HDR

Thao DANG, CR1, CNRS

December 8, 2009

Contents

Chapter 1

Motivations and scientific challenges

Since 2001, after joining the CNRS, the main theme of my research has been "**Methods and tools for computer aided design of embedded systems**".

Due to an increasing utilization of computers, there has been a dramatic rise in interest in embedded systems, that is systems in which the computer interacts with the physical world. Embedded systems are present in a large variety of application domains, including avionics/aeronautics, space, transport, automotive, telecommunications, smart cards, consumer electronics. Embedded systems are composed of hardware and software components specifically designed for controlling a given application device.

Hybrid systems, that is, systems exhibiting both continuous and discrete dynamics, have been recognized as a high-level model appropriate for embedded systems, since this model can describe, within a unified framework, the logical part and the continuous part of an embedded system. They have also been used as a mathematical model for various physical phenomena and engineering systems, such as chemical process control, avionics, robotics, automobiles, manufacturing, and most recently in molecular biology.

Due to the safety critical features of many such applications, formal analysis is a topic of particular interest. Recently, much effort has been devoted to the development of automatic analysis methods and tools for hybrid systems, based on formal verification. This can be seen in a large number of publications on the topic in the recent proceedings of HSCC *Hybrid Systems: Computation and Control*, a major international conference of the domain. The goal of *formal verification* is to prove that the (designed) system satisfies a property. Due to the complexity and scale of real-life applications, automatic analysis is very desirable. This is a motivation to adopt the algorithmic approach which consists in building software tools that can analyze automatically all the behaviors of a

given system. Results in algorithmic verification of hybrid systems resulted in a number of verification tools, such as **Coho**[65] developed at University of British Columbia, **Verdict** [**?**] developed at University of Dortmund, **CheckMate** [31] developed at University of Carnegie Mellon, **VeriShift** [26] developed at University of Berkeley, **d/dt** [17] developed at VERIMAG, MPT [76] developed at ETH Zurich, HJB toolbox [85] developed at Stanford University. Although these methods and tools have been successfully applied to a number of interesting case studies, their applicability is still limited to systems of small size due to the complexity of exhaustive analysis, required by formal verification. It soon became clear that for systems of industrial size, one needs more 'light-weight' methods.

Testing is another validation approach, which can be used for much larger systems and is a standard tool in industry, despite its limitations compared to algorithmic and deductive verification. Indeed, like simulation techniques, testing can only reveal an error but does not permit proving the correctness of the system. The success of the testing approaches in industry is perhaps due to the fact that they suffer less from the problem of state explosion. Indeed, the engineer can choose the degree of validation by changing the number of tests. In fact, the larger number of tests are executed, the larger portion of behaviors of the system is validated, and therefore the more confidence in the correctness of the system we gain. This is different from the results of the type 'yes' or 'no' provided by the formal verification methods. On the other hand, testing can be applied to real systems, and not only on their models.

Although testing has been well studied in the context of finate state machines, it has not been much investigated for continuous and hybrid systems. Therefore, a question of great interest is to bridge the gap between the verification and testing approaches, by defining a formal framework for testing of hybrid systems and developing methods and tools that help automate the testing process, in particular test generation from specifications.

A number of special characteristics of hybrid systems make their validation particularly challenging, in particular the combination of the complexity in both discrete and continuous aspects. While continuous systems have been well studied in control theory and continuous mathematics, and discrete systems have been investigated in computer science, the interaction between continuous and discrete dynamics leads to fundamental problems (such as undecidability) which are not yet well understood or for which a general solution is often impossible.

A number of special characteristics of hybrid systems make their testing particularly challenging, in particular:

- **Combination of the complexity in both discrete and continuous aspects**. While continuous systems have been well studied in control theory and continuous mathematics, and discrete systems have been investigated in

computer science, the interaction between continuous and discrete dynamics leads to fundamental problems (such as undecidability) which are not yet well understood or for which a general solution is often impossible.

- **Infiniteness of the state space of a hybrid system and of the input space**. In general, in order to test an open system, one first needs to feed an input signal to the system and then check whether the behavior of the system induced by this input signal is as expected. When there is an infinite number of possible input signals, it is important to choose the ones that lead to interesting scenarios (with respect to the property/functionality to test).

Methodology and main results

To respond to these challenges, the main objective of my research has been to develop new validation methods and tools, with a focus on semi-formal approaches (such as testing), by applying the ideas from different domains (such as, geometric design, interval computation), and by exploring new application areas (such as, analog and mixed-signal circuits).

I have carried out both theoretical and practical research work. In this thesis. I will describe a selection of main results, which describes best my visions and efforts. These results can be summarized as follows:

1. A number of verification techniques for non-linear hybrid systems
2. A conformance testing framework including a coverage-guided test generation algorithms and a test generation tool.
3. A novel approach for scheduling of real-time multi-threaded programs
4. Exploration of two new application domains which is analog and mixed signal circuits and biological systems.

1.1 Thesis outline

In Chapter **??**, we discuss *hybrid automata* [**?**], the modeling formalism we use for hybrid systems. We then introduce our formal framework for conformance testing, which includes the basic concepts (such as conformance relation, test cases, test executions). The chapter contains the theoretical background necessary for the subsequent developments.

Chapter **??** is concerned with the development of two test coverage measures. We are interested in test coverage measures that describe how well the states

visited during a test execution represent the reachable set. The first coverage measure we propose is based on the star discrepancy notion, and the second is based on the δ-cover notion. We then present the methods to compute these coverage measures.

In Chapter **??**, we develop an algorithm for generating test cases from hybrid automata. This algorithm is an extension of the RRT (Rapidly-exploring Random Tree) algorithm [79] to hybrid systems. An important property of this algorithm, namely probabilistic completeness, is then discussed.

In Chapter **??**, in order to rapidly achieve a good coverage quality, we develop a method for guiding the test generation algorithm using the coverage measures.

The goal of Chapter **??** is to show how to implement the abstract algorithms introduced in the previous chapters. This can be seen as a concrete realization of these algorithms. We also briefly present the prototype test generation tool that we call **HTG**. The tool provides automatic test generation from hybrid automata.

Chapter **??** is devoted to a number of case studies treated using the tool **HTG**. These case studies come from control applications and analog and mixed signal circuits. To illustrate the performance of our test generation algorithms, some linear systems (with up to 100 continuous variables), which are randomly generated, were treated. The experimental results shows the applicability of our approach to high dimensional systems.

Each of the above chapters also includes a discussion of related work. The concluding chapter summarizes the contributions of the thesis and suggests future research directions.

For the best understanding of this thesis, the reader is encouraged to follow chapter by chapter. In particular, Chapter **??** contains important definitions and notations which are used throughout the thesis.

Chapter 2

Reachability analysis of non-linear systems

2.1 Context

Reachability analysis is an important problem in formal verification. A major ingredient in designing a reachability analysis algorithm for hybrid systems is an efficient method to handle their continuous dynamics described by differential equations (since their discrete dynamics can be handled using existing discrete verification methods). While many well-known properties of linear differential equations can be exploited to design relatively efficient methods, non-linear systems are much more difficult to analyze. This motivated me to continue this direction, which I had started in my PhD thesis.

My focus and methodology have however been more application driven, which can be summarized as follows.

Hybrid systems as a means of approximation. Hybrid systems have been proven to be a mathematical model appropriate for reasoning about interactions between discrete and analog parts in embedded applications as well as discontinuous phenomena in molecular biology. My attempt to apply hybrid systems verification technology to analog circuits and biological systems allowed me to have the following understanding which stimulated my research directions in analysis of non-linear systems

- Besides the phenomena which are intrinsically of mixed discrete-continuous nature, hybrid systems can be used to approximate complex systems by

simpler ones. In addition, it also allows to naturally capture stiffness in continuous dynamics, which often causes instability in traditional methods such as numerical simulation.

- While the hybrid systems methodology can provide effective modelling for many applications, its use does not come for free. Indeed, even when continuous dynamics can be efficiently handled, discrete dynamics (which theoretically can be handled using well-developed techniques for discrete systems) may lead to significant computation costs, as high as that for overcoming numerical instability. Indeed, while numerical instability is often addressed by reducing time step sizes in order to adapt to fast changes of some variables, switching continuous dynamics via discrete transitions in a hybrid system may deteriorate "nice" geometric structures of continuous reachable sets. As an example, trajectories starting from points in a convex polyhedron can reach a transition guard at very different times, and the accumulation of starting points for the next continuous dynamics may form a "curved" set with complex geometry.

Taking both the drawbacks and advantages of the hybrid system methodology into account, I revisited the reachability analysis approach for non-linear systems using hybridization, developed in collaboration with Eugene Asarin (LIAFA, Paris) and Antoine Girard (Laboratory LJK, Grenoble). The main idea of this approach is to "decompose" a non-linear vector field into different segments each of which is approximated with a linear vector field. The resulting approximation is thus a piece-wise linear system which can be treated using available techniques for hybrid systems with linear continuous dynamics. This approach is very general and in principle can be applied to a large class of non-linear systems. However, in practice, the price for having "artificial" discrete transitions is often high, since the simpler continuous dynamics are used, the larger number of segments is needed in order to assure a desired precision. I thus seeked a way to get rid of these discrete transitions. In fact, these transitions can be smoothened without compromising the approximation quality. This led to the development of a dynamic hybridization approach, which enabled us to treat a number of biological systems with up to 9 continuous variables. This is a considerable progress since the original hybridization approach was limited to systems with only 3, 4 continuous variables.

Specialization for specific non-linear systems. Besides my efforts in tackling general non-linear systems, I am conscious that it is hard to design an efficient general-purpose algorithm for automatic verification of non-linear hybrid systems, since such an algorithm would not be able to exploit the particularities of each class of non-linearities. I thus centered my attention in multi-affine systems, which have numerous applications in many domains such as biology and

economy. Despite their simple non-linearity form they can exhibit complicated behaviors including chaos. I discovered a number of useful geometric properties not only of multi-affine but also of polynomial functions which were already used in robust control to handle uncertain parameters in transfer functions [**?**]. These properties have also been intensively used in Computer Aided Geometric Design. This inspired me to develop the analysis methods which are specialized for polynomial systems.

Beyond ODEs. Analog circuits are often modeled using differential algebraic equations (DAEs) rather than ordinary differential equations (ODEs). This was the motivation of my work on reachability analysis of DEAs that I will briefly summarize in Section 2.3.

Abstraction as a means of complexity reduction. A major challenge in algorithmic verification of hybrid systems is that the size of practical systems is often much larger than what computational exhaustive analysis can handle. This therefore requires methods for complexity reduction. To address this problem, I worked on two abstraction methods: predicate abstraction and abstraction by projection.

In the following I present some of these results and briefly describe the other. I first focus on the hybridization approach, which can be applied to a large class of non-linear continuous systems. In the next chapter, I will focus on techniques specialized for polynomials systems. It is important to note that, for simplicity of presentation the results are explained for continuous systemsl; however, all the techniques were designed can be applied to hybrid systems which requires additionally Boolean operations when handling discrete transitions.

2.1.1 Related work

Before continuing, I present a brief review of related work. The reachability problem for continuous systems described by differential equations has motivated much research both for theoretical problems, such as computability (see for example [11]), and for the development of computation methods and tools. If the goal is to compute exactly the reachable set or approximate it as accurately as possible, one can use a variety of methods for tracking the evolution of the reachable set under the continuous flows using some set represention (such as polyhedra, ellipsoids, level sets) [64, 39, 31, 75, 13, 102, 85, 59]. Since high quality approximations are hard to compute, other methods seek approximations that

are sufficiently good to prove the property of interest[1] (such as barrier certificates [92], polynomial invariants [101]). Abstraction methods for hybrid systems are also close in spirit to these methods. Indeed, their main idea is to approximate the original system with a simpler system (that one can handle more efficiently) and refine it if the analysis result obtained for the approximate system is too conservative (see for example [100, 9, 32, 15, 14]).

2.2 Hybridization

In this work, our object of study is a non-linear continuous system and we apply the hybrid systems methodology as a systematic approximation method. More concretely, we approximate a complex system with a simpler system, for which well-developed analysis tools exist (such as systems with affine continuous dynamics). To this end, we partition the state space of the system into disjoint regions and then approximate its dynamics in each region by a simpler dynamics. Thus, globally, the approximate system when moving from one region to another changes the dynamics. Due to these switchings, the approximate system behaves like a hybrid system and we thus call this approximation process *hybridization*. Then, the resulting system is used to yield approximate analysis results for the original system. The essence behind this approach was first put forward in [**?**, 58] to provide efficient numerical simulation of differential equations.

The usefulness of this approach (in terms of accuracy and computational tractability) depends on the choice of the approximate system. We developed two methods using two classes of approximate systems: piecewise affine (defined over a simplicial partition) and piecewise multi-affine (defined over a rectangular partition). However, while the continuous part is simplified, this increased the complexity of the discrete part, which makes reachability analysis more difficult. Indeed, frequent treatments of discrete transitions cause non-convex sets with complex geometric form that are hard to handle. We are thus working on a method that can avoid a direct treatment the discrete transitions.

On the other hand, as mentioned in the previous chapter, we developed a reachability analysis method specialized for multi-affine systems in order to deal with sub-systems generated by the hybridization. The hybridization using multi-affine subsystems is more efficient than the one using affine systems because rectangular partitions are easier to handle than simplicial ones.

This work was done in collaboration with Eugene Asarin from LIAFA (Paris), Antoine Girard from LJK (Grenoble). The results were published in the proceedings of the conference HSCC 2003 (*Hybrid Systems - Computation and Con-*

[1] It should be noted that reachable set computations can also be used for controller synthesis where the accuracy criterion is important.

trol) [15] and in the journal *Acta Informatica* January 2007 [16].

Our most recent result is a new scheme for dynamical hybridization. In dynamical hybridization, the partition of the state space is dynamically created, so that the intersection with the boundary of two adjacent regions can be avoided. This greatly improves the efficiency and scalability of the method, and we have successfully applied the method to some biological systems. The work on dynamical hybridization has been done in collaboration with Oded Maler and Colas LeGuernic from VERIMAG. The results were published in the proceedings of the conference CMSB 2009 (*Computational Methods for Systems Biology*) [**?**].

In this chapter I describe only our most recent results on hybridization, obtained in collaboration with Oded Maler and Romain Testylier from VERIMAG. This is indeed a topic of Romain's PhD thesis work, co-advised by myself and Oded.

The dynamic scheme gives us more freedom in the choice of the size and shape of hybridization domains (with each of which simpler dynamics is associated), freedom that we can exploit. The core problem we investigate in this paper is the following: given a convex polytope P representing the reachable set at some iteration of the reachability computation algorithm, find a domain D, an affine function $Ax + b$ and an error set U which *satisfy the condition for approximation convervativeness* and, in addition, are good with respect to the following efficiency and accuracy criteria which are partially-conflicting:

1. The size of the error set U is small;
2. The affine function $Ax + u$ (where $u \in U$) and the error set U can be efficiently computed;
3. The system's evolution remains in Δ as long as possible.

The first optimization criterion is important not only for the approximation accuracy but also for the computation time-efficiency. Let us give an intuitive explanation of this. Non-linear systems often behave in a much less predictable manner than linear systems. A linear system preserves convexity and therefore exploring its behavior starting from a finite number of "extremal" points (for example, the vertices of a convex polytope) is sufficient to construct the set of trajectories from all the points in that set. This is no longer true for most non-linear systems since the boundary of a reachable set can originate from some "non-extremal" points. Hence, the effect of error accumulation in the analysis of non-linear systems is more significant. For example, when the error part contains some points that generate completely different behavior patterns (for example, a significant change in the evolution direction), these spurious behaviors may consume a lot of computation time.

The novelty of the results presented in the following is that we exploit new tighter error bounds for linear interpolation in order to improve both the accuracy and efficiency of reachability computations. These tighter error bounds allow using large domains for the same desired accuracy and thus the linearization procedure is invoked less often.

The third criterion also aims at reducing the frequency of constructing new domains. As we will show, the error bound requirement leaves some freedom for choosing the position and orientation of the domains, which is used to address this criterion.

The rest of the chapter is organized as follows. We first recall the basic principles hybridization and introduce necessary notation and definitions. We then describe the error bound for linear interpolation that we will use in this work and compare it with the (larger) bounds used in our previous work [15, 16]. We then present a method for building simplicial approximation domains that satisfy this error bound while taking into account the above efficiency and accuracy criteria. We finally demonstrate this new method on a biological system with 12 continuous variables, that is $x \in \mathbb{R}^{12}$.

2.2.1 Hybridization: Basic ideas

We consider a non-linear system

$$\dot{x}(t) = f(x(t)),\ x \in \mathcal{X} \subseteq \mathbb{R}^n. \tag{2.1}$$

where the function f is Lipschitz. The set $\mathcal{X}$ is called the state space.

The basic idea of hybridization is to approximate the system (2.1) with another system that is easier to analyze:

$$\dot{x}(t) = g(x(t)),\ x \in \mathcal{X} \subseteq \mathbb{R}^n. \tag{2.2}$$

In order to capture all the behaviors of the original system (2.1), an input is introduced in the system (2.2) in order to account for the approximation error.

Let μ be the bound of $||g - f||$, i.e. for all $x \in \mathcal{X}$

$$||g(x) - f(x)|| \leq \mu$$

where $||\cdot||$ is some norm on $\mathbb{R}^n$. In this work we will consider the norm $||\cdot||_\infty$ which is defined as

$$||x||_\infty = \max(|x_1|, \ldots, |x_n|).$$

The approximate system with input is written as:

$$\begin{cases} \dot{x}(t) = s(x(t), u(t)) = g(x(t)) + u(t), \\ u(\cdot) \in \mathcal{U}_\mu \end{cases} \tag{2.3}$$

where $\mathcal{U}_\mu$ is the set of admissible inputs which consists of piecewise continuous functions u of the form $u : \mathbb{R}^+ \to U$ where U contains all points $u \in \mathbb{R}^n$ such that such that $||u(\cdot)|| \leq \mu$.

The system (2.3) is an overapproximation of the original system (2.1) in the sense that all trajectories of (2.1) are contained in the set of trajectories of (2.3) [?]. From now on, we call (2.3) "approximate system".

The construction of such an approximate system consists of two main steps:

- Inside a zone of interest that contains the current reachable set, we compute an approximation domain of size ϱ_{max}. Then, an approximate vector field is assigned to that domain. When the system's trajectories leave the current approximation domain, a new domain is determined. If the approximate vector field in each domain is affine, the resulting system f is piecewise affine over the whole state space. The use of such approximate systems is motivated by a large choice of available methods for the verification of piecewise affine systems (see for instance [?, ?, ?, ?, ?]). However, other classes of functions can be used, such as constant or multi-affine.
- To construct the error set U, we estimate the error bound μ which depends on the domain size ϱ_{max}. We assume that the chosen function f satisfies the following property: μ tends to 0 when ϱ_{max} tends to 0. Suppose that we can find an upper bound of μ, denoted by $\overline{\mu}$. Then, we can choose the input value set U to be the ball (i.e. a hypercube for the infinity norm) that is centered at the origin and has radius $\overline{\mu}$.

In this work, we focus on the problem of approximating the reachable set of the system (2.1). Some notation related to the reachable sets is needed. Let $\Phi_s(t, x, u(\cdot))$ be the trajectory starting from x of the system (2.3) under input $u(\cdot) \in \mathcal{U}$. The reachable sets of the system (2.3) from a set of initial points $X_0 \subseteq \mathcal{X}$ during the interval $[0, t]$ is defined as: $Reach_s(t, X_0) = \{\, y = \Phi_s(\tau, x, u(\cdot)) \mid \tau \in [0, t], x \in X_0, u(\cdot) \in U_\mu \,\}$. The reachable set of the original system can be defined similarly.

The following theorem shows the convergence of the reachable set of the approximate system to that of the original system [15].

Theorem 1. *Let L be the Lispchitz constant of the vector field f of the system (2.1) on $\mathcal{X}$. Then*

$$d_H\left(Reach_f(T, X_0), Reach_s(T, X_0)\right) \leq \frac{2\mu}{L}(e^{LT} - 1)$$

where d_H denotes the Hausdorff distance associated with the chosen norm $||\cdot||$.

This theorem shows the importance of the magnitude of μ since the error in the reachable set approximation depends linearly on μ. This is a motivation for our search for better error bounds, especially for linear interpolation which is an efficient method for affine hybridization that we explain in the next section.

Affine hybridization

We will now focus on the hybridization that uses affine functions for each approximation domain, which is a simplex. We recall that a simplex in $\mathbb{R}^n$ is the convex hull of $(n+1)$ affinely independent points in $\mathbb{R}^n$.

Suppose that we start with some initial set which is a polytope P_0. Around P_0, we construct an approximation, around P_0 which contains P_0. In our first work [15, 16], each domain is a cell in a simplicial mesh. Inside each cell the approximate vector field is defined using linear interpolation of f over the vertices of the cell. As mentioned earlier, the inconvenience of this hybridization (which we call static hybridization since the mesh is defined a-priori) is it requires expensive intersection operations when handling the transition of the system from one cell to its adjacent cells. To remedy this, rectangular mesh was then proposed. Nevertheless, interpolating over the rectangle vertices results in multi-affine functions which are harder to analyze.

In our recent paper [**?**], we proposed dynamic hybridization, in which a new domain is constructed only when the system is about to leave the current domain. Since intersection is no longer required, we can use a larger choice of approximation domain types for function approximations. In this work, we use again linear interpolation on simplices which is an efficient function approximation method. In addition, we exploit new better error bounds to investigate how the approximation quality of a simplex depends on its shape, size and orientation, in order to significantly improve the function approximation accuracy.

In the remainder of this section, we recall the linear interpolation on the vertices of a simplex. We denote by P_v the set of the vertices of a simplex Δ. We define l as an affine map of the form: $l(x) = Ax+b$ (A is a matrix of size $n \times n$ and $b \in \mathbb{R}^n$) such that l interpolates the function f at the vertices of Δ. More precisely,

$$\forall p \in P_v : \ f(p) = l(p).$$

An important advantage of this approximation method is that using the vertices of each simplex, the affine interpolant l is uniquely determined, since each simplex has exactly $(n+1)$ vertices.

Let us now define an input set U so that l is a conservative approximation of the original vector field f. To this end, we define the interpolation error as:

$$\mu = \sup_{x \in \Delta} ||f(x) - l(x)||.$$

Note that the real distance between the original function f and the approximating function l is key to the approximation quality, however this distance is hard to estimate precisely. It is easy to see the importance of the tightness of error bounds, since this directly impacts the error between the solutions of the two systems. In our previous work we used the following bounds on μ for two cases: the vector field f is Lipschitz and f is a C^2 function.

- If f is Lipschitz and L is its Lipschitz constant, then

$$\mu \leq \varrho_{max} \frac{2n\,L}{n+1} = \overline{\mu}(\varrho_{max}).$$

  where ϱ_{max} is the maximal edge length of the simplex.

- If f is C^2 on Δ with bounded second order derivatives then

$$\mu \leq \frac{Kn^2}{2(n+1)^2} \varrho_{max}^2 = \overline{\mu}(\varrho_{max}) \tag{2.4}$$

  where K is a bound on the second derivatives of f where

$$K = \max_{i \in \{1,\ldots,n\}} \sup_{x \in \Delta} \sum_{p_1=1}^{p_1=n} \sum_{p_2=1}^{p_2=n} \left| \frac{\partial^2 f^i(x)}{\partial x_{p_1} \partial x_{p_2}} \right|.$$

We write the above error bounds as a function of ϱ_{max} to emphasize that it depends on the maximal simplex edge length ϱ_{max}.

2.2.2 Tighter error bounds

In this section, we describe better error bounds on the interpolation over a simplex Δ in $\mathbb{R}^n$. The class of systems we consider are assumed to satify some smoothness conditions. To explain this, we need the notion of *curvature*.

From now on we write $f = (f_1, f_2, \ldots, f_n)$ as a vector of n functions $f_i : \mathbb{R}^n \to \mathbb{R}$. We first define the Hessian matrix associated with the function f_i with $i \in \{1, \ldots, n\}$ as:

$$H_i(x) = \begin{pmatrix} \frac{\partial^2 f_i}{\partial x_1^2} & \frac{\partial^2 f_i}{\partial x_1 x_2} & \cdots & \frac{\partial^2 f_i}{\partial x_1 x_n} \\ \frac{\partial^2 f_i}{\partial x_1 x_2} & \frac{\partial^2 f_i}{\partial x_2^2} & \cdots & \frac{\partial^2 f_i}{\partial x_2 x_n} \\ & & \cdots & \\ \frac{\partial^2 f_i}{\partial x_1 x_n} & \frac{\partial^2 f_i}{\partial x_2 x_n} & \cdots & \frac{\partial^2 f_i}{\partial x_n^2} \end{pmatrix}. \tag{2.5}$$

For any unit directional vector d, the directional curvature of f_i is defined as

$$\partial f_i(x, d) = d^T H_i(x) d.$$

Given a set $X \subseteq \mathcal{X}$, if f satisfies the following condition for all unit vector $d \in \mathbb{R}^n$

$$\forall i \in \{1, \ldots, n\} \; \forall x \in X : \max |\partial f_i(x, d)| \leq \gamma_X, \tag{2.6}$$

the value γ_X is called the maximal curvature of f in X. In other words, the above means that all the eigenvalues of H_i are in $[-\gamma_X, \gamma_X]$.

The following theorem gives a bound on the interpolation error [**?**].

Theorem 2. *Let l be the affine functions that interpolates the functions f over the simplex Δ. Then, for all $x \in \Delta$*

$$||f(x) - l(x)|| \leq \gamma_\Delta \frac{r_c^2(\Delta)}{2}.$$

where γ_Δ is the maximal curvature of f in Δ, and $r_c(\Delta)$ is the radius of the smallest ball that contains the simplex Δ.

For short, we say "the smallest containment ball" to refer to the smallest ball that contains the simplex Δ. Figure 2.2.2 illustrates this notion in two dimensions where simplices are triangles.

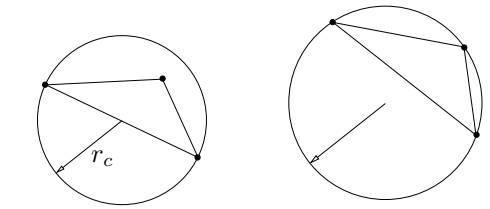


Figure 2.1: The smallest containment circle of the same triangle (shown on the left), which should not be confused with its circumcirle (shown on the right).

Compared to the error bound in (2.4), this error bound is tighter due to the relation between the maximal edge length of a simplex and the radius of its smallest containnement ball. This will be discussed in more detail later (especially in Lemma 2).

We can see that within a ball of radius r_c, if the curvature is constant, the simplices with the largest volume that guarantee the interpolation error bound

$\gamma_\Delta \frac{r_c^2(\Delta)}{2}$ of Theorem 2 are equilateral. We recall that a simplex is called *equilateral* if their edges have the same length. However, this error bound is appropriate only when the directional curvatures are not much different in every direction. There are functions where the largest curvature in one direction greatly exceeds the largest curvature in another, and in these cases it is possible to achieve the same accuracy with non-equilateral simplices. Intuitively, we can stretch an equilateral simplex along a direction in which the curvature is small in order to obtain a new simplex with larger size.

A better way to judge the approximation quality of a simplex is to map it to an "isotropic" space where the curvature bounds are isotropic. Indeed it is possible to derive an error bound similar to the one in Theorem 2 but with the radius of the smallest containment ball in this "isotropic" space [**?**]. To explain this, we define:

$$C = \Omega \Xi \Omega^T$$

where $\Omega = [\omega_1 \omega_2 \dots \omega_n]$ and

$$\Xi = \begin{pmatrix} \xi_1 & 0 & \dots & 0 \\ 0 & \xi_2 & \dots & 0 \\ & & \dots & \\ 0 & 0 & \dots & \xi_n \end{pmatrix}.$$

The vectors ω_i and values ξ_i are the eigenvectors and eigenvalues of a symmetric positive-definite matrix C, defined in the following.

We assume the boundedness of directional curvature of f. Given a subset X of $\mathcal{X}$ and a symmetric positive-definite matrix $C(X)$, if for any unit vector $d \in \mathbb{R}^n$,

$$\forall i \{1, \dots, n\} \; \forall x \in \Delta : \; \max |d^T H_i(x) d| \leq d^T C(X) d,$$

we say that in the set X the directional curvature of f is bounded by C and we call C is a curvature tensor matrix of f in X.

Let ξ_{max} and ξ_{min} be the largest ans smallest eigenvalues of $C(\Delta)$. The curvature matrix $C(\Delta)$ can be specified using an estimate of the Hessian matrices H_i. This will be discussed in more detail in Section 2.2.7.

We now define a matrix T which maps a point in the original space (that is, the domain over which the functions f are defined) to an isotropic space:

$$T = \Omega \begin{pmatrix} \sqrt{\xi_1/\xi_{max}} & 0 & \dots & 0 \\ 0 & \sqrt{\xi_2/\xi_{max}} & \dots & 0 \\ & & \dots & \\ 0 & & \dots & \sqrt{\xi_n/\xi_{max}} \end{pmatrix} \Omega^T. \tag{2.7}$$

Given a set $X \subseteq \mathbb{R}^n$, let $\widehat{X}$ denote the set resulting from applying the linear transformation specified by the matrix T to X, that is, $\widehat{X} = \{Tx \mid x \in X\}$. Geometrically, the transformation T "shortens" a set along the directions in which f has high curvatures. An illustration of this transformation is depicted in Figure 2.2.2, where the application of the transformation T to an ellipsoid produces a circle. When applying T to the triangle inscribed the ellipsoid shown on the left, the result is a more regular triangle shown on the right.

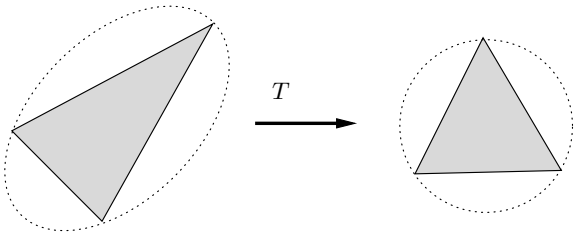


Figure 2.2: Illustration of the transformation to the isotropic space.

Theorem 3. *Let l be the affine functions that interpolates the functions f over the simplex Δ. Then, for all $x \in \Delta$*

$$||f(x) - l(x)|| \leq \gamma_\Delta \frac{r_c^2(\widehat{\Delta})}{2} = \bar{\mu}_{new}(r_c).$$

where γ_Δ is the maximal curvature in Δ and $r_c(\widehat{\Delta})$ is the radius of the smallest containement of the transformed simplex $\widehat{\Delta}$.

Proof. The idea of the proof is as follows. Let

$$\phi(x) = f(T^{-1}x)$$

be the function defined over the isotropic space. Similarly, for the linear interpolating function l, we define

$$\lambda(x) = l(T^{-1}x).$$

Note that $\widehat{f}(\widehat{x}) = f(x)$. So the range of ϕ over the domain $\widehat{\Delta}$ is the same as the range of f over the domain Δ. The curvature of ϕ has a bound that is independent of direction. Let $G_i(x)$ denote the Hessian matrix of $\phi(x)$. Indeed,

$$\begin{aligned} \partial\phi_i(x,d) &= d^T G_i(x) d \\ &= (T^{-1}d)^T H_i(x)(T^{-1}d) \end{aligned}$$

It then follows from the definition of the curvature tensor matrix $C(\Delta)$, we have

$$\begin{aligned} \partial\phi_i(x,d) &\leq d^T T^{-1} C(\Delta) T^{-1} d \\ &\leq \gamma_\Delta \end{aligned}$$

We thus see that $\gamma_{\widehat{\Delta}} = \gamma_\Delta$. Using Theorem 2, the maximum of $||\phi(x) - \lambda(x)||$ over $\widehat{\Delta}$ is $\gamma_{\widehat{\Delta}} \frac{r_c^2(\widehat{\Delta})}{2} = \gamma_\Delta \frac{r_c^2(\widehat{\Delta})}{2}$. By the above definitions of the functions ϕ and λ, we have $f(x) = \phi(\widehat{x})$ and $l(x) = \lambda(\widehat{x})$ we have

$$\max_{x \in \Delta} ||f(x) - l(x)|| = \max_{x \in \widehat{\Delta}} ||\phi(x) - \lambda(x)||.$$

It then follows that

$$||f(x) - l(x)|| \leq \gamma_\Delta \frac{r_c^2(\widehat{\Delta})}{2}.$$

■

To show the interest of this error bound, we first show that using transformation T the smallest containment ball radius is reduced or at worst unchanged; hence we can use larger simplices for the same error bound.

Lemma 1. *Given a simplex $\Delta \subseteq \mathbb{R}^n$, the radius of the smallest contrainment ball of $\widehat{\Delta}$ is not larger than the radius of the smallest contrainment ball of Δ, that is $r_c(\widehat{\Delta}) \leq r_c(\Delta)$.*

The proof can be directly established from the construction of the transformation matrix T. The error bound of Theorem 3 is at least as good as that of Theorem 2. For a "thin" simplex whose longer edges are along the directions of the eigenvectors associated with smaller eigenvalues, the ratio $\dfrac{r_c(\widehat{\Delta})}{r_c(\Delta)}$ can be as small as $\sqrt{\dfrac{\xi_{min}}{\xi_{max}}}$. In the worst case, when the simplex is "parallel" to an eigenvector associated with largest eigenvalue, this ratio is 1. ■

Furthemore, we compare the new error bounds with the ones shown in (2.4) which were used in the previous work. We first notice that the bound K in (2.4) must be larger than γ_Δ. To see this, we notice that any matrix norm is always larger than the maximum of the absolute values of the eigenvalues. It is however not easy to relate the smallest containment ball with the simplex size. For comparison purposes, we can use the following result.

Lemma 2. *Let Δ be a simplex in $\mathbb{R}^n$ with the maximal edge length ϱ_{max}. Then, the radius $r_c(\Delta)$ of its smallest containment sphere satisfies*

$$r_c(\Delta) \leq \varrho_{max} \sqrt{\frac{n}{2(n+1)}}$$

where n is the dimension of the system.

The proof of this inequality can be found, for example, in [**?**]. Indeed, the equality is achieved when the smallest containment ball of a simplex is also its circumscribed ball.

A direct consequence of this result is the following ratio between the old and new error bounds for any simplex.

Theorem 4. *For any simplex Δ with the maximal edge length ϱ_{max}, the ratio between the new error bound $\bar{\mu}_{new}$ of Theorem 3 and the old error bound $\bar{\mu}$ in (2.4) satisfies the following inequality:*

$$\frac{\bar{\mu}_{new}(r_c(\widehat{\Delta}))}{\bar{\mu}(\varrho_{max})} \leq \frac{n+1}{2n}.$$

In two dimensions, compared to the old error bound, the new error bound is reduced at least by the factor 4/3. The reduction factor $\dfrac{2n}{n+1}$ grows when the dimension n increases and approaches 2 when n tends to infinity.

This reduction is very useful especially in high dimensions because when dividing a simplex in order to satisfy some edge length bound, the number of resulting subsets grows exponentially with the dimension. Moreover, as in the above discussion of Lemma 1, by choosing an appropriate orientation we can reduce this ratio further by $\sqrt{\dfrac{\xi_{min}}{\xi_{max}}}$. ■

2.2.3 Construction of simplicial domains

We consider the problem of constructing a simplex around a polytope P (which is for example the reachable set in the current iteration)with the objective of achieving a good approximation quality when performing analysis on the approximate system to yield the result for the original system.

2.2.4 Simplex size and shape

We first consider the accuracy criterion. More precisely, we want to guarantee that the linear function that interpolates f satisfies a given desired error bound, say ρ. Let γ be the maximal curvature within a region of interest around the initial set, and for short we write it without specifying the simplex.

Theorem 3 indicates that the interpolation error variation depends on the radius $r_c(\widehat{\Delta})$. In order to exploit this result, we first transform the polytope P to $\widehat{P} = TP$ in the "isotropic" space. Let B be the ball of radius $\sqrt{2\rho/\gamma}$ the centroid of which coincides with that of the polytope $\widehat{P}$. We assume that $\widehat{P}$ is entirely included in B. If this is not the case, the polytope P should be split. The problem of finding a good splitting method is not addressed in this paper. In the current implementation the splitting direction is perpendicular to the direction along which the polytope is most stretched out.

Let $E = T^{-1}(B)$ be the ellipsoid resulting from applying the inverse transformation T^{-1} to the ball B. Then, according to Theorem 3 the interpolation error associated with any simplex inside the ellipsoid E is guaranteed to be smaller than or equal to ρ.

Since there are many simplices that can be fit in a ball, we proceed with the problem of choosing a simplex that satisfies the other optimization criteria, namely the simplex volume and the time of evolution within the simplex.

Lemma 3. *Let Δ_r be an equilateral simplex that the ball B is its circumscribed ball. Then, $T^{-1}(\Delta_r)$ is a simplex inscribed in the ellipsoid $E = T^{-1}B$ with the largest volume.*

The proof of this result relies on two standard results. First, the linear transformation preserves the volume ratio between two measurable bodies. Second, the simplices inside a ball with the largest volume are equilateral. ■

In follows from the lemma that we only need to consider the simplices resulting from applying T^{-1} to the largest equilateral simplices inscribing in the ball B. Any such simplex is guaranteed to be inscribed in the ellipsoid E and to have the largest volume.

2.2.5 Simplex orientation

It remains to select one of the above simplices to meet the staying time requirement. To this end, we use the following heuristics. We sample trajectories starting at a number of points inside and around the polytope P and then determine an *average evolution direction* e for a given time interval. We then want the simplex to be "aligned" with this direction e, as shown in Figure 2.2.5.

Note that we are considering only the equilateral simplices inscribed in B. We

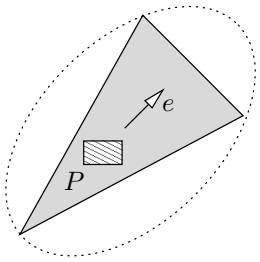


Figure 2.3: Illustration of the average evolution direction e.

now first pick an equilateral simplex Δ_r aligned with an axis, say x_1, as shown in Figure 2.4. This equilateral simplex can be efficiently constructed since, due to its alignment, the construction can be done by recursively reducing to lower dimensions. Without loss of generality, we can assume that the simplex has a vertex p on this axis x_1. We now want to compute the linear transformation M which rotates it to align with $-e$. To do so, we compute its inverse tranformation as follows. Choosing a simplex vertex p as a "pivot" vertex, we define its associated median axis as the line passing through p and perpendicular to the hyperplane containing the corresponding base. Let q be the vector representing this median axis, as shown in Figure 2.4.

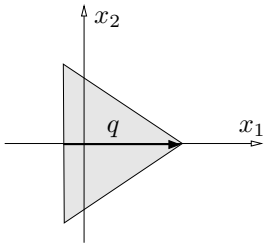


Figure 2.4: Illustration of a simplex median axis.

We want to compute a transformation R that aligns q with $-e$. This transformation is decomposed into $(n-1)$ successive rotations, each of which is around an axis x_i.

These rotations are illustrated with a 3-dimensional example in Figure 2.2.5. The median axis q of the simplex lies on the axis x_1. The bold line segment represents the vector $-e$ to rotate. After the first rotation by the angle θ_1 around the axis x_1, the new vector is on the plan (x_1, x_2). The second rotation by the angle θ_2 is around the axis x_3 to finally align the vector with q. The required transformation M is then obtained by computing the inverse of R, that is $M =$

R^{-1}.

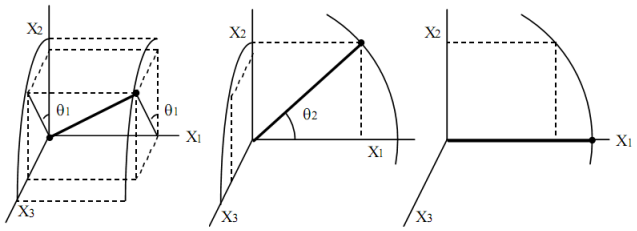


Figure 2.5: Successive rotations needed to align a vector with the axis x_1.

2.2.6 Curvature estimation

The curvature tensor matrix is needed to define the transformation T.

We first consider the case where the Hessian matrices are constant, such as the class of quadratic functions[2]. To compute a curvature tensor matrix, we first define a matrix C_i as the matrix with the same eigenvectors and eigenvalues as H_i, except that each negative eigenvalue ξ of H_i is replaced with the positive eigenvalue $-\xi$. Note that we can, in this case, omit the simplex in the notation of the curvature tensor matrix. Hence, C_i is guaranteed to be positive definite. If any eigenvalue of H_i is zero, we substitute it with some small positive value. That is, for each matrix H_i, we define

$$C_i(\Delta) = [\omega_1^i \dots \omega_n^i] \begin{pmatrix} |\xi_1^i| & 0 & \dots & 0 \\ 0 & |\xi_2^i| & \dots & 0 \\ & & \dots & \\ 0 & 0 & \dots & |\xi_n^i| \end{pmatrix} [\omega_1^i \dots \omega_n^i]^T$$

where ω_j^i (with $j \in \{1, \dots, n\}$) are the eigenvectors of H_i. We denote by ξ_{max}^i the eigenvalue with the largest absolute magnitude of C_i. Among the matrices C_i we can choose the one with the largest absolute eigenvalue to be a curvature tensor matrix.

For more general classes of functions where the Hessian matrices are not constant, we can estimate the curvature tensor matrix using optimization. This optimization can be done a-priori for the whole state space or it can be done locally each time we construct a new approximation domain. The transformation matrix T can then be computed using (2.7).

[2]Multi-affine and quadratic functions have found numerous applications, such as in biology and economy.

2.2.7 Simplex construction algorithm

Before continuing, the developments so far is summarized in an algorithm for computing a simplicial domain around a polytope P. Let r_c be the radius of the smallest containement ball in the isotropic space that satisfies a given desired error bound. The transformation matrix is denoted by T.

Algorithm 1 Simplex construction

$\widehat{P} = TP$
Compute a ball B around $\widehat{P}$
Choose Δ_r as an equilateral simplex inscribed in B such that an median axis q of Δ_r is aligned with the axis x_1.
Compute the average trajectory direction e (by sampling trajectories from P)
$\widehat{e} = Te$
Orientate the simplex Δ_r so that the median axis q is aligned with the direction $-\widehat{e}$
$\Delta = T^{-1}\widehat{\Delta}$
Return Δ

Note that if the Hessian matrices are constant, we can reuse the curvature tensor matrix and the transformation matrix T for the new domain construction if invoked in the next iterations.

Experimental result: a biological system

We implemented the above algorithm using the algorithms in [**?**] for reachability computation for affine approximate systems. We used this implementation to study proteolytic reactions involving Type 1 Collagen that occur in the extracellular matrix using a theorical model of the biochemical network [**?**], which is a differential equation system with 12 variables (see Appendix). The numerical values of parameters are given in Appendix. The vector field of this system is quadratic and therefore its Hessian matrices are constant.

This system describes the type I Collagen Proteolysis initiated by the concentrations MT1 and M2 of MT1-MMP and MMP2 enzymes. The dynamics of the system is quadratic, and its directional curvature varies a lot depending on the directions. The application of this new simplex construction allows to not only obtain a smaller approximate reachable set (due to a smaller error bound used in the input set) and more over significantly reduce the computation time by roughly 2.5 for the same time horizon, compared to the previous method described in [**?**].

Figure 2.2.7 shows the projection of the reachable set evolution on the first three variables, namely $mt1$ and $m2$. The initial set is a small set around the origin. We observe that the variables converge towards some steady values (inside the dense part of the reachable set shown in the figure).

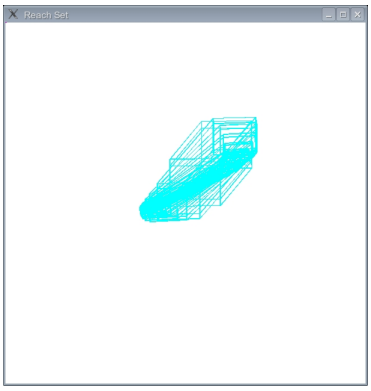


Figure 2.6: Projection of the reachable set on the first three variables $mt1$, $m2$ and $t2$.

2.3 Other results

2.3.1 Algebraic differential equations

The behavior of many physical systems, such as non-linear analog circuits, can be described by a set of differential algebraic equations. The extension of reachability techniques for ordinary differential equations (ODEs) to handle DAEs is not straightforward since these classes of equations differ in both theoretical and numerical properties, and this is captured by the *index* concept. The differential index of (**??**) is the minimal number of differentiations required to solve for the derivatives $\dot{x}$. In general the problem of numerically solving DAEs with index 2 or higher is ill-posed. DAEs that model practical electronic circuits are typically of index 1 or 2 and in this work we focus on the former. In particular, we will study the equivalent semi-explicit form of (**??**):

$$\dot{x}(t) = f(x(t), y(t), p), \tag{2.8}$$

$$0 = g(x(t), y(t), p). \tag{2.9}$$

If the Jacobian $g_y(x, y) = \partial g/\partial y$ is invertible in a neighborhood of the solution, then by differentiating the algebraic equation we obtain

$$\dot{y} = -g_y^{-1} g_x f, \tag{2.10}$$

and in this case, the DAE system is of index 1. A trivial way to compute reachable sets for index 1 DAEs is to transform it into an ODE composed of (2.8) and (2.10) using the above-described differentiation and then apply the existing techniques for ODEs. However, the drawback of this approach is that the solution may drift away from the algebraic constraint. We will retain the algebraic constraint (2.9) and interpret the original DAE as the ODE, composed of (2.8) and (2.10), on the manifold defined by (2.9). The main idea of our method is to combine the commonly-used technique of geometric integration using projection [**?**], with our reachability algorithm, to compute the reachable set.

The approach was applied to a number of examples, in particular a second order biquad low-pass filter circuit. These results were published in the paper [37]. To our knowledge, the reachability problem for DAEs had not been addressed before.

2.3.2 Predicate Abstraction

Predicate abstraction has emerged to be a powerful technique for extracting finite-state models from infinite-state discrete programs. In this work we address the reachability analysis problem for hybrid systems by combining the notion of predicate abstraction with our techniques for approximating the set of reachable states of linear systems using polyhedra [**?**]. Given a hybrid system and a set of user-defined boolean predicates, we consider the finite discrete quotient whose states correspond to all possible truth assignments to the input predicates. The algorithm performs an on-the-fly exploration of the abstract system.

To compute the transitions out of an abstract state, the tool needs to compute the set of discrete and continuous successors, and find out all the abstract states that this set intersects with. The complexity of this computation grows exponentially with the number of abstraction predicates. We developed various optimizations that are aimed at speeding up the search in the abstract state-space, and demonstrate their benefits via case studies. We also studied the completeness of the predicate abstraction technique for proving safety of hybrid systems.

The success of this approach however crucially depends on the choice of the predicates used for abstraction. We thus focus on identifying these predicates automatically by analyzing spurious counter-examples generated by the search in the abstract state-space. We present the basic techniques for discovering new predicates that will rule out closely related spurious counter-examples, optimizations of these techniques, implementation of these in the verification tool, and case studies demonstrating the promise of the approach.

This work was done in collaboration with Rajeev Alur and Franjo Ivancic from University of Pennsylvania. The results were published in the conferences HSCC

2003 - Hybrid Systems: Computation and Control [**?**], TACAS 2003 [**?**], HSCC 2004 - Hybrid Systems: Computation and Control [**?**] and two journal articles TECS [**?**] and TCS [**?**].

2.3.3 Abstraction by projection

In this work, we proposed an abstraction method for dimension reduction, which is along the lines of the hybridization-based approach. Our first observation is that in many practical systems, the properties to verify involve only a subset of variables, and the other variables may not need to be analyzed with great accuracy. The main idea of our method is to project out some continuous variables, say z, and treat them in the dynamics of the remaining variables x as uncertain input. Therefore, the dynamics of x is then described by a differential inclusion. In addition, in order to avoid excessively conservative abstractions, the domains of the projected variables are divided into smaller regions corresponding to different differential inclusions. The final result of our abstraction procedure is a hybrid system of lower dimension with some important properties that guarantee convergence results.

However, this abstraction method does not solve the verification problem by itself. The success depends on the ability to deal with differential inclusions. We thus focused on the reachability problem for uncertain bilinear systems, a simple yet useful class of nonlinear differential inclusions. The combination of the abstraction method and the reachability analysis method for bilinear systems allows to treat multi-affine systems, which can be found in numerous applications in engineering, biology and economics.

This work was done in collaboration with Eugene Asarin from LIAFA, which resulted in a publication in the conference HSCC 2004 - Hybrid Systems: Computation and Control [**?**].

Chapter 3

Reachability analysis of polynomial systems

3.1 Introduction

Polynomial systems can be used to model a variety of physical phenomena, in particular the dynamics of bio-chemical networks. Using the set integration idea, we first derive an integration scheme that approximates the reachable state $\mathbf{x}_{k+1}$ by applying some polynomial map to $\mathbf{x}_k$. In order to use this scheme to approximate the reachable set, we then consider the problem of computing the image of a set by a multivariate polynomial.

Numerical integration is a common method to solve non-linear differential equations. Its goal is to derive a scheme to approximate the solution at each time step based on the solution at one or several previous steps. In general, a typical numerical integration scheme can be written as: $\mathbf{x}_{k+1} = \mathcal{Y}_k(f, h, \mathbf{x}_0, \mathbf{x}_1, \ldots, \mathbf{x}_k)$ where f is the derivative and h is the step size. Nevertheless, while this approach is concerned with computing a single solution at a time and each $\mathbf{x}_k$ here is a point, in reachability analysis one has to deal with sets of all possible solutions (due to non-determinism in initial conditions and in the dynamics of the system). Therefore, wishing to exploit the numerical integration idea for reachable set computation purposes, a question that arises is how to perform such schemes with sets, that is, when each $\mathbf{x}_k$ is a set of points. The essence behind the approach we propose can be described as extending traditional numerical integration to set integration.

In this chapter, we address the following image computation problem: given a set in $\mathbb{R}^n$, compute its image by a polynomial. This problem typically arises when

we deal with a dynamical system of the form $x(k+1) = \pi(x(k))$ where π is a multivariate polynomial. Such a dynamical system could result from a numerical approximation of a continuous or hybrid system. Many existing reachability computation methods for continuous systems can be seen as an extension of numerical integration. For reachability analysis which requires considering all possible solutions (for example, due to non-determinism in initial conditions), one has to solve the above equation with sets, that is $x(k)$ and $x(k+1)$ in the equation are subsets of $\mathbb{R}^n$ (while they are points if we only need a single solution, as in numerical integration). In addition, similar equations can arise in embedded control systems, such as some physical system controlled by a computer program, which is the implementation of some continuous (or possibly hybrid) controller using appropriate discretization.

The image computation problem we are interested in can be formally stated as follows. Given a polynomial map $\pi : \mathbb{R}^n \to \mathbb{R}^n$ of total degree d and a bounded set $X \subseteq \mathbb{R}^n$, we want to compute the image $\pi(X)$ defined as: $\pi(X) = \{\pi(\mathbf{x}) \mid \mathbf{x} \in X\}$.

It should be noted that our method can be extended to a continuous-time system described by a differential equation. Indeed, one can approximate it by a difference equation using a discretization scheme with some uncertainty term to guarantee conservativeness of the approximation.

Notation

Let $\mathbb{R}$ denote the set of reals. Vectors are often written using bold letters. Exceptionally, scalar elements of multi-indices, introduced later, are written using bold letters. Given a vector $\mathbf{x}$, x_i denotes its i^{th} component. Capital letters, such as A, B, X, Y, denote matrices or sets. If A is a matrix, A^i denotes the i^{th} row of A. An affine function is thus represented as $\mathbf{c}^T\mathbf{x} + \mathbf{d}$.

We use B_u to denote the unit box $B_u = [0,1]^n$. We use π to denote a vector of n functions such that for all $i \in \{1, \ldots, n\}$, π_i is an n-variate polynomial of the form $\pi_i : \mathbb{R}^n \to \mathbb{R}$. In the remainder of the paper, we sometimes refer to π simply as "a polynomial".

To discuss Bézier simplices and the Bernstein expansion, we use multi-indices of the form $\mathbf{i} = (\mathbf{i}_1, \ldots, \mathbf{i}_n)$ where each $\mathbf{i}_j$ is a non-negative integer. Given two multi-indices $\mathbf{i}$ and $\mathbf{w}$, we write $\mathbf{i} \leq \mathbf{w}$ if for all $j \in \{1, \ldots, n\}$, $\mathbf{i}_j \leq \mathbf{w}_j$. Also, we write $\frac{\mathbf{i}}{\mathbf{w}}$ for $(\frac{\mathbf{i}_1}{\mathbf{w}_1}, \ldots, \frac{\mathbf{i}_n}{\mathbf{w}_n})$ and $\binom{\mathbf{i}}{\mathbf{w}}$ for $\binom{\mathbf{i}_1}{\mathbf{w}_1}\binom{\mathbf{i}_2}{\mathbf{w}_2}\ldots\binom{\mathbf{i}_n}{\mathbf{w}_n}$.

We define the norm of $\mathbf{i}$ by $||\mathbf{i}|| = \sum_{j=1}^{n+1} \mathbf{i}[j]$ and let $\mathcal{I}_n^d$ denote the set of all multi-indices $\mathbf{i} = (\mathbf{i}[1], \ldots, \mathbf{i}[n+1])$ with $||\mathbf{i}|| = d$. We define two special multi-indices: $\mathbf{e}_j$ is a multi-index that has all the components equal to 0 except for the j^{th} component which is equal to 1, and $\mathbf{o}$ is a multi-index that has all the components equal to 0. We call $\mathbf{o}$ the zero multi-index.

3.2 Set integration

We first show a method for obtaining a numerical scheme suitable for our reachability computation purposes. Indeed, we would like a scheme which can be effectively computed for polynomial systems. The development is however valid for general non-linear systems.

3.2.1 Numerical scheme

We consider a non-linear system:

$$\dot{\mathbf{x}}(t) = g(\mathbf{x}(t)). \tag{3.1}$$

We first rewrite the dynamics of the system as the sum of a linear part $A\mathbf{x}(t)$ and a non-linear part $f(\mathbf{x}(t))$, that is,

$$\dot{\mathbf{x}}(t) = g(\mathbf{x}(t)) = A\mathbf{x}(t) + f(\mathbf{x}(t)). \tag{3.2}$$

We then consider the non-linear term as independent input. In other words, the system is treated as a linear system with input $f(\mathbf{x}(t))$. This trick is to separate the linear part for which we can derive the exact closed-form solution. The interest in doing so will become clearer when we discuss the approximation error. We now develop a numerical solution for (3.2). Let $h > 0$ be a time step and $t_k = kh$ where $k = 0, 1, 2, \ldots$. Then, we have

$$\mathbf{x}(t_{k+1}) = e^{Ah}\mathbf{x}(t_k) + \int_0^h e^{A(h-\tau)} f(\mathbf{x}(t_k + \tau))\, d\tau. \tag{3.3}$$

The idea is to approximate $\mathbf{x}(t_k+\tau)$ in the above equation by its Taylor expansion around t_k to the first order, that is $\boldsymbol{\alpha}(t_k + \tau) = \mathbf{x}(t_k) + g(\mathbf{x}(t_k))\tau$. Denoting $\mathbf{x}(t_k) = \mathbf{x}_k$, $f(\mathbf{x}(t_k)) = f_k$ and $g(\mathbf{x}(t_k)) = g_k$, we have $\boldsymbol{\alpha}(t_k + \tau) = \mathbf{x}_k + g_k\tau = \mathbf{x}_k + (Ax_k + f_k)\tau$. Replacing $\mathbf{x}(t_k+\tau)$ with $\boldsymbol{\alpha}(t_k+\tau)$, we obtain an approximation $\bar{\mathbf{x}}_{k+1}$ of the exact solution $\mathbf{x}_{k+1}$:

$$\bar{\mathbf{x}}_{k+1} = e^{Ah}\mathbf{x}_k + \int_0^h e^{A(h-\tau)} f(\boldsymbol{\alpha}(t_k + \tau))\, d\tau. \tag{3.4}$$

The integral in the above equation is a function of $\mathbf{x}_k$, and we denote it by

$$Q(\mathbf{x}_k) = \int_0^h e^{A(h-\tau)} f(\boldsymbol{\alpha}(t_k + \tau))\, d\tau$$

.

We now focus on the case where $g(\mathbf{x})$ is a polynomial.

Lemma 4. *If $g(\mathbf{x})$ is a multi-variate polynomial, the map $Q(\mathbf{x}_k)$ can be written as a polynomial in $\mathbf{x}_k$.*

Proof. The proof of the lemma is straightforward, however we present it here for the clarity of the development that follows. It is easy to see that if the total degree of $f(\mathbf{x}_k)$ is d, then $\boldsymbol{\alpha}(t_k+\tau)$ is a multivariate polynomial of total degree d in $\mathbf{x}_k$, and therefore $f(\boldsymbol{\alpha}(t_k+\tau))$ is a polynomial of degree d in τ. We can write $f(\boldsymbol{\alpha}(t_k+\tau)) = \sum_{l=0}^{d} \psi_l(\mathbf{x}_k)\tau^l$ where for every $l \in \{0,1,\ldots,d\}$ ψ_l is a polynomial in $\mathbf{x}_k$. We then denote $\Gamma_l = \int_0^h e^{A(h-\tau)}\tau^l \, d\tau$, which can be written in a closed form. It then follows that $\int_0^h e^{A(h-\tau)} f(\boldsymbol{\alpha}(t_k+\tau))\, d\tau = \sum_{l=0}^{d} \Gamma_l \psi_l(\mathbf{x}_k)$. ■ ■

The resulting integration scheme to approximate the solution of (3.1) is:

$$\begin{cases} \bar{\mathbf{x}}_{k+1} &= e^{Ah}\bar{\mathbf{x}}_k + Q(\bar{\mathbf{x}}_k) = P(\bar{\mathbf{x}}_k), \\ \bar{\mathbf{x}}_0 &= \mathbf{x}(0). \end{cases}$$

We call $P(\mathbf{x}_k)$ the *integration map.*

Image computation problem

The problem we are now interested in can thus be formally stated as follows.

Problem 1. *Given a polynomial map $\pi : \mathbb{R}^n \to \mathbb{R}^n$ of total degree d and a bounded set $X \subseteq \mathbb{R}^n$, we want to compute the image $\pi(X)$ defined as: $\pi(X) = \{\pi(\mathbf{x}) \mid \mathbf{x} \in X\}$. We will focus on the case where X is a simplex in $\mathbb{R}^n$.*

Example of multi-affine systems

Let us illustrate the proof with a simple case where $g(\mathbf{x})$ is a multi-affine function of degree 2. This is the case of a biological model we study in Section 3.3.3. The function $f(\mathbf{x})$ can be written as: $f(\mathbf{x}) = \sum_{i,j\in\{1,\ldots,n\}, i\neq j} \mathbf{x}[i]\mathbf{x}[j]\boldsymbol{c}_{ij}$ with $\boldsymbol{c}_{ij} \in \mathbb{R}^n$. Then, replacing $\mathbf{x}(t_k+\tau)$ with $\boldsymbol{\alpha}(t_k+\tau) = \mathbf{x}_k + g_k\tau$, we have:

$$f(\boldsymbol{\alpha}(t_k+\tau)) \;=\; \sum_{i\neq j\in\{1,\ldots,n\}} (g_k[i]g_k[j]\tau^2 + (\mathbf{x}_k[i]g_k[j] + g_k[i]\mathbf{x}_k[j])\tau + \mathbf{x}_k[i]\mathbf{x}_k[j])\boldsymbol{c}_{ij}$$

Therefore, the equation (3.4) becomes:

$$\bar{\mathbf{x}}_{k+1} = P(\mathbf{x}_k) = \Phi\mathbf{x}_k + \sum_{i\neq j\in\{1,\ldots,n\}} (\gamma_2\Gamma_2 + \gamma_1\Gamma_1 + \gamma_0\Gamma_0)\boldsymbol{c}_{ij}. \tag{3.5}$$

where $\Phi = e^{Ah}$ and $\gamma_2 = g_k[i]g_k[j]$, $\gamma_1 = g_k[i]\mathbf{x}_k[j] + \mathbf{x}_k[i]g_k[j]$, $\gamma_0 = \mathbf{x}_k[i]\mathbf{x}_k[j]$. After straightforward calculations, we obtain:

$$\Gamma_l = l!\sum_{i=0}^{\infty} \frac{A^i h^{i+l+1}}{(i+l+1)!} \tag{3.6}$$

It is thus easy to see that, due to the term γ_2, $P(\mathbf{x}_k)$ in (3.5) is a polynomial of degree 4 in $\mathbf{x}_k$. The equation (3.5) can be readily used as a scheme specialized for multi-affine systems of degree 2.

3.2.2 Convergence

A bound on the error in our approximation is given in the following theorem.

Theorem 5. *Let $\bar{\mathbf{x}}(t_{k+1})$ be the approximate solution at time t_{k+1} (computed by (3.4)) and $\mathbf{x}(\cdot)$ be the corresponding exact solution such that $\bar{\mathbf{x}}(t_k) = \mathbf{x}(t_k)$. Then, a bound on the local error is given by: $||\bar{\mathbf{x}}(t_{k+1}) - \mathbf{x}(t_{k+1})|| = \mathcal{O}(h^3)$.*

The proof of this result is presented in Appendix. This theorem shows that the equation (3.4) is a *second order scheme*. In addition. we can show that the global error is also convergent. As one can see from the proof, the error bound depends on the Lipschitz constant of the non-linear function f. So now we can see the interest in separating the linear part since the Lipschitz constant of f is smaller than that of g.

Higher order integration schemes

Note that we have used an approximation of the exact solution $\mathbf{x}(t_k + \tau)$ by the its first order Taylor expansion around t_k. To obtain better convergence orders, we can use higher order expansions which results in integration schemes involving high order derivatives of $f(\mathbf{x})$. The derivation of such schemes is similar to the above development, but the degree of the resulting integration map $P(\mathbf{x}_k)$ can be higher. In the other direction, if we use a simpler approximation $\boldsymbol{\alpha}(t_k + \tau) = \mathbf{x}_k$ for all $\tau \in [t_k, t_{k+1})$, then $Q(\mathbf{x}_k) = \Gamma_0 f(\mathbf{x}_k)$ and we obtain the classic Euler scheme for the non-linear part. The advantage of this scheme is that the resulting polynomial $Q(\mathbf{x}_k)$ has the same degree as $f(\mathbf{x})$. As we will see later, the degree of the integration map is one of the factors determining the complexity of the reachability algorithm. It remains to compute the polynomial map $Q(\mathbf{x}_k)$, the problem we tackle in the next section.

3.3 Using Bézier techniques

To address the image computation problem for polynomial maps, we employ the techniques from computer aided geometric design, in particular the Bézier techniques and the blossoming principle. We also prove that our overall method is of order 2. Although this paper focuses on continuous systems, the proposed method can be extended to hybrid systems, since reachable sets are represented

by convex polyhedra, and Boolean operations (required to deal with discrete transitions) over such polyhedra can be computed using a variety of existing algorithms. This is illustrated through a biological example.

3.3.1 Bézier simplices

Let Δ be a full-dimensional simplex in $\mathbb{R}^n$ with vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_{n+1}\}$. Given a point $\mathbf{x} \in \Delta$, let $\lambda(\mathbf{x}) = (\lambda_1(\mathbf{x}), \ldots, \lambda_{n+1}(\mathbf{x}))$ be the function that gives the barycentric coordinates of $\mathbf{x}$ with respect to the vertices of Δ, that is, $\mathbf{x} = \sum_{j=1}^{n+1} \lambda_j(\mathbf{x})\mathbf{v}_j$ and $\sum_{j=1}^{n+1} \lambda_j(\mathbf{x}) = 1$.

A *Bézier simplex* of degree d of the form $\pi : \mathbb{R}^n \to \mathbb{R}^n$ is defined as[1]:

$$\pi(\mathbf{x}) = \sum_{||\mathbf{i}||=d} \mathbf{b}_\mathbf{i} B^d_\mathbf{i}(\lambda_1(\mathbf{x}), \ldots, \lambda_{n+1}(\mathbf{x})) \tag{3.7}$$

where for a given multi-index $\mathbf{i}$, $\mathbf{b}_\mathbf{i}$ is a vector in $\mathbb{R}^n$ and $B^d_\mathbf{i} : \mathbb{R}^n \to \mathbb{R}$ is a *Bernstein polynomial* of degree d defined as:

$$B^d_\mathbf{i}(y_1, \ldots, y_{n+1}) = \binom{d}{\mathbf{i}} y_1^{\mathbf{i}[1]} y_2^{\mathbf{i}[2]} \ldots y_{n+1}^{\mathbf{i}[n+1]} \tag{3.8}$$

with the multimonial coefficient

$$\binom{d}{\mathbf{i}} = \frac{d!}{\mathbf{i}[1]!\,\mathbf{i}[2]!\,\ldots\,\mathbf{i}[n+1]!}.$$

In the above formula (3.7), each vector $\mathbf{b}_\mathbf{i}$ is called a *Bézier control point* and the set of all such $\mathbf{b}_\mathbf{i}$ form the *Bézier control net* of π with respect to Δ.

Any polynomial can be written in form of a Bézier simplex, as in formula (3.7). This form is a popular way to write polynomials in computer aided geometric design (see [**?**] and references therein).

The following properties of Bernstein polynomials are well-known.

- The Bernstein polynomials form a partition of unity, that is,

$$\sum_{||\mathbf{i}||=d} B^d_\mathbf{i}(y_1, \ldots, y_{n+1}) = 1.$$

- They are non-negative, that is,

$$B^d_\mathbf{i}(y_1, \ldots, y_{n+1}) \geq 0 \text{ for all } 0 \leq y_1, \ldots, y_{n+1} \leq 1.$$

[1]The definition holds for more general polynomials of the form $\pi : \mathbb{R}^n \to \mathbb{R}^m$.

The above properties of Bernstein polynomials imply the following *shape properties* of Bézier simplices, which we will use for reachability computation purposes.

Lemma 5. *Given an arbitrary point* $\mathbf{x} \in \Delta$*,*

1. [**Convex hull property**] *the point* $\pi(\mathbf{x})$ *lies inside the convex hull of the control net, that is* $\pi(\mathbf{x}) \in conv\{\mathbf{b_i} \mid \mathbf{i} \in \mathcal{I}_n^d\}$.
2. [**End-point interpolation property**] π *interpolates the control net at the corner control points specified by* $\mathbf{b}_{d\mathbf{e}_k}$ *for all* $k \in \{1, \ldots, n+1\}$.

Note that the number of multi-indices in $\mathcal{I}_k^d$ is $\binom{d+n}{n}$; therefore, the number of points $\mathbf{b_i}$ is exactly $\binom{d+n}{n} = \frac{(d+n)!}{d!\, n!}$. We denote this number by $\beta(n, d)$.

These shape properties can be used to approximate polynomial maps. Indeed, the *convex hull property* in Lemma 5 shows that one can over-approximate $\pi(\Delta)$ by taking the convex hull of the Bézier control net of π with respect to Δ. In addition, this *over-approximation is tight* due to the above *end-point interpolation property*. In the rest of this section we focus on the problem of computing the Bézier control net of the polynomial π. To avoid confusion, it is worthy to emphasize that for reachability computation purposes, we are dealing with the systems whose vector fields are given in monomial form (i.e. sums of monomials), hence the integration map is also defined in this form. To compute the control points of a polynomial given in monomial forms, we will exploit the techniques for approximating and designing polynomial curves and surfaces. However, it is important to mention that most of such existing tools deal with univariate or bivariate polynomials (often expressed in terms of control points), their application to solve our problem requires an adaptation to multivariate polynomials as well as geometric manipulation in general dimension.

3.3.2 Computing the Bézier control net

Our goal is to obtain the Bézier control net of a polynomial π given in monomial form. By the definition (3.7), the most natural approach is to solve the following interpolation problem. Let S be a set of $\beta(n, d)$ points in Δ. For each $\mathbf{x} \in S$, we evaluate $\pi(\mathbf{x})$ and use (3.7) to obtain a system of linear equations with the coordinates of the Bézier control points $\mathbf{b_i}$ as unknown variables. One can choose the set S such that the unique solution to these linear equations exists [**?**]. Although this method is conceptually simple, it may require solving a large linear system[2] (which is of size $n * \beta(n, d)$). We will use a more efficient approach based

[2]The Gaussian elimination algorithm to solve a linear system of size $m \times m$ has the time complexity $O(m^3)$.

on the blossoming principle, which is summarized in the following theorem. A thorough description of this principle and its various applications can be found in [**?**, 96].

Theorem 6 (Blossoming principle)**.** *For any polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ of degree d, there is a unique symmetric d-affine map $p : (\mathbb{R}^n)^d \to \mathbb{R}^n$ such that for all $\mathbf{x} \in \mathbb{R}^n$ $p(\mathbf{x}, \ldots, \mathbf{x}) = \pi(\mathbf{x})$. The map p is called the* blossom *or the* polar form *of π.*

We recall that a map $q(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ is called *d-affine* if it is affine when all but one of its arguments are kept fixed; it is said to be symmetric if its value does not depend on the ordering of the arguments, that is, for any permutation $(\mathbf{y}_1, \ldots, \mathbf{y}_d)$ of $(\mathbf{x}_1, \ldots, \mathbf{x}_d)$ we have $q(\mathbf{y}_1, \ldots, \mathbf{y}_d) = q(\mathbf{x}_1, \ldots, \mathbf{x}_d)$. Given a polynomial π, the connection between its Bézier control net relative to a simplex Δ and its blossom p is described by the following lemma.

Lemma 6. *For all* $\mathbf{i} \in \mathcal{I}_n^d$, $\mathbf{b}_\mathbf{i} = p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \underbrace{\mathbf{v}_2, \ldots, \mathbf{v}_2}_{\mathbf{i}[2]}, \ \ldots, \ \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]})$ *where $\{\mathbf{v}_1, \ldots, \mathbf{v}_{n+1}\}$ are the vertices of Δ.*

This fact is also well-known [**?**], and we present its proof in Appendix, which can facilitate understanding subsequent development.

Computing the blossom

We have seen that the Bézier control points can be computed by evaluating the blossom values at some particular points shown in Lemma 6. To compute them, we first derive an analytic expression of the polar form and then show how to compute this expression efficiently. We do so by extending the results for bivariate polynomial surfaces [**?**] to multivariate polynomials.

Before proceeding, we mention that the problem of computing the Bézier control net can be formulated as a problem of changing from the monomial basis to the Bézier basis, which can be solved using the algorithms proposed in [**?**, **?**]. These algorithms also make use of the blossoming principle. The idea is to express the coordinates of the new basis vectors in the old basis, and then apply the transformation matrix to the old coefficients. However, when the polynomial representation is "sparse", that is it contains many zero coefficients, this sparsity is not exploited. The method discussed in the following deals better with such sparsity since it considers only the monomials with non-null coefficients. More precisely, by "sparse polynomial representations" we mean those where the number of monomials (with non-null coefficients) is much smaller than the number of all combinations of coordinate variables up to degree d. The sparse case happens in many practical applications we have encountered.

We now show how to compute the blossom of monomials which are products of only two variables, such as $\mathbf{x}[i]^h\mathbf{x}[j]^k$. Similar treatment can be used for monomials involving more variables, but due to the length of the involved formulas we do not detail it here. On the other hand, using linearity, we can obtain the blossom of any polynomial expressed as a sum of monomials.

The blossom of degree d of the monomial $(\mathbf{x}[i])^h(\mathbf{x}[j])^k$ is given by:

$$p^d_{h,k}(\mathbf{u}_1,\mathbf{u}_2,\ldots,\mathbf{u}_d) = \frac{1}{\binom{d}{h}\binom{d-h}{k}} \sum_{\substack{I\cup J\subseteq\{1,\ldots,d\},\\ |I|=h,|J|=k,I\cap J=\emptyset}} \prod_{r\in I}\mathbf{u}_r[i]\prod_{s\in J}\mathbf{u}_s[j].$$

To prove this, it suffices to check that the right hand side is a symmetric multi-affine function, and moreover $p^d_{h,k}(\mathbf{u},\mathbf{u},\ldots,\mathbf{u}) = (\mathbf{u}[i])^h(\mathbf{u}[j])^k$. ∎

To compute the blossom values using the above expression, we make use of a recurrence equation on p, as proposed in [**?**]. We first denote

$$\sigma^d_{h,k} = \frac{1}{\binom{d}{h}\binom{d-h}{k}} p^d_{h,k}(\mathbf{u}_1,\mathbf{u}_2,\ldots,\mathbf{u}_d).$$

The function σ is symmetric and has the following interpretation: this function is computed by choosing h i^{th} coordinates of the argument points and k i^{th} coordinates and forming their product, then summing these products over all possible choices. We can thus derive the following recurrence formula:

$$\begin{cases} \sigma^d_{h,k} &= \sigma^{d-1}_{h,k} + \mathbf{u}_d[i]\sigma^{d-1}_{h-1,k} + \mathbf{u}_d[j]\sigma^{d-1}_{h,k-1} \quad \text{if } h,k\geq 0 \text{ and } h+k\geq 1,\\ \sigma^d_{0,0} &= 0 \end{cases} \tag{3.9}$$

This means that to compute the required blossom value $p^d_{h,k}(\mathbf{u}_1,\mathbf{u}_2,\ldots,\mathbf{u}_d)$ we compute all the intermediate values $p^{d'}_{h',k'}(\mathbf{u}_1,\ldots,\mathbf{u}_{d'})$ with $d'\leq d$, $h'+k'\leq d'$. This computation can be done in time $O(d^3)$.

3.3.3 Approximation error and subdivision

We now estimate an error bound for the approximation of the polynomial map π by its the Bézier control points.

Theorem 7. *For each Bézier control point* $\mathbf{b_i}$ *there exists a point* $\mathbf{y}\in\pi(\Delta)$ *such that* $||\mathbf{b_i}-\mathbf{y}||\leq K\rho^2$ *where* ρ *be the maximal side length of* Δ *and* K *is some constant not depending on* Δ.

The proof of this theorem can be found in Appendix.

Consequently, when the simplicial domain Δ is large, to achieve the desired accuracy we may need to subdivide it into smaller simplices. This subdivision creates new Bézier bases and therefore new control points. However, due to the properties of multi-affine maps, one can compute the new control nets in a clever way which reuses the computations performed for the original simplex. Suppose that we want to partition the simplex Δ by adding a point $\mathbf{x} \in \Delta$ and forming $(n+1)$ new smaller simplices. Then, we can use de Catesljau algorithm [**?**, **?**] to compute the value of the polynomial π at $\mathbf{x}$. It turns out that this computation also produces the control net for the new simplices. Note that this algorithm can only be applied when the Bézier control points of the polynomial are known.

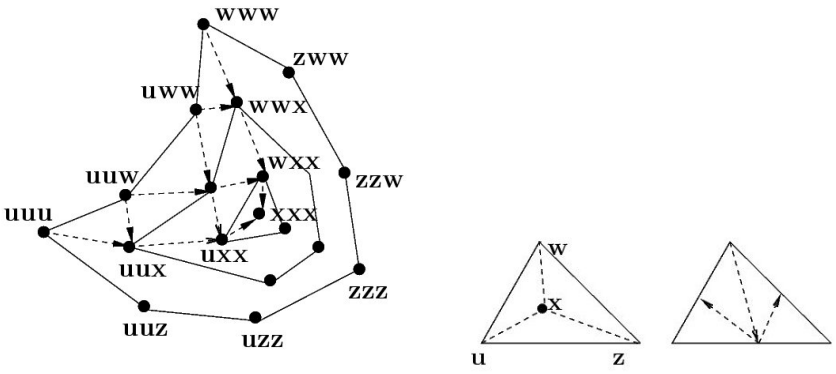


Figure 3.1: Subdividing a Bézier control net

We denote $\mathbf{b}^l_{\mathbf{i}} = p(\underbrace{\mathbf{v}_1, \ldots, \mathbf{v}_1}_{\mathbf{i}[1]}, \ldots, \underbrace{\mathbf{v}_{n+1}, \ldots, \mathbf{v}_{n+1}}_{\mathbf{i}[n+1]}, \underbrace{\mathbf{x}_1, \ldots, \mathbf{x}_l}_{l})$ with $\mathbf{i}[1] + \ldots + \mathbf{i}[n+1] + l = d$. Since p is symmetric and multi-affine, we have:

$$\mathbf{b}^l_{\mathbf{i}} = \lambda_1(\mathbf{x}_l)\mathbf{b}^{l-1}_{\mathbf{i}+\mathbf{e}_1} + \ldots + \lambda_n(\mathbf{x}_l)\mathbf{b}^{l-1}_{\mathbf{i}+\mathbf{e}_n} \tag{3.10}$$

Note that $\mathbf{b}^n_{\mathbf{o}} = p(\mathbf{x}_1, \ldots, \mathbf{x}_n)$ where $\mathbf{o}$ is the zero multi-index. In addition, with $l = 0$, $\mathbf{b}^0_{\mathbf{i}}$ are exactly the Bézier control points of the polynomial. Therefore, by running the above recursion starting from $l = 0$ until $l = n$ we obtain the blossom value at $(\mathbf{x}_1, \ldots, \mathbf{x}_n)$. If all the argument points of the blossom are equal to $\mathbf{x}$, the result of the algorithm is $\pi(\mathbf{x})$. The de Catesljau algorithm is illustrated with a 2-dimensional example in Figure 3.1 where each node is annotated with the arguments of the blossom to evaluate. The nodes on the outermost layer correspond to the control points for the original triangle **uzw**. The incoming arrows of **uux** show that the blossom value at this point is computed from the blossom values at **uuu** and **uuw**. As mentioned earlier, we can see that the computation of $\pi(\mathbf{x})$ indeed produces the Bézier control points for the sub-simplices. Figure 3.1 shows the values $p(\underbrace{\mathbf{u}, \ldots, \mathbf{u}}_{\mathbf{i}[1]}, \underbrace{\mathbf{x}, \ldots, \mathbf{x}}_{\mathbf{i}[2]}, \underbrace{\mathbf{w}, \ldots, \mathbf{w}}_{\mathbf{i}[3]})$ which are the Bézier control points for the triangle **uxw**.

One important remark is that the subdivision at the center of the simplex does not reduce the maximal side length of the simplices. By Theorem 7 this means that the convergence of the Bézier control net towards the polynomial is not guaranteed. However, one can repeat the bisection at the mid-point of the logest edge, as shown in Figure 3.1 to achieve the desired accuracy. More generally, the subdivision of a simplex can be defined as follows. For each barycentric coordinate $\lambda_i(\mathbf{x}) > 0$ of a point $\mathbf{x} \in \Delta$ we define a simplex Δ_i obtained from Δ by replacing the vertex $\mathbf{v}_i$ with $\mathbf{x}$. Hence, when the point $\mathbf{x}$ is the mid-point of an edge we obtain a bisection. It was proved in [**?**] that using the bisection at the mid-point of the longest edge, after n steps (where n is the dimension of the simplex) the simplex diameter is reduced at least by $\sqrt{3}/2$ times. In two dimensions, another method of subdivision via all the mid-points of the edges was discussed in [**?**]. This method is however more complex to implement for dimensions higher than 2.

Image computation algorithm

We are now ready to describe the image computation algorithm for polynomial systems. In Algorithm 2, X_0 is the initial set which is assumed to be a convex polyhedron in $\mathbb{R}^n$, each R_k is a set of convex polyhedra. The function Bez over-approximates the image of a simplex Δ by the integration map P, using the method presented in the previous section. The goal of the function *triangulation* triangulates a set of convex polyhedra and returns the set of all simplices of the triangulation. To do so, we collect all the vertices of the polyhedra and compute a triangulation of this set. We then exclude all the simplices in the triagulation whose interior does not intersect with R_k.

Algorithm 2 Image computation

$R_0 = X_0$, $k = 0$
repeat
  $S_\Delta = triangulation(R_k)$
  $C = \emptyset$
  for all $\Delta \in S_\Delta$ **do**
    $C = C \cup Bez(\Delta)$
  end for
  $R_{k+1} = C$
  $k = k + 1$
until $R_{k+1} = R_k$

Precision. Let us briefly discuss the precision of the algorithm. We suppose that ρ is the maximal size of the simplices that are produced by the function *triangulation* and h is the integration time step. If the integration map P can be exactly computed, using Theorem 5, we know that the integration error is $\mathcal{O}(h^3)$. In addition, Theorem 7 shows that our approximation of the integration map P induces an error $\mathcal{O}(\rho^2)$. By the triangle inequality, the total error in each iteration of Algorithm 2 is bounded by $(\mathcal{O}(h^3) + \mathcal{O}(\rho^2))$. Therefore, by choosing appropriate values ρ in function of h, we can guarantee that Algorithm 2 is a second order method.

We now discuss some computation issues. The first remark is that the total number of the Bézier control points is $\beta(n, d)$, but the actual number of vertices of their convex hull (that is, $Bez(\Delta)$) is often much smaller, depending on the geometric structure of the polynomial map P. On the other hand, in order to speed up the computation (at the price of less precise results), one can approximate C by its convex hull or even by a simplex. Algorithms for doing so have been developed and some algorithms can compute a minimal volume enclosing simplex (such as, [**?**, **?**]).

Example: a biological system

We have implemented Algorithm 2 and applied it to two simplified models of a well-known biological system, namely the system of gene transcription control in the bacteria Vibrio Fisheri (see [22, 21] for a detailed description of the models and the related gene control problems). The first model corresponds to one mode of a simplified hybrid system where the continuous dynamics is described by the following multi-affine system:

$$\begin{cases} \dot{x_1} &= k_2 x_2 - k_1 x_1 x_3 + u_1 \\ \dot{x_2} &= k_1 x_1 x_3 - k_2 x_2 \\ \dot{x_3} &= k_2 x_2 - k_1 x_1 x_3 - n x_3 + n u_2 \end{cases} \tag{3.11}$$

The state variables $\mathbf{x} = (x_1, x_2, x_3)$ represent the cellular concentrations of different species, and the parameters k_1, k_2, n are the binding, dissociation and diffusion constants. The variables u_1 and u_2 are control variables, which respectively represent the plasmid and external source of autoinducer. In [21] the following control law for steering all the states in the rectangle $[1, 2] \times [1, 2] \times [1, 2]$ to the face $x_2 = 2$ was proposed: $u_1(\mathbf{x}) = -10(x_2 + x_1(-1 + 3) - 4x_3)$ and $u_2(\mathbf{x}) = x_1(3 + x_2(-1 + x_3)) - (-2 + x_2)x_3$. This control objective corresponds to the activation of some genes in the system. We consider two cases: with no control (i.e. $u_1 = u_2 = 0$) and with the above control law. Figure 3.2 shows the projection on x_2 and x_3 of the reachable sets obtained using our algorithm for polynomial systems. In [14] we have already treated this model using an abstraction method based on projection. This method approximates the multi-affine

system by a lower dimensional bilinear system. Comparing with the result presented in [14], one can see that our new algorithm for polynomial systems is more accurate, and in addition we have observed that it is also more time-efficient.

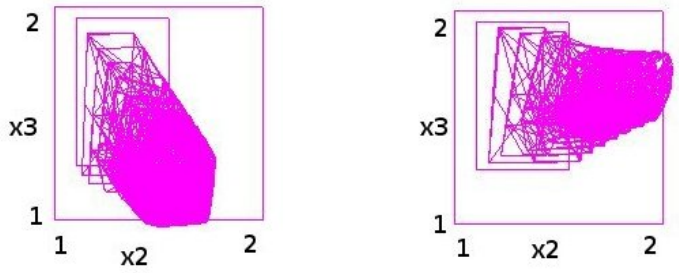


Figure 3.2: Reachable sets: with $u_1 = u_2 = 0$ (left) and with the specified control law (right). The control law indeed drives the system to the face $x_2 = 2$.

The second model is taken from [22]. It is a hybrid model[3] with two modes and one additional continuous variable x_4. The continuous dynamics is

$$\dot{\mathbf{x}} = A\mathbf{x} + g(\mathbf{x}) + b_{ij}$$

where b_{01} and b_{10} correspond respectively to the non-luminescent and luminescent modes, and

$$A = \begin{pmatrix} \frac{-1}{H_{sp}} & 0 & 0 & r_{Co} \\ 0 & 0 & 0 & \frac{-1}{H_{sp}} - r_{Co} \\ 0 & x_0 r_{AII} & \frac{-1}{H_{AI}} & x_0 r_{Co} \\ 0 & \frac{-1}{H_{sp}} & 0 & 0 \end{pmatrix} ; \; g(\mathbf{x}) = \begin{pmatrix} -1 \\ 1 \\ -x_0 \\ 0 \end{pmatrix} r_{AIR} x_1 x_3$$

We are interested in the question of how to determine the sets of states from which the system can reach the luminescent equilibrium. The condition for switching between the two modes is $x_2 = x_{2sw}$. This problem was also previously studied in [22] using the tool **d/dt** that I developed during my PhD thesis. However, in [22] the multi-affine dynamics was approximated by a 3-dimensional linear system, assuming that x_1 remains constant. Using our algorithm for polynomial systems, we can now handle the non-linearity in the dynamics. To deal with the discrete dynamics of the model, it suffices to implement some Boolean operations over the reachable set generated by the continuous dynamics, which are represented by polyhedra. In terms of qualitative behavior, the result obtained for the 4-dimensional multi-affine model is compatible with the result for the linear

[3]The numbering of variables is different from that in [22].

approximate model in [22], that is, from the non-luminescent mode the system can reach the guard to switch to the luminescent mode and then converge to the equilibrium. However, the new result obtained for the 4-dimensional model shows a larger set of states that can reach the equilibrium. This can be explained by the fact that in this model the variable x_1 is not kept constant and can evolve in time.

3.4 Using box splines

We now describe a method based on the box-spline representation of polynomials, the coefficients of which can be used to enclose all the reachable points. Compared to our previous method using the Bézier simplex representation, this method works on more general domains and does not require simplicial decompositions.

We start by providing a brief introduction of box splines and some of their properties, which are useful for our problem. We then show how to use the box-spline representation to approximate the image of a set by a polynomial.

The next sections focus on each step of the algorithm. Finally, we report some experimental results obtained using the algorithm on a biological system and a number of randomly generated examples. Before concluding, we present a brief survey of related work.

3.4.1 Box splines

Definition

We will present an inductive definition of box spline, and other definitions (such as, geometric or by recursion) can be found in [**?**, **?**].

Let n be the dimension (or the number of variables). Given an integer $k \geq n$, let $V = \{\xi_1, \ldots, \xi_k\}$ be a set of k vectors in $\mathbb{R}^n$ where $\xi_1, \xi_2, \ldots, \xi_n$ are linearly independent. Each such vector is called *direction*. We denote by $M = [\xi_1\, \xi_2\, \ldots \xi_k]$ a matrix of size $n \times k$, the columns of which are the vectors in V. For convenience of notation, we use the same letter ξ_i to refer to a vector in V or a column of the matrix M.

Without loss of generality, it could be assumed that the first n columns of M form the n-dimensional identity matrix I. The inductive definition of a box spline B_M associated with V is as follows.

We denote $s = k - n$ and start with the base case where $k = n$ and $s = 0$. Then $M_s = I$ and $B_{M_s}(x) = 1$ if $x \in [0, 1)^n$ and 0 otherwise. Note that this function is piecewise constant and has degree $s = 0$.

If we add a column in M denoted by $M \cup \xi$, then the box spline associated with the new matrix $M_s = M_{s-1} \cup \xi$ is defined as:

$$B_{M_s \cup \xi} = \int_0^1 B_{M_{s-1}}(x - t\xi)dt. \tag{3.12}$$

Thus, each convolution in another direction v increases the degree by 1, and when $s = k - n$, B_M is a piecewise polynomial of degree $k - n$.

We use the well-known Zwart-Powell box spline [**?**] to illustrate the above definition. First, we consider the following matrix of size 2×3

$$M = \begin{pmatrix} 1 & 0 & 1 \\ 0 & 1 & 1 \end{pmatrix}$$

In this case we have $n = 2$, $k = 3$. We start with $k = n = 2$ and the identity matrix formed by two vectors $\xi_1 = (1, 0)$ and $\xi_2 = (0, 1)$ defines a unit box (see Figure 3.4.1-(a)). By adding the vector $\xi_3 = (1, 1)$ the matrix M defines a mesh inside a zonotope, as shown in Figure 3.4.1-(b). Using the definition (3.12), we obtain the box spline B_M which is the hat function shown in Figure 3.4.1.

We further add a new vector $\xi_4 = (-1, 1)$ to form a matrix M' as follows:

$$M' = \begin{pmatrix} 1 & 0 & 1 & -1 \\ 0 & 1 & 1 & 1 \end{pmatrix}.$$

This defines a new mesh (see Figure 3.4.1-(d)) and the corresponding box spline $B_{M'}$ is depicted in Figure 3.4.1. This function is called Zwart-Powell box spline.

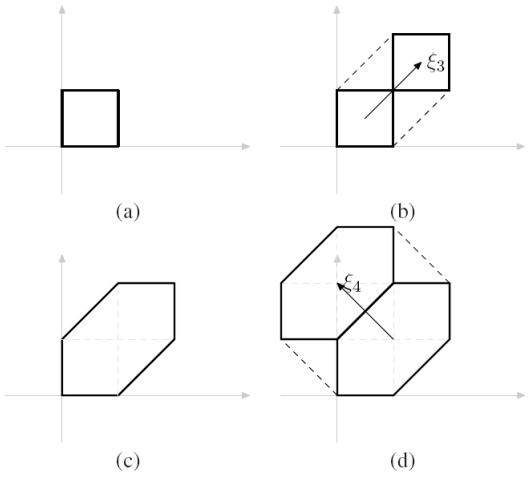


Figure 3.3: Supports of some box splines.

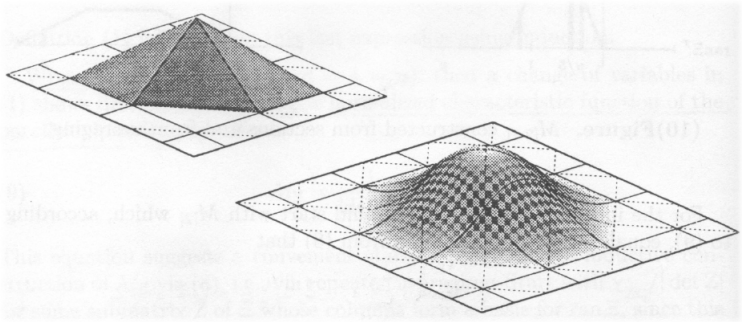


Figure 3.4: The hat function (left) and the Zwart-Powell box spline (right).

Basic properties

The box splines have the following properties which follow directly from their definition.

For all $x \in [\xi_1\, \xi_2\, \ldots \xi_k][0,1)^k$, $B_M(x) > 0$. The support of $B_M(x)$ is

$$Z = [\xi_1 \xi_2 \ldots \xi_k][0,1]^k. \tag{3.13}$$

that is the sum of the vectors in V, or *the zonotope with V as the set of its generators*. The box spline B_M is symmetric with respect to the center of its support.

It should be noted that the box spline B_M is *piecewise polynomial*. Indeed, it is a polynomial of degree *at most* $(k-n)$ within each element of the simplicial mesh defined by V over the support of B_M. The following lemma states two important properties of B_M.

Let ρ de the minimal number of vectors that need to be removed from V so that they do not span $\mathbb{R}^n$. The box spline B_M is $(\rho - 2)$ times continuously differentiable.

Again, we assume that the vectors in $V = \{v_1, \ldots, v_k\}$ span $\mathbb{R}^n$. We define an *integer shift* of the box spline $B_M(x-\mathbf{j})$ with $\mathbf{j} \in \mathbb{Z}^n$. Since $B_M(x)$ is non-negative and the sum of all the integer shifts of $B_M(x)$ is 1, the integer shifts of any box spline $B_M(x)$ form a *partition of unity*.

We are now interested in the space of polynomials spanned by all shifts of B_M. We define the *index set* for each $x \in \mathbb{R}^n$ as follows:

$$\mathcal{I}_M(x) = \{\mathbf{i} \in \mathbb{Z}^n \mid B_M(x - \mathbf{i}) \neq 0\}.$$

Note that this set is finite for any x; therefore, for simplicity we write infinite linear combinations of the integer shifts while keeping in mind that the combinations are only over the associated index set.

Lemma 7. *If the box spline B_M is r times continuously differentiable. Then, for any polynomial* $\mathbf{a}$ *of degree* $(r+1)$*, the function*

$$\pi(x) = \sum_{\mathbf{i} \in \mathbb{Z}^n} \mathbf{a}(\mathbf{i}) B_M(x - \mathbf{i}). \tag{3.14}$$

is a polynomial of degree $(r+1)$*, and any polynomial can be represented as in (3.14) by choosing M appropriately. The coefficients* $\mathbf{a}(\mathbf{i})$ *are called the* control points*, and the function* $\mathbf{a}$ *is called the weight function.*

Lemma 8. *Given a point x, $\pi(x)$ lies in the convex hull of the control points corresponding to the index set $\mathcal{I}_M(x)$:*

$$\pi(x) = conv\{\mathbf{a}(\mathbf{i}) \mid \mathbf{i} \in \mathcal{I}_M(x)\}.$$

This property, called the *convex-hull property*, will be used for our image computation problem.

3.4.2 Image approximation using box splines

In this section we focus on the main problem of the paper, which is computing the image of a bounded set $X \subset \mathbb{R}^n$ by a polynomial map $\pi : \mathbb{R}^n \to \mathbb{R}^n$. This image, denoted by $\pi(X)$, is defined as follows:

$$\pi(X) = \{\pi(x) \mid x \in X\}.$$

To this end, as mentioned earlier, we use the control points in a box spline representation, provided that the box spline B_M must be of appropriate degree. To compute these control points, we derive a symbolic expression of the weight function $\mathbf{a}$, and then determine the index set, that is the set of integer points at which the integer shifts in the box spline representation (3.14) is non-null.

Index set

As mentioned earlier, when writing the infinite sum, it indeed suffices to consider the integer points in the index set. Given x, the index set $\mathcal{I}(x)$ associated with the box spline B_M is $\mathcal{I}_M(x) = \{\mathbf{i} \in \mathbb{Z}^n \mid B_M(x - \mathbf{i}) \neq 0\}$. It is not hard to prove that

$$\mathcal{I}_M(x) = \mathbb{Z}^n \cap (x - M[0,1]^k). \tag{3.15}$$

The index set for all x inside some set $X \subseteq \mathbb{R}^n$, denoted by $\mathcal{I}_M(X)$, is defined as follows:

$$\begin{aligned} \mathcal{I}_M(X) &= \{\mathbf{i} \in \mathbb{Z}^n \mid \exists x \in X : B_M(x - \mathbf{i}) \neq 0\} \\ &= \mathbb{Z}^n \cap (X \oplus (-M[0,1]^k)). \end{aligned}$$

where $\oplus$ denotes the Minkowski sum. Then the computation of the index set $\mathcal{I}_M(X)$ amounts to enumerating all the points with integer coordinates inside the Minkowski sum of P and the zonotope $-M[0,1]^k$.

Image computation algorithm

We are now ready to apply the above image computation method to compute the reachable set of a discrete-time system

$$x_{k+1} = \pi(x_k)$$

where π is a polynomial map and $x_0 \in X_0$.

We assume that we have chosen an appropriate matrix M. This means that the condition on M, stated in Lemma 7, which guarantees that the associated box spline B_M can reproduce all the polynomials of the required degree.

In the following algorithm, the initial set X_0 is assumed to be a polytope. In each iteration, we compute a set of points, the convex hull of which is an over-approximation of the reachable set. It is important to emphasize that in this abstract algorithm in each iteration the reachable set is the convex hull of a point set, however we do not yet specify how this set is represented.

Algorithm 3 Image computation

$P^0 = vertices(X_0)$
$Z = (-M[0,1]^k)$
$i = 0$
repeat
  $I = indexset(P^i, Z)$ /* *computing the index set* */
  $P^{i+1} = \emptyset$
  forall($\mathbf{i} \in I$)
    $p = \mathbf{a}(\mathbf{i})$ /* *computing the control points* */
    $P^{i+1} = P^i \cup \{p\}$
  endforall
  $i = i + 1$
until $i > K_{max}$

The algorithm consists of three main steps:

1. The function *indexset* computes the index set associated with the Minkowski sum $conv\{P^i\} \oplus (-M[0,1]^k)$. This computation is summarized in Algorithm 3.4.3, which will be discussed in the next section. It is important to note that using this algorithm we avoid the computation of the convex hull of the point set P^i.

2. To each integer point in the index set, we apply the weight function $\mathbf{a}$ to obtain the corresponding control point. By the convex-hull property of the box spline representation, the convex hull of all such control points is an over-approximation of the reachable set at the current iteration. If the convex hull computation is expensive, one can choose a representation for the reahcable sets, which is appropriate for the property verification and for achieving a good trade-off between accuracy and computation cost. In addition, the computation of the index set also depends on the chosen representation. This will be discussed in Section 3.4.3.

3. The weight function $\mathbf{a}$ for the given box spline B_M can be precomputed in a symbolic form, which we will show in Section 3.4.5.

Refining the approximation via subdivision.

When approximating a polynomial by its box-spline representation coefficients, it is possible to improve the precision by refining the underlying integer grid. Indeed, given $w \in \mathbb{Z}$, we obtain a finer box-spline representation of the polynomial π as follows:

$$\pi(x) = \sum_{\mathbf{i} \in \mathbb{Z}^n} \mathbf{a}^w(\mathbf{i}) B_M(wx - \mathbf{i})$$

where the points $\mathbf{a}^w(\mathbf{i}) = w^n \mathbf{d}^k(i)$ can be successively computed by the following recursion [**?**]:

$$\begin{aligned} \mathbf{d}^0(\mathbf{i}) &= \begin{cases} 0 & \text{if } \frac{1}{w}\mathbf{i} \notin \mathbb{Z}^n, \\ \mathbf{a}^1(\frac{1}{w}\mathbf{i}) & \text{otherwise.} \end{cases} \\ \mathbf{d}^r(\mathbf{i}) &= \tfrac{1}{w} \textstyle\sum_{l=0}^{w-1} \mathbf{d}^{r-1}(\mathbf{i} - l\xi_r), r = 1, \ldots, k. \end{aligned}$$

It can be proven that the polynomial $\pi(x)$ is approximated by the convex hull of the control points with the error $\mathcal{O}(\frac{1}{w^2})$. This means that the convergence order of this approximation is *quadratic*. Details on the convergence proof of our algorithm can be found in [**?**].

3.4.3 Enumerating integer points

This section is concerned with the computation of the function *indexset* in Algorithm 3.

We first assume that the reachable set at each iteration is a bounded polytope represented by its vertices. We thus focus on the problem of computing the integer points in the Minkowski sum $Q = P \oplus Z$ where P is a bounded polytope and Z is a zonotope. Before continuing, we briefly present a definition of zonotopes.

A zonotope is the Minkowski sum of a finite set of line segments. In this work, we define a zonotope by its center and generators. The zonotope Z, centered at c and with generators $\{g_1, \ldots, g_{m_z}\}$ is the set

$$Z = \{c + \sum_{i=1}^{m_z} \lambda_i \gamma_i \mid \forall i \in \{1, \ldots, m_z\} : \ -1 \leq \lambda_i \leq 1\}.$$

Given two zonotopes Z and Z' respectively described by the centers c and c' and the sets of generators G and G', their Minkowski sum $Z \oplus Z'$ is a zonotope with center $(c + c')$ and its set of generators is $G \cup G'$.

We now go back to our problem. Let the zonotope Z be represented by a center c and m_z generators $\{g_1, \ldots, g_{m_z}\}$ and P be represented by m vertices $\{v_1, \ldots, v_m\}$.

Indeed, the algorithm we use to solve this problem is inspired by the *Mayan pyramid algorithm* in [**?**], which computes a set of integer points with positive distance to the boundary in a given direction.

To avoid an explicit construction of the Minkowski sum, we find the minimum and maximum integral values of the k^{th} when the first $(k-1)$ coordinates have been fixed. In other words, the algorithm examines orthogonal projections of the Minkowski sum to linear subspaces.

Before presenting the algorithm, we need some notations. Given a set $Q \subset \mathbb{R}^n$, we denote by $proj_k(Q)$ the (usual) projection of this set to $\mathbb{R}^k$. Now, given a point $p \in \mathbb{R}^n$, let $proj_k(p, Q)$ be a subset of $proj_k(Q)$ such that each point in $proj_k(p, Q)$ has the same $(k-1)$ first coordinates as p, that is

$$proj_k(p, Q) = \{y \in proj_k(P) \mid (y_1, \ldots, y_{k-1}) = (p_1, \ldots, p_{k-1})\}.$$

Then, we define

$$coord_k(p, Q) = \{y_k \mid \exists y_1, \ldots, y_{k-1} \ : \ (y_1, \ldots, y_{k-1}, y_k) \in proj_k(p, Q)\},$$

which is the set that contains all the k^{th} coordinates of the points in $proj_k(p, Q)$.

Step 2 of the algorithm can be done by solving the following linear programming problems:

$$\begin{gathered}
mn = \min s, \\
mx = \max s, \\
\text{subject to} \\
(\bar{p}, s) = c + \textstyle\sum_{j=1}^{m_z} \lambda_j g_j + \sum_{i=1}^{m} \alpha_i v_i \\
\forall j \in \{1, \ldots, m_z\} : \ -1 \leq \lambda_j \leq 1, \ \textstyle\sum_{j=1}^{m_z} \lambda_i = 1 \\
\forall i \in \{1, \ldots, m\} : \ \alpha_i \geq 0, \ \textstyle\sum_{i=1}^{m} \alpha_i = 1
\end{gathered}$$

Algorithm 4 Enumerating integer points

Step 1: $T = 0$, $k = 1$, $\bar{p} = \emptyset$ /* $\bar{p}$ is initially an empty vector */
Step 2:
$mn = \min\{coord_k(\bar{p}, P \oplus Z)\}$
$mx = \max\{coord_k(\bar{p}, P \oplus Z)\}$
Step 3:
if $k < n$ **then**
  for $s \in [mn, mx]$ **do**
    $\bar{p} = (\bar{p}, s)$
    $k++$
    Goto Step 2
  end for
end if
if $k = n$ **then**
  for $s \in [mn, mx]$ **do**
    $\bar{p} = (\bar{p}, s)$
    $T = T \cup \{\bar{p}\}$
  end for
end if

The algorithm returns the set T of integer points in the Minkowski sum.

Note that in the above we used the assumption that the vertices of the polytope P are known. Indeed, in Algorithm 3 the reachable set at each iteration is represented as a point set whose the convex hull guarantees to be an over-approximation of the reachable set. Nevertheless, the convex-hull computation is expensive in high dimensions and we would like to avoid this.

In this work, we choose to approximate this polytope by a zonotope. It is not hard to see that if P is a zonotope, then the set $P \oplus Z$ is a zonotope. The motivations of this choice are the following. First, the Minkowski sum of zonotopes, unlike that of polytopes, can be efficiently computed (as described earlier). Second, the resulting zonotope can serve as a compact representation of the reachable set at each iteration, which not only can be used to check against a property of interest but also allows further extension to hybrid dynamics (for which treating discrete transitions requires Boolean operations over reachable sets of continuous dynamics). In Section 3.4.4 we will describe how to over-approximate the convex hull of a point set by a zonotope.

Scaling

When the diameter of a set X is large, the number of integer points can be large and their enumeration can thus be expensive. To handle this, we consider the

composition of the polynomial π with an invertible linear transformation τ that maps (or scales) X to some set Y with a small diameter. The function resulting from this composition is still a polynomial, for which we can apply this image computation method. This is explained more formally in the following.

Let the composition $\gamma = (\pi \; o \; \tau^{-1})$ is defined as $\gamma(x) = \pi(\tau^{-1}(x))$. Note that the function γ can be computed symbolically. Hence, $\gamma(Y) = \{\pi(\tau^{-1}(x)) \mid x \in Y\}$. We have $\tau^{-1}(Y) = X$. It then follows that $\pi(X) = \gamma(Y)$. Therefore, to obtain the image $\pi(X)$ we compute $\gamma(Y)$. Since Y is a set with a smaller diameter, the size of the index set of Y associated with γ can be reduced.

3.4.4 Approximation by zonotopes

Let P be a set of a points in $\mathbb{R}^n$, we consider the problem of over-approximating the convex hull $conv\{P\}$ by a zonotope. Let us first sketch the main idea of our method.

First, using the Prinpical Component Analysis (PCA) [69] we can find an oriented hyper-rectangle R that encloses P. We denote this by $R = boundPCA(P)$ and defer a description of this procedure to Appendix. Let $l_1, \ldots, l_n$ be the side length of R and $\eta_1, \ldots, \eta_n$ be the axes of R. Intuitively, $\eta_1, \ldots, \eta_n$ define the orientation of a box which best captures the spread of the point set P. We next try to use the idea of PCA to determine other spreading directions. To this end, we introduce a special *translation operator*. Roughly speaking, the goal of this operator is to filter out the effects of the "spreading directions" $\eta_1, \ldots, \eta_n$.

We define $R_{\boldsymbol{\lambda}}$ the rectangle resulting from scaling R by $\boldsymbol{\lambda} \in (0, 1)^n$ around its center (or centroid) c.

Given a point x, we next define a *translation operator* with respect to $R_{\boldsymbol{\lambda}}$, denoted by $\tau(x, R_{\boldsymbol{\lambda}})$. Let H_i denote the hyper-plane which has η_i as its normal and goes through o. Let $\Pi_i(x)$ be the projection of x on H_i. We define

$$\Delta_i = \frac{\delta_i}{||\Pi_i(x) - x||}(\Pi_i(x) - x)$$

where $\delta_i \in [0, l_i/2]$. Then, the translation operator τ is defined as follows.

$$\tau(x, R_{\boldsymbol{\lambda}}) = \begin{cases} o & \text{if } x \in R_{\boldsymbol{\lambda}}, \\ x + \sum_{i=1}^{n} \Delta_i & \text{otherwise.} \end{cases}$$

We extend this operator to a set of points: $\tau(P, R_{\boldsymbol{\lambda}}) = \{\tau(x, R_{\boldsymbol{\lambda}}) \mid x \in P\}$.

Intuitively, the points which are inside $R_{\boldsymbol{\lambda}}$ are close to the hyperplanes H_i ($i \in \{1, \ldots, n\}$, that is they spread in the directions of the current oriented hyper-rectangle R. These points will be translated to the same centroid point

c. The other points, which are ouxide $R_{\boldsymbol{\lambda}}$, are translated nearer to c along the directions $\eta_1, \ldots, \eta_n$. The intuition behind this is that this translation "reduces" the spread of the point set along the directions $\eta_1, \ldots, \eta_n$ and thus enables the PCA algorithm to detect new spreading directions.

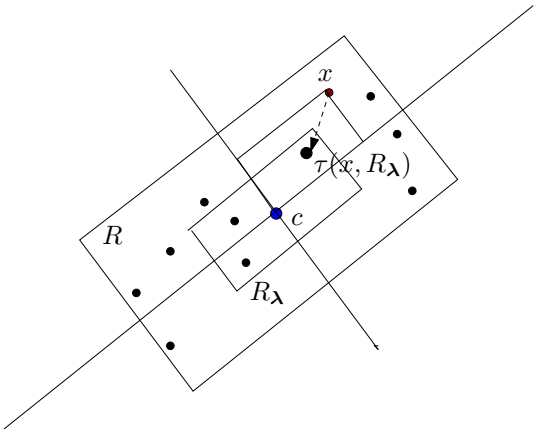


Figure 3.5: Illustration of the translation operator τ with $\delta_1 = \delta_2 = 0.5$.

In the above description, $(\delta_1, \ldots \delta_n)$ and $\boldsymbol{\lambda}$ are user-defined parameters. We can compute a zonotope that over-approximates $conv\{P\}$ using the following iterative algorithm.

Algorithm 5 Approximation the convex hull of a point set P by a zonotope

$Q^0 = P$
$i = 0$
repeat
  $R^i = boundPCA(Q^i)$;
  $Q^{i+1} = \tau(Q^i, R^i_{\boldsymbol{\lambda}})$;
  $i = i + 1$
until $size(R^i) \leq \varepsilon$

The algorithm stops when the size of R^i is sufficiently small, and then all the direction vectors of each rectangle R^i computed in each iteration are used to define the generators of the over-approximating zonotope. The center of the zonotope is the center of R^0.

3.4.5 Control points

Any polynomial can be decomposed to a linear combination of monomials, it is possible to compute the control points for each monomial and then combine them.

More concretely, if the polynomial π is a linear combination of two monomials m_1 and m_2, i.e. $\pi = km + k'm'$. Then, if $m(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n} \mathbf{a}(\mathbf{i})B_M(x-\mathbf{i})$ and $m'(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n} \mathbf{a}'(\mathbf{i})B_M(x-\mathbf{i})$, then $\pi(x) = \sum_{\mathbf{i}\in\mathbb{Z}^n}(\mathbf{a}(\mathbf{i}) + \mathbf{a}'(\mathbf{i}))B_M(x-\mathbf{i})$.

We use the extension of the Marsden identity [**?**] to derive an analytic expression of $\mathbf{a}$. We consider a monomial of the form $x_1^{r_1}\dots x_n^{r_n}$ where each r_i is a non-negative integer, and we denote it as $m_r(x) = x^r$ where $r = (r_1,\dots,r_n)$ is called the multi-index. We define $r' \prec r$ iff $r \neq r'$ and $\forall i \in \{1,\dots,n\} : r'_i \leq r_i$. Similarly, the difference $r - r'$ can be defined componentwise, that is $r - r' = (r_1 - r'_1,\dots,r_n - r'_n)$.

Given a monomial m_r, the goal is to determine the weight function $\mathbf{a}_r$ such that the monomial can be represented using the box spline B_M, that is, $x^r = \sum_{\mathbf{i}} \mathbf{a}(\mathbf{i})B_M(x-\mathbf{i})$. We define the operator

$$\mu : f \rightarrow \sum_{\mathbf{i}} B_M(\mathbf{i})f(-\mathbf{i}). \tag{3.16}$$

The (symbolic) computation of the function $\mathbf{a}$ can be done using the following recurrence [**?**]:

$$\begin{cases} \mathbf{a}_r & = m_r - \sum_{r'\prec r} \mu(m_{r-r'})\,\mathbf{a}_{r'} \\ \mathbf{a}_{(0,\dots,0)} & = 1. \end{cases} \tag{3.17}$$

The generation of $\mu(m_{r'})$ for all $r' \prec r$ can be done before starting the recursion (3.17). For a fixed r, we need to compute the expresions of $c_{r'}$ for all $r' \prec r$, and this requires computing the application of the operator μ on them. To do so, as shown in (3.16), we need to determine the values of the box spline B_M at all the integer points inside its support.

The following recursive method for evaluating B_M [**?**] can be used. For a point x which can be represented as $x = M\beta$ and let β_ξ denote the scalar component of the vector $\beta \in \mathbb{R}^k$ corresponding to the column ξ of the matrix M. Then,

$$(k-n)B_M(x) = \sum_{\xi\in V} \beta_\xi B_{M\setminus\{\xi\}}(x) + (1-\beta_\xi)B_{M\setminus\{\xi\}}(x-\xi) \tag{3.18}$$

where $M \setminus \{\xi\}$ is the matrix resulting from removing the column ξ from M. The base case for the above recurrence corresponds to the square matrix $M = (\xi_1,\dots,\xi_n)$, and in this case

$$B_M(x) = \frac{1}{|detM|}\chi_{M[0,1]^k}(x)$$

where χ is the characteristic function of the set $M[0,1]^k$.

Experimental results

We have implemented the above-described algorithm and carried out a number of experimentations. In this section we first illustrate our method on a biological system.

This is a model of gene transcription control in the bacteria Vibrio Fisheri, which is used to analyze and control the luminescence mechanism in these bacteria [22, 21]. As a free-living organism, Vibrio Fischeri exists at low densities and appears to be non-luminescent. As a symbiont, the bacteria live at high densities and are, usually, luminescent.

A simplified model of this system is described by the following multi-affine differential equations:

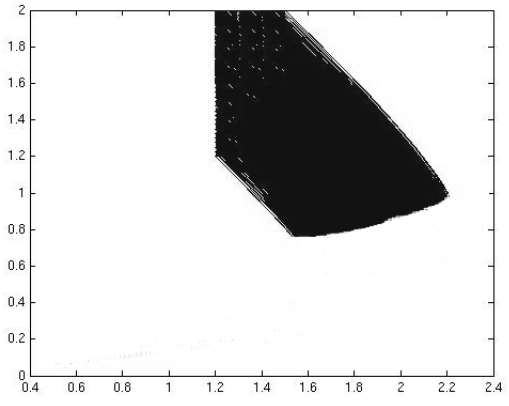
$$\begin{cases} \dot{x_1} &= k_2 x_2 - k_1 x_1 x_3 + u_1 \\ \dot{x_2} &= k_1 x_1 x_3 - k_2 x_2 \\ \dot{x_3} &= k_2 x_2 - k_1 x_1 x_3 - \eta x_3 + \eta u_2 \end{cases} \tag{3.19}$$

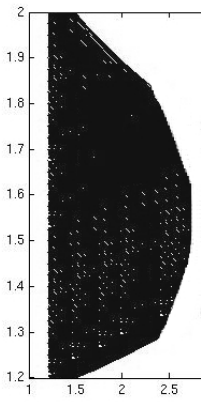
The state variables $x = (x_1, x_2, x_3)$ represent the cellular concentrations of different species, and the parameters $k_1 = 30$, $k_2 = 10$ and $\eta = 10$ are repectively the binding, dissociation and diffusion constants. The control variable u_1 physically represents the plasmid producing protein LuxR and u_2 external source of autoinducer. In [21] the following control law for steering all the states in the rectangle $[1, 2] \times [1, 2] \times [1, 2]$ to the face $x_2 = 2$ was proposed:

$$\begin{aligned} u_1(x) &= -10(x_2 + x_1(-1 + x_3) - 4x_3) \\ u_2(x) &= x_1(3 + x_2(-1 + x_3)) - (-2 + x_2)x_3 \end{aligned}$$

This control objective corresponds to the activation of some genes in the system in order to switch on the lux gene. We analyze the behaviors of the systems with no control (that is when $u_1 = u_2 = 0$) and under the above control law. A discretization scheme, similar to the one proposed in our previous paper [36], was used to transform this differential equation to a difference equation. Figure 3.2 shows the projection on x_2 and x_3 of the reachable sets obtained using Algorithm 3. This shows that without the control the system may exit the rectangle through the face $x_1 = 1$ and not through the face $x_2 = 2$. The computation time on this example was 35 seconds.

Comparing with the result presented in [36], to obtain a comparable accuracy, our new algorithm for polynomial systems is more time-efficient. Indeed, the Bézier method requires the convex-hull and triangulation operations, which on one hand are expensive and, on the other hand, may cause numerical stability problems in practice. This can thus compromise the theoretical accuracy of the method.





Higher dimensional examples.

We have also applied the algorithm to a number of polynomial systems of degree up to 3 in higher dimensions. The average computation time for 100 iterations was: $5.3s$ for $n = 3$; $20s$ for $n = 4$; $120s$ for $n = 5$. We observe that the time complexity of the algorithm is less sensitive to the degree of the polynomial than to their number of variables (or dimension). More experimentation is needed to better evaluate the practical cost and the accuracy of the method.

The novelty of the method we proposed in this paper is an easy way to control approximation power via efficient subdivision. As future work, we intend to study a number of related problems such as zonotopic approximation.

3.5 Using the Bernstein expansion

The drawback of the Bézier method is that it requires expensive convex-hull and triangulation computation, which restricts its application to systems of dimensions not higher than $3, 4$. The essence of the new method we propose in this paper can be summarized as follows. Using a special class of polyhedra together with optimization, we are able to reduce the complexity of the required polyhedral manipulation. Furthermore, by exploiting the Bernstein expansion, we only need to solve linear programming problems instead of polynomial optimization problems.

Our method is similar to a number of existing methods for continuous and hybrid systems in the use of linear approximation. Its novelty resides in the efficient way of computing linear approximations. Indeed, a common method to approximate a non-linear function by a piecewise linear one, as in the hybridization approach [16] for hybrid systems, requires non-linear optimization. Indeed,

the work presented in this paper follows the approach using template polyhedra and optimization for hybrid systems with continuous dynamics proposed in [**?**].

Besides constrained global optimization, other important applications of the Bernstein expansion include various control problems [53] (in particular, robust control). The approximation of the range of a multivariate polynomial over a box is also used in program analysis and optimization (for example [98, 33]). In the hybrid systems verification, polynomial optimization is used to compute barrier certificates [92]. Algebraic properties of polynomials are used to compute polynomial invariants [101] and to study the computability of image computation in [**?**].

In the following we first recall the notions of template polyhedra and the Bernstein expansion. We then describe an optimization-based solution. In order to transform the polynomial optimization problem to a linear programming problem, a method for computing bound functions is presented. We then describe an algorithm summarizing the main steps of our method. Some experimental results, in particular the analysis of a control and a biological systems, are reported.

Template polyhedra. A convex polyhedron is a conjunction of a finite number of linear inequalities described as $A\mathbf{x} \leq \mathbf{b}$, where A is a $m \times n$ matrix, $\mathbf{b}$ is a column vector of size m. Template polyhedra are commonly used in static analysis of programs for computing invariants (see for example [94]). The reader is referred to [94] for a thorough description of template polyhedra.

A template is a set of linear functions over $\mathbf{x} = (x_1, \ldots, x_n)$. We denote a template by an $m \times n$ matrix H, such that each row H^i corresponds to the linear function $H^i\mathbf{x}$. Given such a template H and a real-valued vector $\mathbf{d} \in \mathbb{R}^m$, a template polyhedron is defined by considering the conjunction of the linear inequalities of the form $\bigwedge_{i=1,\ldots,m} H^i\mathbf{x} \leq d_i$. We denote this polyhedron by $\langle H, \mathbf{d}\rangle$.

By changing the values of the elements of $\mathbf{d}$, one can define a family of template polyhedra corresponding to the template H. We call $\mathbf{d}$ a *polyhedral coefficient vector*. Given $\mathbf{d}, \mathbf{d}' \in \mathbb{R}^m$, if $\forall i \in \{1, \ldots, m\} : \ d_i \leq d'_i$, we write $\mathbf{d} \preceq \mathbf{d}'$. Given an $m \times n$ template H and two polyhedral coefficient vectors $\mathbf{d}, \mathbf{d}' \in \mathbb{R}^m$, if $\mathbf{d} \preceq \mathbf{d}'$ then the inclusion relation $\langle H, \mathbf{d}\rangle \subseteq \langle H, \mathbf{d}'\rangle$ holds, and we say that $\langle H, \mathbf{d}\rangle$ is not larger than $\langle H, \mathbf{d}'\rangle$.

The advantage of template polyhedra over general convex polyhedra is that the Boolean operations (union, intersection) and common geometric operations can be performed more efficiently [94].

Bernstein expansion. We consider an n-variate polynomial $\pi : \mathbb{R}^n \to \mathbb{R}^n$ defined as: $\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_\mathbf{w}} \mathbf{a_i}\mathbf{x}^\mathbf{i}$ where $\mathbf{x}^\mathbf{i} = x_1^{\mathbf{i}_1} \ldots x_n^{\mathbf{i}_n}$, $\mathbf{a_i}$ is a vector in $\mathbb{R}^n$; $\mathbf{i}$ and $\mathbf{w}$ are two multi-indices of size n such that $\mathbf{i} \leq \mathbf{w}$; $I_\mathbf{w}$ is the set of all multi-indices

$\mathbf{i} \leq \mathbf{d}$, that is $I_{\mathbf{w}} = \{\mathbf{i} \mid \mathbf{i} \leq \mathbf{w}\}$. The multi-index $\mathbf{w}$ is called the *degree* of π.

Given a set $X \subset \mathbb{R}^n$, *the image of* X *by* π, denoted by $\pi(X)$, is defined as follows: $\pi(X) = \{(\pi_1(\mathbf{x}), \ldots, \pi_n(\mathbf{x})) \mid \mathbf{x} \in \mathbb{R}^n\}$.

In order to explain the Bernstein expansion of the polynomial π, we first introduce Bernstein polynomials. For $\mathbf{x} = (x_1, \ldots, x_n) \in \mathbb{R}^n$, the $\mathbf{i}^{th}$ Bernstein polynomial of degree $\mathbf{w}$ is: $\mathcal{B}_{\mathbf{w},\mathbf{i}}(\mathbf{x}) = \beta_{\mathbf{w}_1,\mathbf{i}_1}(x_1) \ldots \beta_{\mathbf{w}_n,\mathbf{i}_n}(x_n)$ where for a real number y, $\beta_{\mathbf{w}_j,\mathbf{i}_j}(y) = \binom{\mathbf{w}_j}{\mathbf{i}_j} y^{\mathbf{i}_j}(1 - y^{\mathbf{w}_j - \mathbf{i}_j})$. Then, for all $\mathbf{x} \in B_u = [0,1]^n$, π can be written using the Bernstein expansion as follows:

$$\pi(\mathbf{x}) = \sum_{\mathbf{i} \in I_{\mathbf{w}}} \mathbf{b}_{\mathbf{i}} \mathcal{B}_{\mathbf{w},\mathbf{i}}(\mathbf{x})$$

where for each $\mathbf{i} \in I_{\mathbf{w}}$ the Bernstein coefficient $\mathbf{b}_{\mathbf{i}}$ is:

$$\mathbf{b}_{\mathbf{i}} = \sum_{\mathbf{j} \leq \mathbf{i}} \frac{\binom{\mathbf{i}}{\mathbf{j}}}{\binom{\mathbf{w}}{\mathbf{j}}} \mathbf{a}_{\mathbf{j}}. \tag{3.20}$$

The following property of the Bernstein coefficients is of interest. The above enclosure yields: $\forall \mathbf{x} \in B_u : \ \pi(\mathbf{x}) \in \square(\{\mathbf{b}_{\mathbf{i}} \mid \mathbf{i} \in I_{\mathbf{w}}\})$ where $\square$ denotes the bounding box of a point set.

Let us return to the main problem, which is computing the image of a set by a polynomial. Using the above convex-hull property, we can use the coefficients of the Bernstein expansion to over-approximate the image of the unit box B_u by the polynomial π. To compute the image of a general convex polyhedron, one can over-approximate the polyhedron by a box and then transform it to the unit box via some affine transformation. A similar idea, which involves using the coefficients of the Bézier simplex representation, was used in [36] to compute the image of a convex polyhedron. However, the convex-hull computation is expensive especially in high dimensions, which poses a major problem in continuous and hybrid systems verification approaches using polyhedral representations.

In this work, we propose a new method which can avoid complex convex-hull operations over general convex polyhedra as follows. First, we use template polyhedra to over-approximate the images. Second, the problem of computing such template polyhedra can be formulated as a polynomial optimization problem. This optimization problem is computationally difficult, despite recent progress in the development of methods and tools for polynomial programming (see for example [105, 68, 51] and references therein). We therefore seek their affine bound functions for polynomials, in order to transform the polynomial optimization problem to a linear programming one, which can be solved more efficiently (in polynomial time) using well-developed techniques, such as Simplex [8] and interior point techniques [28]. Indeed, the above-described Bernstein expansion

is used to compute these affine bound functions. This is discussed in the next section.

Bound functions. To compute bound functions, we employ the method using the Bernstein expansion, published in [54, 55, 56]. Finding convex lower bound functions for polynomials is a problem of great interest, especially in global optimization. It is important to note that the method described in this section only works for the case where the variable domain is the unit box B_u. We however want to compute the images of more general sets, in particular polyhedra. An extension of this method to such cases will be developed in Section 3.5.2.

A simple affine lower bound function is a constant function, which can be deduced from the above mentioned property of the Bernstein expansion:

$$x_i \leq \min\{\mathbf{b_i} \mid \mathbf{i} \in I_\mathbf{w}\} = \mathbf{b_{i^0}} = \mathbf{b}^0.$$

The main idea of the method is as follows. We first compute the affine lower bound function whose corresponding hyperplane passes through this control point $\mathbf{b}^0$. Then, we aditionally determine $(n-1)$ hyperplanes passing through n other control points. This allows us to construct a sequence of n affine lower bound functions $l_0, l_1, \ldots l_n$. We end up with l_n, a function whose corresponding hyperplane passes through a lower facet of the convex hull spanned by these control points. A detailed description of the algorithm can be found in [41]. Note that we can easily compute upper bound functions of π by computing the lower bound functions for $(-\pi)$ using this method and then multiply each resulting function by (-1).

3.5.1 Image computation using optimization

We want to use a template polyhedron $\langle H, \mathbf{d}\rangle$ to over-approximate the image of a polyhedron P by the polynomial π. The template matrix H, which is of size $m \times n$ is assumed to be given; the polyhedral coefficient vector $\mathbf{d} \in \mathbb{R}^m$ is however unknown. The question is thus to find $\mathbf{d}$ such that

$$\pi(P) \subseteq \langle H, \mathbf{d}\rangle. \tag{3.21}$$

It is not hard to see that the following condition is sufficient for (3.21) to hold: $\forall \mathbf{x} \in P \;:\; H\pi(\mathbf{x}) \leq \mathbf{d}$. Therefore, to determine $\mathbf{d}$, one can formulate the following optimization problem:

$$\forall i \in \{1, \ldots, m\}, d_i = \max(\Sigma_{k=1}^n H_k^i \pi_k(\mathbf{x})) \text{ subj. to } \mathbf{x} \in P. \tag{3.22}$$

where H^i is the i^{th} row of the matrix H and H_k^i is its k^{th} element. Note that the above functions to optimize are polynomials. As mentioned earlier, polynomial optimization is expensive. Our solution is to bound these functions with

affine functions, in order to transform the above optimization problem to a linear programming one. This is formalized as follows.

Optimization-based solution

We earlier discussed lower bound functions for polynomials. Note that these bound functions are valid only when the variables $\mathbf{x}$ are inside the unit box B_u. To consider more general domains, we introduce the following definition.

Definition 1 (Upper and lower bound functions)**.** *Given $f : \mathbb{R}^n \to \mathbb{R}$, the function $\upsilon : \mathbb{R}^n \to \mathbb{R}$ is called an upper bound function of f w.r.t. a set $X \subset \mathbb{R}^n$ if $\forall \mathbf{x} \in X : \ f(\mathbf{x}) \leq \upsilon(\mathbf{x})$. A lower bound function can be defined similarly.*

The following property of upper and lower bound functions is easy to prove.

Lemma 9. *Given $X, Y \subseteq \mathbb{R}^n$ s.t. $Y \subseteq X$, if υ is an upper (lower) bound function of f w.r.t. X, then υ is an upper (lower) bound function of f w.r.t. Y.*

For each $k \in \{1, \ldots, m\}$, let $u_k(\mathbf{x})$ and $l_k(\mathbf{x})$ respectively be an upper bound function and a lower bound function of $\pi_k(\mathbf{x})$ w.r.t. a bounded polyhedron $P \subset \mathbb{R}^n$. We consider the following optimization problem:

$$\forall i \in \{1, \ldots, m\}, d_i = \Sigma_{k=1}^{n} H_k^i \omega_k. \tag{3.23}$$

where the term $H_k^i \omega_k$ is defined as follows:

- If the element $H_k^i > 0$, $H_k^i \omega_k = H_k^i \max u_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$;
- If the element $H_k^i \leq 0$, $H_k^i \omega_k = H_k^i \min l_k(\mathbf{x})$ subj. to $\mathbf{x} \in P$.

The following lemma is a direct result of (3.23).

Lemma 10. *If $\mathbf{d} \in \mathbb{R}^m$ satisfies (3.23), then $\pi(P) \subseteq \langle H, \mathbf{d} \rangle$.*

Proof. It is indeed not hard to see that the solution d_i of the optimization problems (3.23) is greater than or equal to the solution of (3.22). Hence, if $\mathbf{d}$ satisfies (3.23), then $\forall i \in \{1, \ldots, m\}\ \forall \mathbf{x} \in P : \ \Sigma_{k=1}^{n} H_k^i \pi_k(\mathbf{x}) \leq d_i$. This implies that $\forall \mathbf{x} \in P : \ H\pi(\mathbf{x}) \leq \mathbf{d}$, that is the image $\pi(P)$ is included in $\langle H, \mathbf{d} \rangle$. ∎

We remark that if all the bound functions in (3.23) are affine and P is a bounded convex polyhedron, $\mathbf{d}$ can be computed by solving at most $2n$ linear programming problems. It remains now to find the affine bound functions u_k and l_k for π w.r.t. a polyhedron P, which is the problem we tackle in the next section.

3.5.2 Computing affine bound functions over polyhedral domains

The method to compute affine bound functions for polynomials described earlier can be applied only when the function domain is a unit box, anchored at the origin. The reason is that the expression of the control points of the Bernstein expansion in (3.20) is only valid for this unit box. If we over-approximate P with a box B, it is then possible to derive a formula expressing the Bernstein coefficients of π over B. However, this formula is complex and its representation and evaluation can become expensive.

We alternatively consider the composition of the polynomial π with an affine transformation τ that maps the unit box B_u to B. The functions resulting from this composition are still polynomials, for which we can compute their bound functions over the unit box. This is explained more formally in the following.

Let B be the bounding box of the polyhedron P, that is, the smallest box that includes P. The composition $\gamma = (\pi \ o \ \tau)$ is defined as $\gamma(\mathbf{x}) = \pi(\tau(\mathbf{x}))$. The functions τ and γ can be computed symbolically, which will be discussed later.

Lemma 11. *Let $\gamma = \pi \ o \ \tau$. Then, $\pi(P) \subseteq \gamma(B_u)$.*

Proof. By the definition of the composition γ, $\gamma(B_u) = \{\pi(\tau(\mathbf{x})) \mid \mathbf{x} \in B_u\}$. Additionally, $\tau(B_u) = B$. Therefore, $\gamma(B_u) = \pi(B)$. Since the polyhedron P is included in its bounding box B, we thus obtain $\pi(P) \subseteq \pi(B) = \gamma(B_u)$. ■ ■

We remark that the above proof is still valid for any affine function τ. This means that instead of an axis-aligned bounding box, we can over-approximate P more precisely with an oriented (i.e. non-axis-aligned) bounding box. This can be done using the following method.

Computing an oriented bounding box

The directions of an oriented bounding box can be computed using Principal Component Analysis [69]. We first choose a set $S = \{\mathbf{s}^1, \mathbf{s}^2, \ldots, \mathbf{s}^m\}$ of m points[4] in the polyhedron P, such that $m \geq n$. We defer a discussion on how this point set is selected to the end of this section. PCA is used to find an orthogonal basis that best represents the point set S. More concretely, we use $\bar{\mathbf{s}}$ to be the mean of S, that is $\bar{\mathbf{s}} = \frac{1}{m}\sum_{i=1}^{m} \mathbf{s}^i$ and we denote $\tilde{\mathbf{s}}_{i,j} = \mathbf{s}_i^j - \bar{\mathbf{s}}_i$. For two points $\mathbf{s}^i$ and $\mathbf{s}^j$ in S, the covariance of their translated points is: $cov(\mathbf{s}_i, \mathbf{s}_j) = \frac{1}{m-1}\sum_{k=1}^{m} \tilde{\mathbf{s}}_{k,i}\tilde{\mathbf{s}}_{k,j}$. Then, we define the co-variance matrix C such that the element $C_{ij} = cov(\mathbf{s}^i, \mathbf{s}^j)$.

[4]By abuse of notation we use m to denote both the number of template constraints and the number of points here.

The n largest singular values of C provide the orientation of the bounding box. More concretely, since C is symetric, by singular value decomposition, we have $C = U\Lambda U^T$ where Λ is the matrix of singular values. The axes of the bounding box are hence determined by the first n columns of the matrix U, and its centroid is $\bar{\mathbf{s}}$.

We now discuss how to select the set S. When the vertices of P are available, we can include them in the set. However, if P is given as a template polyhedron, this requires computing the vertices which is expensive. Moreover, using only the vertices, when their distribution do not represent the geometric form of the polyhedron, may cause a large approximation error, since the resulting principal directions are not the ones along which the points inside P are mostly distributed. To remedy this, we sample points inside P as follows. First, we compute an axis-aligned bounding box of P (this can be done by solving $2n$ linear programming problems). We then uniformly sample points inside this bounding box and keep only the points that satisfy the constraints of P. Uniform sampling on the boundary of P in general enables a better precision. More detail on this can be found in [41].

3.5.3 Image computation algorithm

The following algorithm summarizes the main steps of our method for over-approximating the image of a bounded polyhedron $P \subset \mathbb{R}^n$ by the polynomial π. The templates are an input of the algorithm. In the current implementation of the algorithm, the templates can be fixed by the user, or the templates forming regular sets are used.

Algorithm 6 Over-approximating $\pi(P)$

/ Inputs: convex polyhedron P, polynomial π, templates H */*
$B = PCA(P)$ */* Compute an oriented bounding box */*
$\tau = UnitBoxMap(B)$ */* Compute the function mapping the unit box B_u to B */*
$\gamma = \pi \; o \; \tau$
$(u, l) = BoundFunctions(\gamma)$ */* Compute the affine bound functions */*
$\bar{\mathbf{d}} = PolyApp(u, l, H)$ */* Compute the coefficient vector $\mathbf{d}$ */*
$Q = \langle H, \bar{\mathbf{d}} \rangle$ */* Form the template polyhedron and return it */*
Return(Q)

The role of the procedure PCA is to compute an oriented bounding box B that encloses P. The procedure $UnitBoxMap$ is then used to determine the affine function τ that maps the unit box B_u at the origin to B. This affine function is composed with the polynomial π, the result of which is the polynomials γ.

The affine lower and upper bound functions l and u of γ are then computed, using the Bernstein expansion. The function *PolyApp* determines the polyhedral coefficient vector $\mathbf{d}$ by solving the linear programs in (3.23) with u, l and the optimization domain is B_u. The polyhedral coefficient vector $\mathbf{d}$ are then used to define a template polyhedron Q, which is the result to be returned.

Based on the analysis so far, we can state the correctness of Algorithm 6.

Theorem 8. *Let $\langle H, \bar{\mathbf{d}} \rangle$ be the template polyhedron returned by Algorithm 6. Then $\pi(P) \subseteq \langle H, \bar{\mathbf{d}} \rangle$.*

We remark that u and l are upper and lower bound functions of γ with respect to B_u. It is not hard to see that $\tau^{-1}(P) \subseteq B_u$ where τ^{-1} is the inverse of τ. Using the property of bound functions, u and l are also bound functions of γ with respect to $\tau^{-1}(P)$. Hence, if we solve the optimization problem over the domain $\tau^{-1}(P)$ (which is often smaller than B_u), using Lemma 10, the resulting polyhedron is still an over-approximation of $\pi(P)$. This remark can be used to obtain more accurate results.

3.5.4 Approximation errors and complexity

We finish this section by briefly discussing the precision and complexity of our method. A more detailed analysis can be found in [41]. The approximation errors are caused by the use of bound functions, the bounding box approximation and template polyhedra.

It can be proven that in one dimensional cases, the error between the bound functions and the original polynomial is quadratic in the length of box domains. This quadratic convergence seems to hold for higher dimensional cases in practice, as shown in [55]. We conjecture that there exists a subdivision method of the box B which allows a quadratic convergence of the error. This subdivision method is similar to the one used for finding roots of a polynomial with quadratic convergence [87].

On the other hand, a polyhedron can be approximated by a set of non-overlapping oriented boxes with arbitrarily small error. Then, for each box, we compute a bounding function, with which we then compute a coefficient for each template. Finally, for each template, we take the largest coefficient to define the template polyhedron. Since the boxes are smaller, the bounding functions are more precise, we can thus improve the coefficients as much as desired.

Concerning the error inherent to the approximation by template polyhedra, it can be controlled by fine-tuning the number of template constraints. If using this method with a sufficient number of templates to assure the same precision as the convex hull in our previous Bézier method [36], then the convergence

of both methods are quadratic. However the Bezier method requires expensive convex-hull and triangulation operations, and geometric complexity of resulting sets may grow step after step. Combining template polyhedra and bounding functions allows a good accuracy-cost compromise.

We now discuss the complexity of our algorithm. Let each polynomial π_i be written as $\pi_i = \sum_{\mathbf{j} \in I_i} a^i_{\mathbf{j}} \mathbf{x}^{\mathbf{j}}$ where each $a^i_{\mathbf{j}} \neq 0$. We denote by $\#(\pi_i)$ the number of such monomials in π_i, *i.e.* the cardinality of I_i. Let K be the maximal number of monomials in each π_i, that is $K = \max_{i \in \{1,\ldots,n\}} \#(\pi_i)$.

First, we remark that the computation of the bound functions and PCA only requires manipulating matrices and linear equations. Additionally, linear programming with n variables and m constraints can be solved in polynomial time $O((mn)^{3.5})$.

The proofs of the following results can be found in [41]. The complexity of the computation of the bound functions is $\mathcal{O}(n^4 + Kn^5)$. The complexity of the computation of an affine function τ mapping the unit box to an oriented box is $\mathcal{O}(nn^{3.5})$ (due to n LP problems). The approximation using a template polyhedron requires solving $2n$ LP problems over the unit box and has thus the complexity $\mathcal{O}(2n(2nn)^{3.5})$ (see (3.23)).

The exponentiel factor in the complexity of our algorithm comes from the composition of π and an affine transformation τ. Let us suppose that we use a simple composition algorithm whose complexity depends on the number of monomials[5]. The following theorem shows some cases for which our algorithm has a polynomial time complexity.

Theorem 9. *If π and τ satisfy two conditions:*

$$(1) \quad \forall i \in \{1,\ldots,n\} : \sum_{\mathbf{j} \in I_i} \sum_{k=1}^{n} \mathbf{j}_k = O(ln(n))$$
$$(2) \quad \forall i \in \{1,\ldots,n\} \#(\tau_i) \leq 2$$

then the composition π o τ has in total $\mathcal{O}(Kn^3)$ monomials, thus the computation of π o τ can be done in $\mathcal{O}(Kn^3)$.

The proof of this can be found in [41]. Note that if we use axis-aligned bounding boxes, each component of τ always have 2 terms, and the second condition of the theorem are satisfied. However, this polynomial time complexity w.r.t. the dimension may not hold if we use oriented bounding boxes to over-approximate the reachable sets before mapping them to the unit box. Indeed, in this case each component of τ may have more than 2 terms.

[5] Advanced composition algorithms, *e.g.* [96], can achieve a better complexity.

Concerning the complexity w.r.t. the number of iterations, if the number of template constraints is constant, we can prove that the complexity depends linearly on the number of iterations (see more in [41]).

Experimental results

We have implemented our method in a prototype tool using the template polyhedral library developed by S. Sankaranarayanan [95] and the library **lpsolve** for linear programming. In the following, we demonstrate the method with two examples: a control system (modelled as a hybrid system) and a biological system (modelled as a continuous system). The time efficiency of the tool is also evaluated by considering using a number of randomly generated polynomials.

A control system. The control system we consider is the Duffing oscillator [70, 51]. Its continuous-time dynamics is described by $\ddot{y}(t)+2\zeta\dot{y}(t)+y(t)+y(t)^3 = u(t)$, where $y \in \mathbb{R}$ is the state variable and $u \in \mathbb{R}$ is the control input. The damping coefficient $\zeta = 0.3$. In [51], using a forward difference approximation with a sampling period $h = 0.05$, this system is approximated by the following discrete-time model: $x_1(k+1) = x_1(k) + hx_2(k)$, $x_2(k+1) = -hx_1(k) + (1 - 2\zeta h)x_2(k) + hu)k) - hx_1(k)^3$.

In [51], an optimal predictive control law $u(k)$ was computed by solving a parametric polynomial optimization problem. In Figure 3.5.4 one can see the phase portrait of the system under this control law and without it (i.e. $\forall kge0\ u(k) = 0$) is shown . We model this control law by the following switching law with 3 modes: $u(k) = 0.5k$ if $0 \leq k \leq 10$, $u(k) = 5 - 0.5(k - 10)/3$ if $10 < k \leq 40$, and $u(k) = 0$ if $k > 40$. The controlled system is thus modelled as a hybrid automaton with 3 discrete modes. The result obtained using our tool on this system is shown in Figure 3.5.4, which is coherent with the phase portrait in [51]. The initial set is a ball with radius $1e - 04$. The number of template constraints is 100. In addition to the reachable set after 120 steps (computed after $3s$), in Figure 3.5.4, we also illustrate the approximation error by visusalizing the template polyhedron after the first step and a cloud of exact points (obtained by sampling the initial set and applying the polynomial to the sampled points).

A biological system. The second example is the well-known Michaelis-Menten enzyme kinetics [67], where E is the concentration of an enzyme that combines with a substrate S to form an enzyme substrate complex ES. In the next step, the complex can be dissociated into E and S or it can further proceed to form a product P.

This pathway kinetics can be described by the following ODEs where x_1, x_2, x_3 and x_4 are the concentrations of S, E, ES and P: $\dot{x}_1 = -\theta_1 x_1 x_2 + \theta_2 x_3$,

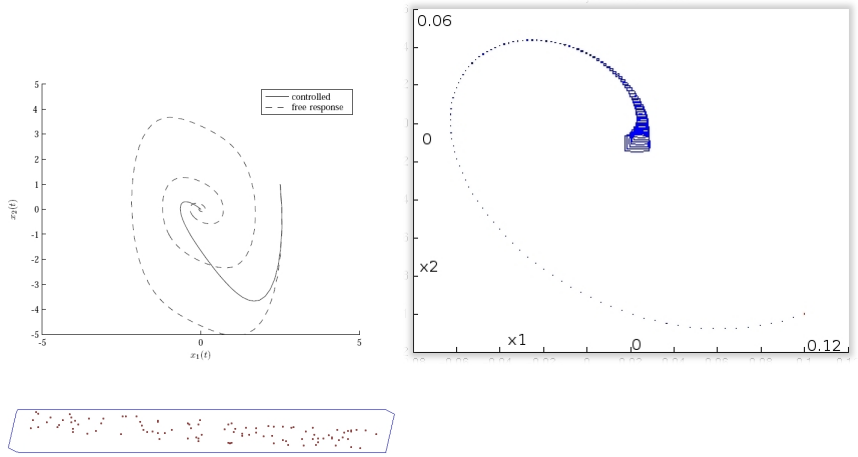


Figure 3.6: The Duffing oscillator: phase portrait, the reachable set, and the reachable set after the first step.

$\dot{x}_2 = -\theta_1 x_1 x_2 + (\theta_2 + \theta_3)x_3$, $\dot{x}_3 = \theta_1 x_1 x_2 + (\theta_2 + \theta_3)x_3$, $\dot{x}_4 = \theta_3 x_3$. Using a second order Runge Kutta discretization with time step 0.3, we obtain

$$\begin{aligned}
\pi_1(\mathbf{x}) &= x_1 - 0.053838x_1x_2 + 0.001458x_1^2x_2 + 0.001458x_1x_2^2 - 3.9366e - 5.x_1^2x_2^2 \\
&\quad +0.005775x_3 - 0.002025x_1x_3 - 0.000162x_2x_3 + 5.9049e - 5x_1x_2x_3 - 6.075e - 6x_3^2 \\
\pi_2(\mathbf{x}) &= x_2 - 0.051975x_1x_2 + 0.001458x_1^2x_2 + 0.001458x_1x_2^2 - 3.9366e - 5x_1^2x_2^2 + 0.0721875x_3 \\
&\quad -0.002025x_1x_3 - 0.000162x_2x_3 + 5.9049e - 5x_1x_2x_3 - 6.075e - 6x_3^2 \\
\pi_3(\mathbf{x}) &= 0.051975x_1x_2 - 0.001458.x1^2x_2 - 0.001458x_1x_2^2 + 3.9366e - 5x1^2x_2^2 \\
&\quad +0.927812x_3 + 0.002025x_1x_3 + 0.000162x_2x_3 - 5.9049e - 5x_1x_2x_3 + 6.075e - 6x_3^2 \\
\pi_4(\mathbf{x}) &= 0.001863x_1x_2 + 0.0664125x_3 + x_4.
\end{aligned}$$

The reachable set computed for all the initial states inside a ball centered at $(12, 12, 0, 0)$ with radius $1e-0.4$ is shown in Figure 3.7. The number of template constraints is 60. In order to compare with the result in [67], the figures depict the evolution of each variable for the first 10 steps (the horizontal axis is time). In the vertical axis, the minimal and maximal values of the variables are shown. This result is conherent with the simulation result in [67]. The computation time for 20 steps is $3.7s$.

Randomly generated systems. In order to evaluate the performance of our method, we tested it on a number of randomly generated polynomials in various dimensions and maximal degrees (the maximal degree is the largest degree for all variables). For a fixed dimension and degree, we generated different examples to estimate an average computation time. In the current implementation, polynomial composition is done symbolically, and we do not yet exploit the possibility

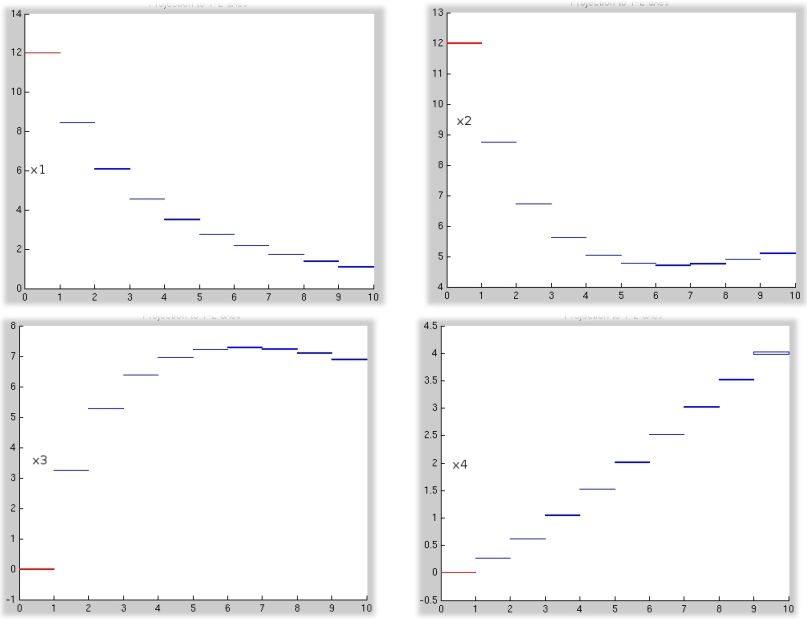


Figure 3.7: Michaelis-Menten enzyme kinetics. The evolution of the reachable set after 10 steps

of sparsity of polynomials (in terms of the number of monomials). The computation time shown in Figures 3.8-3.9 does not include the time for polynomial composition. Note that the computation time for 7-variate polynomials of degree 3 is significant, because the randomly generated polynomials have a large number of monomials; however, practical systems often have a much smaller number of monomials. As expected, the computation time does not grows linearly w.r.t. the number of steps. This can be explained by the use of template polyhedra where the number of constraints can be chosen according to required precisions and thus control better the complexity of the polyhedral operations, compared to general convex polyhedra. Indeed, when using general polyhedra, the operations such as convex hull may increase their geometric complexity (roughly described by the number of vertices and constraints).

The above experimental results show a significant improvement in efficiency compared to our previously developed method using Bézier techniques [36] and box splines [**?**]. These encouraging results also show an important advantage of the method: thanks to the use of template polyhedra as a symbolic set representations, the complexity and precision of the method are more controllable than those using general polyhedra.

There are a number interesting directions to explore. Indeed, different tools from geometric modeling could be exploited to improve the efficiency of the method. For example, polynomial composition can be done for sparse poly-

dim	degree	nb steps	time (s)	nb constraints = 5 x dim
2	2	1	0.02 s	10
2	2	10	0.18 s	
2	2	100	1.82 s	
2	3	1	0.01 s	
2	3	10	0.32 s	
2	3	100	1.98 s	
3	2	1	0.01 s	15
3	2	10	0.37 s	
3	2	100	3.64 s	
3	3	1	0.02 s	
3	3	10	0.3 s	
3	3	100	3.84 s	
4	2	1	0.03 s	20
4	2	10	0.6 s	
4	2	100	6.64 s	
4	3	1	0.06 s	
4	3	10	0.67 s	
4	3	100	6.41 s	

Figure 3.8: Computation time for randomly generated polynomial systems

dim	degree	nb steps	time (s)	nb constraints = 5 x dim
5	2	1	0.12 s	25
5	2	10	1.02 s	
5	2	100	10.2 s	
5	3	1	0.37 s	
5	3	10	1.22 s	
5	3	100	10.74 s	
6	2	1	0.61 s	30
6	2	10	2.01 s	
6	2	100	16.85 s	
6	3	1	3.99 s	
6	3	10	42.47s	
6	3	100	400.30s	
7	2	1	4.11 s	35
7	2	10	60.12s	
7	2	100	568.35s	
7	3	1	30.01s	
7	3	10	345.68s	
7	3	100	3856.4s	

Figure 3.9: Computation time for randomly generated polynomial systems

nomials more efficiently using the blossoming technique [96]. In addition to more experimentation on other hybrid systems case studies, we intend to explore a new application domain, which is verification of embedded control software. In fact, multivariate polynomials arise in many situations when analyzing programs that are automatically generated from practical embedded controllers.

3.6 Other results

3.6.1 Multi-affine systems

On the other hand, we have developed a reachability analysis method specialized for multi-affine systems. The interest of these systems also comes from its usefulness in a systematic approximation of a non-linear system using hybridization (see Section **??**). This method combines the idea of abstraction by projection to transform some multi-affine systems to an uncertain bilinear system. A method for uncertain bilinear systems using the Pontryagin principle from optimal control is then applied to handle the resulting system.

This work was done in collaboration with Eugene Asarin from LIAFA (Paris) and was published in the proceeding of the conference HSCC 2004 (Hybrid Systems - Computation and Control) [**?**].

3.6.2 Set integration and template polyhedra

This work has been carried out in collaboration with Sriram Sankaranarayanan and Franjo Ivancic at NEC Laboratories (Princeton, USA).

We proposed a symbolic technique for the verication of hybrid systems using template polyhedra as a set representation. Our technique uses *higher-order Taylor series approximations* along with repeated optimization problems to bound the terms in the Taylor series expansion. The location invariant is used to enclose the remainder term of the Taylor series, and thus make our technique sound.

Another result obtained along this line is a *policy iteration technique* that computes an over-approximation of the time trajectories of a system using template polyhedra. Given a set of template expressions, we show the existence of a smallest template polyhedron that is a positive invariant w.r.t to the dynamics of the continuous variables, and hence, an over-approximation the time trajectories. Thus, we derive a time elapse operator for template polyhedra using policy iteration that computes tight over-approximations of the time trajectories. We also exploit the result of the policy iteration to improve the precision of Taylor series-based flowpipe construction.

These techniques were implemented by Sriram Sankaranarayanan as a part of the tool **TimePass** for verifying reachability properties of ane hybrid automata, with promising results on different benchmarks. These results were published at HSCC 2008 and TACAS 2008 [**?**, **?**].

Chapter 4

Model-based testing of hybrid systems

4.1 Context

Although the verification methods for hybrid systems have been successfully applied to a number of interesting case studies, their applicability is still limited to systems of small size due to the complexity of formal verification. My attention was thus drawn towards alternative approaches which can be applied to higher dimensional systems with more complex dynamics. One of these is testing. This choice was motivated by my interest in circuit design and the experience I had gained from my previous work on verification of analog circuits. Testing is indeed the main method used in industry for circuit design. Additionally, testing is a "traditional" research topic of a research group in VERIMAG which developed the well-known tool TGV for the generation of conformance test suites for protocols [**?**], which are based on algorithms coming from verification technology. In order to make this methodology meaningful for hybrid systems with particular features (one of which is the infinity of the state and input spaces), I tried to first tackle two important problems: define a test coverage measure and how to generate tests with good test coverages.

The geometric nature of the continuous state space led me to the idea of using the notion of geometric discrepancy [19]. At the same time, I was influenced by the algorithms for path planning from robotics, in particular the RRT algorithm [82], which I believed useful for the exploration of the state space of a hybrid system. In 2005, with my newly recruited PhD student Tarik Nahhal, we began to explore this direction and developed a new version of RRT which is guided by this test coverage. This constituted the main topics of the PhD thesis of Tarik, defended in October 2007.

Project VAL-AMS. In 2006, together with the researchers of **BIPOP** in Inria Rhônes-Alpes (who were developing a novel technique for circuit simulation) and the researchers of the laboratory **LJK** (who shared the interest in hybrid systems), we elaborated a project, called VAL-AMS, for High Confidence Validation of Analog and Mixed-Signal Circuits and obtained an ANR funding for two years. I was the coordinator of this project and much of my work presented in this chapter was carried out in collaboration with the project partners.

The project intends to build an experimental platform for validating the correctness of analog and mixed-signal circuits, a component of increasing importance for the functioning of modern embedded system. The platform will combine two technologies currently being developed by the partners: an efficient and physically-accurate simulator for large analog and mixed-signal circuits and the methods for covering the state space of such circuits by choosing appropriate input signals.

In this project, we were responsible for the work package for the development of search-based methods for validating large-scale continuous and hybrid systems. We recently combined our state space exploration method, developed in the above mentioned test generation framework, with the numerical integration engine of the platform SICONOS [**?**], developed by BIPOP Inria Rhône-Alpes. The prototype tool was used to analyze a number of circuit benchmarks. Our tool development within this project also includes generation of circuit equations from models and optimization.

4.2 Introduction

Testing is a validation approach, which can be used for much larger systems and is a standard tool in industry, although it can only reveal an error but does not permit proving its absence. Although testing has been well studied in the context of finite state machines (e.g. [104] and references therein) and, more recently, of real-time systems (e.g. [78, 5, 24]), it has not been much investigated for continuous and hybrid systems. Therefore, a question of great interest is to bridge the gap between the verification and testing approaches, by defining a formal framework for testing of hybrid systems and developing methods and tools that help automate the testing process.

Classical model-based testing frameworks use Mealy machines or finite labeled transition systems and their applications include testing of digital circuits, communication protocols and software. Recently, these frameworks have been extended to real-time systems. However, hybrid systems conformance testing has not yet been well investigated. A number of special characteristics of hybrid systems make their testing particularly challenging, in particular the infiniteness of

the state space of a hybrid system and of the input space. In general, in order to test an open system, one first needs to feed an input signal to the system and then check whether the behavior of the system induced by this input signal is as expected. When there is an infinite number of possible input signals, it is important to choose the ones that lead to interesting scenarios (with respect to the property/functionality to test).

In this work we adopt a model-based testing approach. This approach allows the engineer to perform validation during the design, where detecting and correcting errors on a model are less expensive than on an implementation.

The main results we present in this chapter can be summarized as follows.

- **Formal framework for conformance testing**. We propose a formal framework for conformance testing of hybrid systems. This framework uses the commonly-accepted hybrid automaton model and is defined according to the international standard for formal conformance testing [103]. This framework allows, on one hand, to formally reason about the conformance relation between a system under test and a specification, and on the other hand, to develop test generation algorithms.

- **Novel test coverage measure**. This is a challenging problem in testing. Intuitively, test coverage is a way to characterize the relation between the number and the type of tests to execute and the portion of the system's behavior effectively tested. The classical notions of coverage, introduced mainly for software testing (such as statement coverage, if-then-else branche coverage, path coverage) are unsuitable for the behaviors of a hybrid system defined as solutions of some differential equations. We thus proposed a novel coverage measures, which on one hand reflect the testing objectives and, on the other hand, can be efficiently computed. This measure is based on the equidistribution degree of a set of states over the state space and furthermore can be used to guide the test generation process.

- **Coverage-guided test generation**. We first developed a test generation algorithm which is based on the RRT (Rapidly-exploring Random Tree) algorithm [80], a probabilistic motion planning technique in robotics. This RRT algorithm has been successful in finding feasible trajectories in motion planning. We then proposed an test generation algorithm based on the above mentioned coverage measures.

- **Disparity-guided test generation.** Besides, we introduced a new notion of disparity between two point sets, in order to tackle "blocking" situations the RRT algorithms may enter. Indeed, in order to increase the coverage, the algorithm may try to explore the directions which are not reachable by the system's dynamics. We can detect such situations by comparing the

distribution of the goal states and the visited states, using their disparity. If the disparity is large, it means that the visited states do not follow the goal states, which indicates that the goal states might not reachable and we should change the goal state sampling strategy.

- **Actuator and sensor imprecision.** Due to the limitations of practical actuators and sensors, the tester cannot realize exactly an input value specified as a real-valued vector as well as measure exactly the state of the system. We handle this using sensitivity analysis.

- **Tool development.** We have implemented a tool for conformance testing of hybrid systems, called **HTG**. The core of the tool is the implementation of the coverage-guided test case generation algorithm and the methods for estimating coverage measures.

- **Applications**. In particular, besides traditional applications of hybrid systems, we explore a new domain which is analog and mixed signal circuits. Indeed, hybrid systems provide a mathematical model appropriate for the modeling and analysis of these circuits. The choice of this application domain is motivated by the need in automatic tools to facilitate the design of these circuits which, for various reasons, is still lagging behind the digital circuit design. Besides hybrid automata described using a textual language, the tool can accept as input electrical circuits specified using SPICE netlists. We have thus treated a number of case studies from control applications as well as from analog and mixed signal circuits. The experimental results obtained using the tool **HTG** show its applicability to systems with complex dynamics and its scalability to high dimensional systems.

Before presenting these results, we first describe our conformance testing framework and test coverage measure.

4.3 Model

Conformance testing provides a means to assess the correctness of an implementation with respect to a specification by performing experiments on the implementation and observing its responses. When the specification is described by a formal model, the international standard "Formal Methods in Conformance Testing" (FMCT) [103] provides a framework of conformance testing, which includes abstract concepts (such as conformance, test cases, test execution, test generation), and the requirements on these concepts.

In this work, as a formal model for embedded systems, use hybrid automata [7]. Intuitively, a hybrid automaton is an automaton augmented with continuous variables that evolve according to some differential equations.

Definition 2 (Hybrid automaton)**.** *A* hybrid automaton *is a tuple* $\mathcal{A} = (\mathcal{X}, Q, E, F, \mathcal{I}, \mathcal{G}, \mathcal{R})$ *where*

- $\mathcal{X}$ *is the continuous state space and is a bounded subset of* $\mathbb{R}^n$*;*
- Q *is a (finite) set of locations (or discrete states);*
- $E \subseteq Q \times Q$ *is a set of discrete transitions;*
- $F = \{F_q \mid q \in Q\}$ *such that for each* $q \in Q$*,* $F_q = (f_q, U_q)$ *defines a differential equation:*
$$\dot{x}(t) = f_q(x(t), u(t))$$
where $u(\cdot) \in \mathcal{U}_q$ *is an admissible input function of the form* $u : \mathbb{R}^+ \to U_q \subset \mathbb{R}^m$*. We assume that all* f_q *are Lipschitz continuous*[1]*. The admissible input functions* $u(\cdot)$ *are piecewise continuous.*
- $\mathcal{I} = \{\mathcal{I}_q \subseteq \mathcal{X} \mid q \in Q\}$ *is a set of staying conditions;*
- $\mathcal{G} = \{\mathcal{G}_e \mid e \in E\}$ *is a set of guards such that for each discrete transition* $e = (q, q') \in E$*,* $\mathcal{G}_e \subseteq \mathcal{I}_q$*;*
- $\mathcal{R} = \{\mathcal{R}_e \mid e \in E\}$ *is a set of reset maps. For each* $e = (q, q') \in E$*,* $\mathcal{R}_e : \mathcal{G}_e \to 2^{\mathcal{I}_{q'}}$ *defines how* x *may change when* $\mathcal{A}$ *switches from* q *to* q'*.*
- *The initial state of the automaton is denoted by* (q_{init}, x_{init})*.*

A *hybrid state* is a pair (q, x) where $q \in Q$ and $x \in \mathcal{X}$. The hybrid state space is $\mathcal{S} = Q \times \mathcal{X}$. In the rest of the paper, for brevity, we often use "state" to refer to a hybrid state. In location q, the evolution of the continuous variables is governed by $\dot{x}(t) = f_q(x(t), u(t))$.

A state (q, x) of $\mathcal{A}$ can change in two ways as follows: (1) by a *continuous evolution*, the continuous state x evolves according to the dynamics f_q while the location q remains constant; (2) by a *discrete evolution*, x satisfies the guard condition of an outgoing transition, the system changes the location by taking this transition and possibly changing the values of x according to the associated reset map. More formally, continuous and discrete evolutions are defined as follows.

Given a real number $h > 0$ and an admissible input function $u(\cdot) \in \mathcal{U}_q$, $(q, x) \overset{u(\cdot),h}{\to} (q, x')$ is a *continuous evolution* at the location q from the hybrid state (q, x) to (q, x'), iff $x' = \xi_{x,\mu(\cdot)}(h)$ and for all $t \in [0, h] : \xi_{x,\mu(\cdot)}(t) \in \mathcal{I}_q$, where

[1]The function f_q is Lipschitz continuous if there exists a constant K such that $\forall x, y : ||f_q(x) - f_q(y)|| \leq K||x - y||$, where $||\cdot||$ is some norm of $\mathbb{R}^n$. This condition ensures the existence and uniqueness of solutions of the differential equations.

$\xi_{x,\mu(\cdot)}(t)$ is the solution of the differential equation at the location q with the initial condition x and under the input $\mu(\cdot)$. In other words, x' is reached from x under the input $u(\cdot)$ after exactly h time, and we say that $u(\cdot)$ is *admissible* starting at (q, x) for h time.

Given a transition $e = (q, q') \in E$, $(q, x) \xrightarrow{e} (q', x')$ is a discrete evolution iff $x \in \mathcal{G}_e$ and $x' \in \mathcal{R}_e(x)$. We say that (q', x') is reachable from (q, x) and the discrete transition e is admissible at (q, x). Unlike *continuous evolutions*, *discrete evolutions* are instantaneous, which means that they do not take time.

It is important to note that this model allows to capture *non-determinism* in both continuous and discrete dynamics. The non-determinism in continuous dynamics is caused be the uncertainty in the input function. For example, when the input is used to model some external disturbances or modelling errors, we do not know the exact input fucntion but only its range. The non-determinism in discrete dynamics is caused by the fact that at some states it is possible for the system to stay at the current location or to switch to another one. In addition, multiple transitions can be enabled at some states. This non-determinism is useful for describing disturbances from the environment and imprecision in modelling and implementation. We assume that the hybrid automata we consider are non-Zeno[2].

4.4 Conformance testing

In this section, we define the main concepts of our testing framework. Our testing goal is to make statements about the conformance relation between the behaviors of an implementation or, more generally, a system under test (SUT) and a specification. The specification is formal and is modeled by a hybrid automaton. The conformance will be defined as a relation $\approx \subseteq \Xi \times HA$ where Ξ is a set of SUTs of interest, and HA is a set of hybrid automata modeling the specifications of interest. The systems under test are physical systems, but it can be assumed that all the SUTs in Ξ can be described by a class of formal models, which is a set HA_s of hybrid automata. It is important to note that we assume that a model for each SUT in Ξ exists but do not assume that we know it. This assumption enables us to include the system under test in our formal framework and to express formally the conformance relation $\approx$ between the models of the SUTs and the specifications, that is $\approx \subseteq HA_s \times HA$. Note that here we use the same notation $\approx$ for the relation between the real SUT and the specification and the relation between the model of the SUT and the specification. A system under test $S_{ut} \in \Xi$ is said to *conform* to a specification $\mathcal{A} \in HA$ if and only if

[2]A Zeno behavior can be described informally as the system making an infinite number of discrete transitions in a finite amount of time.

the model $\mathcal{A}_s \in HA_s$ of S_{ut} is related to $\mathcal{A}$ by $\approx$, that is, $\mathcal{A}_s \approx \mathcal{A}$.

The system under test often operates within some environment. In our testing framework, a tester plays the role of the environment and it performs experiments on the SUT in order to study the conformance relation between the SUT and the specification. Such an experiment is called a *test*, and its specification is called a *test case*. A set of test cases is called a *test suite*, and the process of applying a test to a system under test is called a *test execution*. The tester works as follows. It emits the control inputs to the SUT and measures the observation sequences in order to produce a verdict $\nu \in \{P, F\}$ where P means 'pass' (the observed behavior is allowed by the specification), F means 'fail' (the observed behavior is not allowed by the specification). We continue by giving a detailed description of conformance relation. The problem of how to perform test executions and derive verdicts is discussed at the end of this section.

4.4.1 Conformance relation

Recall that the specification is modeled by a hybrid automaton $\mathcal{A}$ and the system under test SUT by another hybrid automaton $\mathcal{A}_s$. For brevity, when the context is clear, we often say "the system under test" to mean the automaton $\mathcal{A}_s$. To define the conformance relation, we need the notions of *inputs and observations*.

An input of the system which is controllable by the tester is called a *control input*; otherwise, it is called a *disturbance input*. We consider the following input actions.

Continuous input action. All the continuous inputs are assumed to be controllable by the tester. Since we want to implement the tester as a computer program, we are interested in piecewise-constant input functions; indeed, a computer cannot generate a function from reals to reals. Hence, a *continuous control action* $(\bar{u}_q, h)$, where $\bar{u}_q$ is the value of the input and h is the *duration*, specifies that the automaton continues with the continuous dynamics at the location q under the input $u(t) = \bar{u}_q$ for exactly h time. We say that $(\bar{u}_q, h)$ is *admissible at* (q, x) if the input function $u(t) = \bar{u}_q$ for all $t \in [0, h]$ is admissible starting at (q, x) for h time.

Discrete input actions. The discrete transitions are partitioned into controllable corresponding to discrete control actions and uncontrollable corresponding to discrete disturbance actions. The tester emits a discrete control action to specify whether the system should take a controllable transition (among the enabled ones) or continue with the same continuous dynamics. In the former case, it can also control the values assigned to the continuous variables by the associated

reset map. For simplicity of explanation, we will not consider non-determinism caused by the reset maps. Hence, we denote a discrete control action by the corresponding transition, such as (q, q').

We use the following assumption about the inputs: continuous control actions are of higher priority than discrete actions. This means that after a continuous control action $(\bar{u}_q, h)$ is applied, no discrete transitions can occur during h time, i.e. until the end of that continuous control action. This assumption is not restrictive, from a modeling point of view. Indeed, by considering all the possible values of h we can capture the cases where a discrete transition can occur before the termination of a continuous control action.

In this work, we are only interested in testing *non-blocking behaviors*, we thus need the notion of *admissible input sequences*. We write $(q, x) \xrightarrow{\iota} (q', x')$ to indicate that (q', x') is reached after applying the input action ι to the state (q, x).

Definition 3 (Admissible input sequence)**.** *For a state (q, x), a sequence of input actions $\omega = \iota_0, \iota_1, \ldots, \iota_k$ is admissible at (q, x) if*

- *ι_0 is admissible at (q, x), and*
- *for each $i = 1, \ldots, k$, let (q_i, x_i) be the state such that $(q_{i-1}, x_{i-1}) \xrightarrow{\iota_{i-1}} (q_i, x_i)$, then ι_i is admissible at (q_i, x_i).*

The sequence $(q, x), (q_1, x_1), \ldots, (q_k, x_k)$ is called the trace *starting at (q, x) under ω and is denoted by $\tau((q, x), \omega)$.*

We also write $(q, x) \xrightarrow{\omega} (q', x')$ to indicate that (q', x') is reached from (q, x) after ω. We also say that (q', x') is forward reachable from (q, x) and (q, x) is backward reachable from (q, x). In the rest of the paper, we simply say "reachable" to mean "forward reachable"; "backward reachable" is explicitly stated.

By the assumption about the inputs, uncontrollable discrete transitions cannot occur during a continuous control action. However, they can occur between control actions. Hence, the result of applying a control action is non-deterministic. To determine all possible traces that can be generated by applying a sequence of control actions, we need to define an *admissible sequence of control actions* (see [42]). Intuitively, this means that an admissible control action sequence, when being applied to the automaton, does not cause it to be blocked. We denote by $S_{\mathcal{C}}(\mathcal{A})$ the *set of all admissible control action sequences* for the hybrid automaton $\mathcal{A}$ starting at the initial state (q_{init}, x_{init}).

Observations

We use the following assumptions about the *observability* of the hybrid automata $\mathcal{A}$ and $\mathcal{A}_s$:

- The locations of the hybrid automata $\mathcal{A}$ and $\mathcal{A}_s$ are observable.
- We assume a subset $V_o(\mathcal{A})$ and $V_o(\mathcal{A}_s)$ of observable continuous variables of $\mathcal{A}$ and $\mathcal{A}_s$ respectively. In addition, we assume that $V_o(\mathcal{A}) \subseteq V_o(\mathcal{A}_s)$, which means that an observable continuous variable of $\mathcal{A}$ is also an observable variable of $\mathcal{A}_s$.

Since not all the continuous variables are observable, we need the following projection operator. The projection of a continuous state x of $\mathcal{A}$ on the observable variables $V_o(\mathcal{A})$ is denoted by $\pi(x, V_o(\mathcal{A}))$. The projection can be then defined for a trace as follows. The projection of a trace $\tau = (q_0, x_0), (q_1, x_1), (q_2, x_2) \ldots$ on $V_o(\mathcal{A})$ is

$$\pi(\tau, V_o(\mathcal{A})) = (q_0, \pi(x_0, V_o(\mathcal{A}))), (q_1, \pi(x_1, V_o(\mathcal{A}))), (q_2, \pi(x_2, V_o(\mathcal{A}))) \ldots .$$

A pair $(q, \pi(x, V_o(\mathcal{A}))$, where q is a location and x is the continuous state of the automation $\mathcal{A}$, is called an *observation*.

Definition 4 (Observation sequence)**.** *Let ω be an admissible control action sequence starting at the initial state (q_{init}, x_{init}) of $\mathcal{A}$. The set of* observation sequences *associated with ω is $S_{\mathcal{O}}(\mathcal{A}, \omega) = \{\pi(\tau, V_o(\mathcal{A})) \mid \tau \in Tr((q_{init}, x_{init}), \omega)\}$.*

Conformance relation

In the definition of the conformance relation between a system under test $\mathcal{A}_s$ and a specification $\mathcal{A}$, we assume that the set of all admissible control action sequences of $\mathcal{A}$ is a subset of that of $\mathcal{A}_s$, that is $S_{\mathcal{C}}(\mathcal{A}) \subseteq S_{\mathcal{C}}(\mathcal{A}_s)$. This assumption assures that the system under test can admit all the control action sequences that are admissible by the specification.

Definition 5 (Conformance)**.** *The system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$, denoted by $\mathcal{A} \approx \mathcal{A}_s$, iff*

$$\forall \omega \in S_{\mathcal{C}}(\mathcal{A}) : \ \pi(S_{\mathcal{O}}(\mathcal{A}_s, \omega), V_o(\mathcal{A})) \subseteq S_{\mathcal{O}}(\mathcal{A}, \omega).$$

Intuitively, the system under test $\mathcal{A}_s$ is conform to the specification $\mathcal{A}$ if under every admissible control action sequence, the set of observation sequences of $\mathcal{A}_s$ is included in that of $\mathcal{A}$. Note that we have assumed earlier that $S_{\mathcal{C}}(\mathcal{A}) \subseteq S_{\mathcal{C}}(\mathcal{A}_s)$, that is a control action sequence which is admissible for $\mathcal{A}$ is also admissible for

$\mathcal{A}_s$. Detecting the cases where the physical SUT does not admit some inputs that are allowed by the specification requires the ability to identify the states of the system from the observations. We do not consider this problem in this work.

Note that we use the trace inclusion to define conformance relation. In the literature of conformance testing for discrete systems, more complex relations are considered, for example input-output conformance relation (see [104]).

4.4.2 Test cases and test executions

In our framework, a *test case* is represented by a tree where each node is associated with an observation and each path from the root with an observation sequence. Each edge of the tree is associated with a control action. A physical *test execution* can be described as follows:

- The tester applies a test ζ to the system under test S_{ut}.
- It measures and records a number of observations.
- The observations are measured at the end of *each* continuous control action and after *each* discrete (disturbance or control) action.

This procedure is denoted by $exec(\zeta, S_{ut})$ which leads to an observation sequence, or a set of observation sequence if multiple runs of ζ are possible due to non-determinism. The above test execution process uses a number of implicit assumptions. First, observation measurements take zero time, and in addition, no measurement error is considered. Second, the tester is able to realize exactly the continuous input functions, which is often impossible in practice due to actuator imprecision. Under these assumptions, one can only test the conformance of *a model of the system under test* to the specification in discrete time. These issues need to be considered in order to address the actual testing of real systems and this will be discussed in Section 4.12.

We will focus on the case where each test execution involves a single run of a test case. The remaining question is how to interpret the observation sequences in order to produce a verdict. Let Ω denote the observation sequence domain. We thus define a verdict function: $\boldsymbol{v} : \Omega \to \{\mathbf{pass}, \mathbf{fail}\}$. Note that an observation sequence must cause a unique verdict. The observation sequences in Ω are grouped into two disjoint sets: the set O_p of observation sequences that cause a 'pass' verdict, the set O_f that cause a 'fail' verdict. Therefore, saying 'The system under test S_{ut} passes the test ζ' formally means $\boldsymbol{v}(exec(\zeta, S_{ut})) = \mathbf{pass}$. This can then be extended to a test suite.

We now discuss some important requirements for a test suite. A test suite T_s is called *complete* if for a given specification $\mathcal{A} \in HA$:

$$S_{ut} \approx \mathcal{A} \iff S_{ut} \text{ passes } T_s \tag{4.1}$$

This means that a complete test suite can distinguish exactly between all conforming and non-conforming systems. In practice, it is generally impossible to fulfill this requirement, which often involves executing an infinite test suite. A weaker requirement is *soundness*. A test suite is sound if a system does not pass the test suite, then the system is non-conforming. We can see that this requirement is weaker than completeness, since it corresponds only to the left-to-right implication in (4.1).

After defining all the important concepts, it now remains to tackle the problem of generating test cases from a specification model. In particular, we want the test suites to satisfy the *soundness requirement*. A hybrid automaton might have an infinite number of infinite traces; however, the tester can only perform a finite number of test cases in finite time. Therefore, we need to select a finite portion of the input space of the specification $\mathcal{A}$ and test the conformance of the system under test $\mathcal{A}_s$ with respect to this portion. The selection is done using a coverage criterion that we formally define in the next chapter. Hence, our testing problem is formulated as to automatically generate a set of test cases from the specification automaton to satisfy this coverage criterion.

4.5 Test coverage

Test coverage is a way to evaluate testing quality. More precisely, it is a way to relate the number of tests to carry out with the fraction of the system's behaviors effectively explored. As mentioned earlier, the classic coverage notions mainly used in software testing, such as statement coverage and branch coverage, path coverage (see for example [66, 104]), are not appropriate for the trajectories of continuous and hybrid systems defined by differential equations. However, geometric properties of the hybrid state space can be exploited to define a coverage measure which, on one hand, has a close relationship with the properties to verify and, on the other hand, can be efficiently computed or estimated. In this work, we are interested in *state coverage* and focus on a measure that describes how 'well' the visited states represent the reachable set of the system. This measure is defined using the *star discrepancy* notion in statistics, which characterises the uniformity of the distribution of a point set within a region. Note that the reachable sets of hybrid systems are often non-convex with complex geometric form, therefore considering only corner cases does not always cover the behaviors that are important for reachabilily properties, especially in high dimensions. Hence, for a fixed number of visited states (which reflects the computation cost

to produce a test suite), we want the visited states to be equidistributed over the reachable set as much as possible, since this provides a good representation of all possible reachable states.

4.5.1 Star discrepancy

We first briefly recall the star discrepancy. The star discrepancy is an important notion in equidistribution theory as well as in quasi-Monte Carlo techniques (see for example [19]). Recently, it was also used in probabilistic motion planning to enhance the sampling uniformity [82].

Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. Let $\mathcal{J}$ be the set of all sub-boxes J of the form $J = \prod_{i=1}^{n} [l_i, \beta_i]$ with $\beta_i \in [l_i, L_i]$ (see Figure 4.1). The local discrepancy of the point set P with respect to the sub-box J is defined as follows:

$$D(P, J) = \Big| \frac{A(P, J)}{k} - \frac{vol(J)}{vol(\mathcal{B})} \Big|$$

where $A(P, J)$ is the number of points of P that are inside J, and $vol(J)$ is the volume of the box J.

Definition 6 (Star discrepancy)**.** *The star discrepancy of a point set P with respect to the box $\mathcal{B}$ is defined as:*

$$D^*(P, \mathcal{B}) = sup_{J \in \mathcal{J}} D(P, J). \tag{4.2}$$

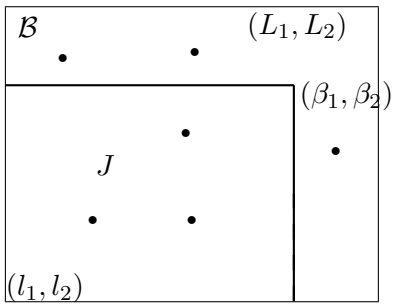


Figure 4.1: Illustration of the star discrepancy notion.

It is not hard to prove the following property of the star discrepancy [99].

Proposition 12. *The star discrepancy of a point set P with respect to a box $\mathcal{B}$ satisfies $0 < D^*(P, \mathcal{B}) \leq 1$.*

Intuitively, the star discrepancy is a measure for the irregularity of a set of points. A large value $D^*(P, \mathcal{B})$ means that the points in P are not much equidistributed over $\mathcal{B}$. When the region is a box, the star discrepancy measures how badly the point set estimates the volume of the box.

Example. To show an intuitive meaning of the star discrepancy, we use some sequences of 100 points inside a 2-dimensional unit box. The first example is the Faure sequence [50], a well-known low-discrepancy sequence (see Figure 4.2). As we can observe from the figure, this set of points 'covers well' the box, in the sense that the points are well-equidistributed over the box. Its star discrepancy value is 0.048. The second example is the Halton sequence [106] shown in Figure 4.3,

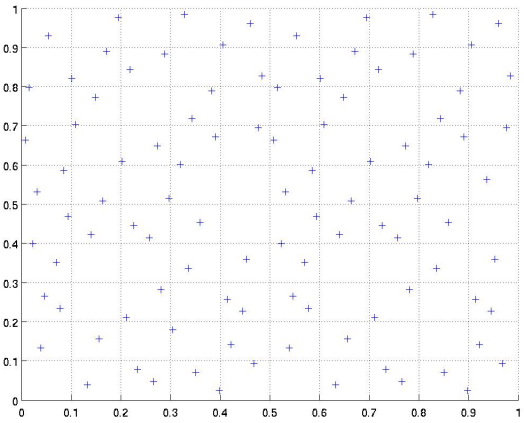


Figure 4.2: Faure sequence of 100 points. Its star discrepancy value is 0.048.

which is also a well-known low discrepancy sequence. The value of the star discrepancy of the Halton sequence is about 0.050, indicating that the Faure sequence is more equidistributed than the Halton sequence. The star discrepancy values of these two sequences are however close, and indeed visually it is hard to see from the figures which one is better equidistributed. We now give another example which is a sequence of 100 points generated by a pseudo-random function provided by the C library system. This sequence is shown in Figure 4.4, from which we can observe that this sequence is not well-equidistributed over the box. This is confirmed by its star discrepancy value 0.1. The star discrepancy is thus a meaningful measure that can characterize the uniformity quality of a point set distribution.

4.5.2 Coverage estimation

To evaluate the coverage of a set of states, we need to compute the star discrepancy of a point set, which is not an easy problem (see for example [44]). Many theoretical results for one-dimensional point sets are not generalizable to higher dimensions, and among the fastest algorithms, the one proposed in [44] has time complexity $\mathcal{O}(k^{1+d/2})$. In this work, we do not try to compute the star

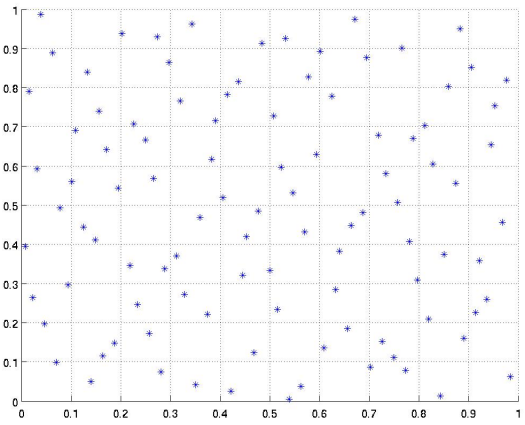


Figure 4.3: Halton sequence of 100 points. The star discrepancy value is 0.05.

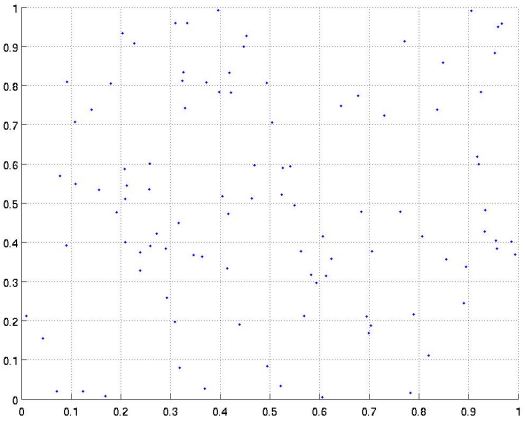


Figure 4.4: A sequence of 100 points generated by a pseudo-random function in the C library. Its star discrepancy value is 0.1.

discrepancy but approximate it by estimating a lower and upper bound. These bounds as well as the information obtained from their estimation are then used to decide which parts of the state space have been 'well explored' and which parts need to be explored more. This estimation is done using a method published in [99]. Let us briefly describe this method for computing the star discrepancy $D^*(P, \mathcal{B})$ of a point set P w.r.t. a box $\mathcal{B}$. Although in [99] the box $\mathcal{B}$ is $[0,1]^n$, we extended it to the case where $\mathcal{B}$ can be any full-dimensional box. Intuitively, the main idea of this estimation method is to consider a finite box partition of the box $\mathcal{B}$, instead of considering an infinite number of all sub-boxes as in the definition of the star discrepancy. Let $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. In what follows, we often call this box $\mathcal{B}$ the *bounding box*. We define a box partition of $\mathcal{B}$ as a set of boxes $\Pi = \{\square^1, \ldots, \square^m\}$ such that $\cup_{i=1}^m \square^i = \mathcal{B}$ and the interiors of the boxes $\square^i$ do not intersect. Each such box is called an *elementary box*. Given a box $\square = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$, we define $\square^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\square^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$ (see Figure 4.5 for an illustration).

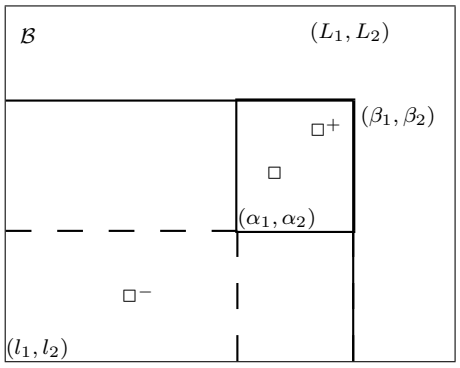


Figure 4.5: Illustration of the boxes $\square^-$ and $\square^+$.

For any finite box partition Π of $\mathcal{B}$, the star discrepancy $D^*(P, \mathcal{B})$ of the point set P with respect to $\mathcal{B}$ satisfies: $C(P, \Pi) \leq D^*(P, \mathcal{B}) \leq B(P, \Pi)$ where the upper and lower bounds are:

$$B(P, \Pi) = \max_{\square \in \Pi} \max\{\frac{A(P, \square^+)}{k} - \frac{vol(\square^-)}{vol(\mathcal{B})}, \frac{vol(\square^+)}{vol(\mathcal{B})} - \frac{A(P, \square^-)}{k}\} \quad (4.3)$$

$$C(P, \Pi) = \max_{\square \in \Pi} \max\{|\frac{A(P, \square^-)}{k} - \frac{vol(\square^-)}{vol(\mathcal{B})}|, |\frac{A(P, \square^+)}{k} - \frac{vol(\square^+)}{vol(\mathcal{B})}|\} \quad (4.4)$$

The imprecision of this approximation is the difference between the upper and lower bounds, which can be bounded by $B(P, \Pi) - C(P, \Pi) \leq W(\Pi)$ where

$$W(\Pi) = \max_{\square \in \Pi} (vol(\square^+) - vol(\square^-))/vol(\mathcal{B}) \quad (4.5)$$

Thus, one needs to find a partition Π such that this difference is small.

4.5.3 Hybrid systems test coverage

Since a hybrid system can only evolve within the staying sets of the locations, we are interested in the coverage with respect to these sets. For simplicity we assume that all the staying sets are boxes.

Definition 7 (Test coverage)**.** *Let* $\mathcal{P} = \{(q, P_q) \mid q \in Q \ \wedge \ P_q \subset \mathcal{I}_q\}$ *be the set of states. The coverage of* $\mathcal{P}$ *is defined as:*

$$Cov(\mathcal{P}) = \frac{1}{||Q||} \sum_{q \in Q} 1 - D^*(P_q, \mathcal{I}_q)$$

where $||Q||$ *is the number of locations in* Q*.*

If a staying set $\mathcal{I}_q$ is not a box, we can take the smallest oriented box that encloses it and apply the star discrepancy definition in (4.2) to that box after an appropriate coordinate transformation. We can see that a large value of $Cov(\mathcal{P})$ indicates a good space-covering quality. If $\mathcal{P}$ is the set of states visited by a test suite, our objective is to maximize $Cov(\mathcal{P})$.

4.6 Test generation

Our test generation is based on a randomized exploration of the reachable state space of the system. It is inspired by the Rapidly-exploring Random Tree (RRT) algorithm, which is a successful motion planning technique for finding feasible trajectories of robots in an environment with obstacles (see [79] for a survey). More precisely, we extend the **RRT** algorithm to hybrid systems. Furthermore, we combine it with a guiding tool in order to achieve a good coverage of the system's behaviors we want to test. To this end, we use the coverage measure defined in the previous section.

In this section, we describe the extension of the **RRT** algorithm to hybrid system, which we call the **hRRT** algorithm. The combination of the **hRRT** algorithm with the guiding tool will be explained in the next section.

The algorithm stores the visited states in a tree, the root of which corresponds to the initial state. The construction of the tree is summarized in Algorithm 7.

The tree constructed at the k^{th} iteration is denoted by $\mathcal{T}^k$. The function SAMPLING samples a hybrid state $s^k_{goal} = (q^k_{goal}, x^k_{goal})$ to indicate the direction towards which the tree is expected to evolve. Then, a starting state $s^k_{near} = (q^k_{near}, x^k_{near})$ is determined as a neighbor of s^k_{goal}. The definition of the distance between two hybrid states will be given later. Expanding the tree from s^k_{near} towards s^k_{goal} is done as follows:

Algorithm 7 Test generation algorithm **hRRT**

$k = 1$
$\mathcal{T}^k.init(s_{init})$ {s_{init}: *initial state*}
repeat
  $s_{goal} = \text{SAMPLING}(\mathcal{S})$ {$\mathcal{S}$: *hybrid state space*}
  $s^k_{near} = \text{NEIGHBOR}(\mathcal{T}^k, s^k_{goal})$
  $(s^k_{new}, u^k_{q_{near}}) = \text{CONTINUOUSSUCC}(s^k_{near}, h)$ {h: *time step*}
  $\text{DISCRETESUCC}(\mathcal{T}^k, s^k_{new})$
  $k++$
until $k \geq k_{max}$

- The function CONTINUOUSSUCC tries to find the input $u^k_{q_{near}}$ such that, after one time step h, the current continuous dynamics at q^k_{near} takes the system from s^k_{near} towards s_{goal}, and this results in a new continuous state x^k_{new}. A new edge from s_{near} to $s^k_{new} = (q^k_{near}, x^k_{new})$, labeled with the associated input $u^k_{q_{near}}$, is then added to the tree. To find s^k_{new}, when the set U is not finite it can be sampled, or one can solve a local optimal control problem.

- Then, from s^k_{new}, the function DISCRETESUCC computes its successors by all possible discrete transitions and add them in the tree. A discrete successor by a transition is computed by testing whether s^k_{new} satisfies its guard and if so applying the associated reset function to s^k_{new}.

The algorithm terminates after some maximal number of iterations. Another possible termination criterion is that a satisfactory coverage value is reached. In the classic **RRT** algorithms, which work in a continuous setting, only x_{goal} needs to be sampled, and a commonly used sampling distribution of x_{goal} is uniform over $\mathcal{X}$. In addition, the point x_{near} is defined as a nearest neighbor of x_{goal} in some usual distance, such as the Euclidian distance. In our **hRRT** algorithm, the goal state sampling is not uniform and the function SAMPLING plays the role of guiding the exploration via a biased sampling of x_{goal}.

The tree constructed by the **hRRT** algorithm can be used to extract a test suite. In addition, when applying such test cases to the system under test, the tree can be used to compare the observations from the real systems and the expected observations in the tree. This allows a decision whether the system satisfies the conformance relation.

4.7 Coverage-guided test generation

In this section we propose a tool for guiding the test generation algorithm. This tool is based on the coverage measure defined using the star discrepancy. The goal of the guiding tool is to use the sampling process to bias the evolution of the tree towards the interesting region of the state space, in order to rapidly achieve a good coverage quality. In each iteration, we use the information of the current coverage to improve it. Indeed, the coverage estimation provides not only an approximate value of the current coverage, but also the information about which regions need to be explored more.

Sampling a goal state $s_{goal} = (q_{goal}, x_{goal})$ in the hybrid state space $\mathcal{S}$ consists of two steps:

1. Sample a goal location q_{goal} from the set Q of all the locations, according to some probability distribution.
2. Sample a continuous goal state x_{goal} inside the staying set $\mathcal{I}_{q_{goal}}$ of the location q_{goal}.

Location sampling

Recall that we want to achieve a good testing coverage quality, which is equivalent to a small value of the star discrepancy of the points visited at each location. More concretely, in each iteration, we want to bias the goal state sampling distribution according to the current coverage of the visited states. To do so, we first sample a location and then a continuous state. Let $\mathcal{P} = \{(q, P_q) \mid q \in Q \wedge P_q \subset \mathcal{I}_q\}$ be the current set of visited states. The location sampling distribution depends on the current continuous state coverage of each location:

$$Pr[q_{goal} = q] = \frac{D^*(P_q, \mathcal{I}_q)}{\sum_{q' \in Q} D^*(P_{q'}, \mathcal{I}_{q'})}$$

where the notation Pr is used for probabilities. As we have shown earlier, the star discrepancy is approximated by a lower bound and an upper bound. We thus compute the above probability $Pr[q_{goal} = q]$ using these bounds and then taking the mean of the results.

Continuous state sampling

We now show how to sample x_{goal}, assuming that we have already sampled a location $q_{goal} = q$. In the remainder of the paper, to give geometric intuitions, we often call a continuous state a point. In addition, since all the staying sets are

assumed to be boxes, we denote the staying set $\mathcal{I}_q$ by the box $\mathcal{B}$ and denote the current set of visited points at the location q simply by P instead of P_q. Let k be the number of points in P. Let Π be a finite box partition of $\mathcal{B}$ that is used to estimate the star discrepancy of P. The sampling process consists of two steps. In the first step, we sample an elementary box $\boldsymbol{b}_{goal}$ from the set Π; in the second step we sample a point x_{goal} in $\boldsymbol{b}_{goal}$ uniformly.

The elementary box sampling distribution in the first step is biased in order to optimize the coverage. Guiding is thus done via the goal box sampling process.

Let Π be the box partition used in the coverage estimation, and we denote by P the current set of visited states. The objective is to define a probability distribution over the set of elementary boxes of Π. This probability distribution is defined at each iteration of the test generation algorithm. Essentially, we favor the selection of a box if adding a new state in this box allows to improve the coverage of the visited states. This is captured by a potential influence function, which assigns to each elementary box $\square$ in the partition a real number that reflects the change in the coverage if a new state is added in $\square$. The current coverage is given in form of a lower and an upper bound. In order to improve the coverage, we aim at reducing both of the bounds. More details on the method can be found in [42].

Let us summarize the developments so far. We have shown how to sample a goal hybrid state. This sampling method is not uniform but biased in order to achieve a good coverage of the visited states. From now on, the algorithm **hRRT** in which the function SAMPLING uses this coverage-guided method is called the **gRRT** algorithm, which means 'guided **hRRT**'.

We can prove that the **gRRT** algorithm preserves the *probabilistic completeness* of **RRT** [42]. Roughly speaking, the probabilistic completeness property [74] states that if the trace we seach for is feasible, then the probability that the algorithm finds it approaches 1 as the number k of iterations approaches infinity.

To demonstrate the performance of **gRRT**, we use two illustrative examples. For brevity, we call the classical **RRT** algorithm using uniform sampling and the Euclidian metric **hRRT**. The reason we choose these examples is that they differ in the reachability property. In the first example, the system is 'controllable' in the sense that the whole state space is reachable from the initial states (by using appropriate inputs), but in the second example the reachable set is only a small part of the state space. These examples will also be used to validate the efficiency of the new guiding method that we propose.

Example 1. *This is a two-dimensional continuous system where the state space* $\mathcal{X}$ *is a box* $\mathcal{B} = [-3, -3] \times [3, 3]$. *The continuous dynamics is* $f(x, t) = u(t)$ *where the input set is* $U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 0.2\}$.

We use 100 input values resulting from a discretization of the set U. The

initial state is $(-2.9, -2.9)$. The time step is 0.002. Figure 4.6 shows the result obtained using **gRRT** and the evolution after each iteration of the coverage of the states generated by **gRRT** (solid curve) and by **hRRT** (dashed curve). The figure indicates that **gRRT** achieved a better coverage quality especially in convergence rate.

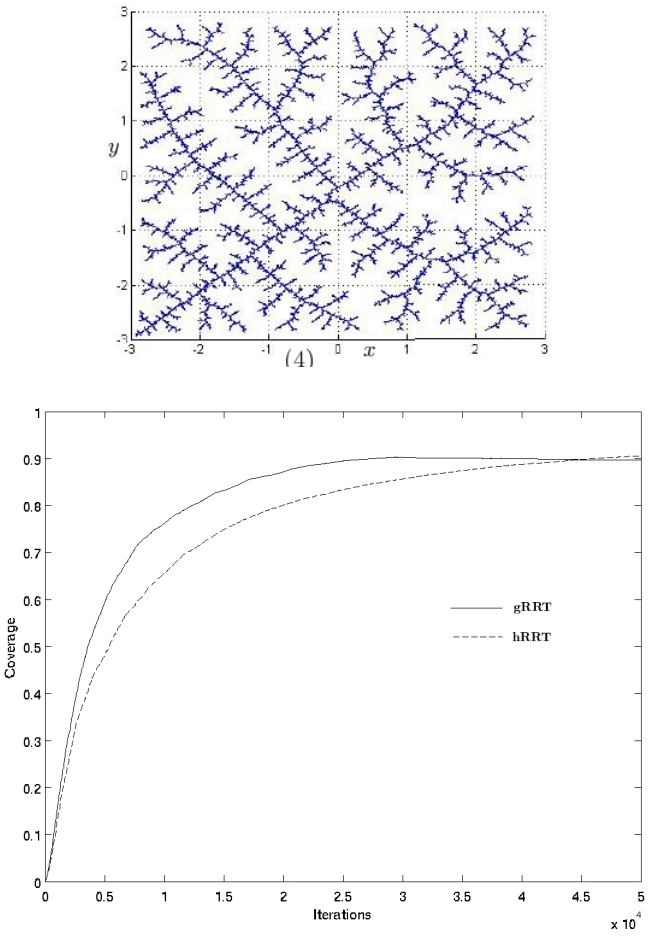


Figure 4.6: Left: The **gRRT** exploration result. Right: Test coverage evolution using **hRRT** and **gRRT**.

Example 2. *This example is a linear system with a stable focus at the origin. Its dynamics is*

$$\begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = \begin{pmatrix} -1 & -1.9 \\ 1.9 & -1 \end{pmatrix} \times \begin{pmatrix} x \\ y \end{pmatrix} + \begin{pmatrix} u_1 \\ u_2 \end{pmatrix}$$

We let the dynamics be slightly perturbed by an additive input u. The state space is the box $\mathcal{B} = [-3, -3] \times [3, 3]$. The input set $U = \{u \in \mathbb{R}^2 \mid ||u|| \leq 0.2\}$.

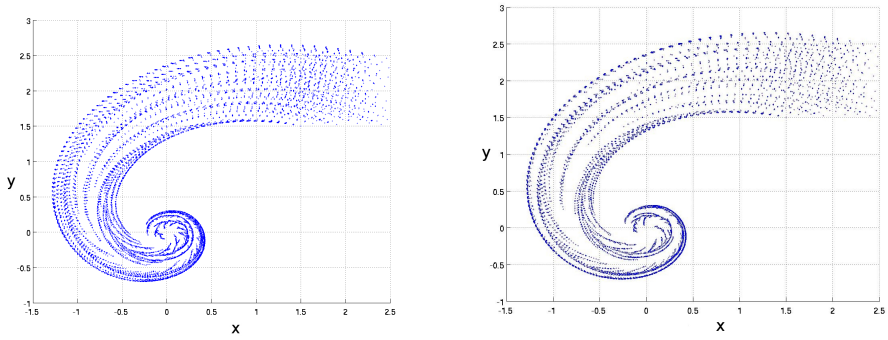


Figure 4.7: Results obtained using the guided sampling method (left) and using the uniform sampling method (right).

Figure 4.7 shows the results obtained after 50000 *iterations. We can see that again the guided sampling method achieved a better coverage result.*

4.8 Controllability issue

From different experiments with Example 2, we observed that the coverage performance of **gRRT** is not satisfying when the reachable space is only a small part of the whole state space. To illustrate this, we increase the state space from $\mathcal{B} = [-3, -3] \times [3, 3]$ to $\mathcal{B}' = [-5, -5] \times [5, 5]$. For the larger state space, the coverage quality is poorer (see Figure 4.8). This can be explained as follows. There are boxes, such as those near the bottom right vertex of the bounding box, which have a high potential of reducing the bounds of the star discrepancy. Thus, the sampler frequently selects these boxes. However, these boxes are not reachable from the initial states, and all attempts to reach them do not expand the tree beyond the boundary of the reachable set. This results in a large number of points concentrated near this part of the boundary, while other parts of the reachable set are not well explored.

It is important to emphasize that this problem is not specific to **gRRT**. The **RRT** algorithm using the uniform sampling method and, more generally, any algorithm that does not take into account the differential contraints of the system, may suffer from this phenomenon. This phenomenon can however be captured by the evolution of the disparity between the set of goal states and the set of visited states. This notion will be formally defined in the next section. Roughly speaking, it describes how different their distributions are. When the disparity does not decrease after a certain number of iterations, this often indicates that the system cannot approach the goal states, and it is better not to favor an expansion

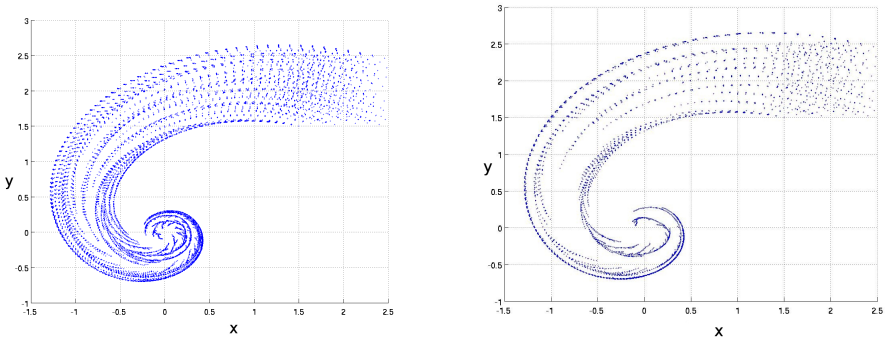


Figure 4.8: Results for the state spaces $\mathcal{B}$ (left) and $\mathcal{B}'$ (right).

towards the exterior but a *refinement*, that is an exploration in the interior of the already visited regions.

Figure 4.9 shows the evolution of the disparity between the set P^k of visited states at the k^{th} iteration and the set G^k of goal states for the two examples. We observe that for the system of Example 1 which can reach any state in the state space (by choosing appropriate admissible inputs), the visited states follow the goal states, and thus the disparity gets stabilized over time. However, in Example 2, where the system cannot reach everywhere, the disparity does not decrease for a long period of time, during which most of the goal states indicate a direction towards which the tree cannot be expanded further.

Figure 4.9 shows the Voronoi diagram[3] of a set of visited states. In this example, the boundary of the reachable set can be seen as an 'obstacle' that prevents the system from crossing it. Note that the Voronoi cells of the states on the boundary are large (because they are near the large unvisited part of the state space). Hence, if the goal states are uniformly sampled over the whole state space, these large Voronoi cells have higher probabilities of containing the goal states, and thus the exploration is 'stuck' near the boundary, while the interior of the reachable set is not well explored.

To tackle this problem, we introduce the notion of disparity to describe the 'difference' in the distributions of two sets of points. The controllability problem can be detected by a large value of the disparity between the goal states and the visited states. We can thus combine **gRRT** with a disparity based sampling method, in order to better adapt to the dynamics of the system. This is the topic of the next section.

[3]The Voronoi diagram of a set V of points in $\mathbb{R}^n$ is the partition of $\mathbb{R}^n$ into k polyhedral regions. Each region corresponds to a point $v \in V$, called the Voronoi cell of v, is defined as the set of points in $\mathbb{R}^n$ which are closer to v than to any other points in V.

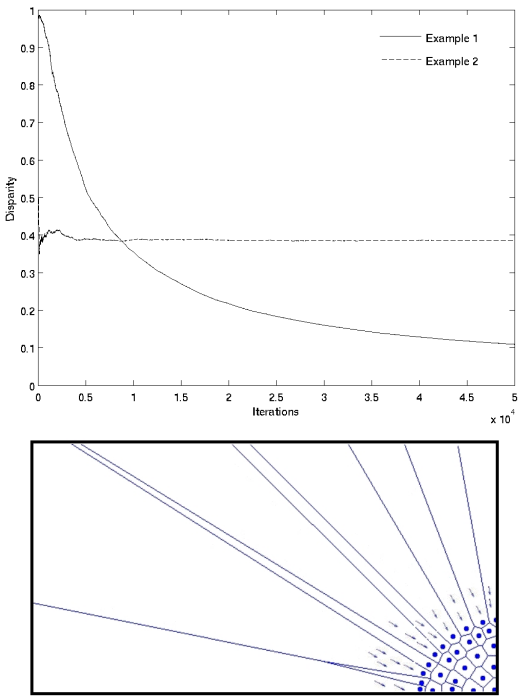


Figure 4.9: Left: The evolution of the disparity between the set P^k of visited states and the set G^k of goal states. Right: Illustration of the controllability issue.

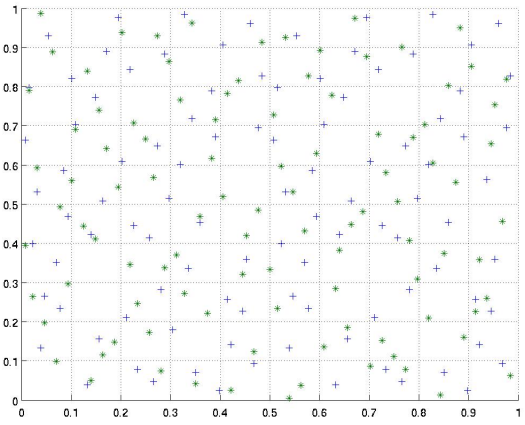


Figure 4.10: The disparity between the Faure sequence (drawn using the + signs) and the Halton sequence (drawin using the $*$ signs) is 0.06.

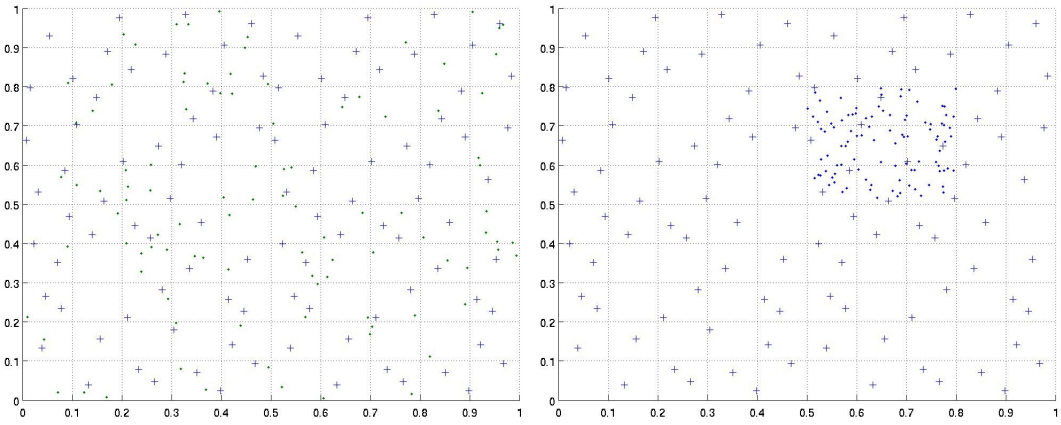


Figure 4.11: Left: The disparity between the Faure sequence (+ signs) and a C pseudo-random sequence (∗ signs) is 0.12. Right: The disparity between the Faure sequence (+ signs) and another C pseudo-random sequence (∗ signs) is 0.54.

4.9 Disparity

The notion of disparity between two point sets that we develop here is inspired by the star discrepancy. Indeed, by definition, the star discrepancy of a set P w.r.t. the box $\mathcal{B}$ can be seen as a comparison between P and an 'ideal' infinite set of points distributed all over $\mathcal{B}$.

Let P and Q be two sets of points inside $\mathcal{B}$. Let J be a sub-box of $\mathcal{B}$ which has the same bottom-left vertex as $\mathcal{B}$ and the top-right vertex of which is a point inside $\mathcal{B}$. Let Γ be the set of all such sub-boxes. We define the local disparity between P and Q with respect to the sub-box J as: $\gamma(P,Q,J) = |\frac{A(P,J)}{||P||} - \frac{A(Q,J)}{||Q||}|$ where $A(P,J)$ is the number of points of P inside J and $||P||$ is the total number of points of P.

Definition 8 (Disparity)**.** *The disparity between P and Q with respect to the bounding box $\mathcal{B}$ is defined as: $\gamma^*(P,Q,\mathcal{B}) = sup_{J\in\Gamma}\gamma(P,Q,J)$.*

The disparity satisfies $0 < \gamma^*(P,Q,\mathcal{B}) \leq 1$. This property is a direct consequence of the above definition. A small value $\gamma^*(P,Q,\mathcal{B})$ means that the distributions of the sets P and Q over the box $\mathcal{B}$ are 'similar'.

To illustrate our notion of disparity, we consider two well-known sequences of points: the Faure sequence [50] and the Halton sequence [106], which are shown in Figure 4.10. Their disparity is 0.06, indicating that they have similar distributions. We then compare the Faure sequence with a sequence generated by the C library. Figure 4.11 displays these two sequences, each of which has 100

points. The star discrepancy coverage of the Faure sequence is much better than that of the C sequence, and in fact their disparity between them (which is 0.12) is twice larger than that between the Faure and Halton sequences. The last example is used to compare the Faure sequence and a set of 100 points concentratred in some small rectangle inside the bounding box. Their disparity is 0.54.

Disparity estimation

The exact computation of the disparity is as hard as the exact computation of the star discrepancy, which is due to the infinite number of the sub-boxes. We propose a method for estimating a lower and an upper bound for this new measure. Let Π be a box partition of $\mathcal{B}$. Let P, Q be two sets of points inside $\mathcal{B}$. For each elementary box $\square \in \Pi$ we denote $\mu_m(\square) = \max\{\mu_c(\square), \mu_o(\square)\}$ where $\mu_c(\square) = \frac{A(P, \square^+)}{||P||} - \frac{A(Q, \square^-)}{||Q||}$, $\mu_o(\square) = \frac{A(Q, \square^+)}{||Q||} - \frac{A(P, \square^-)}{||P||}$. We also denote

$$c(\square) = \max\{|\frac{A(P, \square^-)}{||P||} - \frac{A(Q, \square^-)}{||Q||}|, |\frac{A(P, \square^+)}{||P||} - \frac{A(Q, \square^+)}{||Q||}|\}.$$

Theorem 10. [Upper and lower bounds] *An upper bound $B_d(P, Q, \Pi)$ and a lower bound $C_d(P, Q, \Pi)$ of the disparity between P and Q are: $B_d(P, Q, \Pi) = \max_{\square \in \Pi}\{\mu_m(\square)\}$ and $C_d(P, Q, \Pi) = \max_{\square \in \Pi}\{c(\square)\}$.*

The proof of the theorem can be found in [40].

Estimation error

We now give a bound on the estimation error. For each elementary box $\square = [\alpha_1, \beta_1] \times \ldots \times [\alpha_n, \beta_n] \in \Pi$, we define the $\mathcal{W}$-zone, denoted by $\mathcal{W}(\square)$, as follows: $\mathcal{W}(\square) = \square^+ \setminus \square^-$. We recall that $\square^+ = [l_1, \beta_1] \times \ldots \times [l_n, \beta_n]$ and $\square^- = [l_1, \alpha_1] \times \ldots \times [l_n, \alpha_n]$. We can prove the following bound on the error of the estimation, defined as the difference between the upper and lower bounds [40].

Theorem 11 (Error bounds)**.** *Let $B_d(P, Q, \Pi)$ and $C_d(P, Q, \Pi)$ be the upper and lower bounds of the disparity between P and Q. Then,*

$$B_d(P, Q, \Pi) - C_d(P, Q, \Pi) \leq \max_{\square \in \Pi} \max\{\frac{A(P, \mathcal{W}(\square))}{||P||}, \frac{A(Q, \mathcal{W}(\square))}{||Q||}\}.$$

The above error bounds can be used to dynamically refine the partition.

4.10 Disparity-guided sampling

The essential idea of our disparity based sampling method is to detect when the dynamics of the system does not allow the tree to expand towards the goal states and then to avoid such situations by favoring a refinement, that is an exploration near the already visited states.

A simple way to bias the sampling towards the set P^k of already visited states is to reduce the sampling space. Indeed, we can make a bounding box of the set P^k and give more probability of sampling inside this box than outside it. Alternatively, we can guide the sampling using the disparity information as follows. The objective now is to reduce the disparity between the set G^k of goal states and the set P^k of visited states. Like the guiding method using the star discrepancy, we define for each elementary box $\square$ of the partition a function $\eta(\square)$ reflecting the potential for reduction of the lower and upper bounds of the disparity between P^k and G^k. This means that we favor the selection of the boxes that make the distribution of goal states G^k approach that of the visited states P^k. Choosing such boxes can improve the quality of refinement. The formulation of the potential influence function for the disparity-based sampling method is similar to that for the coverage guided sampling.

A combination of the coverage guided and the disparity guided sampling methods is done as follows. We fix a time window T_s and a threshold ϵ. When using the coverage guide method, if the algorithm detects that the disparity between the G^k and P^k does not decrease by ϵ after T_s time, it switches to the disparity guided method until the disparity is reduced more significantly and switches back to the coverage guide method. Note that a non-decreasing evolution of the disparity is an indication of the inability of the system to approach the goal states. In an interactive exploration mode, it is possible to let the user to manually decide when to switch. As mentioned earlier, we call the resulting algorithm **agRRT** (the letter 'a' in this acronym stands for "adaptive").

Examples. We use the examples in the previous section to demonstrate the coverage improvement of **agRRT**. Figure 4.12 shows that the final result for Example 1 obtained using **agRRT** has a better coverage than that obtained using **gRRT**. The solid curve represents the coverage of the set P^k of visited states and the dashed one the coverage of the set G^k of goal states. The dash-dot curve represents the disparity between G^k and P^k.

The result obtained using **agRRT** for Example 2 is shown in Figure 4.13, which also indicates an improvement in coverage quality. The figure on the right shows the set of generated goal states. The states are drawn in dark color. In this example, we can observe the adaptivity of the combination of **gRRT** and **agRRT**. Indeed, in the beginning, the **gRRT** algorithm was used to rapidly

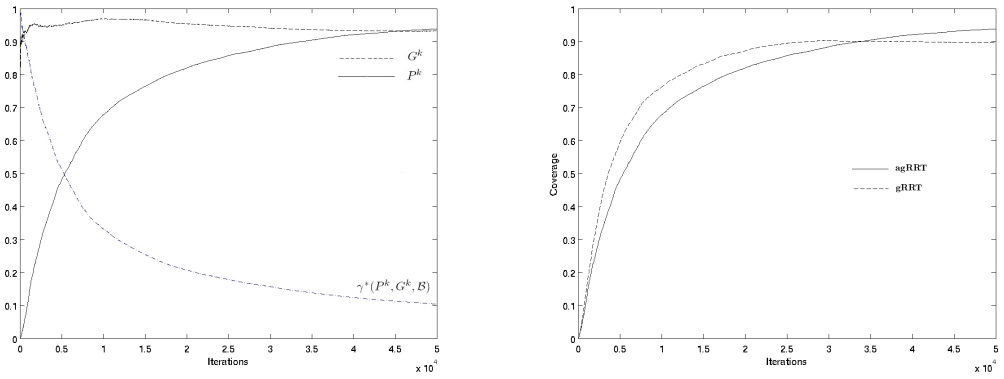


Figure 4.12: Left: Test coverage of the result obtained using **agRRT** for Example 1. Right: Comparing **gRRT** and **agRRT**.

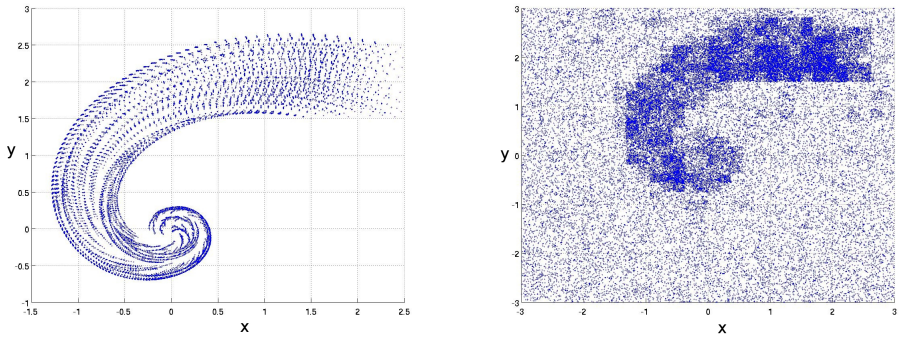


Figure 4.13: Result after $k = 50000$ iterations, obtained using **agRRT** (left: the set of visited states P^k, right: the set of goal states G^k).

expand the tree. After some time, the goal states sampled from the outside of the exact reachable space do not improve the coverage, since they only create more states near the boundary of the reachable set. In this case, the disparity between P^k and G^k does not decrease, and the **agRRT** is thus used to enable an exploration in the interior of the reachable set. The interior the reachable set thus has a higher density of sampled goal states than the outside, as one can see in the figure.

4.11 Termination criterion using disparity

We first explain why it is not suitable to use the star discrepancy coverage to decide the termination of the test generation algorithm. Indeed, the star discrep-

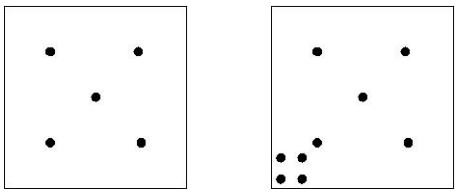


Figure 4.14: The point set P (left) and the point set $P \cup Q$ (right).

ancy allows to compare the spatial coverage quality between the point sets of the same cardinality; it however may give misleading comparison between the sets of different cardinalities. To illustrate this, we consider a set of two-dimensional points: $P = \{(0.25, 0.25), (0.5, 0.5), (0.75, 0.25), (0.25, 0.75), (0.75, 0.75)\}$ inside the bounding box $\mathcal{B} = [0, 1] \times [0, 1]$, shown in Figure 4.14-(left). The star discrepancy estimation gives: $D^*(P, \mathcal{B}) \simeq 0.347$. An arising question is how the star discrepancy changes when we add more points in P. We now consider another non-empty set Q of points inside $\mathcal{B}$ that does not contain any points of P. Different experiments show that the star discrepancy of the union $P \cup Q$ might be larger or smaller than that of P. For example, for the following set $Q = \{(0.06, 0.06), (0.12, 0.12), (0.06, 0.12), (0.12, 0.06)$ the value of $D^*(P \cup Q, \mathcal{B}) \simeq 0.493$ (see Figure 4.14-right). In other words, adding this set Q in P increases the star discrepancy, which can be easily understood because the set Q is not well equidistributed over the box $\mathcal{B}$. However, from a verification point of view, the union $P \cup Q$ provides more information about the correctness of the system than the set P. On the other hand, geometrically speaking, the set $P \cup Q$ "covers more space" than the set P.

Therefore, we do not use the star discrepancy coverage measure to decide when the test generation algorithm can terminate. Instead, to detect when the coverage reaches a "saturation", we use the disparity between two consecutive sets of visited states. If its value remains below some predefined threshold Δ_γ after a predefined number K of iterations, we can stop the algorithm because the distribution of the sets of visited states are not much changed and the coverage is not significantly improved. This criterion can be used together with a maximal number of iterations.

4.12 Actuator and sensor imprecision

Due to the limitations of practical actuators and sensors, the tester cannot realize exactly an input value specified as a real-valued vector as well as measure exactly the state of the system. We first explain how actuator imprecision influences the testing process.

Actuator imprecision. We consider the following continuous dynamics of a location: $\dot{x}(t) = f(x(t), u(t))$. Given an initial state x and an input value u, let $\xi_{x,u}$ denote the unique trajectory during the time interval $[0, h]$.

Due to actuator imprecision, when the tester emits an input u to the system under test, indeed some input $v = u + \delta_u$ with $|\delta_u| \leq \epsilon_u$ is applied. We call ϵ_u the actuator imprecision bound. After one step, this uncertainty causes the new state $y = \xi_{x,v}(h)$ to deviate from the exact state $y' = \xi_{x,u}(h)$. In the next step, this deviation can be considered as an uncertainty in the initial condition, namely $(y' - y)$, which causes further deviation.

Therefore, only when the observation measured by the tester lies outside some admissible deviation neighborhood, the conformance property is considered violated. We need thus to compute the admissible deviation neighborhoods.

Let $M_x = \dfrac{\partial \xi_{x,u}}{\partial x}$ and $M_u = \dfrac{\partial \xi_{x,u}}{\partial u}$ denote the partial derivatives of ξ_x with respect to the initial condition x and to the input. They are called the *sensitivity matrices*. Given a neighboring initial state y and a neighboring input value v, we can get an estimation of $\xi_{y,v}$ by dropping higher order terms in the Taylor expansion of $\xi_{x,u}(t)$ seen as a function of x and of u:

$$\hat{\xi}_{y,v}(h) = \xi_{x,u}(h) + M_x(h)(y - x) + M_u(h)(v - u) \tag{4.6}$$

The deviation is thus $\delta = |M_x(h)(y-x)+M_u(h)\epsilon_u|$. The deviation neighborhood is defined as a ball centered at $\xi_{x,u}(h)$ with radius δ. To compute the sensitivity matrices, we solve the following differential equations, which result from taking the derivative of the solution:

$$\dot{x} = f(x, u), \tag{4.7}$$

$$\dot{M}_x = \frac{\partial f(x,u)}{\partial x} M_x, \tag{4.8}$$

$$\dot{M}_u = \frac{\partial f(x,u)}{\partial x} M_u + \frac{\partial f(x,u)}{\partial u} \tag{4.9}$$

with the initial condition $\xi_x(0) = x$ and $M_x(0) = \mathbf{I}_n$ (the identity matrix) and $M_u(0) = \mathbf{0}_n$ (the zero matrix). These are $n+m+1$ ordinary differential equations of order n. Note that the cost can be made less than that of the resolution of $n+m+1$ different systems since the Jacobian $\dfrac{\partial f(x,u)}{\partial x}$ can be extensively reused in the computation.

Therefore, in the RRT tree construction, when we compute a new state from a given state x and input u, we additionally compute the corresponding sensitivity matrix $M_x(h)$ and $M_u(h)$ and associate these matrices to the new node created to store the new state. We also propagate the neighborhoods in order to detect possible executions of uncontrollable discrete transitions. If the current state does

not enable a uncontrollable transition but the associated neighborhood intersects with its guard, we compute the image of the intersection by the reset map and take its centroid to define a new visited state. The diameter of the image is the radius of the associated neighborhood.

Verdict. During the test, let $\tilde{s}^i = (\tilde{q}^i, \tilde{x}^i)$ be the observation at the i^{th} step and $s^i = (q^i, x^i)$ be the corresponding expected state in the tree. Let M_x^i and M_u^i be the sensitivity matrices associated with the node (q^i, x^i). The system under test is assumed to satisfy the conformance property at the initial state. The verdict is then decided as follows. We suppose that the conformance property is not violated until the i^{th} step. There are two cases:

1. (C1) $s^{(i+1)}$ is reached by the continuous dynamics. Then, if $|\tilde{x}^{(i+1)} - x^{(i+1)}| > M_x^i|\tilde{x}^i - x^i| + M_u^i\epsilon_u$, we conclude that the conformance property is violated and stop the test. Otherwise, we continue the test.

2. (C2) $s^{(i+1)}$ is reached by a discrete transition. Then, if $\tilde{q}^i = q^i$ and $|\tilde{x}^{(i+1)} - x^{(i+1)}| > \delta^{(i+1)}$ where $\delta^{(i+1)}$ is the deviation radius, we conclude that the conformance property is violated and stop the test. Otherwise, we continue the test.

Sensor imprecision. Let the sensor imprecision be modeled by an upper bound ϵ_x on the distance between the actual continuous state and the observation measured by the tester. The distance between the observation and the expected states is now tested againts a bound larger by ϵ_x, that is $|\tilde{x}^{(i+1)} - x^{(i+1)}| > M_x^i(\tilde{x}^i - x^i) + M_u^i\epsilon_u + \epsilon_x$ for the above (C1) and $|\tilde{x}^{(i+1)} - x^{(i+1)}| > \delta^{(i+1)} + \epsilon_x$ for (C2).

4.13 Tool HTG

We have implemented the above algorithms in a test generation tool, called **HTG**. Its implementation uses a data structure, similar to a k-d tree, which facilitates the required operations, such as updating the tree, finding neighbors, updating the star discrepancy and disparity. Indeed, using this data structure, we can efficiently encode a box partition for storing and accessing the visited states as well as for the star discrepancy and disparity estimations. In the following, we briefly describe some important functions. A more detailed description of the implementation can be found in [40]

Numerical simulation

In most classic versions of hybrid automata, continuous dynamics are defined using ordinary differential equations (ODEs). To analyze analog and mixed-signal circuits, the behavior of which are described using differential algebraic equations (DAEs), we adapt the model to capture this particularity and use the well-known RADAU algorithm for solving DAEs [45]. On the other hand, with view to applications in analog and mixed-signal circuits, an efficient and reliable simulation method is key. The state-of-the-art SPICE simulator is prone to convergence problems when dealing with circuit components with stiff characteristics. We also integrated in our test generation tool a connection to the platform **SICONOS**, developed at INRIA Rhônes-Alpes, that contains a number of simulation algorithms based on *the non-smooth approach* [4], which have proved to be efficient for systems with stiff dynamics.

SPICE netlists as input systems.

An important new feature of the tool is its ability to deal with circuits describes using SPICE, in addition to textual descriptions of a hybrid automata. This facilitates the application of the tool to practical circuits. Note that our test generation method works on the differential equations of a hybrid automaton, which is an appropriate mathematical model to describe circuit dynamics. However, the circuit equations need to be generated from a SPICE netlist. Common SPICE parsers do not provide directly the a succinct form of circuit equations but generate complex equations for the numerical resolution of each simulation step. Furthermore, using SPICE descriptions we cannot specify uncertain inputs. The solution we propose to address this can be described as follows:

- The inputs (controllable by tester) can be source currents or voltages in SPICE. Their uncertainty is described in an auxiliary file.
- We use the tool ACEF [3] to generate the numerical simulation equations in each step. The values of the conptrollable inputs, computed by the test generation algorithm, are communicated to ACEF [3] before the generation in each step. The generation can be optimized by only updating some terms involving the modified input in the equations of the previous steps.

The test cases generated by the tool can be visualized by different standard viewers such as **matlab** and **gnu** plots.

The tool can also be used as a systematic simulator to verify properties of a model. For this problem, the tester can manipulate not only control inputs but also disturbance inputs. Systematic simulation can be seen as a good compromise between exhaustive verification and adhoc simulation.

4.14 Applications

The tool has been tested on various examples, which proved its scalability to high dimensional systems (up to a few hundreds continuous variables) [42]. In addition, we have also successfully applied the tool to a number of case studies in control systems and in analog and mixed signal circuits. In the following, to give an overview of the applicability of the tool, we briefly report some of these applications.

4.14.1 Aircraft collision avoidance system

To illustrate the application of our algorithm to hybrid systems, we use the aircraft collision avoidance problem [84], which is a well-known benchmark in the hybrid systems literature. In this paper, the authors treated the problem of collision avoidance of two aircrafts. To show the scalability of our approach we consider the same model with N aircrafts.

As shown in Figure 4.15, all the aircrafts are at a fixed altitude. Each aircraft i has three states (x_i, y_i, θ_i) where x_i and y_i describe the position and θ_i is the relative heading of the aircraft. Each aircraft begins in straight flight at a fixed relative heading (mode 1).

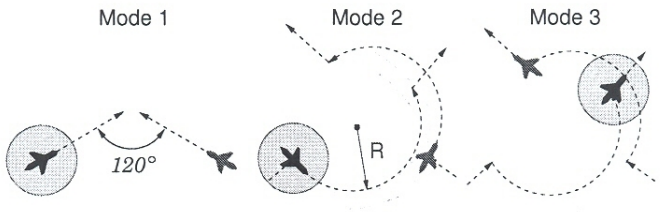


Figure 4.15: Aircraft behavior in the three modes [84].

As soon as two aircrafts are within the distance R between each other, they enter mode 2. In this mode each aircraft makes an instantaneous heading change of 90 degrees, and begins a circular flight for π time units. After that, they switch to mode 3 and make another instantaneous heading change of 90 degrees and resume their original headings from mode 1.

The dynamics of the system are shown in Figure 4.16. The guard transition between mode 1 and mode 2 is given by $Dij < R$, which means that the aircraft

i is at R distance from the aircraft j. The dynamics of each aircraft is as follows:

$$\begin{aligned} \dot{x}_i &= vcos(\theta_i) + dx_i, \\ \dot{y}_i &= vsin(\theta_i) + dy_i \\ \dot{\theta}_i &= \omega \end{aligned}$$

The continuous inputs are dx_i and dx_i describing the external disturbances on the aircrafts (such as wind):

$$\begin{aligned} dx_i &= d_1 sin(\theta_i) + d_2 cos(\theta_i), \\ dy_i &= -d_1 cos(\theta_i) + d_2 sin(\theta_i), \end{aligned}$$

and $-\delta \leq d_1, d_2 \leq \delta$.

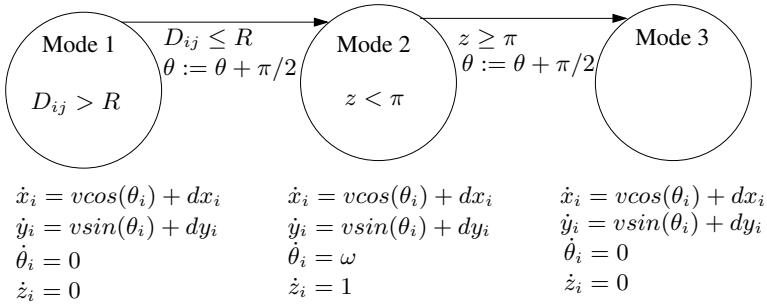


Figure 4.16: System dynamics for the three modes.

Results. For N aircrafts, the system has $3N + 1$ continuous variables (one for modeling a clock). For the case of $N = 2$ aircrafts, when the collision distance is 5, no collision was detected after visiting 10000 visited states, and the computation time was 0.9 min. The result for $N = 8$ aircrafts with the disturbance bound $\delta = 0.06$ is shown in Figure 4.17, where we show the projected positions of the eight aircrafts on a 2-dimensional space. For this example, the computation time for 50000 visited states was 10 min and a collision was found. For a similar example with $N = 10$ aircrafts, the computation time was 14 min and a collision was also found.

4.14.2 Robotic vehicle benchmark

This example is adapted from the robotic navigation system benchmark [91]. We consider a car with the following continuous dynamics with 5 variables: $\dot{x} = vcos(\theta)$, $\dot{y} = vsin(\theta)$, $\dot{\theta} = vtan(\phi)/L$, $\dot{v} = u_0$, $\dot{\phi} = u_1$ where x, y, θ describe the position and heading of the car, v is its speed and ϕ is its steering angle.

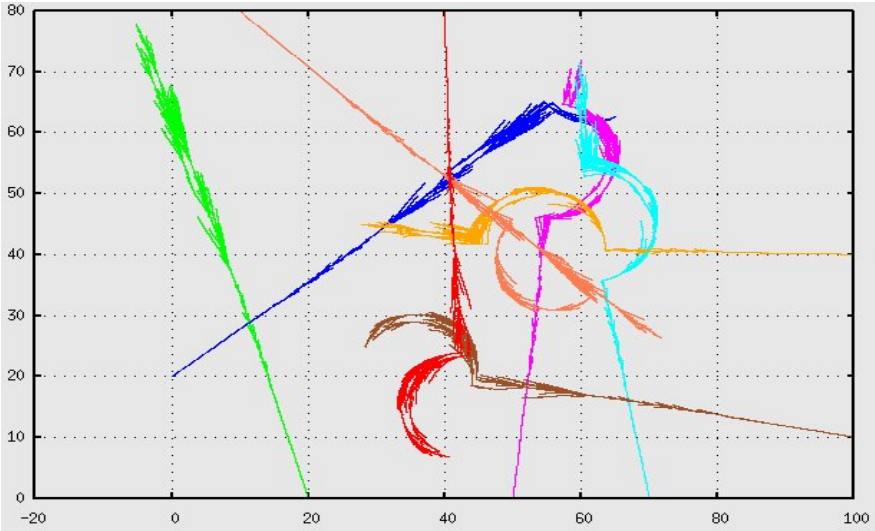


Figure 4.17: Eight-aircraft collision avoidance (50000 visited states, computation time: 10 min.

The car can be in one of three car modes (smooth car, smooth unicycle, smooth differential drive). In this work, we consider only the smooth car mode.

The inputs of the system are u_0 and u_1 which are respectively the acceleration and steering control. The system uses a hybrid control law with 3 driver modes. In the first driver mode, called *RandomDriver*, the control inputs are selected uniformly at random between their lower and upper bounds. In the second driver mode, called *StudentDrive*, when the speed is low, u_0 is randomly chosen as in first mode; otherwise, the strategy is to reduce the speed. In the third driver mode, called *HighwayDrive*, the strategy is to reduce the speed when it is high and increase it when it is low. A detailed description of this control law can be found in [91].

Rather than to analyze a realistic navigation system model, we use this example to test the efficiency of our algorithms on a hybrid system with a larger number of locations. To this end, we created from this system two models. The terrain is partitioned into K rectangles using a regular grid $\mathcal{G} = \{0, \ldots, K_x - 1\} \times \{0, \ldots, K_y - 1\}$. Each rectangle is associated a driver mode. The first model is a hybrid automaton with $K_x K_y$ locations and the system can only switch between the locations corresponding to adjacent rectangles.

In the second model, we allow more complicated switching behavior by letting the system jump between some rectangles which are not necessarily adjacent. The rectangle corresponding to the grid point $(i, j) \in \mathcal{G}$ is $R_{ij} = [il_x, jl_y] \times [(i+1)l_x, (j+1)l_y]$ where l_x and l_y are the sizes of the grid in the x and y coordinates. The absolute index of R_{ij} is an integer defined as follows: $\iota(R_{ij}) =$

$iK_y + j$. From the rectangle R_{ij} with even absolute index, we allow a transition to R_{mn} such that $\iota(R_{mn}) = (\iota(R_{ij}) + J) mod (K_x K_y)$ (where $J > 0$ and mod is the modulo division). The guard set at R_{ij} is the right-most band of width ϵ_g, that is $[(i+1)l_x - \epsilon_g, jl_y] \times [(i+1)l_x, (j+1)l_y]$. After switching to R_{mn}, the car position (x, y) is reset to a random point inside the square of size ϵ_r defined as $[ml_x, (n+1)l_y - \epsilon_r] \times [ml_x + \epsilon_r, (n+1)l_y]$.

We compared the results obtained for the two models using the **gRRT** algorithm and the **hRRT** algorithm. In this experimentation the **hRRT** algorithm uses a uniform sampling (both over the discrete and continuous state space). Since we want to focus on the performance of the guiding tool, the two algorithms use the same hybrid distance definition and implementation. The parameters used in this experimentation are: $l_x = l_y = 20$, $l_x = l_y = 20$, the car position $(x, y) \in [-100, 100] \times [-100, 100]$, $\epsilon_g = \epsilon_r = 6$, $J = 6$. The number of locations in the hybrid automata is 100. For the first model without jumps, in terms of coverage efficiency, the algorithms are comparable. For the model with jumps, **gRRT** systematically produced better coverage results. However, **gRRT** is not always better than **hRRT** in terms of the number of covered locations. This is due to our coverage definition using the average of the continuous-state coverages of all the locations.

In terms of time efficiency, we now report the computation time of **gRRT** for the experimentations with various maximal visited states. For the first model, the computation times of **gRRT** are: 4.7s for 10000 states in the tree, 1min 26s for 50000 states, 6min 7s for 100000 states. For the second model, the computation times of **gRRT** are: 4.2s for 10000 states in the tree, 2min 5s for 50000 states, 4min 40s for 100000 states, and 20min 22s for 150000 states.

4.14.3 Analog and mixed-signal circuits

Due to an increasing utilization of embedded systems, systems in which the computer interacts with the physical world, such as cellular phones and other mobile devices, there has been a dramatic rise in interest in analog and mixed-signal circuits. While digital circuit design can be done with performant CAD tools, analog and mixed signal design is still much less automated. The main goal of this project is to develop new techniques for validating these circuits. The work will be based on new analysis techniques for hybrid systems, systems that combine discrete event systems and continuous systems and can naturally describe the behaviors of such circuits.

Using the above results, we implemented a test generation tool and tested it on a number of control applications, which proved its scalability to high dimensional systems [88]. In this implementation, all the sets encountered in the hybrid automaton definition are convex polyhedra. For circuit applications, we

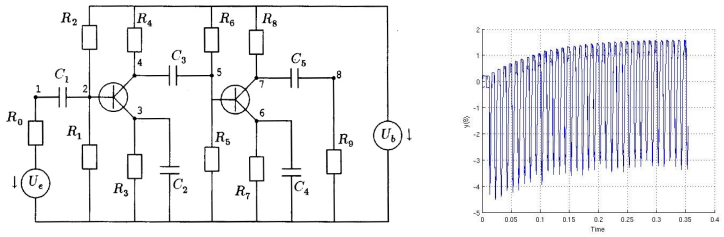


Figure 4.18: Test generation result for the transitor amplifier.

use the well-known RADAU algorithm for solving differential algebraic equations (DAE) [45]. We recall that solving high index[4] and stiff DAEs is computationally expensive, and in order to evaluate the efficiency of the test generation algorithm, we have chosen two practical circuits with DAEs of this type. The three circuits we treated are: a transistor amplifier, a voltage controlled oscillator, and a Delta-Sigma modulator circuit.

Transistor amplifier. The first example is a transistor amplifier model [45], shown in Figure 4.18, where the variable y_i is the voltage at node i; U_e is the input signal and $y_8 = U_8$ is the output voltage. The circuit equations are a system of DAEs of index 1 with 8 continuous variables: $M\dot{y} = f(y, u)$. The function f is given by:

$$\begin{pmatrix} -U_e/R_0 + y_1/R_0 \\ -U_b/R_2 + y_2(1/R_1 + 1/R_2) - (\alpha - 1)g(y_2 - y_3) \\ -g(y_2 - y_3) + y_3/R_3 \\ -U_b/R_4 + y_4/R_4 + \alpha g(y_2 - y_3) \\ -U_b/R_6 + y_5(1/R_5 + 1/R - 6) - (\alpha - 1)g(y_5 - y_6) \\ -g(y_5 - y_6) + y_6/R_7 \\ -U_b/R_8 + y_7/R_8 + \alpha g(y_5 - y_6) \\ y_8/R_9 \end{pmatrix}.$$

The circuit parameters are: $U_b = 6$; $U_F = 0.026$; $R_0 = 1000$; $R_k = 9000$, $k = 1, \ldots, 9$; $C_k = k10^{-6}$; $\alpha = 0.99$; $\beta = 10^{-6}$. The initial state $y_{init} = (0, U_b/(R_2/R_1+1), U_b/(R_2/R_1+1), U_b, U_b/(R_6/R_5+1), U_b/(R_6/R_5+1), U_b, 0)$. To study the influence of circuit parameter uncertainty, we consider is a perturbation in the relation between the current through the source of the two transistors and the voltages at the gate and source $I_S = g(U_G - U_S) = \beta(e^{\frac{U_G - U_S}{U_F}} - 1) + \epsilon$, with $\epsilon \in [\epsilon_{min}, \epsilon_{max}] = [-5e-5, 5e-5]$. The input signal $U_e(t) = 0.1sin(200\pi t)$. The acceptable interval of U_8 in the non-perturbed circuit is $[-3.01, 1.42]$. Once the initial transient period has settled down, the test case indicates the presence of

[4]The differential index of a DAE is a measure of the singularity of the DAE. It characterizes the difficulty in numerically solving the equation.

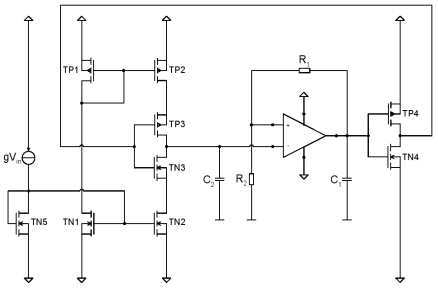


Figure 4.19: Voltage controlled oscillator (VCO) circuit.

traces with overshoots after 18222 iterations (corresponding to 1.1mn of computation time). The total computation time for generating 50000 states was 3mn. Figure 4.18 shows the generated states projected on U_8 over the first 0.03 seconds.

Voltage controlled oscillator. The second circuit we examined is a voltage controlled oscillator (VCO) circuit [63], described by a system of DAEs with 55 continuous variables. In this circuit, the oscillating frequency of the variables v_{C_1} and v_{C_2} is a linear function of the input voltage u_{in}. We study the influence of a time-variant perturbation in C_2, modeled as an input signal, on this frequency. In this example we show that, in addition to conformance relation, using this framework, we can test a property of the input/output relation. The oscillating period $T \pm \delta$ of v_{C_2} can be expressed using a simple automaton with one clock y in Figure 4.20. The question is to know if given an oscillating trace in $\mathcal{A}$, its corresponding trace in $\mathcal{A}_s$ is also oscillates with the same period. This additional automaton can be used to determine test verdicts for the traces in the computed test cases. If an observation sequence corresponds to a path entering the "Error" location, then it causes a 'fail' verdict. Since we cannot use finite traces to prove a safety property, the set of obsevation sequences that cause a "pass" verdict is empty, and therefore the remaining obsevation sequences (that do not cause a "fail" verdict) cause a "inconclusive" verdict. We consider a constant input voltage $u_{in} = 1.7$. The coverage measure was defined on the projection of the state space on v_{C_1} and v_{C_2}. The generated test case shows that *after the transient time*, under a time-variant deviation of C_2 which ranges within $\pm 10\%$ of the value of $C_2 = 0.1e-4$, the variables v_{C_1} and v_{C_2} oscillate with the period $T \in [1.25, 1.258]s$ (with $\varepsilon = 2.8e-4$). This result is consistent with the result presented in [63]. The number of generated states was 30000 and the computation time was 14mn. Figure 4.20 shows the explored traces of v_{C_2} over time.

Delta-Sigma circuit. The third example is a third-order Delta-Sigma modulator [18], which is a mixed-signal circuit shown in Figure 4.21. When the input

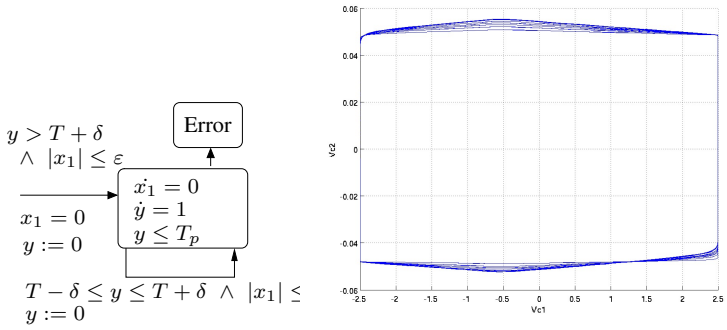


Figure 4.20: Left: Automaton for an oscillation specification. Right: Variable v_{C_2} over time. The number of generated states was 30000 and the computation time was 14mn.

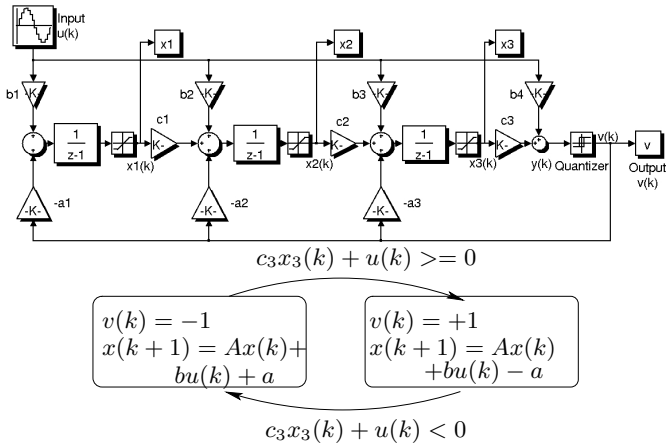


Figure 4.21: Model of a third-order modulator: Saturation blocks model saturation of the integrators.

sinusoid is positive and its value is less than 1, the output takes the +1 value more often and the quantization error is fed back with negative gain and accumulated in the integrator $\frac{1}{z-1}$. Then, when the accumulated error reaches a certain threshold, the quantizer switches the value of the output to -1 to reduce the mean of the quantization error. A third-order Delta-Sigma modulator is modeled as a hybrid automaton, shown in Figure 4.21. The discrete-time dynamics of the system is as follows: $x(k+1) = Ax(k) + bu(k) - sign(y(k))a$, $y(k) = c_3x_3(k) + b_4u(k)$ where $x(k) \in \mathbb{R}^3$ is the integrator states, $u(k) \in \mathbb{R}$ is the input, $y(k) \in \mathbb{R}$ is the input of the quantizer. Thus, its output is $v(k) = sign(y(k))$, and one can see that whenever v remains constant, the system dynamics is affine continuous. A modulator is stable if under a bounded input, the states of its integrators are bounded. The test generation algorithm was performed for the initial state $x(0) \in [-0.01, 0.01]^3$ and the input values $u(k) \in [-0.5, 0.5]$. After exploring only 57 states, saturation was already detected. The computation time was less

than 1 second. Figure 4.22 shows the values of $(\sup x_1(k))_k$ as a function of the number k of time steps. We can see that the sequence $(\sup x_1(k))_k$ leaves the safe interval $[-x_1^{sat}, x_1^{sat}] = [-0.2, 0.2]$, which indicates the instability of the circuit. This instability for a fixed finite horizon was also detected in [37] using an optimization-based method.

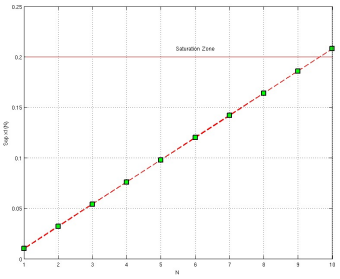


Figure 4.22: Test generation result for the Delta-Sigma circuit. The computation time was less than 1s.

4.15 Related work

Classical model-based testing frameworks use Mealy machines or finite labeled transition systems and their applications include testing of digital circuits, communication protocols and software. Recently, these frameworks have been extended to real-time systems and hybrid systems. Here we only discuss related work in hybrid systems testing. The paper [97] proposed a framework for generating test cases from simulation of hybrid models specified using the language CHARON [8]. In this work, the test cases are generated by restricting the behaviors of an environment automaton to yield a deterministic testing automaton. A test suite can thus be defined as a finite set of executions of the environment automaton. It is mentioned in the paper that to achieve a desired coverage, nondeterminism in the environment automaton is resolved during the test generation using some randomized algorithm. However, this coverage as well as the randomized algorithm were not described in detail. Besides testing a real system, another goal of this work is to apply tests to models, as an alternative validation method. In [72], the testing problem is formulated as to find a piecewise constant input that steers the system towards some set, which represents a set of bad states. To our knowledge, there is no other work in developing a formal framework for conformance testing that follows the standards of FMCT (Formal Methods in Conformance Testing) as closely as the framework we proposed.

The **RRT** algorithm has been used to solve a variety of reachability-related problems such as hybrid systems planning, control, verification and testing (see

for example [46, 25, 72, 29, 91] and references therein). Here we only discuss a comparison of our approach with some existing RRT-based approaches for the validation of continuous and hybrid systems. Concerning the problem of defining a hybrid distance, our hybrid distance is close to that proposed in [72]. The difference is that we use the centroids of the guard sets to define the distance between these sets, while the author of [72] uses the minimal clearance distance between these sets, which is harder to compute. To overcome this difficulty, the author proposed to approximate this clearance distance by the diameter of the state space. An advantage of our hybrid distance is that it captures better the average cases, allowing not to always favor the extreme cases. Note also that our hybrid distance d_H does not take into account the system dynamics. It is based on the spatial positions of the states. In [72] the author proposed a time-based metric for two hybrid states, which can be seen as an approximation of the minimal time required to reach from one state to another, using the information on the derivatives of the variables. Another distance proposed in [72] is called specification-based. This distance is typically defined with respect to some target set specifying some reachability property. It can be however observed that for many systems, this "direct" distance may mislead the exploration due to the controllability of the system. In [46, 72] and in our **hRRT** algorithm, the problem of optimally steering the system towards the goal states was not addressed. In other words, the evolution of the tree is mainly determined by the selection of nearest neighbors. In [25], the problem of computing optimal successors was considered more carefully, and approximate solutions for linear dynamics as well as for some particular cases of non-linear dynamics were proposed. The authors of [91] proposed a search on a combination of the discrete structure and the coarse-grained decomposition of the continuous state space into regions, in order to determine search directions. This can be thought of as an implicit way of defining a hybrid distance as well as a guiding heuristics.

Concerning test coverage for continuous and hybrid systems, in [46] the authors proposed a coverage measure based on a discretized version of dispersion, since the dispersion is very expensive to compute. Roughly speaking, the dispersion of a point set with respect to various classes of range spaces, such as balls, is the area of the largest empty range. This measure is defined over a set of grid points with a fixed size δ. The spacing s_g of a grid point g is the distance from g to the nearest visited state by the test if it is smaller than δ, and $s_g = \delta$ otherwise. Let S be the sum of the spacings of all the grid points. This means that the value of S is the largest when the set of visited state is empty. Then, the coverage measure is defined in terms of how much the vertices of the tree reduce the value of S. It is important to note that while in our work, the coverage measure is used to guide the simulation, in [46] it is used as a termination criterion. The paper [71] addresses the problem of robust testing by quantifying the robustness of some properties under parameter perturbations. This work also considers the problem

of how to generate test cases with a number of initial state coverage strategies.

Concerning guided exploration, sampling the configuration space has been one of the fundamental issues in probabilistic motion planning. Our idea of guiding the test generation via the sampling process has some similarity with the sampling domain control [107]. As mentionned earlier, the **RRT** exploration is biased by the Voronoi diagram of the vertices of the tree. If there are obstacles around such vertices, the expansion from them is limited and choosing them frequently can slow down the exploration. In the dynamic-domain **RRT** algorithm, the domains over which the goal points are sampled need to reflect the geometric and differential constraints of the system, and more generally, the controllability of the system. In [83], another method for biasing the exploration was proposed. The main idea of this method is to reduce the dispersion in an incremental manner. This idea is thus very close to the idea of our guiding method in spirit; however, their concrete realizations are different. This method tries to lower the dispersion by using K samples in each iteration (instead of a single sample) and then select from them a best sample by taking into account the feasibility of growing the tree towards it. Finally, we mention that a similar idea was used in [46] where the number of successful iterations is used to define an adaptive biased sampling. To sum up, the novelty in our guiding method is that we use the information about the current coverage of the visited states in order to improve the coverage quality. Additionally, we combine this with controllability information (obtained from the disparity estimation) to obtain a more efficient guiding strategy.

Future work

A number of directions for future research can be identified. First, we are interested in defining a measure for trace coverage. Partial observability also needs to be considered. Convergence rate of the exploration in the test generation algorithm is another interesting theoretical problem to tackle. This problem is particular hard especially in the verification context where the system is subject to uncontrollable inputs.

Chapter 5

Scheduling of real-time multi-threaded programs

5.1 Context

In addition to the work on design of embedded systems based on hybrid systems analysis techniques, I am interested in the topic of implementation of embedded systems, where scheduling is a recurrent problem. This work was initiated in late 2003 by Philippe Gerner, who was a postdoctoral researcher in VERIMAG. He introduced me to PV diagrams and a timed version of this model which can be used to reason about the behavior of a class of multi-threaded real-time programs. In these programs, the order of events in each of N threads can be represented on one dimension of $\mathbb{R}^N$ and their synchronisation and precedence requirements can be expressed as "feasible" region in $\mathbb{R}^N$. A program schedule corresponds to a trajectory from some starting point to the end point that remains in the feasible region. I was immediately interested in the geometry of PV diagrams that forms orthogonal polyhedra (which is a special class of non-convex polyhedra described by unions of hyper-rectangles). Indeed, I had used this class of polyhedra for reachability computation purposes and developed a number of algorithms for manipulating them.

We first developed a method for finding quick schedules using a decomposition of such orthogonal polyhedra. Although the method was used to analyze a number of interesting programs, its efficiency was limited by the complexity of constructing and manipulating orthogonal polyhedra. To avoid explicit contructions, we looked for a sufficient condition to quickly test whether there exists a feasible schedule connecting two given points. Our discovery of such a condition

brought about a new scheduling algorithm which can exploit random explorations similar to RRTs, which were used for testing purposes (see Chapter 4).

This work shows the computational advantages of PV programs, compared to timed automata, with respect to solving this scheduling problem. In fact, their geometry is simple enough to benefit from efficient geometric computations (on boxes). We are currently continue this work in various directions. One direction is to extend the model towards more complex specifications, such as those with deadlines and branching. Another direction is to focus on problems with particular geometry (such as the job shop scheduling problems).

5.2 Introduction

With the decreasing cost of embedded systems, constructing "more intelligent" embedded systems becomes possible and now product designers ask for more functionalities from an embedded system. This increases the portion of software in the system, to the point that some applications require programming with threads—that is, what was before designed as separate hardware components on a chip can now be achieved as separate threads. These threads can then be executed on one or several (in the case of multiprocessing) hardware components. But parallel programming in a real-time context is rather new, and much work is to be done in order to be able to analyse the real-time behavior of such programs. Indeed, simple extensions of existing analysis tools for sequential programs are not sufficient: parallelism with threads involves purely parallel-specific phenomena like deadlocks. In this work we examine the behavior of a class of multi-threaded programs, from the point of view of the *worst-case response time* (WCRT). In order to address this complex issue, we employ a geometric approach, which enables us not only to better "see" what happens, but also to exploit the geometric nature of the model in order to deal with the state explosion problem arising in the analysis of concurrent programs. We will also address the issue of scheduling parallel programs on a limited number of processors, using the same approach.

Our ideas of exploiting the geometry of parallel programs are inspired by the work on *PV diagrams*. This model for geometrically describing interactions of concurrent processes was introduced by Dijkstra [43]. It has been used, since the beginning of the 90's, for the analysis of concurrent programs [60, 48] (see [62] for a good survey). In particular, we follow the spirit of [48], where the geometry of the diagrams is used to construct an efficient analysis algorithm. We use the notion of *timed PV diagrams*, which can be used in the context of real-time concurrent programming. We focus on a particular problem: finding a schedule which is safe (no deadlocks), and short, that is although we are not looking for a shortest schedule, we want to find one which is as short as possible within a reasonable computational time. More precisely, this schedule specifies how the

resources should be shared by the threads, so that the serving capacity of each resource is respected and, in addition, the execution time of the program is as small as possible. To determine such a schedule, one needs to resolve the conflicts between two or more threads that happen when their simultaneous demand for the same resource exceeds the serving capacity of that resource. A resolution here means a decision to which threads to give the resource first and which threads have to wait until the resource is released. The motivations of this scheduling problem are:

- The program under consideration might be part of a global system (for example, the body of an infinite loop) and subject to a deadline. However, if no precise timing analysis result is available, then to determine the worst-case response time (WCRT) of the program one must assume the worst cases, i.e., all threads share a single protected resource (with a single access). This amounts to sequentialization of the threads, and the WCRT is the sum of the WCRT of each thread considered individually. This measure can easily be greater than the deadline, while the real worst case concurrent behavior of the program is probably better. Here we are interested in analysing the real WCRT and providing a guaranteed worst case execution time (also called worst-case response time). In addition, from the schedule, the designer can gain a lot of insight about other properties of the program executions, such as the frequency and duration of waits.

- Using the scheduling methods we propose, finding a deadlock-free schedule comes "for free" when looking for a short schedule.

- Finally, in a more general consideration, we believe that if finding a short schedule is computationally feasible, it is a good investment to design systems using such efficient schedules. For example, by reducing the computation time one can reduce energy consumption.

The chapter is structured as follows. In Section 5.3 we recall the concepts related to PV programs and diagrams. We then describe our timed version of PV programs, its geometric representation and we define the `worst case response time` WCRT of a schedule. We also discuss the case when there are less processors than threads in Section 5.5. The definitions and notions introduced in this part are necessary for the development of the results that follow. In Section 5.6 we explain a discrete abstraction of schedules. Based on this abstraction, two methods for computing efficient schedules are then developped. The first method, based on a spatial decomposition, is discussed in Section 5.7. In Section 5.8, we describe the second method, which uses a geometric property of the PV model to transform the scheduling problem to a path planning problem. The experimental results obtained using each method are also reported. Finally, in Section 5.9 we

describe some related work, and in Section 5.10 we conclude and present future work.

5.3 PV programs and diagrams

In this section we describe how to model real-time behavior of multi-threaded programs using PV programs, a model introduced by Dijkstra [43]. The reader is referred to [60, 48, 62] and the references therein for the results on the application of this model in the analysis of concurrent programs. We model each thread as a process, and a set of threads running together is modeled as a PV program. In the PV vocabulary, P stands for "lock", and V stands for "unlock" or "release"; it is however important to emphasize that, in our modeling framework, the actions "P" and "V" model the events of taking and releasing a resource, and they do not necessarily mean locking and unlocking a resource in a concrete implementation. These actions are used to specify resource usage constraints, that is some resources need to be used in a certain order and within some amount of time. A classical PV program example, called the Swiss flag example, is as follows:

$$A = \bot_A.P_a.P_b.V_b.V_a.\top_A \tag{5.1}$$
$$B = \bot_B.P_b.P_a.V_a.V_b.\top_B \tag{5.2}$$

where `a` and `b` are resources whose serving capacity is 1. We assume that the threads can always run concurrently, that is a thread can run as soon as it gets all the required resources. Hence, in this example, both threads `A` and `B` are assumed to have their own processor to run on. A model for the cases where the threads have to share processors is discussed in Section 5.5. In the following, we give the formal definition of the model.

5.3.1 PV programs: formal definitions

Letters in bold are often used to denote vectors and subscripts to denote the components of a vector, for example a_i is the i^{th} component of vector $\boldsymbol{a}$. Supercripts are often used to denote the elements of a sequence, such as a^i is the i^{th} element of sequence $\{a^0, a^1, \ldots, a^m\}$.

`Resources`

The shared resources are represented by a set $\Re$ of `resource names`. Each resource has a serving capacity[1], represented by a function *limit* : $\Re \rightarrow \mathbb{N}_+$. To

[1] In the PV vocabulary we say that the resource is protected by a semaphore.

model resource usage, we consider two types of `resource actions`: taking and releasing a resource $r \in \Re$, denoted respectively by P_r and V_r.

`Threads`

We consider a set of N `threads`: $E_1, \ldots, E_N$. Each thread E_i is a total order of `events`. Each event e has an associated resource action, for example P_r. The order relation of E_i is denoted by $\sqsubseteq_{E_i}$ (or simply by $\sqsubseteq$ when the context is clear). Each thread E_i contains at least two special events: its `start event` $\bot_{E_i}$ and its `end event` $\top_{E_i}$, which are respectively the bottom and top elements of the order. The threads are assumed to be *well-behaved*, in the sense that each resource should be released before it is taken again by the same thread. We say that thread E_i is `accessing` resource r at event e iff P_r has occurred before or at e and, additionally, the corresponding release action V_r occurs (strictly) after e. The running together of N threads is modeled by the product $\mathcal{E} = \prod_{i=1,\ldots,N} E_i$. We denote by $\preccurlyeq$ the order of $\mathcal{E}$, which is defined componentwise. An element of $\mathcal{E}$ is called a `state` and often denoted by the letter $\boldsymbol{\varepsilon}$, and thus $\boldsymbol{\varepsilon}_i$ is its event on thread E_i. We denote by $\bot = (\bot_{E_1}, \ldots, \bot_{E_N})$ the `bottom state` of $\mathcal{E}$ and by $\top = (\top_{E_1}, \ldots, \top_{E_N})$ its `top state`. We denote by E the union $\bigcup_{i=1,\ldots,N} |E_i|$ where $|E_i|$ is the set of events of E_i.

If B is a partial order and $b, b' \in B$ are such that $b \sqsubset b'$, the pair of these elements is called an `arc` and denoted by $\langle b, b' \rangle$. Also, if B is a total order and if $b \in B$ and $b \neq \bot_B$, then $\mathrm{pred}_B(b)$ denotes the `direct predecessor` of b in B, that is $\mathrm{pred}_B(b) \sqsubseteq b' \sqsubset b \implies b' = \mathrm{pred}_B(b)$. When the order is clear from the context, we simply write $\mathrm{pred}(b)$. The notion of direct successors can be defined similarly.

`Forbidden states`

A state $\boldsymbol{\varepsilon} \in \mathcal{E}$ is said to be `forbidden` if at $\boldsymbol{\varepsilon}$ there is at least one resource to which the number of concurrent accesses is greater than its limit, that is $\exists r \in \Re : \sum_{i=1,\ldots,N} \mathit{accessing}_i(r, \boldsymbol{\varepsilon}) > \mathit{limit}(r)$ where $\mathit{accessing}_i(r, \boldsymbol{\varepsilon}) = 1$ if thread E_i is accessing resource r at $\boldsymbol{\varepsilon}_i$ and $\mathit{accessing}_i(r, \boldsymbol{\varepsilon}) = 0$ otherwise. We denote by F the set of all forbidden states of $\mathcal{E}$, and by $\mathcal{A}$ the set of all `allowed` states, which is the complement of F.

`Strings`

An arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ is called a `small step` if $\forall\, i \in \{1, \ldots, N\} :\ \mathrm{pred}(\boldsymbol{\varepsilon}'_i) \sqsubseteq_{E_i} \boldsymbol{\varepsilon}_i \sqsubseteq_{E_i} \boldsymbol{\varepsilon}'_i$. For example, in the diagram of Figure 5.1 (left) the dotted arrows are small steps from $\preccurlyeq$.

Definition 9. *A* `string` s *is a total suborder of* $\mathcal{E}$ *such that for each state* $\boldsymbol{\varepsilon}$ *in* $s \setminus \{\bot_s\}$*, the arc* $\langle pred_s(\boldsymbol{\varepsilon}), \boldsymbol{\varepsilon}\rangle$ *is a small step.*

A string, which does not contain a forbidden state and hence does not induce any resource access conflicts, is called a `feasible string`. In [57], we defined a string as a subset of $\mathcal{A}$, and hence all such strings are by definition feasible. The idea of not restricting to the elements of $\mathcal{A}$ is to separate time constraints and resource usage constraints in order to model time more explicitly, as we shall show later.

5.3.2 Geometrization

A mapping of a program and its schedules to a diagram and trajectories is called a `geometrization` (or geometric realization) of the program. Intuitively, it involves mapping the set of schedules to trajectories inside an N-dimensional cube, going from the bottom left vertex of the cube (corresponding to the state $\bot$ of the program) to the top right vertex (corresponding to the state $\top$). In the following, we describe a geometrization in $\mathbb{Z}^N$. We use notation "$\rightharpoonup$" for the mapping; hence, $\overline{s}$ is the image of string s by this mapping. We map each thread E_i onto a subset of $\mathbb{N}$ as follows. Each event e of thread E_i is associated with an ordinate $c(e)$. The ordinates are defined inductively as follows:

- $c(\bot_{E_i}) = 0$.
- $c(e) = c(\text{pred}(e)) + 1$ if $e \neq \bot_{E_i}$.

The order of E_i is mapped onto the order $\leq$ between the integers $c(e)$. We denote by $\overline{E_i}$ the resulting total order $(\{c(e) \mid e \in |E_i|\}, \leq)$. This mapping is clearly an isomorphism of total orders. The geometrization of $\mathcal{E}$, denoted by $\overline{\mathcal{E}}$, is defined as the product of partial orders $\prod_{i=1,\ldots,N} \overline{E_i}$, and is isomorphic to $\mathcal{E}$. Every state $\boldsymbol{\varepsilon} = (\boldsymbol{\varepsilon}_1, \ldots, \boldsymbol{\varepsilon}_N) \in \mathcal{E}$ is sent to a `point` $\overline{\boldsymbol{\varepsilon}} = (c(e_1), \ldots, c(e_N))$ in $\mathbb{Z}^N$. The set of forbidden states F is mapped onto $\overline{F}$. The forbidden states have an intuitive geometric interpretation. To explain this, we need some additional notation.

Geometrically speaking, in $\mathbb{R}^N$, the geometrization $\overline{\mathcal{E}}$ forms a non-uniform N-dimensional grid $\mathcal{G}$ over the bounding box $\mathcal{B} = [0, c(\top_1)] \times \ldots \times [0, c(\top_N)] \subset \mathbb{R}^N$. A point $\overline{\boldsymbol{\varepsilon}} = (c(\boldsymbol{\varepsilon}_1), \ldots, c(\boldsymbol{\varepsilon}_N)) \in \mathbb{R}^N$ is called a `grid point`. Given a box $B = [l_1, u_1] \times \ldots \times [l_N, u_N]$, its associated right-open box is defined as $B' = \{\mathbf{x} \mid \forall i \in \{1, \ldots, N\} : l_i \leq \mathbf{x}_i < u_i\}$ where $\mathbf{x}_i$ is the i^{th} coordinate of the point $\mathbf{x}$. For every $\boldsymbol{\varepsilon} \in F$, let $box(\boldsymbol{\varepsilon})$ be the elementary box whose bottom left vertex is $\overline{\boldsymbol{\varepsilon}}$. An elementary box is a box such that all its vertices are grid points and, additionally, its interior does not contain any grid points. Then, the associated right-open box of $box(\boldsymbol{\varepsilon})$ is called the `elementary forbidden box` associated

with $\boldsymbol{\varepsilon}$, denoted by $obox(\boldsymbol{\varepsilon})$. The union of all such boxes $P_F = \bigcup_{\boldsymbol{\varepsilon} \in F} obox(\boldsymbol{\varepsilon})$ is called the `forbidden region` (whose closure is indeed a non-convex polyhedron with axis-parallel faces). Geometrically, schedules are trajectories that do not touch the front boundary of the forbidden boxes. We can prove that a feasible schedule never enters the forbidden region. Similarly, the `allowed region` is defined as $P_A = \mathcal{B} \setminus P_F$.

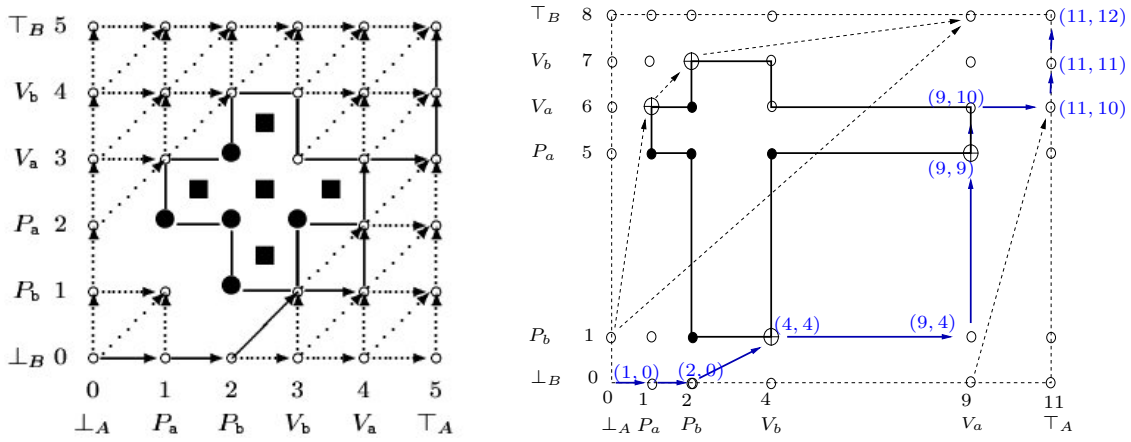


Figure 5.1: The PV diagrams of the Swiss flag program: untimed (left) and timed (right) versions.

The PV diagram of the Swiss flag program is shown in Figure 5.1 (left). The intuitive meaning of the diagram is that a schedule for the program is represented by a sequence of arrows from $(\bot_A, \bot_B)$ to $(\top_A, \top_B)$. Indeed, any possible schedule is a particular order of resource actions of threads `A` and `B`. A schedule is shown in the figure, drawn in solid arrows. All the other arrows, drawn in dotted lines, are those that a schedule could follow. The black circles indicate the forbidden states. For example, point $(2, 1)$ is forbidden because its associated combination of actions $(P_\mathtt{b}, P_\mathtt{b})$ means that both threads `A` and `B` access resource `b` at the same time, which is not possible since the serving capacity of `b` is 1. The small black squares in the figure mark the forbidden boxes. The "Swiss flag" name of the example comes from the cross form of the union of these forbidden squares. An advantage of such diagrams is that they allow to visualize special behaviors of a program. In this example it is easy to see two special cases: point $(1, 1)$ is a *deadlock* and point $(3, 3)$ is *unreachable*.

5.4 Timed PV programs and diagrams

We now introduce our timed version of PV programs and diagrams. This version differs from the existing versions of timed PV programs and diagrams [61, 47],

which is discussed in Section 5.9, where we also explain the motivation of this new version.

5.4.1 Timed PV programs

`Task duration`

Our version of timed PV programs is an enrichment of the classic PV program model with a task duration between every two consecutive events of each thread. Indeed, in practical real-time programming, one can estimate the duration of the execution of the program code between two events. The estimation is usually done to account for the worst cases; such a duration is thus a *worst-case execution time* (WCET). So we associate with each event of a thread the duration (or the WCET) of the task corresponding to the part of the program code which is run between the occurrences of this event and of its direct successor. When event $e \in E_i$ occurs, we say that thread E_i starts task e. The *task durations* are defined by a function $d : E \to \mathbb{R}_+$, and for each thread E_i, $d(\top_{E_i}) = 0$.

As an example, the following is a timed version of the Swiss flag program:

$$\begin{aligned} A &= \bot_A.1.P_a.1.P_b.2.V_b.5.V_a.2.\top_A, \\ B &= \bot_B.1.P_b.4.P_a.1.V_a.1.V_b.1.\top_B \end{aligned}$$

The numbers between the actions are the task durations. For example, the first number 1 in thread `A` is the duration of the task which is executed between its beginning and its first action $P_{\mathtt{a}}$.

`Timed execution`

The notion of strings, introduced in the untimed context, does not capture time information. To this end, we introduce a notion of timed states and timed executions. A `timed state` is a pair $\boldsymbol{\mu} = (\boldsymbol{\varepsilon}, t)$ where $\boldsymbol{\varepsilon} \in \mathcal{E}$ and t is a non-negative real number. Given a timed state $\boldsymbol{\mu} = (\boldsymbol{\varepsilon}, t)$, it is called `forbidden` if $\boldsymbol{\varepsilon}$ is a forbidden state. The meaning of $(\boldsymbol{\varepsilon}, t)$ is that at time point t the latest event on thread E_i is $\boldsymbol{\varepsilon}_i$. A sequence of timed states $\gamma = \boldsymbol{\mu}^1, \ldots, \boldsymbol{\mu}^m = (\boldsymbol{\varepsilon}^1, t^1), \ldots, (\boldsymbol{\varepsilon}^m, t^m)$ is called a `timed execution`. A timed execution is `feasible` iff none of its states is forbidden. In addition, a timed execution is said to be `consistent` iff the event order and time constraints of all the threads are respected. This is formalized as follows.

Definition 10. *A* `timed execution` $\gamma = \boldsymbol{\mu}^1, \ldots, \boldsymbol{\mu}^m = (\boldsymbol{\varepsilon}^1, t^1), \ldots, (\boldsymbol{\varepsilon}^m, t^m)$ *is* `consistent` *iff the following conditions are satisfied for each thread* E_i, $i \in \{1, \ldots, N\}$*:*

1. *The sequence* $\boldsymbol{\varepsilon}_i^1, \ldots, \boldsymbol{\varepsilon}_i^m$ *is a string.*

2. *For each* $j \in \{2, \ldots, m-1\}$ *such that* $\boldsymbol{\varepsilon}_i^j \neq \boldsymbol{\varepsilon}_i^{j-1}$*, let* j' *be the smallest index strictly greater than* j *such that* $\boldsymbol{\varepsilon}_i^{j'} \neq \boldsymbol{\varepsilon}_i^j$*. If such* j' *exists, then* $t^{j'} - t^j \geq d(\boldsymbol{\varepsilon}_i^j)$*.*

The duration of γ *is defined by* $d(\gamma) = t^m - t^1$*. A* `feasible consistent timed execution` $\gamma = \boldsymbol{\mu}^1, \ldots, \boldsymbol{\mu}^m = (\boldsymbol{\varepsilon}^1, t^1), \ldots, (\boldsymbol{\varepsilon}^m, t^m)$ *with* $\boldsymbol{\varepsilon}^1 = \bot$ *and* $\boldsymbol{\varepsilon}^m = \top$ *is called a* `timed schedule`*.*

The first condition guarantees that the required task order (or event order) of each thread is respected. The second condition guarantees that the task duration constraints are satisfied. It can be interpreted as follows: each time lapse between two (different) consecutive events should be larger than the corresponding task duration (specified in the timed PV program). The above definition implies the time progress property of consistent timed executions since the sequence $t^1, \ldots, t^m$ is strictly increasing.

5.4.2 Geometrization

For a timed PV program we could define a geometrization as in the untimed case, since the involved orders are the same. However, it is more convenient to have a diagram where one can visualize durations. We use the same notation "–" for the mapping from a timed PV program to a diagram. Each thread E_i is mapped onto a subset of $\mathbb{R}$, by specifying for each event $e \in E_i$ an ordinate $c(e)$. Note that now we map program executions to trajectories in a subset of $\mathbb{R}^N$, instead of $\mathbb{Z}^N$ as in the untimed case, in order to exploit continuous geometric properties of the diagram, as we shall see in Section 5.8. More concretely, we define a geometrization where the diagram is scaled according to the task durations. The ordinates are inductively defined to visually reflect the task durations as follows:

$$\begin{cases} c(\bot_{E_i}) = 0, \\ c(e) = c(\mathrm{pred}_{E_i}(e)) + d(\mathrm{pred}_{E_i}(e)) \ \text{ if } \ e \neq \bot_{E_i}. \end{cases}$$

For tasks with zero duration, a fixed length $\alpha > 0$ is chosen. When the program does not have any zero duration tasks, such a geometrization is called `exact scaling geometrization`. The order of E_i is thus mapped to the order $\leq$ between the real numbers $c(e)$, and $\overline{E_i}$ is the resulting total order $(\{c(e) \mid e \in |E_i|\}, \leq)$. Again, this mapping is an isomorphism of total orders. The geometrization of $\mathcal{E}$ is defined as the product of partial orders $\overline{\mathcal{E}} = \prod_{i=1,\ldots,N} \overline{E_i}$ and is isomorphic to $\mathcal{E}$.

`Mapping States and Strings`

The set F of forbidden states is mapped to the set $\overline{F}$ of `forbidden points`. A geometrization of the timed version of the Swiss flag program is shown in Figure 5.1 (right), where the black circles indicate the forbidden states, and the white circles indicate the allowed states. A string s is mapped to a sequence $\overline{s}$ of grid points in the bounding box $\mathcal{B}$. In view of exploiting continuous geometric properties, we also define the continuous geometrization of an arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ as the directed line segment from vertex $\overline{\boldsymbol{\varepsilon}}$ to vertex $\overline{\boldsymbol{\varepsilon}'}$ and denote it by $\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$. The reason for this choice is that there is a relation, which we shall show later, between the feasible strings and the corresponding line segments.

`3D example: the timed dining philosophers`

To illustrate, we describe a timed version of the 3 philosophers problem. The philosophers need their left and right forks to eat. Each forks is modeled by a resource with serving capacity 1. We call these resources `a`, `b`, and `c`: the left fork of philosopher `A` is `a`, its right fork is `b`, and so on (and the right fork of `C` is `a`). We add a resource with serving capacity 2 (which we call `room`) to model the access to a small thinking room which can contain no more than 2 philosophers at a time. Each philosopher thinks in the thinking room, then walks to the eating room and eats. Non-zero task durations are given for thinking, walking, and eating. The program is the following:

```
A = 0. P room  .5. V room .4. P a .0. P b .15. V a .0. V b .0
B = 0. P room .14. V room .6. P b .0. P c  .5. V b .0. V c .0
C = 0. P room  .9. V room .9. P c .0. P a  .2. V c .0. V a .0
```

A geometrization of the program is shown in Figure 5.2 (left), where the points corresponding to the $\perp$ and $\top$ are marked with little white cubes. We also show, in Figure 5.2 (right), the geometry of a more complex version where, in addition, the three philosophers share a single anti-stress; and the philosophers `B` and `C` share a single ashtray.

5.4.3 Duration of strings

Note that a string corresponds to a uncountable number of timed executions; we define the `duration` of a string as the duration of the shortest timed executions corresponding to the string. These timed executions indeed correspond to the case where all the tasks take their WCET as effective duration, so this duration is the `worst-case response time` WCRT of the string. Formally, the duration of a string s, denoted by $d(s)$, is computed by the following algorithm, whose goal is to find "what time is at $\top_s$ at least" when time is 0 at $\perp_s$ and the

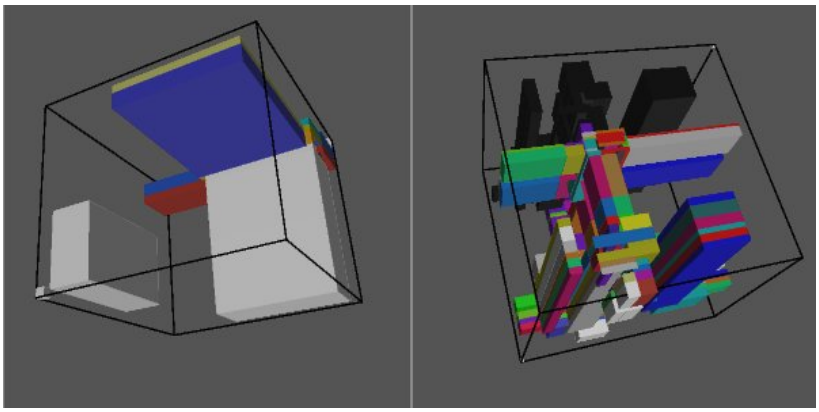


Figure 5.2: The three philosophers: (left) simple version; (right) enriched version

string is executed from $\bot_s$ to $\top_s$. The algorithm uses N `local clocks`, one for each thread, and one `global clock`. Let `H` denote the global clock, and `h` the array (of size N) of the local clocks: `h[i]` is the local clock of thread E_i. The algorithm also uses the notion of `new events`: for a string s and a state $\varepsilon \in s \setminus \{\bot_s\}$, the set of new events, denoted by $new_s(\varepsilon)$, that occur at ε is defined as $\{\varepsilon_i \ (i = 1, \ldots, N) \mid (\text{pred}_s(\varepsilon))_i \neq \varepsilon_i\}$. First, all clocks, global and local, are initialized to 0. Then we iterate over the sequence of states of s, beginning from the state just after $\bot_s$ and ending at $\top_s$.

Algorithm 8 Determining the duration of string s.

For all $i = 1, 2, \ldots, N$ **do** $h[i] := 0$
$H := 0$
for each $\varepsilon \in s$, beginning from the state just after $\bot_s$ and ending at $\top_s$ **do**
  / Step (1): Update the global clock according to new events */*
  For all i **s.t.** $\varepsilon_i \in new_s(\varepsilon)$ **do** $H := max(H, h[i] + d(\varepsilon_i))$

  / Step (2): Update the local clocks of the threads that have a new event. */*
  / This syncronizes these local clocks. */*
  For all i **s.t.** $\varepsilon_i \in new_s(\varepsilon)$ **do** $h[i] := H$
end for

In Step (1) of the algorithm, if thread E_i has a new event at ε, then $d(\varepsilon_i)$ time units have elapsed since time $h[i]$, so the global time now must be at least $h[i] + d(\varepsilon_i)$. The "max" function is needed because it is possible that $h[i] + d(\varepsilon_i)$ is not greater than the last H recorded (see the example that follows).

`Example` The algorithm is illustrated with the schedule shown in Figure 5.1 (right). The vector-like annotations that accompany the trajectory indicate the

values of the local clocks during the execution of the algorithm. (The value of the global clock is always the maximum of the values of the local clocks.) We explain what happens at some particular states. The states are denoted by their coordinates in the diagram. At state $(4, 1)$ a new event happens to thread A and to thread B. Thread A updates the global clock to 4 in step (1) of the algorithm, and thread B does not update the global clock since $0 + 1 < 4$. The global clock `H` is now 4, and both local clocks are updated to 4. That is, the schedule specifies that action P_b of thread B must not happen before action V_b of thread A, so since A runs for 4 time units before executing V_b, B cannot execute P_b before that time point. So B has a lapse of 4 time units for executing its task of duration 1. It means that if it finishes this task before $t = 4$ (for example if it begins the task at $t = 0$), it must *wait* for 3 time units until A releases resource b.

The final value of the global clock, 12, defines the WCRT of this schedule. Note that if at a state, a new event happens to a thread, but the duration of the task before this event is zero, then there is no change to be made to the clock of this thread.

5.5 Threads sharing a limited number of available processors

We have defined the WCRT of a schedule *assuming that the threads run concurrently*. But what does the WCRT of the schedule become when there are only $M < N$ available processors? This is the problem we tackle in the present section. The problem of mapping N threads (or processes) onto M processors has already been treated in [35], but in the untimed context. We are now in the timed context and our goal is to find *short* schedules. We can distinguish two approaches:

1. First compute an efficient schedule with the method shown in Section 5.7 above; and then compute a good mapping of this schedule onto the M processors. The inconvenient of this method is that the best abstract schedules do not necessarily perform well when M processors are shared, while some schedules which are inefficient with N processors might perform well under the constraint of $M < N$ processors.

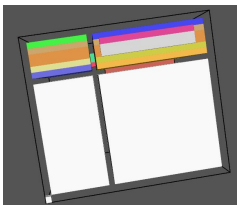
2. Take this processor constraint into account when computing a short schedule. The advantage of this approach is that it is more precise. The inconvenient is that adding this constraint may lead to a state explosion problem. In this section we examine this solution, because it gives some geometric intuition about the mapping, and in addition, for many practical cases the computational complexity is reasonable.

`Processor Resource` The idea is to use a resource with serving capacity M to model the fact that M processors are shared between the threads. This modelling assumes that the threads have no preference on which processor to run on, which is reasonable in the case of an homogeneous architecture—all the processors are identical. The advantage of using a resource is a drastic combinatorial simplification: from the point of view of scheduling, we are only interested in knowing which $Q \leq M$ (among the N) threads are running. So we do not need to tell which processor each of these Q threads is running on. The distribution onto the processors can be done after determining the schedule.

The essential idea of the solution is that the programmer indicates in his program the releasing and re-accessing of the processor resource, which corresponds to a *proposition of preemption*: the thread gives other threads a chance to take the processor. Of course if the schedule which is eventually chosen does not use this preemption opportunity, then in the implementation of the schedule the thread does not need to preempt itself. The mechanism is illustrated with the philosophers program of Section 5.4.2. Suppose that the programmer decides that a philosopher makes a proposition of preemption just before entering the thinking room. We denote by `p` the resource for the processors. The corresponding modified program of philosopher `A` is shown in Figure 5.3(a) (the modification is similar for philosophers `B` and `C`), and the geometry of the new program is shown in Figure 5.3(b). The accesses to `p` creates big boxes, and the propositions of

```
A =   0.  Pp .
  0.  P room .5.  V room .4.
      Vp .0.  Pp .0.
  P a .0.  P b .15.  V a .0.  V b .0
     .  Vp .0
B = ...
C = ...
```

(a)



(b)

Figure 5.3: The three philosophers problem with two processors

premption create some "canyons" (of width α, which is equal to 1 in this example) between these boxes. A trajectory must go through these canyons and avoid the parts of the previous forbidden regions that still emerge from the new boxes (here the bars from the forks). Notice that the forbidden box for `room` is now included in the bottom left `p`-forbidden box (and is thus invisible). Indeed, the `room` resource served to forbid concurrent access to the room by more than two philosophers, which is no longer necessary since only two processors are available.

Since each philosopher accesses 2 times a processor (through a lock of `p`), we get $2^3 = 8$ corresponding forbidden regions. Computationally it means that a thread should not propose preemption too often. On the other hand, finding the optimal schedule for all possible preemptions would imply proposing a preemption between each event of the original program (which can be done automatically). But this would induce an exponential number of forbidden regions for the processor resource. On the other hand, this geometric approach can give new ideas for optimizing the control of programs that run on a limited number of processors. For example, in the above example the geometry indicates that if the given preemption is implemented, then the implementation can dispense with the `room` resource.

5.6 Scheduling problem and abstraction of timed executions

5.6.1 Scheduling problem

With the definition of the duration of a string in Section 5.4.3, we can now formally state our scheduling problem as computing a timed schedule with the shortest duration, which we simply call a `shortest` or `optimal` schedule. To this end, we define an abstractions of timed schedules. We defer a discussion on related models and problems, in particular timed automata and job-shop scheduling, to Section 5.9.

We observe that if we are looking for a shortest schedule, enumerating all possible schedules is not feasible in general. Indeed, the combinatorial explosion comes not only from the number of possible states, but also from the total number of possible schedules from bottom to top. If we include the forbidden schedules (which can pass through some forbidden states) to simplify computations, we get the following numbers: for the timed Swiss flag example, $6 \times 6 = 36$ states and 1683 possible schedules; for the timed philosophers example, $8 \times 8 \times 8 = 512$ states and 75494983297 possible schedules; for the enriched version of the timed philosophers, $16 \times 18 \times 26 = 7488$ states and more than 5×10^{30} possible schedules.[2] Given this complexity problem, we propose to exploit the geometry of the diagrams to construct an abstraction of shortest schedules. Before continuing, we remark that due to the growing complexity of embedded systems, finding an optimal schedule often requires prohibitive computation time, and hence a problem of great interest is to compute `good` or `short` schedules (that is, those close to the optimal ones) in a reasonable time. This is indeed our practical goal, and therefore although the theoretical results in the chapter addressed the optimality

[2]—5589092438965486974774900743393, to be precise.

criterion, we also proposed a practical non-exhaustive method to achieve a good trade-off between computation time and optimality.

5.6.2 Discrete abstraction of timed executions

In our timed PV program model, a string can be thought of as a discrete abstraction of timed executions. Recall that the `duration of a string` is the duration of a shortest (concrete) consistent timed execution corresponding to the string. In other words, it is the duration of such a timed execution where all the resource actions are taken as soon as possible. The algorithm described in Section 5.4.3 to determine the duration of a string can be seen as a constructive definition of this notion.

`Eager Strings` We are now interested in strings with no unnecessary wait, which are called `eager` strings. A thread waits out of necessity when its next resource is unavailable. In Figure 5.1 (right), an example of a non-necessary wait is a schedule that would go, for example, through points $(4, 0)$ and $(9, 0)$ before going to $(9, 1)$, which means that thread B waits until thread A releases resource a before accessing resource b while resource b is already available. Notice that a shortest schedule is necessarily eager; the other direction is however not true in general; in Figure 5.1 (right), a string from $\bot$ to $\top$ that would go *above* the cross could be eager, but not optimal. Indeed, since thread A has to wait for resources a and b to be unlocked by thread B, a shortest string that goes above the cross has duration $5 + 1 + 9 = 15$ time units.

The notion of bows that we introduce in the following is indeed a way to abstract eager strings. The main idea is to see whether it is possible to make a 'big' step instead of small steps in a string.

`Bows: abstractions of wait-rree strings` Intuitively, a bow is an arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ from $\mathcal{A}$ such that the longest side of the box in the geometrization whose bottom left and top right corners correspond to $\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}'$ is equal to the duration of the shortest strings from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$.

Definition 11. *Given any arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ from $\mathcal{A}$, the* `stringing` *of $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$, which we denote by $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\!\searrow$, is the set of all the strings from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$ that have the smallest duration.*

This set is not empty, since $\boldsymbol{\varepsilon} \preccurlyeq \boldsymbol{\varepsilon}'$ implies that there is a sequence of small steps from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$ in $\mathcal{A}$. We call the `tightened length` of an arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ from $\mathcal{A}$, the duration of any element of $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\!\searrow$. We extend notation d to sets of strings that have the same duration, and we denote the tightened length of $\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon} \rangle$ by $d(\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon} \rangle\!\searrow)$. This measure will be compared to the following *distance*.

Definition 12. *The* `max-distance` *between two states* $\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \in \mathcal{E}$ *with* $\boldsymbol{\varepsilon} \preccurlyeq \boldsymbol{\varepsilon}'$ *is defined as:* $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\| = \max_{i=1,\dots,N}(\sigma(\boldsymbol{\varepsilon}_i) - \sigma(\boldsymbol{\varepsilon}'_i))$*, where for any thread* E_i *and event* $e \in E_i$*:* $\sigma(e) = \sum_{\perp_{E_i} \sqsubset e' \sqsubseteq e} d(e)$.

Note that $\sigma(e) \neq c(e)$ in general; $c(e)$ is the ordinate of e for the geometrization, while $\sigma(e)$ is the "true ordinate" of e in term of the sum of the durations of the tasks: the difference comes from the tasks that have a duration equal to zero, and for which $\alpha > 0$ is applied in the geometrization. Now we can define the abstractions of wait-free-strings.

Definition 13. *An arc* $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ *from* $\mathcal{A}$ *where* $\boldsymbol{\varepsilon} \prec \boldsymbol{\varepsilon}'$ *is a* `bow` *if* $d(\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle \searrow) = \|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$.

Geometrically speaking, the max-distance $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$ is the longest side of the box whose bottom left and top right vertices are $\overline{\boldsymbol{\varepsilon}}$ and $\overline{\boldsymbol{\varepsilon}'}$. It is easy to see that one cannot expect to obtain a string from $\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}'$ with duration shorter than $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$ since it is exactly the time needed to execute the longest thread without waiting (i.e. without interruption). On the other hand, if there is at least one string from $\boldsymbol{\varepsilon}$ and $\boldsymbol{\varepsilon}'$ with duration equal to $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$, then the arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ is called a bow. As an example, in Figure 5.1 (right), the arc $\langle (9,0), (11,6) \rangle$ is a bow. Indeed, its max distance is 6, and the shortest feasible string from $(9,0)$ to $(11,6)$ goes directly along the arrow. More formally, this means letting the threads A and B start the resource actions V_a and $\perp_B$, and letting them run continuously until the thread B releases the resource a by the action P_a. This is possible because to execute this string, no common resources need to be used by both the threads. We note that there is an idle period on the thread A because the time needed to finish the task between V_a and $\top_A$ is 2 but the time needed for the thread B to finish the task between $\perp_B$ and V_a is 6.

As another example, the arc $\langle (0,1), (9,8) \rangle$ is not a bow. Indeed, the latter has the max-distance $\|\langle (0,1), (9,8) \rangle\| = 9$, while its tightened length is 13 since a shortest string $(0,1), (1,6), (2,7), (9,8)$ exchanges resource b at point $(1,6)$, and thread A has to wait until this exchange for at least 4 time units.

Remark 1. *Given a feasible string* s *consisting of two consecutive small steps* $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ *and* $\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon}'' \rangle$*, let* $d(s)$ *denote the duration of* s*. Then,* $d(s) \leq \|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\| + \|\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon}'' \rangle\|$.

The above remark can be explained with a simple program that has 2 concurrent threads. We first determine the smallest time δ needed to follow the first small step $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle = \langle (\boldsymbol{\varepsilon}_1, \boldsymbol{\varepsilon}_2), (\boldsymbol{\varepsilon}'_1, \boldsymbol{\varepsilon}'_2) \rangle$. If $\boldsymbol{\varepsilon}_i \neq \boldsymbol{\varepsilon}'_i$, we know that at least $[c(\boldsymbol{\varepsilon}'_i) - c(\boldsymbol{\varepsilon}_i)]$ time units have passed on thread E_i. Suppose that $\boldsymbol{\varepsilon}_1 = \boldsymbol{\varepsilon}'_1$ and $\boldsymbol{\varepsilon}_2 \neq \boldsymbol{\varepsilon}'_2$. Since no new event has occurred on E_1, the lower bound of the time lapse on E_1 is 0, and thus the global time lapse $\delta = [c(\boldsymbol{\varepsilon}'_2) - c(\boldsymbol{\varepsilon}_2)]$. We proceed with the second small step $\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon}'' \rangle$ and consider the following two cases:

- Case 1: No new event has occurred on E_1, the lower bound of the time lapse on thread E_1 is still 0, and the smallest time needed to follow these two consecutive steps is $[c(\boldsymbol{\varepsilon}_2'') - c(\boldsymbol{\varepsilon}_2')] + [c(\boldsymbol{\varepsilon}_2') - c(\boldsymbol{\varepsilon}_2)]$.

- Case 2: A new event occurred on E_1, which allows us to know that the lower bound of the time lapse on thread E_1 is $[c(\boldsymbol{\varepsilon}_1'') - c(\boldsymbol{\varepsilon}_1')]$. If again $\boldsymbol{\varepsilon}_2'' \neq \boldsymbol{\varepsilon}_2'$, combining the lower bounds of the time lapses on both threads, the smallest time lapse of these two consecutive steps is $max\{[c(\boldsymbol{\varepsilon}_1'') - c(\boldsymbol{\varepsilon}_1')], [c(\boldsymbol{\varepsilon}_2'') - c(\boldsymbol{\varepsilon}_2')] + [c(\boldsymbol{\varepsilon}_2') - c(\boldsymbol{\varepsilon}_2)]\}$. By definition, this is exactly the duration of the string s. We can see that $d(s) \leq max_i\{c(\boldsymbol{\varepsilon}_i') - c(\boldsymbol{\varepsilon}_i)\} + max_i\{c(\boldsymbol{\varepsilon}_i'') - c(\boldsymbol{\varepsilon}_i')\} = \|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\| + \|\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon}'' \rangle\|$.

The intuition behind this is that only when an event e occurs on a thread we can determine a lower bound of the time lapse on this thread since the occurrence of the previous event. This lower bound imposes a constraint on the global time at event e. When a new event simultaneously occurs on two or more threads, the global time is determined by combining the constraints imposed by all these threads, and we say that in this situation these threads 'synchronize'. Hence, the way of describing the time constraints of each thread on a separate dimension in a timed PV program can be thought of as de-synchronizing them, and the threads need to be re-synchronized only when their interaction affects the global behavior.

`Critical Potential Exchange Points` We define critical potential exchange points, the only points where an eager string can wait. A `potential exchange point` is an element $\boldsymbol{\varepsilon} \in \mathcal{A}$ where a resource can be exchanged: there exist at least one resource $r \in \Re$ and two indices i, j such that $\boldsymbol{\varepsilon}_i = V_r$ and $\boldsymbol{\varepsilon}_j = P_r$. We use the term "potential" because in order to be a real exchange point, it must be the element of a string s with $\boldsymbol{\varepsilon}_i, \boldsymbol{\varepsilon}_j \in new_s(\boldsymbol{\varepsilon})$.

Definition 14. *A potential exchange point which has accessing*$(r, \boldsymbol{\varepsilon}) = limit(r)$ *for some resource* r *is called a* `critical potential exchange point` *("CPEP" for short).*

In the Swiss flag example of Figure 5.1 (right), CPEPs are indicated by circled addition symbols. We sometimes call these states 'exchange states' for short. It is possible to characterize these states geometrically. We can prove that their geometrizations are indeed the boundary points belonging to at least one positive face of the forbidden polyhedron. Note that every face of the forbidden polyhedron is parallel to one of the axes, and a face is called positive if its normal vector points to the positive direction of the axis.

5.6.3 The abstraction graph

We are now ready to define our *abstraction of all the eager strings* (and hence also of all the shortest schedules). It is the graph having CPEPs, plus $\bot$ and $\top$, has nodes and bows as arrows. We call it the `abstraction graph`. We denote by C the union of all CPEPs with $\{\bot, \top\}$. The abstraction graph is the weighted graph defined as the binary relation $G \subseteq C \times C$ which is characterized by: $\boldsymbol{\varepsilon}\, G\, \boldsymbol{\varepsilon}' \iff \langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ is a bow; and each arc $(\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}')$ of G has weight (or cost) $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$.

For $\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \in C$ with $\boldsymbol{\varepsilon} \preccurlyeq \boldsymbol{\varepsilon}'$, we denote by $\widehat{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ the set of the shortest paths in G from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$. The length (or the cost) of a path in the graph G is the sum of the weights of its edges. By abuse of notation, we denote by $l(\widehat{\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon} \rangle})$ the length (or the cost) of any of the paths in $\widehat{\langle \bot, \top \rangle}$. Graph G has an important property:

Theorem 12. *The duration of a shortest schedule from $\bot$ to $\top$ is the length of a shortest path in G from $\bot$ to $\top$: $d(\langle \bot, \top \rangle\!\searrow) = l(\widehat{\langle \bot, \top \rangle})$.*

Proof. The proof is long so we only sketch it here. The full proof can be found in the technical report [57].

- The abstraction of an eager string s with $\bot_s, \top_s \in C$, which we denote by $s\nearrow$, is defined as the path in G which is constituted of all the CPEPs contained in s. (Since s is eager, bows exist between the CPEPs.)
- Let $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle \in \mathcal{A}$, with $\boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \in C$. We prove that there exists a string $s \in \langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\!\searrow$ which is such that $l(s\nearrow) = d(s)$. (The proof is by induction on the number of CPEPs in $s\nearrow$.)
- We then prove that for such a string, $s\nearrow$ is among the shortest paths in G from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$.
- Take a ρ in $\langle \bot, \top \rangle\!\searrow$, such that $l(\rho\nearrow) = d(\rho)$ (which is possible by the first above result). By the second above result, $\rho\nearrow$ is among the shortest paths from $\bot$ to $\top$.

Thus $d(\langle \bot, \top \rangle\!\searrow) = d(\rho) = l(\rho\nearrow) = l(\widehat{\langle \bot, \top \rangle})$. ∎

The intuitive meaning of the theorem can be explained as follows. It shows a special property of the shortest paths of G: if π is a shortest path, then the length (or the cost) of π is the duration of a shortest schedule, and π is an abstraction of this schedule. The problem of searching for a shortest schedule is thus reformulated as that of finding a shortest path in the graph G. To construct the abstraction graph, it suffices to consider the critical exchange states. The

existence of a bow between two such states indicates that there exists an *eager feasible string* or 'direct route' between these two states.

Note that the number of CPEPs is much smaller than the number of states in $\mathcal{E}$; hence the shortest paths in G from $\bot$ to $\top$ are easier to compute. However, this abstraction is useful only if one can efficiently determine whether an arc is a bow. Checking the condition given in Definition 13 could be complicated since it requires computing the tightened length $d(\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle)$, which is not trivial if one wants to avoid enumerating all feasible strings connecting $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$. In the following, we propose two methods to do so. The first method uses a spatial decomposition of the allowed region into boxes. Essentially, if a box does not contain any forbidden points, then any arc whose geometrization is inside the box is a bow. The second method exploits a property of the geometrization, which we explain in Section 5.8. This property allows us to quickly find long bows (i.e. long direct routes), using path planning techniques, and thus to speed up the search for a short schedule. The following section discusses the first method.

5.7 Scheduling using a spatial decomposition

The construction of graph G has two parts: 1) find the CPEPs; 2) find the bows between these points. Then a shortest path in graph G from $\bot$ to $\top$ is computed. We remark that this approach automatically finds a *deadlock-free path*. Indeed, if a path in G leads to a deadlock point, no bow goes from it; and a shortest path from $\bot$ to $\top$ is, above all, a path from $\bot$ to $\top$, and hence contains no deadlock.

The method presented here uses the geometry of the model for the construction. It is worth emphasizing that this method does not depend on the coordinates $c(e)$, in the sense that it can be used for the function c defined in the untimed, because we use the *structure* of the geometry of the forbidden boxes and not the distances in the embedding. The distances become important in the second method where an exact scaling geometrization is required, as we shall show later.

Notice that is it possible, after finding a satisfying path from $\bot$ to $\top$, to actually *construct* a concrete (eager) string corresponding to this path. The construction operates bow by bow: for each bow $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ construct a wait-free string abstracted by it. For this, start from $\boldsymbol{\varepsilon}$ and follow an adjacent small step to an $\boldsymbol{\varepsilon}''$ which increases the least the duration, among those that have not ${\boldsymbol{\varepsilon}''}_i \sqsupset {\boldsymbol{\varepsilon}'}_i$ for some i, and iterate until $\boldsymbol{\varepsilon}'$ is reached.

`Computing the Critical Potential Exchange Points`

The CPEPs are points on the boundary the forbidden regions. In dimension 2, they are the bottom-right and top-left points of the forbidden regions. In

dimension 3, they are all points on some edges of the boundary. In Figure 5.1 (right), the CPEPs are indicated with the circled addition symbols.

We compute the forbidden regions from the timed PV program and use the geometric characterization of the CPEPs, described in the previous section, to identify them.

Computing the bows of the abstraction graph

A simple method to determine whether an arc $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ is a bow is to determine the tightened length of the arc by enumerating all the strings from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$ and check the condition of Definition 13. However, this method is clearly very expensive, and to remedy this we will exploit some properties of the geometrization. We propose a method to find arcs that are necessarily bows, using a decomposition of forbidden-point-free regions. Using this approach we may not find a shortest schedule but we can find a good schedule.

Definition 15. *A* `decomposition` *of an orthogonal polyhedron P is a set $\mathcal{D}_P = \{B_1, \ldots, B_k\}$ where each B_i ($i \in \{1, \ldots, k\}$) is a full-dimensional box such that the following conditions are satisfied:*

1. *For all $i \in \{1, \ldots, k\}$ the vertices of B_i are grid points.*
2. *$P = \bigcup_{i \in \{1,\ldots,k\}} B_i$.*
3. *For all $i, j \in \{1, \ldots, k\}$, $i \neq j$, the boxes B_i and B_j are non-overlapping, that is their interiors do not intersect with each other.*

Note that the vertices of the boxes in a decomposition are not necessarily CPEPs. If all the vertices of a box are grid points then it is called a *grid box*. Additionally, if a grid box does not contain any other grid boxes, then it is called *elementary box*. We distinguish two types of decompositions: given a decomposition $\mathcal{D}_P = \{B_1, \ldots, B_k\}$, $\mathcal{D}_P$ is called *elementary* if all B_i are elementary boxes; $\mathcal{D}_P$ is called *compact* if there exists no pair of B_i and B_j with $i \neq j$ such that $B_i \cup B_j$ is a grid box. Intuitively, in a elementary decomposition none of its boxes can be split into smaller grid boxes, and in a compact decomposition no pair of its boxes forms a grid box. Note that there exists a unique elementary decomposition of a given orthogonal polyhedron, however there may be many different compact decompositions.

We use decompositions to construct the abstraction graph G. Let $\mathcal{D}_{P_A}$ be a decomposition of the allowed polyhedron P_A. We first recall the observation we use to reduce the complexity of the search for bows: a line segment connecting two vertices of a box $B_i \in \mathcal{D}_{P_A}$ corresponds to a bow. It is however clear that even when $\mathcal{D}_{P_A}$ is the elementary decomposition, the set of all such edges does

not allow to cover all possible bows since two vertices of two different boxes might also form a bow. However, if our goal is to find a path with the shortest duration it is not necessary to construct the whole graph G but we need to include all the CPEPs and bows that form such a path. It can be proved that there exists a decomposition such that the vertices of its boxes are enough to discover a shortest path. We call such a decomposition an *effective decomposition*, and it is of great interest to find such a decomposition, which is our ongoing work.

The essential idea of our current method for computing a compact decomposition of orthogonal polyhedra is as follows. From a given starting box we try to merge it with other elementary boxes, along one or more axes, so as to maximize the volume of the resulting box. To do so, we make use of the efficient algorithms for Boolean operations and membership testing developed based on a compact and canonical representation of such polyhedra (see [27]). In some cases the criterion of maximizing the volume of merged boxes may not be the best one with respect to including a shortest path in the graph. Alternative criteria are merging as many as possible boxes along a fixed axis. Intuitively, a shortest path tends to approach the diagonal between the bottom left and top right corners of the box B while avoiding the forbidden regions; hence, we can combine different merging criteria depending on the relative position to the forbidden regions.

Experimental results

We demonstrate in this section the effectiveness of the method. We implemented the above described method in a prototype tool. To compute the forbidden regions we use a program written in the language Maude [34]. The execution time for this computation is negligible. The program for the decomposition into allowed boxes, the construction of the abstraction graph from them, and the computation of a shortest path from $\bot$ to $\top$ in the graph is written in C++. The decomposition is rather fast, and most of the execution time for this program is spent in the construction of the graph from the allowed boxes, due to the number of vertices we use, as we explain in the following. We present in Figure 5.4 some experiments with this program.

We first tested with a set of the philosophers problem, in various dimensions. In one example, there are N forks, one per philosopher, and a thinking room which can take only $N-1$ philosophers. Then, we take the same example, but with a small thinking room ("s.th.-r") which can contain only $\lfloor N/2 \rfloor$ philosophers. We also tested the method with the enriched version of the philosophers problem ("enr. phil."), whose geometry is shown in Figure 5.2 (right). Program "enr. phil. 4D" is when a fourth philosopher is added alongside the three philosophers of the latter program. Program "more enr. phil." is when still more actions are added to the threads of "enr. phil.". We also experimented with the program of

program	dim	#states	#forbid	#allowed	#nodes	#edges	t (sec.)
3 phil.	3	512	4	35	151	773	0.58
4 phil.	4	4096	5	107	743	7369	17.38
5 phil.	5	32768	6	323	3632	67932	571.12
6 phil.	6	262144	7	971	na	na	na
3 phil. s.th.-r.	3	512	6	59	227	1271	1.50
4 phil. s.th.-r.	4	4096	8	199	1147	13141	60.24
5 phil. s.th.-r.	5	32768	15	1092	na	na	na
6 phil. s.th.-r.	6	262144	21	3600	na	na	na
enr. phil.	3	7488	26	390	1468	7942	51.01
enr. phil. 4D	4	119808	44	5447	na	na	na
more enr. phil.	3	29568	137	1165	4616	30184	461.18
3 phil. 2 procs	3	1728	12	78	352	2358	2.56

Figure 5.4: Results on some timed philosophers examples.

Section 5.5 ("3 phil. 2 procs"), whose geometry is shown in Figure 5.3. In the table, "na" stands for "not available" – the computation was not finishing in less than 10 minutes. We used a PC with a 2.40 GHz Xeon processor, 1 Go of memory and 2 Go of swap. One can observe that the number of allowed boxes is very reasonable compared with the number of states. The number of nodes reflects the fact that in our current prototype we add in the graph some of the vertices of the allowed boxes which are not CPEPs, to compensate for the fact that we do not currently include inter-allowed-box bows: thus we can find paths whose length approximate (conservatively) the weight of inter-box bows. The advantage of this approach is that any decomposition can serve to find a relatively good schedule; its inconvenient is that the number of considered vertices for a box is of order 2^N, so that the number of threads is the main obstacle in our current implementation.

In the case of the enriched 3 philosophers program of Section 5.4.2 (whose image is shown in Figure 5.2 (left)) the durations of the threads are 24, 25 and 20 respectively and the found schedule has duration 39, which is good. In the case of the enriched version of Figure 5.2 (right), the threads have respective durations 83, 94, and 95, and the found schedule has duration 160, which is also good in view of the many forbidden regions which bar the direct way.

5.8 Scheduling using a continuous geometric property

In this section we describe the second method to compute the abstraction graph and short schedules. This method is based on a property of the exact scal-

ing geometrization. It is important to note that, unlike the first method using decomposition, this method is applicable only for the programs which do not involve any zero-duration tasks. However, as the experimental results indicate, this method is more time efficient than the first method.

5.8.1 Continuous geometric property

The following theorem states an important property of bows in relation with the Euclidean intersection in the exact scaling geometrization. This property enables us to efficiently determine whether an arc is a bow.

Theorem 13. *Let $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ be an arc from $\mathcal{A}$. If $\overline{\langle \epsilon, \epsilon' \rangle} \cap P_F = \emptyset$ where P_F is the forbidden region, then $\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle$ is a bow.*

As mentioned earlier, the tightened length $d(\langle \boldsymbol{\varepsilon}', \boldsymbol{\varepsilon} \rangle)$ cannot be smaller than $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$. This means that to prove the theorem, it suffices to find a concrete feasible string from $\boldsymbol{\varepsilon}$ to $\boldsymbol{\varepsilon}'$ whose duration is exactly $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$. By definition of the duration of a string, this also means finding a feasible timed execution with the required duration. The idea of the proof is to construct such a timed execution that we call a *witness timed execution*. This is done by a clipping procedure explained in the following.

5.8.2 Constructing a witness timed execution

`Clipping` Let $\{\mathbf{x}^j\}_{1 \leq j \leq m}$ be the sequence of all intersecting points of the directed line segment $\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ with the grid planes. A *grid plane* is a hyper-plane which is parallel to one of the axes and contains at least one grid point. We denote this by $\{\mathbf{x}^j\}_{1 \leq j \leq m} = clip_{\mathcal{G}}(\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle})$ and call this sequence the *clipping* of $\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ on the grid $\mathcal{G}$.

An example of clipping is shown in Figure 5.5 where $\mathbf{x}^1, \mathbf{x}^2, \ldots, \mathbf{x}^7$ is a sequence of intersecting points. We derive a timed execution from this clipping by mapping each intersecting point to a timed state. Note the time value of a timed state is the absolute time lapse from the beginning of the execution, and in order to determine this absolute time lapse one needs to consider the time lapses relative to the occurrence of the events. This is captured by the notion of `relatively-timed events`.

`Relatively-timed Events` We first remark that a grid point can be directly mapped back to a state in $\mathcal{E}$; however, an intersecting point in the clipping is not necessarily a grid point. Note that a state in $\mathcal{E}$ corresponds to the moment where all the threads take a resource action. An intersecting point which is not a grid

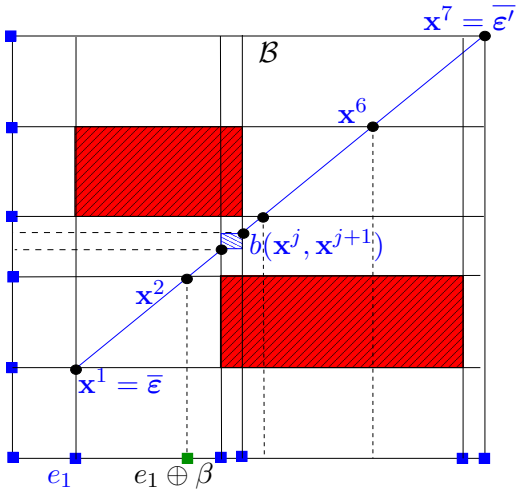


Figure 5.5: Geometric realization

point indeed corresponds to a situation where not all the threads simultaneously perform a resource action.

Given $e \in E_i$ and a real number $\beta \in [0, d(e))$ where $d(e)$ is the duration of task e, $e \oplus \beta$ denotes the relatively-timed event at which *at least* β time units have elapsed since the occurrence of the event e (or since task e is started); β is called the relative time lapse of $e \oplus \beta$. Intuitively, regarding thread E_i, $e \oplus \beta$ is a fictious event because its occurrence is not associated with any resource actions by E_i; however, as we shall see later, it is used to indicate a time point at which at least one or more other threads perform a resource action. Then, the set of relatively-timed events of thread E_i is: $\Upsilon_i = \{e \oplus \beta \mid e \in |E_i| \ \wedge \ \beta \in [0, d(e))\}$ where $|E_i|$ is the set of events of E_i. For each relatively-timed event $\rho = e \oplus \beta \in \Upsilon_i$, $event(\rho)$ gives the associated event e.

We associate with each relatively-timed event $e \oplus \beta$ an ordinate $c(e \oplus \beta) = c(e) + \beta$. Hence, an event $e \in E_i$ can be indeed written as a relatively-timed event of the form $e \oplus 0$. The definition of order on the relatively-timed events in Υ_i can be defined as: $e \sqsubseteq_{\Upsilon_i} e'$ iff $c(e) \leq c(e')$.

Remark 2. *With respect to a thread, the difference between the relative time lapse β of a relatively-timed event $e \oplus \beta$ and the time component t of a timed state $(\boldsymbol{\varepsilon}, t)$ (defined in Section ??) is that the latter is an absolute time (i.e. the time lapse from the beginning of the execution), while the former is a relative time (i.e. the time lapse from the occurrence of the last event, which is e).*

`Relatively-timed states` A vector of relatively-timed events $\boldsymbol{\upsilon} = (\boldsymbol{\upsilon}_1, \ldots, \boldsymbol{\upsilon}_N)$ where $\boldsymbol{\upsilon}_i \in \Upsilon_i$ is called a `relatively-timed state`. We denote $events(\boldsymbol{\upsilon}) =$

$(event(\boldsymbol{\upsilon}_1), \ldots, event(\boldsymbol{\upsilon}_N))$. The order $\preccurlyeq$ on relatively-timed states is defined componentwise, namely $\boldsymbol{\upsilon} \preccurlyeq \boldsymbol{\upsilon}'$ iff $\forall i \in \{1, \ldots, N\} : \ \boldsymbol{\upsilon}_i \sqsubseteq_{\Upsilon_i} \boldsymbol{\upsilon}'_i$, or equivalently $\forall i \in \{1, \ldots, N\} : \ c(\boldsymbol{\upsilon}_i) \leq c(\boldsymbol{\upsilon}'_i)$.

Let us explain the intuitive meaning of relatively-timed states. At relatively-timed state $\boldsymbol{\upsilon} = (\boldsymbol{\varepsilon}_1 \oplus \beta_1, \ldots, \boldsymbol{\varepsilon}_N \oplus \beta_N)$, if the relative time lapse $\beta_i = 0$, then thread E_i is performing the action associated with $event(\boldsymbol{\upsilon}_i)$; if $\beta_i > 0$, thread E_i is performing no resource action and, in addition, at least β_i time units have elapsed since the occurrence of $\boldsymbol{\varepsilon}_i$. By "performing no resource action" we mean that the thread does not take or release a resource, but it might continue the current task if this task is not yet finished.

Given two relatively-timed states $\boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \in \Upsilon$ such that $\boldsymbol{\upsilon} \preccurlyeq \boldsymbol{\upsilon}'$, then $\langle \boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \rangle$ is called a `small timed step` if $events(\boldsymbol{\upsilon}) = events(\boldsymbol{\upsilon}')$ or $\langle events(\boldsymbol{\upsilon}), events(\boldsymbol{\upsilon}') \rangle$ is a small step. In other words, $\boldsymbol{\upsilon}$ and $\boldsymbol{\upsilon}'$ may have the same associated events but differ in the relative time lapse vector.

Definition 16. *If $\langle \boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \rangle$ is a small timed step, the time lapse between $\boldsymbol{\upsilon}$ and $\boldsymbol{\upsilon}'$ is defined as: $\Delta(\boldsymbol{\upsilon}, \boldsymbol{\upsilon}') = max_{i \in \{1,\ldots,N\}}\{c(\boldsymbol{\upsilon}'_i) - c(\boldsymbol{\upsilon}_i)\}$.*

The meaning of a small timed step $\langle \boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \rangle$ is that following the arc $\langle \boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \rangle$ involves letting each thread E_i start the task $\boldsymbol{\upsilon}_i$ and run for exactly $\Delta(\boldsymbol{\upsilon}, \boldsymbol{\upsilon}')$ time. If there exists a thread E_i such that $c(\boldsymbol{\upsilon}'_i) - c(\boldsymbol{\upsilon}_i) < \Delta(\boldsymbol{\upsilon}, \boldsymbol{\upsilon}')$, we say that when taking the small step $\langle \boldsymbol{\upsilon}, \boldsymbol{\upsilon}' \rangle$ the thread E_i 'has to wait' because the time lapse required for the thread E_i to reach $event(\boldsymbol{\upsilon}'_i)$ from $event(\boldsymbol{\upsilon}_i)$ is smaller than $\Delta(\boldsymbol{\upsilon}, \boldsymbol{\upsilon}')$.

`Mapping points to relatively-timed states` Given a real number $y \in [0, c(\top_i)]$, let e be the event in thread E_i and e' is its direct successor such that $c(e) \leq y$ and $c(e') > y$. Such an ordinate $c(e)$ is denoted by $\lfloor y \rfloor$. Then, we define $\mathbb{N}_i(y) = e \oplus \beta$ where $\beta = y - \lfloor y \rfloor$.

As an example, in Figure 5.5, the first coordinate of the point x^2 is mapped to $\mathbb{N}_i(x_1^2) = e_1 \oplus \beta$.

Definition 17. *1. Given a point $\mathbf{x} = (x_1, \ldots, x_N) \in \mathcal{B}$, the map $\mathbb{N}$ of points to relatively-timed states is defined as: $\mathbb{N}(\mathbf{x}) = (\mathbb{N}_1(x_1), \ldots, \mathbb{N}_N(x_N))$.*

2. Given a sequence of points $\{\mathbf{x}^j\}_{1 \leq j \leq m}$, $\mathbb{N}(\{\mathbf{x}^j\}_{1 \leq j \leq m}) = \{\boldsymbol{\upsilon}^j\}_{1 \leq j \leq m}$ where $\boldsymbol{\upsilon}^j = \mathbb{N}(\mathbf{x}^j)$ for all j.

`Witness timed execution construction` Using the map $\mathbb{N}$, from the clipping $clip_{\mathcal{G}}(\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}) = \{\mathbf{x}^j\}_{1 \leq j \leq m}$ we construct the following sequence of relatively-timed states: $\phi = \boldsymbol{\upsilon}^1, \ldots, \boldsymbol{\upsilon}^m = \mathbb{N}(\mathbf{x}^1), \ldots, \mathbb{N}(\mathbf{x}^m)$. It is not hard to see that

each arc $\langle \boldsymbol{v}^i, \boldsymbol{v}^{i+1} \rangle$ is a small timed step and $event(\boldsymbol{v}^1), \ldots, event(\boldsymbol{v}^m)$ is a string. Then, from ϕ we construct a timed execution as follows:

$$\gamma = (event(\boldsymbol{v}^1), t^1), \ldots, (event(\boldsymbol{v}^m), t^m) \tag{5.3}$$

such that $t^1 = 0$ and for $j > 1 : t^j = \sum_{k=2,\ldots,j} \Delta(\boldsymbol{v}^{k-1}, \boldsymbol{v}^k)$. It is easy to verify that the timed execution γ is consistent.

■

To summarize, the construction of a witness timed execution for a bow $\frac{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ consists of three steps. In the first step, the clipping of the geometrization $\frac{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ gives a sequence of intersecting points, each of which corresponds to a moment where at least one thread performs a resource action. In the second step, the intersecting points are mapped to a sequence of relatively-timed states that specify the time lapses necessary to evolve from one state to another in this sequence. These time lapses indeed represent the local time constraints of each thread. In the last step, we combine all the local time constraints to derive the global time constraints in the timed execution γ.

5.8.3 Proof of Theorem 13

To prove that γ is a witness timed execution, we need to prove that: γ is feasible and its duration is indeed $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$.

We begin by proving the first part, that is, γ does not induce any resource conflicts. Due to space limitation, we present only the main idea of the proof: if a point $\mathbf{x}$ is non-forbidden, then the state $events(\mathbb{N}(\mathbf{x}))$ is non-forbidden (see [38] for a detailed proof). The intuitive meaning of this is that with respect to resource usage, a relatively-timed event $e \oplus \beta \in \Upsilon_i$ with $\beta \in (0, d(e))$ is equivalent to $e \in E_i$, since during the time interval between the occurrences of $e \oplus 0$ and $e \oplus \beta$ no resources have been taken or released by thread E_i.

■

We proceed to prove that the duration of γ is $\|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\|$. The following intermediate result is a direct consequence of the definition of the duration of a timed execution.

Lemma 13. *The duration of the timed execution defined in (5.3) is*

$$d(\gamma) = t^m - t^1 = \sum_{1 \le j \le m-1} \Delta(\boldsymbol{v}^j, \boldsymbol{v}^{j+1}).$$

Geometrically speaking, Definition 16 implies that $\Delta(\boldsymbol{v}^j, \boldsymbol{v}^{j+1})$ is equal to the length of the longest side of the box that has $\boldsymbol{x}^j$ as its bottom left vertex and $\boldsymbol{x}^{j+1}$

as its top right vertex, denoted by $b(\boldsymbol{x}^j, \boldsymbol{x}^{j+1})$ (see Figure 5.5 for an example). Let k be the dimension corresponding to the longest side of the box $b(\boldsymbol{x}^1, \boldsymbol{x}^m)$. Note that this box has $\overline{\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle}$ as diagonal. It is easy to see that k is also the dimension corresponding to the longest side of each box $b(\boldsymbol{x}^j, \boldsymbol{x}^{j+1})$. Combining this with Lemma 13, we have

$$\begin{aligned} d(\gamma) &= \sum_{1 \le j < m} \Delta(\boldsymbol{v}^j, \boldsymbol{v}^{j+1}) \\ &= \sum_{1 \le j < m} c(\boldsymbol{v}_k^{j+1}) - c(\boldsymbol{v}_k^j) \\ &= c(\boldsymbol{v}_k^m) - c(\boldsymbol{v}_k^1) = c(\boldsymbol{\varepsilon}'_k) - c(\boldsymbol{\varepsilon}_k) = \|\langle \boldsymbol{\varepsilon}, \boldsymbol{\varepsilon}' \rangle\| \end{aligned}$$

The proof of Theorem 13 is now complete.

■

Remark 3. *The proof of the theorem also provides a method for concretizing strings. Hence, after finding a shortest path in the abstraction graph, one can use the construction in the proof to define a concrete shortest timed schedule.*

`Intersection test` Before continuing we briefly discuss how we implemented the intersection test. In [57] the forbidden region P_F is represented as a non-convex orthogonal polyhedron [27]. This representation has the advantage of being compact since one needs to keep only one polyhedron. However, due to its complexity (depending on the number of vertices and faces that is exponential in dimension), the test of intersection between a line segment and P_F may be expensive in high dimensions. We therefore represent the forbidden region as a list of the forbidden boxes (each box corresponds to the constraints involving the limited number of accesses to a resource). Then, we test the intersection between the line segment with each box separately using an extension of ray tracing techniques to general dimensions. Since a box can be represented by $2N$ linear constraints, the complexity of this test is polynomial in dimension N.

5.8.4 Finding a good schedule via path planning

`Randomized Search`

Using the bow condition in Theorem 13, we can construct the abstraction graph and then search for a shortest path of the graph. The main problem with this approach is that the number of critical exchange states, which is much smaller than the number of all the states, still grows exponentially with the dimension. We thus propose a non-exhautive solution that uses a randomized search, inspired

by the RRT (Rapidly Explored Random Tree), which is one of the successful path planning methods in robotics (see [81] for a survey on the RRT method). Indeed, in the geometrization framework, given two points corresponding to two critical exchange states, by Theorem 13, if the line segment connecting these two points does not intersect with the forbidden region, then a feasible schedule with no unnecessary wait between two states exists (and we can compute it). The problem of constructing the abstraction graph is thus similar to a path planning problem, namely computing a collision-free path between a start point and a goal point in an environment with known obstacles. The constraints on the solution path in the path planning problem arise from the geometry of the obstacles and in our scheduling problem from the geometry of the forbidden region. While the RRT approach, to construct the paths, considers all the points in the obstable-free space, using Theorem 12 only the critical exchange states need to be considered. In addition, the paths we are interested in should satisfy the time progress condition; therefore, the resulting path planning problem is indeed a simple motion planning problem[3] where the robot's motion is governed by the constant derivative dynamics of a clock. This dynamics is easily handled by considering the arcs (which by definition satisfy the time progress condition). In general, the path and motion planning problems are hard, for example the problem of finding a shortest path in 3 dimensions is known to be NP-hard [86]. It is however important to note the simplicity of the obstacles in our problem: they are in fact axis-aligned boxes. Additionally, since our practical goal is to quickly find a good schedule, the good coverage properties of the RRT approach allows achieving a good trade-off between the computation time and the quality of the results.

The method we propose is summarized in Algorithm 9. Essentially, we randomize the selection of the critical exchange states. In order to avoid enumerating *all* such states, the randomized selection is done as follows. Let B_F be the set of all forbidden boxes, hence the forbidden region can be written as $P_F = \bigcup B_F$. We first randomly choose a box in B_F and then randomly choose a vertex of B_F. The procedure is repeated until the sampled vertex is a critical exchange state. The test of critical exchange states is done using the geometric charaterization of these points, mentioned in the previous section.

When a new critical exchange point x_g is selected, we call it a (current) *goal* point. We then find the the graph a nearest neighbor x_n in the max-distance such that x_g and x_n is an arc (due to the time progress condition). The computation of the function FEASIBLEBEST is as follows. It checks whether the line segment from x_n to x_g intersects with the forbidden region. If this intersection is empty, $x = x_g$ and a new edge from x_n to x_g is added in the graph, otherwise it tries to grow the graph from x_n towards the goal point x_g as far as possible, which

[3]In a path planning problem, the dynamics of robots are not considered.

Algorithm 9 Randomized search

```
C = {⊥, ⊤}, k = 0
repeat
  box = random(B_F)
  x_g = random(VERTICES(box))
  if (x_g ∉ C ∧ x_g is a critical exchange point) then
    x_n = NEIGHBOR(G, x_g)
    x = FEASIBLEBEST(x_n, x_g)
    if (x ≠ x_n) then
      NEWEDGE(G, x_n, x)
    end if
    k++
  end if
  π = SHORTESTPATH(G, ⊥, ⊤)
until (k = K_max)
```

results in $x \neq x_g$. When the number of nodes reaches K_{max} which is a user-defined parameter, the algorithm searches for a shortest path in the graph and stops if there is no request to proceed by the user.

An important ingredient of Algorithm 9 is the search for a nearest neighbor in the graph G. To do so, we additionally store the coordinates of points in a kd-tree [52] while the bows are still stored as the edges of the graph G. Due to the use of the max-distance, the operations on the kd-tree we construct is slightly different than those on classic kd-trees. Nonetheless, due to space limitation we do not describe these computations, which can be found in [38]. When running the above algorithm, the graph grows towards the end point $\top$; it is also possible to simultaneously grow the graph towards both $\top$ and $\bot$.

In addition, we can prove that when every goal point has a strictly positive probability of being sampled, then the probability that the algorithm discovers a given schedule is always strictly positive. This property is called 'complete resolution' in the context of RRTs (see for example [30]). Moreover, it is possible to biase the exploration using the intuitions provided by the geometrization. Indeed, the max-distance and the Euclidian distance are closely related with respect to the definition of duration. For example, a schedule that is close in the Euclidian distance to the diagonal of the bounding box (i.e. connecting $\bot$ and $\top$) is likely to be a short schedule. Therefore, one can use the Euclidian distance as a measure to define a non-uniform sampling of the goal points. More precisely, we can define a heuristics which favors the sampling of the critical exchange states that are close to the diagonal of the bounding box in the Euclidian distance.

`Experimental results`

We have implemented the above algorithm for randomized search together with a possibility of biased explorations. The experimental results obtained using the prototype tool on a number of examples are shown in the tables of Figure 5.7 and Figure 5.8. We also include in Figure 5.6 an illustrative picture of the forbidden region and the computed schedule (in white line) of a 3-dimensional example. The first set of examples contains a number of timed versions of the Dining Philosophers problem in various dimensions, which we call the timed N-philosophers problems. The second set contains some well-known job-shop scheduling (JSS) benchmarks. The durations of the schedules obtained using our prototype tool are shown in the column "Duration" of the tables. The goal of this experimentation is to evaluate the scalability and the precision of our geometric approach.

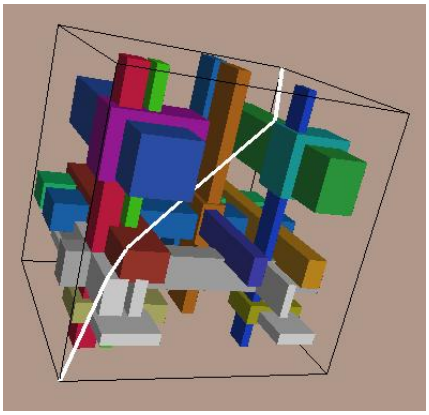


Figure 5.6: Schedule for a 3D example

Since there are variants of job-shop problems, we first briefly describe the problems we solved. In these problems, each job is a sequence of operations, one on a machine, and the serving capacity of each machine is 1. Indeed, the machines can be modeled by resources and the jobs by threads. Note that the constraints in job-shop scheduling are rather specific. When a machine is needed by two different jobs E_i and E_j, their simultaneous access to this machine corresponds to the following forbidden box: $[c(\bot_1), c(\top_1)] \times [c(\boldsymbol{\varepsilon}_i), c(\boldsymbol{\varepsilon}_{i+1})] \times [c(\boldsymbol{\varepsilon}_j), c(\boldsymbol{\varepsilon}_{j+1})] \ldots [c(\bot_N), c(\top_N)]$ where $\boldsymbol{\varepsilon}_i$ and $\boldsymbol{\varepsilon}_j$ are the events of taking the resource in question by thread E_i and thread E_j, and $\boldsymbol{\varepsilon}_{i+1}$ and $\boldsymbol{\varepsilon}_{j+1}$ are the events of releasing this resource. This box covers the whole range of the bounding box on the dimensions of all other threads E_k with $k \neq i$ and $k \neq j$. On the other hand, the JSS problems are by definition deadlock-free. Note that a general timed PV program allows nested resource actions and thus are richer than these JSS models. However, to test the performance of our approach, we did not try to exploit this particularity and treated the job-shop scheduling problems as if they

program	Dim	#forb. boxes	Duration	CPU time (s)
20 phil.	20	20	100	64
50 phil.	50	50	304	248
80 phil.	80	80	490	625
100 phil.	100	100	608	921

Figure 5.7: Computation results for some timed N-philosophers problems

were problems with more complex types of constraints.

Concerning the timed N-philosophers problems, we do not know their optima, but the computed solutions are clearly non-trivial and appear good in comparison with the worst-case upper bound estimation. On the JSS problems, this method found fairly good schedules in reasonable CPU time. On some of these JSS problems the optima were found. In operation research, there exist numerous methods in operation research specific for JSS (see for example [89]) and recently verification techniques for timed automata were also used to solve the JSS problems. These timed automata based methods employ sophisticated search-order strategies, such as branch-and-bound or using estimates of remaining costs [20, 2]. Our experimental results on JSS problems are not as good as the results obtained by these methods, but they are still reasonably comparable.

In summary, we observe that this method is efficient for quickly finding a reasonable solution of large problems. On the other hand, the number of forbidden boxes is a leading factor for complexity (since it not only determines the number of critical exchange states but also the number of intersection tests to perform). In a JJS problem with M machines and J jobs, the number of forbidden boxes is equal to $\frac{1}{2}MJ(J+1)$ and thus grows quadraticaly with the number of jobs (i.e. the dimension). Therefore, this method is suitable for the problems which could be in high dimension but with a reasonable number of forbidden boxes.

5.9 Related works

Timed PV diagrams We now discuss the relationship between our model and some other versions of timed PV diagrams.

- The work [47], which presents a timed version of PV programs and diagrams, attempts to model multiple clocks, as in timed automata [10]. In the approach of [47], time is modeled as an additional dimension – one per clock. Thus, in the case of one clock and three threads, a 4-dimensional space is studied. In this chapter we consider each thread dimension as as a "local time dimension" and define the synchronization of these local times.

Prog	#j, #m	#forb. boxes	Duration	Known optimum	CPU time (s)
ft06	6, 6	90	56	55	66
ft10	10, 10	450	992	930	318
abz6	10, 5	225	1142	943	851
la01	10, 5	225	666	666	646
la05	10, 5	225	596	593	87
la16	10, 10	450	1047	945	247
la19	10, 10	450	1050	842	42
la20	10, 10	450	989	902	125
la24	15, 10	450	1048	935	269
abz9	20, 15	2850	820	679	310

Figure 5.8: Results for some JSS problems

- The work [61] exploits the dimension of each process as a time dimension. In this aspect, this work is close to ours. However there are important differences. First, the definitions in [61] are given in a continuous setting, and therefore topological spaces are considered, such that the duration of a schedule is described with an integral. In our work, it is possible to stay in the discrete domain, and the definition of the duration of a schedule is given by an algorithm on a discrete structure. Also, due to the fact that the definitions in [61] are tied to geometry implies, zero delays between two consecutive actions in a process (such as, two successive locks, which often happens in programs that share resources) are not possible since the two actions cannot be distinguished in the geometry. Note that our method using decomposition can handle zero delays. This is of particular interest if one considers that the practical delay, on most architectures, between two consecutive locks, is too small to be modeled as a non-zero value.

Timed automata A large class of real-time systems can be adequately modeled with *timed automata* [10], and the problem of scheduling using timed automata has been studied in a number of publications [12, 1, 77, 20, 6, 73, 93, 2] and more general optimality criteria (other than execution time) are also considered in some of these work (for example [12, 77, 20]). It is easy to see that timed automata are more expressive than timed PV programs since the former allow to describe more complex synchronization mechanisms. In addition, a timed PV program can be directly rewritten as a product of timed automata. Each automaton corresponds to a thread and its locations represent the events in our model. Its transitions representing the time constraints have the guards of the form $x > d(e)$ (where x is a clock variable) and clock resets. Thus, using timed automata, one could address the scheduling problem for more complex real-time

systems.

Naturally, the geometry resulting from the time constraints in timed automata is more complex than that in timed PV programs. Indeed, in a timed automaton each time constraint is represented by a half-space that could have a slope following the derivatives of the clocks, while in a timed PV program, the half-spaces are all axis-parallel. As mentioned earlier, the reason for this is that in a timed PV program each thread is described separately on one dimension, which can be thought of as a way of 'desynchronizing' them, and then when analyzing the global behavior, the 'synchronization' of local time constraints is handled by using the max-distance. Hence, the geometrization of the product of the threads is, on one hand, very easy to construct, and on the other hand provides a lot of useful insight. The computation is performed on boxes, a geometric object simpler than zones in timed automata. Moreover, the geometry of a PV diagram permits modular combination of its discrete properties (such as to identify special points that contribute to the optimal schedules) and continuous properties (such as to test feasibility of some long direct paths). However, it should be noted that for special cases, such as JSS, one can derive efficient heuristics without manipulating zones (see for example [2]). Finally, we remark that from a semantics point of view, the idea of 'desynchronizing' the threads in our approach is close to the local semantics for networks of timed automata given in [23]. In this work, this semantics is used to address the problem of checking reachability properties using partial order reduction techniques.

5.10 Conclusion and future work

To summarize, we defined a timed version of PV programs and diagrams which can be used to model a large class of multithreaded programs sharing resources. We also introduced the notion of the worst-case response time of a schedule of such programs. This framework was then used to find efficient schedules for multithreaded programs.

In particular, to tackle the complexity problem, we define an abstraction of the shortest schedules and we show how to exploit the geometry of PV diagrams to construct this abstraction and compute efficient schedules. This work demonstrates an interesting interplay between discrete and continuous properties of concurrent real-time programs. Indeed our framework is based on a combination of techniques from different domains: concurrent processes, geometric models, and motion planning. The originality of our work is the way to exploit discrete and continuous properties of the model in order to transform the scheduling problem to a set of subproblems: graph search, computational geometry and path planning, for which well-developed techniques are available. The paper also shows the computational advantages of PV programs. In fact, their geometry

is simple enough to benefit from efficient geometric computations on boxes. An experimental implementation allowed us to validate the scheduling methods we proposed and provided encouraging results. We intend to continue this work in various directions. One direction is to extend the model towards more complex specifications, such as those with deadlines and branching. Our future work will explore the following directions.

- When developing a real-time system one is often interested in the worst-case response time of the whole program, if it is part of a larger system, *for any schedule*. As a definition, this WCRT could be given as the duration of the eager schedule that has the longest duration. We conjecture that we could use abstraction graph G for computing the longest eager schedule by computing the longest path in a subgraph of G. Defining this subgraph is a topic of our future research.

- We are currently investigating the problem of adding *deadlines* in our model. This extension is not straightforward since the "symmetry" with the lower bounds to durations of tasks (the WCET) is not trivial. We also intend to examine the possibility of lifting to the timed case the existing studies on the geometry of loops [49] or branching in PV programs.

- Another direction is to focus on problems with particular geometry (such as the JSS problems). Indeed, it is possible to include optimization that exploits the special structure of the forbidden boxes in these problems. Studying properties of other geometrizations including non-exact-scaling ones is also an interesting theoretical problem to address.

Bibliography

[1] Y. Abdeddaim and O. Maler. Job-shop scheduling using timed automata. In *Proc. of the 13th Int. Conf. on Computer Aided Verification (CAV 2001), LNCS 2102*, pages 478–492. Springer, 2001.

[2] Yasmina Abdeddaim, Eugene Asarin, and Oded Maler. Scheduling with timed automata. *Theor. Comput. Sci.*, 354(2):272–300, 2006.

[3] V. Acary, O. Bonnefon, and P. Denoyelle. Automatic circuit equation formulation for nonsmooth electrical circuits. Technical report, BIPOP-INRIA, ANR VAL-AMS Report, Jan 2008.

[4] V. Acary and F. Pérignon. Siconos: A software platform for modeling, simulation, analysis and control of non smooth dynamical system. In *Proceedings of MATHMOD 2006, 5th Vienna Symposium on Mathematical Modelling*, Vienna, 2006. ARGESIM Verlag, Vienna, 2006 ISBN 3-901608-30-3.

[5] Thierry Jéron Ahmed Khoumsi and Hervé Marchand. Test cases generation for nondeterministic real-time systems. In *Formal Approaches to Software Testing*, 2004.

[6] K. Altisen, G. Gossler, and J. Sifakis. Scheduler modelling based on the controller synthesis paradigm. *Journal of Real-Time Systems, Special issue on control-theoretical approaches to real-time computing*, 23:55–84, 2002.

[7] R. Alur, C. Courcoubetis, N. Halbwachs, T. A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[8] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivan, C. Kumar, I. Lee, P. Mishra, G. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems, 2002.

[9] R. Alur, T. Dang, and F. Ivancic. Reachability analysis via predicate abstraction. In M. Greenstreet and C. Tomlin, editors, *Hybrid Systems: Computation and Control*, LNCS 2289. Springer-Verlag, 2002.

[10] R. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[11] R. Alur, T.A. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. *Proc. of the IEEE*, 2000.

[12] R. Alur, S. La Torre, and G. Pappas. Optimal paths in weighted timed automata. In *Proc. of Fourth Int. Workshop on Hybrid Systems: Computation and Control, LNCS 2034*, pages 49–62, 2001.

[13] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 20–31. Springer-Verlag, 2000.

[14] E. Asarin and T. Dang. Abstraction by projection. In R. Alur and G. Pappas, editors, *Hybrid Systems: Computation and Control*, LNCS 2993, pages 32–47. Springer-Verlag, 2004.

[15] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In Oded Maler and Amir Pnueli, editors, *Hybrid Systems: Computation and Control*, LNCS 2623, pages 20–35. Springer-Verlag, 2003.

[16] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica.*, 43(7):451–476, 2007.

[17] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Computer Aided Verification*, LNCS 2404, pages 365–370. Springer-Verlag, 2002.

[18] P. M. Aziz, H. V. Sorensen, and J. van der Spiegel. An overview of sigma-delta converters. *Signal Processing Magazine, IEEE*, 13(1):61–84, 1996.

[19] J. Beck and W. W. L. Chen. Irregularities of distribution. In *Acta Arithmetica*, UK, 1997. Cambridge University Press.

[20] G. Behrmann and A. Fehnker. Efficient guiding towards cost-optimality in UPPAAL. In *Proceedings of the 7th International Conference on Tools and Algorithms for the Construction and Analysis of Systems, LNCS 2031, Springer*, 2001.

[21] C. Belta, L. C. G. J. M. Habets, and V. Kumar. Control of multi-affine systems on rectangles with an application to gene transcription control. In *Proceedings of CDC*, 2003.

[22] C. Belta, J. Schug, T. Dang, V. Kumar, G.J. Pappas, H. Rubin, and P. Dunlap. Stability and reachability analysis of a hybrid model of luminescence in the marine bacterium *vibrio fisheri*. In *Proceedings of CDC*, 2001.

[23] Johan Bengtsson, Bengt Jonsson, Johan Lilius, and Wang Yi. Partial order reductions for timed systems. In *International Conference on Concurrency Theory*, pages 485–500, 1998.

[24] Saddek Bensalem, Marius Bozga, Moez Krichen, and Stavros Tripakis. Testing conformance of real-time applications by automatic generation of observers. *Electr. Notes Theor. Comput. Sci.*, 113:23–43, 2005.

[25] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *HSCC*, pages 142–156, 2004.

[26] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 73–88. Springer-Verlag, 2000.

[27] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In *Proc. of Hybrid Systems: Computation and Control (HSCC'99), LNCS 1569*, pages 46–60. Springer, March 1999.

[28] S. Boyd and S. Vandenberghe. *Convex optimization*. Cambridge University Press, 2004.

[29] M. Branicky, M. Curtiss, J. Levine, and S. Morgan. Sampling-based reachability algorithms for control and verification of complex systems. In *Thirteenth Yale Workshop on Adaptive and Learning Systems*, 2005.

[30] P. Cheng and S. M. LaValle. Resolution complete rapidly-exploring random trees. In *In Proc. IEEE Int'l Conference on Robotics and Automation*, pages 267–272, 2002.

[31] A. Chutinan and B.H. Krogh. Verification of polyhedral invariant hybrid automata using polygonal flow pipe approximations. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer-Verlag, 1999.

[32] Edmund M. Clarke, Ansgar Fehnker, Zhi Han, Bruce H. Krogh, Joël Ouaknine, Olaf Stursberg, and Michael Theobald. Abstraction and counterexample-guided refinement in model checking of hybrid systems. *Int. J. Found. Comput. Sci.*, 14(4):583–604, 2003.

[33] F. Clauss and I.Yu. Chupaeva. Application of symbolic approach to the bernstein expansion for program analysis and optimization. *Program. Comput. Softw.*, 30(3):164–172, 2004.

[34] M. Clavel, F. Durán, S. Eker, P. Lincoln, N. Martí-Oliet, J. Meseguer, and C. Talcott. *Maude 2.0 Manual*. SRI International, 2003.

[35] R. Cridlig and E. Goubault. Semantics and analysis of Linda-based languages. In *Proc. of WSA'93, LNCS 724*. Springer, 1993.

[36] T. Dang. Approximate reachability computation for polynomial systems. In *HSCC*, pages 138–152, 2006.

[37] T. Dang, A. Donze, and O. Maler. Verification of analog and mixed-signal circuits using hybrid systems techniques. In Alan J. Hu and Andrew K. Martin, editors, *FMCAD'04 - Formal Methods for Computer Aided Design*, LNCS 3312, pages 21–36. Springer-Verlag, 2004.

[38] T. Dang and Ph. Gerner. On scheduling using pv programs. Technical report, Verimag, IMAG, April 2006.

[39] T. Dang and O. Maler. Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 96–109. Springer-Verlag, 1998.

[40] T. Dang and T. Nahhal. Model-based testing of hybrid systems. Technical report, Verimag, IMAG, Nov 2007.

[41] T. Dang and D. Salinas. Computing set images of polynomias. Technical report, VERIMAG, June 2008.

[42] Thao Dang and Tarik Nahhal. Coverage-guided test generation for continuous and hybrid systems. *Formal Methods in System Design*, 34(2):183–213, 2009.

[43] E. W. Dijkstra. Co-operating sequential processes. In F. Genuys, editor, *Programming Languages*, pages 43–110. Academic Press, New York, 1968.

[44] David Dobkin and David Eppstein. Computing the discrepancy. In *SCG '93: Proceedings of the ninth annual symposium on Computational geometry*, pages 47–52, New York, NY, USA, 1993. ACM Press.

[45] C. Lubich E. Hairer and M. Roche. The numerical solution of differential-algebraic systems by runge kutta methods. In *Lecture Notes in Mathematics 1409*. Springer-Verlag, 1989.

[46] J. Esposito, J. W. Kim, and V. Kumar. Adaptive RRTs for validating hybrid robotic control systems. In *Proceedings Workshop on Algorithmic Foundations of Robotics*, Zeist, The Netherlands, July 2004.

[47] U. Fahrenberg. The geometry of timed PV programs. In *Electronic Notes in Theoretical Computer Science*, volume 81. Elsevier, 2003.

[48] L. Fajstrup, E. Goubault, and M. Raussen. Detecting deadlocks in concurrent systems. In *Proc. CONCUR'98*, pages 332–347, 1998.

[49] L. Fajstrup and S. Sokolowski. Infinitely running concurrent processes with loops from a geometric viewpoint. In *Electronic Notes in Theoretical Computer Science*, volume 39. Elsevier, 2001.

[50] Henri Faure. Discrepance de suites associees à un système de numeration., 1978.

[51] I.A. Fotiou, P. Rostalski, P.A. Parrilo, and M. Morari. Parametric optimization and optimal control using algebraic geometriy methods. *International Journal of Control*, 79(11):1340–1358, 2006.

[52] Volker Gaede and Oliver Günther. Multidimensional access methods. *ACM Computing Surveys*, 30(2):170–231, June 1998.

[53] J. Garloff. Application of bernstein expansion to the solution of control problems. In *University of Girona*, pages 421–430, 1999.

[54] J. Garloff, C. Jansson, and A.P. Smith. Lower bound functions for polynomials. *Journal of Computational and Applied Mathematics*, 157:207–225, 2003.

[55] J. Garloff and A.P. Smith. An improved method for the computation of affine lower bound functions for polynomials. In C. A. Floudas and P. M. Pardalos, editors, *Frontiers in Global Optimization*, Series Nonconvex Optimization and Its Applications, pages 135–144. Kluwer Academic Publ.,Boston, Dordrecht, New York, London, 2004.

[56] J. Garloff and A.P. Smith. A comparison of methods for the computation of affine lower bound functions for polynomials. In C. Jermann, A. Neumaier, and D. Sam, editors, *Global Optimization and Constraint Satisfaction*, LNCS, pages 71–85. Springer, 2005.

[57] P. Gerner and T. Dang. Computing schedules for multithreaded real-time programs using geometry. In Y. Lakhnech and S. Yovine, editors, *Joint International Conferences on Formal Modelling and Analysis of Timed Systems FORMAT and Formal Techniques in Real-Time and Fault-Tolerant Systems FTRTFT*, LNCS 3253, pages 325–342. Springer-Verlag, 2004.

[58] A. Girard. Approximate solutions of ODEs using piecewise linear vector fields. In *Proc. CASC'02*, 2002.

[59] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems : Computation and Control*, LNCS 3414, pages 291–305. Springer, 2005.

[60] E. Goubault. Schedulers as abstract interpretations of higher-dimensional automata. In *Proc. of PEPM'95 (La Jolla)*. ACM Press, June 1995.

[61] E. Goubault. Transitions take time. In *Proc. of ESOP'96, LNCS 1058*, pages 173–187. Springer, 1996.

[62] E. Goubault. Geometry and concurrency: A user's guide. *Mathematical Structures in Computer Science*, 10(4), August 2000.

[63] Darius Grabowski, Daniel Platte, Lars Hedrich, and Erich Barke. Time constrained verification of analog circuits using model-checking algorithms. *Electr. Notes Theor. Comput. Sci.*, 153(3):37–52, 2006.

[64] M.R. Greenstreet and I. Mitchell. Integrating projections. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, LNCS 1386, pages 159–1740. Springer-Verlag, 1998.

[65] M.R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 76–90. Springer-Verlag, 1999.

[66] P. A. V. Hall H. Zhu and J. H. R. May. Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR)*, 29(4):366–427, December 1997.

[67] F. He, L. F. Yeung, and M. Brown. Discrete-time model representation for biochemicql pathway systems. *IAENG International Journal of Computer Science*, 34(1), 2007.

[68] D. Henrion and J.B. Lasserre. Gloptipoly: Global optimization over polynomials with matlab and sedumi. In *Proceedings of the IEEE Conference on Decision and Control*, 2002.

[69] I. T. Jolliffe. *Principal Component Analysis*. Springer, 2002.

[70] D.W. Jordan and P. Smith. *Nonlinear Ordinary Differential Equations*. Oxford Applied Mathematics and Computer Science. Oxford University Press, 1987.

[71] A. Agung Julius, Georgios E. Fainekos, Madhukar Anand, Insup Lee, and George J. Pappas. Robust test generation and coverage for hybrid systems. In *HSCC*, pages 329–342, 2007.

[72] J. Kim, J. Esposito, and V. Kumar. Sampling-based algorithm for testing and validating robot controllers. *Int. J. Rob. Res.*, 25(12):1257–1272, 2006.

[73] Chr. Kloukinas, Ch. Nakhli, and S. Yovine. A methodology and tool support for generating scheduled native code for real-time java applications. In R. Alur and I. Lee, editors, *Proc. of the Third Int. Conf. on Embedded Software (EMSOFT 2003), LNCS 2855*, pages 274–289, 2003.

[74] J. Kuffner and S. LaValle. RRT-connect: An efficient approach to single-query path planning. In *Proc. IEEE Int'l Conf. on Robotics and Automation (ICRA'2000), San Francisco, CA*, April 2000.

[75] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 202–214. Springer-Verlag, 2000.

[76] M. Kvasnica, P. Grieder, M. Baoti, and M. Morari. Multi-parametric toolbox (mpt). In *Hybrid Systems: Computation and Control*, volume LNCS 2993, pages 448–462. Springer, 2004.

[77] K. Larsen, G. Behrmann, E. Brinksma, T. S. Hune A. Fehnker, P. Petterson, and J. Romijn. As cheap as possible: Efficient cost-optimal reachability for priced timed automata. In *In Proceedings of CAV, LNSC 2102*, pages 493–505. Springer, 2001.

[78] Kim G. Larsen, Marius Mikucionis, and Brian Nielsen. Online testing of real-time systems using UPPAAL: Status and future work. In E. Brinksma, W. Grieskamp, J. Tretmans, and E. Weyuker, editors, *Perspectives of Model-Based Testing*, volume 04371 of *Dagstuhl Seminar Proceedings:*, September 2004.

[79] S. LaValle and J. Kuffner. Rapidly-exploring random trees: Progress and prospects, 2000. In Workshop on the Algorithmic Foundations of Robotics.

[80] S. M. LaValle and J. J. Kuffner. Randomized kinodynamic planning. *International Journal of Robotics Research*, 20(5):378–400, May 2001.

[81] S. M. LaValle and J. J. Kuffner. Rapidly-exploring random trees: Progress and prospects. In B. R. Donald, K. M. Lynch, and D. Rus, editors, *Algorithmic and Computational Robotics: New Directions*, pages 293–308, 2001.

[82] S.M. LaValle, M.S. Branicky, and S.R. Lindemann. On the relationship between classical grid search and probabilistic roadmaps. *Intl. Journal of Robotics Research*, 23(7-8):673–692, August 2004.

[83] S. R. Lindemann and S. M. LaValle. Incrementally reducing dispersion by increasing Voronoi bias in RRTs. In *Proceedings IEEE International Conference on Robotics and Automation*, 2004.

[84] I. Mitchell and C. Tomlin. Level set method for computation in hybrid systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, LNCS 1790, pages 311–323. Springer-Verlag, 2000.

[85] Ian M. Mitchell and Jeremy A. Templeton. A toolbox of Hamilton-Jacobi solvers for analysis of nondeterministic continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, LNCS. Springer-Verlag, 2005, to appear.

[86] J. S. B. Mitchell and M Sharir. New results on shortest paths in three dimensions. In *Proc. 20th Annual ACM Symposium on Computational Geometry*, pages 124–133, 2004.

[87] B. Mourrain and J. P. Pavone. Subdivision methods for solving polynomial equations. Technical report, INRIA Research report, 5658, August 2005.

[88] T. Nahhal and T. Dang. Test coverage for continuous and hybrid systems. In *CAV*, pages 454–468, 2007.

[89] Wim Nuijten and Claude Le Pape. Constraint-based job shop scheduling with Ilog scheduler. *J. Heuristics*, 3(4):271–286, 1998.

[90] G. Pappas, G. Lafferriere, and S. Yovine. A new class of decidable hybrid systems. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, LNCS 1569, pages 29–31. Springer-Verlag, 1999.

[91] E. Plaku, L. Kavraki, and M. Vardi. Hybrid systems: From verification to falsification. In W. Damm and H. Hermanns, editors, *International Conference on Computer Aided Verification (CAV)*, volume 4590, pages 468–481. Lecture Notes in Computer Science, Springer-Verlag Heidelberg, Berlin, Germany, 2007.

[92] Stephen Prajna and Ali Jadbabaie. Safety verification of hybrid systems using barrier certificates. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 477–492. Springer, 2004.

[93] J. I. Rasmussen, K. G. Larsen, and K. Subramani. Resource-optimal scheduling using priced timed automata. In *Proc. of the 10th Int. Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2004)*, pages 220–235, 2001.

[94] S. Sankaranarayanan, H. Sipma, and Z. Manna. Scalable analysis of linear systems using mathematical programming. In *Verification, Model-Checking and Abstract-Interpretation (VMCAI 2005)*, LNCS 3385. Springer, 2005.

[95] Sriram Sankaranarayanan. Mathematical analysis of programs. Technical report, Standford, 2005. PhD thesis.

[96] H.-P. Seidel. Polar forms and triangular B-spline surfaces. In *Blossoming: The New Polar-Form Approach to Spline Curves and Surfaces, SIGGRAPH '91 Course Notes 26, ACM SIGGRAPH*, pages 8.1–8.52, 1991.

[97] L. Tan, J. Kim, O. Sokolsky, and I. Lee. Model-based testing and monitoring for hybrid embedded systems. In *proceedings of IEEE Internation Conference on Information Reuse and Integration (IRI'04)*, 2004.

[98] I. Tchoupaeva. A symbolic approach to bernstein expansion for program analysis and optimization. In *In 13th International Conference on Compiler Construction, CC 2004*, pages 120–133. Springer, 2004.

[99] Eric Thiémard. An algorithm to compute bounds for the star discrepancy. *J. Complexity*, 17(4):850–880, 2001.

[100] A. Tiwari and G. Khanna. Series of abstractions for hybrid automata. In C. Tomlin and M.R. Greenstreet, editors, *Hybrid Systems: Computation and Control*, LNCS 2289, pages 465–478. Springer-Verlag, March 2002.

[101] Ashish Tiwari and Gaurav Khanna. Nonlinear systems: Approximating reach sets. In *Hybrid Systems: Computation and Control*, volume 2993 of *Lecture Notes in Computer Science*, pages 600–614. Springer, 2004.

[102] C. Tomlin, I. Mitchell, A. Bayen, and M. Oishi. Computational techniques for the verification of hybrid systems. *Proceedings of the IEEE*, 91(7):986–1001, 2003.

[103] Jan Tretmans. A formal approach to conformance testing. In *Proceedings of the IFIP TC6/WG6.1 Sixth International Workshop on Protocol Test systems VI*, pages 257–276, Amsterdam, The Netherlands, The Netherlands, 1994. North-Holland Publishing Co.

[104] Jan Tretmans. Testing concurrent systems: A formal approach. In *CONCUR '99: Proceedings of the 10th International Conference on Concurrency Theory*, pages 46–65, London, UK, 1999. Springer-Verlag.

[105] S. Vandenberghe and S. Boyd. Semidefinite programming. *SIAM Review*, 38(1):49–95, 1996.

[106] X. Wang and F. Hickernell. Randomized halton sequences, 2000.

[107] A. Yershova, L. Jaillet, T. eon, and S. LaValle. Dynamic-domain rrts: Efficient exploration by controlling the sampling domain, 2005.