge \varphi_2, w, t)$




(e) $w(t)$ and $\rho(\neg\varphi_1 U_{[1,2]} \varphi_2, w, t)$

Figure 5.3: Trace $w = \{s_1, s_2\}$ (solid) and quantitative semantics of formulas built from the atomic predicates $\varphi_1 = x_1 > -0.3$ and $\varphi_2 = x_2 > 0.5$ (dashed).

*conjunction* $\neg\varphi_1 \wedge \varphi_2 = \neg(s_1(t) > -0.3) \wedge (s_2(t) > 0.5)$ *obtained by taking the minimum between the robustness signals of* $\neg\varphi_1$ *and* $\varphi_2$*. Finally, Figure 5.3e shows both the signals* $s_1$ *and* $s_2$ *and the robustness signal of* $\neg\varphi_1 U_{[1,2]} \varphi_2 = \neg(x_1 > -0.3) U_{[1,2]} (x_2 > 0.5)$*. The robustness at time* $t$ *is obtained as the maximum of the minimums of* $\rho((x_2 > 0.5), w, t')$ *and the inferiors of* $\rho(\neg(x_1 > -0.3), w, t'')$ *with* $t'' \in [t, t']$ *and* $t' \in t + [1, 2]$*.*

The quantitative and qualitative semantics are connected by two central properties [71]. The first concerns the sign of a qualitative evaluation, specifically, if $\rho(\varphi, w, t) \neq 0$, then its signs indicates the satisfaction status.

**Theorem 3.** *Let* $\varphi$ *be an STL formula,* $w$ *be a trace, and* $t$ *be a time instant.*

$$\begin{aligned} \rho(\varphi, w, t) > 0 &\implies w, t \models \varphi \\ \rho(\varphi, w, t) < 0 &\implies w, t \not\models \varphi \end{aligned} \tag{5.7}$$

The second property concerns the proximity of traces, the respective robustness signals, and their satisfaction status.

**Theorem 4.** *Let $\varphi$ be an STL formula, $w$ and $w'$ traces, and $t$ a time instant.*

$$w, t \models \varphi \text{ and } \|w - w'\|_\infty < \rho(\varphi, w, t) \implies w', t \models \varphi \tag{5.8}$$

*where $\|\cdot\|_\infty$ is the infinity norm.*

This means that if $w$ satisfies $\varphi$ at time $t$, then any trace $w'$ whose distance form $w$ is smaller then $\rho(\varphi, w, t)$ also satisfies $\varphi$ at time $t$.

As previously mentioned, STL is the logic that we will use to specify the admissible behaviors of our dynamical systems. The reason why we opt for STL is that the trajectories generated by a dynamical system can be interpreted as traces on which STL formulas can be evaluated. Therefore, STL is a natural tool to reason on trajectories. However, we aim to study sets of trajectories generated from a dynamical system under different initial conditions and parameters. It is therefore necessary to adapt the semantics of STL in order to be able to work with flows of signals (flowpipes), rather than with single trajectories. For this reason, in the next section we will introduce a new semantics, called *synthesis semantics*, that allows us to reason with formulas over flows of traces and sets of parameters.

## 5.2 STL Synthesis Semantics

In Section 5.1 we have seen how STL can be used to express properties of traces whose components are signals of the form $s : D \to S$ with $D \subseteq \mathbb{R}_{\geq 0}$ interval and $S \subseteq \mathbb{R} \cup \mathbb{B}$. In Section 2.1 we defined trajectories of dynamical systems $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ that are functions of the form $\xi^{\mathbf{p}}_{\mathbf{x}_0} : \mathbb{T} \to \mathcal{X}$ with $\mathbb{T} = \mathbb{R}_{\geq 0} \cup \mathbb{N}$, $\mathbf{x}_0 \in \mathcal{X}$, and $\mathbf{p} \in \mathcal{P}$. It is not hard to see that if the state space of the dynamical system $\mathcal{X}$ is $\mathbb{R}^n$ (that is often the case), then the trajectories of the dynamical system can be interpreted as traces, and then they can be used to instantiate STL formulas. The correspondence between traces and trajectories makes STL a suitable logic for reasoning on dynamical systems.

Let $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a generic dynamical system and $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ be a trajectory of $\mathcal{S}$ with $\mathbf{x}_0 \in \mathcal{X}$ and $\mathbf{p} \in \mathcal{P}$. Let $\varphi$ be an STL formula. With a slight abuse of notation we write:

$$\xi^{\mathbf{p}}_{\mathbf{x}_0} \models \varphi \tag{5.9}$$

to indicate the qualitative semantics of $\varphi$ at time 0 over the trace correspondent to the trajectory $\xi^{\mathbf{p}}_{\mathbf{x}_0}$.

Since we aim to reason on sets of trajectories and parameters, we extend the notion of formula satisfaction to flows of trajectories. Let $X_0 \subseteq \mathcal{X}$ and $P \subseteq \mathcal{P}$. We recall that $\Xi(X_0, P)$ is the set of all the trajectories $\xi^{\mathbf{p}}_{\mathbf{x}_0}$ with initial conditions $\mathbf{x}_0 \in X_0$ and parameters $\mathbf{p} \in P$ (see Section 2.1). We say that a set of trajectories $\Xi(X_0, P)$ *satisfies* a formula $\varphi$ at time 0, denoted with $\Xi(X_0, P) \models \varphi$ if and only if:

$$\forall \xi^{\mathbf{p}}_{\mathbf{x}_0} \in \Xi(X_0, P), \xi^{\mathbf{p}}_{\mathbf{x}_0} \models \varphi. \tag{5.10}$$

At this point, we have all the elements to formalize our *parameter synthesis problem* (already anticipated in Section 2.2.1) over dynamical systems and STL specifications.

**Definition 16** (Parameter Synthesis Problem)**.** *Let $\mathcal{S} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a dynamical system, $X_0 \subseteq \mathcal{X}$ be an initial set, $P \subseteq \mathcal{P}$ be a parameter set, and $\varphi$ be an STL formula. Find the largest subset $P^*_\varphi \subseteq P$ such that starting from $X_0$, all the trajectories of the system satisfy $\varphi$ at time 0, that is:*

$$\Xi(X_0, P^*_\varphi) \models \varphi. \tag{5.11}$$

Note that if we consider discrete-time dynamical systems and formulas with bounded time horizon, we can recast the parameter synthesis problem in LTL formulas involving Boolean and next operators interpreted over finite traces [62]. However, STL offers some advantages. First, an LTL formula expressing the bounded temporal operators (like the until) may be long and difficult to treat. Furthermore, STL is adapt for both continuous-time and discrete-time traces, aspect that makes it suitable for both continuous-time and discrete-time dynamical systems. Moreover, for some classes of systems, the quantitative analysis on a time-discretized system gives complete information also on its continuous-time version [80].

The parameter synthesis problem requires handling of flows of parametric traces, that can be hard, especially when the solution of the dynamical system can be only approximated, as in our discrete-time polynomial case (see Chapter 3). Therefore we need to find a compromise between precision and tractability of the problem. For this reason we introduce a relaxation of the synthesis problem for approximated sets of trajectories.

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 a flowpipe $\mathcal{W}_{(X_0,P)}$, such that, for a given $t \in \mathbb{N}$, $\mathcal{W}_{(X_0,P)}(t) = X_t$, where $X_t$ is obtained with the following recursive scheme:

$$X_{j+1} = \{\mathbf{f}(\mathbf{x}, \mathbf{p}) \mid \mathbf{x} \in X_j, \mathbf{p} \in P\}. \tag{5.12}$$

The over-approximating flowpipe $\mathcal{W}_{(X_0,P)}$ can be computed with the set-integration algorithms presented in Chapter 3. However it is important to note that this is an over-approximation of $\Xi(X_0, P)$ since the relation between a single trace and its corresponding parameter is not kept.

Let us now start combining flowpipes with STL formulas. In the following, we consider STL formulas in *positive normal form* with bounded time horizon, i.e., formulas generated by the following grammar:

$$\varphi := \sigma \mid \varphi \wedge \varphi \mid \varphi \vee \varphi \mid \varphi U_I \varphi \tag{5.13}$$

with $\sigma = p(x_1, \ldots, x_n) \leq 0$, where $p : \mathbb{R}^n \to \mathbb{R}$, and $I = [a, b] \subset \mathbb{N}$. Note that in this grammar, differently from the standard syntax (see Section 5.1), the negation operator in not included and the disjunction is explicitly defined.

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

**Definition 17** (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 $\varphi$ be an STL formula in positive normal form. The* synthesis semantics $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$

*of $\varphi$ at time $t$ over the flowpipe $\mathcal{W}_{(X_0,P)}$ is given by the following inductive definition:*

$$\begin{aligned}
\Phi(\sigma, \mathcal{W}_{(X_0,P)}, t) &= P_\sigma, \text{ where } P_\sigma \subseteq P \text{ is the largest subset such that} \\
&\quad \forall \mathbf{x}_0 \in \mathcal{W}_{(X_0,P)}(t), \forall \mathbf{p} \in P_\sigma, \sigma(\mathbf{f}(\mathbf{x}_0, \mathbf{p})) \text{ is true} \\
\Phi(\varphi_1 \wedge \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t) \cap \Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t) \\
\Phi(\varphi_1 \vee \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t) \cup \Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t) \\
\Phi(\varphi_1 U_I \varphi_2, \mathcal{W}_{(X_0,P)}, t) &= \bigcup_{t' \in t+I} (\Phi(\varphi_2, \mathcal{W}_{(X_0,P)}, t') \cap \bigcap_{t'' \in [t,t']} \Phi(\varphi_1, \mathcal{W}_{(X_0,P)}, t''))
\end{aligned} \tag{5.14}$$

Intuitively, $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$ returns a subset $P_\varphi \subseteq P$ of parameters that ensures that $\varphi$ is satisfied at time $t$ starting from any point in $X_0$ and assigning to the parameters any value in $P_\varphi$. Note that the synthesis semantics at time $t$ returns a set $P_\varphi$ that steers the system from time $t+1$ on. This is slightly counterintuitive since usually the semantics return evaluations referring to the time instant in which they are applied.

We say that a flowpipe $\mathcal{W}_{(X_0,P)}$ *satisfies* a formula $\varphi$, denoted with $\mathcal{W}_{(X_0,P)} \models \varphi$ if and only if $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t) = P$, i.e., the synthesis semantics does not affect the parameter set $P$. It is easy to see that $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, t)$ is idempotent. This implies that the synthesis semantics provides a refined set of parameters that satisfies the formula $\varphi$. This statement is proved in the following theorem.

**Theorem 5.** *If $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0) = P_\varphi$, then $\mathcal{W}_{(X_0,P_\varphi)} \models \varphi$.*

*Proof.* By definition of synthesis semantics, $P_\varphi \subseteq P$ and by its idempotence property $\Phi(\varphi, \mathcal{W}_{(X_0,P_\varphi)}, 0) = P_\varphi$. Thus, by definition of flowpipe satisfaction we immediately get that $\mathcal{W}_{(X_0,P_\varphi)} \models \varphi$. □

Since $\mathcal{W}_{(X_0,P)}$ over-approximates $\Xi(X_0, P_\varphi)$ with $P_\varphi \subseteq P$, where $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0) = P_\varphi$, if a formula $\varphi$ is satisfied by $\mathcal{W}_{(X_0,P)}$, then it is satisfied also by $\Xi(X_0, P_\varphi)$. In the following theorem we show that the parameters generated by the synthesis semantics are also valid for the exact trajectories of a dynamical system.

**Lemma 13.** *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 $\varphi$.

- ($\sigma$) 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 $P_\sigma$ 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 $P_{\varphi_1}$ and $P_{\varphi_2}$, 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 $\varphi_1$ and $\varphi_2$, we attain the thesis.

□

Finally, we demonstrate that the results of Theorem 13, i.e., the validity of the parameters generated by the synthesis semantics with respect to single trajectories, can be extend to flows of trajectories.

**Theorem 6.** *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 13. □

In summary, we proved that all the trajectories starting in $X_0$ and having parameters in $P_\varphi$, where $P_\varphi$ is the result of the synthesis semantics of $\varphi$ on the flowpipe $\mathcal{W}_{(X_0,P)}$, satisfy the formula $\varphi$. This means that the synthesis semantics can be used to produce under-approximations of the parameter synthesis problem (see Definition 16). In the following section we will determine where the under-approximations are introduced.

### A Conservative Solution

Computing $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0) = P_\varphi$ we obtain a conservative under-approximation of the solution $P^*_\varphi$ of the original parameter synthesis problem (see Definition 16), meaning that every parameter from the synthesized set is valid for the given dynamical system. There are three main sources of approximations introduced by our flow-based approach.

First, in the semantics of disjunctions over flows we impose that for a parameter $\mathbf{p}$ either all the states $\mathbf{x}$ satisfy a disjunct or they all satisfy the other one. In a more general setting, this would correspond to approximating a property of the form $\forall y(p(y) \vee q(y))$ with $\forall y(p(y)) \vee \forall y(q(y))$.

Second, $\mathcal{W}_{(X_0,P)}$ is an over-approximation of $\Xi(X_0, P)$, since each state in $\mathcal{W}_{(X_0,P)}(t)$ can reach any other state in $\mathcal{W}_{(X_0,P)}(t+1)$. This affects the semantics of the until operator. In fact, we need to require that there exists a time instant $t'$ at which $\varphi_2$ is satisfied by all the states.

Third, $\mathcal{W}_{(X_0,P)}$ is incrementally computed using the parameter set $P$, so we propagate points of $\mathcal{W}_{(X_0,P)}(t)$ that are not necessarily reachable if we replace $P$ with a proper subset.

In the next section we present an algorithm that computes an under-approximation of the solution of the original parameter synthesis problem. Our algorithm is inspired by the synthesis semantics $\Phi(\varphi, \mathcal{W}_{(X_0,P)}, 0)$, but it produces a better approximation since the parameter set is dynamically refined and the above-mentioned third source of approximation is avoided as at each step only the refined parameter set is used to determine the next set of states.

### 5.2.1 (Un)Decidability

Before proceeding with the synthesis algorithm, we briefly study the decidability of the parameter synthesis problem. In general the parameter synthesis problem (as formulated in this work) is *undecidable*. This is a direct consequence of the undecidability of the reachability problem (see Section 3.4). If we are not able to compute all the trajectories of a dynamical system, then we are not even able to distinguish the ones that satisfy a specification from the ones that do not, and hence we can not separate valid from invalid parameter values.

However we can restrict the problem to some decidable classes of dynamical systems. Similarly to reachability case (see Section 3.4), we can consider polynomial discrete-time dynamical systems and semialgebraic sets, and show that we can write a first-order formula that encodes the solution of a parameter synthesis problem.

#### Semialgebraic Parameter Synthesis

Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a discrete-time parametric polynomial dynamical system, $X_0 \subseteq \mathcal{X}$ be the set of initial conditions, and $P \subseteq \mathcal{P}$ be the parameter set. Let us assume that for each time instant $i \in \mathbb{N}$ the reached set $X_i$ and the synthesized parameter set $P_i$ are semialgebraic sets, i.e., we can define two first-order formulas over the reals $X_i[\mathbf{x}]$ and $P_i[\mathbf{p}]$ whose solutions represent $X_i$ and $P_i$, respectively.

The state-parameter space at time zero can be described by the formula $XP_0[\mathbf{x}, \mathbf{p}] = X_0[\mathbf{x}] \wedge P_0[\mathbf{p}]$, where the semantics of $P_0[\mathbf{p}]$ corresponds to the initial parameter set $P$. Suppose that at time $i \in \mathbb{N}$ we are interested in refining the parameter set with respect to a predicate of the from $p_i(\mathbf{x}) \leq 0$ whit $p_i : \mathbb{R}^n \to \mathbb{R}$ polynomial. This constraint can be described by the formula $C_i[\mathbf{x}] = p_i(\mathbf{x}) \leq 0$.

The synthesis of a parameter set at time $i \in \mathbb{N}$ with respect to the predicate $p_{i+1}(\mathbf{x}) \leq 0$ can be characterized by the following first-order formula:

$$P_{i+1}[\mathbf{p}] = P_i[\mathbf{p}] \wedge \forall \mathbf{x}(XP_i[\mathbf{x}, \mathbf{p}] \implies (C_{i+1}[\mathbf{f}(\mathbf{x}, \mathbf{p})])) \tag{5.15}$$

A parameter satisfies the formula $P_{i+1}[\mathbf{p}]$ if it belongs to the previous set $P_i[\mathbf{p}]$ and it is such that, together with its correspondent state $\mathbf{x}$ retrieved from the subformula $XP_i[\mathbf{x}, \mathbf{p}]$, under the system dynamics $\mathbf{f}(\mathbf{x}, \mathbf{p})$, it leads to a point that satisfies the constraint $C_{i+1}[\mathbf{x}]$.

Once that the refined parameter set $P_{i+1}[\mathbf{p}]$ has been established, we can compute the next reachable set with the formula:

$$XP_{i+1}[\mathbf{x}', \mathbf{p}'] = \exists \mathbf{x}(XP_i[\mathbf{x}, \mathbf{p}'] \wedge P_{i+1}[\mathbf{p}'] \wedge \mathbf{x}' = \mathbf{f}(\mathbf{x}, \mathbf{p}')). \tag{5.16}$$

Note that with this formula we couple each synthesized parameter $\mathbf{p}'$ from $P_{i+1}$ with its correspondent state $\mathbf{x}$ and we compute the next reachable state $\mathbf{x}'$ keeping the relation with $\mathbf{p}'$.

**Example 25.** *Consider the dynamical system of Example 7 whose dynamics are* $\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{p}) = \mathbf{p}\mathbf{x}_k^2$*, with set of initial conditions* $X_0 = [0.1, 0.2]$*, parameter set* $P = [-1.0, 1.0]$*, and constraint* $p_1(\mathbf{x}) = \mathbf{x} - 0.03 \leq 0$.

*The parameter and state-parameter sets at time zero can be described by the formulas:*

$$\begin{aligned} P_0[\mathbf{x}_0, \mathbf{p}] &\equiv -1.0 \leq \mathbf{p} \leq 1.0 \\ XP_0[\mathbf{x}_0, \mathbf{p}] &\equiv 0.1 \leq \mathbf{x}_0 \leq 0.2 \wedge -1.0 \leq \mathbf{p} \leq 1.0 \end{aligned} \tag{5.17}$$

*and the constraint by the formula:*

$$C_1[\mathbf{x}] \equiv \mathbf{x} - 0.03 \leq 0. \tag{5.18}$$

*With this setting, the valid parameter set within a single step can be characterized by the formula:*

$$\begin{aligned} P_1[\mathbf{p}] &\equiv P_0[\mathbf{p}] \wedge \forall \mathbf{x}(XP_0[\mathbf{x}, \mathbf{p}] \implies C_1[\mathbf{f}(\mathbf{x}, \mathbf{p})]) \\ &\equiv (-\mathbf{p} - 1 \leq 0 \wedge \mathbf{p} - 1 \leq 0) \wedge \forall \mathbf{x}(((-10\mathbf{x} + 1 \leq 0 \wedge 5\mathbf{x} - 1 \leq 0) \wedge \\ &\quad (-\mathbf{p} - 1 \leq 0 \wedge \mathbf{p} - 1 \leq 0)) \implies 100\mathbf{p}\mathbf{x}^2 - 3 \leq 0) \end{aligned} \tag{5.19}$$

*whose equivalent version produced by the quantifier eliminator is:*

$$P_1[\mathbf{p}] \equiv \mathbf{p} + 1 \geq 0 \wedge 4\mathbf{p} - 3 \leq 0. \tag{5.20}$$

*Thus, the refined parameter set is the set of points satisfying $P_1[\mathbf{p}]$, i.e, $P_s = [-1, 3/4]$.*

Looking at the big picture, we can decompose an STL formula with bounded time horizon in a finite series of checks on basic predicates (see the discussion between the correspondence of bounded discrete-time STL and LTL in Section 5.2). This means that the parameter synthesis, in this specific case, can be reduced to the satisfiability of first-order formulas, a problem which is well known for being decidable [179, 47].

Unluckily, decidability does not imply feasibility. The quantifier elimination procedure, needed to establish the truth value of a first-oder formula over the reals, is doubly exponential in the degree of the functions appearing in the formula [102] (or in the number of quantifier alternations, depending on the algorithm). Thus, this approach shows that the problem is decidable and exactly solvable but it also points out that it is computationally costly. Hence, if we want to find valid parameter sets for nontrivial systems, we need to develop an alternative technique.

## 5.3 Synthesis Algorithm

An intuitive way to solve our synthesis problem is to compute the reachable sets $X_j$ at each time instant $j$, up to a fixed time instant $k \in \mathbb{N}$. Then, by examining the sets from time $k$ back to time 0, we can derive the conditions on the parameters for the satisfaction of the temporal property, as in the standard bottom-up monitoring approaches [143]. In other terms, we could backwardly compute the synthesis semantics of a formula with respect to a precomputed flowpipe. However, while in monitoring only a single trace at a time is considered and that trace is already given, in our parameter synthesis problem the reachable set needs to be approximated (since exact reachability computation is often impossible). When approximations are used, a major drawback of

such a backward procedure is that the error depends on the size of the parameter set and it is accumulated step after step. The more spurious behaviors are included in the computed set, the more restricted the parameter set is. In order to gain accuracy, it is important to be able to remove, as early as possible, the parameter values that make the system violate the property. This is the reason why we opt for a *forward* procedure.

### 5.3.1 Overall Structure

We describe our top-down forward algorithm ParaSynth$(X, P, \varphi)$ (Algorithm 10), that, for a given discrete-time dynamical system $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$, takes in input a set of states $X \subseteq \mathcal{X}$, a set of parameters $P \subseteq \mathcal{P}$, and an STL formula $\varphi$ in positive normal form, and produces a refinement $P_\varphi \subseteq P$ through a series of recursions driven by the structure of $\varphi$. At each step, we let the system evolve under the parameter set synthesized up to that step.

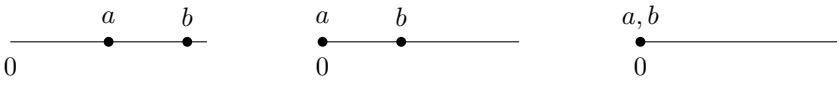
**Algorithm 10** Parameter synthesis.

```
1: function ParaSynth(X, P, φ)
2:     if φ = σ then                                              ▷ Predicate
3:         return RefPredicate(X, P, σ)
4:     end if
5:     if φ = φ1 ∧ φ2 then                                        ▷ Conjunction
6:         return ParaSynth(X, P, φ1) ∩ ParaSynth(X, P, φ2)
7:     end if
8:     if φ = φ1 ∨ φ2 then                                        ▷ Disjunction
9:         return ParaSynth(X, P, φ1) ∪ ParaSynth(X, P, φ2)
10:    end if
11:    if φ = φ1 U_I φ2 then                                      ▷ Until
12:        return UntilSynth(X, P, φ1 U_I φ2)
13:    end if
14: end function
```

The algorithm is structured in four main blocks, one for each type of STL subformula: predicate, conjunction, disjunction, and until. It uses two basic functions ReachStep and RefPredicate:

- ReachStep$(X, P)$ receives in input a set of states $X \subseteq \mathcal{X}$ and a parameter set $P \subseteq \mathcal{P}$, and computes the image $\mathbf{f}(X, P)$;
- RefPredicate$(X, P, \sigma)$ receives in input a set of states $X \subseteq \mathcal{X}$, a parameter set $P \subseteq \mathcal{P}$, and an STL predicate $\sigma$, and computes the largest subset $P_\sigma \subseteq P$ such that all the states in $\mathbf{f}(X, P_\sigma)$ (computable by ReachStep) satisfy $\sigma$, that is $P_\sigma = \{\mathbf{p} \mid \mathbf{p} \in P \ \wedge \ \forall \mathbf{x} \in X, \sigma(\mathbf{f}(\mathbf{x}, \mathbf{p})) \text{ is true}\}$. We call the computation of RefPredicate a *basic refinement*.

The concretization of these two functions depends on the considered dynamical system and set representation. In Section 5.4 we will discuss a possible implementation in the case of polynomial dynamical systems and polytopes.




(a) $a > 0$, $b > 0$ (b) $a = 0$, $b > 0$ (c) $a = 0$, $b = 0$

Figure 5.4: Until interval cases.

The overall structure of the algorithm is the following. The base case of ParaSynth is when the formula $\varphi$ is a predicate $\sigma$ (Line 2). In this case, the algorithm calls the function RefPredicate$(X, P, \sigma)$ that refines the parameter set $P$ with respect to the predicate $\sigma$ and returns the result.

If $\varphi$ is the conjunction of two formulas $\varphi_1 \wedge \varphi_2$ (Line 5), the algorithm, with two recursive calls, produces two refined parameter sets $P_{\varphi_1}$ and $P_{\varphi_2}$, with respect to the subformulas $\varphi_1$ and $\varphi_2$, and returns the intersection $P_{\varphi_1} \cap P_{\varphi_2}$. Similarly, if $\varphi$ is a disjunction $\varphi_1 \vee \varphi_2$ (Line 8), the algorithm returns the union $P_{\varphi_1} \cup P_{\varphi_2}$.

The until case $\varphi_1 U_I \varphi_2$ (Line 11) is slightly more complex and requires a specific procedure.

### 5.3.2 Until Synthesis

The function UntilSynth$(X, P, \varphi_1 U_{[a,b]} \varphi_2)$ (Algorithm 11) 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 (see Figure 5.4a);
2. $a = 0$ and $b > 0$: the interval starts at time 0 and ends somewhere else (see Figure 5.4b);
3. $a = 0$ and $b = 0$: the interval coincides with the single time instant 0 (see Figure 5.4c).

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

Before defining the algorithm, it is important 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 $\varphi_1$ always holds and $\varphi_2$ holds at several time instants inside $[a, b]$. Here the number of basic refinements that $\varphi_1 U_{[a,b]} \varphi_2$ requires is exactly equal to the number of time instants in which $\varphi_2$ 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 analyze the three cases constituting the structure of UntilSynth (Algorithm 11):

1. $a > 0$ and $b > 0$: the until formula is satisfied if $\varphi_1$ holds until $\varphi_2$ is true inside the interval $[a, b]$. We first refine the parameters at time 0 over $\varphi_1$, obtaining the subset $P_{\varphi_1}$ (Line 3). If $P_{\varphi_1}$ is empty, the until formula cannot be satisfied,

**Algorithm 11** Until synthesis.

```
1: function UntilSynth(X, P, φ1 U_[a,b] φ2)
2:     if a > 0 and b > 0 then                                  ▷ Outside interval
3:         P_φ1 ← ParaSynth(X, P, φ1)                           ▷ Check φ1
4:         if P_φ1 = ∅ then
5:             return ∅
6:         else
7:             X' ← ReachStep(X, P_φ1)
8:             return UntilSynth(X', P_φ1, φ1 U_[a−1,b−1] φ2)
9:         end if
10:    end if
11:    if a = 0 and b > 0 then                                  ▷ In interval
12:        P_φ1 ← ParaSynth(X, P, φ1)                           ▷ Check φ1
13:        P_φ2 ← ParaSynth(X, P, φ2)                           ▷ Check φ2
14:        if P_φ1 = ∅ then
15:            return P_φ2                                      ▷ Until unsatisfied
16:        else
17:            X' ← ReachStep(X, P_φ1)
18:            return P_φ2 ∪ UntilSynth(X', P_φ1, φ1 U_[a,b−1] φ2)
19:        end if
20:    end if
21:    if a = 0 and b = 0 then                                  ▷ Base
22:        return ParaSynth(X, P, φ2)
23:    end if
24: end function
```

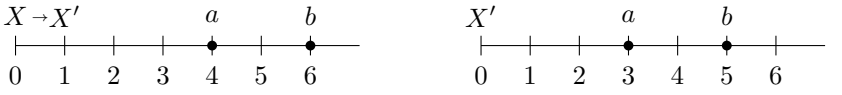
and the algorithm returns the empty set. If $P_{\varphi_1}$ is not empty, the algorithm performs a reachability step using the valid parameter set $P_{\varphi_1}$ to produce the new set $X'$ (Line 7). Now the algorithm proceeds with the recursive call UntilSynth$(X', P_{\varphi_1}, \varphi_1 U_{[a,b]-1} \varphi_2)$ (Line 8). 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 by 1. Hence, the next refinement will be computed always at time 0 (see Figures 5.5a and 5.5b). Roughly speaking, instead of proceeding towards the interval, we pull the interval to us;

2. $a = 0$ and $b > 0$: there are two ways to satisfy the until formula:

   (a) $\varphi_2$ is satisfied right now at time 0;

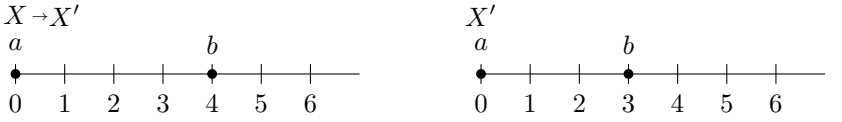
   (b) $\varphi_1$ holds until $\varphi_2$ is satisfied before the time instant $b$.

   In the first case, we need to refine the parameter set with respect to $\varphi_2$. If the resulting $P_{\varphi_2}$ is not empty, it is a valid parameter set that satisfies the until formula. In the second case, the algorithm refines with respect to $\varphi_1$ and checks whether the result $P_{\varphi_1}$ is empty (Line 12). If so, the until formula cannot be satisfied in the future. Hence the algorithm returns the refined set $P_{\varphi_2}$ previously computed. If $P_{\varphi_1}$ is not empty, the procedure performs a reachability step under




(a) Before: $[a, b]$. (b) After: $[a, b] - 1$.

Figure 5.5: Refinement for the case $a > 0$ and $b > 0$.




(a) Before: $[0, b]$. (b) After: $[0, b - 1]$.

Figure 5.6: Refinement for the case $a = 0$ and $b > 0$.

the refined parameter set $P_{\varphi_1}$, obtaining the new set $X'$. Similarly to the previous case, we execute a step forward by shortening the interval by one (Line 18) (see Figures 5.6a and 5.6b). The procedure then returns the union of $P_{\varphi_2}$ 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 $\varphi_2$ and return $P_{\varphi_2}$.

**Example 26.** *We illustrate a schematic execution of* ParaSynth *for the formula* $\varphi = (\varphi_1 \vee \varphi_2)U_{[1,2]}(\varphi_3 \wedge \varphi_4)$.

*With the initial call* ParaSynth$(X, P, (\varphi_1 \vee \varphi_2)U_{[1,2]}(\varphi_3 \wedge \varphi_4))$ *the algorithm enters in the until section and calls* UntilSynth. *The first synthesis is performed inside the* $(a > 0$ *and* $b > 0)$ *case with respect to the subformula* $\varphi_1 \vee \varphi_2$. ParaSynth *computes the refined sets* $P_0^{\varphi_1}$ *and* $P_0^{\varphi_2}$ *with respect to* $\varphi_1$ *and* $\varphi_2$, *and returns the union* $P_0^{\varphi_1 \vee \varphi_2} = P_0^{\varphi_1} \cup P_0^{\varphi_2}$. *Back to the until synthesis, supposing that* $P_0^{\varphi_1 \vee \varphi_2}$ *is not empty, the algorithm computes* $X_0$ *through a reachability step from* $X$ *under the parameter set* $P_0^{\varphi_1 \vee \varphi_2}$, *and calling itself with the updated reachability set, the refined parameter set, and the shifted until interval, i.e.,* UntilSynth$(X_0, P_0^{\varphi_1 \vee \varphi_2}, (\varphi_1 \vee \varphi_2)U_{[0,1]}(\varphi_3 \wedge \varphi_4))$.

*At this point* UntilSynth *enters the* $(a = 0$ *and* $b > 0)$ *section. It first refines with respect to* $(\varphi_3 \wedge \varphi_4)$, *trying to find the first final solution. To do so, it calls* ParaSynth$(X_0, P_0^{\varphi_1 \vee \varphi_2}, \varphi_3 \wedge \varphi_4)$ *that produces the parameter set* $P_1^{\varphi_3 \wedge \varphi_4} = P_1^{\varphi_3} \cap P_1^{\varphi_4}$, *result of the intersection of the two refinements of* $P_0^{\varphi_1 \vee \varphi_2}$ *with respect to* $\varphi_3$ *and* $\varphi_4$. *This set* $P_1^{\varphi_3 \wedge \varphi_4}$, *if not empty, represents the first valid parameter set.*

*Trying to find other possible solutions, the algorithm proceeds by computing the parameter set* $P_1^{\varphi_1 \vee \varphi_2}$ *through the refinement of* $P_0^{\varphi_1 \vee \varphi_2}$ *with respect to* $\varphi_1 \vee \varphi_2$ *and performing a reachability step to the new set* $X_1''$. *It then calls itself reducing the until interval to* $[0, 0]$. *This is the base case* $(a = 0$ *and* $b = 0)$*: the algorithm refines with respect to* $\varphi_3 \wedge \varphi_4$ *and returns the refined parameter set* $P_2^{\varphi_3 \wedge \varphi_4}$.

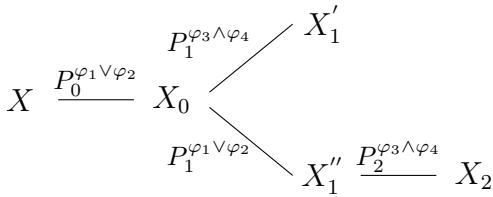



Figure 5.7: ParaSynth execution on $(\varphi_1 \vee \varphi_2) U_{[1,2]} (\varphi_3 \wedge \varphi_4)$.

*The synthesis process is shown in Figure 5.7. The figure depicts the series of refinements and reachable sets that lead to the final result* $P_2^{\varphi_3 \wedge \varphi_4} \cup P_1^{\varphi_3 \wedge \varphi_4}$.

### 5.3.3 Shortcuts

The parameter synthesis algorithm that we defined is structured on the subformulas belonging to the syntax of STL in positive normal form, that are predicate, conjunction, disjunction, and until. These subformulas can be used to define other common operators, such as *true* $\top$, *false* $\bot$, *release* $\varphi_1 R_I \varphi_2$, *eventually* $F_I \varphi$, and *always* $G_I \varphi$. The latter can be seen as shortcuts for:

$$\begin{aligned} \top &:= p_m(\mathbf{x}) \leq 0 \\ \bot &:= p_p(\mathbf{x}) \leq 0 \\ \varphi_1 R_{[a,b]} \varphi_2 &:= (\varphi_2 U_{[a,b]} (\varphi_1 \wedge \varphi_2)) \vee (\varphi_2 U_{[b+1,b+1]} \top) \\ F_{[a,b]} \varphi &:= \top U_{[a,b]} \varphi \\ G_{[a,b]} \varphi &:= \top U_{[a,a]} (\bot R_{[0,b-a]} \varphi) \end{aligned} \tag{5.21}$$

where $p_m(\mathbf{x})$ and $p_p(\mathbf{x})$ are a negative and positive constant function, respectively (e.g., $p_m(\mathbf{x}) = -1, p_p(\mathbf{x}) = 1$), $\varphi, \varphi_1$, and $\varphi_2$ are STL formulas in positive normal form, and $[a, b] \subset \mathbb{N}$.

Intuitively, $\top$ is the logical operator satisfied by any trace, while $\bot$ is the logical operator that cannot be satisfied by any trace. The release $\varphi_1 R_{[a,b]} \varphi_2$ is satisfied if and only if $\varphi_2$ remains true until $\varphi_1$ once becomes true in the interval $[a, b]$. If $\varphi_1$ never becomes true, $\varphi_2$ must hold until the end of the interval $[a, b]$. $F_{[a,b]} \varphi$ says that $\varphi$ eventually becomes true in $[a, b]$, while $G_{[a,b]}$ requires $\varphi$ to be always true in $[a, b]$.

Note that the definition of the derived operators is possible only if the intervals appearing in the temporal operators are bounded, condition satisfied by STL formulas in positive normal form. Moreover, our syntax conversion is slightly more complicated than the usual one since we want to avoid the negation.

The fact that we can express the presented derived operators as combinations of the basic ones implies that our synthesis algorithm can already deal with $\top$, $\bot$, $\varphi_1 R_I \varphi_2$, $F_I \varphi$, and $G_I \varphi$.

However, it is worth to point out that a syntactic translation, even if correct, might be computationally inefficient. For instance, the execution of ParaSynth on the simple false $\bot$ operator, translated into $p_p(\mathbf{x}) \leq 0$, requires the execution of RefPredicate

and returns the empty set. This machinery can be avoided by just enriching the main algorithm with the false case $\bot$ and returning directly the empty set. Similarly, the true case $\top$ can return the parameter set $P$ given in input without doing any modification.

More interesting are the cases of the release, eventually, and always, that can be treated similarly to the until (see Section 5.3.2). We now show how to compute the refinement with respect to the always temporal operator. The release and eventually operators can be obtained with the same scheme and small changes.

### Always Synthesis

Let $G_{[a,b]}\varphi$ be the formula that we want to treat. As for the until case, we define a special routine AlwaysSynth (see Algorithm 12) structured in three cases, depending on the values of the interval $[a, b]$ that defines $G_{[a,b]}$. We recall that in order for $G_{[a,b]}\varphi$ to hold, $\varphi$ must hold in each time instant between $a$ and $b$. The three cases are the following:

1. $a > 0$ and $b > 0$: no refinement is required, hence the algorithm makes a step towards $[a, b]$ without affecting the current parameter set $P$ (Line 3);

2. $a = 0$ and $b > 0$: $\varphi$ must hold. The algorithm, through a recursive call, refines $P$ with respect to $\varphi$ obtaining the refined parameter set $P_\varphi \subseteq P$. Then, it proceeds making a step towards $b$ and continuing the synthesis from the reached set with the obtained parameter set $P_\varphi$ (Lines 7-9);

3. $a = 0$ and $b = 0$: this is the base case of the recursive calls. It suffices to refine $P$ with respect to $\varphi$ and return $P_\varphi$ (Line 12).

**Algorithm 12** Always synthesis.

```
1:  function AlwaysSynth(X, P, G_{[a,b]}φ)
2:      if a > 0 and b > 0 then                          ▷ Outside interval
3:          X' ← ReachStep(X, P)
4:          return AlwaysSynth(X', P, G_{[a−1,b−1]}φ)
5:      end if
6:      if a = 0 and b > 0 then                          ▷ In interval
7:          P_φ ← ParaSynth(X, P, φ)                     ▷ Check φ
8:          X' ← ReachStep(X, P_φ)
9:          return AlwaysSynth(X', P_φ, G_{[a,b−1]}φ)
10:     end if
11:     if a = 0 and b = 0 then                          ▷ Base
12:         return ParaSynth(X, P, φ)
13:     end if
14: end function
```

The definition of a dedicated procedure (as done for the always formulas) instead of relying exclusively on syntactic translations, improves the efficiency of the whole synthesis process, since it avoids several calls to subroutines required by the subformulas introduced by the syntactic translation. Notice that the synthesis results provided by

the dedicated procedures and the shortcut translations are exactly the same (assuming that the shortcuts are correctly implemented).

### 5.3.4 Correctness and Complexity

In the following, we prove the correctness of the synthesis algorithm ParaSynth (Algorithm 10) and determine its computational complexity.

#### Correctness

The following theorem establishes the correctness of ParaSynth (Algorithm 10) with respect to the synthesis semantics and the definition of formula satisfaction under the assumption that the function ReachStep$(X, P)$ exactly computes the image $\mathbf{f}(X, P)$ and the function RefPredicate$(X, P, \sigma)$ provides the largest valid parameter set for the predicate $\sigma$. However, it is not hard to see that, for Theorem 7 to hold, it suffices to provide an over-approximation of the image $\mathbf{f}(X, P)$ and an under-approximation of the valid parameter set.

**Theorem 7.** *If* ParaSynth$(X, P, \varphi)$ *returns* $P_\varphi$*, then* $\Phi(\varphi, \mathcal{W}_{(X,P)}, 0) \subseteq P_\varphi$ *and* $\Xi(X, P_\varphi) \models \varphi$*.*

*Proof.* We proceed by structural induction on $\varphi$.

- $(\sigma)$ The function ParaSynth$(X, P, \sigma)$ returns the result provided by RefPredicate$(X, P, \sigma)$. This last, by assumption of correctness of RefPredicate, returns the set $P_\sigma = \{\mathbf{p} \mid \mathbf{p} \in P \text{ and } \forall \mathbf{x} \in X\, \sigma(\mathbf{f}(\mathbf{x}, \mathbf{p})) \text{ is true}\}$. Hence, $P_\sigma = \Phi(\sigma, \mathcal{W}_{(X,P_\sigma)}, 0)$ and by Theorem 6 we get the thesis.

- $(\varphi_1 \wedge \varphi_2)$ By definition ParaSynth$(X, P, \varphi_1 \wedge \varphi_2) = P_{\varphi_1} \cap P_{\varphi_2}$ with $P_{\varphi_1} =$ ParaSynth$(X, P, \varphi_1)$ and $P_{\varphi_2} =$ ParaSynth$(X, P, \varphi_2)$. By inductive hypothesis it holds that $\Phi(\varphi_1, \mathcal{W}_{(X,P)}, 0) \subseteq P_{\varphi_1}$ and $\Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0) \subseteq P_{\varphi_2}$, that implies $P_{\varphi_1} \cap P_{\varphi_2} \supseteq \Phi(\varphi_1, \mathcal{W}_{(X,P)}, 0) \cap \Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0) = \Phi(\varphi_1 \wedge \varphi_2, \mathcal{W}_{(X,P)}, 0)$.

  Moreover, since for all $\mathbf{p} \in P_{\varphi_1} \cap P_{\varphi_2}$ and $\mathbf{x} \in X$, it holds that $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_1$ and $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_2$, we have that $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_1 \wedge \varphi_2$, and then the thesis $\Xi(X, P_{\varphi_1} \cap P_{\varphi_2}) \models \varphi_1 \wedge \varphi_2$.

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

- $(\varphi_1 U_{[a,b]} \varphi_2)$ We proceed by induction on the length of the interval $[a, b]$.

  - $(a = 0$ and $b = 0)$ In this case $P_\varphi = P_{\varphi_2}$, where $P_{\varphi_2} =$ ParaSynth$(X, P, \varphi_2)$. Hence, by inductive hypothesis on $\varphi_2$, we have $\Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0) \subseteq P_{\varphi_2}$. Moreover, it is easy to see that $\Phi(\varphi_1 U_{[0,0]} \varphi_2, \mathcal{W}_{(X,P)}, 0) = \Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0)$. So $\Phi(\varphi_1 U_{[0,0]} \varphi_2, \mathcal{W}_{(X,P)}, 0) \subseteq P_\varphi$.

    Let $\mathbf{p} \in P_\varphi$ and $\mathbf{x} \in X$. Since $\mathbf{p} \in P_{\varphi_2}$ by inductive hypothesis we have $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_2$. Hence, $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_1 U_{[0,0]} \varphi_2$ and $\Xi(X, P_\varphi) \models \varphi_1 U_{[0,0]} \varphi_2$.
  - $(a = 0$ and $b > 0)$ By inductive hypothesis the thesis holds for any $b' < b$. By definition of ParaSynth we have that $P_\varphi = P_{\varphi_2} \cup P'$, where $P_{\varphi_2} =$ ParaSynth$(X, P, \varphi_2)$ and $P' =$ ParaSynth$(X', P_{\varphi_1}, \varphi_1 U_{[0,b-1]} \varphi_2)$ with

$P_{\varphi_1} = \textsc{ParaSynth}(X, P, \varphi_1)$ and $X' = \textsc{ReachStep}(X, P_{\varphi_1})$. By inductive hypothesis we have that both the following inclusions hold:

$$\Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0) \subseteq P_{\varphi_2}$$
$$\Phi(\varphi_1 U_{[0,b-1]} \varphi_2, \mathcal{W}_{(X',P_{\varphi_1})}, 0) \subseteq P'.$$

To show the inclusion of the thesis, we now show that if a parameter $\mathbf{p}$ belongs to $\Phi(\varphi_1 U_{[0,b]} \varphi_2, \mathcal{W}_{(X,P)}, 0)$, then it belongs also to $P_{\varphi_2} \cup P'$. Since $\mathbf{p} \in \Phi(\varphi_1 U_{[0,b]} \varphi_2, \mathcal{W}_{(X,P)}, 0)$, by definition of synthesis semantics, it must hold that either $\mathbf{p} \in \Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0)$, or there exists $t' \in [0, b]$ such that $\mathbf{p} \in \Phi(\varphi_2, \mathcal{W}_{(X,P)}, t')$ and for all $t'' \in [0, t']$, $\mathbf{p} \in \Phi(\varphi_1, \mathcal{W}_{(X,P)}, t'')$. In the first case, since $\mathbf{p} \in \Phi(\varphi_2, \mathcal{W}_{(X,P)}, 0)$, by inductive hypothesis we have that $\mathbf{p} \in P_{\varphi_2}$. In the second case, we have that:

$$\mathbf{p} \in \bigcap_{t'=0}^{t''} \Phi(\varphi_1, \mathcal{W}_{(X,P)}, t') \cap \Phi(\varphi_2, \mathcal{W}_{(X,P)}, t'')$$
$$\mathbf{p} \in \Phi(\varphi_1, \mathcal{W}_{(X,P)}, 0) \cap \bigcap_{t'=1}^{t''} \Phi(\varphi_1, \mathcal{W}_{(X,P)}, t') \cap \Phi(\varphi_2, \mathcal{W}_{(X,P)}, t'')$$
$$\mathbf{p} \in \Phi(\varphi_1, \mathcal{W}_{(X,P)}, 0) \cap \Phi(\varphi_1 U_{[1,b]} \varphi_2, \mathcal{W}_{(X,P)}, 1)$$
$$\mathbf{p} \in \Phi(\varphi_1, \mathcal{W}_{(X,P)}, 0) \cap \Phi(\varphi_1 U_{[0,b-1]} \varphi_2, \mathcal{W}_{(\mathbf{f}(X,P),P)}, 0)$$

Since $X' = \mathbf{f}(X, P_{\varphi_1}) \subseteq \mathbf{f}(X, P)$, it holds the inclusion:

$$\Phi(\varphi_1 U_{[0,b-1]} \varphi_2, \mathcal{W}_{(\mathbf{f}(X,P),P)}, 0) \subseteq \Phi(\varphi_1 U_{[0,b-1]} \varphi_2, \mathcal{W}_{(X',P_{\varphi_1})}, 0) \subseteq P'$$

from which, by inductive hypothesis, we deduce that $\mathbf{p} \in P'$.

Let $\mathbf{p} \in P_\varphi$ and $\mathbf{x} \in X$. If $\mathbf{p} \in P_{\varphi_2}$, then we immediately get the thesis by inductive hypothesis. If $\mathbf{p} \in P'$, then $\mathbf{p} \in P_{\varphi_1}$ and $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_1$. Hence, since $P'$ is returned by $\textsc{ParaSynth}(X', P_{\varphi_1}, \varphi_1 U_{[0,b-1]} \varphi_2)$, by inductive hypothesis we have that $\xi^{\mathbf{p}}_{\xi^{\mathbf{p}}_{\mathbf{x}}(1)} \models \varphi_1 U_{[0,b-1]} \varphi_2$. Hence, $\xi^{\mathbf{p}}_{\mathbf{x}} \models \varphi_1 U_{[0,b]} \varphi_2$.

– ($a > 0$ and $b > 0$) Identical to the second part of the previous case.

□

### Computational Complexity

Let us now determine the computational complexity of ParaSynth (Algorithm 10). As far as the computational complexity of our algorithm is concerned, let us refer to RefPredicate, ReachStep, $\cup$, and $\cap$ as symbolic operations. If we have a formula without until operators our procedure performs a number of symbolic operations that is linear in the length of the formula. In the case of formulas with possibly nested until operators, in the worst case, we could perform an exponential number of symbolic operations with respect to the minimum between the length of the formula and its time horizon.

Let us consider the case of formulas using only predicates and $U_{[0,1]}$ operators. Let the length of a formulas $\varphi$ be defined as the maximum number of nested until operators. For a formula $\varphi_1 U_{[0,1]} \varphi_2$ having length $m$ and horizon $k$ our recursive procedure has a recursive complexity equation in term of symbolic operations of the form:

$$T(m,k) = \begin{cases} \Theta(1) & \text{if } m = 1 \text{ or } k = 0 \\ T(m_2,k_2) + T(m_1,k_1) + T(m,k-1) + \Theta(1) & \text{otherwise} \end{cases} \tag{5.22}$$

where $m_i$ and $k_i$ are the length and horizon of $\varphi_i$.

In the worst case we could have $m_2 = m-2$ and $k_2 = k-1$. In this case we obtain $T(m,k) \geq 2T(m-2,k-1) + \Theta(1)$, which tells us that in the worst case $T(m,k) = \Omega(2^{min(m,k)})$ (number of symbolic operations). If we were interested in monitoring a formula over a finite set of traces, we could have reduced such complexity to a polynomial one (see, e.g., [143, 71]). As we already pointed out, since we are interested in refining sets of parameters and to avoid rough approximations we do not use a precomputed set of traces, we do not see an easy way to reduce this complexity. However, it is important to notice that the worst case complexity occurs only in very pathological cases, which are not typical in real case studies.

## 5.4 The Polynomial Case

In the previous section we presented ParaSynth (Algorithm 10), an abstract algorithm that synthesizes a parameter set under which the behaviors of a dynamical system satisfy a given specification. A concrete application of ParaSynth depends on the ability of representing parameter sets and implementing the function ReachStep for computing the system evolution, and the function RefPredicate for refining the parameter set.

We now propose a concretization of the algorithm for polynomial discrete-time dynamical systems with polytopic parameter sets. However, notice that the abstract algorithm exposed in Section 5.3 can be used for more general systems as long as an implementation of the required procedures are provided. An example might be continuous-time piecewise-linear dynamical systems, for which reachability techniques and tools have been developed [6, 87].

From now on, we work with polynomial discrete-time dynamical systems of the form $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$, where $\mathcal{X} = \mathbb{R}^n$, $\mathcal{P} = \mathbb{R}^m$, and $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ is a polynomial. We also assume that the threated sets of parameters $P \subset \mathcal{P}$ are polytopes (see Section 4.1.1). For this class of dynamical systems and set representation we will show how to:

1. Manipulate parameter sets, namely, computing their intersections and symbolically representing their unions;

2. Implement the RechStep procedure, that over-approximates the single step evolution of the dynamical system;

3. Implement the RefPredicate procedure, that refines a parameter set so that the evolution of the system satisfies a given predicate within a single step.

### 5.4.1 Parameter Set Representation and Manipulation

The first implementation aspect we consider is the representation and manipulation of sets of parameters. A polytope (see Section 4.1.1) is the simplest form that we use to represent a parameter set. With the notation $P = A\mathbf{p} \leq \mathbf{b}$ we mean that the parameter set $P$ corresponds to the solution of the linear system $A\mathbf{p} \leq \mathbf{b}$. More complex parameter sets can be obtained by the intersection and the union of several basic convex polytopes.

Let $P_1 = A_1\mathbf{p} \leq \mathbf{b}_1$ and $P_2 = A_2\mathbf{p} \leq \mathbf{b}_2$ be two convex polytopes. It is not difficult to see that the intersection $P_1 \cap P_2$ is the convex polytope that corresponds to $P_1 \cap P_2 = A\mathbf{p} \leq \mathbf{b}$, where:

$$A = \begin{pmatrix} A_1 \\ A_2 \end{pmatrix} \text{ and } b = \begin{pmatrix} b_1 \\ b_2 \end{pmatrix}. \tag{5.23}$$

Less trivial is the union of two polytopes that might lead to a nonconvex set and consequently the representation through a linear system may not be possible. For this reason we symbolically represent the union of two polytopes $P_1$ and $P_2$ by simply keeping the list of the corresponding linear systems. Formally, with a slight abuse of notation, $P_1 \cup P_2$ is represented as $P_1 \cup P_2 = \{A_1\mathbf{p} \leq b_1, A_2\mathbf{p} \leq b_2\}$.

It is now important to note that, thanks to the distributivity property, every parameter set can be represented as the union of intersections. If a parameter set $P$ is in the form:

$$P = \bigcup_{i=1}^{n} \bigcap_{j=1}^{m_i} P_{i,j} = (P_{1,1} \cap \ldots \cap P_{1,m_1}) \cup \ldots \cup (P_{n,1} \cap \ldots \cap P_{n,m_n}) \tag{5.24}$$

then it is said to be in *union normal form*. This form is suitable for our set representation since the intersections of sets can be collapsed in a unique linear system while the unions can be stored in single list.

**Example 27.** *Let $P_i = A_i\mathbf{p} \leq b_i$ for $i = 1, \ldots, 4$ be parameter sets and $P = P_1 \cap (P_2 \cap (P_3 \cup P_4))$ whose union normal form is:*

$$P_1 \cap \big(P_2 \cap (P_3 \cup P_4)\big) = P_1 \cap \big((P_2 \cap P_3) \cup \big(P_2 \cap P_4\big)\big) = (P_1 \cap P_2 \cap P_3) \cup (P_1 \cap P_2 \cap P_4). \tag{5.25}$$

*Thus $P = \{A'\mathbf{p} \leq b', A''\mathbf{p} \leq b''\}$ where:*

$$A' = \begin{pmatrix} A_1 \\ A_2 \\ A_3 \end{pmatrix}; \begin{pmatrix} b_1 \\ b_2 \\ b_3 \end{pmatrix} = b' \qquad A'' = \begin{pmatrix} A_1 \\ A_2 \\ A_4 \end{pmatrix}; \begin{pmatrix} b_1 \\ b_2 \\ b_4 \end{pmatrix} = b''. \tag{5.26}$$

### 5.4.2 Single Step Evolution

The second important element of an implementation of ParaSynth is the realization of the procedure ReachStep able to compute the single reachability step of the threated dynamical system.

Chapter 4 has been entirely dedicated to the realization of the set image algorithm ReachStep. We have defined several techniques that depend on the representation of

the reachable set. In particular, given a parametric polynomial $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and a polytope $P \subset \mathbb{R}^m$, we have seen how to over-approximate the set $\mathbf{f}(X, P)$ with:

- $X \subset \mathbb{R}^n$ begin a box (see Section 4.3);
- $X \subset \mathbb{R}^n$ begin a parallelotope (see Section 4.4);
- $X \subset \mathbb{R}^n$ begin a symbolic polytope represented by a parallelotope bundle, i.e., a collection of parallelotopes whose intersection generates a polytope (see Section 4.5).

We briefly recall that all the developed methods for determining an over-approximation set $X' \supseteq \mathbf{f}(X, P)$ share similar key steps:

1. Define a transformation $\mathbf{v}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$ that maps the unit box to the set $X$;
2. Consider the composition $\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p})$ and exploit Bernstein coefficients and their properties to bound the directions of $X'$ over $\mathbf{f}(X, P)$ (see Section 4.2);
3. Use the obtained bounds to construct a set $X' \supseteq \mathbf{f}(X, P)$ that over-approximates the image of $\mathbf{f}(X, P)$.

For the detailed descriptions of the mentioned methods, the reader can refer to Chapter 4.

### 5.4.3 Basic Refinement

Finally, we focus on the third and last basic element of ParaSynth, that is the implementation of the function RefPredicate that refines a parameter set so that the given dynamical system satisfies within a single step a given predicate.

We assume that the considered predicates $\sigma = p(\mathbf{x}) \leq 0$ are linear in $\mathbf{x}$ and that all the coefficients $\mathbf{a_i}(\mathbf{p})$ of the system dynamics $\mathbf{f}(\mathbf{x}, \mathbf{p})$ are linear in the parameters $\mathbf{p}$. These assumptions will allow us to recast the refinement problem in terms of a linear program.

Let $\mathcal{D} = (\mathcal{X}, \mathcal{P}, \mathbf{f})$ be a discrete-time dynamical system and $X \subseteq \mathcal{X}$ be a set. The technique we are going to expose can be applied in both the cases when $X$ is a box or a parallelotope. Let $\mathbf{v}(\mathbf{x}) : \mathbb{R}^n \to \mathbb{R}^n$ be the map that transforms the unit box $[0, 1]^n$ to the set $X$. In the case where $X$ is a box, $\mathbf{v}(\mathbf{x})$ can be obtained with the function mapUnitBoxTo (see Section 4.3). Otherwise, in the case where $X$ is a parallelotope, $\mathbf{v}(\mathbf{x})$ corresponds to the generator function $\gamma_U(\mathbf{q}, \boldsymbol{\beta}, \mathbf{x})$ obtainable with the function con2gen (see Section 4.4).

To check whether the system satisfies the predicate $\sigma = p(\mathbf{x}) \leq 0$ we consider the function $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$ and its Bernstein coefficients $B_g = \{\mathbf{b_i}(\mathbf{p}) \mid \mathbf{i} \in I^g\}$. Since the maximum Bernstein coefficients is an upper bound of the maximum of $g(\mathbf{x}, \mathbf{p})$, if we ensure that all the coefficients are smaller-equal than zero, then we are sure that $g(\mathbf{x}, \mathbf{p})$ is smaller-equal than zero too, that corresponds saying that $\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p})$ satisfies the predicate $\sigma$.

Formally, the following is a sufficient condition for the system to satisfy the predicate $\sigma$ after one discrete step starting from the set $X$ with parameter set $P$:

$$\forall \mathbf{b_i}(\mathbf{p}) \in B_g \forall \mathbf{p} \in P, \mathbf{b_i}(\mathbf{p}) \leq 0. \tag{5.27}$$

Note that since $p(\mathbf{x})$ is by assumption a linear function and the parameters $\mathbf{p}$ appear linearly in the dynamics $\mathbf{f}(\mathbf{x}, \mathbf{p})$, the coefficients in the monomial of $g(\mathbf{x}, \mathbf{p})$ remain linear in $\mathbf{p}$. This means that the constraints of Equation 5.27 are linear inequalities over $\mathbf{p}$. This observation allows us to translate the basic refinement problem in the resolution of a system of linear inequalities.

Suppose that the current states of the system are represented by the set $X \subseteq \mathcal{X}$. The refinement of the parameter set $P$, represented by the linear system $P = A\mathbf{p} \leq \mathbf{b}$, with respect to the predicate $\sigma = p(\mathbf{x}) \leq 0$ can be obtained by the following steps:

1. Build the composition $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$ and compute the set of Bernstein coefficients $B_g = \{\mathbf{b_i}(\mathbf{p}) \mid \mathbf{i} \in I^g\}$;

2. Build a new linear system appending all the new constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$ to $P$, for $\mathbf{b_i}(\mathbf{p}) \in B_g$. We will refer with $P_\sigma$ to the new linear system;

3. Check whether $P_\sigma$ has solutions, i.e., it is not empty.

By adding new constraints to the parameter set $P$ we eliminate the values of $P$ such that the system does not satisfy the predicate $\sigma$. This is de facto the *point in which the parameters refinement happens.*

When we construct the refined set $P_\sigma$ by adding the constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$ to the set $P$, in order to control the growth of $P_\sigma$ in terms of constraints, it is a good idea to check the eventual redundancy of the new constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$ with respect to the set $P$. A trivial way to implement this check, is to verify whether $\mathbf{b_i}(\mathbf{p}) \leq 0$ is already present in $P$ or if there are constraints with the same direction of $\mathbf{b_i}(\mathbf{p}) \leq 0$ but dominating offsets. More sophisticated and complete (but more expensive) approaches consist in the resolution of families of liner programs [122, 136, 180] or ad-hoc symbolical methods [185, 114].

Once that $P_\sigma$ has been constructed, we check if it has solutions. The emptiness test can be efficiently carried out by a single linear program. If the refined parameter set is not empty, then the set $X' = \mathbf{f}(X, P_\sigma)$ satisfies the predicate $\sigma$.

If $P_\sigma$ has no solutions, then either there are no parameter values in $P_\sigma$ such that the system can satisfy the predicate $\sigma$, or the over-approximation of the set $X$ introduced by Bernstein coefficients is not enough accurate to produce a valid parameter set.

**Example 28.** *Let us consider the dynamics of the discrete-time SIR epidemic system of Example 6 with state variables and parameters grouped in the vectors* $\mathbf{x} = (s, i, r)$ *and* $\mathbf{p} = (\beta, \gamma)$*:*

$$\mathbf{f}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f}_s(\mathbf{x}, \mathbf{p}) = s - \beta si \\ \mathbf{f}_i(\mathbf{x}, \mathbf{p}) = i + \beta si - \gamma i \\ \mathbf{f}_r(\mathbf{x}, \mathbf{p}) = r + \gamma i \end{pmatrix}. \tag{5.28}$$

*Let* $X = \langle D, \mathbf{c} \rangle = [0.80, 0.85] \times [0.15, 0.20] \times [0, 0]$ *and* $P = [0.35, 0.36] \times [0.05, 0.06]$*. Let us consider the predicate* $\sigma = p(\mathbf{x}) \leq 0 = i - 0.248 \leq 0$*, i.e., we want to synthesize the parameters of* $P$ *so that the number of infected individuals from* $X$ *is below* $0.248$ *within a single reachability step.*

*From Example 14, we know that the map* $\mathbf{v}$ *that transforms the unit box to* $X$ *is:*

$$\mathbf{v}(\mathbf{x}) = \begin{pmatrix} 0.85 - 0.80 & 0 & 0 \\ 0 & 0.20 - 0.15 & 0 \\ 0 & 0 & 0 - 0 \end{pmatrix} \begin{pmatrix} s \\ i \\ r \end{pmatrix} + \begin{pmatrix} 0.80 \\ 0.15 \\ 0 \end{pmatrix} \tag{5.29}$$

*whose composition with the system dynamics leads to the function:*

$$\mathbf{h}(\mathbf{x}, \mathbf{p}) = \begin{pmatrix} \mathbf{f_s}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = (0.05s + 0.80) - \beta(0.05s + 0.80)(0.05i + 0.15) \\ \mathbf{f_i}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = (1 - \gamma)(0.05i + 0.15) + \beta(0.05s + 0.80)(0.05i + 0.15) \\ \mathbf{f_r}(\mathbf{v}(\mathbf{x}), \mathbf{p}) = \gamma(0.05i + 0.15) \end{pmatrix} \tag{5.30}$$

*The composition* $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{h}(\mathbf{x}, \mathbf{p}))$ *of the constraint function* $p(\mathbf{x})$ *with* $\mathbf{h}(\mathbf{x}, \mathbf{p})$ *is:*

$$g(\mathbf{x}, \mathbf{p}) = (1 - \gamma)(0.05i + 0.15) + \beta(0.05s + 0.80)(0.05i + 0.15) - 0.248 \tag{5.31}$$

*whose Bernstein coefficients are:*

$$B_g = \{\frac{3}{25}\beta - \frac{3}{20}\gamma - \frac{49}{500}, \frac{4}{25}\beta - \frac{1}{5}\gamma - \frac{6}{125}, \frac{51}{400}\beta - \frac{3}{20}\gamma - \frac{49}{500}, \frac{17}{100}\beta - \frac{1}{5}\gamma - \frac{6}{125}\}. \tag{5.32}$$

Constructing and appending the constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$, for $\mathbf{b_i}(\mathbf{p}) \in B_g$, to the linear system of $P$ we obtain the new linear system:

$$\begin{pmatrix} 1 & 0 \\ -1 & 0 \\ 0 & 1 \\ 0 & -1 \\ 3/25 & -3/20 \\ 4/25 & -1/5 \\ 51/400 & -3/20 \\ 17/100 & -1/5 \end{pmatrix} \begin{pmatrix} \beta \\ \gamma \end{pmatrix} \leq \begin{pmatrix} 0.36 \\ -0.35 \\ 0.06 \\ -0.05 \\ 49/500 \\ 6/125 \\ 49/500 \\ 6/125 \end{pmatrix} \tag{5.33}$$

whose solution constitutes the refined valid set $P_\sigma \subseteq P$. The graphical representations of the original parameter set $P$ (in white) and the refined one $P_\sigma$ (in gray) are shown in Figure 5.8.

**Example 29.** *We now show a schematic execution of* ParaSynth *on the base of the implementation choices discussed in this section. Let us consider the formula* $(\phi_1 \vee \phi_2)U_{[1,2]}(\phi_3 \wedge \phi_4)$ *of Example 26.The algorithm starts with* ParaSynth $(X, P, (\phi_1 \vee \phi_2)U_{[1,2]}(\phi_3 \wedge \phi_4))$ *that invokes* UntilSynth *that enters in the case* $(a > 0$ *and* $b > 0)$ *and performs the first refinement of* $P$ *with respect to the subformula* $\phi_1 \vee \phi_2$ *by calling* ParaSynth. *This function synthesizes* $P$ *by refining with respect to both the predicates* $\phi_1$ *and* $\phi_2$ *and merging the partial results. The result is the set* $P^0_{\phi_1 \vee \phi_2} = \{P^0_{\phi_1}, P^0_{\phi_2}\}$. *At this point,* UntilSynth *opens a branch from* $X$ *for each computed refinement. We denote by* $X^0_1$ *the set reached from* $X$ *under* $P^0_{\phi_1}$ *and by* $X^0_2$ *the set reached from* $X$ *under* $P^0_{\phi_2}$. UntilSynth *proceeds with two recursive calls,* UntilSynth$(X^0_1, P^0_{\phi_1}, (\phi_1 \vee$

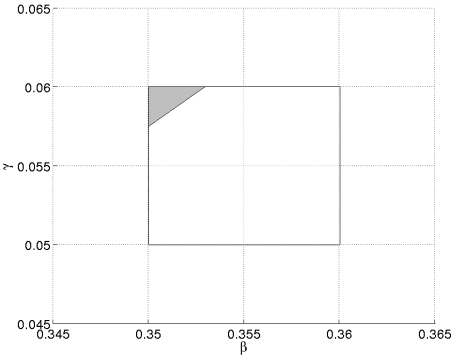



Figure 5.8: Result of a basic refinements. Original (in white) and refined (in gray) parameter sets.

$\phi_2)U_{[0,1]}(\phi_3 \wedge \phi_4))$ *and* UntilSynth$(X_2^0, P_{\phi_2}^0, (\phi_1 \vee \phi_2)U_{[0,1]}(\phi_3 \wedge \phi_4))$*. We now consider the first recursive call. In this phase,* UntilSynth *is in the case* $(a = 0$ *and* $b > 0)$*. First, trying to satisfy the whole until, the algorithm refines the set* $P_{\phi_1}^0$ *with respect to* $\phi_3$ *and* $\phi_4$*. This is done by calling* ParaSynth $(X^0, P_{\phi_1}^0, \phi_3 \wedge \phi_4)$*. We denote the result with* $P_{\phi_3 \wedge \phi_4}^{1,1} = P_{\phi_3}^1 \cap P_{\phi_4}^1$*. If not empty,* $P_{\phi_3 \wedge \phi_4}^{1,1}$ *is the first valid parameter set. Trying to find other solutions,* UntilSynth *refines also with respect to* $\phi_1 \vee \phi_2$ *opening two new branches, one for each disjunct. Each branch corresponds to a recursive call of the form* ParaSynth $(X_2^1, P_{\phi_1}^{1,1}, (\phi_1 \vee \phi_2)U_{[0,0]}(\phi_3 \wedge \phi_4))$*. The synthesis process is shown in Figure 5.9.*

### A More Sophisticated Basic Refinement

At the moment the refinement algorithm halts whenever the parameter set becomes empty or it has analyzed the whole specification. If we are able to distinguish whether the parameter set is truly empty (i.e., there are no valid parameter values) or there is an accumulation of error introduced by Bernstein coefficients, then we can either definitely halt the synthesis, or we can try to improve the precision of the refinement, providing more accurate results. This section is dedicated to the detection of false empty parameter sets.

The idea is to exploit the sharpness property (see Lemma 2) that states that the values of a polynomial $\pi(\mathbf{x}, \mathbf{p})$ are exactly matched by the Bernstein coefficients $\mathbf{b_i}(\mathbf{p})$ when $\mathbf{i}$ is a vertex of the box $[0, \mathbf{d}_1] \times \cdots \times [0, \mathbf{d}_n]$, with $\mathbf{d}_j$ degree of the $j$-th variable of $\pi(\mathbf{x}, \mathbf{p})$, for $j \in \{1, \ldots, n\}$.

Let us consider the refined parameter set $P_\sigma$, that is the intersection of the previous parameter set $P$ with the constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$, where $\mathbf{b_i}(\mathbf{p}) \in B_g$ are the Bernstein coefficients of the function $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$. If $P_\sigma$ is empty and the multi-index $\bar{\mathbf{i}}$ of maximum coefficients $\mathbf{b}_{\bar{\mathbf{i}}}(\mathbf{p})$ lies on one of the vertices of the box $[0, \mathbf{d}_1] \times \cdots \times [0, \mathbf{d}_n]$, by the sharpness property we deduce that the exact maximum of $g(\mathbf{x}, \mathbf{p})$ is equal to $\mathbf{b}_{\bar{\mathbf{i}}}(\mathbf{p})$, and thus there are no parameter values such that the predicate $\sigma$ can be satisfied. This implies that at this point the parameter set is truly empty, and thus we can halt the

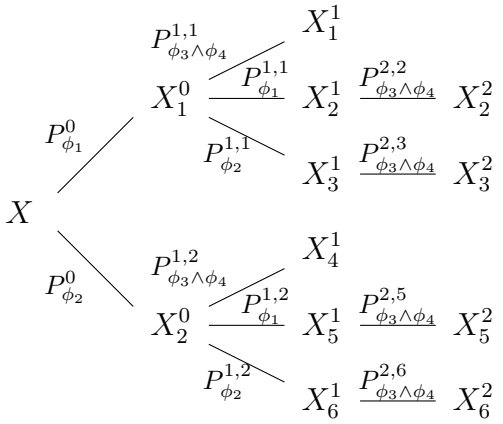



Figure 5.9: ParaSynth execution on $(\phi_1 \vee \phi_2)U_{[1,2]}(\phi_3 \wedge \phi_4)$ with polynomials and polytopes.

synthesis procedure.

On the other hand, if the sharpness property does not hold, i.e., the multi-index $\bar{\mathbf{i}}$ lies on a non-vertex point of the box $[0, \mathbf{d}_1] \times \cdots \times [0, \mathbf{d}_n]$, the parameter set $P_\sigma$ might be empty because of the over-approximation introduced by Bernstein coefficients, and not because of the unfeasibility of the predicate. Thus, we can subdivide the computed Bernstein coefficients (using the subdivision methods exposed in Section 4.6.3) in the hope of improving the upper bound of $g(\mathbf{x}, \mathbf{p})$ provided by its Bernstein coefficients and obtaining a nonempty parameter set $P_\sigma$. The subdivision can be repeated until a nonempty set is found or a tolerance precision threshold is reached.

We now define a more sophisticated version of the RefPredicate algorithm that takes into account these observations. RefPredicate (Algorithm 13) implements the refinement of a parameter set $P$ with respect to a predicate $\sigma = p(\mathbf{x}) \leq 0$ involving the sharpness property and the subdivision of Bernstein coefficients. It receives in input a box or parallelotope set $X \subset \mathbb{R}^n$, a polytopic parameter set $P \subset \mathbb{R}^m$, an STL predicate $\sigma = p(\mathbf{x}) \leq 0$, and produces in output a polytopic set $P_\sigma \subseteq P$ such that for all $\mathbf{p} \in P_\sigma$ and $\mathbf{x} \in X$, $\mathbf{f}(\mathbf{x}, \mathbf{p})$ satisfies $\sigma$.

The algorithm begins by computing the map $\mathbf{v}(\mathbf{x})$ that transforms the unit box to the current set $X$ (Line 2). The map $\mathbf{v}(\mathbf{x})$ can be computed with the function mapUnitBoxTo in the case where $X$ is a box (see Section 4.3) or with con2gen when $X$ is a parallelotope (see Section 4.4). Then it defines the composition $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$ (Line 3) and computes the set of Bernstein coefficients $B_g$ (Line 4). The first attempt of refinement is done by the function BuildLS that merges the current parameter set $P$ with the collection of constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$, where $\mathbf{b_i}(\mathbf{p}) \in B_g$, and generates the set $P_\sigma$ (Line 5). BuildLS can also contain some subroutines to detect and eliminate redundant constraints from the refined parameter set $P_\sigma$ (see the discussion in Section 5.4.3).

If $P_\sigma$ is nonempty, it means that a valid refinement has been found, and the algorithm returns it. Otherwise the algorithm checks whether the sharpness property holds.

If not, there is no valid refinement, hence the empty set is returned. Otherwise, the algorithm tries to refine the constraints $\mathbf{b_i}(\mathbf{p}) \leq 0$ by subdividing the Bernstein coefficients (Line 9). This is realized with a **while** loop that iterates until a precision threshold on the coefficients is reached (Line 8) or a nonempty parameter set is found (Line 12).

**Algorithm 13** Parameter set refinement with respect to a predicate with subdivision.

```
1:  function RefPredicate(X, P, σ)          ▷ X box or parallelotope, σ = p(x) ≤ 0
2:      v(x) ←mapUnitBoxTo(X)                ▷ or con2gen(X) for X parallelotope
3:      g(x, p) ← p(f(v(x), p))
4:      B_g ←BernCoeffs(g)
5:      P_σ ←BuildLS(P, B_g)                 ▷ First refinement
6:      if P_σ = ∅ then
7:          if ¬Sharpness(B_g) then          ▷ Check whether sharpness holds
8:              while LargeEnough(B_g) do
9:                  B_g ←Subdivide(B_g)      ▷ Increase coefficients precision
10:                 P_σ ←BuildLS(P, B_g)     ▷ Preciser refinement
11:                 if P_σ ≠ ∅ then
12:                     return P_σ
13:                 end if
14:             end while
15:         end if
16:     end if
17:     return P_σ
18: end function
```

This procedure allows one to attain preciser parameter sets and establish when there are no valid refinements. The precision of the refinements, checked by the function LargeEnough, is left as a parameter tunable by the user.

### Symbolic Refinement

In conclusion, we discuss a further improvement in the basic refinement that concerns the computation of the Bernstein coefficients of the composition $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$. Similarly to the set image case (see Section 4.6.2), also here we can think of symbolically precomputing the Bernstein coefficients for all the predicates of a specification rather than computing them all the times in which the synthesis algorithm needs a basic refinement.

With the purpose of speeding up the parameter synthesis, we define the procedure InitPredCoefficients (Algorithm 14) that receives in input an STL formula $\varphi$ in positive normal form, that is the specification imposed on our system, and initializes a data structure $\Sigma$ that contains the Bernstein coefficients related to each predicate $\sigma$ appearing in $\varphi$. If $p$ distinct predicates appear in the formula $\varphi$, then $\Sigma$ will contain $p$ sets of Bernstein coefficients.

The algorithm recursively works on the specification $\varphi$. If $\varphi$ is already a predicate $\sigma = p(\mathbf{x}) \leq 0$, the algorithm builds the composition $g(\mathbf{x}, \mathbf{p}) = p(\mathbf{f}(\mathbf{v}(\mathbf{x}), \mathbf{p}))$, computes its Bernstein coefficients, and stores them in the data structure $\Sigma$ at the correspondent

entry $\Sigma_\sigma$ (Line 4). If $\varphi$ is not a predicate, the algorithm extracts the two subformulas $\varphi_1$ and $\varphi_2$ that compose $\varphi$ and recursively calls itself on $\varphi_1$ and $\varphi_2$. Note that since $\varphi$ is assumed to be in positive normal form, it will always have two subformulas (since it can be a conjunction, a disjunction, or an until).

**Algorithm 14** Symbolic precomputation of predicates coefficients.

```
1:  function InitPredCoefficients(φ)                          ▷ φ STL formula
2:      if φ = σ then                                         ▷ Predicate σ = p(x) ≤ 0
3:          g(x, p) ← p(f(v(x), p))
4:          Σ_σ ←BernCoeffs(g)                                ▷ Compute Bernstein coefficients
5:      else
6:          [φ_1, φ_2] ←SubFormulas(φ)                        ▷ Fetch subformulas
7:          InitPredCoefficients(φ_1)                         ▷ and recursively initialize their predicates
8:          InitPredCoefficients(φ_2)
9:      end if
10:     return P_σ
11: end function
```

This precomputation can be invoked before the execution of the synthesis algorithm ParaSynth and the data structure $\Sigma$ can be exploited by the basic refinement RefPredicate. Let $t, p \in \mathbb{N}$ be the time horizon and the number of predicates appearing in $\varphi$, respectively. With this astuteness, Bernstein coefficients are computed $\Theta(p)$ times against the $\Theta(p \cdot t)$ times of the nonsymbolic approach.

# 6

# Tool and Experimental Results

This chapter is devoted to the description and evaluation of the tool *Sapo* that implements the reachability and parameter synthesis techniques previously described. The developed techniques and tool are assessed on some polynomial dynamical systems for both reachability analysis and parameter synthesis. We will analyze artificial, biological, and mechanical models.

The chapter is organized in two main sections. In the first (Section 6.1), we will describe our tool, focusing on its structure, highlighting its main features, and our implementation choices. In the second (Section 6.2), we will exercise and evaluate Sapo on some polynomial parametric discrete-time dynamical systems.

## 6.1 Architecture

We begin with the description of the tool Sapo that gathers the implementations of the reachability and parameter synthesis techniques previously described. The overall architecture of Sapo is summarized in Figure 6.1.

The main module of the tool is `Sapo` (Section 6.1.1), that handles the computation of reachable sets and the synthesis of parameters. Another important module is the `Base Converter` (Section 6.1.4) that allows us to compute the Bernstein coefficients of a given polynomial. The `Bundle` and `Parallelotope` modules (Sections 6.1.5 and 6.1.6) are used to represent sets of states of dynamical systems, while `Linear System` and `Linear System Set` (Sections 6.1.8 and 6.1.9) are used for representing sets of parameters. With `STL` and `Model` (Sections 6.1.2 and 6.1.3) the user can formalize a specification and instantiate a polynomial dynamical system. Finally, `Vars Generator` (Section 6.1.7) is a utility module that can be used to automatize the declaration of the variables necessary to instantiate a parallelotope.

Our tool is entirely implemented in C++. It relies on the libraries GiNaC (GiNaC is not a CAS) [19] and GLPK (GNU Linear Programming Kit).[1] GiNaC is used for the symbolic representation of polynomials. In the following, by "symbolic expression" we

[1] `http://www.gnu.org/software/glpk/glpk.html`

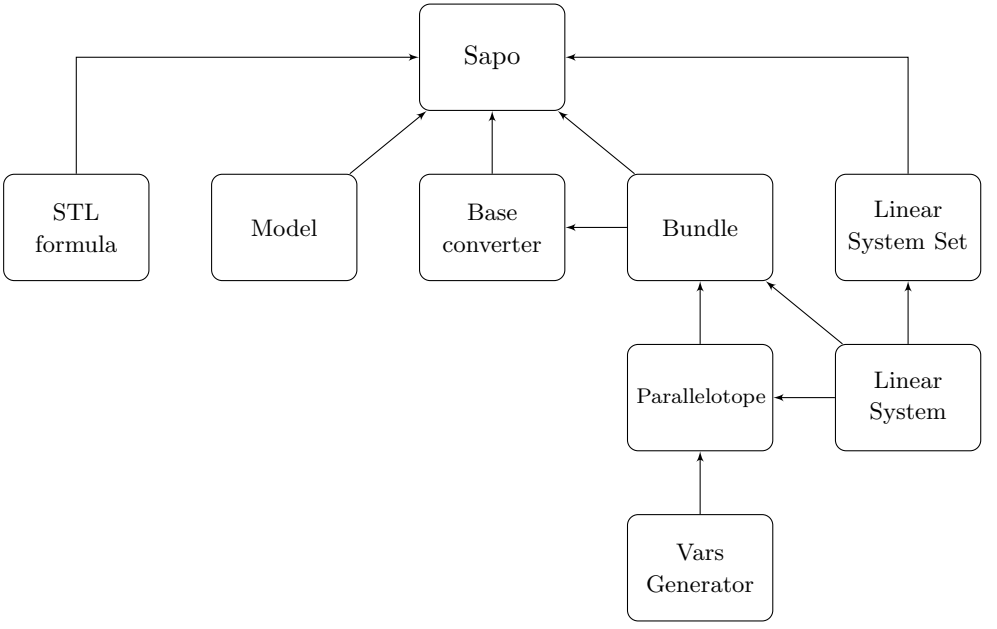



Figure 6.1: Tool architecture.

mean a formula declared using the data-type provided by GiNaC. We will also use GiNaC to extract coefficients from polynomials, perform symbolic operations on formulas, and substitute variables with expressions in our symbolic formulas. The library GLPK is used to solve linear programs, i.e., to optimize linear functions over a system of linear inequalities. It will be useful to bound directions of parallelotopes, canonize bundles, but also for establishing the emptiness of polytopic sets of parameters.

The source code of our tool can be freely downloaded from `https://github.com/tommasodreossi/parasynth`.

### 6.1.1 Sapo

`Sapo` is the main class responsible for the computation of the reachable sets and the synthesis of parameters of polynomial dynamical systems. The class constructor receives in input an object of the class `Model` that describes the dynamical system under study together with a collection of options. The options are represented by a structure named `sapo_opt` through which it is possible to specify:

- `trans` (Integer in $\{0, 1\}$): the kind of transformation to be applied to the bundle (0 for OFO, 1 for AFO; see Section 4.5.1);
- `decomp` (Integer): number of decompositions to be applied to the bundle after the transformation (see Section 4.5.1);
- `alpha` (Double in $[0, 1]$): (optional) weight in the decomposition of the bundle (see Section 4.5.1);

- `plot` (String): (optional) name of the file where to print the reachability set;
- `verbose` (Boolean): (optional) display all the information.

`Sapo` contains two important functions `reach` and `synthesize` that allow the user to analyze the reachable set or synthesize the parameters of the given model. In both cases the set of initial conditions has to be specified as a `Bundle`. Note that this choice allows the user to work also with single boxes or parallelotopes, since it is possible to declare a bundle composed by a single set. We recall that the reachability feature supports arbitrary bundles, while the parameter synthesis can be performed only on single boxes or parallelotopes, i.e., bundles of cardinality one.

The function `reach` receives in input an initial set described by a `Bundle` and a maximum number of steps $k$, and returns a vector of `Bundle`s that represents the computed flowpipe. The flowpipe is calculated iterating a series of bundle transformations. `reach` supports also parametric reachability. In this case, it is necessary to provide to `reach` also the polytopic parameter set as an object of the class `Linear System`.

The function `synthesize` is responsible for the synthesis of the parameters. It receives in input an initial set described by a `Bundle`, a set of parameters in form of `Linear System Set`, and an `STL` formula. It produces in output a refined set of parameters described by the class `Linear System Set`. This function is implemented following the structure of ParaSynth (Algorithm 10) described in Section 5.3.1.

It is worth to mention that `Sapo` is equipped with the two private data structures `reachControlPts` and `synthControlPts` that store the symbolic control points necessary to compute the reachable sets and to synthesize valid parameter sets, respectively. `reachControlPts` and `synthControlPts` are containers that store elements formed by a combination of a key and a mapped value. The key is a vector of integers with the indices of the directions used to build a parallelotope $X$. The mapped value is a pair containing the generator function of $X$ and the symbolic Bernstein coefficients either computed in combination with the systems dynamics in the `reachControlPts` case, or with the system dynamics and the constraint of an `STL` predicate in the `synthControlPts` case. Whenever `Sapo` needs to compute the Bernstein coefficients for a specific parallelotope, it first looks for them in `reachControlPts` or `synthControlPts`. If the coefficients are not present, `Sapo` computes them using the class `BaseConverter` and stores them in the relative data structure. Otherwise, it just retrieves the coefficients and uses them for computing a reachable set or synthesizing a parameter set. This machinery, explained in detail in Section 5.4.3, allows the tool to compute the Bernstein coefficients only once for a specific combination of parallelotope, direction to bound, or constraint to impose, instead of computing the coefficients at each reachability or synthesis step. This choice sensibly improves the performances of our tool

### 6.1.2 STL

`STL` is the class used to instantiate an STL formula and formalize the specification that the user wants to impose on the dynamical system. `STL` is the base class for the derived classes `Atom`, `Conjunction`, `Disjunction`, `Until`, `Eventually`, and `Always` (not reported in Figure 6.1 for space reasons). All these classes have functions to extract subformulas and temporal interval information. The constraints appearing in `Atom` are

symbolic expressions and are associated with a key specified by the user. The role of the key is to uniquely identify an atomic formula and associate it to the relative collection of Bernstein coefficients (for more details, see Section 6.1.1).

### 6.1.3 Model

This class is used to describe a dynamical system. Its constructor receives in input the lists of symbolic expressions describing the dynamics of the dynamical system under study, the variables appearing in the dynamics, and any of its parameters.

### 6.1.4 Base Converter

`Base Converter` is an important class that converts the power basis of a given polynomial into another basis. In our case, we are interested in the Bernstein basis and in particular in the Bernstein coefficients. The class constructor receives in input a polynomial and a list of variables. The variables are necessary to differentiate the system state variables from the parameters appearing in the polynomial.

The two most important functions are `getBernCoeffs` and `getBernCoeffsMatrix`. The first computes the Bernstein coefficients using the classical iterative method, as described in Equations 4.13 and 4.20 in Section 4.2. The second implements our improved matrix method defined in Section 4.6.1. By default, the tool is configured to use the matrix method. The function `getBernCoeffsMatrix` uses the functions `n2t`, `t2n`, and `transp` (described in Section 4.6.1) to compute the fast transposition of a multi-dimensional matrix.

### 6.1.5 Bundle

This class is used to represent and handle bundles of parallelotopes. Bundles are stored using the data structure defined in Section 4.5.1. The most important functions belonging to this class are `getPolytope`, `transform`, `decompose`, and `canonize`.

`getPolytope` returns a `Linear System` that describes the symbolic polytope represented by the bundle. The system of inequalities is built by collecting all the directions and offsets that compose the parallelotopes stored in the bundle.

The function `transform` receives in input a collection of transforming functions (in our case, the dynamics of the dynamical system), a map of Bernstein control points, and an option that specifies the kind of transformation to apply (OFO or AFO, see Section 4.5.1). The transformation is done going through all the parallelotopes of the bundle and bounding them either with their own directions (in the OFO case), or with all the directions of the bundle (in the AFO case). Whenever a direction has to be bounded over the image of a parallelotope, the function verifies whether the correspondent Bernstein coefficients are already present in the collection given in input. If this is not the case, `transform` calls the `Base Converter`, computes the Bernstein coefficients, and updates the collection. Once that all the bounding offsets of the directions involved in the transformation have been determined, the function `transform` returns a new bundle that over-approximates the image of the symbolic polytope described by the current bundle with respect to the polynomials given in input.

The decomposition of the bundle realized by the function `decompose`, that gets in input a weigh $\alpha$ that regulates the orthogonal proximity and the constraint distances (for details, see Section 4.5.2 ), and a maximum number of iterations. It returns a new bundle were the parallelotope directions are differently organized from the current one. The decomposition is done by generating template matrices and weighting them using the evaluation function defined in Section 4.5.2. At each iteration, `decompose` checks whether the generated template is valid and, if this is the case, it keeps the best templates matrix accordingly with the evaluation function. The evaluation is performed exploiting the functions `maxOffsetDist` and `maxOrthProx` that compute the maximum constraint distance and orthogonal proximity, respectively, of the generated template.

Finally, the function `canonize` returns a canonical bundle (see Section 4.5) with the same directions and templates matrix of the current one, but with the constraints tangent to the polytope described by the threated bundle. The canonization is done by maximizing the directions of the bundle over the `Linear System` that describes the polytope represented by the bundle.

### 6.1.6 Parallelotope

This class, mainly used by the class `Bundle`, represents and handles a single parallelotope. The instantiated parallelotope can be described in both constraint and generator representations (see Section 4.4), depending on whether the constructor of `Parallelotope` is invoked passing as arguments a `Linear System` (in the constraint case) or a vector of versors (in the generator case). In any case, it possible to convert one representation into the other one. The conversion from generators to constraints is done by the function `gen2con`, while the conversion from constraints to generators is carried out by `const2gen`.

### 6.1.7 Variables Generator

`Variables Generator` is a utility class that can be used to generate the variables to build the generator functions of parallelotopes. In high dimensions, the declaration of the variables for base vertices, generators lengths, and versors can be tedious. Hence, this class automates this process generating the necessary variables. The variables for base vertices are declared with names `q` followed by an integer; those for the the generators lengths with `b` followed by an integer; those for the the versors with `u` followed by an integer. Names that belong to this group of variables are reserved for the tool.

### 6.1.8 Linear System Set

This class is used to store and handle sets of `Linear Systems`. It is used by `Sapo` to deal with several parameter sets represented as linear systems. `Linear System Set` supports the intersection and the union of two sets of linear systems.

The intersection, implemented in the function `intersectWith`, intersects each linear system of the current set with each linear system of the set given in input.

The union, implemented in the function `unionWith`, appends the list of linear systems constituting the set given in input, to the list of linear systems of the current set.

### 6.1.9 Linear System

The class `Linear System` allows the tool to store and work with systems of linear inequalities. A linear system $A\mathbf{x} \leq \mathbf{b}$ can be instantiated invoking the class constructor either passing the matrix $A$ and the vector $\mathbf{b}$ as arguments, or a list of symbolic expressions that are the constraints representing the inequalities of the system and the variables that appear in them.

The functions `maxLinearSystem` and `minLinearSystem` offer the possibility of maximizing or minimizing, respectively, an objective linear function over the current linear system. The optimization of the linear function is performed relying on the library GLPK (GNU Linear Programming Kit). Finally, the function `appendLinearSystem` appends a linear system to the current one. This function can be used to intersect two polytopes represented through linear systems.

## 6.2 Case Studies

After describing the overall structure of our tool, we now focus on the application of it to various polynomial dynamical systems. The goal is to evaluate our methods and test the precision and scalability of our implementation. We present six case studies exposed in order of increasing complexity.

We will begin with a two-dimensional test model; then we will analyze three epidemic models (i.e., models that describe the evolution of diseases in populations) composed by three, four, and five variables, respectively; we will also consider a five-dimensional model that characterizes the decision-making process mechanism adopted by a swarm of honeybees; finally, we will focus on a seventeen-dimensional model of a quadcopter drone. We either compute the reachable sets of these models or synthesize their parameters, or, in some cases, we will do both the studies. We will also focus on the scalability of the tool with respect to the complexity of the specifications used to synthesize the parameters. Finally, we mention that some of the following models were born as discrete-time systems, others have been discretized from their original continuous-time versions using the Euler's method.

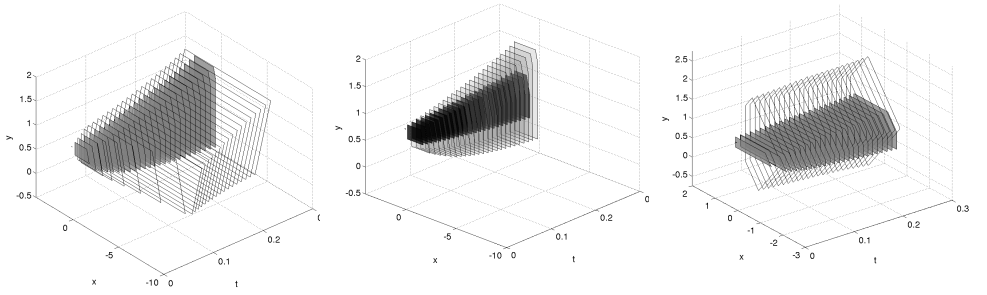
### 6.2.1 Test System

Our first experiment is meant to show the differences between the various bundle transformation methods in the reachability analysis of a dynamical system. We consider an illustrative 2-dimensional system whose dynamics are the following:

$$\begin{aligned} x_{k+1} &= x_k + (0.5x_k^2 - 0.5y_k^2)\Delta \\ y_{k+1} &= y_k + (2x_k y_k)\Delta \end{aligned} \tag{6.1}$$

The chosen directions constituting the bundles are $L_0 = (1, 0), L_1 = (0, 1), L_2 = (-1, 1)$, $L_3 = (1, 1)$, the initial set is a box with $x \in [0.05, 0.1]$ and $y \in [0.99, 1.00]$, and $\Delta = 0.01$.

Figure 6.2 shows the reachable sets computed with the different techniques plotted over time up to 25 steps. Figure 6.2a shows the sets computed using the OFO and AFO transformations (in white and gray, respectively), without the bundle decomposition. In both cases the bundle is composed by two parallelotopes obtained by coupling $L_0$



(a) OFO (white, 0.13s) and AFO (gray, 0.24s) transformations.

(b) AFO transformation without (gray, 0.24s) and with (black, 0.97s) decomposition ($\alpha = 0.5$).

(c) AFO transformation with decomposition. $\alpha = 0$ (gray, 0.95s) and $\alpha = 1$ (white, 0.98s).

Figure 6.2: Reachable set of 2-dimensional test system.

with $L_1$ and $L_2$ with $L_3$, respectively. The picture shows that the AFO transformation is finer than the OFO one. The OFO computation took 0.14s, the AFO 0.21s.

Figure 6.2b compares the sets computed using the AFO transformation with (in black) and without (in gray) the bundle decomposition. In the decomposition function, the parameter $\alpha$ is equal to 0.5 and the number of decompositions randomly generated at each step is 500. The computation without decomposition took 0.22s against 1.94s of the one with decomposition. Note how the black flow is always included in the gray one, meaning that decomposition, applied with the AFO transformation, is finer than non-decomposed AFO and OFO transformations.

Finally, Figure 6.2c depicts the AFO transformation with decomposition for $\alpha = 0$ (in gray) and $\alpha = 1$ (in white), computed in 1.92s and 1.95s (also here 500 decompositions are generated at each step). This experimental evaluation shows how the parameter $\alpha$ affects the reachable set computation. In this case, it is difficult to establish which is the best technique, since there is not a strict inclusion. However, the areas of the sets computed with $\alpha = 0$ are smaller than the ones with $\alpha = 1$.

### 6.2.2 SIR Epidemic Model

As the second case study we consider the classic 3-dimensional SIR epidemic model [123], whose aim is to describe the evolution of a disease in a population. In this model a population of individuals is divided in three compartments: $s$, the healthy individuals *susceptible* to the disease; $i$, the *infected* individuals; $r$ the individuals *removed* from the system (e.g., recovered). Two parameters regulate the evolution of the system variables: $\beta$, the contraction rate and $\gamma$, where $1/\gamma$ is the mean infective period. $\Delta$ is the discretization step. The dynamics of the SIR model are the following:

$$\begin{aligned} s_{k+1} &= s_k - (\beta s_k i_k)\Delta \\ i_{k+1} &= i_k + (\beta s_k i_k - \gamma i_k)\Delta \\ r_{k+1} &= r_k + (\gamma i_k)\Delta \end{aligned} \tag{6.2}$$

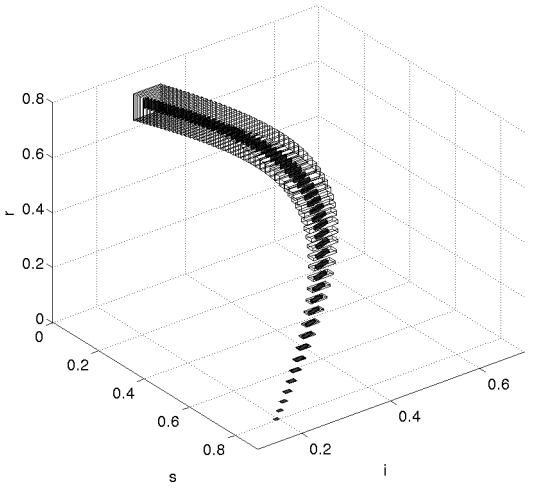



Figure 6.3: Reachable set of 3-dimensional SIR model. Sets have been computed with 1 template/3 directions (in white, 0.12s), and 4 templates/6 directions (in black, 2.83s).

### Reachability

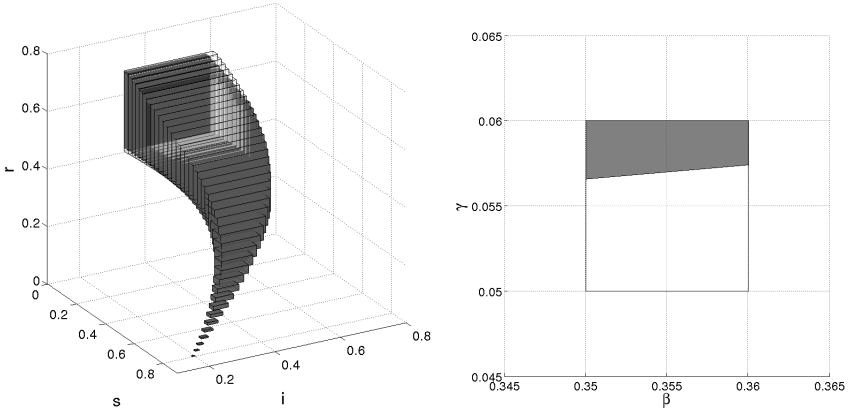
In this study we applied the AFO transformation without bundle decomposition. First, we computed the reachable set using a single axis-aligned template (i.e., a single box). Then, we added 5 directions not aligned with the axis and grouped them in 4 different templates. In both cases we computed the reachable sets up to 60 steps.

Figure 6.3 shows the computed results, i.e., the single template computation (in white) and the four templates one (in black). In both cases the population is normalized and the initial set is the box with $s \in [0.79, 0.80]$, $i \in [0.19, 0.20]$, and $r = 0.00$. The chosen parameter values are $\beta = 0.34$, $\gamma = 0.05$, and $\Delta = 0.5$. The single parallelotope computation required 0.05s against the 1.04s of the 4 parallelotopes one. From the figure we can observe that multiple templates lead to a much finer result: the black flow is always included in the white one.

### Parameter Synthesis

For our first experiment of parameter synthesis we fix an axis-aligned template with initial conditions $s_0 \in [0.79, 0.80], i_0 \in [0.19, 0.20]$, and $r = 0.00$. The initial parameter set is the box with $\beta \in [0.35, 0.36]$ and $\gamma \in [0.05, 0.06]$, while the constraint to be satisfied is $G_{[10,30]}(i \leq 0.682)$, i.e., between time 10 and 30 we want the infected individuals $i$ to be always less than 0.682.

Figure 6.4 shows the results computed by our tool in 0.10s. Figure 6.4a depicts the reachable sets of the system without and with the imposition of constraint (in white and black, respectively), while Figure 6.4b shows the original and synthesized parameter sets (in white and black, respectively). From the figures it is possible to see how the tool produces a subset of the original parameter set that leads to a flowpipe included in the




(a) Reachable sets with (in black, 0.10s) and without (white, 0.08s) constraint.

(b) Original (in white) and synthesized (in black) parameter sets.

Figure 6.4: Reachable and parameter sets of 3-dimensional SIR model with 2 parameters. Specification: $G_{[10,30]}(i \leq 0.682)$.

unconstrained evolution of the system.

For experimental purposes, we now consider the same configurations of the model and tool, but we tighten the specification to $G_{[10,30]}(i \leq 0.681)$. Figure 6.5 depicts the original and synthesized parameter sets (in white and black, respectively) obtained in 0.10s. Comparing this result with the parameter set obtained in the previous case (see Figure 6.4b), it is possible to see how a tighter constraint leads to a smaller parameter set.

Finally, we synthesize the parameters of the SIR model fixing a parallelotopic template with constraints not aligned with the axis. The set of initial conditions is $X_0 = \langle \Lambda, \mathbf{c} \rangle$ where:

$$\Lambda = \begin{pmatrix} 0.7071 & 0.7071 & 0 \\ -0.7071 & 0.7071 & 0 \\ 0 & 0 & 1 \\ -0.7071 & -0.7071 & 0 \\ 0.7071 & -0.7071 & 0 \\ 0 & 0 & -1 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} 0.7071 \\ -0.4172 \\ 0.00 \\ -0.6930 \\ 0.4313 \\ 0 \end{pmatrix}. \tag{6.3}$$

The initial parameter set is the box with $\beta \in [0.18, 0.20]$ and $\gamma \in [0.05, 0.06]$, and the considered specification is $G_{[20,50]}(i \leq 0.50)$. Figure 6.6 shows the results computed by our tool in 0.65s. Note from Figure 6.6a that in this case the computed sets are parallelotopes whose constraints are not aligned with the axis, and that the constrained evolution of the system (in black) is included in the unconstrained one (in white).

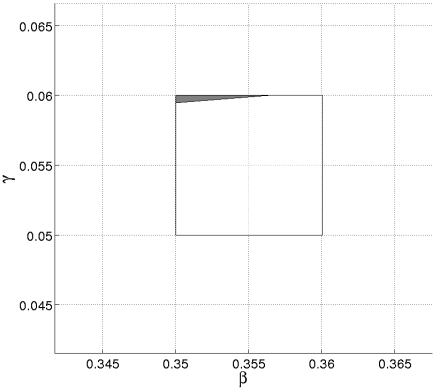
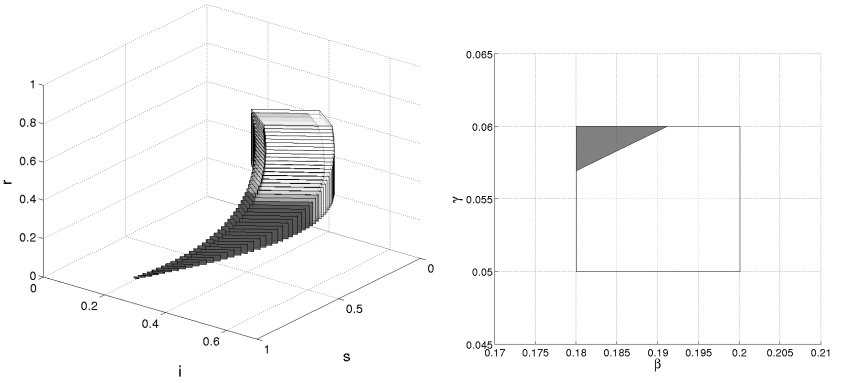



Figure 6.5: Original (in white) and synthesized (in gray) parameter sets of 3-dimensional SIR model with 2 parameters. Specification: $G_{[10,30]}(i \leq 0.681)$.




(a) Reachable sets with (in black, 0.65s) and without (white, 0.53s) constraint.

(b) Original (in white) and synthesized (in black) parameter sets.

Figure 6.6: Reachable and parameter sets of 3-dimensional SIR model with 2 parameters. Specification: $G_{[20,50]}(i \leq 0.50)$.

### 6.2.3 Influenza

We now consider a simplification of the influenza model described in [156]. This model is a 4-dimensional variation of the standard SIR model with two additional controllable parameters: the antiviral treatment $\tau$ and the social distancing $d$, i.e., the infected individuals who receive the antiviral treatment and the number of contacts per unit time between individuals, respectively. Another difference from the SIR model is that in this case the population, composed by $N$ individuals, is grouped in four classes: $s$ is the number of individuals *susceptible* to the influenza and not infected; $i$ are the individuals *infected* by the disease; $t$ are those who are under *treatment*; $r$ are the *removed* patients. The dynamics of the model are defined by the following system of difference equations:

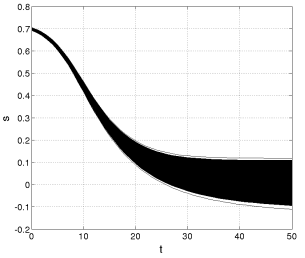
$$\begin{aligned} s_{k+1} &= s_k(1-g_k) \\ i_{k+1} &= (1-\tau)(1-\sigma_1)(1-\delta)i_k + s_k g_k \\ t_{k+1} &= (1-\sigma_2)t_k + \tau(1-\sigma_1)(1-\delta)i_k \\ r_{k+1} &= r_k + \sigma_1(1-\delta)i_k + \sigma_2 t_k \end{aligned}$$

where $g_k = \rho(1-d)(i_k + \varepsilon t_k)/(N_k)$. Variable $g_k$ represents the number of susceptible people that at time $k$ remains so also at time $k+1$. The dynamics of the model involve seven parameters: $\tau$ characterizes the fraction of individuals who gets the treatment; $\sigma_1$ and $\sigma_2$ are the probabilities of recovering individuals due to natural causes and treatment, respectively; $\delta$ is the ratio of induced deaths while $\beta$ is the disease transmission rate; $d$ is the social distancing, that is the number of contacts between individuals by unit time, and $\rho$ is the reduction in transmissibility for the treated compartment. The controllable parameters are the antiviral treatment $\tau$ and the social distancing $d$.

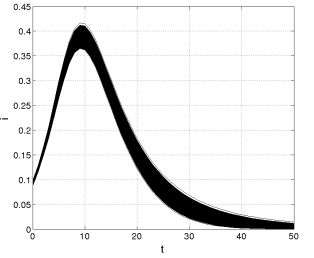
We now simulate and study the model trying to synthesize the two controllable parameters. The recovering probabilities without and with treatment are $\sigma_1 = 1/7$ and $\sigma_2 = 1/5$; the transmissibility coefficient of the treated class is $\epsilon = 0.7$; the mortality and susceptibility rates are fixed to $\delta = 8 \times 10^{-5}$ and $\rho = 0.5$. The controllable parameters, that are the antiviral treatment $\tau$ and the social distancing $d$, can vary inside the initial sets $\tau \in [0.001, 0.002]$ and $d \in [0.005, 0.010]$. The imposed safety constraint is $G_{[0,50]} i \leq 0.4235$, while the set of normalized initial conditions is $X_0 = \langle \Lambda, \mathbf{c} \rangle$, where:

$$\Lambda = \begin{pmatrix} 0.7053 & 0.7053 & 0.7053 & 0.0 \\ 0.0 & 0.9806 & 0.1961 & 0.0 \\ 0.0 & 0.0 & 1.0 & 0.0 \\ 0.0 & 0.7071 & 0.0 & 0.7071 \\ -0.7053 & -0.7053 & -0.7053 & 0.0 \\ 0.0 & -0.9806 & -0.1961 & 0.0 \\ 0.0 & 0.0 & -1.0 & 0.0 \\ 0.0 & -0.7071 & 0.0 & -0.7071 \end{pmatrix} \quad \mathbf{c} = \begin{pmatrix} 0.7053 \\ 0.0981 \\ 0.00 \\ 0.0707 \\ -0.6912 \\ -0.0883 \\ 0.00 \\ -0.0636 \end{pmatrix}. \tag{6.4}$$

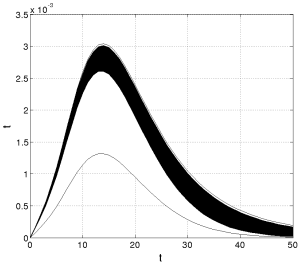
The offsets of the vector $\mathbf{c}$ were chosen in such a way that the parallelotope of initial conditions encloses the box with $s \in [0.89, 0.90], i \in [0.09, 0.10], t = 0.00$, and $r = 0.00$. From the initial parameter set with $\tau \in [0.001, 0.002]$ and $d \in [0.005, 0.010]$, our tool found the safe parameter subset depicted in Figure 6.7d in 6.27s. It is interesting to see



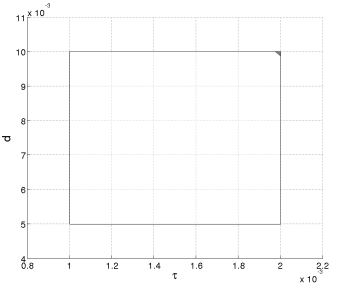

(a) Susceptible ($s$).




(b) Infected ($i$).




(c) Under treatment ($t$).




(d) Original (in white) and synthesize (in black) parameter sets.

Figure 6.7: Projections of reachable set and parameter sets of 4-dimensional influenza model with 2 controllable parameters. Sets have been computed with 1 template, unconstrained (in white, 0.51s) and constrained (in black, 6.27s). Specification: $G_{[0,50]} i \leq 0.4235$.

how the tool synthesized a tiny valid parameter set (upper-right corner of the original set). Figure 6.7 also shows the projections of unconstrained and constrained evolutions (in white and black, respectively) of the variables $s$, $i$, and $t$.

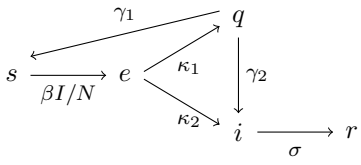
### 6.2.4 Ebola

We now focus on the third and last epidemic model. This model is a variation of the system that describes the Ebola outbreak in Congo 1995 and Uganda 2000 presented in [40]. A population composed of $N$ individuals, is classified in five compartments $s, e, q, i$, and $r$. Each individual, at a certain time, belongs to a specific compartment depending on the disease status. All the individual displacements between compartments are regulated by the parameters $\beta, \kappa_1, \kappa_2, \gamma_1, \gamma_2$, and $\sigma$.

Similarly to the previous epidemic models, $s$ contains the healthy individuals that are *susceptible* to the disease. A member of $s$ who enters in contact with a sick person, moves to $e$, that is the class of individuals who have been *exposed* to the disease. The ratio $i/N$ is the probability that a susceptible individual enters in contact with an infected one, while $\beta$ is the transmission rate. An exposed individual is either moved in *quarantine* in $q$, or directly in the *infected* compartment $i$, depending on whether the malady was

diagnosed. The controllable quarantine rate is $\kappa_1$, while $1/\kappa_2$ is the mean incubation period. A person in quarantine, if considered healthy after the isolation period, is moved back to the susceptible group. The unfortunate case is when the individual manifests symptoms and moves from the quarantine to the infected group. The reintegration with the susceptible people happens after a period of $1/\gamma_1$, while the incubation period is $1/\gamma_2$. Finally, an individual is *removed* from the system by migrating in $r$ at a recovering or death rate $\sigma$. $\Delta$ is the discretization step. The dynamics of the Ebola epidemic model are defined by the following system of difference equations (on the right there is schematic representation of the individuals migration between compartments):

$$
\begin{aligned}
s_{k+1} &= s_k - (s_k\beta i_k/N + \gamma_1 q_k)\Delta \\
e_{k+1} &= e_k + (s_k\beta i_k/N - (\kappa_1+\kappa_2)e_k)\Delta \\
q_{k+1} &= q_k + (\kappa_1 e_k - (\gamma_1+\gamma_2)q_k)\Delta \\
i_{k+1} &= i_k + (\gamma_2 q_k + \kappa_2 e_k - \sigma i_k)\Delta \\
r_{k+1} &= r_k + (\sigma i_k)\Delta
\end{aligned}
$$




The difference between our model and the one presented in [40] is that we introduce the quarantine compartment and consider the reintegration of individual in the susceptible population. In doing so, we enrich the original model by making it more realistic and interesting. Also, our model is defined on discrete time. Note that in the literature there are various works presenting epidemic models directly with discrete-time dynamics or difference equations (see, e.g., [1, 191]).

We first considered the normalized population with $s = 0.80$ and $i = 0.20$. We fixed the parameters values as specified in [40] in the case of the Ebola outbreak in Uganda during 2000. The uncontrollable parameter values are $\beta = 0.35, \kappa_2 = 0.3, \gamma_2 = 0.6$, and $\sigma = 0.28$, and the controllable parameters are $\kappa_1 \in [0.2, 0.3]$ and $\gamma_1 \in [0.2, 0.5]$ that represent the quarantine rate and mean isolation period, respectively. The discretization step is $\Delta = 1$.

We considered the specification $\phi_1 \equiv (i \leq 0.2)U_{[7,10]}(q \leq 0.01394)$ whose meaning is to avoid the saturation of the quarantine compartment especially in the time interval between 7 and 10 when a number of infected individuals higher than 0.01394 is expected. Our tool found two feasible parameters sets in 0.06 seconds, one of which is shown in Figure 6.8.

In a second experiment, we changed the uncontrollable parameter values to $\beta = 0.9$, $\kappa_2 = 0.5$, $\gamma_2 = 0.5$, and $\sigma = 0.28$, while the controllable parameters to $\kappa_1 \in [0.2, 0.3]$ and $\gamma_1 \in [0.2, 0.5]$. Instead of imposing directly a constraint on the system, we could imagine a scenario where we have a maximum number of 0.045 patients in quarantine unless the number of infected patients is below 0.230. This means that if there are less than 0.230 infected individuals, then we have free resources that can be devoted to the quarantine. This property can be formalized with the formula $\phi_2 \equiv (q \leq 0.045)U_{[10,15]}(i \leq 0.230)$. Our tool found six valid parameter sets in 0.15 seconds.

Finally, on the same system configuration, we test a more complex until formula that imposes upper and lower bounds on the compartments, i.e, $(q > 0.01 \wedge q \leq 0.05)U_{[10,15]}(i > 0.13 \wedge i \leq 0.29)$. Our tool found five parameter refinements in 0.50s. Figures 6.9a and 6.9b depict the projections of the variables $q$ and $i$ evolving in time with respect to one of the synthesized parameter sets.

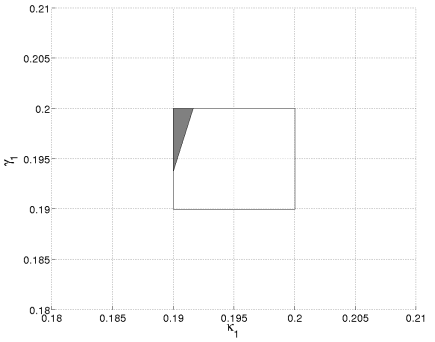
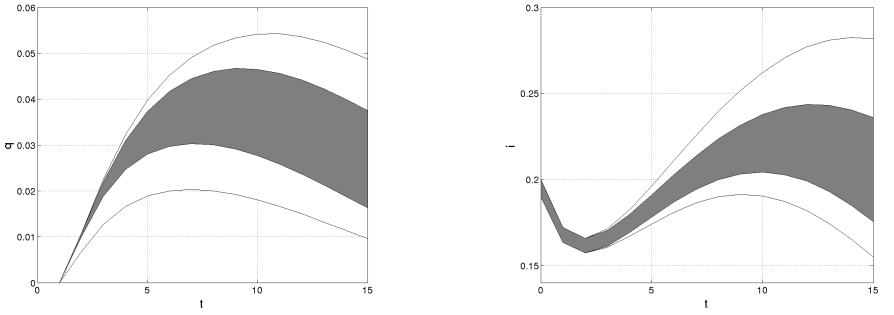



Figure 6.8: Original (in white) and synthesized (in gray) parameter sets of 5-dimensional Ebola model with 2 parameters. Specification: $(i \leq 0.2)U_{[7,10]}(q \leq 0.01394)$.




(a) Quarantine ($q$)

(b) Infected ($i$)

Figure 6.9: Projections of reachable set of 5-dimensional Ebola model. Sets have been computed with original (in white) and synthesized parameters (in gray, 0.50s).

### Scalability Evaluation

In order to evaluate the scalability of our method in terms of specification complexity, we now consider non-trivial formulas that we artificially created.

First, we consider the three until formulas $\phi_1, \phi_2$, and $\phi_3$ previously studied on the Ebola model (see Section 6.2.4), i.e.:

$$\begin{aligned} \phi_1 &\equiv (i \leq 0.23) U_{[a,b]} (q \leq 0.045) \\ \phi_2 &\equiv (q \leq 0.045) U_{[a,b]} (i \leq 0.23) \\ \phi_3 &\equiv (q \leq 0.045) U_{[a,b]} (e > 0.10 \vee q > 0.025) \end{aligned} \tag{6.5}$$

and we stretch their temporal intervals $[a, b]$. In the worst case, such growth exponentially increases the number of branches that our algorithm must open. Table 6.1 reports the running times for this test. For $\phi_1$ and $\phi_2$ we stretched the interval up to $[100, 200]$, that deals to parameter sets composed by 101 convex polytopes. As concerns $\phi_3$, we notice that enlargement of the until window does not affect the parameter refinement, that is valid refinements are found only for the initial part of the interval. Not growing in the size of the results, the algorithm needs linear time in the until time horizon.

As second evaluation, we nest several until formulas on the (most critical) right hand side. For instance, the double nesting of $\phi_1$ with $N = 2$ is:

$$\phi_1^N \equiv (i \leq 0.23) U_{[6,10]} (i \leq 0.23) U_{[6,10]} (q \leq 0.045)).$$

Table 6.2 reports the running times for this evaluation. We computed refinements of $\phi_1$ and $\phi_2$ nested 20 times, finding 81 and 84 convex parameter sets. As in the previous case, nesting $\phi_3$ does not increase the size of the result. It is interesting to notice that even if $\phi_1$ and $\phi_2$ are composed by less atomic formulas than $\phi_3$, their running times and sizes of results, are often larger than the latter. This suggests that it might be hard to estimate a priori the algorithm performance just by looking at the specification, since its execution time "numerically" depends on the system behavior and mostly on the computed partial refinements.

Finally, the experiments show that our tool tends to generate a large number of polytopes. So far, parameter set are represented as lists of linear systems composed by collections of inequalities. To reduce memory consumption it might be interesting to introduce mechanisms to avoid the insertion of redundant constraints and polytopes included in larger ones.

## 6.2.5 Honeybees Site Choice

We now abandon epidemics and move to macrobiology. We examine a model that describes the decision-making process mechanism adopted by a swarm of honeybees to choose one among two different nest-sites.

In this model [32], a population of honeybees is divided in five groups: $x$, the *neutral* bees that have not chosen a site; $y_1$ and $y_2$, *evangelic* bees dancing for the first and second site, respectively; $z_1$ and $z_2$, *non-evangelic* bees that have been converted to the first or second site, respectively, but who do not dance. The dynamics of the system are

| $a$ | $b$ | $\phi_1$ | $\phi_2$ | $\phi_3$ |
|---|---|---|---|---|
| 5 | 15 | 0.12 (11) | 0.16 (11) | 0.13 (9) |
| 5 | 20 | 0.15 (16) | 0.26 (16) | 0.18 (9) |
| 5 | 30 | 0.25 (26) | 0.50 (26) | 0.26 (9) |
| 5 | 50 | 0.44 (46) | 1.43 (46) | 0.45 (9) |
| 5 | 100 | 1.06 (96) | 7.84 (96) | 0.90 (9) |
| 5 | 125 | 1.45 (121) | 13.55 (121) | 1.11 (9) |
| 15 | 20 | 0.14 (6) | 0.22 (6) | 0.14 (0) |
| 20 | 30 | 0.20 (11) | 0.42 (11) | 0.23 (0) |
| 30 | 50 | 0.35 (21) | 1.10 (21) | 0.35 (0) |
| 50 | 100 | 0.81 (51) | 5.70 (51) | 0.73 (0) |
| 100 | 200 | 2.00 (101) | 23.84 (101) | 1.43 (0) |

Table 6.1: Increasing until interval. Times are expressed in seconds. Values in parenthesis are the computed polytopes per refinement. $\phi_1 \equiv (i \leq 0.23)U_{[a,b]}(q \leq 0.045)$, $\phi_2 \equiv (q \leq 0.045)U_{[a,b]}(i \leq 0.23)$, $\phi_3 \equiv (q \leq 0.045)U_{[a,b]}(e > 0.10 \vee q > 0.025)$.

| N | $\phi_1^N$ | $\phi_2^N$ | $\phi_3^N$ |
|---|---|---|---|
| 1 | 0.07 (5) | 0.09 (5) | 0.10 (5) |
| 2 | 0.14 (9) | 0.19 (12) | 0.22 (3) |
| 3 | 0.21 (13) | 0.28 (16) | 0.37 (3) |
| 4 | 0.27 (17) | 0.37 (20) | 0.52 (3) |
| 5 | 0.33 (21) | 0.46 (24) | 0.69 (3) |
| 10 | 0.69 (41) | 0.93 (44) | 1.97 (3) |
| 15 | 1.03 (61) | 1.45 (64) | 3.39 (3) |
| 20 | 1.44 (81) | 1.97 (84) | 6.53 (3) |

Table 6.2: Nested until. Times are expressed in seconds. Values in parenthesis are the computed polytopes per refinement. $\phi_1 \equiv (i \leq 0.23)U_{[a,b]}(q \leq 0.045)$, $\phi_2 \equiv (q \leq 0.045)U_{[a,b]}(i \leq 0.23)$, $\phi_3 \equiv (q \leq 0.045)U_{[a,b]}(e > 0.10 \vee q > 0.025)$.

the following:

$$\begin{aligned}
x_{k+1} &= x_k + (-\beta_1 x_k y_{1_k} - \beta_2 x_k y_{2_k})\Delta \\
y_{1_{k+1}} &= y_{1_k} + (\beta_1 x_k y_{1_k} - \gamma y_{1_k} + \delta\beta_1 y_{1_k} z_{1_k} + \alpha\beta_1 y_{1_k} z_{2_k})\Delta \\
y_{2_{k+1}} &= y_{2_k} + (\beta_2 x_k y_{2_k} - \gamma y_{2_k} + \delta\beta_2 y_{2_k} z_{2_k} + \alpha\beta_2 y_{2_k} z_{1_k})\Delta \\
z_{1_{k+1}} &= z_{1_k} + (\gamma y_{1_k} - \delta\beta_1 y_{1_k} z_{1_k} - \alpha\beta_2 y_{2_k} z_{1_k})\Delta \\
z_{2_{k+1}} &= z_{2_k} + (\gamma y_{2_k} - \delta\beta_2 y_{2_k} z_{2_k} - \alpha\beta_1 y_{1_k} z_{2_k})\Delta
\end{aligned} \tag{6.6}$$

The parameters $\beta_1$ and $\beta_2$ are the persuasion parameters, i.e., how vigorously the evangelic bees $y_1$ and $y_2$ dance; $\delta$ is the per capita rate at which the bees spontaneously leave the neutral and non-dancing groups $x, z_1, z_2$ for the dancing classes $y_1, y_2$; $\gamma$ is the per capita rate of ceasing to dance from the dancing classes $y_1, y_2$ to the non-dancing ones $x_1, x_2$; $\alpha$ is the proportionality of switching back spontaneously to the neutral state $x$; $\Delta$ is the discretization step.

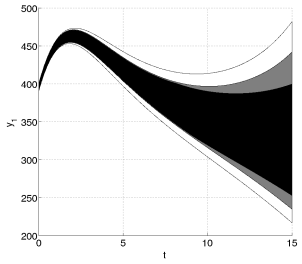
The goal of this test is to study the scalability of our reachability method in terms of number of directions and templates, and verify eventual improvements in the precision of the computed reachable set.

For the simulation of the model we choose as initial set the box with $x_0 = 500, y_1 \in [390, 400], y_2 \in [90, 100], z_1 = z_2 = 0$. The parameter values are $\beta_1 = \beta_2 = 0.001, \gamma = 0.3, \delta = 0.5, \alpha = 0.7$, and $\Delta = 0.01$. Figure 6.10 shows the projections of the dancing bees $y_1$ and $y_2$ computed with three different configurations up to 1500 steps. The bundles have been transformed with the AFO method and no decomposition. In the first configuration (in white), the computation has been carried out with a single template composed by 5 axis-aligned directions (6.57s); the second (in gray) involved 2 templates composed by 6 directions, some of which were not aligned with the $x$ and $y_1$ axis (26.90s). In the third configuration we defined 3 templates composed by 7 directions, some of which not aligned with $x, y_1$, and $y_2$ axis (81.27s). Adding directions and templates prolongs the execution times, but increases the precision of the computed flowpipwes leading to finer reachability sets. As a matter of fact, from Figure 6.10 we can see how the precision of the computed reachable set increases with the addition of directions and templates. The flowpipe computed with 3 templates and 7 directions (in black in Figure 6.10) is always included in the other flowpipes and its computation required the reasonable amount of time 81.27s for 1500 discrete steps.

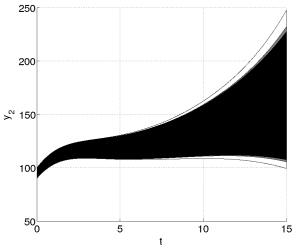
### 6.2.6 Quadcopter

In our last study we focus again on the scalability of our reachability method in terms of system dimension. For this study, we consider the model of a quadrotor drone composed by 17 variables regulated by quadratic dynamics. The model consists of 13 dynamics that drive the drone itself, plus 4 dynamics modeling its controller. The state variables of the drone include the inertial position $(p_n, p_e, h)$, linear velocity $(u, v, w)$, Euler angles expressed using quaternions[2] $(q_0, q_1, q_2, q_3)$, and angular velocity $(p, q, r)$, while the controller variables involve some parameters of position, speed, and angle $(h_I, u_I, v_I, \psi_I)$.

[2] Quaternions are a number system extending complex numbers useful for calculations involving 3-dimensional rotations. In our case, quaternions increase the original model size, but allow us to work with polynomials instead of trigonometric functions.




(a) Dancing bees $y_1$.




(b) Dancing bees $y_2$.

Figure 6.10: Projections of reachable set of 5-dimensional honeybees decision-making model. Sets have been computed with 1 template/5 directions (in white, 6.57s), 2 templates/6 directions (in gray, 26.90s), and 3 templates/7 directions (in black, 81.27s).

| Parameter | Value |
|---|---|
| $M$ | 0.0015 |
| $m_r$ | 0.001 |
| $R$ | 0.020 |
| $l$ | 0.045 |
| $g$ | 9.81 |
| $m$ | $M + 4m_r$ |
| $J_x$ | $(2MR^2)/5 + 2l^2m_r$ |
| $J_y$ | $(2MR^2)/5 + 2l^2m_r$ |
| $J_z$ | $(2MR^2)/5 + 4l^2m_r$ |

Table 6.3: Parameter values used in the quadcopter model (see Equation 6.7).

Given a reference height $h_r$, horizontal speeds $u_r, v_r$, and nose angle $\psi_r$, the goal of the controller is to bring the drone from its actual configuration to the one specified by the reference values. The detailed description of the model and its dynamics can be found in [51].

The difference equations that regulate the behavior of the quadcopter are the following:

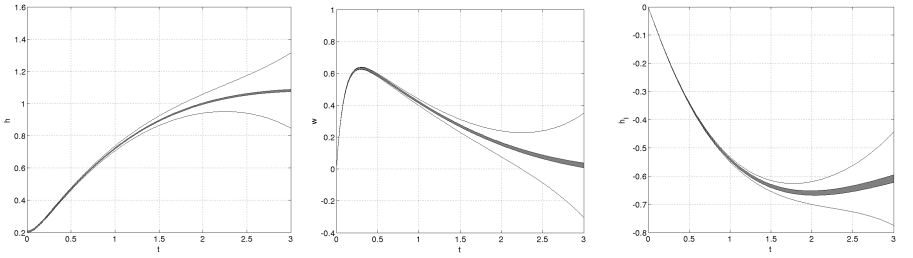
$$
\begin{aligned}
p_{n_{k+1}} &= p_{n_k} + (u_k(2q_{0_k}^2 + 2q_{1_k}^2 - 1) - v_k(2q_{0_k}q_{3_k} - 2q_{1_k}q_{2_k}) + w_k(2q_{0_k}q_{2_k} + 2q_{1_k}q_{3_k}))\Delta \\
p_{e_{k+1}} &= p_{e_k} + (v_k(2q_{0_k}^2 + 2q_{2_k}^2 - 1) + u_k(2q_{0_k}q_{3_k} + 2q_{1_k}q_{2_k}) - w_k(2q_{0_k}q_{1_k} - 2q_{2_k}q_{3_k}))\Delta \\
h_{k+1} &= h_k + (w_k(2q_{0_k}^2 + 2q_{3_k}^2 - 1) - u_k(2q_{0_k}q_{2_k} - 2q_{1_k}q_{3_k}) + v_k(2q_{0_k}q_{1_k} + 2q_{2_k}q_{3_k}))\Delta \\
u_{k+1} &= u_k + (r_k v_k - q_k w_k - g(2q_{0_k}q_{2_k} - 2q_{1_k}q_{3_k}))\Delta \\
v_{k+1} &= v_k + (p_k w_k - r_k u_k + g(2q_{0_k}q_{1_k} + 2q_{2_k}q_{3_k}))\Delta \\
w_{k+1} &= w_k + (q_k u_k - p_k v_k - F/m + g(2q_{0_k}^2 + 2q_{3_k}^2 - 1))\Delta \\
q_{0_{k+1}} &= q_{0_k} + (-(q_{1_k}/2)p_k - (q_{2_k}/2)q_k - (q_{3_k}/2)r_k)\Delta \\
q_{1_{k+1}} &= q_{1_k} + ((q_{0_k}/2)p_k - (q_{3_k}/2)q_k + (q_{2_k}/2)r_k)\Delta \\
q_{2_{k+1}} &= q_{2_k} + ((q_{3_k}/2)p_k + (q_{0_k}/2)q_k - (q_{1_k}/2)r_k)\Delta \\
q_{3_{k+1}} &= q_{3_k} + ((q_{1_k}/2)q_k - (q_{2_k}/2)p_k + (q_{0_k}/2)r_k)\Delta \\
p_{k+1} &= p_k + ((1/J_x)\tau_\phi + ((J_y - J_z)/J_x)q_k r_k)\Delta \\
q_{k+1} &= q_k + ((1/J_y)\tau_\theta - ((J_x - J_z)/J_y)p_k r_k)\Delta \\
r_{k+1} &= r_k + ((1/J_z)\tau_\psi + ((J_x - J_y)/J_z)p_k q_k)\Delta \\
h_{I_{k+1}} &= h_{I_k} + (h_k - h_r)\Delta \\
u_{I_{k+1}} &= u_{I_k} + (u_k - u_r)\Delta \\
v_{I_{k+1}} &= v_{I_k} + (v_k - v_r)\Delta \\
\psi_{I_{k+1}} &= \psi_{I_k} + (\psi_k - \psi_r)\Delta
\end{aligned}
\tag{6.7}
$$

All the parameters (such as mass, axis moment of inertia, propeller masses, etc.) have been set accordingly to the real quadcopter CrazyFlie Nano by Bitcraze[3] and are shown in Table 6.3. The discretization step is $\Delta = 0.01$.

The chosen initial conditions are $h_0 \in [0.20, 0.21]$, $q_0 = 1$, and all the other variables are set to zero. The reference height is $h_r = 1$, while speeds and angle are $u_r = v_r = \psi_r = 0$. We computed the reachable set up to 300 steps, corresponding to 3s of flight.

We adopted 2 configurations, both based on AFO transformation without the bundle decomposition: the first consists in a single box template with axis-aligned constraints, the second has an additional parallelotope involving the dimensions that more vary during the flight (height, vertical speed, angle quaternions, and controller height). Figure 6.11 shows the projections of the computed reachable sets. The figure reports the evolutions over time of height $h$ (Figure 6.11a), vertical speed $w$ (Figure 6.11b), and the height computed by the controller $h_I$ (Figure 6.11c), obtained with a single (in white) and two templates (in gray). The first technique took 9.40s of computations, the second 20.32s. Note how a single additional template sensibly improves the precision of the

[3] `https://www.bitcraze.io/`



(a) Height ($h$). (b) Vertical speed ($w$). (c) Controller height ($h_I$).

Figure 6.11: Projections of reachable set of 17-dimensional quadcopter model. Sets have been computed with 1 template/17 directions (in white, 17.74s), 2 templates/18 directions (in gray, 39.07s).

computed reachable set and avoid the wrapping effect. In this case, a single additional direction and template generates a sensibly finer flowpipe maintaining the computation in a reasonable amount of time (20.32s for 300 discrete steps).

## 6.3 Related Tools

In Sections 1.2.1 and 1.2.2 we presented some of the existing techniques for the reachable set computation and the parameter synthesis of dynamical and hybrid systems. As pointed out, efficient methods and tools for the verification of linear dynamical systems have been developed. Some examples are CheckMate [43], HyTech [107], d/dt [6], MPT [134], PHAVer [85], SpaceEx [87], and Ellipsoidal Toolbox (ET) [133].

Nevertheless, the analysis of nonlinear dynamical systems remains a problem that has not yet found a definitive solution. Some remarkable attempts to reduce the gap between linear and nonlinear verification have been done, for instance, by the authors of the tools d/dt [6], Ariadne [9], Coho [101], dReach [126], and Flow* [39].

The parameter synthesis and estimation problems have been considered by different communities and analyzed from perspectives. As stated in Section 1.2.2, often the definition of the parameter synthesis problem differs from the context in which it is considered. Consequently, it is rather difficult to put under one roof different tools for the synthesis of parameters. Despite this discrepancy, it is worth mentioning the tools HyTech [109], RoVerGeNe [15], Breach [69], dReach [126], and SpaceRover [25].

In this thesis we presented original theoretical results for the reachability computation and the synthesis of parameters for nonlinear dynamical systems. We validated our methods showing applications of our image computation and parameter synthesis techniques based on the Bernstein technique. The 17-dimensional case study of Section 6.2.6 shows potential applications of our methods. Due to the differences between the problem formulations and the purposes for which the existing tools have been developed, a fair and informative tool comparison should be based on examples and settings carefully defined. Since this is not the main goal of this work, we preferred to focus on our tool showing its features rather than directly compare it with the other tools.

# 7

# Conclusion

## 7.1 Thesis Overview

With this thesis we contributed to the development of theoretical and practical techniques for the analysis of discrete-time dynamical systems.

Our theoretical results involve the reachable set computation and the synthesis of parameters. Concerning reachability computation, we developed methods to over-approximate the images of sets with respects to polynomial functions (Chapter 4). Sets to be transformed can be over-approximated using boxes (Section 4.3), parallelotopes (Section 4.4), and our new data structure called parallelotope bundles (Section 4.5), that can be used to represent polytopes as intersections of parallelotopes. All these methods are based on the representation of polynomials in Bernstein basis, and in particular on Bernstein coefficients and their properties. We introduced new methods to efficiently compute Bernstein coefficients and improve the bounds that they provide (Section 4.6).

Regarding parameter synthesis, we formalized the problem from a formal verification perspective involving the Signal Temporal Logic (STL) for the characterization of the specifications that the system must meet. We developed a new semantics for STL, called synthesis semantics, suitable for working with flowpipes and sets of parameters (Section 5.2). We defined an algorithm to compute the synthesis semantics of STL specifications and obtain valid sets of parameters for discrete-time dynamical systems (Section 5.3). We also provided an instantiation of our synthesis algorithm for dynamical systems with polynomial dynamics, in which the reachable set can be approximated with boxes or parallelotopes, while parameter sets are represent by polytopes (Section 5.4). The synthesis technique broadly consists in the resolution of linear programs involving, also in this case, Bernstein coefficients.

From the practical point of view, we implemented a tool called Sapo that integrates all the techniques developed in work (Section 6.1). Sapo can be either used to compute flowpipes that over-approximate the reachable sets of dynamical systems, or to synthesize sets of parameters with respect to STL specifications. We shown the effectiveness of our tool and analyzed its performances on several case studies coming from epidemiology, macrobiology, and embedded systems (Section 6.2).

## 7.2 Further Developments

There are several aspects of this work that can be further investigated.

**Parallelization** There are several stages of our methods that can be easily parallelized. For instance, the computation of Bernstein coefficients of the functions that encode the lifting of a direction over a reached set, can be independently performed for each direction of a box or parallelotope. Or else, when working with bundles, every parallelotope can be independently treated, meaning that the bundle-based reachability method can be straightforwardly parallelized. Also, in a hypothetical error containment approach where the reachability computation or parameter synthesis problems are split in several subproblems, each instance can be separately treated in parallel.

It might be interesting to investigate the parallelization of our methods exploiting some environment for parallel computations (such as MPI [103] or CUDA [153]).

**Error Approximation** In Section 4.2.1 we have seen how the error between the maximum of a polynomial and its maximum Bernstein coefficient can be bound. In the bundle transformation we provided a heuristic to minimize this error. Moreover, in Section 4.5.2, we provided a subdivision method to obtain tight bounds. However, it remains the question of how automatically control the error introduced by Bernstein coefficients in the image of sets and in the synthesis of parameters. One can imagine a procedure that automatically splits the Bernstein coefficients so that to maintain the errors below a given threshold.

Since Bernstein coefficients are sensitive to the size of the set on which they are exploited, an additional technique to produce finer results, could consist in splitting the reachability and parameter sets. This approach could be integrated in a backtracking algorithm that, once it determines that the results are not enough precise, jumps back in the constructed flowpipe, splits the sets, solves the subproblems for the split sets, and finally combines the results.

**Input Synthesis** The parameter synthesis methods developed in this work can be adapted to synthesize valid inputs for open dynamical systems.

A discrete-time dynamical system with inputs can be described by difference equations of the form:

$$\mathbf{x}_{k+1} = \mathbf{f}(\mathbf{x}_k, \mathbf{u}_k)$$

with $\mathbf{f} : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$. Differently from parameters, inputs $\mathbf{u}_k$ can change in a given input space $\mathcal{U} \subseteq \mathbb{R}^m$ at every time step. The evolution $\mathbf{x}_0, \mathbf{x}_1, \mathbf{x}_2, \ldots$ of a dynamical system with inputs starts from an initial condition $\mathbf{x}_0$ and it is driven by a sequence of inputs $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \ldots$.

The input synthesis problem asks to determine a sequence of input sets $U_0, U_1, U_2, \ldots$, with $U_i \subseteq \mathcal{U}$, that leads to a flowpipe of inputs, such that all the trajectories starting in a given set of initial conditions $X_0$ with input sequence $\mathbf{u}_0, \mathbf{u}_1, \mathbf{u}_2, \ldots$, with $\mathbf{u}_i \in U_i$, satisfy a given specification.

Our parameter refinements can be used for input synthesis. Instead of accumulating a sequence of restrictions on the initial set to refine (in the case of parameter synthesis, the set $P$), at each step we can independently compute a refined set of inputs $U_i \subseteq$

$\mathcal{U}$, producing a flowpipe of inputs $U_0, U_1, U_2, \ldots$ so that the system satisfies a given specification.

**Hybrid Automata Verification** In this thesis we considered the computation of the reachable set of dynamical systems. A natural extension of this work could involve *hybrid automata*, that are mathematical models particularly suitable for the description of systems that exhibit discrete and continuous behaviors.

Intuitively, hybrid automata are directed graphs with decorated nodes and edges. Each node of the graph, often called mode or location, is equipped with a dynamical system (usually a set of difference or differential equations) that drives the system within a mode. Edges are decorated with activation and reset constraints that tell the system when a discrete transition between modes can occur, and with which values the system is reset once it jumps from a mode to another one.

The verification of hybrid automata with nonlinear dynamics is a difficult problem for which not many efficient methods have been developed (in comparison with those for the linear case; see Section 1.2.1 for an overview on the existing nonlinear reachability methods). The set image computation techniques developed in this work could be used to compute the reachable set of hybrid automata with polynomial dynamics: the evolutions within modes could be directly carried out applying our reachability methods; the activations would require the intersection between sets (here some attention should be placed on the sets described by the activations); the reset would consist in a single set image computable with one of our techniques (assuming that the reset function is polynomial).

# Bibliography


[1] Linda JS Allen. Some discrete-time si, sir, and sis epidemic models. *Mathematical biosciences*, 124(1):83–105, 1994.

[2] Matthias Althoff and Bruce H Krogh. Zonotope bundles for the efficient computation of reachable sets. In *Conference on Decision and Control and European Control Conference, CDC-ECC*, pages 6814–6821. IEEE, 2011.

[3] Matthias Althoff, Colas Le Guernic, and Bruce H Krogh. Reachable set computation for uncertain time-varying linear systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 93–102. ACM, 2011.

[4] Hirokazu Anai and Volker Weispfenning. Reach set computations using real quantifier elimination. In *Hybrid Systems: Computation and Control, HSCC*, pages 63–76, 2001.

[5] Yashwanth Annpureddy, Che Liu, Georgios E. Fainekos, and Sriram Sankaranarayanan. S-TaLiRo: A tool for temporal logic falsification for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 254–257, 2011.

[6] Eugene Asarin, Olivier Bournez, Thao Dang, and Oded Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 20–31. Springer, 2000.

[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] M. Avriel. *Nonlinear Programming: Analysis and Methods*. Dover Books on Computer Science Series. Dover Publications, 2003.

[9] Andrea Balluchi, Alberto Casagrande, Pieter Collins, Alberto Ferrari, Tiziano Villa, and Alberto L Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Mathematical Theory of Networks and Systems, MTNS*. Citeseer, 2006.

[10] Jiri Barnat, Lubos Brim, Adam Krejci, Adam Streck, David Safránek, Martin Vejnar, and Tomas Vejpustek. On parameter synthesis by parallel model checking. *IEEE/ACM Trans. Comput. Biology Bioinform.*, 9(3):693–705, 2012.

[11] 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.

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

[13] Grégory Batt, Calin Belta, and Ron Weiss. Model checking genetic regulatory networks with parameter uncertainty. In *Hybrid Systems: Computation and Control, HSCC*, pages 61–75, 2007.

[14] Grégory Batt, Calin Belta, and Ron Weiss. Model checking liveness properties of genetic regulatory networks. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 323–338, 2007.

[15] Grégory Batt, Calin Belta, and Ron Weiss. Temporal logic analysis of gene networks under parameter uncertainty. *IEEE Trans. Automat. Contr.*, 53:215–229, 2008.

[16] Grégory Batt, Michel Page, Irene Cantone, Gregor Goessler, Pedro T. Monteiro, and Hidde de Jong. Efficient parameter search for qualitative models of regulatory networks using symbolic model checking. *Bioinformatics*, 26(18), 2010.

[17] Grégory Batt, Delphine Ropers, Hidde de Jong, Johannes Geiselmann, Radu Mateescu, Michel Page, and Dominique Schneider. Analysis and verification of qualitative models of genetic regulatory networks: A model-checking approach. In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 370–375, 2005.

[18] Grégory Batt, Boyan Yordanov, Ron Weiss, and Calin Belta. Robustness analysis and tuning of synthetic gene networks. *Bioinformatics*, 23(18):2415–2422, 2007.

[19] Christian Bauer, Alexander Frink, and Richard Kreckel. Introduction to the GiNaC framework for symbolic computation within the C++ programming language. *Journal of Symbolic Computation*, 33(1):1–12, 2002.

[20] Spring Berman, Ádám Halász, and Vijay Kumar. Marco: a reachability algorithm for multi-affine systems with applications to biological systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 76–89. Springer, 2007.

[21] Sergei Natanovich Bernstein. Démonstration du théorème de weierstrass fondée sur le calcul des probabilités. *Communications de la Société Mathématique de Kharkov 2*, 1(4/5):1–2, 1912.

[22] Martin Berz and Kyoko Makino. Verified integration of odes and flows using differential algebraic methods on high-order taylor models. *Reliable Computing*, 4(4):361–369, 1998.

[23] Sergiy Bogomolov, Thomas A. Henzinger, Andreas Podelski, Jakob Ruess, and Christian Schilling. Adaptive moment closure for parameter inference of biochemical reaction networks. In *Computational Methods in Systems Biology, CMSB*, pages 77–89, 2015.

[24] Sergiy Bogomolov, Christian Schilling, Ezio Bartocci, Grégory Batt, Hui Kong, and Radu Grosu. Abstraction-based parameter synthesis for multiaffine systems. In *Hardware and Software: Verification and Testing, Haifa Verification Conference, HVC*, pages 19–35, 2015.

[25] Sergiy Bogomolov, Christian Schilling, Ezio Bartocci, Gregory Batt, Andreas Podelski, and Radu Grosu. Spacerover: Parameter synthesis for multiaffine systems beyond RoVerGeNe. *To appear*, 2015.

[26] Luca Bortolussi and Guido Sanguinetti. A statistical approach for computing reachability of non-linear and stochastic dynamical systems. In *Quantitative Evaluation of Systems, QEST*, pages 41–56, 2014.

[27] Oleg Botchkarev and Stavros Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Hybrid Systems: Computation and Control, HSCC*, pages 73–88. Springer, 2000.

[28] Olivier Bournez, Oded Maler, and Amir Pnueli. Orthogonal polyhedra: Representation and computation. In *Hybrid Systems: Computation and Control, HSCC*, pages 46–60. Springer, 1999.

[29] Lubos Brim, Milan Ceska, Martin Demko, Samuel Pastva, and David Safránek. Parameter synthesis by parallel coloured CTL model checking. In *Computational Methods in Systems Biology, CMSB*, pages 251–263, 2015.

[30] Lubos Brim, Milan Ceska, and David Safránek. Model checking of biological systems. In *Formal Methods for Dynamical Systems, SFM*, pages 63–112, 2013.

[31] M. Brin and G. Stuck. *Introduction to Dynamical Systems*. Cambridge University Press, 2002.

[32] NF Britton, NR Franks, SC Pratt, and TD Seeley. Deciding on a new home: how do honeybees agree? *Royal Society of London B: Biological Sciences*, 269(1498):1383–1388, 2002.

[33] Christopher W. Brown. QEPCAD B: A program for computing with semi-algebraic sets using CADs. *SIGSAM BULLETIN*, 37:97–108, 2003.

[34] Laurence Calzone, François Fages, and Sylvain Soliman. BIOCHAM: an environment for modeling biological systems and formalizing experimental knowledge. *Bioinformatics*, 22(14):1805–1807, 2006.

[35] Alberto Casagrande and Tommaso Dreossi. pyhybrid analysis: A package for semantics analysis of hybrid systems. In *Euromicro Conference on Digital System Design, DSD*, pages 815–818, 2013.

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

[37] Xin Chen, Edo Abraham, and Sriram Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *Real-Time Systems Symposium, RTSS*, pages 183–192. IEEE, 2012.

[38] Xin Chen, Erika Ábrahám, and Goran Frehse. Efficient bounded reachability computation for rectangular automata. In *Reachability Problems, RP*, pages 139–152, 2011.

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

[40] Gerardo Chowell, Nick W Hengartner, Carlos Castillo-Chavez, Paul W Fenimore, and JM Hyman. The basic reproductive number of Ebola and the effects of public health measures: the cases of Congo and Uganda. *Journal of Theoretical Biology*, 229(1):119–126, 2004.

[41] Alongkrit Chutinan and Bruce H Krogh. Computing polyhedral approximations to flow pipes for dynamic systems. In *Conference on Decision and Control, CDC*, volume 2, pages 2089–2094. IEEE, 1998.

[42] Alongkrit Chutinan and Bruce H Krogh. Computing approximating automata for a class of linear hybrid systems. In *Hybrid Systems V*, pages 16–37. Springer, 1999.

[43] Alongkrit Chutinan and Bruce H Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control, HSCC*, pages 76–90. Springer, 1999.

[44] Vasek Chvatal. *Linear programming*. Macmillan, 1983.

[45] Eugenio Cinquemani, Andreas Milias-Argeitis, Sean Summers, and John Lygeros. Stochastic dynamics of genetic networks: modelling and parameter identification. *Bioinformatics*, 24(23):2748–2754, 2008.

[46] Edmund M Clarke, Orna Grumberg, and Doron Peled. *Model checking*. MIT press, 1999.

[47] George E Collins. Quantifier elimination for real closed fields by cylindrical algebraic decompostion. In *Automata Theory and Formal Languages*, pages 134–183. Springer, 1975.

[48] Michael A Colón, Sriram Sankaranarayanan, and Henny B Sipma. Linear invariant generation using non-linear constraint solving. In *Computer Aided Verification, CAV*, pages 420–432. Springer, 2003.

[49] Patrick Cousot and Radhia Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *SIGACT-SIGPLAN symposium on Principles of programming languages*, pages 238–252. ACM, 1977.

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

[51] Antonio Eduardo Carrilho da Cunha. Benchmark: Quadrotor attitude control. In *Applied Verification for Continuous and Hybrid Systems, ARCH*, 2015.

[52] Thao Dang. Approximate reachability computation for polynomial systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 138–152. Springer, 2006.

[53] Thao Dang, Alexandre Donzé, Oded Maler, and Noa Shalev. Sensitive state-space exploration. In *Conference on Decision and Control, CDC*, pages 4049–4054, 2008.

[54] 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.

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

[56] Thao Dang and David Salinas. Image computation for polynomial dynamical systems using the Bernstein expansion. In *Computer Aided Verification, CAV*, pages 219–232. Springer, 2009.

[57] Thao Dang and Romain Testylier. Reachability analysis for polynomial dynamical systems using the Bernstein expansion. *Reliable Computing*, 17(2):128–152, 2012.

[58] Thi Xuan Thao Dang. *Verification and synthesis of hybrid systems*. PhD thesis, Institut National Polytechnique de Grenoble-INPG, 2000.

[59] Ben D'Angelo, Sriram Sankaranarayanan, César Sánchez, Will Robinson, Bernd Finkbeiner, Henny B. Sipma, Sandeep Mehrotra, and Zohar Manna. LOLA: runtime monitoring of synchronous systems. In *Temporal Representation and Reasoning, TIME*, pages 166–174, 2005.

[60] George B Dantzig and Mukund N Thapa. *Linear programming 2: theory and extensions*. Springer Science & Business Media, 2006.

[61] Philip J Davis and Philip Rabinowitz. *Methods of numerical integration*. Courier Corporation, 2007.

[62] Giuseppe De Giacomo and Moshe Y Vardi. Linear temporal logic and linear dynamic logic on finite traces. In *International Joint Conference on Artificial Intelligence, IJCAI*, pages 854–860. AAAI Press, 2013.

[63] Leonardo De Moura and Nikolaj Bjørner. Z3: An efficient smt solver. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 337–340. Springer, 2008.

[64] Andreas Dolzmann and Thomas Sturm. REDLOG computer algebra meets computer logic. *ACM SIGSAM Bulletin*, 31:2–9, 1996.

[65] M.M. Donahue, G.T. Buzzard, and A.E. Rundell. Robust parameter identification with adaptive sparse grid-based optimization for nonlinear systems biology models. In *American Control Conference, ACC*, pages 5055–5060, June 2009.

[66] Robin Donaldson and David R. Gilbert. A model checking approach to the parameter estimation of biochemical pathways. In *Computational Methods in Systems Biology, CMSB*, pages 269–287, 2008.

[67] A. Donzé. *Trajectory-Based Verification and Controller Synthesis for Continuous and Hybrid Systems*. PhD thesis, 2007.

[68] A. Donzé, E. Fanchon, L. M. Gattepaille, O. Maler, and P. Tracqui. Robustness analysis and behavior discrimination in enzymatic reaction networks. *PLOS One*, 6(9):e24246, 2011.

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

[70] Alexandre Donzé, Gilles Clermont, and Christopher James Langmead. Parameter synthesis in nonlinear dynamical systems: Application to systems biology. *Journal of Computational Biology*, 17(3):325–336, 2010.

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

[72] Alexandre Donzé and Oded Maler. Systematic simulation using sensitivity analysis. In *Hybrid Systems: Computation and Control, HSCC*, pages 174–189, 2007.

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

[74] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Method, NFM*, pages 127–142, 2015.

[75] Tommaso Dreossi, Thao Dang, and Carla Piazza. Parallelotope bundles for polynomial reachability. In *Hybrid Systems: Computation and Control, HSCC*, 2016. to appear.

[76] Freeman Dyson. A meeting with Enrico Fermi. *Nature*, 427(6972):297–297, 2004.

[77] Andreas Eggers, Nacim Ramdani, Nedialko S Nedialkov, and Martin Fränzle. Improving the sat modulo ode approach to hybrid systems analysis by combining different enclosure methods. *Software & Systems Modeling*, 14(1):121–148, 2012.

[78] Georgios E. Fainekos, Antoine Girard, and George J. Pappas. Temporal logic verification using simulation. In *Formal Modeling and Analysis of Timed Systems, FORMATS*, pages 171–186, 2006.

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

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

[81] Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Robustness of model-based simulations. In *Real-Time Systems Symposium, RTSS*, pages 345–354. IEEE Press, 2009.

[82] Gerald E Farin. *Curves and surfaces for CAGD: a practical guide*. Morgan Kaufmann, 2002.

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

[84] Richard P Feynman. Simulating physics with computers. *International journal of theoretical physics*, 21(6/7):467–488, 1982.

[85] Goran Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems: Computation and Control, HSCC*, pages 258–273. Springer, 2005.

[86] Goran Frehse, Sumit Kumar Jha, and Bruce H. Krogh. A counterexample-guided approach to parameter synthesis for linear hybrid automata. In *Hybrid Systems: Computation and Control, HSCC*, pages 187–200, 2008.

[87] 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.

[88] Sicun Gao. *Computable Analysis, Decision Procedures, and Hybrid Automata: A New Framework for the Formal Verification of Cyber-Physical Systems*. PhD thesis, PhD thesis, Carnegie Mellon University, 2012.

[89] Jürgen Garloff. The Bernstein algorithm. *Interval Computations*, 2(6):154–168, 1993.

[90] Jürgen Garloff and Andrew P Smith. Investigation of a subdivision based algorithm for solving systems of polynomial equations. *Nonlinear Analysis: Theory, Methods & Applications*, 47(1):167–178, 2001.

[91] Jürgen Garloff and Andrew P Smith. *Solution of systems of polynomial equations by using Bernstein expansion*. Springer, 2001.

[92] R. Ghaemi and D. Del Vecchio. Evaluating the robustness of a biochemical network model. In *Conference on Decision and Control, CDC*, pages 615–620, Dec 2007.

[93] Reza Ghaemi, Jing Sun, Pablo A Iglesias, and Domitilla Del Vecchio. A method for determining the robustness of bio-molecular oscillator models. *BMC systems biology*, 3(1):95, 2009.

[94] Michael E Gilpin. Do hares eat lynx? *American Naturalist*, pages 727–730, 1973.

[95] Frank Giordano, William P Fox, and Steven Horton. *A first course in mathematical modeling*. Cengage Learning, 2013.

[96] Antoine Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control, HSCC*, pages 291–305. Springer, 2005.

[97] Antoine Girard, Colas Le Guernic, and Oded Maler. Efficient computation of reachable sets of linear time-invariant systems with inputs. In *Hybrid Systems: Computation and Control, HSCC*, pages 257–271. Springer, 2006.

[98] Antoine Girard and George J Pappas. Verification using simulation. In *Hybrid Systems: Computation and Control, HSCC*, pages 272–286. Springer, 2006.

[99] Antoine Girard and George J Pappas. Approximation metrics for discrete and continuous systems. *IEEE Transactions on Automatic Control*, 52(5):782–798, 2007.

[100] Diana-Elena Gratie, Bogdan Iancu, and Ion Petre. ODE analysis of biological systems. In *Formal Methods for Dynamical Systems, SFM*, pages 29–62, 2013.

[101] Mark R Greenstreet and Ian Mitchell. Reachability analysis using polygonal projections. In *Hybrid Systems: Computation and Control, HSCC*, pages 103–116. Springer, 1999.

[102] D Yu Grigor'ev. Complexity of deciding tarski algebra. *Journal of symbolic Computation*, 5(1):65–108, 1988.

[103] William Gropp, Ewing Lusk, Nathan Doss, and Anthony Skjellum. A high-performance, portable implementation of the mpi message passing interface standard. *Parallel computing*, 22(6):789–828, 1996.

[104] Ashutosh Kumar Gupta, Rupak Majumdar, and Andrey Rybalchenko. From tests to proofs. *International Journal on Software Tools for Technology Transfer*, 15(4):291–303, 2013.

[105] Ernst Hairer, Christian Lubich, and Gerhard Wanner. *Geometric numerical integration: structure-preserving algorithms for ordinary differential equations*, volume 31. Springer Science & Business Media, 2006.

[106] John Havlicek, Scott Little, Oded Maler, and Dejan Nickovic. Property-based monitoring of analog and mixed-signal systems. In *Formal Modeling and Analysis of Timed Systems, FORMATS*, pages 23–24, 2010.

[107] 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.

[108] 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.

[109] Thomas A. Henzinger and Howard Wong-Toi. Using hytech to synthesize control parameters for a steam boiler. In *Formal Methods for Industrial Applications*, pages 265–282, London, UK, UK, 1996. Springer-Verlag.

[110] Francis Begnaud Hildebrand. *Introduction to numerical analysis*. Courier Corporation, 1987.

[111] Frederick S Hillier. *Introduction to operations research*. Tata McGraw-Hill Education, 1995.

[112] Alan C Hindmarsh, Peter N Brown, Keith E Grant, Steven L Lee, Radu Serban, Dan E Shumaker, and Carol S Woodward. Sundials: Suite of nonlinear and differential/algebraic equation solvers. *Transactions on Mathematical Software, TOMS*, 31(3):363–396, 2005.

[113] Faraz Hussain, Sumit Kumar Jha, Susmit Jha, and Christopher James Langmead. Parameter discovery in stochastic biological models using simulated annealing and statistical model checking. *IJBRA*, 10(4/5):519–539, 2014.

[114] Jean-Louis Imbert and Pascal Van Hentenryck. Redundancy elimination with a lexicographic solved form. *Annals of Mathematics and Artificial Intelligence*, 17(1):85–106, 1996.

[115] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *Hybrid Systems: Computation and Control, HSCC*, pages 43–52. ACM, 2013.

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

[117] A Agung Julius and George J Pappas. Trajectory based verification using local finite-time invariance. In *Hybrid Systems: Computation and Control, HSCC*, pages 223–236. Springer, 2009.

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

[119] James Kapinski, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan, and Nikos Aréchiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 133–142. ACM, 2014.

[120] Jim Kapinski, Bruce H Krogh, Oded Maler, and Olaf Stursberg. On systematic simulation of open continuous systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 283–297. Springer, 2003.

[121] Narendra Karmarkar. A new polynomial-time algorithm for linear programming. In *Symposium on Theory of computing, STOC*, pages 302–311. ACM, 1984.

[122] Mark H Karwan, Vahid Lotfi, Jan Telgen, and Stanley Zionts. *Redundancy in mathematical programming: A state-of-the-art survey*, volume 206. Springer Science & Business Media, 2012.

[123] 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.

[124] Jongrae Kim, Declan G Bates, Ian Postlethwaite, Lan Ma, and Pablo A Iglesias. Robustness analysis of biochemical network models. *IEE Proceedings-Systems Biology*, 153(3):96–104, 2006.

[125] Sharon E Kingsland. *Modeling nature*. University of Chicago Press, 1995.

[126] Soonho Kong, Sicun Gao, Wei Chen, and Edmund Clarke. dreach: $\delta$-reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 200–205. Springer, 2015.

[127] EK Kostousova. State estimation for dynamic systems via parallelotopes optimization and parallel computations. *Optimization Methods and Software*, 9(4):269–306, 1998.

[128] E. K. Kostousovat. Control synthesis via parallelotopes: optimzation and parallel compuations. *Optimization Methods and Software*, 4(14):267–310, 1 2001.

[129] Konstantinos Koutroumpas, Zoi Lygerou, and John Lygeros. Parameter identification for a DNA replication model. In *Conference on Bioinformatics and Bioengineering, BIBE*, pages 1–6, 2008.

[130] Arnold R Krommer. *Numerical Integration: On Advanced Computer Systems*, volume 848. Springer Science & Business Media, 1994.

[131] Alexander B. Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis. In *Hybrid Systems: Computation and Control, HSCC*, pages 202–214, 2000.

[132] Alexander B Kurzhanski and Pravin Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Systems & control letters*, 41(3):201–211, 2000.

[133] Alex A Kurzhanskiy, Pravin Varaiya, et al. Ellipsoidal toolbox. *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2006-46*, 2006.

[134] Michal Kvasnica, Pascal Grieder, Mato Baotić, and Manfred Morari. Multiparametric toolbox (mpt). In *Hybrid Systems: Computation and Control, HSCC*, pages 448–462. Springer, 2004.

[135] Gerardo Lafferriere, George J Pappas, and Sergio Yovine. Symbolic reachability computation for families of linear vector fields. *Journal of Symbolic Computation*, 32(3):231–253, 2001.

[136] Jean-Louis Lassez, Tien Huynh, and Ken McAloon. *Simplification and elimination of redundant linear arithmetic constraints*. Mit Press, 1993.

[137] Colas Le Guernic. *Reachability analysis of hybrid systems with linear continuous dynamics*. PhD thesis, Université Joseph-Fourier-Grenoble I, 2009.

[138] Colas Le Guernic and Antoine Girard. Reachability analysis of linear systems using support functions. *Nonlinear Analysis: Hybrid Systems*, 4(2):250–262, 2010.

[139] Gabriele Lillacci and Mustafa Khammash. Parameter estimation and model selection in computational biology. *PLoS Comput Biol*, 6(3):e1000696, 2010.

[140] Ernest Lindelöf. Sur lapplication de la méthode des approximations successives aux équations différentielles ordinaires du premier ordre. *Comptes rendus hebdomadaires des séances de l'Académie des sciences*, 114:454–457, 1894.

[141] Bing Liu, Soonho Kong, Sicun Gao, Paolo Zuliani, and Edmund M. Clarke. Parameter synthesis for cardiac cell hybrid models using delta-decisions. *CoRR*, abs/1407.1524, 2014.

[142] Alfred Lotka. Elements of physical biology. 1925.

[143] 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.

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

[145] Jürgen Mayer, Khaled Khairy, and Jonathon Howard. Drawing an elephant with four complex parameters. *American Journal of Physics*, 78(6):648–649, 2010.

[146] Pedro Mendes and D Kell. Non-linear optimization of biochemical pathways: applications to metabolic engineering and parameter estimation. *Bioinformatics*, 14(10):869–883, 1998.

[147] Robin Milner. *An Algebraic Definition of Simulation Between Programs*. Stanford University, Stanford, CA, USA, 1971.

[148] Robin Milner. *Communication and Concurrency*. Prentice-Hall, Inc., Upper Saddle River, NJ, USA, 1989.

[149] Marvin L Minsky. *Computation: finite and infinite machines*. Prentice-Hall, Inc., 1967.

[150] Carmen G Moles, Pedro Mendes, and Julio R Banga. Parameter estimation in biochemical pathways: a comparison of global optimization methods. *Genome research*, 13(11):2467–2474, 2003.

[151] Bernard Mourrain and Jean Pascal Pavone. Subdivision methods for solving polynomial equations. *J. Symb. Comput.*, 44(3):292–306, 2009.

[152] PSV Nataraj and M Arounassalame. A new subdivision algorithm for the Bernstein polynomial approach to global optimization. *International journal of automation and computing*, 4(4):342–352, 2007.

[153] CUDA Nvidia. Programming guide, 2008.

[154] Sucheendra K. Palaniappan, Benjamin M. Gyori, Bing Liu, David Hsu, and P. S. Thiagarajan. Statistical model checking based calibration and analysis of bio-pathway models. In *Computational Methods in Systems Biology, CMSB*, pages 120–134, 2013.

[155] David Park. *Concurrency and automata on infinite sequences*. Springer, 1981.

[156] P. A. Gonzlez Parra, S. Lee, L. Velzquez, and C. Castillo-Chavez. A note on the use of optimal control on a discrete time model of influenza dynamics. *Mathematical Biosciences and Engineering*, 8(1):183–197, 2011.

[157] Pablo A Parrilo. Semidefinite programming relaxations for semialgebraic problems. *Mathematical programming*, 96(2):293–320, 2003.

[158] Pablo A Parrilo and Bernd Sturmfels. Minimizing polynomial functions. *Algorithmic and quantitative real algebraic geometry, DIMACS Series in Discrete Mathematics and Theoretical Computer Science*, 60:83–99, 2003.

[159] Carla Piazza, Marco Antoniotti, Venkatesh Mysore, Alberto Policriti, Franz Winkler, and Bud Mishra. Algorithmic algebraic model checking i: Challenges from systems biology. In *Computer Aided Verification, CAV*, pages 5–19. Springer, 2005.

[160] André Platzer. Differential dynamic logic for verifying parametric hybrid systems. In *Automated Reasoning with Analytic Tableaux and Related Methods, TABLEAUX*, pages 216–232, 2007.

[161] André Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[162] André Platzer and Jan-David Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning, IJCAR*, pages 171–178, 2008.

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

[164] Pavithra Prabhakar and Mahesh Viswanathan. A dynamic algorithm for approximate flow computations. In *Hybrid Systems: Computation and Control, HSCC*, HSCC '11, pages 133–142, New York, NY, USA, 2011. ACM.

[165] Nacim Ramdani and Nedialko S Nedialkov. Computing reachable sets for uncertain nonlinear hybrid systems using interval constraint-propagation techniques. *Nonlinear Analysis: Hybrid Systems*, 5(2):149–162, 2011.

[166] Dietmar Ratz and Tibor Csendes. On the selection of subdivision directions in interval branch-and-bound methods for global optimization. *Journal of Global Optimization*, 7(2):183–207, 1995.

[167] Shashwati Ray and PSV Nataraj. An efficient algorithm for range computation of polynomials using the Bernstein form. *Journal of Global Optimization*, 45(3):403–426, 2009.

[168] Shashwati Ray and PSV Nataraj. A matrix method for efficient computation of bernstein coefficients. *Reliable Computing*, 17(1):40–71, 2012.

[169] Sriram Sankaranarayanan, Thao Dang, and Franjo Ivančić. Symbolic model checking of hybrid systems using template polyhedra. In *Tools and Algorithms for the Construction and Analysis of Systems, TACAS*, pages 188–202. Springer, 2008.

[170] Sriram Sankaranarayanan, Henny B. Sipma, and Zohar Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model Checking, and Abstract Interpretation, VMCAI*, pages 25–41, 2005.

[171] Mohamed Amin Ben Sassi and Sriram Sankaranarayanan. Bernstein polynomial relaxations for polynomial optimization problems. *arXiv preprint arXiv:1509.01156*, 2015.

[172] Mohamed Amin Ben Sassi, Sriram Sankaranarayanan, Xin Chen, and Erika Ábrahám. Linear relaxations of polynomial positivity for polynomial lyapunov function synthesis. *IMA Journal of Mathematical Control and Information*, page dnv003, 2015.

[173] Hermann Schichl. Models and the history of modeling. In Josef Kallrath, editor, *Modeling Languages in Mathematical Optimization*, volume 88 of *Applied Optimization*, pages 25–36. Springer US, 2004.

[174] O Shisha. The Bernstein form of a polynomial. *Journal of Research of the National Bureau of Standards: Mathematics and mathematical physics. B*, 70:79, 1966.

[175] Andrew Paul Smith. *Enclosure Methods for Systems of Polynomial Equations and Inequalities*. PhD thesis, 2012.

[176] AndrewPaul Smith. Fast construction of constant bound functions for sparse polynomials. *Journal of Global Optimization*, 43(2-3):445–458, 2009.

[177] S. Stoma, A. Donzé, F. Bertaux, O. Maler, and G. 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.

[178] Olaf Stursberg and Bruce H Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control, HSCC*, pages 482–497. Springer, 2003.

[179] Alfred Tarski. A decision method for elementary algebra and geometry. *1948*, 1951.

[180] Jan Telgen. Redundancy and linear programs. *MC Tracts*, 137:1–125, 1981.

[181] Romain Testylier. *Reachability analysis of non-linear systems*. PhD thesis, 2012.

[182] Ufuk Topcu, Andrew Packard, and Peter Seiler. Local stability analysis using simulations and sum-of-squares programming. *Automatica*, 44(10):2669–2675, 2008.

[183] Ufuk Topcu, Andrew Packard, Peter Seiler, and Timothy Wheeler. Stability region analysis using simulations and sum-of-squares programming. In *American Control Conference, ACC*, pages 6009–6014. IEEE, 2007.

[184] Tayssir Touili, Byron Cook, and Paul Jackson, editors. *Computer Aided Verification, CAV*, volume 6174 of *Lecture Notes in Computer Science*. Springer, 2010.

[185] Pascal Van Hentenryck and Thomas Graf. Standard forms for rational linear arithmetic in constraint logic programming. *Annals of Mathematics and Artificial Intelligence*, 5(2-4):303–319, 1992.

[186] Pravin Varaiya. Reach set computation using optimal control. In M.Kemal Inan and RobertP. Kurshan, editors, *Verification of Digital and Hybrid Systems*, volume 170 of *NATO ASI Series*, pages 323–331. Springer Berlin Heidelberg, 2000.

[187] Vito Volterra. *Variazioni e fluttuazioni del numero d'individui in specie animali conviventi*. C. Ferrari, 1927.

[188] J Wei. Least square fitting of an elephant. *Chemtech*, 5(2):128–129, 1975.

[189] Michael Zettler and Jürgen Garloff. Robustness analysis of polynomials with polynomial parameter dependency using bernstein expansion. *IEEE Transactions on Automatic Control*, 43(3):425–431, 1998.

[190] Yan Zhang, Sriram Sankaranarayanan, Fabio Somenzi, Xin Chen, and Erika Ábrahám. From statistical model checking to statistical model inference: characterizing the effect of process variations in analog circuits. In *International Conference on Computer-Aided Design, ICCAD*, pages 662–669, 2013.

[191] Xiaoliang Zhou, Xiaopei Li, and Wu-Sheng Wang. Bifurcations for a deterministic sir epidemic model in discrete time. *Advances in Difference Equations*, 2014(1):1–16, 2014.