INSTITUT NATIONAL POLYTECHNIQUE DE GRENOBLE

THESE

pour obtenir le grade de

DOCTEUR DE L'INPG

Spécialité : « Automatique, Productique »

préparée au laboratoire **VERIMAG**
dans le cadre de l'École Doctorale « ***Electronique, Electrotechnique, Automatique, Télécommunications, Signal*** »

présentée et soutenue publiquement

par

DANG Thi Xuan Thao

le 10 Octobre 2000

Titre :

VÉRIFICATION ET SYNTHÈSE DES SYSTÈMES HYBRIDES

Directeur de thèse :

Oded Maler

JURY

M. **Jean Della Dora**,	Président
M. **Bruce Krogh**,	Rapporteur
M. **Marcel Staroswiecki**,	Rapporteur
M. **Oded Maler**,	Directeur de thèse
M. **Eugene Asarin**,	Examinateur
M. **Pravin Varaiya**,	Examinateur

Công cha nhu núi Thái Son
Nghiã me. nhu nuóc trong nguôǹ cha?y ra.

Acknowledgements

I wish to express my sincere gratitude to Professor Joseph Sifakis for giving me this opportunity to work with extraordinary people. I would like to thank Professors Eugene Asarin, Jean Della Dora, Bruce Krogh, Marcel Staroswiecki, and Pravin Varaiya for accepting to be members of my jury and for their valuable comments on my thesis.

My greatest thanks go to my advisor, Dr. Oded Maler, for offering me technical, moral, and financial support, not to mention an interesting research subject. His knowledge, creative mind, continuous inspiration, and excellent guidance not only led me to the results presented in this thesis but also made these years the most fruitful I have ever had. Moreover, my PhD studies have been so enjoyable thanks to his humor, generosity, and unbounded enthusiasm. I will always be grateful to him for what he taught me about science, research, and life. Thanks for being a great advisor.

It has been a great pleasure for me to work closely with Professor Eugene Asarin. I am thankful for his valuable guidance and numerous discussions which have been sources of inspiration to my work. He also helped me solve many concrete problems. Without him, this thesis would probably not exist.

I am happy to thank Dr. Paul Caspi, who has assisted me in many different ways through my stay here. He has been very generous with his time to me and taught me a lot about science and culture.

Special thanks are also given to Olivier for his superb library Cubes, his collaboration, and many nice conversations about things other than verification.

I would especially like to thank Marius Minea for his careful proofreading of my thesis and detailed comments which greatly improved the quality of the presentation.

I am pleased to acknowledge my friendly and talented colleagues in Verimag. To Yannick and Stavros, let me simply say thank you. Thanks for teaching me an excellent programming course and for being very understanding friends. I thank Sergio for being encouraging, and I really appreciate his insightful suggestions and advice. He spent much time answering my questions about verification. Sorav deserves special thanks for his contributions to some applications. I also thank Bertrand, Marius, Moez, Peter for helpful exchanges, and all other colleagues for useful seminars. Thanks to Nicolas for helping me with daily computer problems.

The friend who has made my time in Verimag memorable is Yasmina. Thanks for a course on job-shop scheduling, ‘aatini el naya wa rghani’, ‘la lune et la rivière’, and many other things I fail to name here.

Finally, I would like to thank all the members of Verimag, who have created such a nice atmosphere in the lab.

Ca?m on Raphaël vì những năm tháng ha.nh phúc này. Ca?m on ban bè thân yêu vì những gì mang ddêń cho tôi và Hanoi nho? bé ddã o? bên tôi ddêń ngày hôm nay.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

Continuous systems have, traditionally, been the focus of system theory. Due to significant advances in digital-processor technology in the past few decades, the use of digital controllers has successfully improved the automation level in the control of physical plants. One typical example is a chemical batch plant where computers are used to supervise complex sequences of chemical reactions, each of which is modeled as a continuous process. As another example, consider a digital engine controller in a car, which has to interact with the physical processes in the engine as well as with the events generated by the driver. The increasing integration of such controllers results in highly complex systems involving both continuous and discrete event dynamics. In addition to discontinuities introduced by the computer, most physical processes admit components (e.g. valves, gears, switches) and phenomena (e.g. collision, emptying of tanks) whose most useful models are discrete. Systems that consist of a combination of discrete and continuous features are called *hybrid systems*, and they arise in many applications, such as chemical process control [68], air traffic management systems [111], robotics [6], and automobiles [18].

During system design, formal verification and controller synthesis are two important issues. The goal of *formal verification* is to prove that the system performs as expected. As today's automated systems are growing in scale and complexity, the possibility of subtle errors is much greater. In particular, for *safety-critical systems*, any error during the operation may cause loss not only in money but also in human life, and, as a result, it is crucial to ensure that the system is always safe (no error occurs). Whereas the goal of verification is to ensure a desired property of the designed system, the goal of *controller synthesis* is to design controllers so that the controlled system satisfies a desired specification.

This thesis is concerned with the *formal verification and synthesis of hybrid systems*. The best way to understand the problematics in this field of research and the approach we propose is to follow simple examples. In the sequel we discuss two hybrid system examples, the role of which is to illustrate several characteristics of hybrid systems that make verification and synthesis difficult and to motivate the reader to proceed to the definitions and the algorithms in the next chapters. The second example is a variation of the first, and both are very simple one-dimensional linear systems, which can be analyzed by hand. The discussion will be informal and not rigourous, but the ideas underlie the motivation of our approach.

1.2 Hybrid System Examples

1.2.1 Example 1

Consider a thermostat that is used to control the temperature of a room. The thermostat consists of a heater and a thermometer. Its lower and upper thresholds are set at θ_m and θ_M such that $\theta_m < \theta_M$. The heater is maintained on as long as the room temperature is below θ_M, and it is turned off whenever the thermometer detects that the temperature reaches θ_M. Similarly, the heater remains off if the temperature is above θ_m and is switched on whenever the temperature falls to θ_m. One can think of the room temperature and the thermostat as a *dynamical system* whose *state* is defined by the room temperature x, which changes continuously, and the operation mode of the thermostat, which changes between 'on' and 'off'. The evolution of the temperature is described as follows:

$$\dot{x} = \begin{cases} f_1(x) = -x + 4 & \text{if the heater is on,} \\ f_2(x) = -x & \text{otherwise.} \end{cases}$$

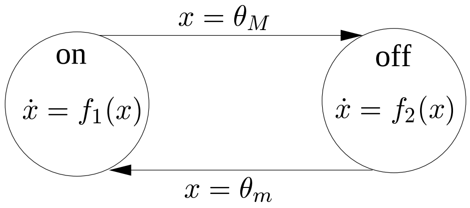


Figure 1.1: The model for the thermostat.

A visual description of the thermostat is given in form of a directed graph whose vertices represent two operation modes 'on' and 'off' (see Figure 1.1). We associate with the edges the conditions for switching from one mode to another.

Our *verification problem* is to prove that the temperature always stays in the desired range, that is,

$$m \leq x \leq M \tag{1.1}$$

In this simple example, for a given initial condition $x(0) = \theta$ the solution of the differential equations of the modes 'on' and 'off' can be written as $x(t) = \theta e^{-t} + 4(1 - e^{-t})$ and $x(t) = \theta e^{-t}$, respectively.

Let us now describe a *scenario* of the system's behavior starting from an initial state where the temperature $x = \theta_0$ and the operation mode of the thermostat is 'on'. Suppose that the initial temperature is in the desired range, that is, $m \leq \theta_0 \leq M$. The heater initially being on, the temperature evolves as follows:

$$x(t) = \theta_0 e^{-t} + 4(1 - e^{-t}).$$

Increasing monotonically over time, the temperature reaches θ_M after t_1 time, and the heater is then shut off. The temperature is next governed by the differential equation of the mode 'off' and can be written as

$$x(t + t_1) = \theta_M e^{-t+t_1}.$$

The temperature decreases monotonically over time and falls to θ_m, at which point the heater is turned on, and the process continues as shown in Figure 1.2. One can see that the trajectory of the temperature alternates between two phases corresponding to the two operation modes of the thermostat.

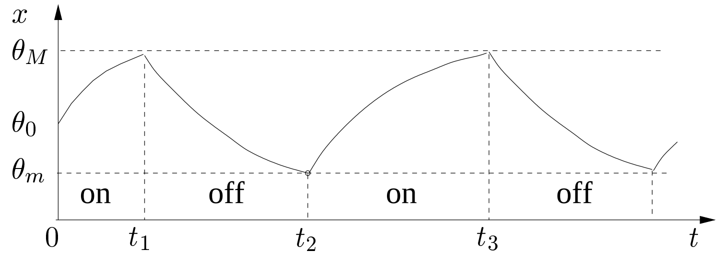


Figure 1.2: A trajectory of the room temperature.

It is not hard to see that the thermostat satisfies the property (1.1) if only the thresholds θ_m and θ_M satisfy the following condition:

$$\theta_m \geq m \quad \wedge \quad \theta_M \leq M. \tag{1.2}$$

Note that such a verification problem can be *analytically* solved only when solutions to the differential equations are known[1]. In more general cases, *numerical simulations* are used to obtain an approximation of the behavior of the system from a given state. For most systems, the state of the art in simulation techniques allows the approximate solution to be as close as desired to the exact one. However, in practice the initial conditions are usually not known exactly but only known to lie within some range. Consequently, instead of a single trajectory, one needs to consider an infinite number of trajectories. Concerning numerical techniques, this gives rise to the special difficulty in simulating *sets of trajectories*.

[1]Many differential equations tend to not have closed-form solutions.

1.2.2 Example 2

Before attempting to analyze a model, we should make sure that it captures all possible behaviors of the physical process. The described model assumes ideal conditions, namely the thermometer can detect exactly the moments the temperature reaches the thresholds and the switching time between 'on' and 'off' is zero. Nevertheless, in practical situations exact threshold detection is impossible due to incertitude of sensors. Similarly, the reaction time of the on/off switch is usually non-zero. The effect of these inaccuracies is that we cannot guarantee switching exactly at the nominal values θ_m and θ_M but only in their neighborhoods. To make the model reflect this uncertainty, we modify the switching conditions as follows. The thermostat switches off the heater if the temperature satisfies $\theta_M - \epsilon \leq x \leq \theta_M + \epsilon$ and switches it on if $\theta_m - \epsilon \leq x \leq \theta_m + \epsilon$, for some $\epsilon > 0$. This means that when the temperature enters the interval $[\theta_M - \epsilon, \theta_M + \epsilon]$ the thermostat can either turn the heater off immediately or keep it on for some time provided that $x \leq \theta_M + \epsilon$. We say that the behavior of the system is *non-deterministic* in the sense that from a given state the temperature can follow more than one trajectory (see Figure 1.4). The enhanced model with uncertainty is depicted in Figure 1.3.

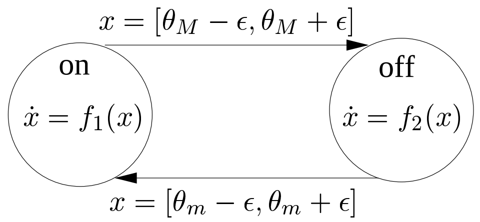


Figure 1.3: The model of the thermostat with uncertainty.

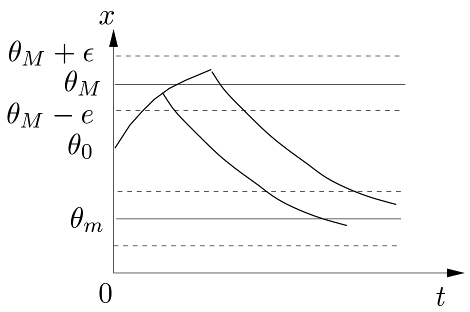


Figure 1.4: Two different trajectories of the temperature starting at θ_0.

Another source of non-determinism can come from the continuous dynamics which are of the form $\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u})$ where $\mathbf{u}$ is an under-specified input. In this case, from every initial state there might be a "tube" of possible trajectories, each of which corresponds to a different input signal. In the example of the thermostat, a source of such uncertainty can be fluctuations in the outside temperature which we obviously cannot control.

The model with uncertainty is harder to analyze both analytically and using simulation since, to characterize all possible behaviors, we have to deal with sets of trajectories. Furthermore, it is imposible to simulate the system with all inputs. This shows that the simulation approach is not suitable for verifying such a system since sample solutions do not give absolute assurance that the system satisfies the property. Although the simple one-dimensional thermostat example can still be analyzed analytically (since we know the solutions of the differential equations and, moreover, they are monotonous), the reader can imagine that for high-dimensional systems with many modes and complex continuous dynamics, there is no existing methodology to solve such verification problems.

We turn now to the *controller synthesis* problem. Our *objective* is to keep the temperature of the room in the range $[m, M]$. Suppose we are given a thermostat whose structure is fixed, but we are free to set the thermostat at the appropriate thresholds θ_m and θ_M so that the room temperature is maintained as desired. In other words, we use the thresholds to characterize the feed-back function in order to achieve the desired behavior. Then, the goal is to find switching rules based on state observation such that the system always satisfies the desired property. We will show, in this thesis, how the methods proposed for verifying hybrid systems can also be used to *synthesize automatically* such *switching rules*.

To sum up, the essence of the above examples is that in order to develop a framework for verification and synthesis of hybrid systems we need:

- An appropriate model capable of capturing the interaction between discrete and continuous dynamics and constraints on the physical system as well as on the controller.
- Analysis methods which are rigourous in the sense that they can characterize all possible behaviors of the system and derive feasible controls.

In addition, hybrid systems in practice are often complex, which makes automatic analysis very desirable. This motivates us to adopt the algorithmic approach to verification which consists in building a software tool which can analyze automatically all the behaviors of a given system and decide whether it satisfies a given property. This approach has been applied successfully to purely discrete systems (e.g. digital circuits, communication protocols) [33], but its adaptation to continuous and hybrid systems is still a serious challenge.

1.3 Thesis Outline

In Chapter 2 we discuss *hybrid automata* [4], the modeling formalism we will use for hybrid systems. Various hybrid system models have been proposed; the reason we have chosen this model is that it can capture naturally a wide range of hybrid behaviors and, moreover, provides a framework suitable for the verification and synthesis problems we tackle in this thesis.

The chapter includes the theoretical background neccessary for subsequent discussions.

Chapter 3 is concerned with the *verification problem* for hybrid systems. We are interested in verifying *invariance* properties which state that all trajectories of the system remain inside a subset of the state space. Proving such properties can be done using reachability analysis, that is, computing all states which can be reached by any trajectory of the system. We examine the problematics of extending the *algorithmic* methodology to hybrid systems and propose an approach which consists in first developing reachability techniques for purely continuous systems and then adapting them for hybrid systems. We then discuss computability issues and present a basic reachability algorithm for continuous systems, based on numerical integration and polyhedral approximations. This chapter serves as an introduction to the next three chapters.

In Chapter 4 we develop a technique for approximating reachable sets of *linear continuous systems*. This technique exploits the special properties of linear systems so that the computation is relatively fast and the approximation error does not propagate from iteration to iteration. This technique has some common features with the work of Chutinan and Krogh [31], which has been developed independently. We also extend this technique to deal with linear systems with *uncertain input* (differential inclusions) based on some ideas proposed by Varaiya [114].

In Chapter 5 we present an alternative reachability technique which can be applied to *non-linear continuous systems*. The technique, which we call "face lifting", is inspired by earlier work of Greenstreet [47]. The novelty in our approach is in the way we approximate high dimensional subsets of the state space using orthogonal polyhedra. This makes both the linear and non-linear analysis techniques applicable, in principle, to any dimension. Of course, computational complexity is a major limiting factor.

The goal of Chapter 6 is to adapt the above techniques to the *verification of hybrid systems*. Strategies and methods to increase the performance of the verification algorithm and other important computational issues are also investigated.

Chapter 7 is devoted to the *controller synthesis* problem for hybrid systems. More precisely, we consider systems with several modes, each of which has different dynamics. Our goal is to *automatically* synthesize a controller which switches the system between modes in order to satisfy invariance properties. We present an abstract synthesis algorithm, based on the algorithm in [15] for timed automata, and then apply the reachability techniques of Chapters 4 and 6 to derive an approximate version of the algorithm.

In Chapter 8 we describe the *experimental tool d/dt* , in which most of the algorithms discussed in the thesis have been implemented. The tool provides automatic verification and switching controller synthesis for hybrid systems with linear differential inclusions.

Each of the above chapters ends with some examples (academic and real-life), which illustrate the applicability of our approach, as well as 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 2 contains important definitions and notations which are refered to throughout the thesis. For reference, we include a glossary of notation in Table 1.1.

Table 1.1: Glossary of Notation

$\mathbb{N}$	natural numbers
$\mathbb{R}$	real numbers
$\mathcal{T}$	time domain, p. 29
$\mathbb{Z}$	integer numbers
$\mathbf{0}$	zero vector
$\emptyset$	empty set
$\mathcal{O}(\cdot)$	asymptotic notation, p. 60
Continuous Systems	
$\delta(F)$	reachable set from set F, p. 38
$\delta_r(F)$	successors of set F at time r, p. 38
$\delta_{[0,r]}(F)$	successors of set F within time interval $[0,r]$, p. 38
$\Delta(F)$	backward-reachable set from set F, p. 39
$\xi_{\mathbf{x}}$	trajectory starting from point $\mathbf{x}$, p. 32
$\xi_{\mathbf{x},\mu}$	trajectory starting from point $\mathbf{x}$ under input μ, p. 74
Hybrid Automata	
δ_c	continuous-successor operator, p. 39
δ_d	discrete-successor operator, p. 41
$\delta_{qq'}$	discrete-successor operator w.r.t transition from q to q', p. 40
$\gamma = (\alpha, \beta)$	hybrid automaton trajectory, p. 35
π	one-step predecessor operator, p. 146
π_q^∞	unbounded-time predecessor operator w.r.t discrete state q, p. 146
$\mathcal{U}_q$	until operator w.r.t discrete state q, p. 146
Geometric Operations	
B	the unit ball at the origin, p. 59
$bloat(C, \vartheta)$	bloating convex polyhedron C by ϑ, p. 63
$conv(F)$	convex hull of set F

$d(\mathbf{x}, \mathbf{y})$	distance between points $\mathbf{x}$ and $\mathbf{y}$, p. 59
$grid_o(C)$	orthogonal polyhedron over-approximating convex polyhedron C, p. 64
$grid_u(C)$	orthogonal polyhedron under-approximating convex polyhedron C, p. 64
$h(X, Y)$	the Hausdorff distance between sets X and Y, p. 59
$\Lambda(b, \mathbf{d}^-, \mathbf{d}^+)$	lifting the faces of hyper-rectangle b by $\mathbf{d}^-$ (left) and $\mathbf{d}^+$ (right), p. 94
$N(X, \epsilon)$	ϵ-neighborhood of set X, p. 59
$N_s(G, \epsilon)$	rectangular ϵ-neighborhood of orthogonal polyhedron G, p. 71
$N_{\boldsymbol{v}}(C)$	neighborhood of convex polyhedron C obtained by pushing outward its faces by $\boldsymbol{v}$, p. 88
$\rho(X)$	diameter of set X, p. 59
$C \sqcap_o G$	orthogonal polyhedron over-approximating intersection of convex polyhedron C and orthogonal polyhedron G, p. 118
$C \sqcap_u G$	orthogonal polyhedron under-approximating intersection of convex polyhedron C and orthogonal polyhedron G, p. 120
$\langle \mathbf{x}, \mathbf{y} \rangle$	scalar product of vectors $\mathbf{x}$ and $\mathbf{y}$
$\oplus$	Minkowski sum
$\|\| \cdot \|\|$	Euclidian norm
$\lfloor d \rfloor$	integer part of real number d, p. 94

Part I

Modeling Formalism

Chapter 2

Hybrid Automata

As we have seen from the informal examples in the previous chapter, in order to analyze hybrid systems we need a model which is rich enough to describe both continuous-time dynamics and discrete transitions. In addition, the model should be potentially non-deterministic allowing more than one behavior from a given state. Multiple behaviors can be due to uncertainties in both continuous and discrete dynamics which can arise from external disturbances, uncontrollable events, the modeling abstraction, or imprecision in our knowledge of the system. Continuous dynamics are traditionally considered within the context of differential equations, and discrete-event dynamics are often modeled and analyzed using automata. Hybrid automata [4] combine these two models and provide an effective formalism which satisfies the above requirements. In hybrid automata, the continuous behavior is captured by differential equations associated with the discrete states and the discrete behavior is captured by transitions between states. In this work we use hybrid automata as a modeling formalism for hybrid systems.

This chapter is organized as follows. The first part is devoted to some basic definitions of automata and continuous systems, the main components of hybrid automata. Next, we present the syntax, semantics for hybrid automata, and some basic reachability notions. Finally, we discuss briefly some other hybrid models considered in the literature.

2.1 Notation

Automata and continuous dynamics can both be viewed as dynamical systems whose states evolve in *time*. Their definition is therefore associated with a time set. For automata, the underlying time set is any discrete set which is order isomorphic to $(\mathbb{N}, \leq)$. For continuous dynamical systems, time ranges over the real numbers. We begin by introducing some notations needed for describing the temporal behavior of the systems under consideration.

Throughout this thesis we will use the *time domain* $\mathcal{T} = \mathbb{R}^+$.

A *time sequence* is a strictly increasing sequence of time points $t_0, t_1, t_2, ..$ where $t_k \in \mathcal{T}$ for every $k \in \mathbb{N}$.

Definition 1 (Temporal Behavior)
A temporal behavior over a topological space S is a partial function $\beta : \mathcal{T} \to S$ whose domain of definition is an interval $[0, r)$ for some $r \in \mathcal{T} \cup \{\infty\}$.

We call r the metric length of β, denote it by $|\beta|$, and say that β is infinite if $r = \infty$.

Definition 2 (Piecewise-Continuous Behavior)
A behavior β is piecewise-continuous if it admits a time sequence $\mathcal{J}(\beta) = 0, t_1, t_2, \ldots$ such that for every $k \in \mathbb{N}$, β is continuous on the interval $I_k = [t_k, t_{k+1})$.

Note that there may be many time sequences $I_0, I_1, \ldots$ on which a behavior β is piecewise-continuous. We denote by $\mathcal{I}(\beta)$ such time interval sequence with the smallest number of intervals. This number is called the *logical length* of β, denoted by $|\beta|_l$.

A piecewise-constant behavior is a special case of piecewise-continuous behaviors where β is constant on every interval I_k. The *untimed abstraction* of a piecewise-constant behavior β is the sequence $\bar{\beta} = s_0, s_1, \ldots$ such that for every k, $s_k = \beta(t_k)$.

Figure 2.1 shows an example of piecewise-continuous and piecewise-constant behaviors.

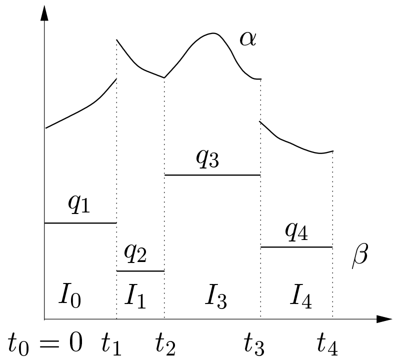


Figure 2.1: A piecewise-continuous behavior α and a piecewise-constant behavior β of logical length 4 with $\bar{\beta} = q_1, q_2, q_3, q_4$.

2.2 Automata

We define a finite automaton without input and output as follows.

Definition 3 (Automaton)
An automaton is $\mathcal{M} = (Q, \delta)$ where

- *Q is a finite set of states.*
- *$\delta : Q \to 2^Q$ is the transition function.*

The transition relation describes how the system may evolve. Notice that the behavior of the system is potentially *non-deterministic* since for a given state $q \in Q$, $\delta(q)$ represents a set of possible next states rather than a unique state.

Definition 4 (Behavior of automata)
Given an initial state $q \in Q$, a behavior of $\mathcal{M}$ is a sequence $\sigma : \mathbb{N} \to Q$ such that $\sigma(0) = q$ and for every $k > 0$, $\sigma(k+1) \in \delta(\sigma(k))$.

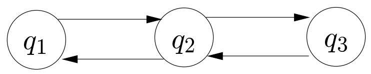


Figure 2.2: A 3-state automaton.

It is important to note that in studying the behavior of automata, only the *ordering* information between the transitions is considered and there is no attempt to embed the behaviors into real metric time. Consider the automaton of Figure 2.2 where $Q = \{q_1, q_2, q_3\}$ and the initial state is q_1. The automaton admits a behavior $\sigma = q_1, q_2, q_1, q_2, q_3, q_2$. This description of the behavior does not say anything about the time elapsed while the automaton is at each state. In other words, the behavior σ could be the untimed abstraction of infinitely many piecewise-constant behaviors of the form $\beta : \mathcal{T} \to Q$ (see Figure 2.3). The behavior of hybrid automata, as we shall soon see, is, however, grounded in the real time axis.

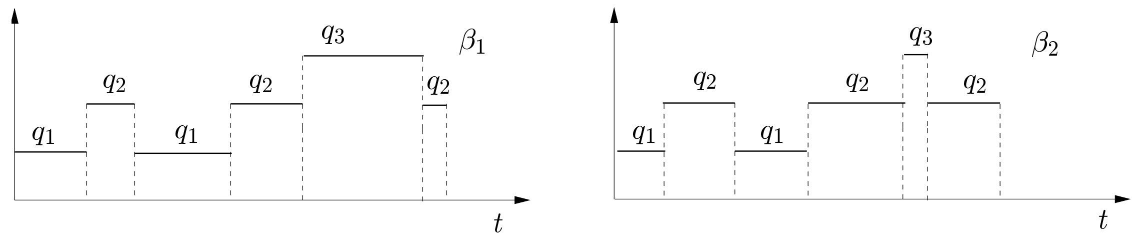


Figure 2.3: Two piecewise-constant behaviors β_1 and β_2 which can be associated with the sequence $\sigma = q_1, q_2, q_1, q_2, q_3, q_2$ of the automaton of Figure 2.2.

2.3 Continuous Dynamical Systems

Definition 5 (Continuous Dynamical System)
A continuous dynamical system is $\mathcal{C} = (\mathcal{X}, f)$ where

- $\mathcal{X} = \mathbb{R}^n$ *is the state space.*
- $f : \mathcal{X} \to \mathbb{R}^n$ *is a continuous vector field.*

The behavior of the system is governed by the differential equation:

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

where $\mathbf{x} \in \mathcal{X}$ is the state of the system.

The behavior of a continuous dynamical system is characterized by the solutions to the initial-value problems of its differential equation.

Definition 6 (Trajectory of Continuous Dynamical Systems)
A trajectory of $\mathcal{C}$ starting from $\mathbf{x} \in \mathcal{X}$ is a continuous behavior $\xi_{\mathbf{x}} : \mathcal{T} \to \mathcal{X}$ such that $\xi_{\mathbf{x}}$ is the solution of (2.1) with initial condition $\mathbf{x}(0) = \mathbf{x}$.

We assume that f is globally Lipschitz in $\mathbf{x}$, which guarantees that there exists a unique solution to (2.1) for every initial condition in $\mathcal{X}$ [61]. The continuous system of Definition 5 is *deterministic* in the sense that from a given point it generates a *unique* trajectory.

Given two points $\mathbf{x}, \mathbf{x}' \in \mathcal{X}$, we say that $\mathbf{x}'$ is *reachable* from $\mathbf{x}$ in time $t < \infty$ if $\mathbf{x}' = \xi_{\mathbf{x}}(t)$. We denote this by $\mathbf{x} \xrightarrow{t} \mathbf{x}'$.

2.4 Hybrid Automata

A hybrid automaton is an automaton augmented with continuous variables whose dynamics at every discrete state are defined by differential equations. Transitions between states are enabled by conditions on the values of these variables.

2.4.1 Syntax

Definition 7 (Hybrid Automaton)
An n-dimensional hybrid automaton is a tuple $\mathcal{A} = (\mathcal{X}, Q, f, H, G, R)$ where

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is the continuous state space. Elements of $\mathcal{X}$ are written as $\mathbf{x} = (x_1, x_2, \dots, x_n)$.*
- Q *is a finite set of discrete states.*
- $f : Q \to (\mathcal{X} \to \mathbb{R}^n)$ *assigns a continuous vector field on $\mathcal{X}$ to each discrete state. While the hybrid automaton stays in discrete state q the evolution of the continuous variables is governed by the differential equation $\dot{\mathbf{x}} = f(q)(\mathbf{x})$.*
- $H : Q \to 2^{\mathcal{X}}$ *are staying conditions ("invariants"). $H(q)$ is the condition that must be satisfied by the continuous variables if the automaton is to stay in discrete state q.*

- $G : (Q \times Q) \to 2^{\mathcal{X}}$ *are transition guards determining the conditions for switching from one discrete state to another. When the automaton is in discrete state* q *and* $\mathbf{x} \in G_{qq'}$*, it can make a transition from* q *to* q'*.*
- $R : (Q \times Q) \to (\mathcal{X} \to 2^{\mathcal{X}})$ *is the reset map which assigns to each transition a multi-valued function.* $R(q, q')$ *defines how the continuous variables may change when the automaton switches from* q *to* q'*.*

We assume that for every $q \in Q$, $f(q)$ is *globally Lipschitz* over $\mathbf{x} \in H(q)$. This assumption ensures the existence and uniqueness of solutions of the differential equation at q for every initial condition in $H(q)$ [61]. We also assume that for all q, $q' \in Q$, $H(q)$ and $G(q, q')$ are *closed* sets.

The intuitive meaning of Definition 7 is that the set of discrete states Q denotes all the possible continuous "modes" of the hybrid automaton. The system can evolve in a discrete state q only if the current continuous state is in $H(q)$. These staying conditions can arise from constraints imposed by physical systems or decisions in system design. While being at discrete state q where $\mathbf{x} \in H(q)$, the system can evolve according to the dynamics $f(q)$. Whenever it reaches a point $\mathbf{x}' \in G(q, q')$ the transition from q to q' is enabled and the system can switch to discrete state q'. At q' the continuous variables will be reset to new values according to $R(q, q')$, which become the initial states for the evolution according to $f(q')$. Our convention is that if there is no transition from q to q' then $G(q, q')$ is defined as empty set. If the reset map $R(q, q)$ is the identity, i.e. $R(q, q)(\mathbf{x}) = \{\mathbf{x}\}$ for all $\mathbf{x} \in \mathcal{X}$, we may assume further that $G(q, q) = \emptyset$ because a transition from a discrete state to itself without a change in the continuous variables is meaningless. Notice that in the examples so far all the reset maps were the identity.

It is customary to represent graphically the hybrid automaton $\mathcal{A}$ by a directed graph whose vertices represent the discrete states and edges represent the transitions. We write the staying conditions and the differential equations inside the vertices. With the edges we associate the transition guards and the resets (identity resets will be omitted). From now on we will use the notation f_q for $f(q)$, $G_{qq'}$ for $G(q, q')$, G_q for $\bigcup_{q'} G_{qq'}$, H_q for $H(q)$, H for $\bigcup_q H_q$, and $R_{qq'}$ for $R(q, q')$. We will often use 'automaton' to refer to a hybrid automaton with the context indicating the meaning.

We can now define formally hybrid automata for the thermostats examined in the previous chapter (see Figure 2.4). The two operation modes 'on' and 'off' are represented by two discrete states q_1 and q_2. The continuous variable x models the temperature, and its dynamics in each mode is given by a differential equation. We define the conditions for the thermostats to stay in each mode by the staying conditions, such as $x \leq \theta_M$ for q_1 in case of the ideal thermostat. The conditions on the temperature for the thermostats to switch between two modes are specified by the transition guards. Notice that the upper (lower) bound of the switching condition for the transition from q_1 to q_2 (from q_2 to q_1) need not be defined due to the specified staying conditions at the source state of the transition. In both models, the temperature does not change at the switching points, and the resets are thus

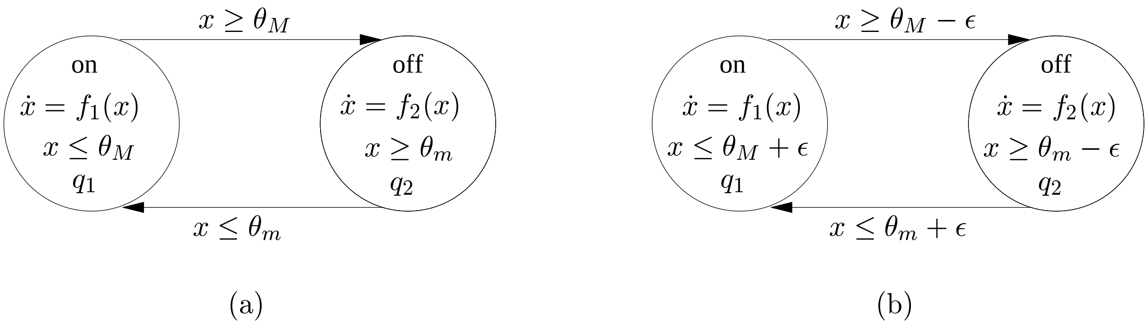


Figure 2.4: The hybrid automata for the ideal thermostat (a) and for the thermostat with uncertainty (b).

identity functions.

Although the continuous evolution is deterministic at every discrete state, the hybrid evolution may be *non-deterministic* for the following reasons:

- The sets H_q and $G_{qq'}$ may intersect with each other not only on their boundaries. As a result, there are points from which the system, when being at q, can either switch to q' or continue the continuous evolution at q.
  To illustrate, consider again the automata of the thermostats. Suppose that the system starts at q_1 ('on'). After some time the temperature rises to θ_M, which enables the transition from q_1 to q_2. At this point the staying condition of q_1 in (a) forces the system to switch to q_2 while in (b) it allows the system to either stay there for some more time as long as $x \leq \theta_m + \epsilon$ or switch at once. In other words, the system in (b) can switch from q_1 to q_2 anywhere in the interval $[\theta_M - \epsilon, \theta_M + \epsilon]$. Therefore, the behavior of the system in (a) is deterministic while the behavior of the system in (b) is non-deterministic.

- The resets can be set-valued maps, and this causes non-deterministic changes in continuous variables whenever a transition is taken.

- There can be multiple transitions enabled at the same time because $H_q \cap G_{qq'} \cap G_{qq''}$ may be non-empty.

2.4.2 Semantics

We now turn to the *semantics* of a hybrid automaton like $\mathcal{A}$. By the semantics of such an object we mean *the set of all behaviors it can generate.* In this sense, the semantics of a continuous system defined by a differential equation is the set of all solutions of its initial value problem, namely trajectories.

For the hybrid automaton $\mathcal{A}$ we will consider temporal behaviors over the hybrid state space $Q \times \mathcal{X}$. A state $(q, \mathbf{x})$ of $\mathcal{A}$ can change in two ways:

- By *continuous evolution*: the values of the continuous variables change according to the dynamics f_q while the discrete state q remains constant.
- By *discrete evolution*: the system changes the discrete state by making a transition and possibly changes the values of the continuous variables according to the reset function.

Let $\mathbf{x}$, $\mathbf{x}' \in \mathcal{X}$ and q, $q' \in Q$. Let $\alpha : \mathcal{T} \to \mathcal{X}$ be the (unique) solution of $\dot{\mathbf{x}} = f_q(\mathbf{x})$ with the initial condition $\alpha(0) = \mathbf{x}$.

The dynamics q is enabled from $\mathbf{x}$ for time $t > 0$ if $\alpha(t') \in H_q$ for every $t' \in [0, t]$. This is denoted by $\mathbf{x} \xrightarrow{q,t}$. The state $(q, \mathbf{x}')$ is reachable from $(q, \mathbf{x})$ by the continuous dynamics f_q if there exists $t < \infty$ such that $\alpha(t) = \mathbf{x}'$ and $\alpha(t') \in H_q$ for every $0 \leq t' \leq t$. In this case we say that $\mathbf{x}'$ is *q-reachable* from $\mathbf{x}$ and denote it by $\mathbf{x} \xrightarrow{q,t} \mathbf{x}'$.

A state $(q', \mathbf{x}')$ is reachable from $(q, \mathbf{x})$ via a discrete transition if $\mathbf{x} \in G_{qq'}$, $\mathbf{x}' = R_{qq'}(\mathbf{x})$ and $\mathbf{x}' \in H_{q'}$.

Definition 8 (Trajectory of Hybrid Automata)
A trajectory of a hybrid automaton $\mathcal{A}$ starting from a state $(q_0, \mathbf{x}_0)$ is a pair of behaviors $\gamma = (\alpha, \beta)$ of the same metric length where $\alpha : \mathcal{T} \to \mathcal{X}$ is piecewise-continuous and $\beta : \mathcal{T} \to Q$ is piecewise-constant satisfying the following conditions:

1. *Initiality: $\alpha(0) = \mathbf{x}_0$ and $\beta(0) = q_0$.*
2. *Continuous evolution: for every interval $I_k = [t_k, t_{k+1}) \in \mathcal{I}(\beta)$ such that $\bar{\beta}_k = q$, $\alpha(t_k) \xrightarrow{q,t} \alpha(t_k + t)$ for every $t \in [0, t_{k+1} - t_k)$.*
3. *Transition condition: for every $t_k \in \mathcal{J}(\beta)$ such that $\bar{\beta}_{k-1} = q$ and $\bar{\beta}_k = q'$, $\alpha(t_k^-) \in G_{qq'}$ and $\alpha(t_k) = R_{qq'}(\alpha(t_k^-))$ where $\alpha(t_k^-)$ denotes the left limit of α at t_k.*

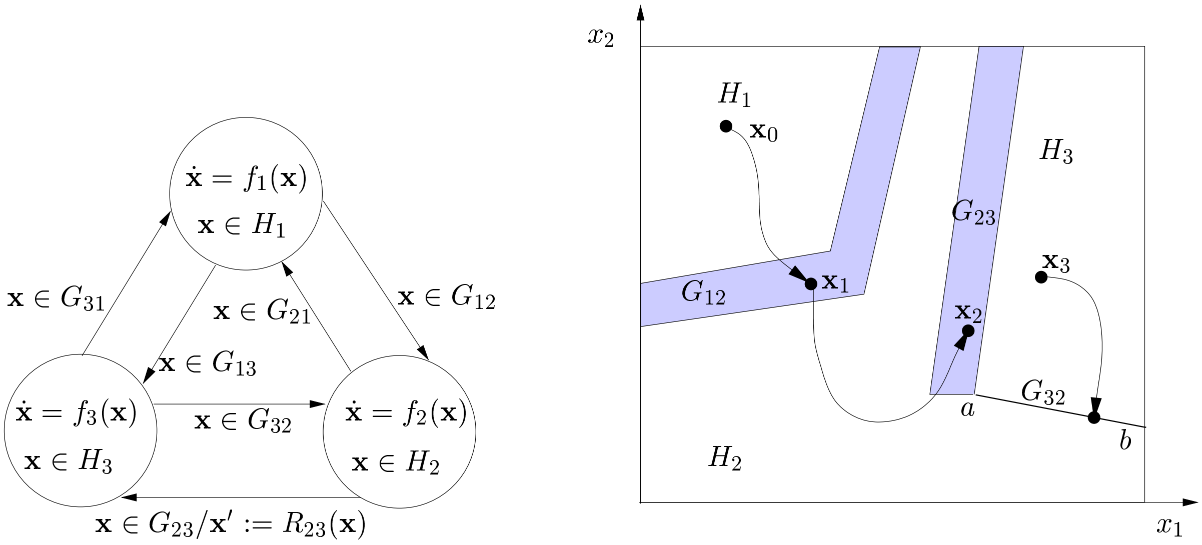
The logical length of γ, denoted by $|\gamma|_l$, is the logical length of β.

Figure 2.5 sketches another hybrid automaton and a projection of one of its trajectories onto the continuous state space $\mathcal{X} = \mathbb{R}^2$.

The use of staying conditions and the ability to switch in zero time between states can create phenomena which are impossible in models of well-behaving continuous systems. The two behavioral anomalies that can be generated by hybrid automata are *blocking* and *Zeno behaviors*.

Blocking Behaviors

A trajectory is blocking if it reaches a point $(q, \mathbf{x})$ from which it is impossible to continue neither by continuous nor by discrete dynamics, i.e. $\mathbf{x} \notin H_q$ and $\mathbf{x} \notin G_{qq'}$ for every q, $q' \in Q$.



(1) (2)

Figure 2.5: (1) A 3-state hybrid automaton $\mathcal{A}$ ($\mathcal{X} = \mathbb{R}^2$, the guards G_{12} and G_{23} are the shaded regions and G_{32} is the line segment ab); (2) A sketch of a behavior of $\mathcal{A}$ starting from $(q_1, \mathbf{x}_0)$. The jump from $\mathbf{x}_2$ to $\mathbf{x}_3$ is due to the reset R_{23}.

An automaton is *non-blocking* if from every $(q, \mathbf{x})$ such that $\mathbf{x} \in H_q$ there are no blocking trajectories.

There are two ways to enter into a blocking situation:

- After a discrete transition, the system moves from $(q, \mathbf{x})$ to $(q', \mathbf{x}')$ but $\mathbf{x}'$ does not satisfy the staying condition of q'; as a result, the continuous evolution cannot be continued. This means that $R_{qq'}(G_{qq'}) \not\subseteq H_{q'}$. As an example, consider the case when a careless user sets the thresholds of the thermostat such that $\theta_M < \theta_m$ and then turns on the heater. The temperature rises to $\theta_M + \epsilon$, at which point the thermostat must switch off the heater, but the current temperature does not satisfy the staying condition of the mode 'off'; the automaton is thus blocked.

- The system leaves H_q without entering any $G_{qq'}$; consequently, the continuous evolution at q is no longer possible, but the system cannot switch to another dynamics.

One reason for blocking behaviors is the inconsistency of the model. An automaton can be guaranteed to be non-blocking as follows. To prevent the first source of blocking, for every $q, q' \in Q$, $G_{qq'}$ can be replaced by $G'_{qq'} = G_{qq'} \cap R^{-1}_{qq'}(H_{q'})$ where $R^{-1}_{qq'} : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is defined as

$$R^{-1}_{qq'}(X) = \{\mathbf{x}' \mid \exists \mathbf{x} \in X \ \mathbf{x} = R_{qq'}(\mathbf{x}')\}.$$

The second type of blocking can be removed by adding a new discrete state that the system enters whenever it goes out of every H_q.

Zeno Behaviors

A *Zeno behavior* is a piecewise-constant behavior having an infinite number of transitions in a bounded interval of time. For example, consider a behavior β such that $\mathcal{J}(\beta)$ is the time sequence $\{t_k\},\ k = 0 \ldots \infty$, defined as $t_0 = 0$ and $t_k = t_{k-1} + 1/2^k$ which converges to 1, and $\beta(t_k) = 0$ for even k and $\beta(t_k) = 1$ for odd k. Such a behavior switches infinitely many times between 0 and 1 in the interval $[0, 1)$. An automaton is *non-Zeno* if it does not admit Zeno behaviors.

In the example of the thermostat with uncertainty, if the thresholds θ_m and θ_M are chosen such that the intervals $[\theta_m - \epsilon, \theta_m + \epsilon]$ and $[\theta_M - \epsilon, \theta_M + \epsilon]$ overlap, then from points in the intersection of these intervals the thermostat can make an infinite number of switchings between two modes in finite time. The requirement of non-Zenoness, which ensures the progress of time past any real number, comes from the fact that physical systems cannot be infinitely fast which implies that only a finite number of actions can be executed in a finite amount of time. Since trajectories for dynamical systems are supposed to be well-defined on $\mathbb{R}^n$, blocking and Zeno hybrid automata are inadequate models, which often result from modeling errors. In this thesis, when solving the verification problem we assume that *the hybrid automata we address are non-Zeno*[1]. We will discuss Zeno behaviors in Chapter 7 where we deal with the safety controller synthesis problem for hybrid systems. Since Zeno behaviors are mathematically possible but not physically achievable, when solving the synthesis problem, it is important that the automaton is non-Zeno, otherwise the synthesis algorithm might produce a controller which succeeds in avoiding bad states by preventing time from progressing.

2.5 Reachability Notions

The set reachable from a given set of states $\mathcal{F}$ by a hybrid automaton can be defined as the set of all states visited by the trajectories starting from states in $\mathcal{F}$. A trajectory of the hybrid automaton in the continuous state space can be thought of as a sequence of trajectory segments of continuous dynamics. Thus, for a clear understanding of the reachability notions of hybrid automata, it is convenient to consider first discrete and continuous systems separately.

2.5.1 Reachability of Automata

Consider an automaton $\mathcal{M} = (Q, \delta)$ of Definition 3.

[1] The reader is refered to [79] for the work on finding existence and uniqueness conditions under which all trajectories can be extended to infinite time.

We define the *successor* operator $\texttt{Post}$ for a set $F \subseteq Q$ as the set of states that can be reached from F *in one step*:

$$\texttt{Post}(F) = \{q' \mid \exists q \in F\ q' \in \delta(q)\}.$$

We also define the *predecessor* operator $\texttt{Pre}$ for F as the set of states from which F can be reached in *one step*:

$$\texttt{Pre}(F) = \{q \mid \exists q' \in F\ q' \in \delta(q)\}.$$

Hence, the set of states that can be reached from F is the least fixed point ϕ of the equation $\phi = F \cup \texttt{Post}(\phi)$ and can be computed as the limit of the recursion

$$\begin{aligned} P_0 &= F \\ P_{i+1} &= P_i \cup \texttt{Post}(P_i) \end{aligned}$$

Similarly, the set of states from which F can be reached is the least fixed point of $\phi = F \cup \texttt{Pre}(\phi)$ and can be computed using the iteration

$$\begin{aligned} P_0 &= F \\ P_{i+1} &= P_i \cup \texttt{Pre}(P_i) \end{aligned}$$

2.5.2 Reachability of Continuous Systems

We consider a continuous system $\mathcal{C}$ of Definition 5.

Definition 9 (Successors)
The successor operator $\delta_I : 2^X \to 2^X$ is defined for a subset F of $\mathcal{X}$ and a time interval $I \subseteq \mathcal{T}$ as

$$\delta_I(F) = \{\mathbf{x}' \mid \exists \mathbf{x} \in F\ \exists t \in I\ \mathbf{x} \xrightarrow{t} \mathbf{x}'\}.$$

Recall that the notation $\mathbf{x} \xrightarrow{t} \mathbf{x}'$ indicates that $\mathbf{x}'$ is the state reachable from $\mathbf{x}$ after exactly t time, in other words, $\mathbf{x}' = \xi_{\mathbf{x}}(t)$ where $\xi_{\mathbf{x}}$ is the trajectory of $\mathcal{C}$ starting from $\mathbf{x}$.

For simplicity, we use the notation δ_r for $\delta_{[r,r]}$ (states reachable after exactly r time) and $\delta_I(\mathbf{x})$ for $\delta_I(\{\mathbf{x}\})$. The reachable set from F is therefore $\delta_{[0,\infty)}(F)$ (all states reachable after any non-negative amount of time), denoted by $\delta(F)$.

Figure 2.6 illustrates the above notions. In this two-dimensional example, the shaded region represents $\delta_r(F)$, that is, the set of successors of F at time point r, and the set reachable within the time interval $[0, r]$ lies between two dotted lines and the outer boundary of F and of $\delta_r(F)$.

The set $\delta(F)$ can be thought of as the union of all trajectories over all possible initial points in F and is also known in the literature under the names *reach tube* [71] and *flow pipe* [122, 31].

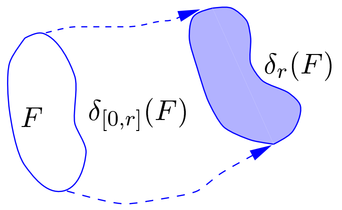


Figure 2.6: Successors of F by a continuous system.

Similarly, we define the predecessor operator. The predecessors of a set F are all the states from which F can be reached.

Definition 10 (Predecessors)
The predecessor operator $\Delta_I : 2^X \to 2^X$ is defined for a subset F of $\mathcal{X}$ and a time interval $I \subseteq T$ as

$$\Delta_I(F) = \{\mathbf{x} \mid \exists \mathbf{x}' \in F\ \exists t \in I\ \mathbf{x} \xrightarrow{t} \mathbf{x}'\}.$$

The operators δ and Δ satisfy the *semi-group property* [64], that is,

$$\delta_{I_2}(\delta_{I_1}(F)) = \delta_{I_1 \oplus I_2}(F)$$

where $\oplus$ is the Minkowski sum, and, in particular, $\delta_{[0,r_2]}(\delta_{[0,r_1]}(F)) = \delta_{[0,r_1+r_2]}(F)$.

2.5.3 Reachability of Hybrid Automata

We can now proceed with the reachability of hybrid automata. Consider a hybrid automaton $\mathcal{A} = (\mathcal{X}, Q, f, G, H, R)$ of Definition 7.

As we have already seen, there are two types of evolutions from a state $(q, \mathbf{x})$, namely the continuous evolution by the dynamics f_q and the discrete evolution by taking a transition. Accordingly, we define two types of successors: successors by continuous evolution, which we call *continuous-successors*, and successors by discrete evolution, which we call *discrete-successors.*

Definition 11 (Continuous-Successors)
Given a set of states (q, F) where $q \in Q$ and $F \subseteq \mathcal{X}$, we define the set of continuous-successors of (q, F), denoted by $\delta_c(q, F)$, as

$$\delta_c(q, F) = \{(q, \mathbf{x}') \mid \exists \mathbf{x} \in F\ \exists t > 0\ \mathbf{x} \xrightarrow{q,t} \mathbf{x}'\}.$$

Recall that the notation $\mathbf{x} \xrightarrow{q,t} \mathbf{x}'$, introduced in Section 2.4.2, indicates that $\mathbf{x}'$ is *q-reachable* from $\mathbf{x}$. Intuitively, this notation means that the trajectory segment of the dynamics f_q from $\mathbf{x}$ to $\mathbf{x}'$ lies inside H_q.

Figure 2.7 depicts the continuous-successors of a set (q, F) in the continuous state space. Unlike continuous systems, the behavior of the hybrid automaton at discrete state q is constrained by the staying conditions H_q, which are defined in this example by the half-space on the left-hand side of the vertical straight line. The dotted trajectory is thus impossible, and the point $\mathbf{z}$ is not reachable from F by the continuous dynamics f_q. However, the automaton admits the segment of this trajectory from $\mathbf{x}$ to $\mathbf{y}$. As a result, all the continuous-successors of (q, F) lie on one side of the vertical line.

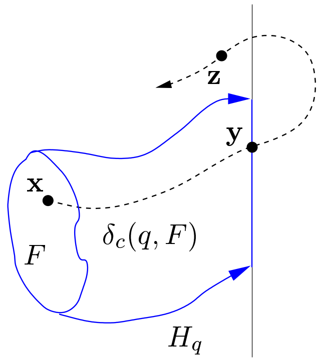


Figure 2.7: Continuous-successors of F by the dynamics f_q

Let $\mathcal{F}$ be a set of states defined as $\mathcal{F} = \{(q, F_q) \mid q \in Q \;\wedge\; F_q \subseteq \mathcal{X}\}$. We can naturally extend the above definitions to the continuous-successors of $\mathcal{F}$ as follows:

$$\delta_c(\mathcal{F}) = \bigcup_{q \in Q} \delta_c(q, F_q).$$

We turn now to discrete-successors.

Definition 12 (Discrete-Successors)
Given a transition from q to q' and a set of states (q, F) where $q, q' \in Q$ and $F \subseteq \mathcal{X}$, we define the set of discrete-successors of (q, F) with respect to the transition from q to q', denoted by $\delta_{qq'}(q, F)$, as

$$\delta_{qq'}(q, F) = \{(q', \mathbf{x}') \mid \exists \mathbf{x} \in F \cap G_{qq'} \;\wedge\; \mathbf{x}' \in R_{qq'}(\mathbf{x}) \cap H_{q'}\}.$$

The set $\delta_{qq'}(q, F)$ contains the states $(q', \mathbf{x}')$ where $\mathbf{x}'$ are the points in $H_{q'}$ resulting from applying the reset relation $R_{qq'}$ to the points $\mathbf{x}$ in F that satisfy the guard $G_{qq'}$.

To illustrate, consider the example shown in Figure 2.8 where the set F is the set of continuous-successors in the example of Figure 2.7. The set $H_{q'}$ is the half-space below the horizontal line, and the guard $G_{qq'}$ is the rectangle. Suppose that the continuous variables do not change after taking the transition from q to q'. The discrete-successors of (q, F) with respect to this transition can be obtained by intersecting F with $G_{qq'}$ and then with $H_{q'}$, which gives the shaded region in the figure.

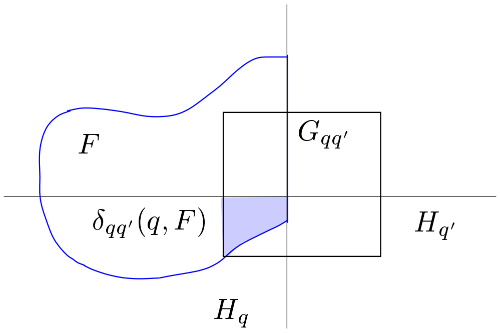


Figure 2.8: Discrete-successors of F by transition from q to q'.

The set of discrete-successors of (q, F) by executing all enabled transitions from q, denoted by $\delta_d(q, F)$, is

$$\delta_d(q, F) = \bigcup_{q' \in Q} \delta_{qq'}(q, F).$$

Hence, the set of discrete-successors for a set of states $\mathcal{F} = \{(q, F_q) \mid q \in Q \ \wedge \ F_q \subseteq \mathcal{X}\}$ is

$$\delta_d(\mathcal{F}) = \bigcup_{q \in Q} \delta_d(q, F_q).$$

We define now the operator δ for $\mathcal{F}$ as the set of states reachable from $\mathcal{F}$ by continuous dynamics and then discrete transitions:

$$\delta(\mathcal{F}) = \delta_d(\delta_c(\mathcal{F})).$$

Then, the reachable set from $\mathcal{F}$ by the automaton $\mathcal{A}$ is the limit of the recursion

$$\begin{aligned} P_0 &= \mathcal{F} \\ P_{i+1} &= \delta(P_i) \end{aligned}$$

The successor operators are basic ingredients in forward reachability analysis. For backward reachability, we introduce subsequently the *predecessor operators*.

Definition 13 (Continuous-Predecessors)
Given a set of states (q, F) where $q \in Q$ and $F \subseteq \mathcal{X}$, we define the set of continuous-predecessors of (q, F), denoted by $\Delta_c(q, F)$, as

$$\Delta_c(q, F) = \{(q, \mathbf{x}') \mid \exists \mathbf{x} \in F \ \exists t > 0 \ \mathbf{x}' \xrightarrow{q,t} \mathbf{x} \ \}.$$

Definition 14 (Discrete-Predecessors)
Given a set of states (q, F) where $q \in Q$ and $F \subseteq \mathcal{X}$, we define the set of discrete-predecessors of (q, F) with respect to the transition from q' to q, denoted by $\Delta_{q'q}(q, F)$, as

$$\Delta_{q'q}(q, F) = \{(q', \mathbf{x}') \mid \exists \mathbf{x} \in F \cap R_{q'q}(\mathbf{x}') \;\wedge\; \mathbf{x}' \in H_{q'} \cap G_{q'q}\}.$$

Then, the set of discrete-predecessors of (q, F) is

$$\Delta_d(q, F) = \bigcup_{q' \in Q} \Delta_{q'q}(q, F),$$

and the set of discrete-predecessors of $\mathcal{F} = \{(q, F_q) \mid q \in Q \;\wedge\; F_q \subseteq \mathcal{X}\}$ is

$$\Delta_d(\mathcal{F}) = \bigcup_{q \in Q} \Delta_d(q, F_q).$$

2.6 Other Hybrid System Models

Hybrid systems have been intensively studied by both computer science and control communities. However, the respective approaches to the modeling of hybrid systems are slighly different due to the different types of problems of interest and analysis/design techniques in these two disciplines.

The approach pursued by computer scientists is to extend traditional finite-state automata by introducing progressively more complex continuous dynamics. Timed automata [5] can be viewed as a very restricted class of hybrid automata in which the derivative of all continuous variables is 1. The first model which augmented discrete transition systems with variables governed by differential equations is the phase-transition system [84], from which the hybrid automaton model was derived. In [3] it was shown that the reachability problem for hybrid automata is undecidable, i.e. there is no general algorithm for any hybrid automaton. The research in the computer science approaches continued in two major directions. One was to build tools for classes of hybrid systems for which the verification problem is solvable (e.g. Kronos [120] and Uppaal [74] for timed automata) or semi-solvable (HyTech [56] for 'linear' hybrid automata[2]). The other was to find further restrictions on continuous and discrete dynamics that guarantee decidability. Several models along these lines are multirate timed automata [3], piecewise-constant derivative systems PCD [14], integration graphs [65], and rectangular hybrid automata [99]. In these models, essentially, the guard and staying sets are polyhedra and the vector fields are constant in every discrete state. These works gave important insights concerning the difficulties in exporting exact verification methodology to hybrid systems. In addition, the hybrid automaton model has been accepted as a working model by the control community.

[2] Linear hybrid automata should not be confused with hybrid automata where continuous dynamics are linear.

Control theorists, on the other hand, approach hybrid systems by incorporating discrete behaviors into their continuous dynamical descriptions, specifically ordinary differential equations (see [50, 27] and references from there). Other models which deal explicitly with continuous dynamics and discrete event dynamics consist of a continuous plant (represented by differential equations) supervised by a discrete controller (represensed by an automaton). Examples of such models are [45, 92, 8]. A more general hybrid model is proposed in [28], which considers a variety of hybrid behaviors exhibited in hybrid control systems within a unified dynamical setting. Surveys of the hybrid models developed from the control point of view can be found in [28, 27, 75] and a comparison between discrete and continuous dynamical systems in [83].

Part II

Verification

Chapter 3

Algorithmic Verification

3.1 Problematics

Having a formal model for hybrid systems and their behaviors, we need methods for proving that these systems behave as required. We will adopt the *algorithmic* verification methodology (also known as model checking [87]), which has been developed for discrete systems and applied successfully to digital circuits and communication protocols. In this thesis, we will concentrate on *invariance properties*, which are the simplest type of safety properties [86] and can be phrased as follows: no trajectory of the system should ever reach a certain subset $\mathcal{B}$ of the state space, or equivalently, the system will always stay in the complement of $\mathcal{B}$. For finite automata, there are two straightforward methods to verify invariance properties, namely by forward and backward reachability. The forward reachability method consists in starting with an initial set F of states and computing iteratively their successors until the set of all reachable states is computed (this is guaranteed to happen after a finite number of steps), and this set is then checked for intersection with $\mathcal{B}$. This is summarized by the following algorithm which makes use of the successor operator `Post`, introduced in the previous chapter.

Algorithm 1 (Forward Reachability)

$R^0 := F$;
repeat $k = 0, 1, 2, \ldots$
  if $(R^k \cap \mathcal{B} \neq \emptyset)$ **return** *unsafe*
  $R^{k+1} := R^k \cup \texttt{Post}(R^k)$;
until $R^{k+1} = R^k$
return *safe*

Backward reachability starts with the set $\mathcal{B}$, calculates iteratively its predecessors until convergence, and then checks whether the computed set intersects with the initial set F. The verification algorithm using backward reachability is given below (`Pre` is the predecessor operator).

Algorithm 2 (Backward Reachability)

> R^0 := $\mathcal{B}$;
> **repeat** $k = 0, 1, 2, \ldots$
>   **if** $(R^k \cap F \neq \emptyset)$ **return** *unsafe*
> $R^{k+1} := R^k \cup \mathtt{Pre}(R^k)$;
> **until** $R^{k+1} = R^k$
> **return** *safe*

The verification problem for any finite-state discrete system can be solved using either of the above algorithms since the transition function, the initial set F, the set $\mathcal{B}$, and the set of reachable states accumulated over the execution are finite and can be represented explicitly.

Now, if extended to hybrid automata, these algorithms involve the computation of the following functions over subsets of the state space of hybrid systems:

- Successors or predecessors: $Q \times 2^{\mathcal{X}} \rightarrow Q \times 2^{\mathcal{X}}$;
- Union and intersection: $2^{\mathcal{X}} \times 2^{\mathcal{X}} \rightarrow 2^{\mathcal{X}}$; and
- Emptiness checking: $2^{\mathcal{X}} \rightarrow \{0, 1\}$.

Set union is needed to accumulate the reachable states; emptiness checking and set intersection are needed to check whether the system reaches states in $\mathcal{B}$. Note that to detect the termination of the algorithms, i.e. when $R^{k+1} = R^k$, one can check emptiness of the set difference $R^{k+1} - R^k$.

In order to be able to compute these functions, the first ingredient we need is a finite syntactic representation of the sets encountered during the execution of the algorithms. The continuous state space $\mathcal{X}$ of hybrid automata is in $\mathbb{R}^n$, and hence, unlike in finite-state systems, subsets of $\mathcal{X}$ do not admit an enumerative representation and can only be represented *symbolically*, such as by formulas of some logic. Examples of classes of subsets of $\mathcal{X}$ which admit a symbolic representation are the polyhedral sets (represented by Boolean combinations of linear inequalities) and the semi-algebraic sets (represented by combinations of polynomial inequalities).

Another difficulty comes with the two-phase evolution of hybrid systems, which requires the ability to compute the successors or predecessors of sets of states not only by discrete transitions but also by continuous dynamics. In the continuous phase, this associates with the special problem of characterizing trajectories of continuous systems. For concreteness, we illustrate this problem by means of a hybrid automaton with only one discrete state whose staying set is the whole state space. Suppose that the initial set F can be characterized by a formula $\phi_F(\mathbf{x})$ whose truth value is 1 iff $\mathbf{x} \in F$, and similarly, the set $\mathcal{B}$ by a formula $\phi_{\mathcal{B}}(\mathbf{x})$. Suppose further that the differential equation $\dot{\mathbf{x}} = f(\mathbf{x})$ of the continuous dynamics admits a closed-form solution $\xi_{\mathbf{x}}(t)$ for every initial condition $\mathbf{x}$; hence the reachable set from F is exactly the set of $\mathbf{x}$ for which the formula

$$r(\mathbf{x}) = \exists \mathbf{x}' \ \phi_F(\mathbf{x}') \wedge \exists t \geq 0 \ \mathbf{x} = \xi_{\mathbf{x}'}(t)$$

is true. Similarly, proving that the system is safe amounts to proving that the formula

$$\forall \mathbf{x}' \ \phi_F(\mathbf{x}') \Rightarrow \ \forall t : t \geq 0 \ \neg \phi_{\mathcal{B}}(\xi_{\mathbf{x}'}(t)) \tag{3.1}$$

is true, which can be done by eliminating the quantifiers. If $\phi_{\mathcal{B}}$, ϕ_F, and $\xi_{\mathbf{x}}(t)$ are definable in a theory for which quantifier elimination is possible, then the problem can, in principle, be solved by symbolic manipulation of formulas [37].

When the derivative f is constant: $f(\mathbf{x}) = \mathbf{c}$, we have $\xi_{\mathbf{x}}(t) = \mathbf{x} + \mathbf{c}t$; the quantifiers in (3.1) can thus be eliminated using linear algebra. This is the basis for verification algorithms for classes of hybrid systems [120, 74, 56] in which the derivatives of continuous variables are constant, staying conditions and transition guards are specified as combinations of linear inequalities; hence, reachable sets are definable by linear formulas (see Figure 3.1 for an example).

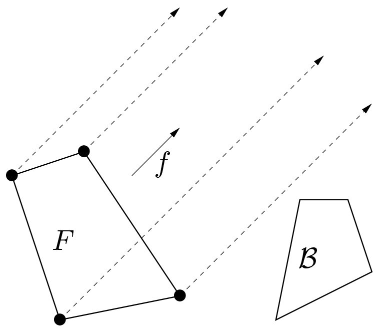


Figure 3.1: Computation of the reachable set of a one-state hybrid system with constant derivatives where the initial set F and the bad set $\mathcal{B}$ are polyhedral. Checking the intersection of the reachable set with $\mathcal{B}$ can be done by linear algebra.

For systems with non-trivial dynamics, the situation is much more complicated. First, in many cases we do not know explicit solutions of the differential equations. Furthermore, even when we know such solutions, their forms may not allow a general method for proving equation (3.1). For example, for linear systems $\dot{\mathbf{x}} = A\mathbf{x}$ we have a closed-form solution $\xi_{\mathbf{x}}(t) = e^{At}\mathbf{x}$, but a proof of (3.1) is possible only for a very restricted class of matrices. Recent results concerning the applicability of algebraic manipulation techniques for the reachability analysis of hybrid systems with linear continuous dynamics appear in [94, 7].

In addition to the problem of characterizing the states reachable in one continuous phase, applying Algorithms 1 or 2 to hybrid automata may result in a computation which alternates indefinitely between two or more discrete states, each time adding more and more successors. Figure 3.2 depicts the reachable set computation for a 4-state PCD (piecewise-constant derivative) system [14] where $\mathcal{X} = \mathbb{R}^2$ and the staying conditions of the discrete states are the disjoint rectangles. The system starts from the line segment F at discrete state q_1. One can see from the figure that for this system the reachability algorithm does not terminate. It has been proved that even for simple systems with constant derivatives, where the computation of continuous-successors in each discrete state can be done exactly, there is no general reachability algorithm which is guaranteed to terminate [54]. In other words, the reachability problem for hybrid automata is undecidable. The reader might wish to consult [7] for a survey of the decidability results.

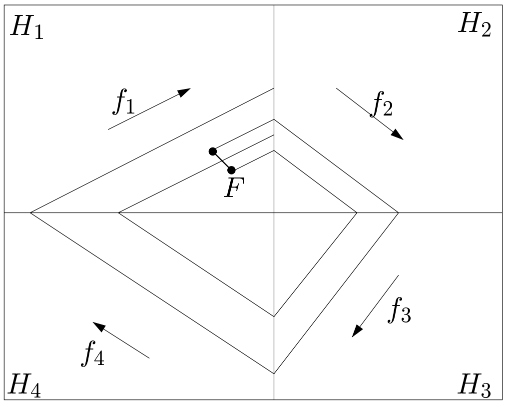


Figure 3.2: An example of a non-terminating computation of reachable states.

We now turn to approximate methods. Numerical simulation is a powerful tool for approximating solutions of differential equations for a given initial condition. Starting from a single point $\mathbf{x}$ in the state space, one can numerically integrate the differential equation to obtain an approximation $\hat{\xi}_{\mathbf{x}}$ of the solution $\xi_{\mathbf{x}}$. Although the approximate solution is computed only at discrete time points, the distance between an approximate value $\hat{\xi}_{\mathbf{x}}(t)$ and the true values at $\xi_{\mathbf{x}}(t')$ for $t' \in [t - \epsilon, t + \epsilon]$ can be bounded. Hence, if we want to verify whether the trajectory starting from a point $\mathbf{x}$ reaches $\mathcal{B}$, we can, in principle, simulate the trajectory forward and check at every step if the approximate solution is close to $\mathcal{B}$. Termination is, however, not guaranteed since trajectories of continuous systems are not always periodic. Although simulation techniques are very useful for simulating *single* trajectories, they are less so when it comes to deal with *sets of trajectories* arising whenever the initial condition of the system is specified as a *set of initial states* rather than a single state, or when the continuous dynamic is influenced by *under-specified inputs.* This is, in fact, the major difference between simulation/testing and verification.

The conclusion from the discussion thus far is that the main obstacle towards extending the algorithmic verification methodology outside the world of discrete systems or hybrid systems with trivial continuous dynamics is the lack of *effective methods for characterizing reachable sets of continuous dynamics.* In the next section, we propose a framework for algorithmic analysis which, despite the theoretical difficulty, allows practical approximate solutions to verification and synthesis problems.

3.2 Approach to Solution

3.2.1 Representation of Sets

Given the difficulties in computing exactly reachable sets of hybrid automata, we resort to approximating them by polyhedra. The reason we choose polyhedra as the *symbolic representation of sets in* $\mathbb{R}^n$ is that, from the computational point of view, they are among the geometric objects which are easier to describe and manipulate. We replace all the operations on the real sets in the abovementioned verification algorithms by operations on their polyhedral approximations and compute a sequence of polyhedra P^k approximating R^k. If over-approximations are used and the algorithm terminates, then the result is an over-approximation of the reachable set. Nevertheless, even with polyhedral approximations the problems of effectiveness and termination are not completely resolved.

First, termination is still not guaranteed since there are infinitely many polyhedral sets, even in a bounded subset of $\mathbb{R}^n$. Second, the sets of reachable states, which are iteratively computed by the verification algorithms, may have complicated forms and their approximating polyhedra might be hard to represent and manipulate. Polyhedral sets can be divided into two types: convex and non-convex. The former are simpler and admit canonical representations, namely the dual vertex-based and constraint-based representations, which allow efficient implementations of intersection, membership, and equivalence testing. However, if we restrict ourselves to convex polyhedra, whenever we make a union of two sets, we need to approximate it by their convex hull, which might result in a too coarse approximation. Consequently, we need to use non-convex polyhedra, which are much more complex objects.

In two dimensions, any polygon can be uniquely defined by an ordered list of its vertices, but the treatment of non-convex polyhedra in higher dimensions is, unfortunately, very complicated. Non-convex high dimensional polyhedra constitute a challenging object of computational geometry. The topological representation of polygons can be generalized to graphs of incidence and adjacency between vertices, ridges, and faces [117], yet with a lot of requirements for a polyhedron to be well-defined. Still, data structures and algorithms based on this representation are very sophisticated when it comes to deal with degeneracy and become inefficient as the dimension grows. Alternatively, non-convex polyhedra can also be represented by unions of convex polyhedra[1], or by series of Boolean operations on convex polyhedra, such as CSG[2] [69] and Octree [103]. Although these representations have proved successful in two- or three-dimensional applications, such as computer graphics and solid modeling, they are not appropriate for higher dimensions since geometric operations become

[1]In the verification tools Kronos [120] and Uppaal [74] for timed automata, reachable sets are also represented by lists of simple convex polyhedra which can be written as conjunctions of inequalities of the form $x_i - x_j \leq b$.

[2]CSG stands for Constructive Solid Geometry.

prohibitively expensive.

For these reasons, we restrict ourselves further to orthogonal polyhedra which can be described as unions of closed full-dimensional hyper-rectangles with rational coordinates. The justification for this choice is that for orthogonal polyhedra we have a compact and, moreover, canonical representation, which allows relatively efficient manipulation including Boolean operations, equivalence checking, and all other geometric operations. However, while the use of orthogonal polyhedra makes algorithm design easier, a price for this is that the quality of the approximation is poorer.

It should be noted that approximating *sets* of states for verification or controller synthesis purposes is different in nature from approximating single points or trajectories, as is done in numerical simulation. The goal of simulation is to ensure that the approximating object is close enough to the real one. However, in set-based approximation, we often want to guarantee that the approximating set contains the real one (over-approximation) or, in some cases, is contained in the real one (under-approximation). We will show later that orthogonal polyhedra are also suitable for these purposes.

To summarize, in order to give an approximate solution to the verification problem for hybrid systems, we intend to represent sets of reachable states using non-convex orthogonal polyhedra. The main challenge in the implementation of the verification algorithms is to find techniques for approximating the continuous-successor operator. This requires *the ability to compute successors for purely continuous systems*, which is the major problem we attack in this thesis. In the rest of this section, we give the key ideas of the orthogonal polyhedron representation, which will help the reader to understand the reachability techniques proposed in the following chapters. For more details on the representation of orthogonal polyhedra, the reader is refered to [26, 25].

Orthogonal Polyhedron Representation

Without loss of generality, we assume that we are working in the set $[-M, M]^n$ for some integer $M > 0$. With every rational number $\beta \geq 0$ we define the corresponding uniform grid of size β as the set of *grid points* $\mathcal{G}_\beta = \{(z_1\beta, \dots, z_n\beta) \mid z_i \in \mathbb{Z}\}$ (see Figure 3.3-(a)).

With every grid point $\mathbf{v} = (v_1, \dots, v_n)$ we associate an *elementary hyper-cube* $g(\mathbf{v}) = [v_1, v_1 + \beta] \times \dots \times [v_n, v_n + \beta]$, namely the hyper-cube of size β whose leftmost corner is $\mathbf{v}$. The set of all such hyper-cubes is denoted by Ω_β. An *orthogonal polyhedron* is any subset of Ω_β (see Figure 3.3-(b)).

A more general class of orthogonal polyhedra can be obtained by using *non-uniform grids*. Such polyhedra can be thought of as unions of arbitrary axis-parallel hyper-rectangles (see Figure 3.3-(c)).

A representation scheme for a class of objects (e.g. orthogonal polyhedra) is canonical if each representation corresponds to at most one object and each object has a unique representation. A compact canonical representation makes the checking of equivalence between polyhedra simple. For example, the representation of convex polyhedra by the set of their vertices

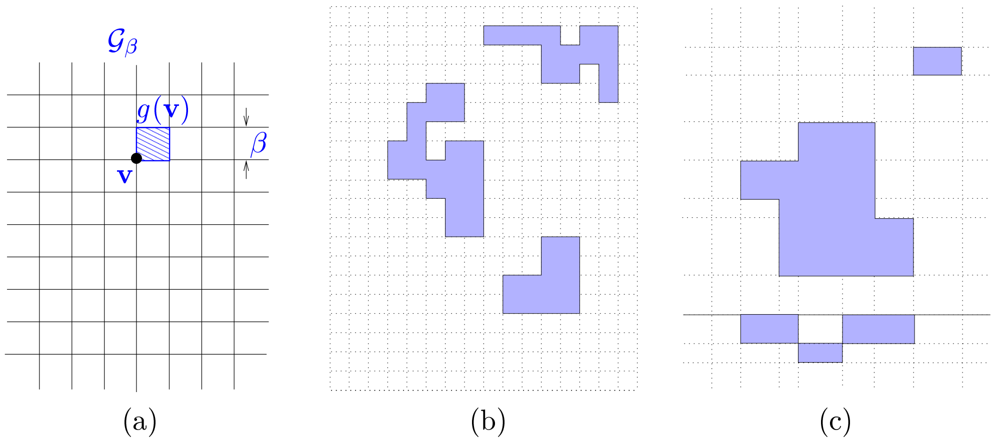


Figure 3.3: (a) A uniform grid $\mathcal{G}_\beta$; (b) an orthogonal polyhedra on a uniform grid; (c) an orthogonal polyhedra on a non-uniform grid.

is canonical: every set of vertices defines exactly one convex polyhedron and every convex polyhedron has a unique set of vertices. However, for non-convex polyhedra, a set of vertices is not a representation at all since two different polyhedra can have the same set of vertices (see Figure 3.4). If we represent non-convex polyhedra as unions of convex ones, we may have more than one representation of the same polyhedron. In the sequel we discuss a vertex-based canonical representation scheme for orthogonal polyhedra, developed in [26].

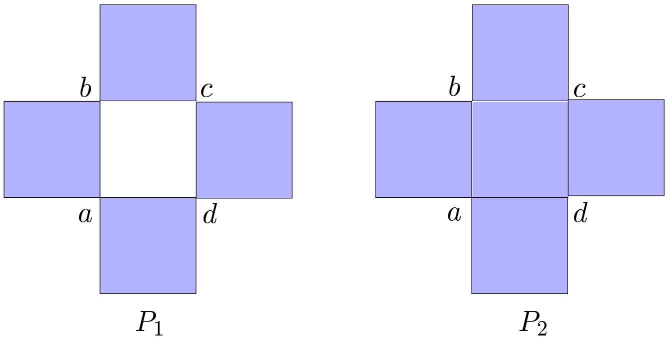


Figure 3.4: The polyhedra P_1 and P_2 are distinct but have the same vertices.

Given a grid point $\mathbf{v}$ and an orthogonal polyhedron P, we say the *color* of $\mathbf{v}$ is *black* if $g(\mathbf{v}) \in P$ and *white* otherwise. We define the *neighborhood* $\mathcal{N}(\mathbf{v})$ of a vertex $\mathbf{v}$ of P as the set of the vertices of the elementary hyper-cube lying between $(v_1 - 1, \ldots, v_n - 1)$ and $\mathbf{v}$. A vertex is called *extreme* if its neighborhood contains an odd number of black points. Figure 3.5 illustrates the above notions. The color of vertex $\mathbf{v}$ is black and that of $\mathbf{v}'$ is white. The neighborhood of grid point $\mathbf{v}$ is $\mathcal{N}(\mathbf{v}) = \{\mathbf{v}_1, \mathbf{v}_2, \mathbf{v}_3, \mathbf{v}\}$.

It has been proved in [26] that *an orthogonal polyhedron is uniquely represented by the set of*

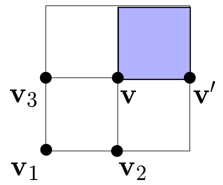


Figure 3.5: Orthogonal polyhedron notions.

its extreme vertices. As an example, consider again the polyhedra of Figure 3.4. It is easy to see that all the vertices of P_2 are extreme while the vertices a, b, c, and d of P_1 are not. Hence, P_2 is represented by all its vertices and P_1 by all its vertices except a, b, c, d. Based on this canonical representation, algorithms for Boolean operations, membership testing, equivalence checking as well as other geometric operations, such as face detection, were developed and reported in [26, 25].

3.2.2 Reachability Analysis of Continuous Systems

Having chosen a representation scheme for sets of states, the remaining problem is to find techniques for computing reachable sets of continuous systems using this representation. We first formally state the problem and then present a basic reachability algorithm.

Basic Computation Procedure

Consider a continuous system $C = \{\mathcal{X}, f\}$, as in Definition 5. The reachability problem we consider is stated as follows.

Problem 1 *Given a set $F \subseteq \mathcal{X}$, we want find an approximation of the set $\delta(F)$ of states reachable from F by $\mathcal{C}$.*

Numerical integration is a common method to approximate solutions of differential equations. The basic idea of this approach is the following. For a given initial condition $\mathbf{x}(0) = \mathbf{x}_0$, the solution of the differential equation of $\mathcal{C}$ can be written as

$$\xi_{\mathbf{x}_0}(t) = \mathbf{x}_0 + \int_0^t f(\xi_{\mathbf{x}_0}(s))ds. \tag{3.2}$$

Given a time step $r > 0$ and a sequence $0, r, 2r, \ldots,$ we denote by $\mathbf{x}^k$ a numerical estimate of the exact solution $\xi_{\mathbf{x}_0}(kr)$, $k = 0, 1, 2, \ldots.$ We can then obtain an approximate solution using the following rescursive scheme:

$$\mathbf{x}^{k+1} = \mathbf{x}^k + \Gamma(\mathbf{x}^k, r). \tag{3.3}$$

where Γ is an approximation of the integral in (3.2). The main concern of this approach is to find the time step r and Γ with respect to the desired accuracy and computational cost, and this leads to numerous advanced schemes, such as multistep and Runge-Kutta [62].

This method, although not adequate for solving Problem 1 since our interest is to compute solutions for all points in the initial set and for all time points $t \geq 0$, inspires us to approximate reachable sets on a step-by-step basis using the following iterative algorithm.

Algorithm 3 (Computation of $\delta(F)$)

$P^0 := F$;
repeat $k = 0, 1, 2, \ldots$
  $P^{k+1} := P^k \cup \delta_{[0,r]}(P^k)$;
until $P^{k+1} = P^k$

The set P^k here is the set of states reachable from F during the interval $[0, kr]$, i.e. $\delta_{[0,kr]}(F)$. The algorithm terminates whenever no new reachable states are found. Algorithm 3 can be easily extended to account for variable time steps.

To be effective, we use orthogonal polyhedra to represent the sets encountered in Algorithm 3 and replace all the operations with their approximate versions on orthogonal polyhedra. Since orthogonal polyhedra are closed under the union operation, only $\delta_{[0,r]}$ needs to be approximated. Note also that, using orthogonal polyhedra, the set of reachable states accumulated over the execution is represented as a unique object, and hence termination checking can be done efficiently. This also proves the advantage of orthogonal polyhedra over arbitrary polyhedra or ellipsoids since there is no easy way for deciding if a union of convex polyhedra or ellipsoids is included into another. Moreover, the use of orthogonal polyhedra may guarantee the termination of the algorithm if we analyze the system only in a bounded subset of the state space, which can be represented as a union of a finite number of orthogonal polyhedra. This issue will be discussed in more detail in the next chapters.

Nonetheless, using orthogonal polyhedra to approximate smooth sets, it is clear that we cannot avoid approximation errors. No matter which method to compute $\delta_{[0,r]}$ is proposed, the following question about error accumulation must be answered: "How does the error in each iteration affect the global error in the obtained result?".

It is important to emphasize that the global error is the distance between the exact solution and the approximate one and should not be confused with the local error, namely the error incurred when we compute P^{k+1} from P^k under the assumption that P^k is 'exact'. It is not sufficient to maintain the local error at less than a given tolerance since P^k itself contains errors, which may propagate to the next iterations, and consequently the global error can grow over the execution. Therefore, another desirable property of the approximation scheme is that *the error in each iteration does not propagate.*

Let us examine, in a rather general way, how the error accumulation phenomenon is manifested when using orthogonal polyhedra (and over-approximation in general) in Algorithm 3. Consider the example shown in Figure 3.6 where we want to over-approximate the set of states reachable from the box D_0. The exact set is shown as the shaded region. Since $\mathbf{x}'$ is reachable from $\mathbf{x}$, we must include the box D_1 (which is an elementary hyper-cube of the underlying grid) in the set of successors. This box contains points, such as $\mathbf{y}$, not reachable from D_0, which bring in the next iteration new points, such as $\mathbf{y}'$; as a result, we end up adding the box D_2' which contains no reachable points from D_0 at all.

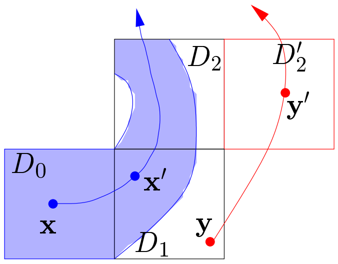


Figure 3.6: Illustration of the over-approximation error accumulation.

The phenomenon is not specific to orthogonal polyhedra (although it is aggravated by the coarseness of the approximation). Similar phenomena are exhibited, for example, in abstract interpretation of programs over the integers [35] where over-approximation is called *widening*.

The intuitive reason for the error accumulation is that Algorithm 3 advances the computation from kr to $(k+1)r$ using P^k as the new initial set and thus includes in P^{k+1} the 'fake' successors in P^k. Even though we can control the local error, the global error can still grow if the continuous dynamics f by its very nature expands regions. Note that, by the semi-group property, computing successors of $\delta_{[0,kr]}(F)$ is equivalent to computing successors of $\delta_{kr}(F)$, or more precisely,

$$\delta(\delta_{[0,kr]}(F)) = \delta(\delta_{kr}(F)) \cup \delta_{[0,kr]}(F).$$

Thus, one can employ δ_{kr} as the basis for the computation in the next iterations. This may be helpful in reducing the accumulation error effect because one might expect that there is more accumulated error in the approximation of $\delta_{[0,kr]}$ than in that of δ_{kr}. This solution, however, requires the additional computation of δ_{kr}, which is not more feasible except for linear systems, as we will see in the next chapter. For non-linear systems, this can still be done yet with much sophistication.

In the following two chapters, we present two techniques for over-approximating reachable sets of continuous systems based on Algorithm 3. One technique is specialized for linear systems, and the other can be used for non-linear systems. The advantage of both techniques is that they can be easily adapted to the verification of hybrid systems.

Before proceeding, we discuss briefly some other approaches reported in the literature.

3.3 Other Approaches

The standard algorithmic approaches to the verification of hybrid systems can be divided into two categories: direct and indirect. The *direct* approach works *directly* on the continuous state space of the system (as in this thesis). Examples are the verification algorithms implemented in the tools Kronos [120], Uppaal [74], and HyTech [56]. Recent works in [24, 59], which we will outline in Chapter 8, also solve the verification problem for hybrid systems with more complex continuous dynamics in a direct manner.

The *indirect* approach, on the other hand, reduces first the system, via abstraction, to a *finite-state automaton*. The abstraction procedure consists in finding a finite partition of the state space such that the reachability between partition blocks is faithfully described by the transition relation of the automaton (see [7] for more details on this issue). Once the finite abstraction has been constructed, the verification can be performed on the abstracted system with a termination guarantee, using standard tools for finite-state systems. Nevertheless, it was shown that such a finite abstraction exists only for restricted classes of hybrid systems either with simple continuous dynamics [5, 4, 93] or with simple discrete dynamics [73]. Even when a given class of systems is proved to admit a finite quotient, another difficulty comes with the problem of actually computing the quotient, which requires calculating successors of every block in the partition and is consequently at least as hard as the verification problem. However, it should be noted that this approach is important for proving correctness and termination of the verification algorithms for some classes of systems [5, 54] and allows to verify general temporal logic properties.

Alternatively, other works consider discrete approximations, that is, instead of exact finite quotient, they search for a finite discrete system whose behaviors include all behaviors of the original system. Recently, in [31], discrete approximations are done by iteratively refining state partitions using an approximate reachability method for continuous dynamics, which is similar in some aspects to the method we propose in the next chapter. In [95] the author developed an analysis technique for piecewise linear systems using a geometric model describing switching boundaries and some algorithms for mapping the patterns of trajectory movements between boundaries into a node graph.

An approach which can be viewed as a mixture of the direct and indirect approaches consists in deriving from the original system an approximate system for which verification algorithms and tools are available [101, 57, 107, 109]. The approximate system can be generated by over-approximating the complex continuous dynamics with simple dynamics (such as constant slopes and rectangular inclusions) based on discretizations of the continuous state space. The main drawback of this approach is that, in addition to the considerable effort of initially abstracting continuous dynamics with respect to the desired accuracy, the size of the resulting system might be prohibitively large for the verification algorithms.

Besides the algorithmic approaches, the *deductive* approaches have also been used [86]. Deductive methods involve proving a property by induction based on a set of axioms and inference rules. These approaches are often aided by theorem provers (e.g. [22]) and can verify a more general class of systems. However, unlike the algorithmic approaches, they are not automatic and require human intelligence in the process of finding proofs.

Optimal control is another approach to the verification of hybrid systems, mostly used by the control community. Instead of exploring all trajectories of the system, as is done in the algorithmic methods, one can turn the verification problem into an equivalent optimal control problem, that is, finding the worst possible trajectory with respect to the property to be verified [100, 78]. Nevertheless, solving optimal control problems for hybrid systems is non-trivial, both analytically and computationally.

Chapter 4

Reachability Analysis of Linear Continuous Systems

Linear dynamical systems have been extensively studied by the control community for years because they have a rich structure and provide sufficiently good models to design controllers for a broad class of processes. The theory of linear systems is well developed and has been successfully used in practice. In this chapter we develop a reachability technique for linear continuous systems. Although this technique is less efficient than classical methods for analyzing linear continuous systems, its main advantage is its straighforward adaptation to verification and controller synthesis for hybrid systems.

An outline of the chapter is as follows. After some preliminaries, we describe an algorithm for approximating reachable sets of linear systems and then study the error in the approximation. We next show how this technique can be extended to linear systems with uncertain input. This chapter is a review of the results presented in [10].

Preliminaries

As the metric for our approximations, we will use the Hausdorff distance [41], which is a good measure for difference between sets. We give below some basic definitions and properties.

Let $\mathbf{x}$, $\mathbf{y}$ be two points in $\mathbb{R}^n$ and X, Y be two subsets of $\mathbb{R}^n$. We denote by $\langle \mathbf{x}, \mathbf{y} \rangle$ the scalar product of $\mathbf{x}$ and $\mathbf{y}$. Let B be the unit ball at the origin: $B = \{\mathbf{x} \mid \langle \mathbf{x}, \mathbf{x} \rangle \leq 1\}$. The Minkowski sum of X and Y is defined as $X \oplus Y = \{\mathbf{x} + \mathbf{y} \mid \mathbf{x} \in X,\ \mathbf{y} \in Y\}$. For $\epsilon \in \mathbb{R}$, $\epsilon X = \{\epsilon \mathbf{x} \mid \mathbf{x} \in X\}$. The set $N(X, \epsilon) = X \oplus \epsilon B$ is called the *ϵ-neighborhood* of X.

Definition 15 (The Hausdorff distance)

1. *The distance between* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined as* $d(\mathbf{x}, \mathbf{y}) = ||\mathbf{x} - \mathbf{y}||$ *where* $||\cdot||$ *is the Euclidian norm, or equivalently,* $d(\mathbf{x}, \mathbf{y}) = (\langle \mathbf{x} - \mathbf{y}, \mathbf{x} - \mathbf{y} \rangle)^{\frac{1}{2}}$.
2. *The diameter of* X *is defined as* $\varrho(X) = \sup\{d(\mathbf{x}, \mathbf{y}) \mid \mathbf{x}, \mathbf{y} \in X\}$. *In other words,* $\varrho(X)$

is the maximal distance between any two points in X.

3. *The Hausdorff semi-distance from* X *to* Y *is defined as*

$$h_+(X,Y) = \inf\{\epsilon \mid X \subseteq N(Y,\epsilon)\},$$

or equivalently,

$$h_+(X,Y) = \sup_{\mathbf{x}\in X} \inf_{\mathbf{y}\in Y} \{d(\mathbf{x},\mathbf{y})\}.$$

4. *The Hausdorff distance between* X *and* Y *is*

$$h(X,Y) = \sup\ \{h_+(X,Y), h_+(Y,X)\}.$$

If X and Y are closed sets, inf and sup in the above definitions can be substituted by min and max, respectively. It is well-known that the Hausdorff distance has the following properties [41].

Lemma 1

(h1) Given sets X, Y, Z *in* $\mathbb{R}^n$, $h(X,Z) + h(Z,Y) \geq h(X,Y)$.

(h2) Given sets X_1, X_2, Y_1, Y_2 *in* $\mathbb{R}^n$, *if* $h(X_1,Y_1) \leq \epsilon$ *and* $h(X_2,Y_2) \leq \epsilon$ *then*

$$h(X_1 \cup X_2, Y_1 \cup Y_2) \leq \epsilon.$$

Here and further the term 'distance' refers to the Hausdorff distance (if not explicitly stated otherwise). We also introduce the following notation [52] which will be used in error estimation.

Definition 16 (Asymptotic notation) *Let* x *be a real variable tending to some limit. Let* $v(x)$ *be a positive function and* $u(x)$ *any function. Then, we write* $u = \mathcal{O}(v)$ *if there exists a constant* b *such that* $|u(x)| < bv(x)$ *for all values of* x.

4.1 Computation Procedure

Consider a continuous linear system $\mathcal{C} = \{\mathcal{X}, f\}$ where $\mathcal{X} \subseteq \mathbb{R}^n$. The dynamics of $\mathcal{C}$ is described by the linear differential equation

$$\dot{\mathbf{x}} = A\mathbf{x} \tag{4.1}$$

where A is an $n \times n$ matrix.

For a given initial set $F \subseteq \mathcal{X}$, our goal is to find an orthogonal polyhedron over-approximating the reachable set $\delta(F)$. We begin by stating an important property of linear systems.

Lemma 2 *If F is a convex set, then for every $t \geq 0$ the set $\delta_t(F)$ of states reachable from F at time point t is convex.*

Proof
Let $\xi_{\mathbf{x}} : \mathcal{T} \to \mathcal{X}$ be the trajectory of $\mathcal{C}$ starting from a point $\mathbf{x} \in \mathcal{X}$. By solution of equation (4.1), we have

$$\xi_{\mathbf{x}}(t) = e^{At}\mathbf{x}. \tag{4.2}$$

Therefore, the set $\delta_t(F)$ can be written as $\delta_t(F) = e^{At}F$. The matrix exponential e^{At} is a linear operator [61] and hence preserves convexity, which implies that $\delta_t(F)$ is convex. ∎

We suppose further that the initial set F is a *convex bounded polyhedron*. Then, F can be written as $F = conv(V)$ where *conv* denotes the convex-hull operator and $V = \{\mathbf{v}_1, \ldots, \mathbf{v}_{m_v}\}$ is a finite set of vertices. Note that the successor of a single point can be easily computed either by matrix exponentiation, as shown in (4.2), or by numerical integration. Thus, to determine $\delta_t(F)$, it suffices to compute the set of successors at time t of the vertices of F, that is, $\{\delta_t(\mathbf{v}_1), \ldots, \delta_t(\mathbf{v}_{m_v})\}$, and then

$$\delta_t(F) = conv\{\delta_t(\mathbf{v}_1), \ldots, \delta_t(\mathbf{v}_{m_v})\}. \tag{4.3}$$

Figure 4.1 illustrates the above computation in two dimensions. If we were working in discrete

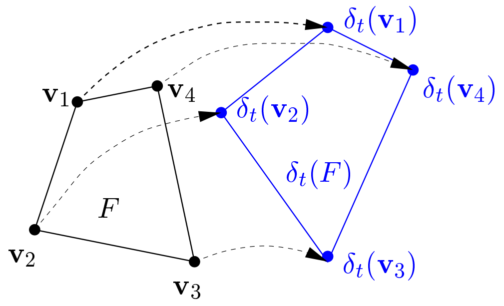


Figure 4.1: Illustration of the computation of δ_t for a convex polygon F with vertices $\{\mathbf{v}_1, \ldots, \mathbf{v}_4\}$.

time, this computation would be sufficient. We want, however, to compute all the states reachable at *any* time $t \geq 0$. In the sequel we present a technique for over-approximating $\delta(F)$.

4.1.1 Approximation Scheme

We consider the following sub-problem.

Problem 2 (Over-approximating $\delta_{[0,r]}$)
Given a convex polyhedron F and a time step $r \geq 0$, find an orthogonal over-approximation of $\delta_{[0,r]}(F)$, denoted by $\widehat{\delta}_{[0,r]}(F)$.

Note that the solution can be easily generalized to problems where the initial sets are non-convex polyhedra since these can be decomposed into finitely many convex polyhedra.

We have just shown that $\delta_r(F)$ can be effectively computed using (4.3), and we will exploit this to over-approximate $\delta_{[0,r]}(F)$. To begin, we make some preliminary observations. An obvious approximation of $\delta_{[0,r]}(F)$ can be obtained by taking the convex hull $C = conv(F \cup \delta_r(F))$. The polyhedron C is, in general, neither a subset nor a superset of $\delta_{[0,r]}(F)$. The problem is then to find a neighborhood of C that is guaranteed to include all the states reachable within the time interval $[0, r]$.

Let $\xi_{\mathbf{x}} : \mathcal{T} \to \mathcal{X}$ be the trajectory starting from an arbitrary point $\mathbf{x} \in F$, and let $\mathbf{y} = \xi_{\mathbf{x}}(r)$ be the point reachable from $\mathbf{x}$ at time r. The line segment from $\mathbf{x}$ to $\mathbf{y}$ can be thought of as an approximation of $\xi_{\mathbf{x}}(t)$ for $t \in [0, r]$ by a linear interpolation $s_{\mathbf{x}} : \mathcal{T} \to \mathcal{X}$ defined as

$$s_{\mathbf{x}}(t) = \mathbf{x} + \frac{t}{r}(\xi_{\mathbf{x}}(r) - \mathbf{x}),\ t \in [0, r].$$

Let S be the set of all these line segments:

$$S = \{s_{\mathbf{x}}(t) \mid \mathbf{x} \in F \ \wedge \ t \in [0, r]\}.$$

Note that the set S is often a curved object. Since $\mathbf{x} \in F$ and $\mathbf{y} \in \delta_r(F)$, by convexity we have $S \subseteq C$. Similarly, the reachable set $\delta_{[0,r]}(F)$ can be written as

$$\delta_{[0,r]}(F) = \{\xi_{\mathbf{x}}(t) \mid \mathbf{x} \in F \ \wedge \ t \in [0, r]\}.$$

Theorem 1 *Given a time step $r \geq 0$, there exists $\vartheta(r)$ which is $\mathcal{O}(r^2)$ such that $\delta_{[0,r]}(F) \subseteq N(C, \vartheta(r))$.*

Proof
Since $S \subseteq C$, we have $N(S, \vartheta(r)) \subseteq N(C, \vartheta(r))$; hence, to prove the theorem, we will prove that $\delta_{[0,r]}(F) \subseteq N(S, \vartheta(r))$. To this end, we estimate the distance between $\delta_{[0,r]}(F)$ and S. Using Lemma 1-(h2), $h(\delta_{[0,r]}(F), S)$ is the upper bound on the distance between $\xi_{\mathbf{x}}(t)$ and $s_{\mathbf{x}}(t)$ for every $\mathbf{x} \in F$ and for every $t \in [0, r]$. This distance is written as

$$||s_{\mathbf{x}}(t) - \xi_{\mathbf{x}}(t)|| = ||\mathbf{x} + \frac{t}{r}(e^{Ar} - I)\mathbf{x} - e^{At}\mathbf{x}||.$$

By Taylor's theorem

$$e^{At} = I + At + \frac{1}{2}A^2t^2 + \sum_{i=3}^{\infty} \frac{1}{i!}A^i t^i.$$

We find after obvious simplifications

$$||s_{\mathbf{x}}(t) - \xi_{\mathbf{x}}(t)|| = ||[\frac{1}{2}A^2 t(r - t) + \sum_{i=3}^{\infty} \frac{1}{i!}A^i t(r^{i-1} - t^{i-1})]\mathbf{x}||.$$

Let M be the constant bounding the norm $||\mathbf{x}||$. Then, the following inequality holds for all $t \in [0, r]$:

$$||s_{\mathbf{x}}(t) - \xi_{\mathbf{x}}(t)|| \leq \frac{1}{8} M ||A||^2 r^2 + \mathcal{O}(r^3) = \vartheta(r). \tag{4.4}$$

This means that $\delta_{[0,r]}(F) \subseteq N(S, \vartheta(r))$, and consequently $\delta_{[0,r]}(F) \subseteq N(C, \vartheta(r))$. Formula (4.4) also shows that $\vartheta(r)$ is indeed $\mathcal{O}(r^2)$. This completes the proof of the theorem. ■

Theorem 1 suggests the following scheme to solve Problem 2.

Scheme 1 (Over-approximating $\delta_{[0,r]}(F)$)

1. Compute $\delta_r(F)$.
2. Compute $C = conv(F \cup \delta_r(F))$.
3. Find the neighborhood $N(C, \vartheta(r))$ with $\vartheta(r)$ from Theorem 1.
4. Find an orthogonal polyhedron over-approximating $N(C, \vartheta(r))$.

Although $N(C, \vartheta(r))$ is already an over-approximation of $\delta_{[0,r]}(F)$, the goal of the last step is to represent the reachable set after successive iterations succintly as a unique orthogonal polyhedron, so that termination checking can be done efficiently.

We show now how to implement Scheme 1. The first step can be done, as mentioned earlier, by numerical integration or matrix exponentiation, and the second can be done using standard convex-hull algorithms. Let us proceed with the third step.

Constructing $N(C, \vartheta(r))$

The set $N(C, \vartheta(r)) = C \oplus \vartheta(r)B$ is not polyhedral, and in order to stay in the world of polyhedra we find a polyhedral over-approximation of $N(C, \vartheta(r))$. For doing this, we define the operator *bloat* as follows.

Definition 17 *Let C be a convex polyhedron $C = \bigcap_{i=0}^{m_c} \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i\}$, and let d be a real number. We define*

$$bloat(C, d) = \bigcap_{i=0}^{m_c} \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq b_i + d||\mathbf{a}_i||\}.$$

In words, $bloat(C, d)$ is the convex polyhedron resulting from pushing outward the half-spaces of C by the amount d. As an example, consider the polygon C shown in Figure 4.2, where the enveloping polygon represents $bloat(C, d)$ and the boundary of the neighborhood $N(C, d)$ is the curved line.

It can be easily shown that $bloat(C, \vartheta(r))$ is indeed an over-approximation of $N(C, \vartheta(r))$ and the distance between C and $bloat(C, \vartheta(r))$ is bounded by $\sqrt{n}\vartheta(r)$.

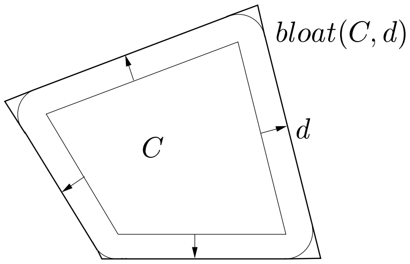


Figure 4.2: Illustration of $bloat(C, d)$.

Orthogonal Approximation

The last step of Scheme 1 is the over-approximation of a convex polyhedron by an orthogonal polyhedron. Under-approximations are needed for other types of analysis, which we shall discuss later. It is important to emphasize that every orthogonal polyhedron that is used in our approximation procedure should be defined on the *same underlying grid*.

Definition 18 (Orthogonal approximation)
We define the operators $grid_o$ and $grid_u$ for a convex polyhedron C and a grid $\mathcal{G}_\beta$ as follows.

- *$grid_o(C)$ is the smallest orthogonal polyhedron defined on $\mathcal{G}_\beta$ such that $grid_o(C) \supseteq C$.*
- *$grid_u(C)$ is the largest orthogonal polyhedron defined on $\mathcal{G}_\beta$ such that $grid_u(C) \subseteq C$.*

Followed immediately from the definition, $grid_o(C)$ and $grid_u(C)$ can be written as

$$\begin{aligned} grid_o(C) &= \{g(\mathbf{v}) \mid \mathbf{v} \in \mathcal{G}_\beta \ \wedge \ g(\mathbf{v}) \cap C \neq \emptyset\}, \\ grid_u(C) &= \{g(\mathbf{v}) \mid \mathbf{v} \in \mathcal{G}_\beta \ \wedge \ g(\mathbf{v}) \subseteq C\} \end{aligned}$$

where $g(\mathbf{v})$ is the elementary hyper-cube associated with a grid point $\mathbf{v}$ in $\mathcal{G}_\beta$. In other words, $grid_u(C)$ is the union of all elementary hyper-cubes which are *inside* C, and $grid_o(C)$ is the union of all elementary hyper-cubes *whose intersection with C is not empty* (see Figure 4.3).

Note that for a given underlying grid, $grid_o(C)$ and $grid_u(C)$ are *tight orthogonal approximations* of the convex polyhedron C in the sense that there exists no orthogonal polyhedron smaller than $grid_o(C)$ that includes C and there exists no orthogonal polyhedron larger than $grid_u(C)$ that is inscribed in C.

An obvious method for computing $grid_o(C)$ and $grid_u(C)$ is to test all the elementary hyper-cubes $g(\mathbf{v})$ residing in the bounding box of C. This is, evidently, inefficient since the number of tests, which depends on the ratio of the volume of C to the volume of the elementary hypercubes of the grid, can be large. In Chapter 8 (Implementation), we will present a more efficient method for computing $grid_o$ and $grid_u$, inspired by the Binary Space Partition concept [44].

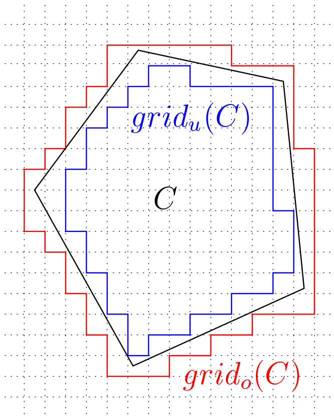


Figure 4.3: Orthogonal approximations: the convex polyhedron C is over- and under-approximated by $grid_o(C)$ and $grid_u(C)$.

It is not hard to see that the error in both over-approximation and under-approximation in terms of the Hausdorff distance is bounded by $\sqrt{n}\beta$, and one can obtain more accurate orthogonal approximations by using finer grids.

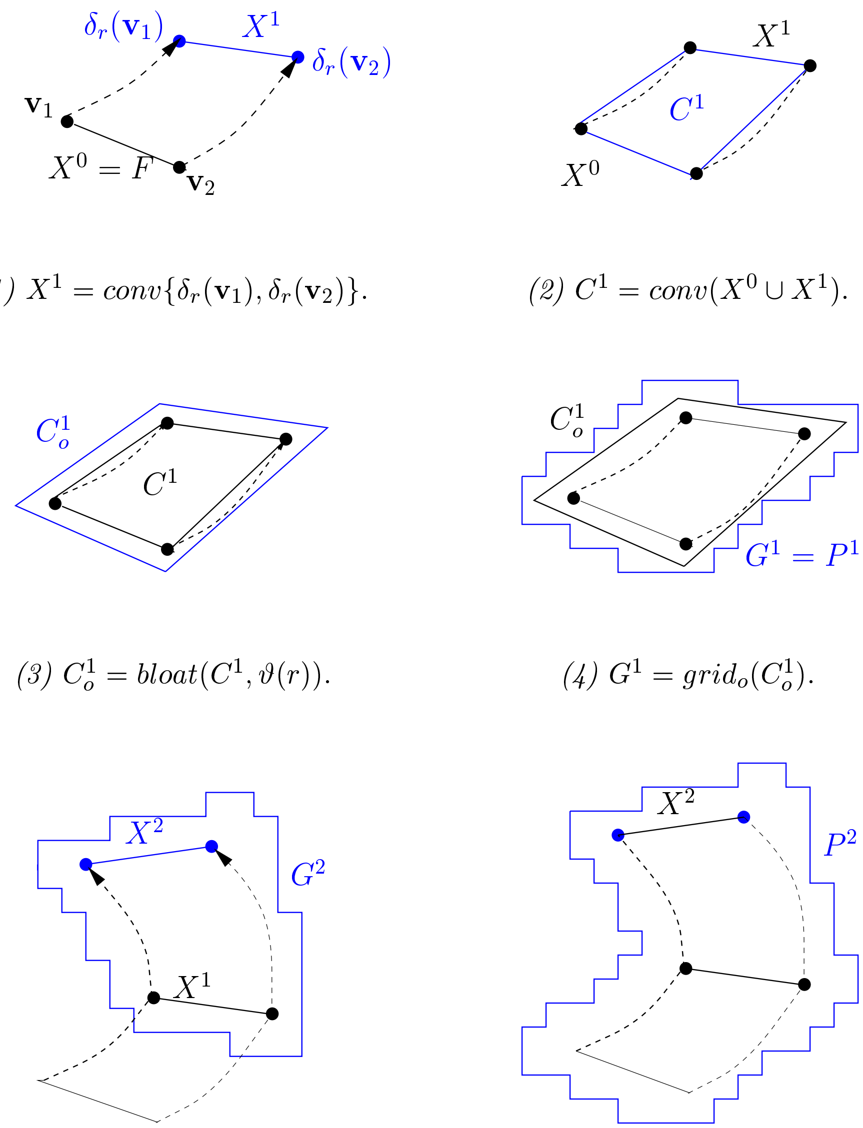
4.1.2 Reachability Algorithm

We have now all the ingredients needed to compute an orthogonal over-approximation of $\delta_{[0,r]}(F)$. Embedding this procedure in Algorithm 3, we obtain the following algorithm for over-approximating the reachable set $\delta(F)$.

Algorithm 4 (Over-approximating $\delta(F)$)

$P^0 := \emptyset$; $X^0 := F$;
repeat $k = 0, 1, 2, \ldots$
  $X^{k+1} := \delta_r(X^k)$;
  $C^{k+1} := conv(X^{k+1} \cup X^k)$;
  $C_o^{k+1} := bloat(C^{k+1}, \vartheta(r))$;
  $G^{k+1} := grid_o(C_o^{k+1})$;
  $P^{k+1} := P^k \cup G^{k+1}$;
until $P^{k+1} = P^k$
return P^{k+1}

Figure 4.4 illustrates the steps of Algorithm 4 with a simple two-dimensional example where the initial set F is a line segment with two extreme points $\mathbf{v}_1$ and $\mathbf{v}_2$. The set X^0 is initialized to the initial set F, and the algorithm basically repeats the following four steps:



(1) $X^1 = conv\{\delta_r(\mathbf{v}_1), \delta_r(\mathbf{v}_2)\}$. *(2)* $C^1 = conv(X^0 \cup X^1)$.

(3) $C_o^1 = bloat(C^1, \vartheta(r))$. *(4)* $G^1 = grid_o(C_o^1)$.

The second iteration computes G^2 *based on* X^1. $P^2 = G^1 \cup G^2$.

Figure 4.4: Two iterations of Algorithm 4 on a simple example where the initial set F is a two-dimensional line segment with two extreme points $\mathbf{v}_1$ and $\mathbf{v}_2$. The computation in the second iteration starts from X^1.

1. Compute $X^{k+1} = \delta_r(X^k)$, which represents the reachable set after exactly $(k+1)r$ time. In Figure 4.4, the dotted lines are the real trajectories starting from the vertices of X^0, and the exact set $\delta_{[0,r]}(X^0)$ lies between these lines.

2. Compute C^{k+1} as the convex hull of $(X^k \cup X^{k+1})$ or equivalently the convex-hull of the vertices of both X^k and X^{k+1}. The convex-hull algorithm provides us with information about the orientation of the faces of C^{k+1}, which is used in the next step.

3. Bloat C^{k+1} by the amount $\vartheta(r)$ determined according to (4.4) to obtain the convex polyhedron C_o^{k+1}, which is guaranteed to contain $\delta_{[0,r]}(X^k)$.

4. Compute the orthogonal over-approximation G^{k+1} of C_o^{k+1}. The polyhedron G^{k+1} representing $\widehat{\delta}_{[kr,(k+1)r]}(F)$ is then added to P^{k+1} that is used to store all the states reachable over the execution.

In the example shown in Figure 4.4, after the first iteration we get the orthogonal polyhedron $G^1 \supseteq \delta_{[0,r]}(F)$. In the next iteration, we repeat the above four steps *starting from* X^1 to obtain $G^2 \supseteq \delta_{[r,2r]}(F)$. The orthogonal polyhedron $P^2 = G^1 \cup G^2$ is thus an over-approximation of the set of states reachable after two iterations.

Careful readers may realize that approximate calculations are used in Algorithm 4, and hence the condition $P^{k+1} = P^k$ is not sufficient to ensure that whenever the algorithm terminates, it gives an over-approximation of the whole reachable set. We defer this problem to Section 4.4 and continue with a brief discussion on the main factors which influence the time cost of Algorithm 4.

Computational Cost

The computation time of the first three steps depends on the number of vertices of X^k because it determines the number of numerical integrations to perform as well as the time complexity of the convex-hull algorithm. Note that the *bloat* operation requires intersecting half-spaces, which can be transformed into a convex-hull problem. It is worth mentioning that although the number of vertices of X^k is unchanged over the execution, the time needed for numerical integration[1], being sensitive to the stiffness of differential equations, may vary. We have already seen that the computation time needed for orthogonal approximations depends mostly on the ratio of the volume of X^k to the granularity of the underlying grid. Moreover, the use of finer grids results in more vertices in P^k and consequently more computation time for the union operation and equivalence testing in the next two steps. We will show, in the next section, that the size of the underlying grid must be chosen according to the desired accuracy. Therefore, a crucial problem is to find the right compromise between accuracy and computational cost.

[1] The time step r of our approximation scheme should not be confused with the step-size of numerical integration procedures.

4.2 Error Analysis

In order to guarantee the desired accuracy, we need to determine the bound on error in the approximate solution and find conditions for ensuring an error under the specified tolerance.

4.2.1 Error Propagation

The main advantage of Algorithm 4 is that it avoids the effect of over-approximation error accumulation due to the use of orthogonal polyhedra. To clarify this, let us indicate two types of errors that are introduced into the computation in each iteration.

- *Numerical integration error* which results from the computation of X^{k+1} from X^k.
- *Over-approximation error* which is inherent in the approximation of $\delta_{[0,(k+1)r]}(F)$ by P^{k+1} based on X^{k+1} and X^k.

Indeed, the polyhedron P^k contains the over-approximation error while X^k does not. Since P^{k+1} is computed based on X^k and not on P^k, *the over-approximation error does not propagate* from iteration to iteration.

We assume that the numerical integration error is negligible[2]; therefore, every polyhedra X^k is the exact set $\delta_{kr}(F)$. Under this assumption, we can state an important property of Algorithm 4: *the global error does not accumulate over the execution.* In other words, the error in the result is the upper bound on the local error incurred in each iteration.

4.2.2 Error Estimation

We now estimate the error in our approximation under the assumption that there is no numerical integration error. We will prove that the error in terms of the Hausdorff distance can be made arbitrarily small by choosing the adequate value of the time step and the grid size.

We are interested in finding *the worst-case error*, that is, the largest error for any input system, so that we can guarantee that the error in the results produced by our algorithm will never be beyond this.

Since the over-approximation error does not propagate from one iteration to another, it suffices to study the local error incurred in each iteration. To get a bound on it, we estimate the distance between the exact set $\delta_{[0,r]}(F)$ and the approximate set $\widehat{\delta}_{[0,r]}(F)$, which is obtained by making the convex hull C, bloating C to get C_o, and transforming C_o into orthogonal. By the triangle inequality, we have

$$h(\delta_{[0,r]}(F), \widehat{\delta}_{[0,r]}(F)) \leq h(\delta_{[0,r]}(F), C) + h(C, C_o) + h(C_o, \widehat{\delta}_{[0,r]}(F)).$$

[2] The numerical integration error is sometimes inevitable due to the effects of round-off errors in the arithmetic of the computer. To handle this problem, special arithmetics, such as interval arithmetic [1], can be used.

We have shown earlier that the distance between C_o and $\widehat{\delta}_{[0,r]}(F) = grid_o(C_o)$ is bounded by the grid size $\sqrt{n}\beta$. Then,

$$h(\delta_{[0,r]}(F), \widehat{\delta}_{[0,r]}(F)) \leq h(\delta_{[0,r]}(F), C) + h(C, C_o) + \sqrt{n}\beta. \tag{4.5}$$

We estimate first $h(\delta_{[0,r]}(F),\ C)$. The distance between an arbitrary point $\mathbf{x} \in F$ and its successors $\delta_t(\mathbf{x}) = e^{At}\mathbf{x}$ for every $t \in [0, r]$ satisfies the following inequality:

$$||\delta_t(\mathbf{x}) - \mathbf{x}|| \leq M||e^{Ar} - I||$$

where M is the constant bounding $||\mathbf{x}||$.

It then follows by Lemma 1-(h2) that

$$h(\delta_{[0,r]}(F), F) \leq M||e^{Ar} - I||.$$

Since $C = conv(F \cup \delta_r(F))$, we also have $h(C, F) \leq M||e^{Ar} - I||$. This leads to

$$h(\delta_{[0,r]}(F), C) \leq h(\delta_{[0,r]}(F), F) + h(F, C) \leq 2M||e^{Ar} - I||.$$

Expanding e^{Ar} in a Taylor series, we obtain

$$h(\delta_{[0,r]}(F), C) \leq 2M||A||r + \mathcal{O}(r^2) = \eta. \tag{4.6}$$

This means that the distance between the real set $\delta_{[0,r]}(F)$ and the convex hull C is bounded by η, which is of the order $\mathcal{O}(r)$.

As stated in the previous section, $h(C,\ C_o) \leq \sqrt{n}\vartheta(r)$ where

$$\vartheta(r) = \frac{1}{8}M||A||^2r^2 + \mathcal{O}(r^3)$$

is the bloating amount applied to the convex hull C and n is the dimension of the system. Therefore,

$$h(\delta_{[0,r]}(F), \widehat{\delta}_{[0,r]}(F)) \leq 2M||A||r + \sqrt{n}\beta + \mathcal{O}(r^2). \tag{4.7}$$

Note that $h(C,\ C_o)$ is now included in the $\mathcal{O}(r^2)$ term in the right-hand side of the above inequality. Formula (4.7) yields the following theorem.

Theorem 2 *The error in the approximation is bounded by*

$$\epsilon = 2M||A||r + \sqrt{n}\beta + \mathcal{O}(r^2)$$

where r is the time step and β is the size of the underlying grid.

The theorem also shows that ϵ is of the order $\mathcal{O}(r)$, and this provides us with the information about how fast the approximate solution approaches the true solution by reducing the time step. In addition, by refining the underlying grid, β can be made as small as desired. The following result is an immediate consequence of Theorem 2.

Result 1 *The error in the approximation can be made arbitrarily small by changing the time step r and the grid size β.*

Remark 1 *The upper bound ϵ on the error from Theorem 2 can be much larger than the real error in practice.*

As we have already seen, the error in our approximation is mostly due to the approximation by the convex hull C. This comes as no surprise since the reachable set $\delta_{[0,r]}(F)$ need not be convex. The error incurred in the convex-hull step is of the order $\mathcal{O}(r)$ while the other errors are of the order $\mathcal{O}(r^2)$. However, the bound η on the distance $h(\delta_{[0,r]}(F), C)$, given in (4.6), is approached only when the set $\delta_{[0,r]}(F)$ is very concave. In many other cases, this distance can be much smaller.

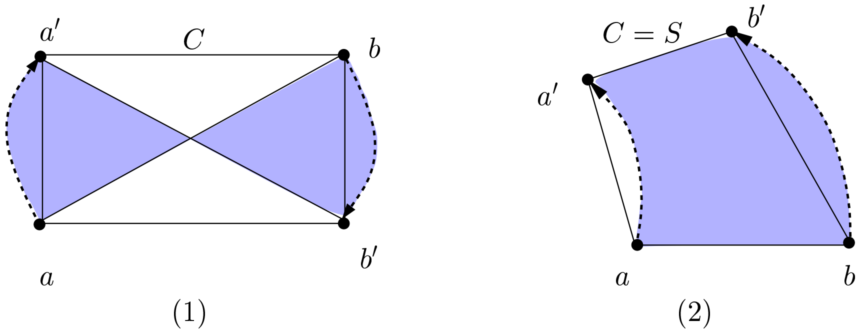


Figure 4.5: In (1) the distance $h(\delta_{[0,r]}(F), C)$ is of the order $\mathcal{O}(r)$ while in (2) it is of the order $\mathcal{O}(r^2)$.

To illustrate this point, consider two examples shown in Figure 4.5. The initial set F is the two-dimensional line segment ab, and the set of its successors at time r is the line segment $a'b'$. The dotted curves are the true trajectories from a and b, and the exact reachable sets $\delta_{[0,r]}(F)$ are shown as the shaded regions in both examples. One can see that the distance $h(\delta_{[0,r]}(F), C)$ in (1) is $\mathcal{O}(r)$. However, in (2) the set S and the convex hull C coincide. In addition, we have shown in the proof of Theorem 1 that $h(\delta_{[0,r]}(F), S) = \mathcal{O}(r^2)$; therefore $h(\delta_{[0,r]}(F), C)$ in (2) is $\mathcal{O}(r^2)$.

4.2.3 Accuracy Improvement

A conclusion from the error analysis is that the intermediate convex-hull approximation causes the most significant error in the results. A solution to remedy this consists in, first, partitioning the polyhedron F into sub-polyhedra, whose successors at time r can be easily computed from $\delta_r(F)$ by simple linear transformations, and, second, applying the approximation scheme to each sub-polyhedron separately. If the sub-polyhedra are small enough, then the convex-hull approximation incurs considerably less error. To illustrate how partioning can improve the approximation accuracy, consider again the example with the worst-case

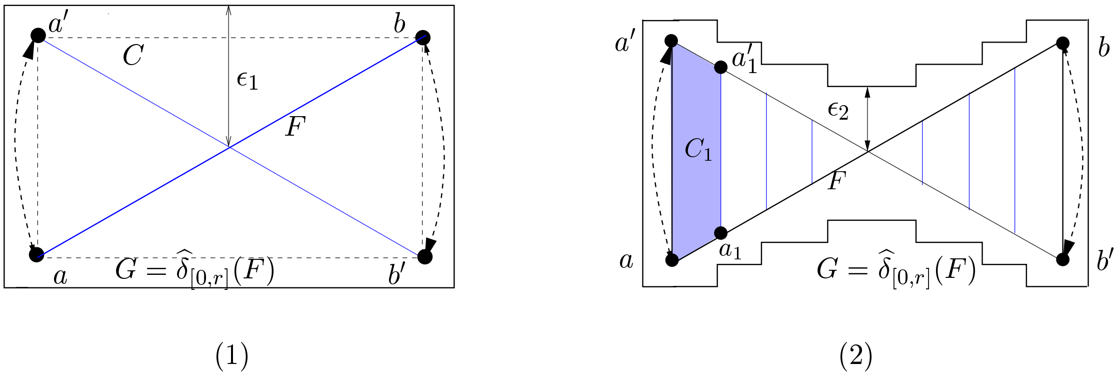


Figure 4.6: Accuracy improvement by partitioning of the polyhedron.

error shown in Figure 4.5. Figure 4.6-(1) and -(2) depict the results obtained without and with prior partitioning the initial polyhedron. In (2), the segment ab is split into eight sub-segments. The successors of the sub-segment aa_1 is $a'a'_1$, and the shaded polygon C_1 is their convex hull. One can see that by treating each sub-segment separately the error due to the convex-hull approximation is reduced, and the accuracy of the result is remarkably improved. As shown in the figures, the global error ϵ_2 in the result obtained with partitioning in (2) is much smaller than the error ϵ_1 in (1). Nevertheless, it is clear that this solution results in a loss of efficiency in terms of computation time.

4.3 Under-approximation

While the safety verification problem requires over-approximation of the δ operator, other tasks such as controller synthesis (characterizing all states from which the system satisfies a given property) require under-approximation of this operator. In this section, we show how to compute an orthogonal under-approximation of $\delta_{[0,r]}(F)$, denoted by $\widetilde{\delta}_{[0,r]}(F)$.

The idea is the following. Let G be the orthogonal over-approximation of $\delta_{[0,r]}(F)$: $G = \widehat{\delta}_{[0,r]}(F)$. We know that the distance between $\delta_{[0,r]}(F)$ and G is bounded by ϵ from Theorem 2, and hence we can compute $\widetilde{\delta}_{[0,r]}(F)$ by 'narrowing' the polyhedron G by the amount ϵ. For doing this, we define the rectangular neighborhood of G as follows.

Let E be the set of faces of G. Consider a face $e \in E$ whose normal is parallel to the axis $i \in \{1, \ldots, n\}$. The face e is indeed an $(n-1)$-dimensional hyper-rectangle and can be written as

$$e = [l_1, u_1] \times \ldots \times [l_i, l_i] \times \ldots \times [l_n, u_n].$$

The *rectangular ϵ-neighborhood* of e, denoted by $N_s(e, \epsilon)$, is simply a full-dimensional hyper-rectangle written as

$$N_s(e, \epsilon) = [l_1 - \epsilon, u_1 + \epsilon] \times \ldots \times [l_i - \epsilon, l_i + \epsilon] \times \ldots \times [l_n - \epsilon, u_n + \epsilon],$$

and the rectangular ϵ-neighborhood of the boundary of G is

$$N_s(\partial G, \epsilon) = \bigcup_{e \in E} N_s(e, \epsilon).$$

We compute the under-approximation $\widetilde{\delta}_{[0,r]}(F)$ as follows:

$$\widetilde{\delta}_{[0,r]}(F) = G \setminus N_s(\partial G, \epsilon). \tag{4.8}$$

Then, to obtain an under-approximation of the whole reachable set, we need just to insert this computation in Algorithm 4.

An important remark is that the under-approximation $\widetilde{\delta}_{[0,r]}(F)$ described above may be empty even if the interior of the initial polyhedron F is not empty. This happens when ϵ, which is $\mathcal{O}(r)$, is large and the polyhedron F is narrow. In such cases, in order to obtain a non-empty under-approximation, one needs to reduce the time step r.

4.4 Termination Condition

In order for Algorithm 4 to be correct, the termination condition must ensure that when the algorithm terminates, it gives an over-approximation of the whole reachable set.

By construction, the set computed in each iteration is guaranteed to be an over-approximation of the required set. However, the equivalence between the approximate sets obtained in two consecutive iterations is not sufficient for the termination decision. In fact, P^k can be written as $P^k = \delta_{[0,kr]}(F) \cup \mathcal{E}^k$ where $\mathcal{E}^k$ represents the over-approximation error. If the polyhedron X^{k+1} is, unfortunately, included in $\mathcal{E}^k$, then using $P^{k+1} = P^k$ as termination condition one may decide to stop whereas X^{k+1} can still generate new reachable states. The example shown in Figure 4.7 illustrates this phenomenon. In this example, the system starts from a line segment X^0. As integration is performed, the line segment spirals towards the origin. After some iterations, X^{k+1} is included in the previously computed set P^k because of the over-approximation error, and we have then $P^{k+1} = P^k$. If the algorithm stops at this point, some reachable states will be missed.

As stated earlier, the polyhedron X^{k+1} is the exact set $\delta_{(k+1)r}(F)$. Hence, for the termination decision, it is sufficient to check the condition

$$X^{k+1} \subseteq \delta_{[0,kr]}(F) \tag{4.9}$$

Indeed, when this condition is satisfied, the polyhedron X^{k+1} contains uniquely the states that the system has already visited during the time interval $[0, kr]$; as a result, X^{k+1} will not contribute any new reachable states. Nevertheless, we do not know the exact set $\delta_{[0,kr]}(F)$. To be sound, we use a stronger condition, that is, we check whether X^{k+1} is included in $\widetilde{\delta}_{[0,kr]}(F)$, which is a subset of $\delta_{[0,kr]}(F)$. Note that the checking of this condition, in some cases, cannot detect that the set X^{k+1} is included in X^k, and we will therefore check both. To be precise, the algorithm using this termination condition is as follows.

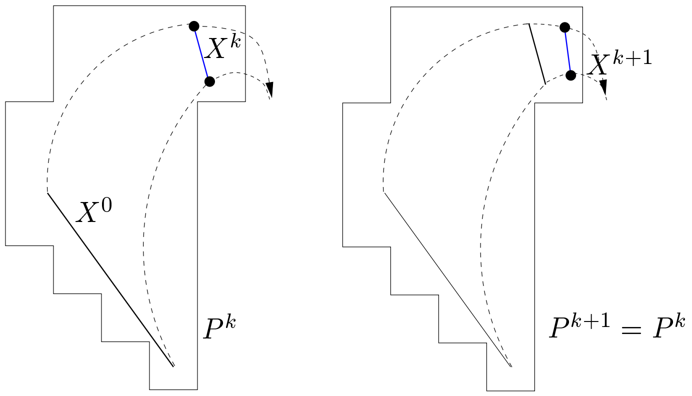


Figure 4.7: The condition $P^{k+1} = P^k$ is satisfied, but not all the reachable states are visited (the exact reachable set lies between two dotted curves).

Algorithm 5 (Over-approximating $\delta(F)$)

$P^0 := \emptyset$; $X^0 := F$; $P_u^0 := \emptyset$;
repeat $k = 0, 1, 2, \ldots$
  $X^{k+1} := \delta_r(X^k)$;
  $C^{k+1} := conv(X^{k+1} \cup X^k)$;
  $C_o^{k+1} := bloat(C^{k+1}, \vartheta(r))$;
  $G^{k+1} := grid_o(C_o^{k+1})$;
  $G_u^{k+1} := G^{k+1} \setminus N_s(\partial G^{k+1}, \epsilon)$;
  $P_u^{k+1} := P_u^k \cup G_u^{k+1}$;
  $P^{k+1} := P^k \cup G^{k+1}$;
until $X^{k+1} \subseteq P_u^k \;\vee\; X^{k+1} \subseteq X^k$
return P^{k+1}

In each iteration, the orthogonal polyhedron G_u^{k+1} is an under-approximation of $\delta_{[kr,(k+1)r]}(F)$ and P_u^{k+1} is an under-approximation of $\delta_{[0,(k+1)r]}(F)$.

We can now state an important property of Algorithm 5, which is a direct consequence of the above analysis.

Theorem 3 (Soundness)
If Algorithm 5 terminates then it produces an over-approximation of the reachable set.

As mentioned above, the new termination condition being stronger than (4.9), the price for soundness is that the algorithm may not terminate in some cases.

Before continuing with a discussion on linear systems with uncertain input, we remark that the results presented so far can be straightforwardly extended to systems with constant input of the form $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}$ where $\mathbf{u}$ is a constant in $\mathbb{R}^n$.

4.5 Extension to Linear Systems with Uncertain Input

It is of great interest to study systems which are subject to external disturbances about which we know only some constraints. An example of such systems is a thermostat whose behavior is influenced by fluctuations in the outside temperature about which we know only the minimum and maximum values.

We have proposed a reachability algorithm for linear systems without input. In the following we discuss an extension of this algorithm to linear systems with uncertain input. We present first some basic notions related to non-deterministic behavior of such systems.

4.5.1 Additional Notations

Consider a continuous system $\mathcal{C} = \{\mathcal{X}, U, f\}$ where $\mathcal{X} \subseteq \mathbb{R}^n$ is the state space of the system and $U \subset \mathbb{R}^m$ is the input set. The behavior of the system is described by the following differential equation

$$\dot{\mathbf{x}} = f(\mathbf{x}, \mathbf{u}) \tag{4.10}$$

where $\mathbf{x} \in \mathcal{X}$ is the state of the system and $\mathbf{u} \in U$ is the input. We assume a set of admissible inputs $\mathcal{U}$ consisting of measurable functions of the form $\mu : \mathcal{T} \to U$.

The behavior of such systems can also be analyzed using differential inclusions [17].

Definition 19 (Trajectory of Continuous Dynamical Systems with Input) *A trajectory of $\mathcal{C}$ starting from a point $\mathbf{x} \in \mathcal{X}$ under a given input μ: $\mathcal{T} \to U$ is a continuous behavior $\xi_{\mathbf{x},\mu} : \mathcal{T} \to \mathcal{X}$ such that $\xi_{\mathbf{x},\mu}(t)$ is the solution of $\dot{\mathbf{x}}(t) = f(\mathbf{x}(t), \mu(t))$ with the initial condition $\mathbf{x}(0) = \mathbf{x}$.*

The trajectory $\xi_{\mathbf{x},\mu}$ is also called the 'response' of the system to μ when starting at $\mathbf{x}$.

We assume that the function f is globally Lipschitz in $\mathbf{x}$ and continuous in $\mathbf{u}$. This assumption guarantees existence and uniqueness of the solution of the differential equation (4.10) for a given $\mu \in \mathcal{U}$ [53, 61]; therefore the trajectory $\xi_{\mathbf{x},\mu}$ is unique.

When the input cannot be observed, the behavior of the system is non-deterministic. For a given initial condition $\mathbf{x}$, every fixed input μ generates a different solution to (4.10). As a consequence, under all admissible inputs the system produces a dense 'bundle' of trajectories.

Let F be a subset of $\mathcal{X}$. The set of states reachable from F at time point t under a given input $\mu \in \mathcal{U}$, denoted by $\delta_{t,\mu}(F)$, is simply the set of states visited at time t by all the trajectories starting from points in F under μ:

$$\delta_{t,\mu}(F) = \bigcup_{\mathbf{x}\in F} \xi_{\mathbf{x},\mu}(t).$$

The set of all states reachable from F at time point t, denoted by $\delta_t(F)$, is

$$\delta_t(F) = \bigcup_{\mu\in\mathcal{U}} \delta_{t,\mu}(F).$$

Then, the set of all states reachable from F during the time interval $[0, r]$ is

$$\delta_{[0,r]}(F) = \bigcup_{t\in[0,r]} \delta_t(F),$$

and the set of all states reachable from F after any non-negative amount of time is thus $\delta_{[0,\infty]}(F)$, which we denote by $\delta(F)$ for brevity. The above notions are illustrated in Figure 4.8. One can see from the figure that under different inputs the system, when started at point $\mathbf{x}$, generates different trajectories.

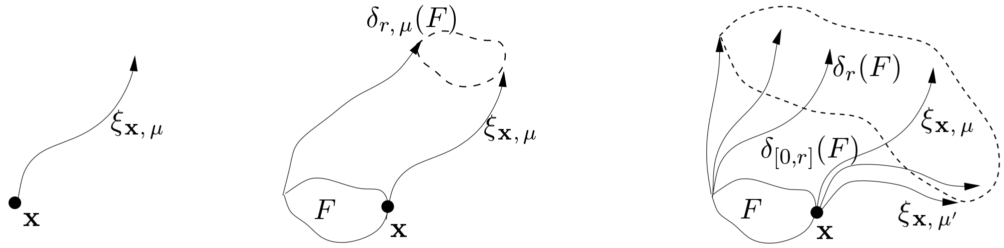


Figure 4.8: Reachable set of a continuous dynamical system with input.

4.5.2 Reachability Algorithm

We consider a linear system $\mathcal{C} = \{\mathcal{X}, U, f\}$ where $\mathcal{X} \subseteq \mathbb{R}^n$ and $U \subset \mathbb{R}^n$. The dynamics of the system is defined by the linear differential equation

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u} \tag{4.11}$$

where $\mathbf{x} \in \mathcal{X}$ and $\mathbf{u} \in U$. We assume that the input set U is *convex* and *compact*.

Note that any system $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{v}$ where $\mathbf{v} \in \mathcal{V} \subset \mathbb{R}^m$ can be transformed into a system of the form (4.11) by letting $\mathbf{u} = B\mathbf{v}$ and defining the input set $U = \{\mathbf{u} \mid \mathbf{u} = B\mathbf{v} \ \wedge \ \mathbf{v} \in \mathcal{V}\}$.

Let us recall our reachability problem. Given a convex polyhedron F, we want to compute an over-approximation of the reachable set from F, that is, $\delta(F)$.

A natural inclination is to use the underlying idea of the method for linear systems without input. Concretely, to over-approximate $\delta_{[0,r]}(F)$ we compute the convex hull $C = conv(F \cup \delta_r(F))$, bloat C by a certain amount, and over-approximate it by an orthogonal polyhedron. However, in order to extend this method to systems with uncertain input, one has to consider the following two facts:

1. The technique for computing $\delta_t(F)$, presented in Section 4.1, is no longer appropriate because it is impossible to simulate trajectories from the vertices of F with all possible inputs. To solve this problem, we will make use of the technique suggested by P. Varaiya [114], which is based on the Maximum Principle of optimal control [64, 82].

2. The estimation of the bloating amount applied to the convex hull C that guarantees over-approximations must take into account uncertainty in the input.

We begin by presenting the technique of [114] for approximating $\delta_t(F)$, i.e. the set of states reachable from F at time point t.

Approximating $\delta_t(F)$

Consider a face e of F whose supporting hyper-plane is

$$P = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle = \langle \mathbf{a}, \mathbf{y} \rangle\}$$

where $\mathbf{a}$ is the *outward normal* to e and $\mathbf{y}$ is an arbitrary point on the face e, which we call *supporting point* of P. Hence, the polyhedron F lies inside the half-space $H = \{\mathbf{x} \mid \langle \mathbf{a}, \mathbf{x} \rangle \leq \langle \mathbf{a}, \mathbf{y} \rangle\}$.

The key idea is the following. By the Maximum Principle, for every face e of F there exists an input $\mu^* \in \mathcal{U}$ such that calculating the successors of its supporting plane P under μ^* is sufficient to derive a tight polyhedral approximation of $\delta_t(F)$.

It can be proved that the evolution of the normal to P is governed by the adjoint system of $\mathcal{C}$, denoted by $\mathcal{C}^T$, whose dynamics is described by the following differential equation [82]:

$$\dot{\mathbf{x}} = -A^T \mathbf{x}. \tag{4.12}$$

Let $\lambda_{\mathbf{a}} : \mathcal{T} \to \mathcal{X}$ be the trajectory of $\mathcal{C}^T$ starting from $\mathbf{a}$, in other words, $\lambda_{\mathbf{a}}$ is the solution to the differential equation (4.12) with the initial condition $\mathbf{x}(0) = \mathbf{a}$:

$$\lambda_{\mathbf{a}}(t) = e^{-A^T t}\mathbf{a}. \tag{4.13}$$

One can see that the trajectory $\lambda_{\mathbf{a}}$ of the normal to e *does not depend on the input.*

Recall that $\xi_{\mathbf{x},\mu}$ denotes the trajectory of $\mathcal{C}$ starting from point $\mathbf{x}$ and under the input function μ. By solution of (4.11), we have

$$\xi_{\mathbf{x},\mu}(t) = e^{At}\mathbf{x} + \int_0^t e^{A(t-s)}\mu(s)ds.$$

Thus,

$$\langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{x},\mu}(t)\rangle = \langle \lambda_{\mathbf{a}}(t), e^{At}\mathbf{x}\rangle + \langle \lambda_{\mathbf{a}}(t), \int_0^t e^{A(t-s)}\mu(s)ds\rangle. \tag{4.14}$$

Since $\lambda_{\mathbf{a}}(t) = e^{-A^T t}\mathbf{a}$ and, in addition, every initial point $\mathbf{x} \in F$ satisfies $\langle \mathbf{a}, \mathbf{x}\rangle \leq \langle \mathbf{a}, \mathbf{y}\rangle$, after obvious simplifications we obtain from (4.14) the following inequality which holds for all inputs $\mu \in \mathcal{U}$, for all $\mathbf{x} \in F$ and for all $t \geq 0$:

$$\langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{x},\mu}(t)\rangle \leq \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu}(t)\rangle. \tag{4.15}$$

Intuitively, this means that the set $\delta_{t,\mu}(F)$ of states reachable from F at time t under the fixed input μ is inside the half-space $H_\mu(t) = \{\mathbf{x} \mid \langle \lambda_{\mathbf{a}}(t), \mathbf{x}\rangle \leq \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu}(t)\rangle\}$.

Similarly, since every point $\mathbf{x}$ in the supporting hyper-plane P of the face e satisfies $\langle \mathbf{a}, \mathbf{x}\rangle = \langle \mathbf{a}, \mathbf{y}\rangle$, using (4.14) again, it is also easy to prove that the set $\delta_{t,\mu}(P)$ is the hyper-plane $P_\mu(t)$ with the normal $\lambda_{\mathbf{a}}(t)$ and the supporting point $\xi_{\mathbf{y},\mu}(t)$:

$$\delta_{t,\mu}(P) = P_\mu(t) = \{\mathbf{x} \mid \langle \lambda_{\mathbf{a}}(t), \mathbf{x}\rangle = \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu}(t)\rangle\}.$$

We remark that the normal $\lambda_{\mathbf{a}}(t)$ to the hyper-plane $P_\mu(t)$ is independent of the input; therefore for all $\mu \in \mathcal{U}$ the hyper-planes $P_\mu(t)$ are parallel to each other as shown in Figure 4.9.

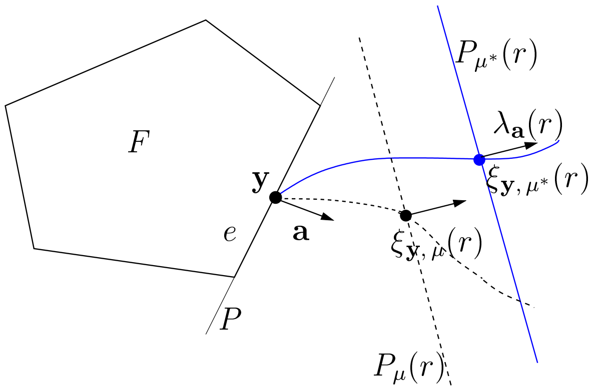


Figure 4.9: The solid and the dotted curves are the trajectories $\xi_{\mathbf{y},\mu^*}$ under μ^* and $\xi_{\mathbf{y},\mu}$ under μ. At time point r, the hyper-plane $P_{\mu^*}(r) = \delta_{r,\mu^*}(P)$ is determined by the normal $\lambda_{\mathbf{a}}(r)$ and the supporting point $\xi_{\mathbf{y},\mu^*}(r)$.

Let $\mu^* \in \mathcal{U}$ be an input function such that the following holds for all $t \geq 0$:

$$\langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu^*}(t)\rangle = \max\{\langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu}(t)\rangle \mid \mu \in \mathcal{U}\}. \tag{4.16}$$

The above and (4.15) imply that for all inputs $\mu \in \mathcal{U}$, for all initial points $\mathbf{x} \in F$ and for all $t \geq 0$

$$\langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{x},\mu}(t) \rangle \leq \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu^*}(t) \rangle. \tag{4.17}$$

Formula (4.17) simply says that *all the states reachable from F at time t are inside the half-space $H_{\mu^*}(t) = \{\mathbf{x} \mid \langle \lambda_{\mathbf{a}}(t), \mathbf{x} \rangle \leq \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu^*}(t) \rangle\}$* and *the hyper-plane $P_{\mu^*}(t) = \{\mathbf{x} \mid \langle \lambda_{\mathbf{a}}(t), \mathbf{x} \rangle = \langle \lambda_{\mathbf{a}}(t), \xi_{\mathbf{y},\mu^*}(t) \rangle\}$ supports the reachable set $\delta_t(F)$ at $\xi_{\mathbf{y},\mu^*}(t)$.*

We will exploit this important result to derive an over-approximation of $\delta_t(F)$. For doing this, we need to find the input function μ^* which steers the initial hyper-plane P to $P_{\mu^*}(t)$ at every time point t.

By solution of the differential equation (4.11), (4.16) is equivalent to the following (the terms involving $\mathbf{y}$ cancel):

$$\langle \lambda_{\mathbf{a}}(t), \int_0^t e^{A(t-s)} \mu^*(s) ds \rangle = \max\{\langle \lambda_{\mathbf{a}}(t), \int_0^t e^{A(t-s)} \mu(s) ds \rangle \mid \mu \in \mathcal{U}\},$$

or equivalently,

$$\int_0^t \langle \lambda_{\mathbf{a}}(t), e^{At} e^{-As} \mu^*(s) \rangle ds = \max\{\int_0^t \langle \lambda_{\mathbf{a}}(t), e^{At} e^{-As} \mu(s) \rangle ds \mid \mu \in \mathcal{U}\}. \tag{4.18}$$

Since $\lambda_{\mathbf{a}}(t) = e^{-A^T t}\mathbf{a}$, we have

$$\begin{aligned} \langle \lambda_{\mathbf{a}}(t), e^{At} e^{-As} \mu(s) \rangle &= \langle e^{-A^T t}\mathbf{a}, e^{At} e^{-As} \mu(s) \rangle \\ &= \langle \mathbf{a}, e^{-As} \mu(s) \rangle. \end{aligned}$$

Therefore, (4.18) becomes

$$\int_0^t \langle \mathbf{a}, e^{-As} \mu^*(s) \rangle ds = \max\{\int_0^t \langle \mathbf{a}, e^{-As} \mu(s) \rangle \; ds \mid \mu \in \mathcal{U}\}.$$

Note again that $\langle \mathbf{a}, e^{-As}\mu(s) \rangle = \langle e^{-A^T s}\mathbf{a}, \mu(s) \rangle$. Hence, the input function μ^* satisfies the following for every time point t

$$\mu^*(t) \in \arg\max\{\langle \lambda_{\mathbf{a}}(t), \mathbf{u} \rangle \mid \mathbf{u} \in U\}.$$

Now, we apply the above analysis to the initial polyhedron F, which can be represented as the intersection of, say, m_h half-spaces H_i as follows:

$$F = \bigcap_{i=1}^{m_h} \{\mathbf{x} \mid \langle \mathbf{a}_i, \mathbf{x} \rangle \leq \langle \mathbf{a}_i, \mathbf{y}_i \rangle\}$$

Let $\mu_i^*(t) \in \arg\max\{\langle \lambda_{\mathbf{a}_i}(t), \mathbf{u} \rangle \mid \mathbf{u} \in U\}$ for every $t \geq 0$, $i = 1, \ldots, m_h$. The following proposition is a direct consequence of the above results.

Proposition 3

$$\delta_t(F) \subseteq \bigcap_{i=0}^{m} \{\mathbf{x} \mid \langle \lambda_{\mathbf{a}_i}(t), \mathbf{x}\rangle \leq \langle \lambda_{\mathbf{a}_i}(t), \xi_{\mathbf{y}_i, \mu_i^*}(t)\rangle\}.$$

The reader might wish to consult [114] for another proof of Proposition 3.

Proposition 3 provides the following scheme for computing an over-approximation of the reachable set from F at time point t. We denote this by $\widehat{\delta}_t(F)$.

For brevity we denote $\mathbf{y}_i^*(t) = \xi_{\mathbf{y}_i, \mu_i^*}(t)$ and $\mathbf{a}(t) = \lambda_{\mathbf{a}_i}(t)$. Let $\mathbf{a}_i(t)$, $\mathbf{y}_i^*(t)$ be solutions to the differential equations (4.19) and (4.20), $i = 0, \ldots, m_h$.

$$\begin{aligned} \dot{\mathbf{a}}_i(t) &= -A^T\mathbf{a}_i(t);\ \mathbf{a}_i(0) = \mathbf{a}_i, &(4.19)\\ \dot{\mathbf{y}}_i^*(t) &= A\mathbf{y}^*(t) + \mu_i^*(t);\ \mathbf{y}_i^*(0) = \mathbf{y}_i, &(4.20)\\ \mu_i^*(t) &\in \arg\max \{\langle \mathbf{a}_i(t), \mathbf{u}\rangle \mid \mathbf{u} \in U\}. &(4.21) \end{aligned}$$

Scheme 2 (Over-Approximating $\delta_t(F)$)

1. For $i = 1, \ldots, m_h$:
   (a) Compute $\mathbf{a}_i(t)$ by solving (4.19).
   (b) Compute $\mu_i^*(t) = \arg\max \{\langle \mathbf{a}_i(t), \mathbf{u}\rangle \mid \mathbf{u} \in U\}$.
   (c) Compute $\mathbf{y}_i^*(t)$ by solving (4.20) with $\mu_i^*(t)$ obtained in step 1(b).
2. $\widehat{\delta}_t(F) = \bigcap_{i=1}^{m_h} \{\mathbf{x} \mid \langle \mathbf{a}_i(t), \mathbf{x}\rangle \leq \langle \mathbf{a}_i(t), \mathbf{y}_i^*(t)\rangle\}$.

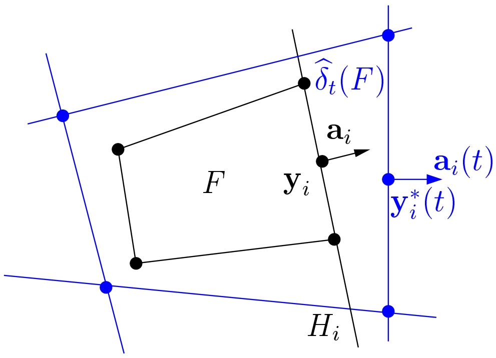


Figure 4.10: Over-approximation of $\delta_t(F)$.

The step 1(a) can be done by numerical integration. Note that if the input set U is a bounded convex polyhedron then $\mu_i^*(t)$ can be selected at one of its vertices at every time point t. The step 2 consists in intersecting all the half-spaces defined by the normal vectors $\mathbf{a}_i(t)$ and the points $\mathbf{y}_i^*(t)$ to obtain the convex polyhedron $\widehat{\delta}_t(F)$, which is an over-approximation of $\delta_t(F)$ (see Figure 4.10 for an illustration of the algorithm).

It is important to note that for linear systems without input (or with constant input) we can compute the set $\delta_t(F)$ exactly, but for systems with uncertain input the above scheme produces only an over-approximation of this set.

Conservative Approximations

With a view to over-approximating $\delta_{[0,r]}(F)$ based on the convex hull $C = conv(F \cup \widehat{\delta}_r(F))$, we need to estimate the amount by which C is bloated (by pushing outward its faces). This section is concerned with this estimation.

For every input $\mu \in \mathcal{U}$ we denote

$$\mu_m = \frac{1}{r}\int_0^r \mu(s)ds, \tag{4.22}$$

and let $\bar{\mu}$ be an input which is constant on the interval $[0, r]$:

$$\bar{\mu}(t) = \mu_m, \; t \in [0, r]. \tag{4.23}$$

Consider the trajectory $\xi_{\mathbf{x},\bar{\mu}}(t)$ starting from a point $\mathbf{x} \in F$ under the input $\bar{\mu}$. As is done for linear systems without input, we define the linear interpolation $s_{\mathbf{x},\bar{\mu}}(t)$ of $\xi_{\mathbf{x},\bar{\mu}}(t)$ on the interval $[0, r]$ as the line segment with two extreme points $\mathbf{x}$ and $\xi_{\mathbf{x},\bar{\mu}}(r)$. Let S_u be the set of all segments $s_{\mathbf{x},\bar{\mu}}(t)$ over all the initial points $\mathbf{x} \in F$ and all possible inputs $\mu \in \mathcal{U}$:

$$S_u = \{s_{\mathbf{x},\bar{\mu}}(t) \mid \mathbf{x} \in F \;\wedge\; t \in [0, r] \;\wedge\; \mu \in \mathcal{U}\}.$$

The exact reachable set $\delta_{[0,r]}(F)$ can be written similarly as

$$\delta_{[0,r]}(F) = \{\xi_{\mathbf{x},\mu}(t) \mid \mathbf{x} \in F \;\wedge\; t \in [0, r] \;\wedge\; \mu \in \mathcal{U}\}.$$

By convexity, the convex hull C contains the set S_u. Therefore, a sufficient bloating amount applied to C is the distance between S_u and $\delta_{[0,r]}(F)$. This distance is indeed the upper bound of $||\xi_{\mathbf{x},\mu}(t) - s_{\mathbf{x},\bar{\mu}}(t)||$ for all $t \in [0, r]$, for all $\mathbf{x} \in F$ and for all inputs $\mu \in \mathcal{U}$. We denote this bound by ϑ_u.

The estimation of ϑ_u is done in three steps:

1. Estimate the bound ϑ_1 on the distance between $\xi_{\mathbf{x},\mu}(t)$ and $\xi_{\mathbf{x},\bar{\mu}}(t)$ for all $t \in [0, r]$, for all $\mathbf{x} \in F$ and for all inputs $\mu \in \mathcal{U}$.

2. Estimate the bound ϑ_2 on the distance between $\xi_{\mathbf{x},\bar{\mu}}(t)$ and its linear interpolation $s_{\mathbf{x},\bar{\mu}}(t)$ for all $t \in [0, r]$, for all $\mathbf{x} \in F$ and for all inputs $\mu \in \mathcal{U}$.

3. The bound ϑ_u is then obtained by using the triangle inequality, that is,

$$\vartheta_u \leq \vartheta_1 + \vartheta_2. \tag{4.24}$$

To estimate the distance between $\xi_{\mathbf{x},\bar{\mu}}(t)$ and $\xi_{\mathbf{x},\mu}(t)$, we use the following proposition which is a direct consequence of the results on time-discretization of control systems obtained by V. Veliov in [115].

Proposition 4 *Given an arbitrary input $\mu \in \mathcal{U}$ and $r \geq 0$, let $\bar{\mu}$ be the input defined as in (4.22) and (4.23). Let $\xi_{\mathbf{x},\mu}(t)$ be the trajectory under μ and $\xi_{\mathbf{x},\bar{\mu}}(t)$ be the trajectory under $\bar{\mu}$. Then, for all $t \in [0,r]$*

$$||\xi_{\mathbf{x},\mu}(t) - \xi_{\mathbf{x},\bar{\mu}}(t)|| \leq 2M_u r^2 e^{||Ar||}$$

where M_u is the constant bounding $||\mathbf{u}||$.

Using Proposition 4, we have $\vartheta_1 = 2M_u r^2 e^{||Ar||}$.

We proceed now with the second step. Since $\bar{\mu}$ is constant on the interval $[0,r]$, we write

$$s_{\mathbf{x},\bar{\mu}}(t) = \mathbf{x} + \frac{t}{r}(e^{Ar}\mathbf{x} + \int_0^r e^{A(r-s)}ds\ \mu_m\ - \mathbf{x}).$$

Hence, for every $t \in [0,r]$

$$\begin{aligned}
\xi_{\mathbf{x},\bar{\mu}}(t) - s_{\mathbf{x},\bar{\mu}}(t) &= e^{At}\mathbf{x} + \int_0^t e^{A(t-s)}ds\ \mu_m - \mathbf{x} - \frac{t}{r}(e^{Ar}\mathbf{x} + \int_0^r e^{A(r-s)}ds\ \mu_m - \mathbf{x}) \\
&= [e^{At} - I + \frac{t}{r}I - \frac{t}{r}e^{Ar}]\mathbf{x} + [\int_0^t e^{A(t-s)}ds - \frac{t}{r}\int_0^r e^{A(r-s)}ds]\mu_m
\end{aligned}$$

Expanding $e^{A(t-s)}$ in a Taylor series and after direct calculations the above leads to

$$\begin{aligned}
\xi_{\mathbf{x},\bar{\mu}}(t) - s_{\mathbf{x},\bar{\mu}}(t) &= [e^{At} - I + \frac{t}{r}I - \frac{t}{r}e^{Ar}]\mathbf{x} + [\sum_{i=0}^{\infty}\frac{A^i t^{i+1}}{(i+1)!} - \frac{t}{r}\sum_{i=0}^{\infty}\frac{A^i r^{i+1}}{(i+1)!}]\mu_m. \\
&= [\frac{A^2 t(t-r)}{2} + \sum_{i=3}^{\infty}\frac{A^i t(t^{i-1} - r^{i-1})}{i!}]\mathbf{x} + [\frac{At(t-r)}{2} + \sum_{i=2}^{\infty}\frac{A^i t(t^i - r^i)}{(i+1)!}]\mu_m \\
&= \frac{t(t-r)}{2}(A^2\mathbf{x} + A\mu_m) + \sum_{i=3}^{\infty}\frac{A^i\mathbf{x} + A^{i-1}\mu_m}{i!}t(t^{i-1} - r^{i-1})
\end{aligned}$$

Thus, we have for all $t \in [0,r]$, for all $\mathbf{x} \in F$ and for all $\mu \in \mathcal{U}$

$$||\xi_{\mathbf{x},\bar{\mu}}(t) - s_{\mathbf{x},\bar{\mu}}(t)|| \leq \frac{1}{8}(M||A||^2 + M_u||A||)r^2 + \mathcal{O}(r^3) = \vartheta_2$$

where M is the constant bounding the norm $||\mathbf{x}||$.

Using (4.24) the bound ϑ_u on $||\xi_{\mathbf{x},\mu}(t) - s_{\mathbf{x},\bar{\mu}}(t)||$ for all $t \in [0,r]$, for all inputs $\mu \in \mathcal{U}$ and for all $\mathbf{x} \in F$ is

$$||\xi_{\mathbf{x},\mu}(t) - s_{\mathbf{x},\bar{\mu}}(t)|| \leq \frac{1}{8}(M||A||^2 + M_u||A||)r^2 + 2M_u r^2 e^{||Ar||} + \mathcal{O}(r^3) = \vartheta_u. \tag{4.25}$$

This means that by bloating the convex hull $C = conv(F \cup \widehat{\delta}_r(F))$ by the amount ϑ_u given in the above formula we obtain a polyhedron which is guaranteed to contain $\delta_{[0,r]}(F)$. Formula (4.25) also shows that, like in linear systems without input, the bloating amount is of the order $\mathcal{O}(r^2)$. The scheme for over-approximating $\delta_{[0,r]}(F)$ is presented below.

Scheme 3 (Over-approximating $\delta_{[0,r]}(F)$)

1. Compute the convex polyhedron $\widehat{\delta}_r(F)$ over-approximating $\delta_r(F)$ using Scheme 2.

2. Compute $C = conv\{F \cup \widehat{\delta}_r(F)\}$.

3. Bloat C by the amount ϑ_u given in (4.25), that is, $C_o = bloat(C, \vartheta_u)$.

4. Compute an orthogonal polyhedron $G = grid_o(C_o)$, which is guaranteed to contain $\delta_{[0,r]}(F)$.

The algorithm for over-approximating the whole reachable set from F can be obtained from Algorithm 4 with the computation in one iteration replaced by the above scheme.

4.6 Examples

Let us now illustrate our approach by means of three examples. The results and the running times were obtained using *d/dt* on a Sun Ultra Sparc-10.

Two Linear Systems Without Input

Consider first a 3-dimensional system whose dynamics and initial set are given below.

$$A = \begin{pmatrix} -1.0 & -4.0 & 0.0 \\ 4.0 & -1.0 & 0.0 \\ 0.0 & 0.0 & 0.5 \end{pmatrix}, \quad F = [0.025, 0.05] \times [0.1, 0.15] \times [0.05, 0.1];$$

Figure 4.13 shows the reachable set $\delta_{[0,3.5]}(F)$, computed in $8s$ (with run-time visualization). The time step is $r = 0.2$ and the grid size is 0.005.

The second example is a 6-dimensional system of the Jordan form with the following matrix

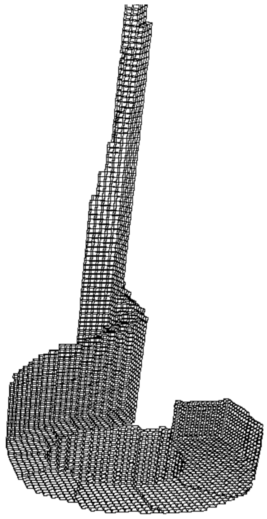


Figure 4.11: The reachable set of the 3-dimensional system (the system diverges in dimension 3).

and initial set.

$$A = \begin{pmatrix} -0.8 & 1.0 & 0.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & -0.8 & 1.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & -0.8 & 1.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & -0.8 & 1.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & -0.8 & 1.0 \\ 0.0 & 0.0 & 0.0 & 0.0 & 0.0 & -0.8 \end{pmatrix},$$

$$F = [0.025, 0.05] \times [0.01, 0.03] \times [0.05, 0.15] \times \\ [-0.05, 0.1] \times [-0.05, 0.1] \times [0.03, 0.08] \times [-0.01, 0.05].$$

The reachable set of the system, computed with time step $r = 0.15$ and grid size $\beta = 0.01$, appears in Figure 4.12. The running time is $60s$.

A Linear System With Uncertain Input

We consider now a 4-dimensional system with uncertain input, adapted from Example 4.5.1 of [71], pp. 279-285.

The dynamics of the system is $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}$ where

$$A = \begin{pmatrix} 0.0 & 1.0 & 0.0 & 0.0 \\ -8.0 & 0.0 & 0.0 & 0.0 \\ 0.0 & 0.0 & 0.0 & 1.0 \\ 0.0 & 0.0 & -4.0 & 0.0 \end{pmatrix}$$

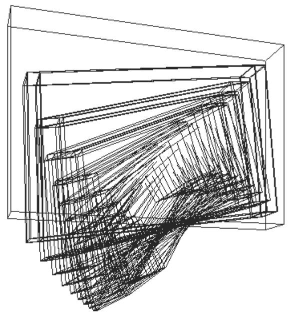


Figure 4.12: The 6-dimensional system: the projection of the reachable set at time points kr on dimensions 2, 3, and 4 (r is the time step). One can see that the system converges to the origin.

and the input $\mathbf{u}$ ranges inside the hyper-rectangle $[-1.0, 1.0] \times [-0.01, 0.01] \times [-1.0, 1.0] \times [-0.01, 0.01]$.

The initial set is a hyper-rectangle $F = [-1.0, 1.0] \times [0.0, 2.0] \times [-1.0, 1.0] \times [0.0, 2.0]$. The reachability analysis is performed for bounded time $T_m = 0.35$, i.e. we compute the reachable set $\delta_{[0,0.35]}(F)$. With time step $r = 0.05$ and grid size $\beta = 0.35$, the running time is $18s$. In Figure 4.13 one can see the evolution of the projection of the reachable set on dimensions 3 and 4 over time, similar to the results in [71] obtained by using ellipsoidal techniques.

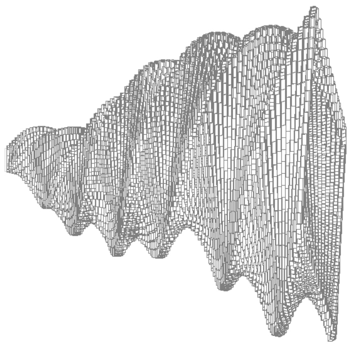


Figure 4.13: The reachable set of the system with uncertain input projected on x_3, x_4 and t.

Remark 2

Experiments with various examples showed that a major factor that influences the computation time is the dimensionality. For a 7-dimensional system similar to the example of the Jordan form, the computation took 220 seconds. In addition, the complexity of the algorithm depends on the granularity of the orthogonal approximations and the coupling of the continuous variables.

4.7 Summary and Related Work

We have proposed a technique for computing over-approximations and under-approximations of reachable sets of linear systems. Our technique has the advantage of not propagating over-approximation error and thus allows to guarantee the desired accuracy for unbounded time. Moreover, as will be shown later, it can be easily adapted to the verification and synthesis of hybrid systems. We have also showed an extension of this technique to linear systems with uncertain input.

There have been other works on computing reachable sets of linear continuous systems. The closest work to ours is that of Chutinan and Krogh [31]. The authors consider linear systems with *constant input* and use a method similar to ours to compute polyhedral over-approximations of the sets $\delta_{[kr,(k+1)r]}(F)$ ("flow pipe" segments). Their approach differs from ours in their way to guarantee conservative approximations by solving some optimization problems. Moreover, by exploiting some linear system properties, the flow pipe is computed only for the first time interval $[0, r]$, to which some linear transformations are then applied to obtain the flow pipe segments on the next intervals. However, this idea seems difficult to extend to systems with time-varying, uncertain input since these properties no longer hold. In addition, flow pipes are represented as unions of convex polyhedra, which makes termination harder to check than in our method.

Another approach is to approximate reachable sets by classes of domains of some fixed shapes. Among the methods in this direction are those based on ellipsoidal techniques proposed by Kurzhanski and Varaiya [71, 72]. Approximations via parallelotopes were also investigated by Kostousova in [66]. These methods can deal with more general systems than those we consider, namely linear control systems with time-varying coefficients. Recent results show that reachable sets can be represented exactly by parametrized families of ellipsoids or parallelotopes. However, these works are mainly concerned with the approximation of reachable sets in discrete time.

Chapter 5

Reachability Analysis of Non-Linear Continuous Systems

This chapter is concerned with the reachability problem for non-linear continuous systems. We propose a technique which is a variation of the technique suggested by Mark Greenstreet in [47] for over-approximating reachable sets of two-dimensional systems. The main advantage of our technique is that it can be applied to any dimension. We describe an implementation of this technique along with a procedure of error control. Finally, we illustrate our techniques with some examples. This chapter reviews the results presented in [39].

5.1 The Face Lifting Concept

Consider a continuous non-linear system $\mathcal{C} = \{\mathcal{X}, f\}$ where $\mathcal{X} \subseteq \mathbb{R}^n$. The dynamics of the system is defined by the differential equation

$$\dot{\mathbf{x}} = f(\mathbf{x}). \tag{5.1}$$

We assume that the function f is Lipschitz.

As before, we are interested in the reachability problem: given an initial set $F \subseteq \mathcal{X}$ we want to compute an over-approximation of the set $\delta(F)$ of states reachable from F by the system $\mathcal{C}$.

Trying to compute reachable sets of arbitrary non-linear systems, our first observation is that the information obtained from simulations of a finite number of trajectories is, in general, not sufficient to derive an over-approximation of the reachable set (even in discrete time). For some special non-linear systems, if we can prove that this property still holds then the technique for linear systems, presented in the previous chapter, might be useful. However, approximations by convex hull are often too coarse. In the following we describe a technique, inspired by the work of Mark Greenstreet in [47], which we call *face lifting*.

The technique is based, first of all, on the following basic observation concerning *continuity of trajectories*. Consider a trajectory $\xi_{\mathbf{x}}$ starting from some interior point $\mathbf{x} \in F$. It is clear that the trajectory $\xi_{\mathbf{x}}$ either remains in F forever or traverses the boundary of F after some time. In the former case, $\xi_{\mathbf{x}}$ does not contribute any reachable points outside F. In the latter, if a point $\mathbf{y}$ outside F is reachable from $\mathbf{x}$ after t_1 time then there exists a point $\mathbf{x}'$ on the boundary ∂F of F and $t' < t_1$ such that $\mathbf{y} = \xi_{\mathbf{x}}(t')$. It then follows that

$$\delta_{[0,t]}(F) = F \cup \delta_{[0,t]}(\partial F). \tag{5.2}$$

Hence, to compute $\delta(F)$, it is suffcient to look at the boundary of F.

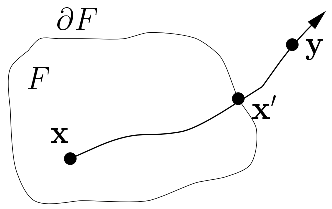


Figure 5.1: Continuity of trajectories.

We assume now that F is a *convex polyhedron*. The boundary of F is then the union of its *faces*. Let E be the set of faces of F. Consider a face $e \in E$ whose supporting plane is written as

$$P(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle = c_e\}$$

where $\mathbf{n}(e)$ is the *unit outward normal* to e. Thus, the polyhedron F lies inside the half-space $H(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle \leq c_e\}$ and can be written as

$$F = \bigcap_{e \in E} H(e).$$

We start by constructing the neighborhoods of the faces of F as follows. We define the neighborhood of F as the polyhedron obtained by pushing outward each face e of F by an amount v_e, denoted by $N_{\boldsymbol{v}}(F)$ where $\boldsymbol{v} = \{v_e \mid e \in E\}$. In other words,

$$N_{\boldsymbol{v}}(F) = \bigcap_{e \in E} \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle \leq c_e + v_e\}.$$

Next, we define the neighborhood $N(e)$ of face e as

$$N(e) = H_o(e) \cap N_{\boldsymbol{v}}(F) \tag{5.3}$$

where $H_o(e)$ is the half-space defined as $H_o(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle \geq c_e\}$.

Figure 5.2 illustrates the neighborhood construction with a two-dimensional example. The set F here is a polygon with vertices $\{a, b, c, d\}$, and $N_{\boldsymbol{v}}(F)$ is the dotted polygon. The neighborhood $N(e)$ of the face e is then the shaded polygon.

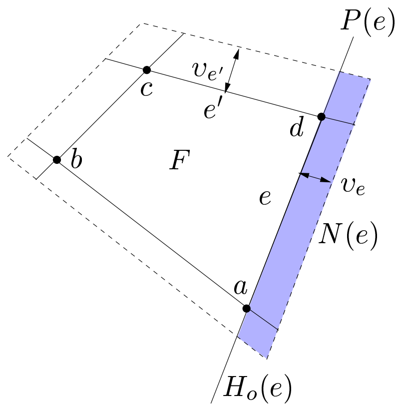


Figure 5.2: Neighborhood construction for a polygon.

Let $f_e(\mathbf{x})$ denote the *outward component* of $f(\mathbf{x})$ relative to e, that is, the projection of $f(\mathbf{x})$ on $\mathbf{n}(e)$:

$$f_e(\mathbf{x}) = \langle \mathbf{n}(e), f(\mathbf{x}) \rangle,$$

and let $\hat{f}_e$ denote the maximum of f_e over $N(e)$:

$$\hat{f}_e = \max\{f_e(\mathbf{x}) \mid \mathbf{x} \in N(e)\}. \tag{5.4}$$

The following assumption should be kept in mind during subsequent discussions since it plays a crucial role in our analysis.

A trajectory starting from the boundary of F which goes out of F at some time $t \in (0, r]$ is called *outward trajectory*.

Assumption 1 *All trajectories starting from the boundary of F stay in $N_{\boldsymbol{v}}(F)$ for at least r time.*

In Section 5.3, we will show how this important assumption can be guaranteed to hold by properly choosing r and v_e.

Consider an outward trajectory $\xi_{\mathbf{x}}$ starting from point $\mathbf{x} \in e$. By solution of the differential equation (5.1) we have

$$\xi_{\mathbf{x}}(t) = \mathbf{x} + \int_0^t f(\mathbf{x}(s))ds.$$

Thus,

$$\langle \mathbf{n}(e), \xi_{\mathbf{x}}(t) \rangle = c_e + \int_0^t f_e(\mathbf{x}(s))ds.$$

Due to (5.4), the trajectory $\xi_{\mathbf{x}}$ at every time point $t \in [0, r]$ satisfies the following

$$\langle \mathbf{n}(e), \xi_{\mathbf{x}}(t) \rangle \leq c_e + t\hat{f}_e.$$

We denote

$$\bar{f}_e = \begin{cases} \hat{f}_e & \text{if } \hat{f}_e > 0, \\ 0 & \text{otherwise.} \end{cases}$$

It is not hard to see that for all $t \in [0, r]$

$$\langle \mathbf{n}(e), \xi_{\mathbf{x}}(t) \rangle \leq c_e + r\bar{f}_e. \tag{5.5}$$

This means that all the points reachable from e within the time interval $[0, r]$ remain inside the half-space defined as $H_l(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle \leq c_e + r\bar{f}_e\}$. Geometrically speaking, $H_l(e)$ is the half-space resulting from lifting $H(e)$ in the direction $\mathbf{n}(e)$ by the amount $r\bar{f}_e$. It then follows that

$$\delta_{[0,r]}(e) \subseteq H_l(e).$$

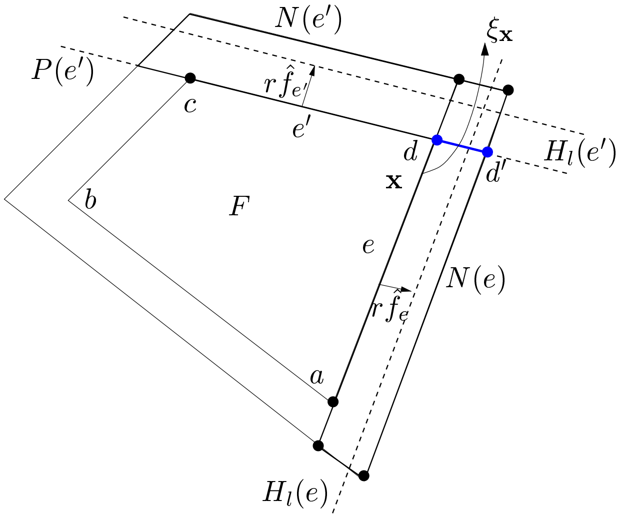


Figure 5.3: Illustration of the face lifing concept.

Now, consider a face e' which is *adjacent to* e. Similarly, we have $\delta_{[0,r]}(e') \subseteq H_l(e')$.

Claim 5 *Any outward trajectory from e cannot leave the half-space $H_l(e')$ at time $t \in [0, r]$.*

Sketch of Proof
To prove this, we observe that by convexity any trajectory from F can leave $H_l(e')$ only by crossing the hyperplane $P(e')$ (see Figure 5.3). Consider now an outward trajectory $\xi_{\mathbf{x}}$ starting from a point $\mathbf{x} \in e$. By construction $\xi_{\mathbf{x}}$ stays within $N(e)$ during the time interval $[0, r]$, which means that $\xi_{\mathbf{x}}$ can go out of $H_l(e')$ only by crossing the set $M = P(e') \cap N(e)$ (in Figure 5.3 the set M is the line segment dd'). Clearly, $M \subseteq N(e')$. Since any point $\mathbf{y}$ in M satisfies $\langle \mathbf{n}(e'), \mathbf{y} \rangle = c_{e'}$ and, in addition, the maximum of $f_{e'}$ is calculated over $N(e')$, it follows by (5.5) that $\xi_{\mathbf{x}}$ can leave $H_l(e')$ only at time $t \geq r$. ∎

Lemma 6 *Let* $F_l = \bigcap_{e \in E} H_l(e)$. *Then,* $\delta_{[0,r]}(F) \subseteq F_l$.

Sketch of Proof
To prove the lemma, it suffices to prove that all the trajectories from a face e of F stay inside the polyhedron F_l for at least r time. Due to Claim 5, any outward trajectory from e cannot leave the lifted half-spaces of its adjacent faces during the time interval $[0, r]$. In addition, any inward trajectory from e can leave F only by crossing another face e' of F and thus stays inside $H_l(e')$ for $t \geq r$ time. ■

Lemma 6 suggests the following scheme for over-approximating $\delta_{[0,r]}(F)$.

Scheme 4 (Over-approximating $\delta_{[0,r]}(F)$**)**

1. *Neighborhood construction*
   - For every face e of F, compute the appropriate v_e. The description of this is deferred to Section 5.3.
   - Construct the neighborhood $N(e)$ for each face e, as shown in (5.3).
2. *Lifting operation*
   - For every face e of F:
     - Calculate $\hat{f}_e$, i.e. the maximum of $f_e(\mathbf{x})$ over $N(e)$, and determine $\bar{f}_e$ accordingly.
     - Lift the half-space $H(e)$ outward by the amount $r\bar{f}_e$ to obtain $H_l(e)$. Note that if $\hat{f}_e$ is negative or null $H_l(e)$ is just $H(e)$.
   - Intersect all the half-spaces $H_l(e)$ to obtain a new convex polyhedron which is an over-approximation of $\delta_{[0,r]}(F)$.

As an illustration, consider the two-dimensional example shown in Figure 5.4 where the initial set is a polygon F. Some values of f on its edges are sketched. Only edges e_1, e_2 have a positive outward component of f, and the corrresponding half-spaces are thus pushed outward. The other half-spaces remain unchanged. The intersection of these half-spaces gives the polygon F_l which contains $\delta_{[0,r]}(F)$.

An important observation is that the above scheme may produce poor approximations since the calculation of the lifting amount for every face e is based uniquely on the maximum of f_e over the neighborhood $N(e)$. In order to avoid excessively conservative approximations, the neighborhood should be small enough and, as a result, some of the faces must be split a priori. Note that the approximation scheme works for convex polyhedra, which means that face splitting amounts to partitioning F into small convex polyhedra P_i

$$F = \bigcup_{i=1}^{m_p} P_i$$

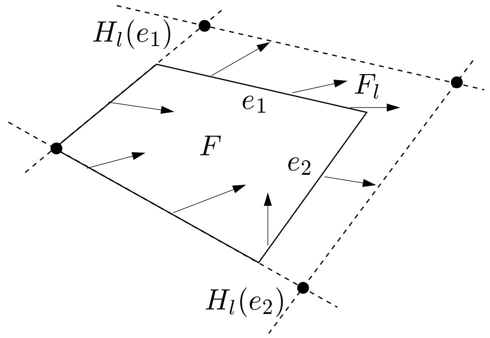


Figure 5.4: Illustration of face lifting.

where m_p is the number of polyhedra in the partition. Then,

$$\delta_{[0,r]}(F) = \bigcup_{i=1}^{m_p} \delta_{[0,r]}(P_i).$$

Being a union of convex polyhedra, $\delta_{[0,r]}(F)$ need not be convex. Consequently, when coming to compute the whole reachable set from F, we are faced again with the problem of *representing and manipulating non-convex polyhedra*.

In two dimensions, the technique can be easily implemented since an arbitrary polygon can be represented by an ordered list of its vertices, and its faces are simply the line segments defined by two neighboring vertices in the list. Furthermore, thanks to the special properties of planar geometry, we can compute the successors of a non-convex polygon without decomposing it into convex ones as follows. Instead of lifting the half-spaces, we just need to lift the supporting planes of the faces and then the vertices of the new polygon can be obtained by intersecting the new supporting planes corresponding to adjacent faces. As an example, consider the polygon F shown in Figure 5.5. Since the vertex $\mathbf{v}_2$ belongs to the faces e_1 and e_2, the new vertex $\mathbf{v}_2'$ is obtained by intersecting the lifted supporting planes of these faces. This is indeed the basis of the technique suggested by Mark Greenstreet in [47] for two dimensional systems.

Nevertheless, this method cannot be generalized to three or more dimensions since the representation by vertices is not applicable to higher dimensions. For this reason, we restrict ourselves to orthogonal polyhedra which offer important advantages regarding the required operations, namely

1. The faces of an orthogonal polyhedron are $(n-1)$-dimensional hyper-rectangles and can be systematically enumerated.

2. Using orthogonal polyhedra, we can benefit from relatively efficient algorithms for the union operation and convex decomposition.

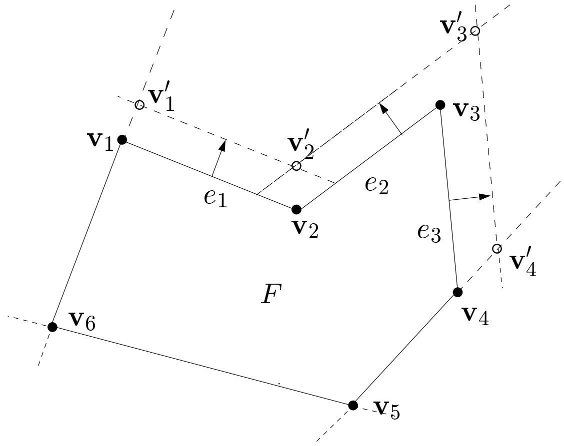


Figure 5.5: Face lifting on a non-convex polygon.

Our approximation scheme takes as input convex polyhedra; therefore non-convex orthogonal polyhedra should be decomposed a priori into hyper-rectangles. Since hyper-rectangles are closed under the lifting operation, so are orthogonal polyhedra (see Figure 5.6).

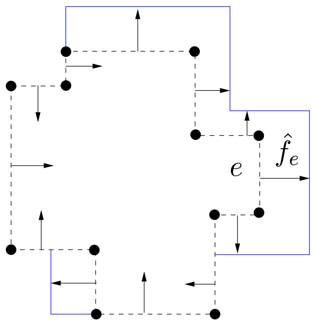


Figure 5.6: Orthogonal polyhedra are closed under the lifting operation: $\hat{f}_e$ are always parallel to one of the axes and the resulting polyhedron is orthogonal.

5.2 Computation Procedure

In this section, we describe an algorithm for face lifting using orthogonal polyhedra. From now on we assume that the initial set F is an orthogonal polyhedron. We first introduce some notations.

Additional Notations

Let $\mathbf{x}$ be a point in $\mathbb{R}^n$ and $i \in \{1, \ldots, n\}$ be a direction. The i^{th} coordinate of $\mathbf{x}$ is denoted by x_i. For a real number d, we denote by $\lfloor d \rfloor$ the integer part of d.

A hyper-rectangle b in $\mathbb{R}^n$ can be written as $b = [l_1, u_1] \times \ldots \times [l_n, u_n]$. Let $E(b) = \{e_1^-, e_1^+, \ldots, e_n^-, e_n^+\}$ be the set of faces of b where e_i^- is the face consisting of points $\mathbf{x}$ satisfying $x_i = l_i$ and e_i^+ the face consisting of points $\mathbf{x}$ satisfying $x_i = u_i$. In other words, e_i^- and e_i^+ are the faces whose outward normal vector is parallel to the axis i and oriented to the negative and positive direction, respectively.

The projection of the function $f(\mathbf{x})$ on the outward normal to e_i^- is $-f_i(\mathbf{x})$ where $f_i(\mathbf{x})$ is the i^{th} component of $f(\mathbf{x})$. Similarly, the projection of $f(\mathbf{x})$ on the outward normal to e_i^+ is $f_i(\mathbf{x})$.

We define the *lifting operator* for a hyper-rectangle as follows.

Definition 20 *The lifting operator Λ is defined for a hyper-rectangle $b = [l_1, u_1] \times \ldots \times [l_n, u_n]$ in $\mathbb{R}^n$ and two vectors $\mathbf{d}^-$, $\mathbf{d}^+ \in \mathbb{R}^n$ as*

$$\Lambda(b, \mathbf{d}^-, \mathbf{d}^+) = [l_1 + d_1^-, u_1 + d_1^+] \times \ldots \times [l_n + d_n^-, u_n + d_n^+].$$

Thus, $\mathbf{d}_i^-$ and $\mathbf{d}_i^+$ are the lifting amounts for the faces e_i^- and e_i^+, respectively.

5.2.1 Reachability Algorithm

Suppose the orthogonal polyhedron F (defined over a uniform grid $\mathcal{G}_\beta$) is decomposed into m_b non-overlapping hyper-rectangles b_j and can be thus represented as

$$F = \bigcup_{j=1}^{m_b} b_j.$$

Note that the above representation is not unique since there can be more than one convex decomposition for an orthogonal polyhedron. Although the problem does not affect the correctness of our subsequent reasoning, the decomposition that is used may have impact on the efficiency of the algorithm. This issue will be discussed later.

It follows that

$$\delta(F) = \bigcup_{j=1}^{m_b} \delta(b_j).$$

We have shown earlier that $\delta(F) = F \cup \delta(\partial F)$. Therefore

$$\delta(F) = F \cup \bigcup_{j=1}^{m_b} \delta(\partial b_j \cap \partial F).$$

Hence, to compute $\delta(F)$ it suffices to look at those faces of the hyper-rectangles b_j which lie on the boundary of F.

For the time being we suppose that no face splitting is needed and the time step r and the neighborhoods are given. The tuning of these parameters to achieve the desired accuracy is presented in Section 5.3. The basic algorithm for face lifting using orthogonal polyhedra is sketched below.

Algorithm 6 (Face Lifting Algorithm)

P^0 := F;
repeat $k = 0, 1, 2, \ldots$
  for all $b_j \in \mathit{decomp}(P^k)$ {
    $\mathbf{d}^-$:= $\mathbf{0}$; $\mathbf{d}^+$:= $\mathbf{0}$;
    for all $i \in \{1, \ldots, n\}$ {
      if $(e_i^- \subseteq \partial P^k)$ {
        $\hat{f} := \max\{-f_i(\mathbf{x}) \mid \mathbf{x} \in N(e_i^-)\}$;
        if $(\hat{f} > 0)$ $d_i^- := \beta\lfloor r\hat{f}/\beta \rfloor$;
      }
      if $(e_i^+ \subseteq \partial P^k)$ {
        $\hat{f} := \max\{f_i(\mathbf{x}) \mid \mathbf{x} \in N(e_i^+)\}$;
        if $(\hat{f} > 0)$ $d_i^+ := \beta(\lfloor r\hat{f}/\beta \rfloor + 1)$;
      }
    }
    $b'_j := \Lambda(b_j, \mathbf{d}^-, \mathbf{d}^+)$;
    $P^{k+1} := P^k \cup b'_j$;
  }
until $P^{k+1} = P^k$
return P^{k+1}

The algorithm uses the function *decomp* which takes as input an orthogonal polyhedron and returns a convex decomposition of the polyhedron in form of a list of non-overlapping hyper-rectangles. In each iteration the orthogonal polyhedron P^{k+1} is computed as follows. For each hyper-rectangle b_j in P^k, we compute the lifting amount for every face of b_j which lies on the boundary of P^k and then apply the lifting operator to b_j. This gives a new hyper-rectangle b'_j, which is next added to P^k. The lifting amount for a face e whose $\hat{f}_e$ is negative is simply zero.

The following theorem states the correctness of Algorithm 7.

Theorem 4 *If the state space $\mathcal{X}$ of the system is bounded, then Algorithm 7 always terminates and produces an over-approximation of $\delta(F)$.*

Proof
We need to prove: (a) if the algorithm terminates then it produces an over-approximation of the reachable set, (b) the algorithm always terminate. We begin with the proof of (a).

By construction, in each iteration the hyper-rectangle b'_j is guaranteed to include all the points reachable from $\partial b_j \cap \partial P^k$. It then follows that

$$\bigcup_{j=1}^{m_b} b'_j \supseteq \bigcup_{j=1}^{m_b} b_j \cup \delta_{[0,r]}(\partial b_j \cap \partial P^k) = P^k \cup \delta_{[0,r]}(\bigcup_{j=0}^{m_b} (\partial b_j \cap \partial P^k)).$$

Thus,

$$P^{k+1} \supseteq P^k \cup \delta_{[0,r]}(\partial P^k). \tag{5.6}$$

The above leads to $P^k \supseteq P^k \cup \delta_{[0,r]}(P^k)$, and hence $P^k \supseteq \delta_{[0,kr]}(F)$.

If the algorithm terminates, that is, the condition $P^{k+1} = P^k$ is satisfied, formula (5.6) implies that $\delta_{[0,r]}(\partial P^k) = \emptyset$, which means that all the trajectories from P^k stay in P^k forever and so do all the trajectories from $\delta_{[0,kr]}(F)$. We then deduce that when the algorithm terminates, the computed set P^{k+1} is indeed an over-approximation of $\delta(F)$. This proves (a).

The proof of termination is based on the fact that the state space $\mathcal{X}$, being bounded, can be represented as the union of a *finite number* of orthogonal polyhedra. ■

5.2.2 Computational Aspects

We have presented an algorithm for over-approximating reachable sets of continuous non-linear systems. When it comes to a real implementation, a common desired feature is that the implemented algorithm fulfills its goal not only accurately but also economically. In the context of Algorithm 7, several details must be considered.

Face Splitting and Variable Time Steps

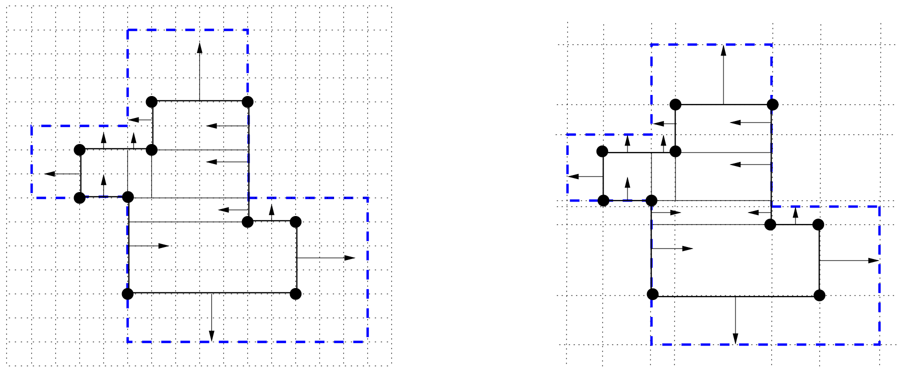
As mentioned earlier, the lifting amount for the face e of a hyper-rectangle, determined based on the maximum of f_e over its neighborhood, guarantees over-approximations but at the same time can make the result too conservative especially when f_e has a large variation over the neighborhood. In this case, more accurate approximations can be achieved by splitting the hyper-rectangle into smaller ones.

In addition, depending upon the variation of f near the boundary of P^k, the time step for one iteration can be different from another. The use of variable time steps can improve significantly the time-efficiency of the algorithm.

Consequently, for both accuracy and efficiency purposes, the algorithm should exert some adaptative control over the progress of the computation: making frequent changes in the time steps and splitting the hyper-rectangles if necessary. We devote the next section to this important issue.

Convex Decomposition

As we have already seen, our algorithm decomposes non-convex orthogonal polyhedra into hyper-rectangles before applying the lifting operation. Obviously, such a decomposition is not unique in the sense that there is more than one way to represent an orthogonal polyhedron as a union of non-overlapping hyper-rectangles. It is clear that the complexity of the algorithm depends on the number of hyper-rectangles in the decomposition. Nevertheless, it is not always advantageous to decompose the orthogonal polyhedron into few large hyper-rectangles if partitioning is then needed. On the other hand, in some cases we can merge small hyper-rectangles sharing the same face of the orthogonal polyhedron.



(a) (b)

Figure 5.7: Illustration of the face lifting algorithm on a uniform grid (a) and a non-uniform one (b). The initial orthogonal polygon F is decomposed into rectangles, whose faces lying on the boundary of F are annotated by the arrows representing $r\hat{f}_e$. In (b) the algorithm can cause a refinement of the grid.

Uniform and Non-Uniform Grids

Notice that our algorithm can be applied to both orthogonal polyhedra defined on *uniform* and *non-uniform* grids. The main advantage of non-uniform grids over uniform ones is the accuracy of approximation. With an uniform grid, we need to push every face further to the next grid point, which sometimes creates unnecessary over-approximation beyond what is inherent in the face lifting alone (see Figure 5.7-(a)). With a non-uniform grid we can push the faces as little as we want (see Figure 5.7-(b)). This, however, implies that the grid should be changed dynamically over the execution, and thus the algorithm may be less efficient in terms of computation time. Therefore, one should consider a combination of the desirable features of the two to achieve better performance and accuracy.

5.3 Error Analysis

This section is concerned with the problem of determining two important parameters, namely the time step and the size of neighborhoods, with respect to the correctness and accuracy of the algorithm. We establish the relation between these two parameters which guarantees the desired properties by analyzing the local error in the approximation and then examine the problem of error accumulation.

5.3.1 Local Error Control

Recall that the correctness of our algorithm is guaranteed by Assumption 1 which states that all trajectories departing from the boundary of F remain in $N_{\boldsymbol{v}}(F)$ within the time interval $[0, r]$. We now show how to determine the time step r and the size of the neighborhood $\boldsymbol{v}$ so that this assumption is always fulfilled. We also prove that *locally the error can be made arbitrarily small* by fine-tuning these parameters.

Our discussion is preceded by some auxiliary results related to Lipschitz functions [61].

Lemma 7 *Let f, $g : \mathbb{R}^n \to \mathbb{R}^n$ be continuous functions. Suppose that for all $\mathbf{x} \in \mathbb{R}^n$,*

$$||f(\mathbf{x}) - g(\mathbf{x})|| \leq \epsilon.$$

Let K be a Lipschitz constant for f. If $\xi(t)$, $\xi'(t)$ are solutions to $\dot{\mathbf{x}} = f(\mathbf{x})$ and $\dot{\mathbf{x}} = g(\mathbf{x})$, respectively, and $\xi(0) = \xi'(0)$, then

$$||\xi(t) - \xi'(t)|| \leq \frac{\epsilon}{K}(e^{Kt} - 1)$$

for all $t \geq 0$.

Proof
Since $\xi(0) = \xi'(0)$, for $t \geq 0$ we have

$$\xi(t) - \xi'(t) = \int_0^t [f(\xi(s)) - g(\xi'(s))]\, ds.$$

Hence,

$$\begin{aligned} ||\xi(t) - \xi'(t)|| &\leq \int_0^t ||f(\xi(s)) - f(\xi'(s))||\, ds + \int_0^t ||f(\xi'(s)) - g(\xi'(s))||\, ds \\ &\leq \int_0^t K||\xi(s) - \xi'(s)|| ds + \int_0^t \epsilon\, ds. \end{aligned}$$

Denote $\kappa(t) = ||\xi(t) - \xi'(t)||$. The above becomes

$$\kappa(t) \leq K \int_0^t (\kappa(s) + \frac{\epsilon}{K}) ds.$$

Using Grownwall's inequality, we obtain

$$\kappa(t) + \frac{\epsilon}{K} \leq \frac{\epsilon}{K} e^{Kt},$$

which yields Lemma 7. ■

Lemma 8 *Let $f : \mathbb{R}^n \to \mathbb{R}^n$ be a Lipschitz function with a Lipschitz constant K. Let $\xi(t)$ be a solution to $\dot{\mathbf{x}} = f(\mathbf{x})$. Then*

$$||\xi(t) - \xi(0)|| \leq \frac{||f(\xi(0))||}{K}(e^{Kt} - 1).$$

Proof
By solution of the differential equation we have

$$||\xi(t) - \xi(0)|| = || \int_0^t f(\xi(s))\, ds|| \leq \int_0^t ||f(\xi(s))||\, ds.$$

From the Lipschitz conditions, it is not hard to see that $||f(\xi(s))|| \leq ||f(\xi(0))|| + K||\xi(s) - \xi(0)||$. Therefore,

$$||\xi(t) - \xi(0)|| \leq ||f(\xi(0))||t + \int_0^t K||\xi(s) - \xi(0)||\, ds.$$

Using Grownwall's inequality again, this leads to

$$||\xi(t) - \xi(0)|| \leq ||f(\xi(0))||t + \int_0^t ||f(\xi(0))||sKe^{K(t-s)}\, ds.$$

Developing the integral in the right-hand side of the above and after obvious calculations we establish the lemma. ■

We return now to the problem of finding the appropriate time step and the size of neighborhoods which fulfill Assumption 1 and ensure the local error under the pre-specified tolerance, say, ϵ.

The lifting operation, when applied for a face e, can be thought of as replacing the real derivative $f_e(\mathbf{x})$ in the neighborhood $N(e)$ by a constant derivative $\hat{f}_e$. Denote

$$\delta_{f_e} = \max\{||f_e(\mathbf{x}) - \hat{f}_e|| \mid \mathbf{x} \in N(e)\}.$$

Let $\varrho(e)$ be the diameter of $N(e)$. If L is a Lipschitz constant of f_e then we have

$$\delta_{f_e} \leq \varrho(e)L. \tag{5.7}$$

Let $\xi_{\mathbf{x}}$ be the real trajectory starting from a point $\mathbf{x} \in e$, and let $\hat{\xi}_{\mathbf{x}}$ be the trajectory corresponding to the constant derivative $\hat{f}_e$. Due to Lemma 7, the bound on the distance between $\xi_{\mathbf{x}}(t)$ and $\hat{\xi}_{\mathbf{x}}(t)$ for all initial points $\mathbf{x} \in e$ and for all $t \geq 0$ is given by the following inequality:

$$||\xi_{\mathbf{x}}(t) - \hat{\xi}_{\mathbf{x}}(t)|| \leq \frac{\delta_{f_e}}{L}(e^{tL} - 1).$$

Therefore, in order to keep the local error under ϵ, the time step r and the neighborhood diameter $\varrho(e)$ must satisfy

$$\varrho(e)(e^{rL} - 1) \leq \epsilon. \tag{5.8}$$

In addition, we need to guarantee that all the trajectories from the boundary of F remain in $N_{\boldsymbol{\upsilon}}(F)$ for at least r time. Recall that υ_e is the amount of pushing outward the face e in the neighborhood construction. We denote by M_e the maximum value of $||f(\mathbf{x})||$ for all $\mathbf{x} \in e$. It follows by Lemma 8 that for all $t \in [0, r]$

$$||\xi_{\mathbf{x}}(t) - \mathbf{x}|| \leq \frac{M_e}{L}(e^{Lr} - 1).$$

Thus, the time step r must satisfy the following

$$\frac{M_e}{L}(e^{Lr} - 1) \leq \upsilon_e,$$

or equivalently,

$$r \leq \frac{1}{L}\ln(\frac{\upsilon_e L}{M_e} + 1). \tag{5.9}$$

Formulas (5.8) and (5.9) establish the relation between the neighborhood, the time step and the local error. Replacing r in (5.8) with the term in the right-hand side of the inequality (5.9) and after direct calculations we obtain

$$\varrho(e)\upsilon_e \leq \frac{\epsilon M_e}{L}. \tag{5.10}$$

We now apply the above analysis to a hyper-rectangle b. Note that the neighborhood construction for hyper-rectangles can be done via the lifting operator Λ of Definition 20. More concretely, the neighborhood of b is $N(b) = \Lambda(b, \boldsymbol{\upsilon}^-, \boldsymbol{\upsilon}^+)$ where $\boldsymbol{\upsilon}^-$, $\boldsymbol{\upsilon}^+ \in \mathbb{R}^n$ and for every $i \in \{1, \ldots, n\}$ υ_i^- and υ_i^+ are the lifting amounts for the faces e_i^- and e_i^+, respectively (see Figure 5.8).

Denote l_i the length of the hyper-rectangle b in direction i. The diameter of e_i^+ is thus written as

$$\varrho(e_i^+) = \sqrt{\sum_{k \neq i}(l_k + \upsilon_k^- + \upsilon_k^+)^2 + {\upsilon_i^+}^2}.$$

Then, due to (5.10) we choose $\boldsymbol{\upsilon}^-$ and $\boldsymbol{\upsilon}^+$ satisfying the following inequalities, for $i = 1, \ldots, n$

$$\begin{cases} \varrho(e_i^-)\upsilon_i^- \leq \epsilon M_i^-/L \\ \varrho(e_i^+)\upsilon_i^+ \leq \epsilon M_i^+/L \end{cases} \tag{5.11}$$

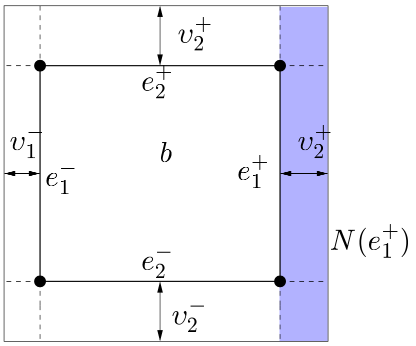


Figure 5.8: Neighborhood construction for a rectangle: the neighborhood of the face e_1^+ is the shaded rectangle.

where M_i^- and M_i^+ are the maxima of $||f(\mathbf{x})||$ over the faces e_i^- and e_i^+, respectively.

We next estimate the time step r. For each face e, let r_e be the value of the term in the right-hand side of the inequality (5.9), i.e.

$$r_e = \frac{1}{L}\ln(\frac{\upsilon_e L}{M_e} + 1).$$

All the trajectories from e are then guaranteed to stay in the neighborhood $N(e)$ within the time interval $[0, r_e]$, and we choose the time step r for the hyper-rectangle b as the minimum of these values:

$$r = \min\{\frac{1}{L}\ln(\frac{\upsilon_i^- L}{M_i^-} + 1), \frac{1}{L}\ln(\frac{\upsilon_i^+ L}{M_i^+} + 1) \mid i \in \{1, \dots, n\}\}. \tag{5.12}$$

The Lipschitz constant of f_e, which is needed for the above calculations, can be estimated by solving the following optimization problem:

$$L = \max\{\ ||Df_e(\mathbf{x})|| \mid \mathbf{x} \in \mathcal{X}\ \}$$

where $Df_e(\mathbf{x})$ is the $n \times n$ matrix of partial derivatives whose element ij is defined as $\partial(f_e)_i/\partial x_j$.

We are now ready to indicate the steps of our error control procedure to be applied to each hyper-rectangle b prior to the lifting operation.

Error control procedure

1. For each direction $i \in \{1, \dots, n\}$, compute M_i^- and M_i^+ as the maxima of $||f(\mathbf{x})||$ over the face e_i^- and e_i^+, respectively.

2. Choose $\boldsymbol{v}^-$ and $\boldsymbol{v}^+$ according to (5.11) and use them to construct the neighborhoods of the faces.

3. Determine the time step r using (5.12).

We remark that, for a given error tolerance, when the diameter of a face is large, formula (5.10) shows that its neighborhood should be small, and due to (5.9) the time step is accordingly small. In this case, partitioning the hyper-rectangle a priori into smaller ones allows larger time steps and can thus help to achieve better performance in terms of computation time.

5.3.2 Error Accumulation

We have presented a procedure for controlling the *local error*, that is, the error incurred in each iteration. However, the goal of ensuring the global error under the desired tolerance is still problematic because of *the accumulation of over-approximation error*. As a result, there are cases where, in the long run, the over-approximation error becomes too large for the result to be useful.

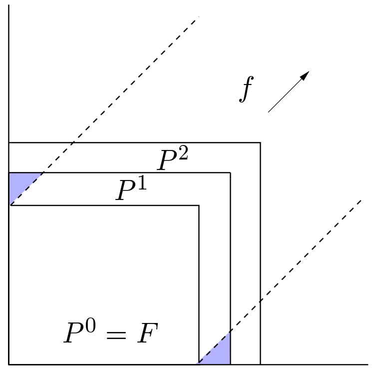


Figure 5.9: An example where the over-approximation error accumulation leads to poor accuracy.

Consider a two-dimensional example shown in Figure 5.9 where the vector field f is constant with non-zero components in both dimensions and the initial set is the rectangle F. The exact reachable set lies between the two dotted diagonal lines. After the first iteration we obtain the rectangle P^1. The shaded regions in the figure represent the over-approximation error in P^1. In the next iteration we compute P^2 from P^1, and one can see that the over-approximation error in P^1 propagates to P^2. It is clear that the algorithm ends up with the whole upper-right quadrant. For such systems, the use of local error control does not help to avoid the growth of the global error. To remedy this, we propose in the following section a modification to the face lifting algorithm.

5.4 Mixed Face Lifting

The reason for the accumulation of over-approximation error is that for every lifted face we look only at the normal direction (which points outward) and ignore the other directions (which may point inward as in the above example). By exploiting some of the inward information we can significantly reduce the accumulation of errors and use a computation scheme similar in spirit to the one used in the previous chapter for linear systems. The idea is to keep separately the sets P^k of states reachable so far and the sets Q^k which represent the 'frontiers' of the set of reachable states at time points kr. This can be formalized as follows.

Let $P^0 = F$ and $Q^0 = F$. The reachable set from F can be computed using the following iterative algorithm:

$$\begin{aligned} Q^{k+1} &= \delta_r(Q^k) \\ P^{k+1} &= P^k \cup \delta_{[0,r]}(Q^k) \end{aligned} \tag{5.13}$$

Note that, due to the semi-group property, $\delta_{[0,(k+1)r]}(F) = \delta_{[0,kr]}(F) \cup \delta_{[0,r]}(\delta_{kr}(F))$, which guarantees that the above algorithm is correct. Next, we show how to over-approximate $\delta_r(F)$.

In order to take inward evolution into account, we need a new neighborhood construction for the faces of F. Recall that $N_{\boldsymbol{\upsilon}}(F)$ is the neighborhood of F resulting from lifting outward the faces e of F by the amounts υ_e.

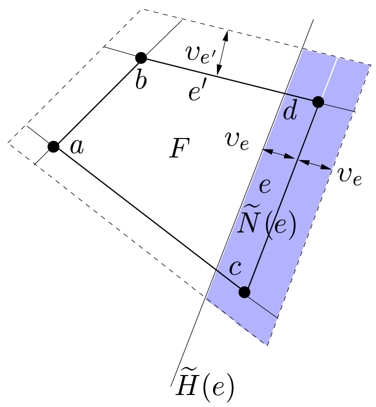


Figure 5.10: Neighborhood construction for the polygon F with vertices $\{a, b, c, d\}$: $N_{\boldsymbol{\upsilon}}(F)$ is the dotted polygon, and $\widetilde{N}(e)$ is the shaded one.

Consider a face e whose supporting hyper-plane is $P(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x}\rangle = c_e\}$ where $\mathbf{n}(e)$ is the *unit outward normal* to e. We define the neighborhood $\widetilde{N}(e)$ of e as

$$\widetilde{N}(e) = N_{\boldsymbol{\upsilon}}(F) \cap \widetilde{H}(e)$$

where $\widetilde{H}(e)$ is the half-space defined as $\widetilde{H}(e) = \{\mathbf{x} \mid \langle \mathbf{n}(e), \mathbf{x} \rangle \geq c_e - v_e\}$. Thus, the new neighborhood includes points on both sides of the hyper-plane $P(e)$ (see Figure 5.10 for an illustration).

We denote by $\tilde{f}_e$ the maximum of f_e over the neighborhood $\widetilde{N}(e)$, that is, $\tilde{f}(e) = \max\{f_e(\mathbf{x}) \mid \mathbf{x} \in \widetilde{N}(e)\}$.

We observe that the successor of every point $\mathbf{x} \in e$ at time point r satisfies

$$\langle \mathbf{n}(e), \delta_r(\mathbf{x}) \rangle \leq c_e + r\tilde{f}_e. \tag{5.14}$$

The above formula provides us with useful information about the boundary of the reachable set $\delta_r(F)$. We suggest a variation of the basic face lifting scheme: we lift the half-space of every face e by the amount $r\tilde{f}_e$, and the direction of lifting (inward or outward) is determined by the sign of $r\tilde{f}_e$. The intersection of the lifted faces gives a new polyhedron F', which is guaranteed to contain $\delta_r(F)$.

We note that the polyhedron F' contains many points reachable before time r and is, therefore, a rough over-approximation of $\delta_r(F)$; however, it contains less over-approximation error than the polyhedron obtained by pushing the faces uniquely outward.

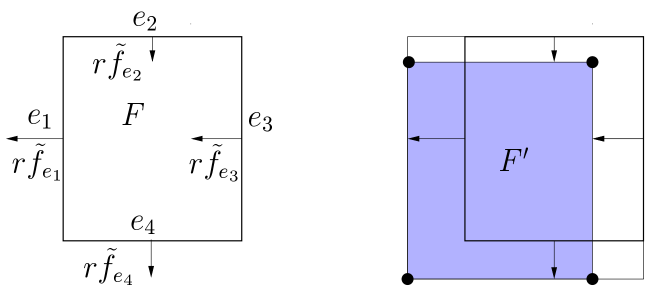


Figure 5.11: Combination of lifting inward and outward to derive an over-approximation of $\delta_r(F)$.

As an example, the over-approximation of δ_r for a rectangle F is depicted in Figure 5.11. The arrows in the left figure show the directions and amounts of lifting. The faces e_1 and e_4 whose $\tilde{f}_e$ are positive are lifted outward, and the other faces whose $\tilde{f}_e$ are negative are lifted inward. The polyhedron F' is the shaded rectangle in the right figure, and the polyhedron resulting from lifting outward alone is the enveloping rectangle.

Embedding the above scheme for over-approximating δ_r in (5.13), we obtain Algorithm 7 shown on the next page. This algorithm, which we call *mixed face lifting*, can reduce the accumulation of error by considering both *outward* and *inward* evolutions. Basically, in each iteration we perform the additional computation of Q^k, in which the *lifting amounts do not depend on the sign of* $\tilde{f}_e$. The boundary of Q^k is a better approximation of $\partial(\delta_r(P^k))$ than P^k, and Q^k is used as the basis for the computation of $\delta_{[0,r]}$ in the next iteration. It should be noted that P^k is computed from Q^k using the original face lifting algorithm; hence, the result obtained is guaranteed to be an over-approximation of $\delta(F)$.

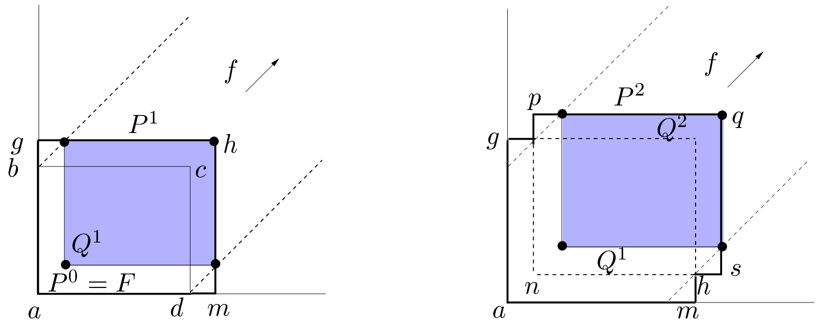


Figure 5.12: Illustration of the mixed face lifting algorithm.

Figure 5.12 illustrates the behavior of the mixed face lifting algorithm on the example of Figure 5.9. The initial set F is the rectangle with the vertices $\{a, b, c, d\}$. After the first iteration, the set Q^1 is the shaded rectangle shown in the left figure, and the set P^1 of states reachable within the first iteration is the rectangle with vertices $\{a, g, h, m\}$. Next, starting from Q^1 (the dotted rectangle in the right figure) the set of new reachable states in the second iteration is the rectangle with vertices $\{n, p, q, s\}$, and the set Q^2 is the shaded rectangle. The set P^2 of reachable states accumulated so far is the enveloping orthogonal polygon with vertices $\{a, g, p, q, s, h, m\}$. Here, one can see that the modified algorithm produces a much more accurate approximation. Note that in this example, due to the constant derivative, the sets Q^k are the exact successors of F at time points kr, i.e. $Q^k = \delta_{kr}(F)$. In more general cases, these sets contain over-approximation errors, and the modified algorithm can only *reduce* the accumulation of over-approximation error. Note also that when $\tilde{f}_e$ is positive everywhere, the mixed face lifting algorithm gives exactly the same result as the original one.

5.5 Examples

We have implemented the above algorithms using both uniform and non-uniform grids. We wish to mention that at the time we were exploring the technique, the work on the canonical representation for orthogonal polyhedra was still in progress. All the results described below are for demonstration purposes and obtained by a simple implementation. The data structure for uniform grids is an n-dimensional matrix. For non-uniform grids, in addition to the matrix, we used a linked list to represent the variable grid coordinates. Although advanced data structures such as dictionaries or dynamic arrays allow compact storage for sparse matrices, this implementation is clearly very costly for the required geometric operations. We intend to implement face lifting using the canonical representation of orthogonal polyhedra in a near future and believe that this implementation will improve significantly the efficiency of the technique both in terms of space and time usage and thus increase the applicability to high dimensional systems.

Algorithm 7 (Mixed Face Lifting Algorithm)

$P^0 := F$; $Q^0 = F$;
repeat $k = 0, 1, 2, \ldots$

/* *Computing* $\delta_{[0,r]}$ *of* Q^k *by lifting outward alone* */
  for all $b_j \in decomp(Q^k)$ {
      $\mathbf{d}^- := \mathbf{0}$; $\mathbf{d}^+ := \mathbf{0}$;
      for all $i \in \{1, \ldots, n\}$ {
        if $(e_i^- \subseteq \partial P^k)$ {
            $\hat{f} := \max\{-f_i(\mathbf{x}) \mid \mathbf{x} \in N(e_i^-)\}$;
          if $(\hat{f} > 0)$ $d_i^- := \beta\lfloor r\hat{f}/\beta\rfloor$;
        }
        if $(e_i^+ \subseteq \partial P^k)$ {
            $\hat{f} := \max\{f_i(\mathbf{x}) \mid \mathbf{x} \in N(e_i^+)\}$;
          if $(\hat{f} > 0)$ $d_i^+ := \beta(\lfloor r\hat{f}/\beta\rfloor + 1)$;
        }
      }
      $b'_j := \Lambda(b_j, \mathbf{d}^-, \mathbf{d}^+)$;
      $P^{k+1} := P^k \cup b'_j$;
  }

/* *Computing* Q^{k+1} *from* Q^k *by lifting outward and inward* */
  $Q^{k+1} := \emptyset$;
  forall $b_j \in decomp(Q^k)$ {
      $\mathbf{d}^- := \mathbf{0}$; $\mathbf{d}^+ := \mathbf{0}$;
      for all $i \in \{1, \ldots, n\}$ {
        if $(e_i^- \subseteq \partial Q^k)$ {
            $\tilde{f} := \max\{-f_i(\mathbf{x}) \mid \mathbf{x} \in \widetilde{N}(e_i^-)\}$;
            $d_i^- := \beta\lfloor r\tilde{f}/\beta\rfloor$;
        }
        if $(e_i^+ \subseteq \partial Q^k)$ {
            $\tilde{f} := \max\{f_i(\mathbf{x}) \mid \mathbf{x} \in \widetilde{N}(e_i^+)\}$;
            $d_i^+ := \beta(\lfloor r\tilde{f}/\beta\rfloor + 1)$;
        }
      }
      $b'_j := \Lambda(b_j, \mathbf{d}^-, \mathbf{d}^+)$;
      $Q^{k+1} := Q^{k+1} \cup b'_j$;
  }
until $P^{k+1} = P^k$
return P^{k+1}

5.5.1 Linear Systems

Linear Systems in $\mathbb{R}^2$

We demonstrate the behavior of the algorithm on various classes of linear systems of the form $\dot{\mathbf{x}} = A\mathbf{x}$ (see [61] for the classification). The examples treated are described in Figure 5.13 and the results obtained appear in Figure 5.14.

Type	A	Initial set
Center	$\begin{pmatrix} 0.0 & -6.0 \\ 3.0 & 0.0 \end{pmatrix}$	$[-0.25, 0.25] \times [-0.25, 0.25]$
Node	$\begin{pmatrix} -5.0 & 0.0 \\ 0.0 & -2.0 \end{pmatrix}$	$[0.2, 0.5] \times [0.2, 0.4]$
Saddle	$\begin{pmatrix} -5.0 & 0.0 \\ 0.0 & 4.0 \end{pmatrix}$	$[0.0, 0.4] \times [-0.0, 0.4]$
Sink	$\begin{pmatrix} -2.0 & -3.0 \\ 3.0 & -2.0 \end{pmatrix}$	$[-0.1, 0.3] \times [0.1, 0.3]$

Figure 5.13: Examples of linear systems.

Sometimes, the use of a fixed grid generates an over-approximation which covers all the space. This is evident the case of the first example (center) where every edge has a non-zero outward component in some dimension. To obtain the desired result, we have changed in these cases the rounding rule, that is, we push a face to the nearest grid unit and not necessarily outward. Consequently, the resulting polyhedron is not guaranteed to be an over-approximation of the reachable set. Using a variable grid is another way to solve this problem. Note that the optimization of f_e is much cheaper computationally in the linear case.

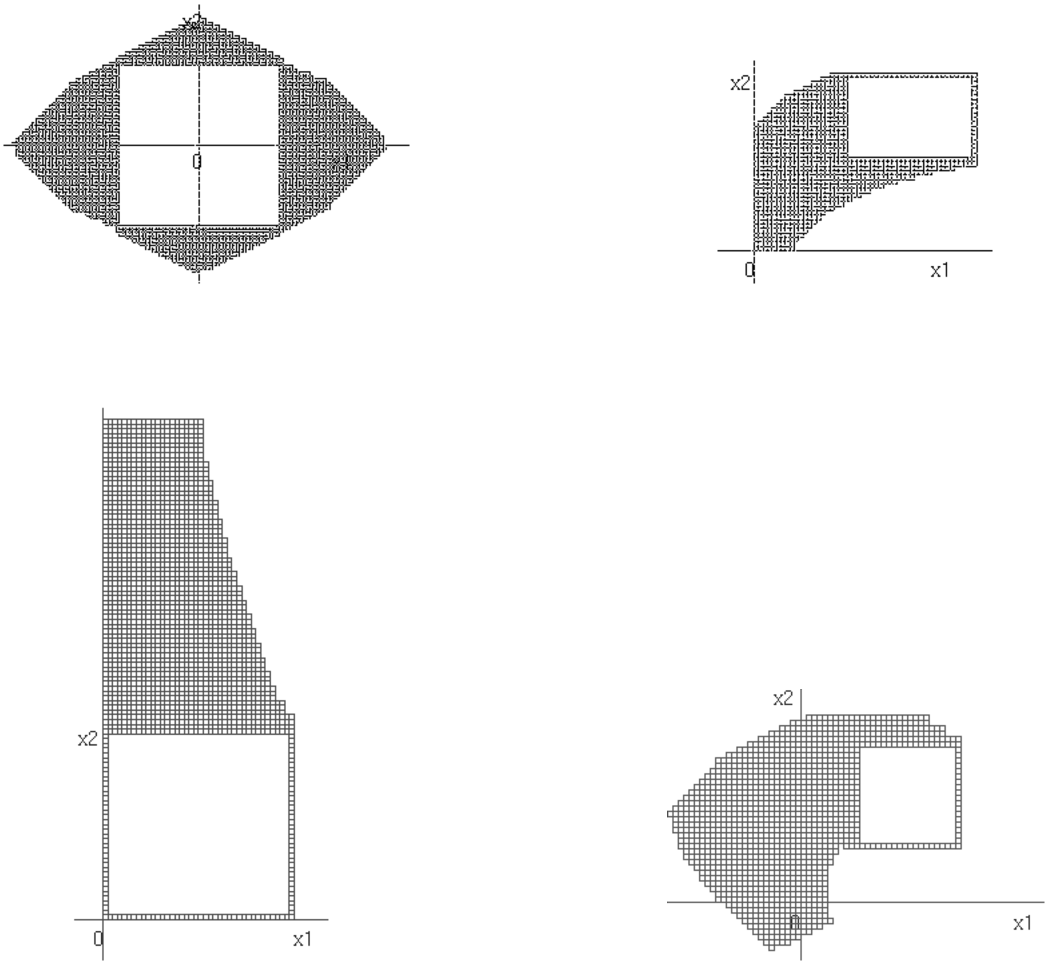


Figure 5.14: Reachable sets of linear systems of type: 1) Center, 2) Node, 3) Saddle and 4) Sink. The white rectangles are the initial sets.

Linear Systems in $\mathbb{R}^3$

In Figure 5.15 one can see the reachable set of a 3-dimensional system with

$$A = \begin{pmatrix} -2 & 0 & 0 \\ 1 & -2 & 0 \\ 0 & 1 & -2 \end{pmatrix}$$

starting from the initial region $[-0.025, 0.025] \times [-0.1, 0.1] \times [0.05, 0.07]$.

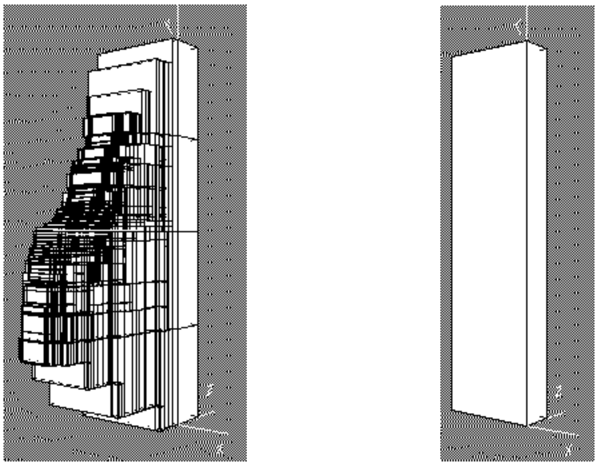


Figure 5.15: Reachable states (left) starting from an initial region (right) for a 3-dimensional linear system.

5.5.2 Mixing Tank

This example, taken from [108], is a typical non-linear equation encountered in chemical engineering. The mixing tank, depicted in Figure 5.16, consists of a free outlet stream v_3 and of two controlled inlets with different rates v_1, v_2 and different concentrations c_1, c_2. The variables x_1 and x_2 represent the height and the concentration of liquid, respectively. The dynamics of these variables are described by the following differential equations:

$$\begin{aligned} \dot{x}_1 &= \frac{1}{k_1}(v_1 + v_2 - k_2\sqrt{x_1}) \\ \dot{x}_2 &= \frac{1}{k_1 x_1}(v_1(c_1 - x_2) + v_2(c_2 - x_2)) \end{aligned}$$

where k_1, k_2 are geometrical parameters.

With the following choice of parameters:

$$k_1 = 1\ m^2,\ k_2 = 0.02\ m^{2.5}/s,\ c_1 = 1\ mole/l,\ c_2 = 2\ mole/l,$$

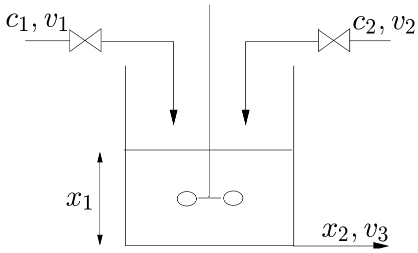


Figure 5.16: A Mixing Tank.

(1.322, 1.652) is an equilibrium state of the system. Figure 5.17 depicts the set of states reachable from an initial set $[1.12, 1.17] \times [1.56, 1.68]$ (white rectangle), and one can see the convergence to the equilibrium.

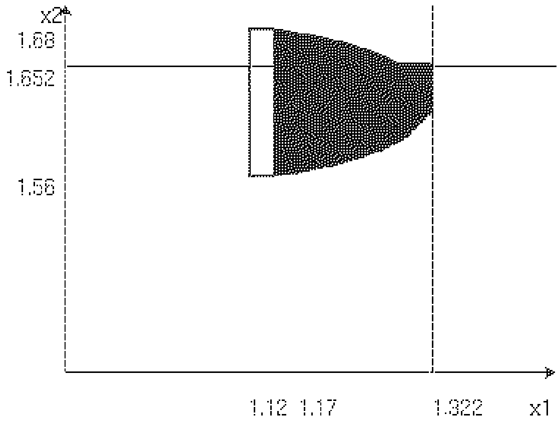


Figure 5.17: Reachable set of the mixing tank system.

5.5.3 Airplane Safety

The next example is taken from [80]. The state variables x_1, x_2 represent, respectively, the velocity and the flight path angle of an aircraft. Their evolution is governed by

$$\begin{aligned}
\dot{x}_1 &= -\frac{a_D x_1^2}{m} - g \sin x_2 + \frac{u_1}{m} \\
\dot{x}_2 &= \frac{a_L x_1 (1 - c x_2)}{m} - \frac{g \cos x_2}{x_1} + \frac{a_L c x_1}{m} u_2
\end{aligned}$$

where u_1 (the thrust), u_2 (the pitch angle) are bounded control inputs such that $u_1 \in [T_{min}, T_{max}]$ and $u_2 \in [\Theta_{min}, \Theta_{max}]$; m is the mass of the aircraft; g is the gravitational acceleration; c, a_L, and a_D are the system parameters.

The problem is to determine the safe subset of the state space, that is, the states from which the system does not leave the envelope P defined as the rectangle $[V_{min}, V_{max}] \times [\gamma_{min}, \gamma_{max}]$. For doing this, we take the complement of P, that is, $\mathcal{X} \setminus P$, as the set of initial states and compute the set R reachable by the reverse system $\dot{\mathbf{x}} = -f(\mathbf{x})$. Therefore, the set R contains all the states which can leave P, and the safe subset will be $P \setminus R$. Note that our technique produces an over-approximation of R and, as a result, the computed safe subset is an under-approximation of the real set. The results, depicted in Figure 5.19, correspond to the following parameters:

$$m = 85000kg,\ c = 6,\ a_L = 30;\ a_D = 2,\ T_{min} = 40000N,\ T_{max} = 80000N,\ \Theta_{min} = -22.5^o,$$
$$\Theta_{max} = 22.5^o,\ V_{min} = 180m/s,\ V_{max} = 240m/s,\ \gamma_{min} = -22.5^o,\ \gamma_{max} = 22.5^o,$$

and for the specific choices of the controls $u_1 = T_{max}, u_2 = \Theta_{min}$. (for the left boundary of the rectangle P) and $u_1 = T_{min}, u_2 = \Theta_{max}$ (for the right boundary of P). These results are consistent with those in [80] obtained using analytical methods.

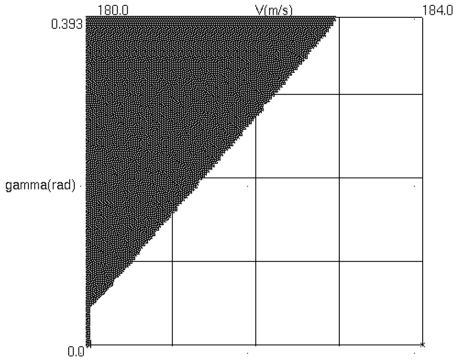


Figure 5.18: Airplane Safety: $u_1 = T_{max}, u_2 = \Theta_{min}$.

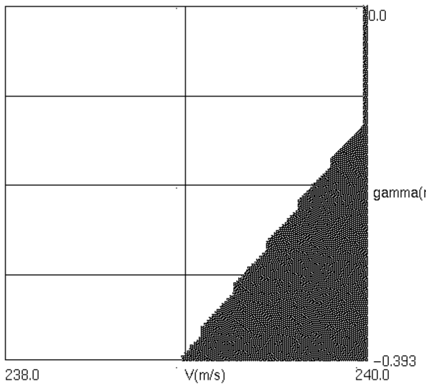


Figure 5.19: Airplane Safety: $u_1 = T_{min}, u_2 = \Theta_{max}$.

5.6 Summary and Related Work

We have presented an algorithm for over-approximating reachable sets of non-linear continuous systems of arbitrary dimensions. The novelty of our method is the way of representing high dimensional sets, which guarantees termination of the algorithm and, moreover, allows an easy extension to hybrid systems, as we will show in the next chapter.

There have been various works on the computation of reachable sets for non-linear continuous systems. However, few of these are concerned with the important questions of data structures for high dimensional sets.

To our knowledge, the first appearance of the ideas underlying face lifting was the work of Kurshan and McMillan in [70]. Their approach can be classified as indirect since they were trying to construct a finite-state discrete abstraction of an electrical circuit (defined using the differential equations for voltage and current of transistor). They partitioned the continuous state space into hyper-cubes using a fixed grid and computed the reachability relations between these cubes by optimizing normal derivatives, as in face lifting. No general purpose tool has been built based on these ideas.

Our work was triggered by the paper of Greenstreet [47], motivated by similar problems as in [70], where the idea of face lifting in two dimensions for arbitrary polyhedra was first proposed. Later works of Greenstreet and Mitchell [48, 49] concentrated on an alternative method to represent high dimensional polyhedra by their two-dimensional projections (a polyhedron is thus the largest set satisfying the constraints of its projections). By doing this, all the operations are performed on the projections of the polyhedron. The advantage of this representation is that successors of projections can be more efficiently and accurately computed. However, its obvious drawback is that polygonal projections, even non-convex, cannot capture the real geometric form of the full-dimensional reachable sets, and this results in further over-approximation error which is hard to control.

The reachability method of Chutinan and Krogh [31], which we have outlined in the previous chapter, was first developed for non-linear systems and then specialized for linear systems. Their method differs from ours in their trajectory-based approximations and in our use of orthogonal polyhedra to store reachable states.

The idea of approximating reachable sets over a grid was used by Puri, Borkar, and Varaiya in [98] for differential inclusions $\dot{\mathbf{x}} \in f(\mathbf{x})$. Their method consists in partitioning a priori the state space into a finite number of boxes such that the variation of f within each box is bounded by a given constant ϵ and then associating with each box a constant rectangular differential inclusion of the form $\mathbf{c}_l \leq \dot{\mathbf{x}} \leq \mathbf{c}_u$. Then, the computation of the reachable set is performed on the resulting *approximate* model (for which the reachability problem is decidable). This approach, like ours, can guarantee error bounds only for a *finite* time horizon.

Finally, the reachability problem can be formulated using partial differential equations (see, e.g., [29, 111]). The reachable set $\delta_{[0,t]}(F)$ can be described as $\delta_{[0,t]}(F) = \{\mathbf{x} \mid l(\mathbf{x}, t) \leq 0\}$

where $l : \mathcal{X} \times \mathcal{T} \to \mathbb{R}$ is the solution to the following equation:

$$\frac{\partial l}{\partial t} = -\mathrm{grad}(l) \cdot f$$

with the initial conditions:

$$\begin{cases} l(\mathbf{x}, 0) = 0 & \text{if } \mathbf{x} \text{ is on the boundary of } F, \\ l(\mathbf{x}, 0) < 0 & \text{if } \mathbf{x} \text{ is in the interior of } F. \end{cases}$$

Various methods for tracking the evolution of l exist (see, e.g., [105]). However, numerical solutions are often complicated, and so far we have found no special computational advantage of this formulation over the direct ODE formulation.

Chapter 6

Verification of Hybrid Systems

In this chapter we show how the approximate reachability techniques for continuous systems, presented in the previous chapters, are adapted to hybrid systems. Reachability analysis is a basic component of many verification and synthesis procedures. We are interested in the verification of *invariance* properties, which is equivalent to the reachability problem.

An outline of this chapter is as follows. We begin by stating formally the verification problem for hybrid automata and then give an abstract verification algorithm, based on which we develop an effective approximate algorithm. We also present a concrete implementation of the approximate algorithm and discuss important computational issues. Finally, we illustrate the application of our approach with some examples. Some of the results of this chapter have been presented in [10].

6.1 Problem Statement

Consider a hybrid automaton $A = (\mathcal{X}, Q, f, G, H, R)$ of Definition 7. Let $\mathcal{F} = \{(q, F_q) \mid q \in Q \ \wedge \ F_q \subseteq \mathcal{X}\}$ be the set of initial states. The problem of safety verification is stated as follows.

Problem 3 (Safety verification problem)
Given a set $\mathcal{S} = \{(q, S_q) \mid q \in Q \ \wedge \ S_q \subseteq \mathcal{X}\}$ we want to verify that the hybrid automaton $\mathcal{A}$ satisfies $\Box\mathcal{S}$, i.e. all trajectories of $\mathcal{A}$ remain inside $\mathcal{S}$.

We say that the hybrid automaton $\mathcal{A}$ is *safe* if it satisfies the specification $\Box\mathcal{S}$ and *unsafe* otherwise.

In the hybrid automaton model to be considered, the sets H_q and $G_{qq'}$ are *convex polyhedra* in $\mathbb{R}^n$ for all $q, q' \in Q$. This assumption is not restrictive for two reasons. First, in many practical systems, staying conditions and transitions guards are specified as conjunctions of linear inequalities, which define convex polyhedra. Second, the algorithms presented here can be extended to hybrid automata whose staying and guard sets are non-convex polyhedra. Similarly, we require the sets S_q and F_q to be polyhedral (convex or orthogonal). Non-polyhedral sets should be replaced a priori by their polyhedral approximations.

We consider affine, set-valued reset maps, more precisely, for all $q, q' \in Q$, $R_{qq'}(\mathbf{x}) = D_{qq'}\mathbf{x} + J_{qq'}$ where $D_{qq'}$ is an $n \times n$ matrix and $J_{qq'}$ is a convex polyhedron in $\mathbb{R}^n$. Of particular interest are the following special cases of $R_{qq'}$:

1. Deterministic resets: $J_{qq'}$ is a singleton $\{\boldsymbol{j}_{qq'}\}$. The reset $R_{qq'}$ maps every point $\mathbf{x} \in \mathcal{X}$ to a unique point $D_{qq'}\mathbf{x} + \boldsymbol{j}_{qq'}$.
2. Identity resets: $D_{qq'}$ is the identity matrix and $J_{qq'} = \{\mathbf{0}\}$, which means that the continuous variables remain unchanged after the transition from q to q'.
3. Memoryless resets: $D_{qq'}$ is the zero matrix, and $R_{qq'}$ is thus an arbitrary set-valued map. If all $R_{qq'}$ satisfy this condition, we say that the system is memoryless.

6.2 Verification Algorithm

Proving that the hybrid automaton $\mathcal{A}$ satisfies $\Box\mathcal{S}$ is equivalent to proving that $\mathcal{A}$ never reaches the set $\mathcal{B} = \{(q, B_q) \mid q \in Q \wedge B_q = \mathcal{X} \setminus S_q\}$, the complement of $\mathcal{S}$, which represents the set of unsafe ("bad") states. Problem 3 can be solved by using either forward or backward reachability analysis. We focus first on forward analysis and present an adaptation for backward analysis in Section 6.5.

In essence, we compute the set reachable by the system from $\mathcal{F}$ and then check emptiness of its intersection with the bad set $\mathcal{B}$. The abstract algorithm for solving Problem 3 is shown below.

Algorithm 8 (Forward Verification Algorithm)

> $\mathcal{P}^0 := \mathcal{F}$;
> **repeat** $k = 0, 1, 2, \ldots$
>   **if** $(\mathcal{P}^k \cap \mathcal{B} \neq \emptyset)$ **return** *unsafe*
> $\mathcal{P}_c := \delta_c(\mathcal{P}^k)$;
> $\mathcal{P}_d := \delta_d(\mathcal{P}_c)$;
> $\mathcal{P}^{k+1} := \mathcal{P}^k \cup \mathcal{P}_c \cup \mathcal{P}_d$;
> **until** $\mathcal{P}^{k+1} = \mathcal{P}^k$
> **return** *safe*

The reachable set is initialized with the initial set: $\mathcal{P}^0 = \mathcal{F}$. In each iteration the continuous-successor operator δ_c is applied to $\mathcal{P}^k$. This gives the set of states reachable by continuous dynamics, to which the discrete-successor operator δ_d is next applied to obtain the set of states reachable by executing discrete transitions. If the intersection of the computed set $\mathcal{P}^k$

with the bad set $\mathcal{B}$ is not empty, then the algorithm reports that the system is unsafe and terminates. Otherwise, the algorithm continues until no new reachable states are found. If the algorithm terminates and the bad set is not reached, then the system is proved to satisfy the safety specification.

This abstract algorithm is in fact the basic model-checking algorithm implemented in the verification tools such as Kronos [120] for timed automata and HyTech [56] for 'linear' hybrid automata. As discussed earlier, for systems with arbitrary continuous dynamics, exact computation of reachable sets is, in general, not possible. Our approach is to use Algorithm 8 with successive approximations of continuous- and discrete-successors using orthogonal polyhedra. Note that in the context of safety verification, over-approximations are required. In the following we show how to over-approximate the operators δ_c and δ_d. From now on all the sets we manipulate are either *convex* or *orthogonal* polyhedra.

6.2.1 Continuous-Successors

Given a set of states (q, F) where $q \in Q$ and F is a polyhedron in $\mathbb{R}^n$, we want to compute an *orthogonal over-approximation of* $\delta_c(q, F)$, denoted by $\widehat{\delta}_c(q, F)$.

We have presented in the previous chapters two algorithms for over-approximating the successor operator δ of continuous systems. One is specialized for *linear systems* that can admit *uncertain input*. For brevity, from now on we call it the LIN algorithm. The other algorithm, based on the face lifting technique, can be applied to *non-linear systems*, and we call it the FL algorithm. We will approximate the continuous-successors operator of hybrid automata using these algorithms. Note that for the FL algorithm non-orthogonal initial polyhedra are replaced by their orthogonal over-approximations, and for the LIN algorithm non-convex initial polyhedra are first decomposed into convex ones.

The only difference between the operator δ_c of hybrid automata and the operator δ of continuous systems is the presence of staying conditions in the former. During the continuous evolution at a discrete state q, some trajectories may go out of the staying set H_q and from there no further continuous evolution at the current discrete state is possible. Hence, the reachability algorithms for continuous systems should be modified to account for staying conditions.

We consider first the FL algorithm. Essentially, starting from F this algorithm iteratively computes

$$P^{k+1} = P^k \cup \widehat{\delta}_{[0,r_k]}(P^k)$$

where $\widehat{\delta}_{[0,r_k]}$ is an over-approximation of the reachability operator $\delta_{[0,r_k]}$ and every P^k is an orthogonal polyhedron. Recall that for accuracy and efficiency purposes the FL algorithm uses variable time steps.

The treatment of staying conditions consists in removing from the sets P^{k+1} the states that do not satisfy H_q and starting the next iteration from the resulting set. In other words, in the modified algorithm we compute P^{k+1} as follows:

$$P^{k+1} = (P^k \cup \widehat{\delta}_{[0,r_k]}(P^k)) \cap H_q. \tag{6.1}$$

Since H_q is a convex polyhedron, the scheme (6.1) involves intersections of orthogonal and convex polyhedra, which cannot be exactly computed (orthogonal polyhedra are obviously not closed under such an operation). To remedy this, we approximate these intersections using the following operator.

Definition 21 *Given an orthogonal polyhedron G and a convex polyhedron C, $C \sqcap_o G$ is the smallest orthogonal polyhedron defined on the grid of G such that $C \sqcap_o G \supseteq C \cap G$.*

We defer the computation of the $\sqcap_o$ operator to the end of this section.

Now, replacing $\cap$ with $\sqcap_o$ in (6.1) we obtain the following scheme

$$P^{k+1} = (P^k \cup \widehat{\delta}_{[0,r_k]}(P^k)) \sqcap_o H_q, \tag{6.2}$$

which guarantees to produce an orthogonal over-approximation $\widehat{\delta}_c$ of δ_c.

We turn now to the LIN algorithm. The basic iterative scheme of the LIN algorithm is written as

$$\begin{aligned} P^{k+1} &= P^k \cup \widehat{\delta}_{[0,r]}(X^k) \\ X^{k+1} &= \delta_r(X^k) \end{aligned}$$

Recall that besides P^k, which is used to accumulate the reachable states, the LIN algorithm maintains a convex polyhedron X^k which is the exact reachable set at time point kr and used as the basis for the computation of P^{k+1}. Therefore, to adapt the LIN algorithm for computing $\widehat{\delta}_c$, we need interesect not only P^{k+1} but also X^{k+1} with H_q. Again, we make use of the $\sqcap_o$ operator to compute $\widehat{\delta}_c$ as follows:

$$\begin{aligned} P^{k+1} &= (P^k \cup \widehat{\delta}_{[0,r]}(X^k)) \sqcap_o H_q \\ X^{k+1} &= \delta_r(X^k) \cap H_q \end{aligned} \tag{6.3}$$

Since every $\delta_r(X^k)$ is a convex polyhedron, X^{k+1} can be exactly computed.

Before proceeding, we make two important observations concerning the accuracy and efficiency of the above modified algorithms.

- In both algorithms, the $\sqcap_o$ operator introduces further error into the approximation. The FL algorithm computes P^{k+1} based on P^k, it it clear that this error propagates from iteration to iteration. However, the LIN algorithm computes P^{k+1} based on X^k, which is exactly computed. Consequently, *the LIN algorithm, when applied for computing* $\widehat{\delta}_c$*, preserves the property of not propagating over-approximation errors in the continuous phase.*

- Recall that the computation cost of the LIN algorithm depends on the number of vertices of the convex polyhedra X^k. For purely continuous systems the number of vertices remains constant, but for hybrid systems it might change due to the intersection with H_q (see Figure 6.1).

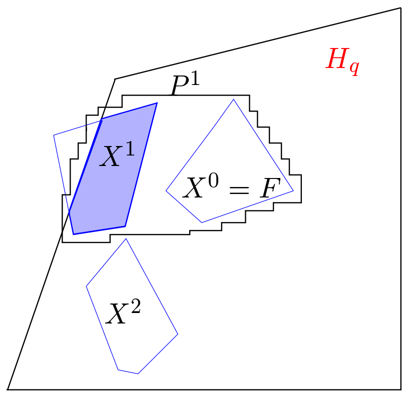


Figure 6.1: Over-approximating $\delta_c(q, F)$ using the LIN algorithm: the intersection of X^1 and H_q results in more vertices in X^2.

We now show how to compute the $\sqcap_o$ operator, which is used by both modified algorithms.

Computing the $\sqcap_o$ operator

The $\sqcap_o$ operator takes as input a convex polyhedron C and an orthogonal polyhedron G and returns the *smallest orthogonal polyhedron* defined on the grid of G that includes the intersection $C \cap G$.

Let $\mathcal{G}_\beta$ be the underlying grid of G. We can represent the orthogonal polyhedron G as the union of, say, m_g elementary hyper-cubes of $\mathcal{G}_\beta$: $G = \{g_i \mid i = 1, \ldots, m_g\}$. It is not hard to see that $(C \sqcap_o G)$ is the union of the elementary hyper-cubes g_i in G whose intersection with C is not empty (see Figure 6.2). We write

$$C \sqcap_o G = \{g_i \mid g_i \cap C \neq \emptyset,\ i = 1, \ldots, m_g\}.$$

The algorithm for computing $\sqcap_o$ is described in Chapter 8 (Implementation).

By construction, the distance between each hyper-cube g_i in $C \sqcap_o G$ and $C \cap G$ is at most the size β of the underlying grid. It then follows by Lemma 1-(h1) that $h(C \sqcap_o G, C \cap G) \leq \beta$, which means that the error incurred in replacing $\cap$ by $\sqcap_o$ is bounded by $\sqrt{n}\beta$.

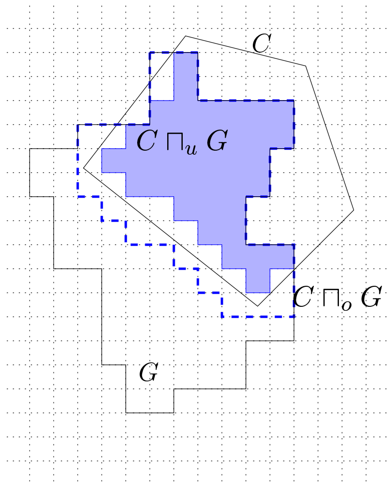


Figure 6.2: Orthogonal approximations of $C \cap G$: $C \sqcap_o G$ is the dotted orthogonal polygon and $C \sqcap_u G$ is the shaded one.

We note that the above results can be naturally extended to *under-approximate* the continuous-successor operator. For doing this, we define $C \sqcap_u G$ as the *largest orthogonal polyhedron* such that $C \sqcap_u G \subseteq C \cap G$. It is easy to see that

$$C \sqcap_u G = \{g_i \mid g_i \subseteq C,\ i = 1, \dots, m_g\}$$

and the distance $h(C \sqcap_u G, C \cap G)$ is also bounded by $\sqrt{n}\beta$. Hence, an orthogonal under-approximation of $\delta_c(q, F_q)$ can be obtained by replacing $\sqcap_o$ with $\sqcap_u$ in the schemes (6.2) and (6.3).

6.2.2 Discrete-Successors

For a given set of states (q, F) where $q \in Q$ and F is an orthogonal polyhedron in $\mathbb{R}^n$, we want to compute an *orthogonal over-approximation* of $\delta_d(q, F)$, denoted by $\widehat{\delta}_d(q, F)$.

The reason we consider the problem where F is orthogonal is that continuous-successors, as shown above, are represented by orthogonal polyhedra. In addition, the algorithm presented in the following is also applicable for convex polyhedra.

The set of discrete-successors of (q, F) is written as

$$\delta_d(q, F) = \bigcup_{q' \in Q} \delta_{qq'}(q, F)$$

where $\delta_{qq'}(q, F_q)$ is the set of discrete-successors of (q, F) with respect to the transition from q to q'. Followed immediately from the definition of $\delta_{qq'}$,

$$\delta_{qq'}(q, F) = (q', R_{qq'}(F \cap G_{qq'}) \cap H_{q'}).$$

Recall that by convention the guard $G_{qq'}$ is empty if there is no transition from q to q'. We consider first the problem of finding an orthogonal polyhedron P such that

$$P \supseteq R_{qq'}(F \cap G_{qq'}) \cap H_{q'}.$$

As mentioned earlier, the resets in the hybrid automata considered are linear transformations, that is, $R_{qq'}(\mathbf{x}) = D_{qq'}\mathbf{x} + J_{qq'}$ where $D_{qq'}$ is an $n \times n$ matrix and $J_{qq'}$ is a convex polyhedron in $\mathbb{R}^n$.

Since the polyhedron F is orthogonal and $G_{qq'}$ is convex, we replace again $F \cap G_{qq'}$ by $F \sqcap_o G_{qq'}$. The orthogonal polyhedron $F \sqcap_o G_{qq'}$ can be decomposed into, say, m_b non-overlapping hyper-rectangles b_i:

$$F \sqcap_o G_{qq'} = \bigcup_{i=1}^{m_b} b_i.$$

We can represent the convex polyhedron $J_{qq'}$ by the convex hull of its vertices $\{\boldsymbol{j}_1, \dots, \boldsymbol{j}_{m_j}\}$: $J_{qq'} = conv\{\boldsymbol{j}_1, \dots, \boldsymbol{j}_{m_j}\}$. Then, the image of a point $\mathbf{x} \in \mathbb{R}^n$ by $R_{qq'}$ can be written as

$$R_{qq'}(\mathbf{x}) = conv\{D_{qq'}\mathbf{x} + \boldsymbol{j}_1, \dots, D_{qq'}\mathbf{x} + \boldsymbol{j}_{m_j}\}. \tag{6.4}$$

Hyper-rectangles are convex, and their convexity is preserved by linear transformation; therefore, the image of the hyper-rectangle b_i by $R_{qq'}$ can be obtained by applying $R_{qq'}$ to its 2^n vertices $\{\mathbf{v}_1, \dots, \mathbf{v}_{2^n}\}$ as shown in (6.4) and then taking the convex hull of the resulting sets. In other words, $R_{qq'}(b_i) = conv(R_{qq'}(\mathbf{v}_1) \cup \dots \cup R_{qq'}(\mathbf{v}_{2^n}))$. Hence, to compute P, we need just transform every convex polyhedron $R_{qq'}(b_i) \cap H_{q'}$ into orthogonal using the $grid_o$ operator:

$$P = \bigcup_{i=1}^{m_b} grid_o(R_{qq'}(b_i) \cap H_{q'}).$$

We derive from the above formula the following algorithm for computing an orthogonal over-approximation $\widehat{\delta}_{qq'}(q, F_q)$ of $\delta_{qq'}(q, F_q)$. Recall that *decomp* denotes the function that decomposes an orthogonal polyhedron into a list of non-overlapping hyper-rectangles.

Algorithm 9 (Computing $\widehat{\delta}_{qq'}(q, F)$)

```
P := ∅;
F_g := F ⊓_o G_qq';
for all b_i ∈ decomp(F_g) {
   h_i := R_qq'(b_i) ∩ H_q;
   G := grid_o(h_i);
   P := P ∪ G;
}
δ̂_qq'(q, F_q) := (q', P);
```

Note that the union of all the convex polyhedra h_i is also an over-approximation of the set $R_{qq'}(F \cap G_{qq'}) \cap H_{q'}$. This fact will be exploited in Section 6.3 to increase the efficiency of the verification algorithm in certain cases.

Now, $\widehat{\delta}_d(q, F)$ can be readily computed as follows:

$$\widehat{\delta}_d(q, F) = \bigcup_{q' \in Q} \widehat{\delta}_{qq'}(q, F).$$

We proceed with the estimation of the approximation error. The error is first introduced by the $\sqcap_o$ operator and bounded at this stage by $\sqrt{n}\beta$ where n is the dimension of the hybrid automaton. After the linear transformation $R_{qq'}$, the bound on this error becomes $\sqrt{n}\beta||D_{qq'}||$, and hence after applying $grid_o$ the total error in $\widehat{\delta}_{qq'}(q, F)$ is bounded by $\sqrt{n}\beta(||D_{qq'}|| + 1)$.

As before, to *under-approximate* $\delta_{qq'}(q, F)$, one can use Algorithm 9 with the operators $\sqcap_o$ and $grid_o$ replaced by $\sqcap_u$ and $grid_u$, respectively.

It remains now to replace the exact operators δ_c and δ_d in Algorithm 8 by $\widehat{\delta}_c$ and $\widehat{\delta}_d$ to obtain an approximate verification algorithm for hybrid automata.

Algorithm 10 (Approximate Forward Verification Algorithm)

> $\mathcal{P}^0 := \mathcal{F}$;
> **repeat** $k = 0, 1, 2, \ldots$
>   **if** $(\mathcal{P}^k \cap \mathcal{B} \neq \emptyset)$ **return** *bad-set-reached*
> $\mathcal{P}_c := \widehat{\delta}_c(\mathcal{P}^k)$;
> $\mathcal{P}_d := \widehat{\delta}_d(\mathcal{P}_c)$;
> $\mathcal{P}^{k+1} := \mathcal{P}^k \cup \mathcal{P}_c \cup \mathcal{P}_d$;
> **until** $\mathcal{P}^{k+1} = \mathcal{P}^k$
> **return** *safe*

In Algorithm 10, checking emptiness of $\mathcal{P}^k \cap \mathcal{B}$, when the bad set $\mathcal{B}$ is not orthogonal, is done by decomposing $\mathcal{P}^k$ into hyper-rectangles rather than by using $\sqcap_o$ in order to avoid introducing additional error. Since the algorithm over-approximates the reachable set, it might declare the system unsafe even though the exact reachable set does not intersect $\mathcal{B}$. On the other hand, Algorithm 10 is *sound*: if it terminates by declaring the system to be safe, then the system is indeed safe.

6.2.3 Implementation

Algorithm 11, a more concrete version of Algorithm 10, appears, in pseudo-code form, in Figure 6.3. Before detailing the implementation of this algorithm, let us recall the computational procedures on which it is based:

- Over-approximation of reachable sets of continuous dynamics by orthogonal polyhedra;
- Over-approximation of intersections of orthogonal polyhedra and convex polyhedra by orthogonal polyhedra;
- Exact Boolean operations on orthogonal polyhedra;
- Tests of equivalence between polyhedra.

Algorithm 11 maintains two arrays, *Reached* and *Explore*, both of size m where m is the number of discrete states. The q^{th} element of *Reached*, denoted by $Reached[q]$, contains an orthogonal polyhedron that represents the reachable set at discrete state q. Each element of the array *Explore* contains a list of polyhedra, each of which can be orthogonal or convex. The polyhedra of $Explore[q]$ represent the sets to be explored at discrete state q. The use of lists of polyhedra for *Explore* facilitates some modifications which will be explained later. The algorithm uses two additional arrays of orthogonal polyhedra, R_c and R_d, both of size m to store reachable states in one iteration.

Although the FL algorithm can work for linear systems, for efficiency purposes it is preferable to use the LIN algorithm to approximate δ_c whenever possible. Recall that the FL algorithm accepts as input only orthogonal polyhedra and the LIN algorithm only convex polyhedra; in order to use the LIN and FL algorithms in a uniform way, a pre-processing phase is needed: if the dynamics at q is linear and the set F to explore is an orthogonal polyhedron, we decompose it into hyper-rectangles. Similarly, if the dynamics at q is non-linear and F is convex, we over-approximate it by $grid_o(F)$.

We detail now the steps of Algorithm 10. In the continuous phase, we compute the continuous-successors of the unexplored states at every discrete state. Thus, the orthogonal polyhedron in $R_c[q]$ represents new states reachable by the continuous evolution at q. In the discrete phase, we compute the discrete-successors of the states in R_c, and hence $R_d[q]$ contains the polyhedra representing new reachable states generated by transitions leading to q. Therefore, the polyhedra in R_c and R_d represent the states reachable in one iteration. For the termination decision we check whether these polyhedra are included in the previously computed *Reached* and then add them to *Reached* in the update phase. The array *Explore* is next replaced with R_d since only the new states reachable by discrete evolution need to be explored in the next iteration. Safety checking is done at the beginning of every iteration, and the flag *Bad* indicates whether the bad set is reached.

6.3 Efficient Implementation

When coming to actual computation, our consideration is to economize on computational cost while ensuring the desired accuracy. The main practical limitation of Algorithm 10 is its computational cost, mostly due to the approximation of continuous-successors. One reason for this comes from the continuous dynamics: if the differential equations are stiff, the numerical integration procedure needs to reduce significantly the step-size (in a portion of the

Algorithm 11 (Concrete Verification Algorithm)

- **Initialization**
  $Bad = Stop = \texttt{False}$.
  For every $q \in \{1, \ldots, m\}$
  - $Reached[q] = \emptyset$, $Explore[q] = F_q$.
- **Main computation loop**
  As long as $Stop = \texttt{False}$ the algorithm repeats the following steps:
  1. Safety checking: for every $q \in \{1, \ldots, m\}$
     - If $Reached[q] \cap \mathcal{B} \neq \emptyset$ then $Bad = \texttt{True}$ and **go to Output**.
  2. Pre-processing: for every $q \in \{1, \ldots, m\}$
     - If f_q is linear
       Every non-convex orthogonal polyhedron P in $Explore[q]$ is replaced by $decomp(P)$.
     - If f_q is non-linear
       Every convex polyhedron P in $Explore[q]$ is replaced by $grid_o(P)$.
  3. Continuous phase
     Suppose $Explore[q]$ contains m_p polyhedra $\{P_j \mid j = 0, \ldots m_p\}$.
     For every $q \in \{1, \ldots, m\}$
     $$R_c[q] = \bigcup_{j=1}^{m_p} \widehat{\delta}_c(P_j)$$
  4. Discrete phase: for every $q \in \{1, \ldots, m\}$
     $$R_d[q] = \bigcup_{q' \in Q} \widehat{\delta}_{q'q}(R_c[q'])$$
  5. Termination checking:
     If $\forall q \in \{1, \ldots, m\}$ $(R_c[q] \cup R_d[q]) \subseteq Reached[q]$ then $Stop = \texttt{True}$.
  6. Update: for every $q \in \{1, \ldots, m\}$
     $$\begin{aligned} Reached[q] &= Reached[q] \cup R_c[q] \cup R_d[q] \\ Explore[q] &= R_d[q] \\ R_d[q] &= \emptyset \end{aligned}$$
- **Output**
  If $Bad = \texttt{True}$ then report that **the bad set is reached**. Otherwise, report that **the system is safe** and **stop**.

Figure 6.3: A concrete version of Algorithm 10.

time interval) to avoid instability. The other reason is the structure of the polyhedra to be explored: if they are too complex, the geometric operations can be prohibitively expensive. The measure of complexity for a polyhedron with regard to the LIN algorithm is the number of vertices and with regard to the FL algorithm is the number of faces.

We propose the following methods and strategies for improving the performance of the algorithm.

Convex-Hull approximation

When using the LIN algorithm to compute $\widehat{\delta}_c$ of a non-convex orthogonal polyhedron, we first need decompose it into hyper-rectangles, as described in the pre-processing phase, and then treat each hyper-rectangle in the continuous phase separately. This is evidently costly due to redundancy. An alternative is to over-approximate this orthogonal polyhedron by its convex hull (see Figure 6.4). This may reduce signinificantly the number of vertices and thus improve the efficiency in both time and space without affecting the correctness of the algorithm. Of course, the price is an increase of the approximation error, and a good compromise between precision and efficiency should be made. The idea of using convex hull as an abstract operator to accelerate the computation is also employed in [40] for timed automata and in [55] for 'linear' hybrid automata.

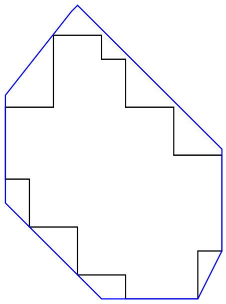


Figure 6.4: Over-approximating an orthogonal polyhedron by its convex hull to reduce the number of vertices.

Order of Exploration

The current algorithm works in a combination of breadth-first and depth-first search. The algorithm is breadth-first with respect to discrete transitions, but from the point of view of elapsed time, in each iteration we explore first *all* the continuous-successors at discrete state q_i and then all the continuous-successors at discrete state q_{i+1}. Alternatively, we can explore the discrete states in a breadth-first way. For example, we can fix a time limit for the continuous-successor computation in each discrete state. Beyond this limit, if the computation does not terminate we store unfinished sets in a local variable and add them to *Explore* at the end of the iteration. In case the system is unsafe, a different search order might accelerate the detection of intersection with the bad set. In general, some understanding of the system's dynamics is required to exploit this possibility. For a system which is safe, we need to explore the whole state-space, whatever order we use. However, experience with timed automata [20] shows that even in such situations the order of the search might have a notable influence on the performance of the algorithm.

Geometric Decompositions

For a given discrete state q, the polyhedra in $Explore[q]$ may contain states which have already been explored in the previous iterations. It is thus sufficient to continue with the states which are not included in the set $Reached[q]$. If the intersection of each polyhedron in $Explore[q]$ with $Reached[q]$ results in a less complex polyhedron (with regard to the algorithm used for computing $\widehat{\delta}_c$), then this will reduce the computational cost of the continuous phase at q. However, the resulting polyhedra may also be more complex, and in such cases the separation is no more of interest. To illustrate, consider two examples in Figure 6.5. The polyhedra $Explore[q]$ are drawn in dotted lines and the polyhedra resulting from separating $Explore[q]$ from $Reached[q]$ are shaded regions. One can see that in (a) the separation is advantageous since the resulting polyhedron has fewer faces as well as fewer vertices than $Explore[q]$, which is not the case in (b).

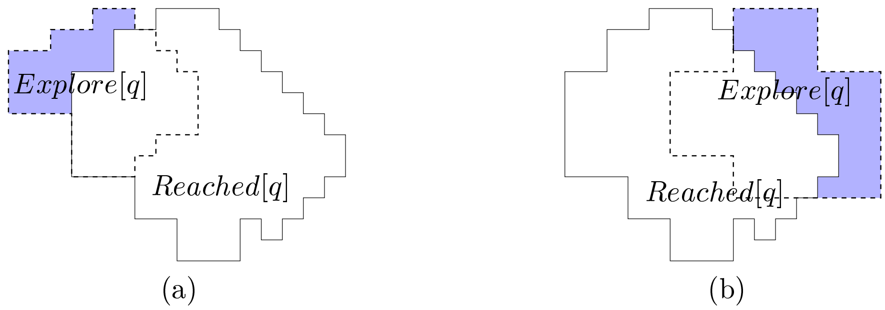


Figure 6.5: Separating $Explore[q]$ from $Reached[q]$ is advantageous in (a) but not in (b).

Another improvement can be made by exploiting the fact that Algorithm 9 for computing $\widehat{\delta}_{qq'}(q, F_q)$ can produce, as an intermediate result, a list of convex polyhedra whose union is already an over-approximation of $\delta_{qq'}(q, F_q)$. If $f_{q'}$ is linear then we can use these convex polyhedra instead of the orthogonal polyhedron $\widehat{\delta}_{qq'}(q, F_q)$ as the initial sets when treating the continuous dynamics at q'. This not only reduces the over-approximation error but also avoids decomposing the orthogonal polyhedron.

6.4 Error Analysis

In the sequel we discuss briefly the error in our approximation.

In the continuous phase, besides the error specific to the approximate algorithm (LIN or FL), the use of the $\sqcap_o$ operator introduces further error. The error incurred in the discrete phase is due to both $\sqcap_o$ and $grid_o$ operators. Although using the LIN algorithm the over-approximation errors do not accumulate during the continuous phase, they propagate to other continuous dynamics whenever a discrete transition is taken.

To illustrate how the over-approximation errors propagate after successive discrete and continuous evolutions, let us consider a single discrete transition, say, from discrete state q_1 to q_2 where $R_{q_1q_2}(\mathbf{x}) = D_{q_1q_2}\mathbf{x} + J_{q_1q_2}$.

Suppose that after treating the continuous dynamics at q_1, the error in the approximate set is ϵ_1. As shown in Section 6.2.2, when making the transition to q_2, we introduce an additional error bounded by $\epsilon_{12} = \sqrt{n}\beta(||D_{q_1q_2}|| + 1)$ where β is the grid size and n is the dimension of the hybrid automaton. Consequently, the bound on the total error is $\epsilon = \epsilon_1 + \epsilon_{12}$. At the target discrete state q_2, besides the new error inherent in the computation of $\widehat{\delta}_c$, the error ϵ evolves under the continuous dynamics f_{q_2}, in the worst case, as:

$$\epsilon(t) = e^{L_2 t}\epsilon, \quad t \geq 0.$$

where L_2 is a Lipschitz constant of f_{q_2}.

Hence $\epsilon(t)$ at this stage is bounded by $e^{L\tau}\epsilon$ where τ is the time needed for the computation to terminate at q_2. However, depending on the nature of the continuous dynamics at q_2, $\epsilon(t)$ may increase or decrease over time, and it is thus difficult to make a general statement about the magnitude of $\epsilon(t)$ when the next transition is taken. Due to the insensivity of the Lipschitz constant, the above estimation may not reflect the real error in practical situations. A better assessement requires a mechanism integrated in the computational package whereby the local error is evaluated and possibly controlled in the course of the computation.

6.5 Backward Verification Algorithm

The verification algorithm using forward reachability can be easily adapted for backward reachability analysis. It remains to compute the orthogonal over-approximations $\widehat{\Delta}_c$ and $\widehat{\Delta}_d$ of the continuous-predecessor Δ_c and discrete-predecessor Δ_d operators.

To compute $\widehat{\Delta}_c(q, F)$, one can use the algorithm for $\widehat{\delta}_c(q, F)$ on the reverse dynamics, i.e. f_q is replaced with $-f_q$.

We now show how to over-approximate discrete-predecessors. The exact set $\Delta_{q'q}(q, F)$ can be written as

$$\Delta_{q'q}(q, F) = (q', R^{-1}_{q'q}(F) \cap G_{q'q} \cap H_{q'})$$

where $R^{-1}_{q'q} : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is the inverse map of $R_{q'q}$ defined as $R^{-1}_{q'q}(X) = \{\mathbf{x}' \mid \exists \mathbf{x} \in X \; \mathbf{x} = R_{q'q}(\mathbf{x}')\}$. We consider the following two cases of $R_{q'q}$.

1. The map $R_{q'q}$ is memoryless, that is, $R_{q'q} = J_{q'q}$. Then,

$$\widehat{\Delta}_{q'q}(q, F) = \begin{cases} \emptyset & \text{if } F \cap J_{q'q} = \emptyset, \\ (q', grid_o(G_{q'q} \cap H_{q'})) & \text{otherwise.} \end{cases}$$

2. The map $R_{q'q}$ is $R_{q'q}(\mathbf{x}) = D_{q'q}\mathbf{x} + J_{q'q}$ where $J_{q'q} = conv\{\boldsymbol{j}_1, \ldots, \boldsymbol{j}_{m_j}\}$. Assume that the matrix $D_{q'q}$ is invertible. It is not hard to see that $R^{-1}_{q'q}(\mathbf{x}) = conv\{D^{-1}_{q'q}\mathbf{x} - \boldsymbol{j}_1, \ldots, D^{-1}_{q'q}\mathbf{x} - \boldsymbol{j}_{m_j}\}$, and for a convex polyhedron $P = conv\{\mathbf{v}_1, \ldots, \mathbf{v}_{m_p}\}$

$$R^{-1}_{q'q}(P) = conv(R^{-1}_{q'q}(\mathbf{v}_1) \cup \ldots \cup R^{-1}_{q'q}(\mathbf{v}_{m_p})).$$

   Hence, we can compute $\widehat{\Delta}_{q'q}(q, F)$ by decomposing F into m_b non-overlapping hyper-rectangles b_j and then

$$\widehat{\Delta}_{q'q}(q, F) = (q', \bigcup_{j=1}^{m_b} grid_o(R^{-1}_{q'q}(b_j) \cap G_{q'q} \cap H_{q'}))$$

With $\widehat{\Delta}_{q'q}(q, F)$ characterized as above, we can next compute the orthogonal over-approximation of discrete-successors as

$$\widehat{\Delta}_d(q, F) = \bigcup_{q' \in Q} \widehat{\Delta}_{q'q}(q, F).$$

The backward verification algorithm is sketched below. The algorithm computes the set backward reachable from the bad set $\mathcal{B}$ and checks whether it intersects with the initial set $\mathcal{F}$. Note that for a given safety verification problem, backward reachability may be more efficient than forward reachability and vice versa.

Algorithm 12 (Approximate Backward Verification Algorithm)

$\mathcal{P}^0 := \mathcal{B}$;
repeat $k = 0, 1, 2, \ldots$
  if $(\mathcal{P}^k \cap \mathcal{F} \neq \emptyset)$ **return** *bad-set-reached*
  $\mathcal{P}_c := \widehat{\Delta}_c(\mathcal{P}^k)$;
  $\mathcal{P}_d := \widehat{\Delta}_d(\mathcal{P}_c)$;
  $\mathcal{P}^{k+1} := \mathcal{P}^k \cup \mathcal{P}_c \cup \mathcal{P}_d$;
until $\mathcal{P}^{k+1} = \mathcal{P}^k$
return *safe*

6.6 Verification Examples

In the sequel we illustrate our approach with some results obtained using the above algorithms implemented in *d/dt* .

6.6.1 Example 1

Consider the hybrid automaton sketched in Figure 6.6. This automaton has two discrete states q_1 and q_2 whose dynamics are the following:

$$A_1 = \begin{pmatrix} -2.0 & -3.0 \\ 3.0 & -2.0 \end{pmatrix}, \quad A_2 = \begin{pmatrix} 0.0 & -0.6 \\ 3.0 & 0.0 \end{pmatrix}.$$

The initial set is $\mathcal{F} = (q_1, [0.3, 0.6] \times [-0.2, 0.2])$. In other words, the system starts at discrete state q_1 from the rectangle shown in Figure 6.7-(a). The successors by A_1 (a center dynamics) are computed until the trajectories all go out of the staying conditions H_1 ($x_1 \geq -0.15$). The intersection with the guard G_{12} ($x_1 = -0.15$) is then computed and from there the dynamics A_2 is applied, shrinking the set until all the trajectories go out of H_2 (see Figure 6.7-(b)). From the intersection with the guard G_{21} ($x_1 = -0.02$) the dynamics A_1 induces a "ring" of states which stay in q_1 forever (see Figure 6.7-(c)).

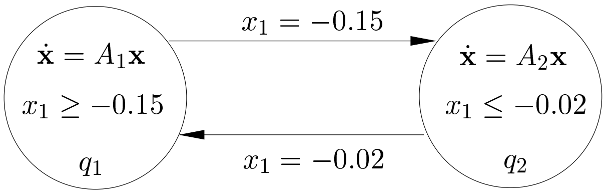


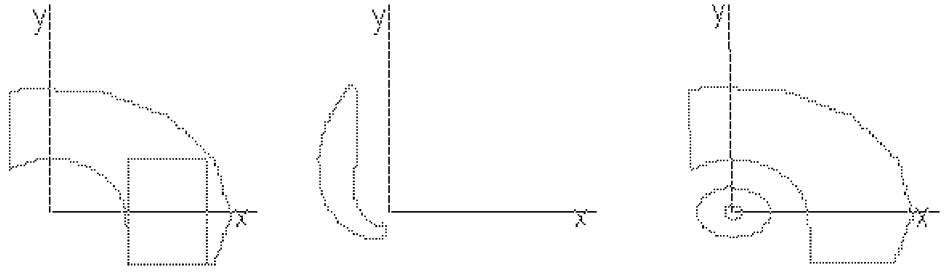
Figure 6.6: A hybrid automaton.

6.6.2 Collision Avoidance

The second example is the model of a single lane of highway of an Automated Vehicle/Highway System (AVHS), taken from [100].

Consider a vehicle i on the lane. Let d_i be the distance of vehicle i from the origin, and let s_i and a_i be its speed and acceleration, respectively (see Figure 6.8). The dynamics of each vehicle depends on the state of the vehicle in front; therefore, to avoid an infinite dimensional problem, when studying vehicle i, the dynamics of the vehicle $i-1$ is conservatively abstracted by $a_{i-1} = [A_l, A_u]$ where A_l is the maximum deceleration and A_u is the maximum acceleration. Let $d = d_{i-1} - d_i$ be the distance between vehicles i and $i-1$. We focus on the leader control mode in which vehicle i follows vehicle $i-1$, and the control law is written as follow:

$$\dot{a}_i = -3a_i - 3(s_i - s_{i-1}) + d - s_i + 10. \tag{6.5}$$



q_1 q_2 q_1

(a) (b) (c)

Figure 6.7: The 3 stages in the computation of the reachable set.

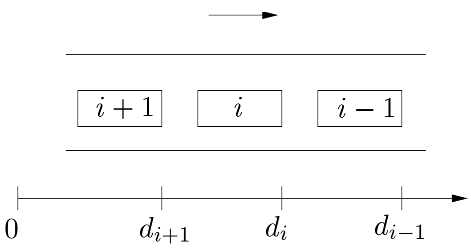


Figure 6.8: A single lane of highway.

This control mode is applied when the inter-vehicle distance is small or the relative speed between vehicles is large. More precisely, it is applied to vehicle i if the following conditions are satisfied.

$$\begin{cases} d \geq 5 \\ s_i \in [0, 30] \\ s_{i-1} \in [0, 30] \\ a_i \in [-5, 2] \\ d + (s_i^2 - s_{i-1}^2)/2A_l - (s_i - s_{i-1}) \geq 10 \end{cases} \tag{6.6}$$

A detailed description of the AVHS can be found in [100].

The goal is to prove that the control law (6.5), when applied to vehicle i, guarantees that collision between vehicles i and $i-1$ never happens, i.e. the distance d is always positive, regardless of the behavior of vehicle $i-1$. For doing this, we consider the following dynamical system:

$$\begin{aligned}
\dot{d} &= s_i - s_{i-1} \\
\dot{s}_i &= a_i \\
\dot{s}_{i-1} &= a_{i-1} \\
\dot{a}_i &= -3a_i - 3(s_i - s_{i-1}) + d - s_i + 10
\end{aligned}$$

The continuous state of the system is (d, s_i, s_{i-1}, a_i), and the last differential equation describes the control law. Note that the acceleration a_{i-1} of vehicle $i-1$ is now the input (disturbance) of the system ranging inside $[A_l, A_u]$ where $A_l = -5\,m/s^2$ and $A_u = 2\,m/s^2$. In addition, the system is subject to the constraints $\dot{s}_{i-1} \geq 0$ and $\dot{s}_i \geq 0$. Therefore we model the system as a one-state hybrid automaton with these constraints as staying conditions at the discrete state. To prove that the system never reaches a state where $d \leq 0$, we take the set S_l described by (6.6) as the initial set and compute its successors. The last inequality of (6.6) is non-linear, and to over-approximate the initial set by a convex polyhedron we replace it with

$$s_{i-1} - \frac{27}{40} s_i \geq 3.$$

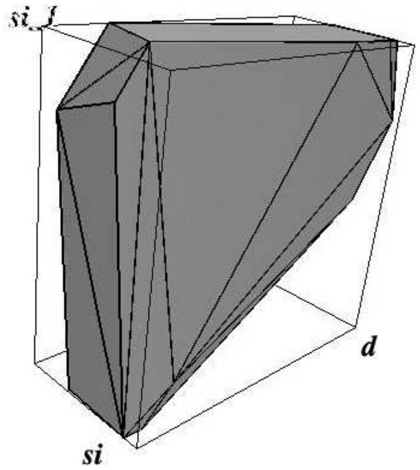


Figure 6.9: The reachable set projected on the first three dimensions at time point $t = 0.2s$.

We perform bounded time reachability analysis, and the result obtained shows that the system is safe until $200s$. Termination can be checked if we restrict the analysis to a bounded set. The projection of the reachable set at time point $t = 0.2s$ on the first three dimensions is drawn within its bounding box in Figure 6.9. This safety property was proved by the authors of [100] using optimal control techniques.

6.6.3 Double Pendulum

Another example we consider is inspired by the biped robot developed at Inria Rhone-Alpes. We consider a simplified model of a robotic leg having two motors installed at the hip and knee (see Figure 6.10). The continuous state of the system is $\mathbf{z} = \begin{pmatrix} z_1 \\ z_2 \end{pmatrix}$ where z_1 and z_2 are the hip and knee angles. The dynamics of the robot leg is described by the well-known double pendulum equations:

$$M(\mathbf{z})\ddot{\mathbf{z}} + N(\mathbf{z}, \dot{\mathbf{z}}) = W\Gamma \tag{6.7}$$

where M is the matrix defining the kinetic energy; N gathers generalized gravity, Coriolis, and centrifugal forces; Γ includes all external generalized forces; W is a constant matrix. We focus on the case where the system is under-actuated: $W = \begin{pmatrix} 1 \\ 0 \end{pmatrix}$, that is, only the hip is actuated.

Consider the following coordinate transformation parametrized by $\mathbf{a} = (a_1, a_2) \in \mathbb{R}^2$:

$$\mathbf{z} \to \begin{pmatrix} e_1(\mathbf{z}) \\ e_2(\mathbf{z}) \end{pmatrix} = \begin{pmatrix} z_1 - a_1 z_2 - a_2 \\ z_2 \end{pmatrix}.$$

We are interested in the zero dynamics, that is, e_1 is stabilized at 0 by using the control Γ (the control law that drives the system to the zero dynamics is detailed in [42]). Once the zero dynamics is reached, the motion of the pendulum is free since no more control is available ($dim(e_1) = dim(W) = 1$). In many cases, this free motion is periodic, i.e. the trajectory of the system starting from a given point $\mathbf{z}$ is a periodic orbit (corresponding to an energy level) and determined uniquely by the value of $\mathbf{a}$. We denote this orbit by $\xi_{\mathbf{z},\mathbf{a}}$. Hence, the parameter $\mathbf{a}$ can be used as an additional control (or a supplementary degree of freedom). In this study, the parameter $\mathbf{a}$ is discretized, and we are interested in the following question: "Is a goal orbit reachable from an initial orbit by a *finite* number of discrete jumps in $\mathbf{a}$?". A more general question is whether a set of target orbits can be reached from a connected set of initial orbits. In particular, we want to find the sequence of discrete values of $\mathbf{a}$ that steers the system from low-energy orbits to high-energy ones and, moreover, with the *least number of jumps*. In the following we will show how to use our reachability algorithms implemented in *d/dt* to solve this synthesis problem.

The evolution of the system in the zero dynamics is described by the following equation resulting from the projection of (6.7) on the constraint $e_1 = 0$:

$$\mathcal{M}(\mathbf{a}, z_2)\ddot{z}_2 + \mathcal{C}(\mathbf{a}, z_2)\dot{z}_2 + \mathcal{G} = 0$$

To facilitate reachability analysis, we will linearize the system. Without getting into detail (see [42]), the above equation, linearized about an equilibrium point $\begin{pmatrix} z_2^* \\ \dot{z}_2^* \end{pmatrix} = \begin{pmatrix} \frac{a_2}{1+a_1} \\ 0 \end{pmatrix}$, becomes

$$\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}, \tag{6.8}$$

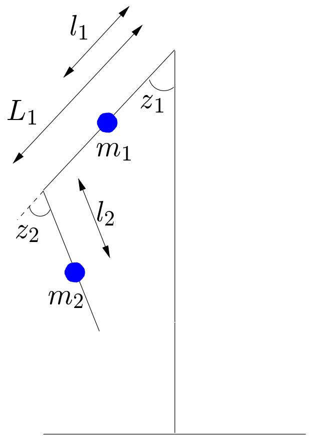


Figure 6.10: A double pendulum.

where

$$\mathbf{x} = \begin{pmatrix} z_2 \\ \dot{z}_2 \end{pmatrix}, \ \mathbf{u} = \begin{pmatrix} 0 \\ \alpha z_2^* \end{pmatrix},$$

$$A = \begin{pmatrix} 0 & 1 \\ -\alpha & 0 \end{pmatrix}, \ \alpha = l_2 + \frac{a_1}{1+a_1} L_1 cos(\frac{a_2}{1+a_1}).$$

The condition for ensuring the existence of periodic orbits is $\alpha > 0$.

The geometric parameters of the double pendulum at the robotics laboratory in Inria [42] are as follows: $L_1 = 0.52m$, $l_1 = 0.3m$, $l_2 = 0.29m$, $m_1 = 6kg$, $m_2 = 4kg$, and the gravitational acceleration $g = 9.81m/s^2$.

Since the pendulum is submitted to physical bounds on the joints: $z_i \in [z_i^{min}, z_i^{max}]$, the equation $e_1 = z_1 - a_1 z_2 - a_2 = 0$ of the zero dynamics leads to linear constraints in the parameter $\mathbf{a}$. We consider 7 discrete values of $\mathbf{a}$:

$$\begin{aligned} &\mathbf{p}_0 = (-0.32, -0.1), \ \mathbf{p}_1 = (-0.28, 0.0), \ \mathbf{p}_2 = (-0.25, 0.05), \\ &\mathbf{p}_3 = (-0.15, 0.0), \ \mathbf{p}_4 = (-0.1, 0.1), \ \mathbf{p}_5 = (0.05, 0.2), \ \mathbf{p}_6 = (0.15, 0.3). \end{aligned} \tag{6.9}$$

Thus, the pendulum can be modeled as a 7-state hybrid automaton. In each discrete state q_i corresponding to a parameter value $\mathbf{p}_i$, the linearized continuous dynamics is given in (6.8). Since the controller is subject to mechanical constraints, the transitions between discrete states (or the jumps in $\mathbf{a}$) cannot be arbitrary. More precisely, we can switch from $\mathbf{a}$ to $\mathbf{a}'$ if $|a_i - a_i'| \leq \delta a_i, \ i = 1, 2$. The transition relation of the hybrid automaton is shown in Figure 6.11. Moreover, when we switch from $\mathbf{a}$ to $\mathbf{a}'$ there might be a transient period until the system reaches the new zero dynamics. Hence, we need to make sure that e_1' and $\dot{e}_1'$ be already close to their zero. For e_1' this means

$$|e_1'| = |z_1 - a_1' z_2 - a_2'| < \epsilon_1$$

Since $e_1 = z_1 - a_1 z_2 - a_2 = 0$, the above leads to

$$|(a_1 - a_1')z_2 + (a_2 - a_2')| < \epsilon_1 \tag{6.10}$$

For $\dot{e}_1'$ we have

$$|\dot{e}_1'| = |(a_1 - a_1')\dot{z}_2| < \epsilon_2 \tag{6.11}$$

The conditions (6.10) and (6.10), which form rectangles in the phase-space of the zero dynamics, will be used as transition guards in the hybrid automaton model. Note that these conditions are symmetric, i.e. they are the same for the transition from (a_1', a_2') to (a_1, a_2).

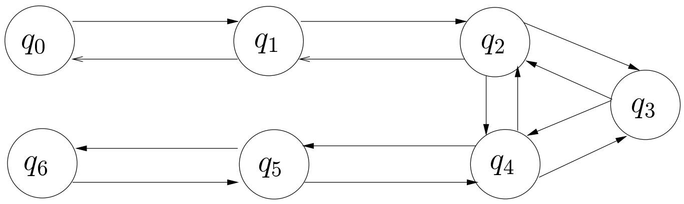


Figure 6.11: The transition relation of the hybrid automaton of the pendulum with 7 discrete values of $\mathbf{a}$ given in (6.9).

Let O_t be the set of goal orbits $\{\xi_{\mathbf{x},\mathbf{a}_t} \mid \mathbf{x} \in F_t\}$, and let O_0 be the set of initial orbits $\{\xi_{\mathbf{x},\mathbf{a}_0} \mid \mathbf{x} \in F\}$. We describe briefly the procedure for finding the switching sequence (the detailed algorithm is given in [9]). Starting with an initial orbit in the $(z_2, \dot{z}_2)$ space, we calculate in a breadth-first manner all the continuous-successors of F at the discrete state corresponding to $\mathbf{a}_0$, and then, via instersection with the guards, the discrete-successors. We continue until at some level k of the search tree, the computed reachable sets at one or more nodes intersect with O_t. From these non-empty intersections we do backward reachability analysis to find the subset F_k of F from which the goal orbits can be reached using k transitions. Next, we restart the process from F' where $F' = F \setminus F_k$ is the subset of F consisting of the points not covered yet.

We illustrate the process where the initial set F is the rectangle $[0.7\times 0.9]\times[0.01, 0.02]$ at state q_3 (i.e. $\mathbf{a}_0 = \mathbf{p}_3$), and the goal orbits are specified by the rectangle $F_t = [1.05, 1.3]\times[0.01, 0.02]$ at the same state (i.e. $\mathbf{a}_t = \mathbf{p}_3$). We choose $\epsilon_1 = 0.05$ and $\epsilon_2 = 0.02$. The search tree of the first iteration is shown in Figure 6.12 where there are two intersections with the goal orbit after 4 transitions. Going backward we conclude that initial states satisfying $z_2 \in [0.7552, 0.9]$ can reach the goal orbits by following the sequence q_3, q_2, q_3, q_2, q_3 and those satisfying $z_2 \in [0.7152, 0.9]$ can do it following the sequence q_3, q_2, q_1, q_2, q_3. Note that from states in the set $[0.7552, 0.9] \times [0.01, 0.02]$ both sequences can be used. In the second iteration, we start with the remaining initial states, i.e. $z_2 \in [0.7, 0.7152]$ and find that from there the goal orbits can be reached by either one of the three following 6-transition sequences: $q_3, q_2, q_3, q_2, q_3, q_2, q_3$; $q_3, q_2, q_3, q_2, q_1, q_2, q_3$; $q_3, q_2, q_1, q_2, q_1, q_2, q_3$. The fact that $\dot{z}_2$ does not matter here is particular to this example — with other sets of parameters the partition of the initial set did involve conditions on $\dot{z}_2$. The reachable states which correspond to the finding of the sequence q_3, q_2, q_3, q_2, q_3 are depicted in Figures 6.13 and 6.14.

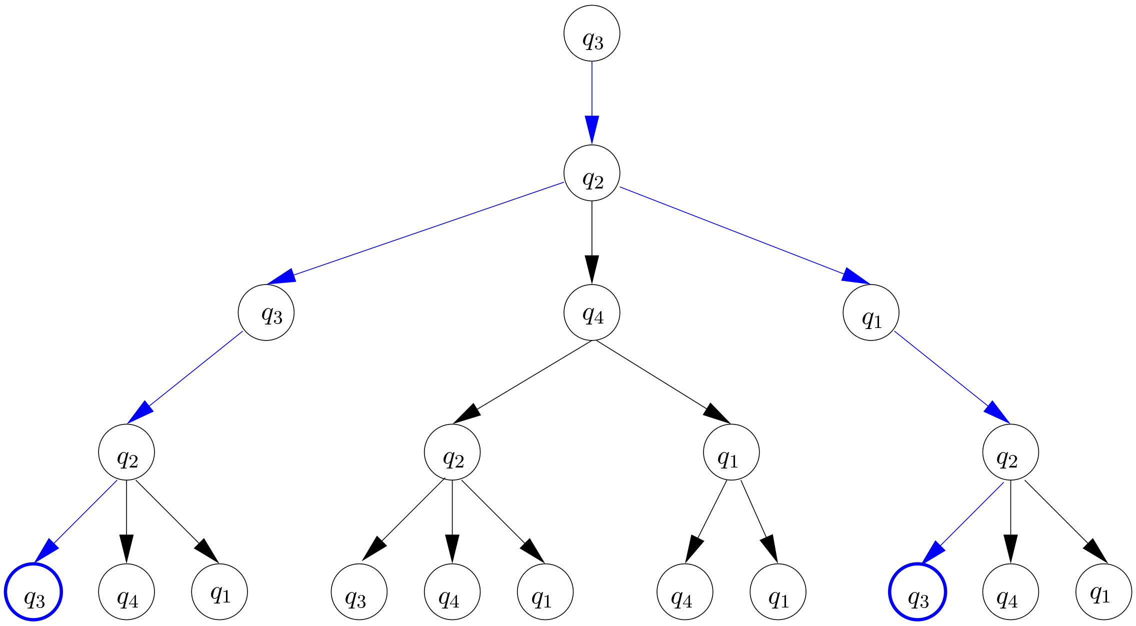


Figure 6.12: The search tree of the first iteration: the goal orbits are first reached after 4 switchings along two paths q_3, q_2, q_3, q_2, q_3 and q_3, q_2, q_1, q_2, q_3.

6.7 Summary

We have described a verification algorithm for hybrid automata based on the approximate reachability techniques for continuous systems. Our algorithm can work with a large class of hybrid systems with arbitrary continuous dynamics and rather general discrete dynamics. We have also discussed various methods for increasing the efficiency of the algorithm and an adaptation for backward reachability analysis. To illustrate the applicability of our approach, we have shown some examples treated using the implemented algorithms.

Along these lines, there are the works of [31], [24], and [59], the results of which are the hybrid system verification tools CheckMate, VeriShift, and HyperTech, respectively. We defer a discussion on these works to Chapter 8 after presenting our tool *d/dt* .

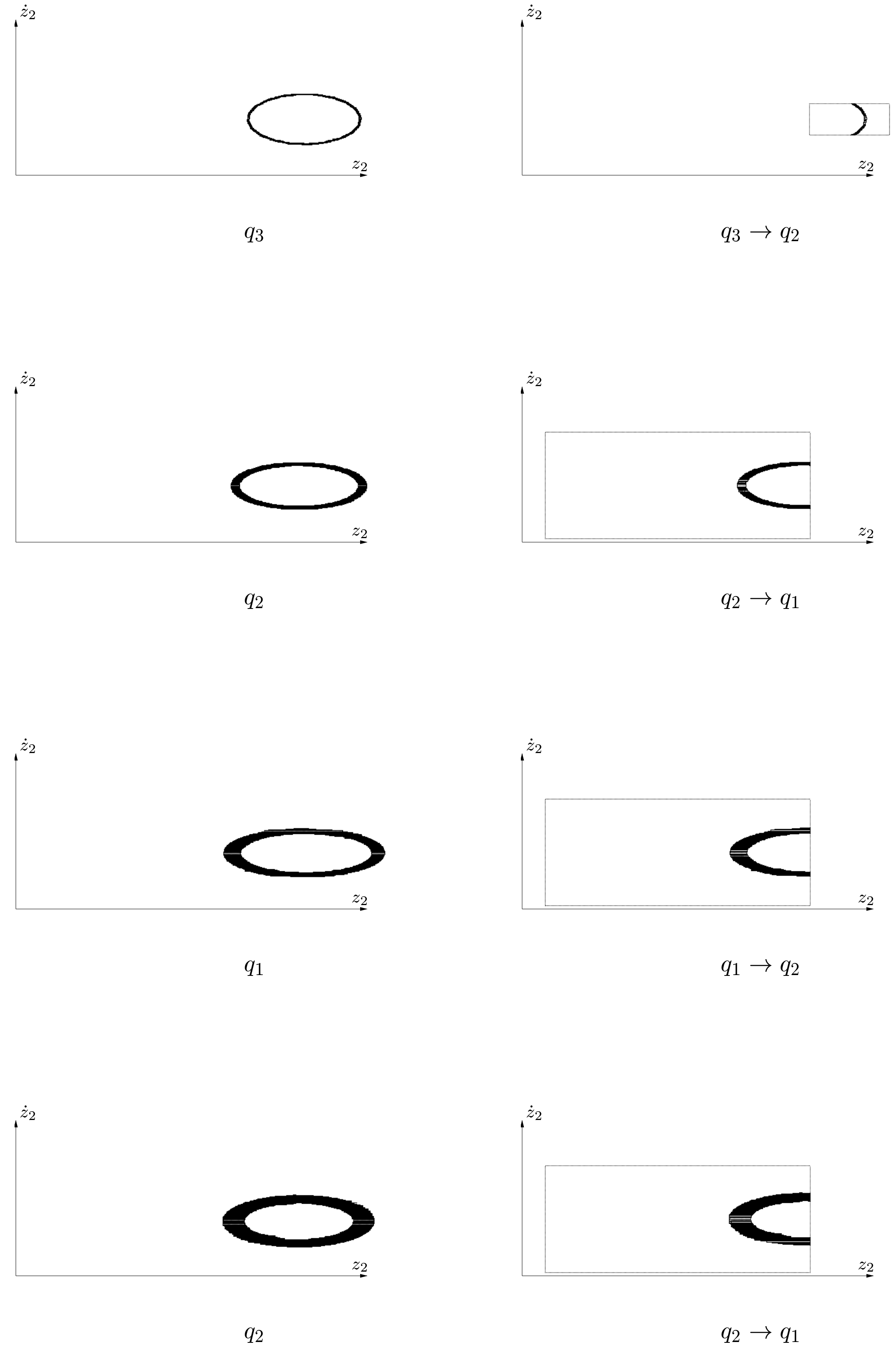


Figure 6.13: Computation of reachable states for the sequence $q_3, q_2, q_1, q_2, q_1, q_2, q_3$: on the left we see the reachable set at mode q_i, and on the right we show the intersecion with the guard from q_i to q_j.

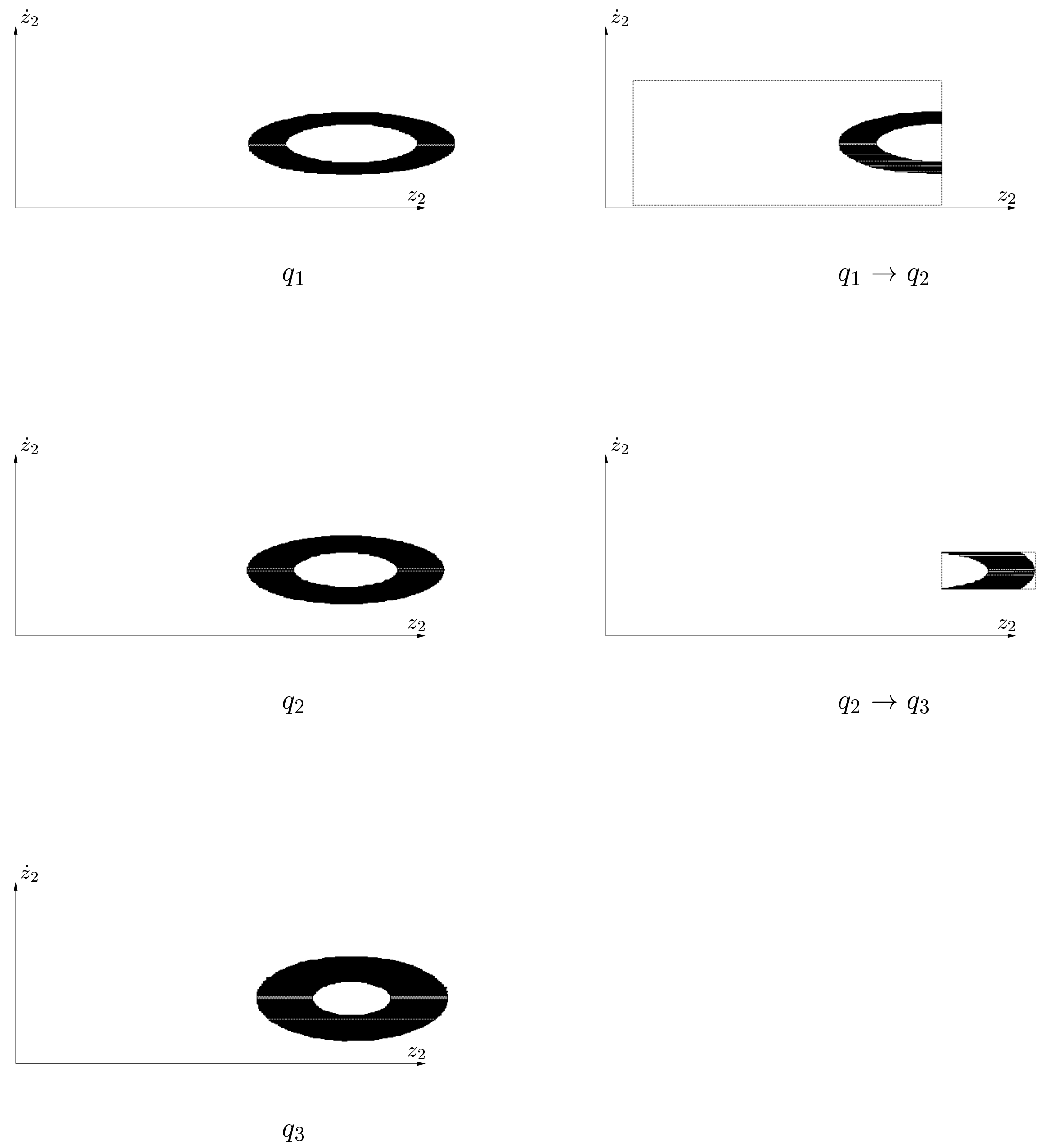


Figure 6.14: Computation of reachable states for the sequence $q_3, q_2, q_1, q_2, q_1, q_2, q_3$ continued from Figure 6.13.

Part III

Controller Synthesis

Chapter 7

Switching Controller Synthesis

In this chapter we study the problem of synthesizing switching controllers for hybrid systems. The setup of the system we consider is shown in Figure 7.1. The system, which we refer to as the *plant*, can be in one of several "modes", in each of which its behavior is governed by a distinct continuous dynamics. In some zones of the continuous state space $\mathcal{X} \subseteq \mathbb{R}^n$ the system can switch from one mode to another. These modes can arise from different structures of a continuous system (such as gears in a car or combination of open and closed valves in a liquid container), the use of different operation ranges of continuous regulators, the approximation of non-linear continuous systems by piecewise-linear ones, etc.. The choice between the modes is made by a *discrete controller*, which continuously observes the state of the plant and decides continuously which mode to select. The discrete controller can thus be modeled as an automaton with a set Q of states, where each state is identified with a mode of the plant. We assume the controller has complete observability of the plant, in other words, the observation space of the controller is $\mathcal{X}$. Hence, the domain of the feed-back map is $Q \times \mathcal{X}$.

We allow the controller to be non-determinisitic, i.e. a function $s : Q \times \mathcal{X} \rightarrow 2^Q$. This means that when the current mode is q and the state of the plant is $\mathbf{x}$, the controller might choose to stay in q as well as to switch to one of several other modes. In synthesis problems, one usually starts with a "liberal" controller which allows the system to be in one of several modes at a given state. In other words, $s(q, \mathbf{x})$ contains all the modes to which it is physically possible to switch from $(q, \mathbf{x})$. The result of the synthesis process is a more restrictive controller which allows to stay in a mode or to switch to another mode only if this does not lead to bad consequences. As in the previous sections, we concentrate on invariance properties, namely the avoidance of bad states. In our setting, the combined system which is a product of the controller and the environment is viewed as a hybrid automaton over the state space $Q \times \mathcal{X}$, and the synthesis algorithm is performed on this automaton.

We begin by discussing some aspects of hybrid automata behavior, which are important in the synthesis context. Next, we define formally the controller synthesis problem and give an abstract algorithm to solve it. We then describe an approximate implementation of this

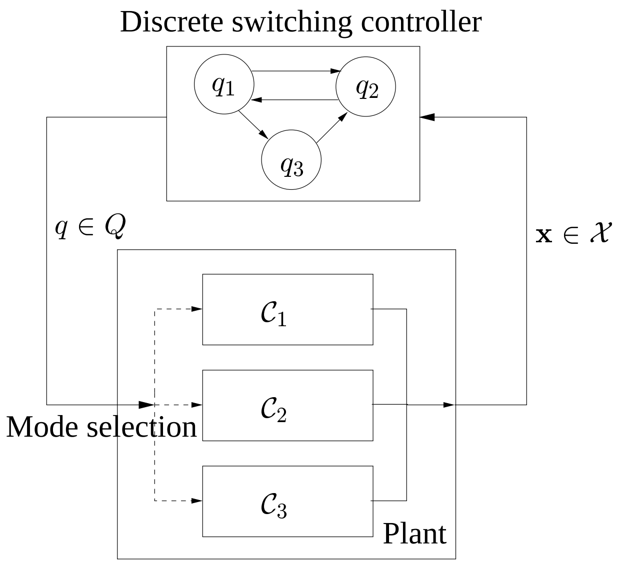


Figure 7.1: A plant with a switching controller.

algorithm for systems with linear continuous dynamics, which is based on the reachability techniques presented in Chapters 4 and 6 and gives an approximate switching controller that guarantees correctness. We illustrate our approach with some examples and conclude with a discussion of related work. The material of this chapter is developed from the presentation in [11].

7.1 Preliminaries

The automata-theoretic formulation of the synthesis problem is as follows: given a hybrid automaton, restrict the transition guards and staying conditions of the automaton so that all remaining trajectories satisfy some safety specification. Our approach is to iteratively reduce the staying and guard sets, so that the automaton will be forced to leave a continuous dynamics which leads to bad states and will not take transitions which lead to such states, until we obtain an automaton all of whose behaviors are good. There are two anomalies that may result from a careless application of this approach, namely *blocking* and *Zeno behaviors*.

Blocking Behavior

In the process of iteratively restricting the automaton, one should ensure that the resulting automata are non-blocking, that is, from *every reachable state* there is always either a continuous dynamics or a discrete transition which is enabled. To prevent blocking after transitions, for every q, $q' \in Q$, we replace $G_{qq'}$ with $G'_{qq'} = G_{qq'} \cap R^{-1}_{qq'}(H_{q'})$ where $R^{-1}_{qq'}$ is the inverse map of $R_{qq'}$. Hence,

$$R_{qq'}(G'_{qq'}) \subseteq H_{q'} \tag{7.1}$$

There is always a trivial way to synthesize controllers with respect to a given safety property by generating an automaton with $H = G = \emptyset$. This automaton has no trajectories at all and thus satisfies the property. In order to obtain sensible solutions, we are interested in finding the *maximal non-blocking* sub-automaton of $\mathcal{A}$ which satisfies the property. As in the theory of supervisory control of discrete-event systems [102], such a maximal controller exists for safety properties. However, since we are dealing here with hybrid systems, we will eventually produce an *approximation* of this maximal automaton.

Zeno Behavior

The importance of avoiding Zeno behavior in synthesis problems cannot be underestimated. Indeed, because of this inevitable anomaly associated with modeling of interaction between discrete and continuous dynamics, a synthesis algorithm may come to wrong conclusions and produce a controller that 'avoids' bad states by generating Zeno behaviors. Similar phenomena have been extensively studied in sliding mode control [43, 113, 12, 2]. To illustrate the problem, consider a car driving towards a wall. The driver's only means of control is to turn the radio on and off. The system can be modeled as an automaton with two discrete states corresponding to two modes (on and off) of the radio. In both discrete states, the derivative of the continuous variable x which models the distance from the car to the wall is $-v$ where $v > 0$ is the constant speed of the car (see Figure 7.2).

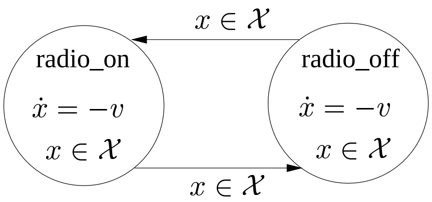


Figure 7.2: Car driving automaton (the staying and guard sets are the whole state space $\mathcal{X}$).

Clearly such a car is doomed to reach the bad state $x = 0$ and bump into the wall. However, formally, there is a behavior of the automaton where the driver switches the radio on and off infinitely many times in a bounded time interval, and consequently the car does not progress beyond a given point $x > 0$.

One straightforward way to ensure that our synthesis algorithm will not rely on such phenomena is to restrict its scope to hybrid automata which are non-Zeno by construction. This motivates the following definitions.

We consider a hybrid automaton $\mathcal{A} = (\mathcal{X}, Q, f, H, G, R)$. The set of all trajectories starting from $(q, \mathbf{x}) \in Q \times \mathcal{X}$ is denoted by $L(\mathcal{A}, (q, \mathbf{x}))$, and the set of trajectories starting from any $(q, \mathbf{x})$ such that $\mathbf{x} \in H_q$ is denoted by $L(\mathcal{A})$. The automaton $\mathcal{A}$ is non-Zeno if $L(\mathcal{A})$ contains no Zeno behaviors.

For the time being we assume that in the hybrid automata to be considered the *reset map* $R_{qq'}$ *is the identity* for all q, $q' \in Q$, that is, there are no jumps or resets in the values of the continuous variables when transitions are taken, and hence R will be omitted. An extension of the synthesis algorithm to automata with resets will be examined later.

Definition 22 (Strongly Non-Zeno Hybrid Automaton)

- *A state cycle of $\mathcal{A}$ is a sequence of states $q_1, \ldots q_s$ such that $q_1 = q_s$.*
- *A cycle is non-Zeno if there exists a sub-sequence of states q, q', q'' in the cycle such that $cl(G_{qq'}) \cap cl(G_{q'q''}) = \emptyset$ where cl is the closure operator.*
- *A hybrid automaton is strongly non-Zeno if all its cycles are non-Zeno.*

A Zeno cycle may allow the automaton to make a sequence of transitions leading from a state $(q, \mathbf{x})$ to itself indefinitely in non-diverging time and result hence in a Zeno behavior.

Lemma 9 *If the automaton $\mathcal{A}$ is strongly non-Zeno, then it is non-Zeno.*

Proof
By definition of strongly non-Zeno automata, every state cycle σ of $\mathcal{A}$ has at least a sub-sequence of states q, q', q'' such that the intersection of $cl(G_{qq'})$ and $cl(G_{q'q''})$ is empty, which means that there is a positive lower bound on the distance between $G_{qq'}$ and $G_{q'q''}$. Hence, every traversal of σ must do some continuous evolution between a point $\mathbf{x} \in G_{qq'}$ and a point $\mathbf{y} \in G_{q'q''}$. This implies that every behavior whose discrete part is cyclic must pass some time in the continuous phase and thus has a positive lower bound on its duration. Let d be the minimal such lower bound over all cycles.

Let consider a behavior $\gamma = (\alpha, \beta)$ of infinite logical length. Let q be a state which repeats infinitely often in the discrete behavior β. Then, the behavior β can be decomposed into $q, \ldots, q, \ldots, q, \ldots, q, \ldots$, that is, a concatenation of *finite* cyclic behaviors. As we have just shown, each cyclic behavior spends at least $d > 0$ time in the continuous phase, and hence any behavior of infinite logical length has an infinite metric length and is non-Zeno. ■

It can be verified that Lemma 9 is also true with a weaker definition of strongly non-Zeno automata, that is, the automaton $\mathcal{A}$ is strongly non-Zeno if every cycle $q_1, \ldots q_s$ of $\mathcal{A}$ satisfies

$$\bigcap_{i=1}^{s-1} G_{q_i, q_{i+1}} = \emptyset.$$

It is important to note that these conditions are both sufficient but not necessary for a hybrid automaton to be non-Zeno[1]. In addition, the above conditions on the intersection of guards are defined for automata without resets; for those with reset maps, more complex conditions need to be defined.

Definition 23 (Hybrid Automaton Restriction)
Let $\mathcal{A} = (\mathcal{X}, Q, f, H, G)$ and $\mathcal{A}' = (\mathcal{X}, Q, f, H', G')$ be two hybrid automata. We say that $\mathcal{A}'$ is more restrictive (in terms of behaviors) than $\mathcal{A}$, denoted by $\mathcal{A}' \preceq \mathcal{A}$, if $H' \subseteq H$ and $G' \subseteq G$, i.e. $H'_q \subseteq H_q$ and $G'_{qq'} \subseteq G_{qq'}$ for every $q, q' \in Q$.

Clearly if $\mathcal{A}' \preceq \mathcal{A}$ then $L(\mathcal{A}') \subseteq L(\mathcal{A})$ and, in addition, if $\mathcal{A}$ is non-Zeno, so is $\mathcal{A}'$.

Before formulating the synthesis problem we need some additional notations related to continuous evolutions of hybrid automata. Let us first recall the notations introduced in Chapter 2. A trajectory of the hybrid automaton $\mathcal{A} = (\mathcal{X}, Q, f, H, G)$ is a pair (α, β) where $\alpha : \mathcal{T} \to \mathcal{X}$ is a piecewise-continuous behavior and $\beta : \mathcal{T} \to Q$ is a piecewise-constant behavior. Let $\mathbf{x}$, $\mathbf{x}'$ be points in $\mathcal{X}$ and $q \in Q$. The notations $\mathbf{x} \xrightarrow{q,t}$ indicates that the dynamics q is enabled from $\mathbf{x}$ for time $t > 0$ and $\mathbf{x} \xrightarrow{q,t} \mathbf{x}'$ indicates that $\mathbf{x}'$ is q-reachable from $\mathbf{x}$ in time t.

Definition 24 (Continuous Evolution)
Let F and G be subsets of $\mathcal{X}$. Let $\gamma = (\alpha, \beta)$ be a trajectory of $\mathcal{A}$ starting from $(q, \mathbf{x})$.

- *If $\mathbf{x} \xrightarrow{q,t}$, and, in addition, $\alpha(t') \in F$ for every $t' \in [0, t]$ we write it as $\mathbf{x} \xrightarrow[F]{q,t}$.*
- *If $\mathbf{x} \xrightarrow{q,t} \mathbf{x}'$ and, in addition, $\alpha(t') \in F$ for every $t' \in [0, t]$ we write it as $\mathbf{x} \xrightarrow[F]{q,t} \mathbf{x}'$.*
- *The set G is q-reachable from $\mathbf{x}$ in time t if $\mathbf{x} \xrightarrow{q,t} \mathbf{x}'$ for some $\mathbf{x}' \in G$. We denote this by $\mathbf{x} \xrightarrow{q,t} G$. If, in addition, $\alpha(t') \in F$ for every $t' \in [0, t]$, we write it as $\mathbf{x} \xrightarrow[F]{q,t} G$ (F until G).*

7.2 The Problem and An Abstract Solution

We formulate a simple control problem of avoiding bad states in a non-trivial way.

Problem 4 (Safety Synthesis for Hybrid Automata)
Let $\mathcal{A} = (\mathcal{X}, Q, f, H, G)$ be a hybrid automaton and let $\mathcal{F}$ be a subset of $Q \times \mathcal{X}$. The safety controller synthesis problem is to find the maximal non-blocking hybrid automaton $\mathcal{A}^* \preceq \mathcal{A}$ such that for every trajectory $\gamma \in L(\mathcal{A}^*)$ and every $t \in \mathcal{T}$, $\gamma(t) \in \mathcal{F}$.

[1] In this thesis we do not address the problem of finding more precise non-Zenoness conditions. Recent works on this topic can be found in [79, 121].

The existence and uniqueness of the solution $\mathcal{A}^*$ to Problem 4 is not self-evident and will be shown in the proof of Theorem 5.

In the following we present a synthesis algorithm which works for strongly non-Zeno hybrid automata and then show how to generalize it to arbitrary hybrid automata. Our approach to Problem 4 is to calculate the maximal set $\mathcal{P}^*$ of 'winning' states, that is, the states from which the controller, by switching properly, ensures that all the trajectories of the controlled system lie within $\mathcal{F}$. Then, the switching rule can be derived from $\mathcal{P}^*$, and we will call $\mathcal{P}^*$ the *maximal invariant set.*

7.2.1 Characterizing the Maximal Invariant Set

The calculation of the set $\mathcal{P}^*$ is the core of any synthesis algorithm, and for doing this we make use of the following operators.

Definition 25 (Unbounded-time-predecessor Operator)
Given $q \in Q$, the unbounded-time-predecessor operator $\pi_q^\infty : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is defined for a set $X \subseteq \mathcal{X}$ as

$$\pi_q^\infty(X) = \{\mathbf{x} \mid \mathbf{x} \xrightarrow[X]{q,\infty}\}.$$

Intuitively, $\pi_q^\infty(X)$ is the set of states from which it is possible to continue indefinitely with the dynamics f_q while staying in X (see Figure 7.3(a)).

Definition 26 (Until Operator)
Given $q \in Q$ the until operator $\mathcal{U}_q : 2^{\mathcal{X}} \times 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ is defined for two sets X, $Y \subseteq \mathcal{X}$ as

$$\mathcal{U}_q(X,Y) = \{\mathbf{x} \mid \exists t \ \mathbf{x} \xrightarrow[X]{q,t} Y\}.$$

The set $\mathcal{U}_q(X,Y)$ consists of states from which it is possible to continue with the dynamics f_q and stay inside X until Y is reached (see Figure 7.3(b)).

Definition 27 (One-step Predecessor Operator)
The one-step predecessor operator $\pi : 2^{Q\times\mathcal{X}} \to 2^{Q\times\mathcal{X}}$ is defined for a set

$$\mathcal{F} = (q_1, F_1) \cup \ldots \cup (q_m, F_m)$$

where m is the number of discrete states in Q as

$$\pi(\mathcal{F}) = \{(q,\mathbf{x}) \mid \mathbf{x} \xrightarrow[F_q]{q,\infty} \ \vee \ (\exists t \ \exists q' \in Q \ \exists \mathbf{x}' \in \mathcal{X} \ \ \mathbf{x} \xrightarrow[F_q]{q,t} \mathbf{x}' \ \wedge \ \mathbf{x}' \in G_{qq'} \ \wedge \ (q',\mathbf{x}') \in \mathcal{F})\}.$$

The intuition behind this definition is the following. A state $(q,\mathbf{x})$ is in $\pi(\mathcal{F})$ if either there is an infinite trajectory without switching starting from $(q,\mathbf{x})$ and always staying in $\mathcal{F}$, or that it is possible to stay in $\mathcal{F}$ for some time and then make a transition to another state $(q',\mathbf{x}')$ which is still in $\mathcal{F}$. Note that, due to (7.1), the continuous evolution can always be

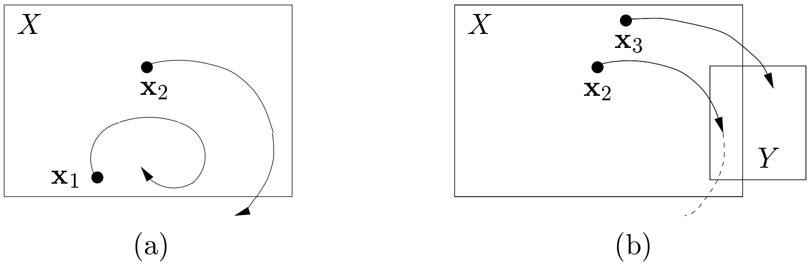


Figure 7.3: (a) Characterization of $\pi_q^\infty(X)$: the trajectory from point $\mathbf{x}_1$ stays in X forever while the trajectory from $\mathbf{x}_2$ leaves X after some time; therefore, $\mathbf{x}_1$ is in $\pi_q^\infty(X)$ but $\mathbf{x}_2$ is not. (b) Characterization of $\mathcal{U}_q(X, Y)$: the trajectory from $\mathbf{x}_2$ stays in X until it reaches Y while the trajectory from $\mathbf{x}_3$ leaves X before, and hence $\mathbf{x}_2$ is in $\mathcal{U}_q(X, Y)$ but $\mathbf{x}_3$ is not.

continued after a discrete transition is taken; hence, the condition $\mathbf{x}' \in H_{q'}$ can be omitted.

It is not hard to see that $\pi(\mathcal{F})$ can be expressed via the operators $\mathcal{U}_q$ and π_q^∞ as

$$\pi(\mathcal{F}) = (q_1, F_1') \cup \ldots \cup (q_m, F_m')$$

where for every q

$$F_q' = \pi_q^\infty(F_q) \cup \bigcup_{q' \neq q} \mathcal{U}_q(F_q, G_{qq'} \cap F_{q'}). \tag{7.2}$$

Figure 7.4 sketches the computation of F_q' where the dynamics f_q is the same as in the example of Figure 7.3.

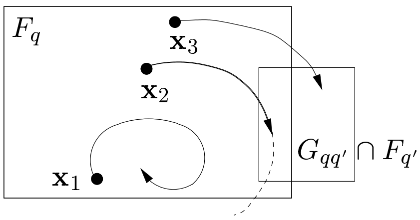


Figure 7.4: Computation of F_q': points $\mathbf{x}_1$ and $\mathbf{x}_2$ are in F_q' since $\mathbf{x}_1 \in \pi_q^\infty(F_q)$ and $\mathbf{x}_2 \in \mathcal{U}_q(F_q, G_{qq'} \cap F_{q'})$ (but $\mathbf{x}_2 \notin \pi_q^\infty(F_q)$). Point $\mathbf{x}_3$ is in neither and hence it is not in F_q'.

Based on the one-step predecessor operator we obtain the following abstract algorithm for computing the maximal invariant set $\mathcal{P}^*$.

Algorithm 13 (Computing $\mathcal{P}^*$)

> $\mathcal{P}^0 := \mathcal{F} \cap H;$
> **repeat** $k = 0, 1, 2, \ldots$
> $\quad \mathcal{P}^{k+1} := \mathcal{P}^k \cap \pi(\mathcal{P}^k);$
> **until** $\mathcal{P}^{k+1} = \mathcal{P}^k$
> $\mathcal{P}^* := \mathcal{P}^k;$

It is clear that the states in $\mathcal{F}$ but not in H are not admitted by the system, and it is thus sufficient to start with the set $\mathcal{P}^0 = \mathcal{F} \cap H$. Algorithm 13 produces a decreasing sequence $\{\mathcal{P}^k\}$, and if the algorithm terminates it gives the fixed point $\mathcal{P}^*$.

Lemma 10 (Property of Algorithm 13)
For every k, $(q, \mathbf{x}) \in \mathcal{P}^k$ iff $L(\mathcal{A}, (q, \mathbf{x}))$ contains a trajectory remaining invariantly in $\mathcal{F}$ which is either of logical length smaller than k and infinite metric length, or else of logical length not less than k.

Proof
The proof concerning the length of trajectories is done by induction. For the base case, all states in $\mathcal{P}^0$ admit empty trajectories of length zero and all states outside $\mathcal{P}^0$ (and outside $\mathcal{F}$) do not admit such trajectories. Consider a state $(q, \mathbf{x})$ in $\mathcal{P}^k$. If $(q, \mathbf{x})$ is in $\pi(\mathcal{P}^{k-1})$, it can either admit an infinite trajectory remaining invariantly in $\mathcal{P}^{k-1}$ (and thus in $\mathcal{P}^0 = \mathcal{F}$), or make one transition to $\mathcal{P}^{k-1}$ and then $(k-1)$ transitions from there. On the other direction, if $(q, \mathbf{x}) \notin \pi(P^{k-1})$, then it cannot make a transition to $\mathcal{P}^{k-1}$ nor an infinite trajectory, and hence it can make at most $(k-1)$ transitions. This proves Lemma 10. ■

The set $\mathcal{P}^*$ contains all the states for which there exist switching controllers that can prevent the system from going out of $\mathcal{F}$. We restrict the automaton $\mathcal{A}$ to $\mathcal{P}^*$ as follows.

Theorem 5 *The automaton $\mathcal{A}^* = (\mathcal{X}, Q, f, H^*, G^*)$ where for every q, q' $H^*_q = \{\mathbf{x} \mid (q, \mathbf{x}) \in \mathcal{P}^*\}$ and $G^*_{qq'} = G_{qq'} \cap H^*_q \cap H^*_{q'}$ is the solution of the safety controller synthesis problem.*

Proof
By Lemma 10, the set $\mathcal{P}^*$ is the set of all states which admit either a trajectory inside $\mathcal{F}$ of finite logical length whose last interval is infinite or a trajectory of infinite logical length, which (for strongly non-Zeno hybrid automata) implies an infinite metric length. This shows that $\mathcal{A}^*$ is a non-blocking automaton whose trajectories always stay in $\mathcal{F}$.

In addition, in each iteration Algorithm 13 computes $\mathcal{P}^{k+1}$ by removing from $\mathcal{P}^k$ the states from which leaving $\mathcal{P}^k$ is unavoidable. We then deduce that all the trajectories from $\mathcal{X} \setminus \mathcal{P}^k$ leave $\mathcal{F}$ after at most k transitions; as a result, any automaton larger than $\mathcal{A}^*$ will contain states outside $\mathcal{P}^*$ from which the system goes out of $\mathcal{F}$ after a finite amount of time. We next conclude that $\mathcal{A}^*$ *is the largest automaton whose trajectories can be extended to infinity without leaving* $\mathcal{F}$. ■

7.2.2 Switching Controller

A switching controller can be derived from $\mathcal{A}^*$ by defining a feed-back map $s : Q \times \mathcal{X} \to 2^Q$ as

$$s(q, \mathbf{x}) = \{q' \mid (q' = q \wedge \mathbf{x} \in H_q^*) \vee (q' \neq q \wedge \mathbf{x} \in {G^*}_{qq'})\}. \quad (7.3)$$

Notice that, unlike in continuous systems, the feed-back control depends not only on the continuous state but also on the discrete state of the system.

This switching controller is non-deterministic since the sets H_q^* and $G_{qq'}^*$ might not intersect with each other only on their boundaries, and hence in some parts of the state space the choice between continuing with dynamics q and switching to q' is not specified. This is similar to the notion of "least restrictive supervisor" [102], that is, for all $(q, \mathbf{x}) \in Q \times \mathcal{X}$, all other switching controllers that keep the system inside $\mathcal{F}$ have the feed-back maps which are contained in s.

A deterministic controller can be obtained by reducing H^* and G^* so that the feed-back map becomes a function $s : Q \times \mathcal{X} \to Q$. In general, there is no "canonical" reduction preferable over the others, and we consider it an implementation issue.

Let us review what has been resolved so far. We have presented an abstract solution to the safety controller synthesis problem which consists in restricting the automaton $\mathcal{A}$ to the set $\mathcal{P}^*$ characterized as the maximal fixed point of the equation $\mathcal{P} = \mathcal{F} \cap \pi(\mathcal{P})$. This solution can be useful if one is able to effectively implement Algorithm 13 whose main ingredient is the π operator. The following section is concerned with the problem of computing this operator.

7.3 From Abstract to Effective Algorithm

From now on we restrict the *continuous dynamics* to be *linear* of the form $f_q(\mathbf{x}) = A_q\mathbf{x}$ for every $q \in Q$.

Given a set $\mathcal{F} = \{(q, F_q) \mid q \in Q \ \wedge \ F_q \subseteq \mathcal{X}\}$, our goal is to compute the set $\pi(\mathcal{F})$. We derive from (7.2) Algorithm 14 for characterizing $\pi(\mathcal{F})$.

The algorithm uses the operators $\mathcal{U}_q$ and π_q^∞ which, like the successor and predecessor operators in the verification algorithms, cannot, in general, be exactly computed. Our approach to an effective synthesis algorithm is to use our reachability techniques to under-approximate these operators. It should be noted that for synthesis problems *under-approximations* are

Algorithm 14 (Computing $\pi(\mathcal{F})$)

$\mathcal{P} := \emptyset$;
for all $q \in Q$ {
  $X := \pi_q^\infty(\mathcal{F}_q)$;
  for all $q' \neq q$ {
    $X := X \cup \mathcal{U}_q(\mathcal{F}_q, G_{qq'} \cap F_{q'})$;
  }
  $\mathcal{P} := \mathcal{P} \cup (q, X)$;
}
return $\mathcal{P}$

required since one needs to guarantee that the computed maximal invariant set is a subset of the exact set $\mathcal{P}^*$.

We begin with the until operator. Let us rephrase the meaning of this operator in terms of reachable sets of hybrid automata. The set $\mathcal{U}_q(X, Y)$ is simply the set of states from which the system can reach Y while remaining in X. Thus, $\mathcal{U}_q(X, Y)$ can be characterized as the set of continuous-predecessors of (q, Y), i.e. $\Delta_c(q, Y)$, with X as the staying condition at q. Note that computing continuous-predecessors is equivalent to computing continuous-successors of the reverse dynamics: $\dot{\mathbf{x}} = -A_q\mathbf{x}$.

We proceed with the operator π_q^∞. Let $\overline{X}$ be the complement of X. It is not hard to see that the set $\pi_q^\infty(X)$ can be obtained by removing from X the states from which the continuous dynamics f_q leads the system to $\overline{X}$. In other words, one needs first to compute the set of predecessors of $\overline{X}$ by the dynamics f_q, i.e. $\Delta(\overline{X})$ where Δ denotes the predecessor operator of continuous systems. Then, $\pi_q^\infty(X) = X \setminus \Delta(\overline{X})$.

We conclude from the formulation of π using the predecessor operators that we can over-approximate it by orthogonal polyhedra using the machinery for linear continuous and hybrid systems, developed in Chapters 4 and 6.

Example

Let us now illustrate the above computations with an example where the sets X and Y are rectangles defined as

$$\begin{aligned} X &= [-0.1, 0.1] \times [-0.03, 0.1], \\ Y &= [0.02, 0.06] \times [-0.05, -0.02], \end{aligned}$$

and the linear dynamics f_q is defined by the matrix

$$A_q = \begin{pmatrix} -0.5 & 4.0 \\ -3.0 & -0.5 \end{pmatrix}.$$

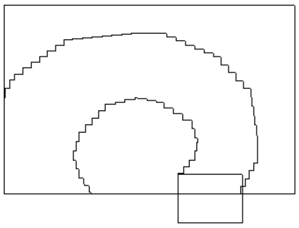


Figure 7.5: Under-approximation of $\mathcal{U}_q(X,Y)$: all the trajectories from points in the orthogonal polyhedron remain in the rectangle X until they reach the rectangle Y.

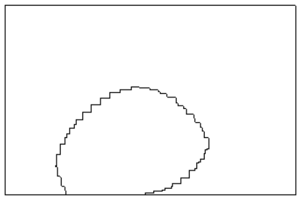


Figure 7.6: Under-approximation of $\pi_q^\infty(X)$: all the trajectories from points in the orthogonal polyhedron can stay in the rectangle X forever.

The results obtained for the approximation of $\mathcal{U}_q(X,Y)$ and $\pi_q^\infty(X)$ are depicted in Figures 7.5 and 7.6, respectively. In Figure 7.5, the set X is the big rectangle and Y is the small one underneath X. The linear dynamics is a sink, similar to the example of Figure 7.4, and its trajectories spiral clockwise to the origin. The orthogonal polyhedra lying between the stair-like lines in the figures are the under-approximations of $\mathcal{U}_q(X,Y)$ and $\pi_q^\infty(X)$.

Plugging the approximate algorithm for π into Algorithm 13, we obtain a sequence $\{\widetilde{\mathcal{P}}^k\}$ of orthogonal polyhedra such that $\widetilde{\mathcal{P}}^k \subseteq \mathcal{P}^k$ for every k. The computed solution $\widetilde{\mathcal{P}}^*$ is guaranteed to be included in $\mathcal{P}^*$, and hence the restricted automaton $\widetilde{\mathcal{A}}^*$ with respect to $\widetilde{\mathcal{P}}^*$ satisfies the same properties as $\mathcal{A}^*$ except, of course, being maximal. This gives an effective solution to the controller synthesis problem for hybrid automata with linear continuous dynamics.

Recall that we have also developed a reachability technique for continuous dynamics of the form $\dot{\mathbf{x}} = A\mathbf{x} + \mathbf{u}$ where $\mathbf{u}$ is the input and takes values in a convex set U (see Section 4.5 of Chapter 4). Combining this with the computation procedure described above, one is able to solve the safety controller synthesis problem for systems with *continuous disturbances*, as the example in Section 7.6.2 will illustrate.

7.4 Uncontrollable Switching

In the framework described so far, we assumed that all discrete transitions are controllable, i.e. generated by the controller. However, in practice the environment in which physical systems work is often *uncontrollable* and can induce some of the switching in a non-deterministic way. For instance, human interaction (such as an operator pushing a button) or a discrete change in a physical process (such as a collision) can be modeled as uncontrolled transitions. In addition, such transitions are very useful to describe the passage from one region of the state space to another when piecewise-linear systems are used to approximate non-linear dynamics.

Here we discuss an extension of the synthesis algorithm to hybrid automata with uncontrollable switching. In this setting, it is important to distinguish non-determinism of the controller, which corresponds to the design choices, from non-determinism of the environment, which reflects our imprecise knowledge about the latter's actions. Therefore, the *transitions* of our systems are now labeled as *controllable* and *non-controllable*.

In the model to be considered we assume that the controller has no dominion over the actions of the environment, i.e. if an uncontrollable transition and a controllable one are enabled at a given state, the former has higher priority. The synthesis problem is formulated as finding a controller by *restricting the controllable actions* such that *the controlled system always stays within a given set $\mathcal{F}$ regardless of the environment's behavior.* We show now how to adapt the π operator to take uncontrollability into account.

When all transitions are controllable, in order to stay in $\mathcal{F}$ the controller's strategy might be to wait some time $t > 0$ and *then* take a transition. However, in the presence of uncontrollable transitions, one should consider the possibility that at some time $t' < t$ the environment might take a transition that will lead the system outside $\mathcal{F}$.

In order to incorporate uncontrollable transitions, we augment the model with a set $T^u \subseteq Q \times Q$ of uncontrollable transitions. We observe that if (q, q') is in T^u, then from any state $(q, \mathbf{x})$ such that $\mathbf{x} \in G_{qq'}$ the environment can enforce a transition to $(q', \mathbf{x})$, and if $(q', \mathbf{x}) \notin \mathcal{F}$ this will make the system violate the safety specification. From this observation, we modify the π operator as follows.

Let $\mathcal{F} = (q_1, F_1) \cup \ldots \cup (q_m, F_m)$. The computation of $\mathcal{F}' = \pi(\mathcal{F}) = (q_1, F_1') \cup \ldots \cup (q_m, F_m')$ is done in two steps:

1. Compute $\breve{\mathcal{F}} = (q_1, \breve{F}_1) \cup \ldots \cup (q_m, \breve{F}_m)$ by letting

$$\breve{F}_q = F_q \setminus \bigcup_{(q,q') \in T^u} G_{qq'} \cap \overline{F}_{q'}$$

   where $\overline{F}_{q'}$ is the complement of $F_{q'}$. By doing this, we remove from $\mathcal{F}$ all states $(q, \mathbf{x})$ from which the environment can lead the system to some $(q', \mathbf{x})$ outside $\mathcal{F}$.

2. For every $q \in Q$ compute F'_q as follows:

$$F'_q = \pi_q^\infty(\breve{F}_q) \cup \bigcup_{(q,q' \notin T^u)} \mathcal{U}_q(\breve{F}_q, G_{qq'} \cap \breve{F}_{q'}). \tag{7.4}$$

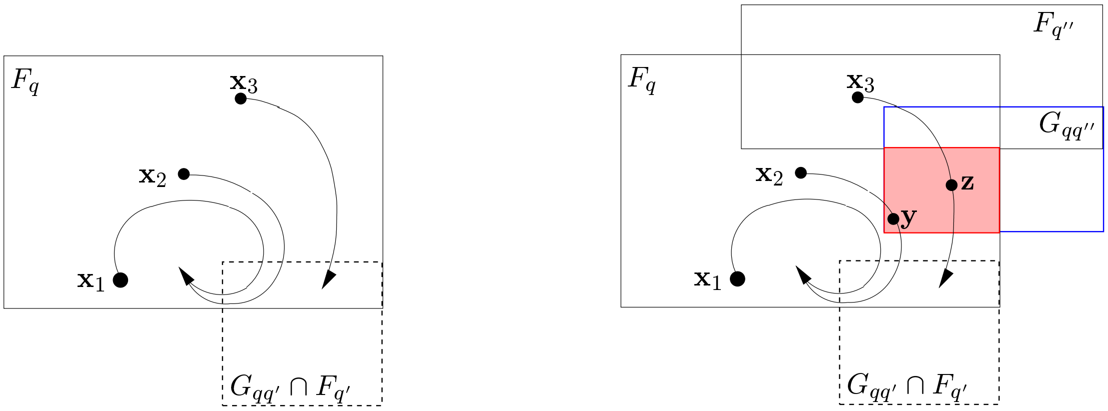


Figure 7.7: The operator π for systems with uncontrollable switching.

Figure 7.7 illustrates the above modification. In the left figure, (q, q') is the only outgoing transition from q, and this transition is controllable. It is easy to see that points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ are all in F'_q since the trajectories from them either stay invariantly in F_q or stay there until $G_{qq'} \cap F_{q'}$ is reached. We now add to the system an uncontrollable transition (q, q'') as shown in the right figure. The shaded rectangle consists of points inside F_q from which the environment can force a transiton to q'' and take the system outside $\mathcal{F}$. Hence $\breve{F}_q$ is the result of removing this rectangle from F_q. Consequently, points like $\mathbf{x}_2$ and $\mathbf{x}_3$ are no longer in F'_q.

With the characterization of the operator $\pi(\mathcal{F})$ given in (7.4), the extension of our synthesis algorithm to systems with uncontrollable switching is straightforward.

7.5 Anti-Zeno Synthesis

We have developed an effective synthesis algorithm for strongly non-Zeno hybrid automata, i.e. automata whose all state cycles are non-Zeno. Whenever the input automaton does not satisfy this condition, Algorithm 13 might produce wrong results where $\mathcal{P}^*$ contains states from which bad behaviors can be avoided only due to Zeno behaviors. To remedy this, we need to transform a-priori every hybrid automaton into a non-Zeno one.

An obvious method for doing this is to eliminate the Zeno cycles by reducing the guards of the transitions involved in the cycles. The problem with this approach is that there are infinitely many different ways to 'separate' the guards, and the results might depend on the choice we have made. For some choices of guard reduction, the maximal invariant set will be empty while for other choices there is a way to control the system. Making the right choice requires knowledge about the qualitative behavior of the continuous dynamics. As an alternative, we propose a systematic method for transforming any hybrid automaton into a strongly non-Zeno automaton at the price of increasing the dimensionality of the system by one, and adding non-identity resets to the transitions.

The idea is very simple: we force the automaton to spend a certain positive amount of time at every discrete state by adding a clock variable c (a variable with $\dot{c} = 1$ at any discrete state), resetting it to zero at every transition, and adding the condition $c \geq d_{qq'}$ ($d_{qq'}$ is a positive constant) to every transition guard $G_{qq'}$ (see Figure 7.8 for an example). In fact, it is sufficient to have one transition which resets c and one transition which is guarded by $c \geq d$ in any cycle of the automaton where $d > 0$. This idea is similar to the use of dwell-time switching logics [91] in switching control to suppress "chattering", i.e. very fast switching. Another well-known method to avoid this phenomenon is called hysteresis switching logic [91, 104].

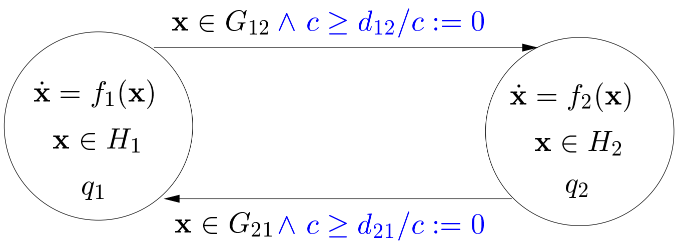


Figure 7.8: Transforming an automaton into a strongly non-Zeno one using a clock c.

This construction guarantees that the guards are separated and the augmented automaton is strongly non-Zeno. However, this solution clearly eliminates some behaviors that are possible in the original automaton. Putting positive lower-bounds on inter-transition times also adds some realism to the model of some physical systems which require some time to switch from one mode to another. The appropriate choice of $d_{qq'}$ might come from such realistic considerations or can be based on knowledge of the continuous dynamics. In any case, tuning $d_{qq'}$ seems to be much simpler than separating the guards of the original system. In addition to increasing the system dimensionality, the price of adding a clock is that the synthesis algorithm should be modified to account for clock resetting. The rest of this section is concerned with this modification.

We begin by giving a formal definition of hybrid automata augmented with a clock.

Definition 28 (Hybrid Automaton with Anti-Zeno Clock)
Let $\mathcal{A}^o = \{\mathcal{X}^o, Q^o, f^o, H^o, G^o, R^o\}$ be a hybrid automaton where $R^o_{qq'}$ are the identity for all $q, q' \in Q^o$. The automaton $\mathcal{A} = \{\mathcal{X}, Q, f, H, G, R\}$ constructed from $\mathcal{A}^o$ by adding a clock is defined as follows.

- *The continuous state space is* $\mathcal{X} = \mathcal{X}^o \times [0, \infty)$.
- *The discrete state space is* $Q = Q^o$.
- *The vector fields* $f_q = (f_q^o, 1)$ *for every* $q \in Q$.
- *The staying conditions* $H_q = H_q^o \times [0, \infty)$ *for every* $q \in Q$.
- *For all* $q,\ q' \in Q$, $G_{qq'} = G_{qq'}^o \times [d_{qq'}, \infty)$ *where* $d_{qq'} > 0$.
- *For all* $q,\ q' \in Q$, $R_{qq'}(x_1, \ldots, x_{n-1}, x_n) = (x_1, \ldots, x_{n-1}, 0)$. *In other words,* $R_{qq'}$ *leaves the first* $(n-1)$ *continuous variables intact and resets* x_n *(the clock) to* 0.

Every safety synthesis problem on $\mathcal{A}^o$ characterized by a set $\mathcal{F}^o = \{(q, F_q^o) \mid q \in Q\}$ is transformed to a problem on $\mathcal{A}$ with $\mathcal{F} = \{(q, F_q^o \times [0, \infty)) \mid q \in Q\}$.

Geometrically speaking, all the sets H_q, $G_{qq'}$, and F_q of the augmented automaton $\mathcal{A}$ are the prisms extending infinitely in the positive direction of the axis x_n from the corresponding sets of $\mathcal{A}^o$ (see Figure 7.9).

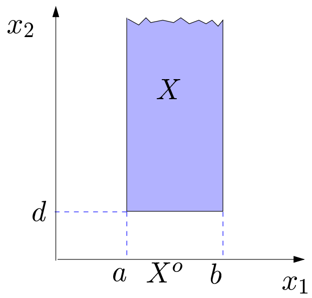


Figure 7.9: Construction of $X = X^o \times [d, \infty)$ where $X^o = [a, b]$ in one dimension (the jagged lines mean that the set extends infinitely).

To deal with the resets in the clock values, we modify slightly the π operator (the new definition holds for arbitrary resets and not only for those described in Definition 28).

Let $R_{qq'}^{-1} : 2^{\mathcal{X}} \to 2^{\mathcal{X}}$ be the inverse map of $R_{qq'}$ defined as

$$R_{qq'}^{-1}(X) = \{\mathbf{x}' \mid \exists \mathbf{x} \in X \ \mathbf{x} = R_{qq'}(\mathbf{x}')\}.$$

Then, the π operator can be modified to handle the resets as follows:

$$\pi(\mathcal{F}) = \{(q, \pi_q^\infty(F_q) \cup \bigcup_{q' \neq q} \mathcal{U}_q(F_q, G_{qq'} \cap R_{qq'}^{-1}(F_{q'}))) \mid q \in Q\}. \tag{7.5}$$

The modification is made to the second argument of the operator $\mathcal{U}_q$. Intuitively, we need to guarantee that the system stays in $\mathcal{F}$ after every reset $R_{qq'}$. The computation of the until operator is done backwards as described in Section 7.3 but this time starting with the initial set $G_{qq'} \cap R_{qq'}^{-1}(F_{q'})$. It remains now to compute the map $R_{qq'}^{-1}$.

Computing $R^{-1}_{qq'}$

The inverse reset map $R^{-1}_{qq'}$ for the hybrid automaton $\mathcal{A}$ of Definition 28 is characterized as

$$R^{-1}_{qq'}(X) = \{(x_1, ..., x_n) \mid x_n \geq 0 \ \wedge \ (x_1, .., x_{n-1}, 0) \in X\}.$$

The condition $x_n \geq 0$ is due to the fact that the clock can have only non-negative values. To compute $R^{-1}_{qq'}(X)$, we intersect the set X with the hyper-plane $P_0 = \{(x_1, ..., x_n) \mid x_n = 0\}$ and build over the resulting set a prism extending infinitely in the positive direction of the axis x_n (see Figure 7.10 for an example).

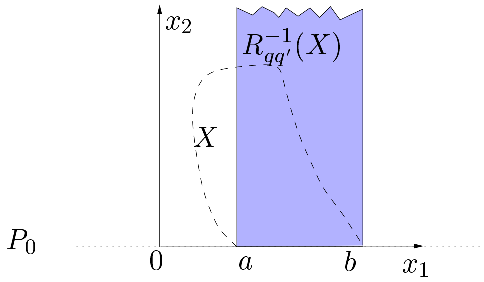


Figure 7.10: Computing $R^{-1}_{qq'}(X)$: the set X is drawn in dotted lines. First, we intersect X with P_0, which gives the line segment ab. Then, $R^{-1}_{qq'}(X)$ is the shaded prism with the base ab.

Before proceeding, we illustrate the computation of the set $Y = \mathcal{U}_q(F_q, G_{qq'} \cap R^{-1}_{qq'}(F_{q'}))$ with a simple example shown in Figure 7.11 where the dynamics f_q is constant. In the first step, we compute $R^{-1}_{qq'}(F_{q'})$ and then intersect it with the set $G_{qq'}$ (see Figure 7.11-(2)). Next, we compute Y as the set of continuous-predecessors of $G_{qq'} \cap R^{-1}_{qq'}(F_{q'})$ with F_q as staying conditions (see Figure 7.11-(3)).

Let us give an intuition behind this result. Indeed, the point $\mathbf{y}$ is not in Y because the trajectory from it stays in the set F_q for less than $d_{qq'}$ time. On the other hand, from any point in Y, such as $\mathbf{x}$, the system can reach a point $\mathbf{x}'$ in the guard $G_{qq'}$ after staying in F_q for at least $d_{qq'}$ time. From $\mathbf{x}'$ the system makes the transition to q' and, by clock resetting, jumps to a point $\mathbf{x}''$ in $F_{q'}$ as shown in Figure 7.11-(1).

Having computed an under-approximation $\widetilde{\mathcal{P}}^*$ of the maximal invariant set $\mathcal{P}^*$ of the augmented automaton $\mathcal{A}$, we restrict $\mathcal{A}$ to $\widetilde{\mathcal{P}}^*$ using Lemma 5 and derive a switching controller as in (7.3). The clock will be part of the controller, and the controller observes the state of the system and the value of the clock, switches according to the switching rule and resets the clock while doing so.

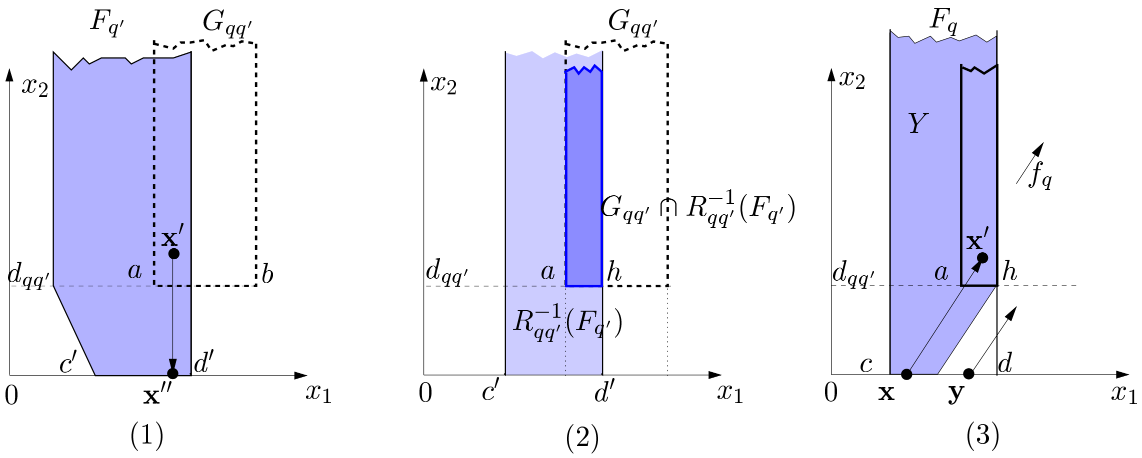


Figure 7.11: Computing $Y = \mathcal{U}_q(F_q, G_{qq'} \cap R^{-1}_{qq'}(F_{q'}))$ for an automaton with anti-Zeno clock: (1) the sets $F_{q'}$ (the shaded prism) and $G_{qq'}$ (the dotted prism based on $[a, b]$); (2) the set $R^{-1}_{qq'}(F_{q'})$ and its intersection with $G_{qq'}$ (the darker prism based on $[a, h]$); (3) the sets F_q (the prism based on $[c, d]$) and Y (the shaded region).

7.6 Examples

We have implemented the synthesis algorithms described above into *d/dt* and we now illustrate the behavior of the algorithms on two examples. Note that the results are obtained in a fully automatic manner once the model has been written.

7.6.1 Two spiral system

The first example is a system with two discrete states where the goal is to stay within a set $F = [-0.65, 0.35] \times [-0.35, 0.68]$. The dynamics are defined by

$$A_1 = \begin{pmatrix} 0.05 & -0.5 \\ 2.0 & 0.05 \end{pmatrix}, \qquad A_2 = \begin{pmatrix} 0.05 & -2.0 \\ 0.5 & 0.05 \end{pmatrix}$$

The continuous dynamics in both discrete states are characterized as 'sources' and their trajectories are diverging spirals. Therefore the only way to keep the system within F is to switch between two discrete states. The initial transition guards are:

$$G_{12} = [-0.2, -0.01] \times [-0.2, 0.01], \qquad G_{21} = [0.01, 0.32] \times [-0.01, 0.1].$$

One can see that the guards do not intersect with each other, and the system is thus strongly non-Zeno. The synthesis algorithm terminates after three iterations, and the running time is $75s$ (with run-time visualization) on a Sun Ultra Sparc-10. Figure 7.12 depicts the sets F_q and $G_{qq'} \cap F_{q'}$ obtained in each iteration (the latter set lies inside the former). These sets are used as initial sets to compute $\mathcal{U}_q$ in the next iteration (all the sets $\pi^\infty_q(F_q)$ are

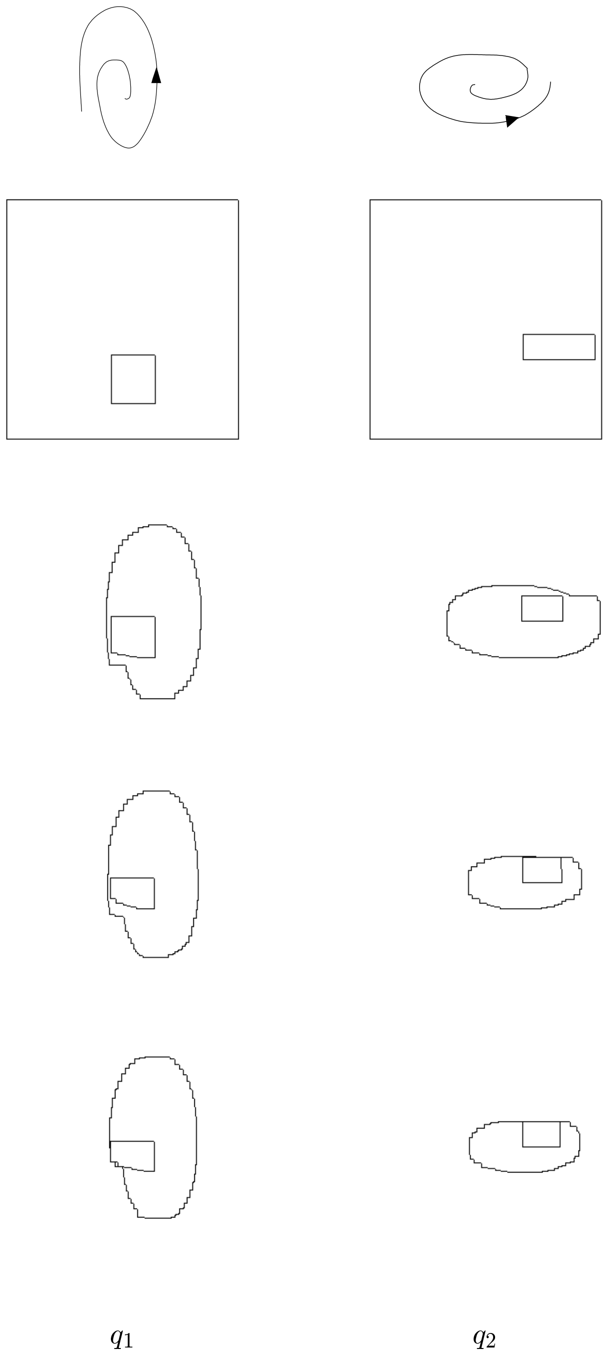


Figure 7.12: The phase portrait of the two spiral system and the evolution of F_1 and $G_{12} \cap F_2$ (left) and of F_2 and $G_{21} \cap F_1$ (right) within 3 iterations. The final results show for each discrete state the safe set where the system can spiral and then make a transition to the safe set of the other discrete state.

empty since the continuous dynamics are diverging). One can see from the figures that the safe sets become smaller after each iteration until they remain unchanged and the algorithm terminates.

7.6.2 Thermostat with Delay and Disturbances

Consider a thermostat with two modes (on and off). The continuous variable x_1 models the temperature and the input u models uncontrolled disturbances. The dynamics of the modes 'on' and 'off' are described by the differential equations

$$\dot{x_1} = -x_1 + u,$$

and

$$\dot{x_1} = -x_1 + 4 + u$$

where u ranges inside the interval $[-0.5, 0.5]$.

We augment the system with an additional clock variable x_2 as in Definition 28. We choose $d_{qq'} = 0.5$ for every transition, which means that the thermostat stays in each mode at least 0.5 time. We let the staying conditions and initial guards be the whole continuous state space. The hybrid automaton of the thermostat with delay and disturbances appears in Figure 7.13.

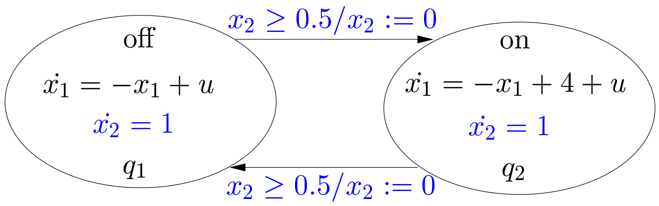


Figure 7.13: The hybrid automaton of the thermostat.

Our goal is to keep the temperature x_1 within the interval $[1.5, 3.6]$, and hence we start with $\mathcal{F} = \{q_1, q_2\} \times [1.5, 3.6] \times [0.5, \infty)$. The synthesis algorithm is performed on the augmented system and converges after three iterations. The results are shown in Figure 7.14 and the running time is $18s$ (with run-time visualization) on a Sun Ultra Sparc-10. By intersecting the two dimensional safe sets with $x_2 = 0$ we obtain the safe sets for x_1: $[2.48365, 3.5]$ at discrete state 'on' and $[1.5, 3.15736]$ at 'off'.

From the computed safe sets, we can define a deterministic switching controller which turns the thermostat on when $x_1 = \theta_{12} \wedge x_2 \geq 0.5$ and turns the thermostat off when $x_1 = \theta_{21} \wedge x_2 \geq 0.5$ for any θ_{12} and θ_{21} satisfying

$$2.48365 < \theta_{12} < \theta_{21} < 3.15736.$$

The automaton of the thermostat with the controller can be viewed in Figure 7.15.

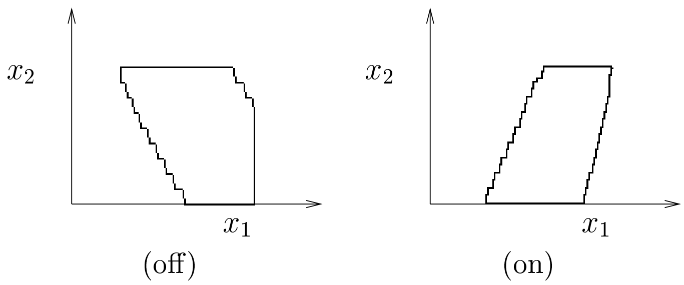


Figure 7.14: The safe sets of the thermostat.

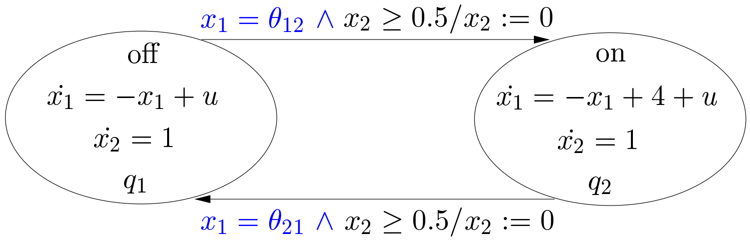


Figure 7.15: The hybrid automaton of the thermostat with the controller.

7.7 Summary and Related Work

In this chapter we have developed a simple framework for studying control by switching. We have proposed an effective algorithm for synthesizing switching controllers for hybrid automata with linear continuous dynamics subject to safety requirements, making use of the approximate reachability techniques. We have also presented an adaptation of the algorithm for systems with uncontrollable switching and a method to guarantee non-Zenoness of any synthesized system.

In the rest of this chapter we present some related work on controller synthesis for hybrid systems.

Controller synthesis for discrete systems is well-known in computer science (see, for example, the surveys in [15, 85]) as well as in control theory (supervisory control of discrete-event systems [102]). The first extension toward hybrid systems appeared in the work of Wong-Toi and Hoffmann [119], which can be characterized as indirect: they transform a timed automaton into a finite automaton (by using the finite partition of the state space, known as the region graph [5]). Algorithm 13, presented in this chapter, is based on the direct algorithm suggested in [85, 16] for timed automata. In that work, due to the special properties of timed automata, the *exact* computation of winning states and of a controller is guaranteed to terminate. These results were extended recently to the synthesis of *time-optimal* controllers for timed automata [13].

Another class of hybrid systems with simple continuous dynamics for which an exact controller synthesis algorithm always terminates are the initialized rectangular hybrid automata, studied by Henzinger et al. [60, 58]. For other classes of hybrid systems with constant derivatives, synthesis procedures are not guaranteed to terminate, although the π operator can be exactly computed by using linear algebra. An exact synthesis algorithm for 'linear' hybrid automata, which also considers non-Zenoness, was given in [118] and implemented in HyTech. Earlier work on controller synthesis for eventuality for such systems was reported in [110].

Outside the world of hybrid systems with trivial continuous dynamics, results have been hard to come by due to the difficulties discussed in Chapter 3. Recent work in [81, 112], using a game-theoretic approach, is very close in spirit to ours. The authors try to solve the controller synthesis problem for arbitrary continuous dynamics with time-varying piecewise continuous control and disturbance inputs, using an abstract algorithm similar to Algorithm 13. The π operator is characterized using Hamilton-Jacobi partial differential equations. Techniques to solve such equations were investigated in [112, 89]. However, numerical solutions can be complicated, and no evidence has been given so far of the computational advantages of this point of view. In [106] it has been shown that the synthesis problem for the sub-class of linear systems where the matrices are either diagonal or nilpotent is solvable by using computer algebra.

In addition, the synthesis problem has been studied in the discrete-event supervisory control framework by various authors (see [67, 76] and references from there). Since the continuous dynamics treated by these authors are non-trivial, they look for approximating automata [90, 36] rather than exact finite state abstraction as in [119]. In [67], the problem of extracting a discrete-event system from the continuous part of the system has been investigated. Once the DES has been extracted, discrete-event supervision techniques can be applied to synthesize a controller.

Part IV

Implementation

Chapter 8

The Tool *d/dt*

In this chapter we describe *d/dt* , an experimental tool for the verification and synthesis of hybrid systems. It is a C++ implementation of the algorithms presented in the previous chapters. We present first the modules of the implementation and then the main features of the tool.

8.1 Implementation

The modules of the implementation are summarized in Figure 8.1. The parts enclosed in the solid boxes are the modules that we implemented and the others are the software packages we use. The verification and synthesis algorithms are described in the previous chapters; here we present only the geometric manipulation, numerical integration and interface modules.

8.1.1 Geometric Algorithms

One important component of our verification and synthesis algorithms are procedures for manipulating convex and orthogonal polyhedra. Besides common geometric operations (Boolean operations, membership testing, etc..), for which we can use available software packages, some orthogonal approximation operations specific to our approach need to be implemented. We begin by presenting the data structures for polyhedra.

Data Structure for Convex and Orthogonal Polyhedra

Since we are interested in analyzing the system only in a bounded subset of the state space, called the analysis set, all polyhedra of interest are bounded. Every bounded convex polyhedron P can be represented either by the convex hull of a finite number of vertices or by the intersection of a finite number of half-spaces. Given either form, the other can be computed using standard algorithms. Since both forms are needed for different operations and,

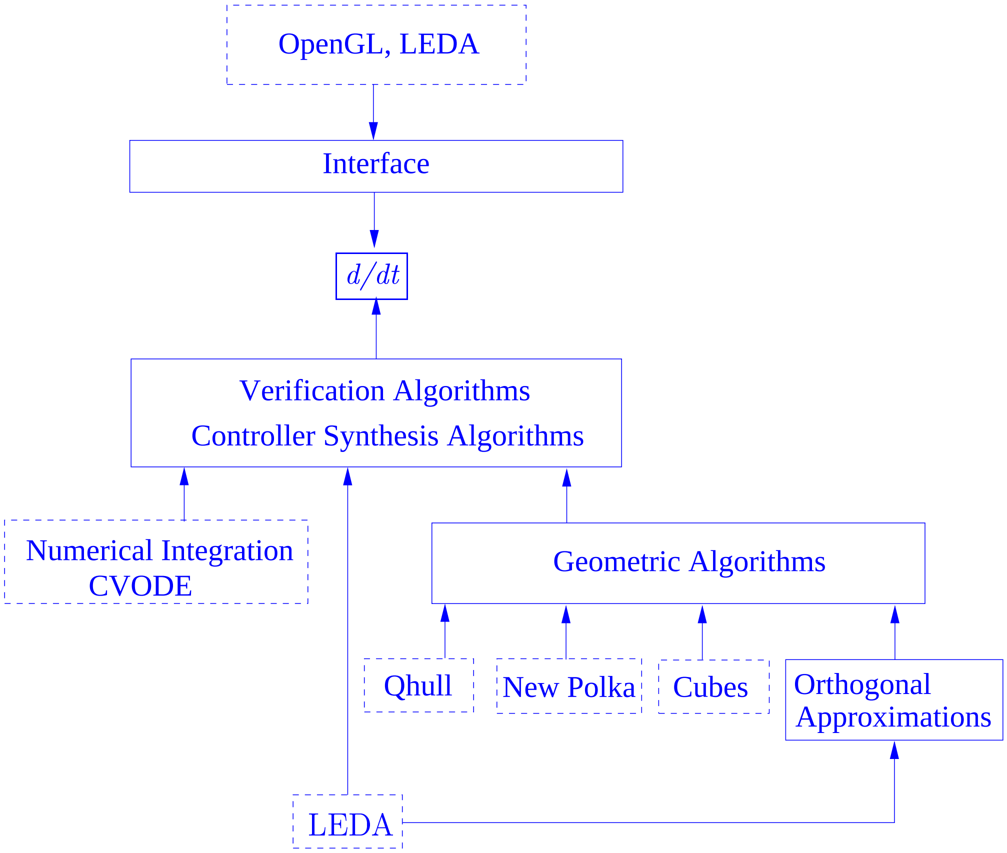


Figure 8.1: The modules of the tool.

moreover, the duality computation is expensive, our data structure for convex polyhedra maintains both forms, which means that the trade-off between memory and computation time was made in favor of the latter. An n-dimensional vertex is encoded as a real (or rational) vector, `vertex`, of size n whose elements are the coordinates of the vertex. An n-dimensional half-space is encoded as a real (or rational) vector, `halfspace`, of size $(n+1)$ whose the first n elements represent the normal to the half-space and the last one represents the offset, i.e. the distance from the half-space to the origin if the normal is an unit vector. Rationals are used to avoid precision problems in certain cases. The basic C-like data structure for a convex polyhedron is as follows.

```
Convex-polyhedron {
  int n, nbvertices, nbhalfspaces;
  vertex *V;
  halfspace *H;
}
```

Concerning orthogonal polyhedra, as we have already seen, a bounded orthogonal polyhedron can be represented by a finite number of *extreme vertices* (see Section 3.2 of Chapter 3). Thus, the data structure for orthogonal polyhedra is simply a list of extreme vertices as shown below.

```
Orthogonal-polyhedron {
  int n, nbvertices;
  vertex *V;
}
```

In all the implemented algorithms, basic data types (rational, array, dictionary, etc..) are provided by LEDA[1] library [88].

Convex and Orthogonal Polyhedron Operations

For operations on convex polyhedra, we use two libraries: Qhull [19] and a new implementation of Halbwachs's library [51]. Although Qhull does not provide ready polyhedral operations (and some auxiliary programs were thus implemented to fit the needs), the motivation of this choice is that convex hull is the most frequently used operator and the general dimensional convex-hull algorithm implemented in Qhull is one of the fastest available. In addition, Qhull provides an efficient algorithm for half-space intersection, a crucial element in our computations.

The 'new Polka' library implemented by B. Jeannet [63], using exact arithmetic, is much less time-efficient than Qhull, but its advantage is the ability to deal with degeneracies for which Qhull, using floating point arithmetic, may fail due to precision problems. In order to obtain a good compromise between time usage and accuracy, we combine these two libraries as follows: the algorithms of Qhull are used whenever possible and those of 'new Polka' for degenerate cases.

To manipulate orthogonal polyhedra, we use the library Cubes, implemented by O. Bournez [25], which provides algorithms for Boolean operations, inclusion test, convex decomposition, and face detection.

Orthogonal Approximations

Our verification and synthesis algorithms make use of four orthogonal approximation operators $grid_o$, $grid_u$, $\sqcap_o$, and $\sqcap_u$ to over- and under-approximate convex polyhedra and intersections of convex and orthogonal polyhedra. In the sequel we describe how these operators are implemented.

[1] Library of Efficient Data types and Algorithms.

The Operators $grid_o$ and $grid_u$

We describe first an algorithm for computing $grid_o$. For a clear understanding, let us recall the definition of this operator. Let $\mathcal{G}_\beta$ be the uniform underlying grid over which our orthogonal polyhedra are defined. Given a convex polyhedron C, $grid_o(C)$ is the *smallest* orthogonal polyhedron G defined on $\mathcal{G}_\beta$ such that $grid_o(C) \supseteq C$. We have shown that $grid_o(C)$ is the union of all the elementary hyper-cubes of the grid whose intersection with C is not empty.

Let V be the set of vertices of the polyhedron C. We denote for all $i \in \{1, \ldots, n\}$

$$l_i = \min\{\lfloor v_i \rfloor \beta \mid \mathbf{v} \in V\},\ u_i = \max\{(\lfloor v_i \rfloor + 1)\beta \mid \mathbf{v} \in V\}$$

where $\lfloor v_i \rfloor$ is the integer part of v_i/β. We define the bounding box Bb of C on the grid $\mathcal{G}_\beta$ as $Bb = [l_1, u_1] \times \ldots \times [l_n, u_n]$.

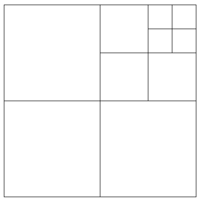


Figure 8.2: Refinement using the Binary Space Partition principle.

The key idea of the algorithm is the following. Using the principle of Binary Space Partition from computational geometry (see [44] for an introduction to this method), we split the bounding box Bb into sub-boxes and then refine recursively those sub-boxes which intersect C until they become elementary hyper-cubes of the grid (see Figure 8.3).

We denote by `bsp` the function that takes as input a box b and returns a list of 2^n sub-boxes. Note that the vertices of the sub-boxes must be grid points. Let `len`(b) be the maximal side length of the box b. The pseudo-code of the recursive refinement algorithm for computing $grid_o(C)$ is sketched below. The algorithm starts with the bounding box Bb of the input polyhedron C.

Algorithm 15 (Computing $grid_o(C)$)

```
refine(b: box) {
    if (b ∩ C ≠ ∅)  {
        if (len(b) ≤ β) G := G ∪ b;
        else {
            Lb := bsp(b);
            for all (b_i ∈ Lb) { refine(b_i); }
        }
    }
   return G;
}
```

Essentially, the refinement algorithm works as follows. If the box b intersects C, we distinguish the following two cases:

- If b is bigger than elementary hyper-cubes of the grid $\mathcal{G}_\beta$, then it is split into sub-boxes and each of these boxes is recursively refined.
- If b is an elementary hyper-cube of $\mathcal{G}_\beta$, then it is added to G.

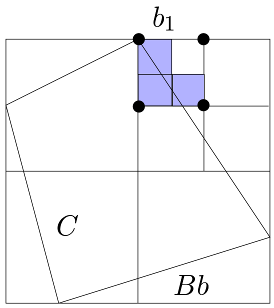


Figure 8.3: Illustration of the computation $grid_o(C)$.

As an example, consider a two-dimensional convex polyhedron shown in Figure 8.3. After successive refinements of the bounding box Bb we obtain the box b_1 whose sub-boxes become elementary hyper-cubes, and three shaded ones are added into G.

It is clear that efficient box-polyhedron intersection detection is crucial to the performance of the algorithm since the number of boxes encountered can be large. A straightforward way is to compute the intersection using standard convex polyhedron intersection algorithms. Nevertheless, the expense of finding geometric intersections makes it quite costly. To reduce the number of intersections we combine several tests exploiting the geometry of boxes, which are commonly used for interference detection in computer graphics [46, 77, 97].

The algorithm can be easily adapted to grids where a different constant β_i is used for every dimension and therefore elementary hyper-cubes become hyper-rectangles. Depending on the geometric form of the input polyhedron C, the use of such grids can improve significantly the performance of the algorithm.

We turn now to the operator $grid_u$. Given a convex polyhedron C, $grid_u(C)$ is the *largest* orthogonal polyhedron defined on the grid $\mathcal{G}_\beta$ that is included in C. It is not hard to see that $grid_u(C)$ can be computed in a similar way, but only the sub-boxes that are entirely inside C will be added to G. The algorithm for $grid_u$ is as follows.

Algorithm 16 (Computing $grid_u(C)$)

```
refine(b: box) {
    if (b ⊆ C) { G := G ∪ b; }
    else {
        if (b ∩ C ≠ ∅ ∧ len(b) > β) {
            Lb := bsp(b);
            for all (b_i ∈ Lb) { refine(b_i); }
        }
    }
    return G;
}
```

Algorithm 16 requires the additional inclusion test, which is trivial since testing whether a box is included in a convex polyhedron amounts to testing whether all the vertices of the former are inside the latter.

Remarks

- The approximation accuracy can be fine-tuned by defining a tolerance $\kappa \in \mathbb{N}$ and only the boxes whose size is greater than $\kappa\beta$ need to be refined. This can help to increase time-efficiency, but the approximations are no longer tight.

- An alternative method to compute orthogonal approximations is to use linear programming to find the largest hyper-rectangle inscribed in the polyhedron C and do it recursively [21]. This method is time costly in case the input polyhedron C is 'narrow', i.e. its bounding box has very different side lengths. The reason is that the volume of hyper-rectangles inscribed in C is small compared to that of C, which results in a large number of linear programming problems to solve. We have also implemented this method into *d/dt* , and the choice between this and the algorithm based on refinement is a user-defined parameter.

The Operators $\sqcap_o$ and $\sqcap_u$

We discuss only the computation of $\sqcap_o$ ($\sqcap_u$ is similar). Recall that given a convex polyhedron C and an orthogonal polyhedron G, $C \sqcap_o G$ is the smallest orthogonal polyhedron G_o defined on the grid of G such that $G_o \supseteq C \cap G$. The polyhedron G_o is thus the union of all the elementary hyper-cubes in G which intersect with C.

A naive implementation consists in testing all the elementary hyper-cubes in G. This can be very expensive in case G contains many hyper-cubes. A more efficient implementation consists in decomposing G into non-overlapping hyper-rectangles and then applying the refinement procedure, as is done for $grid_o$ and $grid_u$.

8.1.2 Numerical Integration

For numerical integration, we use CVODE [34], a software package for solving initial value problems for ordinary differential equations. The main attractive feature of CVODE is that it can deal with both stiff and non-stiff systems. CVODE implements two linear multistep methods, namely variable-coefficient Adams and BDF (Backward Differentiation Formula) [62]. The former is used for non-stiff problems and the latter for stiff ones. Both methods, being implicit, require solving non-linear systems, and to this end CVODE employs a variety of linear solvers and thus allows efficient solutions to a large class of problems.

8.1.3 Interface

Visualization is an increasingly important component in the design of a software package since it helps the user to easily interpret the results obtained. However, writing efficient 3D animation programs integrated into a computation tool is a highly professional and time-demanding task. Moreover, the computation time needed to obtain good visualization quality is considerable and sometimes exceeds the computation time of the reachability algorithms. The solution we adopt here is to develop only some simple OpenGL [116] programs (omitting advanced visualization features), which allow the user to display the results during the execution and provide an option to generate data in the input formats of other standard viewers. The interface programs, which manage the input and output as well as optional settings, are implemented using the Windows library of LEDA [88].

8.2 Functionalities

The current version of the tool handles hybrid automata in which

- Continuous dynamics are linear of the form $f(\mathbf{x}) = A\mathbf{x} + \mathbf{u}$ where $\mathbf{u}$ is the input and ranges inside a convex polyhedron.
- All the staying conditions and transition guards are specified as convex polyhedra.

In the current version of *d/dt* , the face lifting algorithm (which has been implemented separately) and the treatment of resets have not yet been integrated. Adding these features is straightforward and can be done without modifying the current modules.

We present first the input languague and then the functionalities of the tool.

8.2.1 Input Language

The input hybrid automaton and the specification are described in a model file (`.hyb`). The specification is given in form of a polyhedron which represents the bad set or the safe set. Parameters and formulas referring to them can be used to describe the system. The input language is simple and can be easily understood through an example. The textual description of a 2-state hybrid automaton is shown below.

`dimension` : 2; /* *dimension of the system* */

`parameters` :

$$\begin{array}{lcl} a0 & = & -0.32, \\ a1 & = & -0.28, \\ b0 & = & -0.1, \\ b1 & = & 0.0, \\ eps1 & = & 0.05, \\ eps2 & = & 0.02, \\ l2 & = & 0.29, \\ L1 & = & 0.52; \end{array}$$

`badset` : `type rectangle`

$$\begin{array}{rl} 0.62 & 0.67, \\ -0.1 & 0.1; \end{array}$$

`initloc` : 0;

`initset` : `type griddy`

$$\begin{array}{rl} -0.2 & 0.2, \\ 0.2 & 0.2, \\ -0.2 & 0.6, \\ 0.2 & 0.6; \end{array}$$

`location` : 0;

`matrixA` :

$$\begin{array}{ll} 0.0 & -6.0, \\ [-l2-(a0/(1+a0))*L1*cos(b0/(1+a0))] & 0.0; \end{array}$$

`inputset` : `type convex_vert`

$$0.0 \quad [(b0/(1+a0))*(-l2-(a0/(1+a0))*L1*cos(b0/(1+a0)))];$$

`stayset` : `type rectangle`

$$\begin{array}{rl} -0.15 & 1.0, \\ -1.0 & 1.0; \end{array}$$

`transition` :

`label` *to1* :

`if in guard` : `type rectangle`

$$\begin{array}{ll} [-(-eps1+(b0-b1))/(a0-a1)] & [-(eps1+(b0-b1))/(a0-a1)], \\ [eps2/(a0-a1)] & [-eps2/(a0-a1)]; \end{array}$$

`goto` 1;

`location` : 1;

`matrixA` :

```
  −2.0  −3.0,
   3.0  −2.0;
inputset : type convex_vert
  0.5  0.5, /* vertex (0.5, 0.5) */
  0.5  1.0,
  1.0  0.5;
stayset : type rectangle
  −1.0  −0.02,
  −1.0    1.0;
transition :
   label to0 :
     if in guard : type convex_halfsp
          1.0  2.0  −0.02, /* x[0] + 2x[1] ≤ −0.02 */
        −1.5  0.8  −1.5,
        −1.0  1.0    1.5,
        −1.0  0.5  −1.0;
    goto 0;
;

limits: /* the analysis set defined by a set of inequalities */
  x[0] >= −1.0   and
  x[0] <=   1.0   and
  x[1] >= −1.0   and
  x[1] <=   1.0
```

Convex polyhedra can be specified using two formats: a list of vertices (`convex_vert`) and a list of half-spaces (`convex_halfsp`). In case a polyhedron is a hyper-rectangle, it can be defined simply by intervals. While the staying and guard sets must be convex polyhedra, the initial set can be orthogonal. Orthogonal polyhedra are specified by lists of extreme vertices following the keyword `griddy`. The analysis set (`limits`) can be specified either as a conjunction of inequalities, like in the example, or as a convex polyhedron. The grammar of the input language is detailed in [38].

Computation Parameters

Since approximations are used in our algorithms, the tuning of some computation parameters can be useful to find a good compromise between computation time and accuracy. The tool allows for computation parameters to be defined textually in a parameter file (`.par`) or through the graphical interface. If they are not defined, default parameters will be used. Here we outline only some important parameters (see [38] for more details).

- Time step: the choice of the time step for each discrete state depends on the desired accuracy and also on the matrices of the continuous dynamics. It must be chosen according to Theorem 2 in Chapter 4.

- Grid size: errors in orthogonal approximations depend on the granularity of the grid. By reducing the grid size one can achieve better approximations at the price of more computation time.
- Convex-hull approximation option: this option is used only for reachability and verification purposes. In order to speed up the computation in the continuous phase, the user has an option to over-approximate orthogonal initial sets by their convex hull (see Section 6.3 of Chapter 6). This, however, reduces the approximation accuracy.

It is clear that prior knowledge of continuous dynamics will facilitate the fine-tuning of computation parameters in order to achieve the maximal computational efficiency.

8.2.2 Function Modes

An overview of the functionalities of the tool is shown in Figure 8.4. Given a model file and optionally a parameter file, the tool can work in the following three modes:

1. *Reachability:* this mode performs forward reachability analysis from the initial set. The output is an over-approximation of the reachable set.
2. *Safety Verification:* using forward reachability analysis, this mode can check whether the system starting from the initial set can reach the bad set. The output is a yes/no answer accompanied by a set of bad states that the system has reached, in case the answer is yes.
3. *Safety Controller Synthesis:* by computing an under-approximation of the maximal invariant set, this mode can synthesize a switching controller so that the system always remains inside the safe set. The output is the under-approximation of the maximal invariant set and the synthesized automaton.

The analysis results are stored in `.res` files in form of a sequence of sets of states (q, P) where q is a discrete state and P is a polyhedron.

8.2.3 Graphical User Interface

The goal of the graphical interface is to ease the use of the tool and facilitate interactive analysis. It consists of a menu bar and a window where the results are displayed. The menu bar has five sub-menus: `Input`, `Preferences`, `Run`, `View`, and `OOGL-Save`. The user selects the model file to work with through the `Input` menu and the type of analysis (reachability, verification, synthesis) to perform through the `Run` menu. During the analysis of a system, the main computation parameters can be changed via the `Preferences` menu. The tool offers the possibility to display the results from the output files (`.res`) using the `View` menu. This is useful when the user might wish to skip run-time visualization in order to reduce computation time. The `OOGL-Save` menu is used to transform results stored in output files

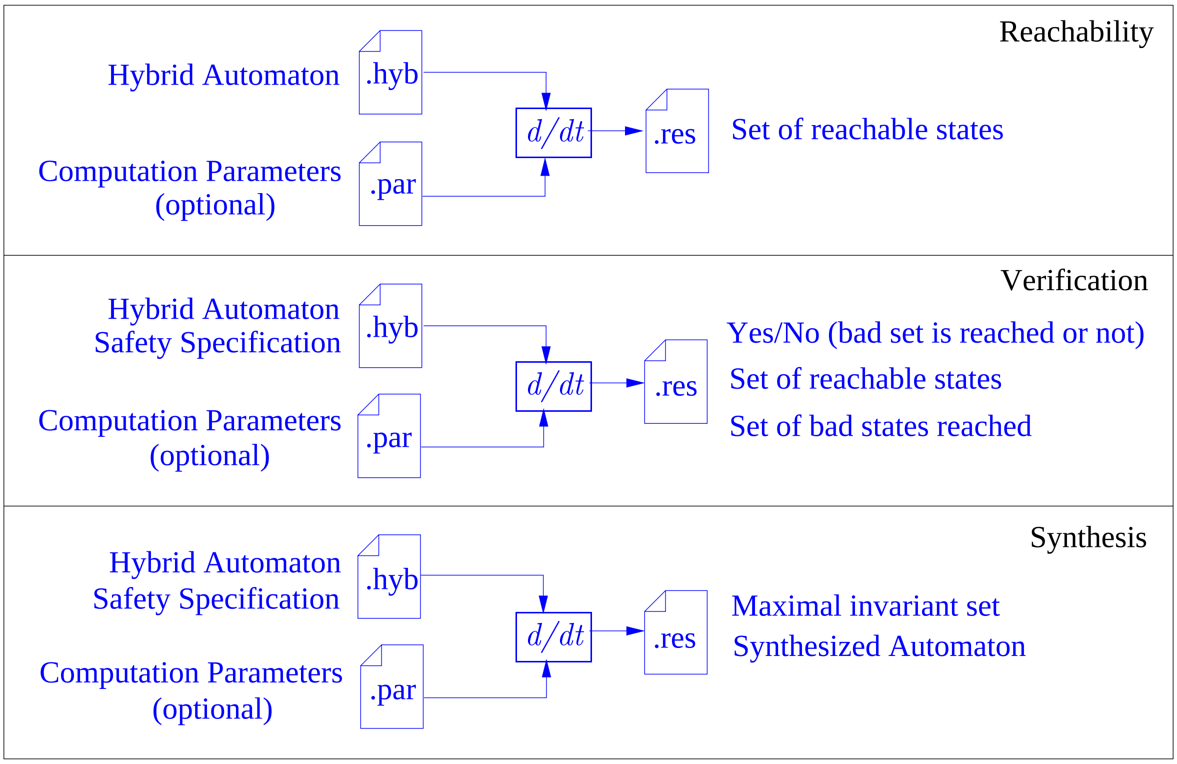


Figure 8.4: The functionalities of the tool.

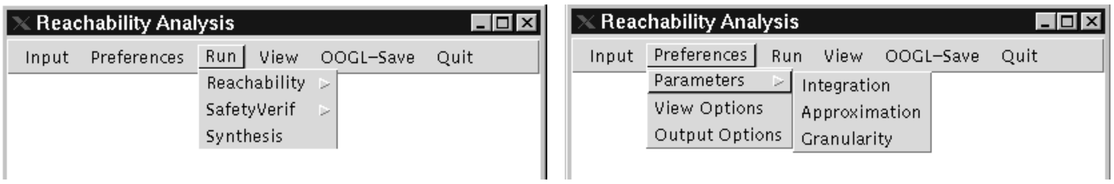


Figure 8.5: The menu bar of the tool.

into OOGL[2] format data, which can be then input to GeomView [96], an interactive viewer with many attractive 3D features. A snapshot of the graphical interface can be viewed in Figure 8.5.

8.3 Summary and Related Work

We have presented the overall structure of the experimental tool *d/dt* for automatically analyzing hybrid systems. We were able so far to treat rather easily continuous systems in up to 6 dimensions and recently a 3-dimensional hybrid system with 7 discrete states [9]. The tool is still under development, and there is an ongoing effort to add new features and to improve the implementation in order to increase its applicability. We have analyzed using *d/dt* many academic examples and several examples inspired by real-life applications. We are also currently investigating more applications in automotive control, robotics, and process control.

It is not easy to compare our tool to other tools in the domain for the following reasons. The hybrid systems research is still young, and hence it is not always possible to understand exactly the functionalities of some innovative tools from the papers which describe them. Moreover, due to the complexity of the problem and the approximate nature of the solution, it is still hard to define performance measures and to compare tools according to standard benchmarks. Here we discuss only the relationship between *d/dt* and three tools for the analysis of hybrid systems with non-trivial continuous dynamics, of which we are aware. The first difference between these tools and *d/dt* is that they do not support controller synthesis.

- *CheckMate* [31, 30]
  CheckMate is a verification tool, developed by Chutinan and Krogh, for threshold-event-driven hybrid systems where continuous dynamics are defined by general ODEs. The input system is modeled using Simulink block diagrams and then converted into a hybrid automaton with the following restrictions: all the guard sets $G_{qq'}$ lie on the boundary of the staying sets H_q, which are convex polyhedra; the reset maps are the identity, i.e. there is no jump in the continuous variables at discrete transitions. Unlike our tool, CheckMate takes the indirect approach, that is, it computes a finite-state abstraction of the original system using approximate reachability analysis and then applies standard verification algorithms to the resulting discrete model to verify ACTL specifications[3]. As mentioned in Chapter 4, the reachability algorithm used by CheckMate for linear continuous dynamics is similar to and potentially more efficient than ours, but it is not easy to extend to systems with uncertain input. One attractive feature of CheckMate is its interface for Matlab/Simulink, a commonly used formalism for specifying and simulating continuous and hybrid systems. The most complex example

[2] OOGL stands for Object Oriented Graphics Language.
[3] ACTL is a restriction of CTL (computation tree logic) [32] which allows only universal formulas.

reportedly treated using CheckMate is the batch evaporator, which is modeled as a hybrid automata with 3 continuous variables and 5 discrete states where continuous dynamics are non-linear.

- *HyperTech* [59]
  HyTech [56], developed by Henzinger, Ho and Wong-Toi, was the most popular tool for verifying systems with piecewise-constant continuous dynamics. HyperTech is an attempt to extend HyTech to systems with arbitrary differential equations. However, the tool treats only simple discrete dynamics, i.e. the resets are either arbitrary set-valued maps or the identity. The design philosophy of HyperTech is to use existing interval arithmetic packages in order to over-approximate reachable sets. Concretely, the typical computation step of HyperTech in the continuous phase starts with a hyper-rectangle F (a product of intervals) and uses the numerical integration of the interval arithmetic package to over-approximate the reachable states at time r by a hyper-rectangle F'. Then, the reachable set within the interval $[0, r]$, i.e. $\delta_{[0,r]}(F)$, is approximated by a hyper-rectangle F'' containing both F and F'. In [59] the authors do not detail how F'' can be guaranteed to be an over-approximation of $\delta_{[0,r]}(F)$. The reachable states accumulated over the execution are stored as a union of hyper-rectangles. The most complex example treated using HyperTech in [59] is an air-trafic conflict resolution system modeled as an automaton with 3 discrete states and 3 continuous variables whose dynamics are non-linear.

- *VeriShift* [24]
  VeriShift is a tool, developed by Botchkarev and Tripakis, for hybrid automata with linear differential inclusions. The basic hybrid automaton model treated by VeriShift is similar to ours, and the tool can accept systems of communicating hybrid automata. VeriShift is designed to perform bounded time verification. To over-approximate continuous-successors, VeriShift employs the ellipsoidal techniques, developed by Kurzhanski and Varaiya [72]. To treat discrete transitions, new methods for over-approximating unions of ellipsoids and intersections of ellipsoids and convex polyhedra are proposed. Input models should be written in C++ code, which is not always trivial for users not having a computer science background. A recent example treated by VeriShift is a train-gate system [23] which consists of three 4-state communicating automata sharing one continuous variable.

Note that in all the abovementioned tools reachable sets are represented in a non-canonical way (as unions of convex polyhedra/hyper-rectangles/ellipsoids), which limits their applicability to high dimensional systems. The tool *d/dt* has been designed with generality in mind, and hence the problem of representing polyhedra of arbitrary dimension has been tackled and solved before the development of the rest of the algorithms. Therefore, one positive feature of *d/dt* is that it extends easily to more general systems (in terms of the dimensionality and the complexity of dynamics).

Chapter 9

Conclusions

9.1 Contributions

Hybrid systems which combine continuous and discrete dynamics have been considered in this thesis. We have presented a practical framework for algorithmic analysis of hybrid systems, using the commonly accepted hybrid automaton model. The main contributions of the thesis are summarized as follows.

- *Formal Verification:*
  The lack of methods for computing reachable sets of continuous dynamics has been the main obstacle towards an algorithmic verification methodology for hybrid systems. This motivated us to tackle first the reachability problem of continuous systems. Unlike the conventional approaches which attempt to find exact solutions and are thus limited by undecidability of most non-trivial systems, our approach is based on an efficient method for representing sets and a combination of techniques from simulation, computational geometry, optimization, and optimal control. We have developed two effective approximate reachability techniques for continuous systems: one is specialized for linear systems and extended to systems with uncertain input; the other can be applied for general non-linear systems.

  Next, we have shown how these techniques can be adapted for hybrid systems and developed a safety verification algorithm which can work for a broad class of hybrid systems (with arbitrary continuous dynamics and rather general switching behavior). The main advantage of our verification algorithm over other existing algorithms is its easy application for high dimensional systems due to the canonical representation of reachable sets. In addition, with this representation our algorithm terminates in many cases while other algorithms do not.

- *Controller Synthesis:*
  We have considered the problem of synthesizing switching controllers for hybrid systems with respect to a safety specification. A safety specification is specified as a subset

of the state space within which the system must remain. We have presented an abstract synthesis algorithm based on the calculation of the maximal invariant set. The usefulness of this approach depends on the ability to effectively implement the π operator, the main ingredient of the synthesis algorithm. We have shown how our reachability techniques can be used for this purpose and provided an effective and automatic procedure for synthesizing controllers. Furthermore, we have extended this procedure to systems with uncontrolled switching coming from the environment. We have also proposed a simple method for ensuring non-Zenoness of any synthesized system.

- *Tool:*
  Another, not less important, goal of this thesis is to develop a working tool for analyzing hybrid systems. Many verification and synthesis algorithms have been proposed, but so far not many tools exist. We have implemented most of the algorithms presented in this thesis in the tool *d/dt* . The current version of the tool deals with hybrid systems with linear differential inclusions and provides automatic safety verification and controller synthesis. Some effort has been made to develop a graphical interface which helps the user to gain insight into the analysis and facilitates user intervention. Besides numerous academic examples used to evaluate the implementation, we have successfully applied the tool to verify some practical systems.

9.2 Future Research Directions

There are many promising research directions to pursue.

- *Formal Verification:*
  Much work can be done to improve the face lifting technique. The main drawback of this technique is the accumulation of over-approximation error. We have proposed a method to remedy this. However, further investigations should be made to devise a more clever approximation scheme by exploiting the qualitative behavior of the system. On the other hand, the current implementation of face lifting uses linked lists and matrices to encode orthogonal polyhedra. A new implementation using the canonical representation will increase significantly the efficiency of the algorithm.

  The performance of our verification algorithm can be improved in numerous ways. Experiments with many examples showed that most of the computation time is spent for geometric operations especially in high dimensions. We are currently exploring more efficient orthogonal approximation algorithms combining diverse techniques from computational geometry. Enhancing the implementation of some operations of the Cubes library is another way to make geometric manipulations more time-efficient.

  In addition, we have highlighted in Chapter 6 how search order can influence computation time, and we need thus to find the search strategies suitable for each problem instance. This could be done using qualitative reasoning. Since our verification algorithm can be readily used for simulation purposes, a method to reduce the search space is to define search order during the execution based on the information obtained by some simulations. In addition, prior simulation results can also suggest ways to do verification more efficiently.

In this thesis we have concentrated on the complexity of the system rather than of the properties. Methods for verifying more general temporal logic specifications are also a subject of future research.

- *Controller Synthesis:*
  We consider the following extensions of the synthesis results presented in this thesis:

  - Synthesis for eventuality: the dual synthesis problem of safety, eventuality, is concerned with finding the set of states from which the controller can enforce the system into a target set in finite time and finite number of switchings and compute the strategy for these states. Adapting our techniques for this performance criterion is rather straightforward.
  - Hybrid game automata: we have extended the synthesis algorithm to hybrid automata with uncontrollable switching which the controller cannot govern. The next extension is to more general hybrid games where the controller and the environment can have joint moves, as is done in [16] for timed automata. The synthesis of controllers in this setting can be solved at the price of adding a quantifier to the one-step predecessor operator.
  - Differential games: we believe that our techniques can be adapted to construct strategies for linear differential games of the form $\dot{\mathbf{x}} = A\mathbf{x} + B\mathbf{u} + C\mathbf{v}$ where $\mathbf{u}$ is a control input and $\mathbf{v}$ represents uncontrolled disturbance. One way to solve this problem is to discretize $\mathbf{u}$ and hence restrict the control to be piecewise-constant. This reduces the synthesis problem to the problem of mode switching solved in Chapter 7.

  More ambitiously, our synthesis algorithm could be generalized for systems with non-linear continuous dynamics. This requires a method for under-approximating reachable sets, and hence an extension of face lifting or a new technique needs to be investigated.

- *Tool:*
  The current version of the tool is not yet as general purpose as we would like. Many features can be added, such as the integration of the face lifting algorithm. As verification is often expensive, we are considering an extension of the tool to include the analysis in a 'simulation' fashion, that is, reachability is performed from only some subsets of the initial set. Although this analysis does not give a formal proof that the system is safe, it provides more reliable results than traditional simulation techniques. Besides the improvements on the algorithmic level, the graphical interface needs to be enhanced to allow more interactive analysis. This feature should not be underestimated since it facilitates better understanding of the behavior of the model and can serve for diagnostics purposes.

Clearly there is significant work that needs to be completed. Experimentation is not only a way to assess the methods and tools, but also a source of inspiration. The tool is currently under testing with examples taken from traffic control, engine control, robotics, and chemical process control, and more improvements can be made based on the accumulated experience. Recent results seem encouraging, and we feel hopeful that the techniques developed in this thesis will eventually be applied to real-life problems.

Bibliography

[1] G. Alefeld and J. Hezberger. *Introduction to Interval Computations*. Academic Press, New York, 1983.

[2] J.C. Alexander and T.I. Seidman. Sliding modes in intersecting switching surfaces. *Houston J. Math.*, 24:545–569, 1994.

[3] 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*, 138:3–34, 1995.

[4] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 209–229. Springer-Verlag, 1993.

[5] R. Alur and D.L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[6] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in CHARON. In N. Lynch and B.H. Krogh, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 6–19. Springer-Verlag, 2000.

[7] R. Alur, T.A. Henzinger, G. Lafferriere, and G. Pappas. Discrete abstractions of hybrid systems. In *Proc. of the IEEE*, 2000.

[8] P.J. Antsaklis, J.A. Stiver, and M.D. Lemmon. Hybrid system modeling and autonomous control systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 366–392. Springer-Verlag, 1993.

[9] E. Asarin, S. Bansal, B. Espiau, T. Dang, and O. Maler. On hybrid control of underactuated mechanical systems. In *Hybrid Systems: Computation and Control*, to appear in Lecture Notes in Computer Science. Springer-Verlag, 2001.

[10] 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*, Lecture Notes in Computer Science 1790, pages 20–31. Springer-Verlag, 2000.

[11] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, July, 2000.

[12] E. Asarin and R.N. Izmailov. Determining the sliding speed on a discontinuity surface. *Avtomatika Telemekhanika*, 9:1181–1185, 1989. transl., Automation and Remote Control 50.

[13] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In F. Vaandrage and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1569, pages 19–30. Springer-Verlag, 1999.

[14] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–66, 1995.

[15] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In P. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 1–20. Springer-Verlag, 1995.

[16] E. Asarin, O. Maler, A. Pnueli, and J. Sifakis. Controller synthesis for timed automata. In *IFAC Symposium on System Structure and Control*, pages 469–474. Elsevier, 1998.

[17] J.P. Aubin and A. Cellina. *Differential Inclusions: Set-valued Maps and Viability Theory*. Springer, 1984.

[18] A. Balluchi, L. Benvenuti, G.M. Miconi, U. Pozzi, T. Villa, M.D. Di Benedetto, H. Wong-Toi, and A. L. Sangiovanni-Vincentelli. Maximal safe set computation for idle speed control of an automative engine. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 32–44. Springer-Verlag, 2000.

[19] C.B. Barber, D.P. Dobkin, and H.T. Huhdanpaa. The quickhull algorithm for convex hulls. *ACM Transactions on Mathematical Software*, 22(4):469–483, 1996.

[20] G. Behrmann, T. Hune, and F.W. Vaandrager. Distributed timed model checking - how the search order matters. In *Computer Aided Verification, CAV'2000*, Lecture Notes in Computer Science. Springer-Verlag, 2000.

[21] A. Bemporad and F.D. Torrisi. Inner and outter approximation of polytopes using hyper-rectangles. Technical report, Automatic Control Lab, ETH, Zurich, 2000.

[22] N. Bjorner, A. Browne, E. Chang, A.M. Colon, A. Kapur, Z. Manna, H. Sipma, and T. Uribe. STeP: Deductive-algorithmic verification of reactive and real-time systems. In *Computer Aided Verification*, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[23] O. Botchkarev. Ellipsoidal techniques for verification of hybrid systems. Technical report, University of California in Berkeley, 2000. Available at: http://robotics.eecs.berkeley.edu/~olegb/VeriSHIFT.

[24] 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*, Lecture Notes in Computer Science 1790, pages 73–88. Springer-Verlag, 2000.

[25] O. Bournez. *Complexité algorithmique des systèmes dynamiques continus et hybrides.* PhD thesis, École Normale Supérieure de Lyon, Laboratoire de l'Informatique du Parallélisme, 1999.

[26] O. Bournez, O. Maler, and A. Pnueli. Orthogonal polyhedra: Representation and computation. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1569, pages 46–60. Springer-Verlag, 1999.

[27] M.S. Branicky. *Studies in Hybrid Systems: Modelling, Analysis, and Control.* PhD thesis, Massachusetts Intistute of Tech., 1995.

[28] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framwork for hybrid control. In *Proc. of the 33rd International Conference on Decision and Control*, pages 4228–4234, 1994.

[29] P. Caspi. Global simulation via partial differential equations. Unpublished note, Verimag, 1993.

[30] A. Chutinan. *Hybrid System Verification Using Discrete Model Approximations.* PhD thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, May 1999.

[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*, Lecture Notes in Computer Science 1569, pages 76–90. Springer-Verlag, 1999.

[32] E.M. Clarke, O. Grumberg, and D. Long. Verification tools for finite-state concurrent systems. In *Proceedings of A Decade of Concurrency: Reflections and Perspectives, REX School/Symposium*, pages 124–175. Springer-Verlag, June 1994.

[33] E.M. Clarke and J.M. Wing. Formal methods: State of the art and future directions. *ACM Computing Surveys*, 28:623–643, August 1996.

[34] S.D. Cohen and A.C. Hindmarsh. CVODE, a stiff/nonstiff ode solver in C. *Computers in Physics*, 10(2):138–143, 1996.

[35] P. Cousot and R. Cousot. Abstract interpretation and application to logic programs. *Journal of Logic Programming*, pages 103–179, 1992.

[36] J.E.R. Cury, B.H. Krogh, and T. Niinomi. Supervisory controllers for hybrid systems based on approximating automata. *IEEE Trans. on Automatic Control*, 43:564–568, 1998.

[37] D. Van Dalen. *Logic and Structure*. Springer-Verlag, 1994.

[38] T. Dang. d/dt manual. Technical report, Verimag, Grenoble, 2000.

[39] T. Dang and O. Maler. Reachability analysis via face lifting. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1386, pages 96–109. Springer-Verlag, 1998.

[40] C. Daws and S. Tripakis. Model checking of real-time reachability properties using abstractions. In B. Steffen, editor, *Tools and Algorithms for the Construction and Analysis of Systems, TACAS'98*, Lecture Notes in Computer Science 1384. Springer-Verlag, 1998.

[41] G.A. Edgar. *Measure, Topology, and Fractal Geometry*. Kluwer Springer, 1995.

[42] B. Espiau and C. Canudas de Wit. Sur la commande orbitale de certains systems mécaniques sous-actionés. Technical report, BIP, Inria, France, June 2000.

[43] A.F. Filippov. *Differential Equations with Discontinuous Righthand Sides*. Kluwer, 1988.

[44] H. Fuchs, Z. Kedem, and B. Naylor. On visible surface generation by a priori tree structures. In *Proc. of SIGGRAPH'80*, pages 124–133, 1980.

[45] A. Göllü and P. Varaiya. Hybrid dynamical systems. In *Proc. of the 28th International Conference on Decision and Control*, pages 2708–2712, 1989.

[46] N. Greene. *Graphics Gems IV*, chapter Detecting intersection of a rectangular solid and a convex polyhedron, pages 74–82. Academic Press, 1994.

[47] M.R. Greenstreet. Verifying safety properties of differential equations. In R. Alur and T.A. Henzinger, editors, *Computer Aided Verification, CAV'96*, Lecture Notes in Computer Science 1102, pages 277–287. Springer-Verlag, 1996.

[48] M.R. Greenstreet and I. Mitchell. Integrating projections. In T.A. Henzinger and S. Sastry, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1386, pages 159–1740. Springer-Verlag, 1998.

[49] M.R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In F. Vaandrager and J. van Schuppen, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1569, pages 76–90. Springer-Verlag, 1999.

[50] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, Lecture Notes in Computer Science 736. Springer-Verlag, 1993.

[51] N. Halbwachs. Delay analysis in synchronous programs. In C. Courcoubetis, editor, *Computer Aided Verification, CAV'93*, Lecture Notes in Computer Science 697, pages 333–346. Springer-Verlag, 1993.

[52] G. H. Hardy and E. M. Wright. *An Introduction to the Theory of Numbers*. Clarendon Press, 1979.

[53] P. Hartman. *Ordinary Differential Equations*. Wiley, 1964.

[54] T.A. Henzinger. Hybrid automata with finite bisimulations. In F. Vaandrager and J. van Schuppen, editors, *Proc. ICALP'95*, Lecture Notes in Computer Science 944, pages 324–335. Springer-Verlag, 1995.

[55] T.A. Henzinger and P.-H. Ho. A note on abstract-interpretation strategies for hybrid automata. In P. Antsaklis, A. Nerode, W. Kohn, and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999, pages 252–264. Springer-Verlag, 1995.

[56] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. HyTech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1:110–122, 1997.

[57] T.A. Henzinger, P.-H. Ho, and H. Wong-Toi. Analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[58] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In F. Vaandrager and J. van Schuppen, editors, *Concurrency Theory, CONCUR'99*, Lecture Notes in Computer Science 1664, pages 320–335. Springer-Verlag, 1999.

[59] T.A. Henzinger, B. Horowitz, R. Majumdar, and H. Wong-Toi. Beyond HyTech: Hybrid system analysis using interval numerical methods. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 130–144. Springer-Verlag, 2000.

[60] T.A. Henzinger and P.W. Kopke. Discrete-time control for rectangular hybrid automata. *Theoretical Computer Science*, 221:369–392, 1999.

[61] M.W. Hirsch and S. Smale. *Differential Equations, Dynamical Systems and Linear Algebra*. Academic Press, 1974.

[62] A. Iserles. *A first course in Numerical Analysis of Differential Equations*. Cambridge University Press, 1996.

[63] B. Jeannet. *Partitionnement dynamique dans l'analyse de relations linéaires et application à la vérification de programmes synchrones*. PhD thesis, Institut National Polytechnique de Grenoble, Laboratoire Verimag, 2000.

[64] R.E. Kalman, P.L. Falb, and M.A. Arbib. *Topics in Mathematical System Theory.* McGraw-Hill Book Company, 1968.

[65] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs, a class of decidable hybrid systems. In P.J. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors, *Proc. of Workshop on Theory of Hybrid Systems*, Lecture Notes in Computer Science 736, pages 179–208. Springer-Verlag, 1992.

[66] E.K. Kostoukova. State estimation for dynamic systems via parallelotopes: Optimization and parallel computations. *Optimization Methods and Software*, 9:269–306, 1999.

[67] X.D. Koutsoukos, P.J. Antsaklis, M.D. Lemmon, and J.A. Stiver. Supervisory control of hybrid systems. *Proc. of the IEEE*, 2000.

[68] S. Kowalewski, O. Stursberg, M. Fritz, H. Graf, J. Preußig, S. Simon, and H. Treseler. A case study in tool-aided analysis of discretely controlled continuous systems: The two-tanks-problem. In P.J. Antsaklis, W. Kohn, M. Lemmon, A. Nerode, and S. Sastry, editors, *Hybrid Systems V*, Lecture Notes in Computer Science 1567, pages 163–185. Springer-Verlag, 1999.

[69] S. Krishnan, A. Narkhede, and D. Manocha. Representation and computation of boolean combinations of sculptured models. In *Proc. 11th Annual ACM Symposium Computational Geometry*, pages C8–C9, 1995.

[70] R.P. Kurshan and K.L. McMillan. Analysis of digital circuits through symbolic reduction. *IEEE Trans. on Computer-Aided Design*, 10:1350–1371, 1991.

[71] A. Kurzhanski and I. Valyi. *Ellipsoidal Calculus for Estimation and Control.* Birkhauser, 1997.

[72] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 202–214. Springer-Verlag, 2000.

[73] G. Lafferriere, G. Pappas, and S. Yovine. Reachability computation for linear systems. In *Proc. of the 14th IFAC World Congress*, volume E, pages 7–12, 1999.

[74] K. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *Software Tools for Technology Transfert*, 1(1), 1997.

[75] M.D. Lemmon, K.X. He, and I. Markovsky. A tutorial introduction to supervisory hybrid system. Technical Report ISIS-98-004, ISIS Group, University of Notre Dame, October 1998.

[76] M.D. Lemmon, K.X. He, and I. Markovsky. Supervisory hybrid system. *IEEE Control Systems Magazine*, 19, 1999.

[77] M.C. Lin and D. Manocha. Fast interference detection between geometric models. *The Visual Computer*, 11(10), 1995.

[78] J. Lygeros, D.N. Godbole, and S. Sastry. Optimal control approach to multiagent, hierarchical system verification. In *Proc. of IFAC World Congress*, 1996.

[79] J. Lygeros, K.H. Johansson, M. Egerstedt, and S. Sastry. On the existence of executions of hybrid automata. In *Proc. of the 38th Annual International Conference on Decision and Control, CDC'99*. IEEE, 1999.

[80] J. Lygeros, C. Tomlin, and S. Sastry. Multiobjective hybrid controller synthesis. In O. Maler, editor, *Proc. International Workshop on Hybrid and Real-Time Systems*, Lecture Notes in Computer Science 1201, pages 109–123. Springer, 1997.

[81] J. Lygeros, C. Tomlin, and S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, 35, 1999.

[82] J. Macki and A. Strauss. *Introduction to Optimal Control Theory*. Springer, 1982.

[83] O. Maler. A unified approach for studying discrete and continuous dynamical systems. In *Proc. of the 37th Annual International Conference on Decision and Control, CDC'98*. IEEE, 1998.

[84] O. Maler, Z. Manna, and A. Pnueli. From timed to hybrid systems. In J.W. de Bakker, C. Huizing, W.P. de Roever, and G. Rozenberg, editors, *Real-Time: Theory in Practice*, Lecture Notes in Computer Science 600, pages 447–484. Springer-Verlag, 1992.

[85] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In E.W. Mayr and C. Puech, editors, *Proc. STACS'95*, Lecture Notes in Computer Science 900, pages 229–242. Springer-Verlag, 1995.

[86] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer, 1995.

[87] K.L. McMillan. *Symbolic Model Checking*. Kluwer Academic, 1993.

[88] K. Mehlhorn and S. Naher. *The LEDA Platform of Combinatory and Geometric Computing*. Cambridge University Press, 1999.

[89] 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*, Lecture Notes in Computer Science 1790, pages 311–323. Springer-Verlag, 2000.

[90] T. Moor and J. Raisch. Discrete control of switched linear systems. In *Proc. ECC'99*, 1999.

[91] A.S. Morse. Control using logic-based switching. In A. Isidori, editor, *Trends in Control - An European Perspective*, pages 69–113. Springer-Verlag, New York, 1995.

[92] A. Nerode and W. Kohn. Models for hybrid systems. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 317–356. Springer-Verlag, 1993.

[93] X. Nicolin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid automata. In R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors, *Hybrid Systems*, Lecture Notes in Computer Science 736, pages 149–178. Springer-Verlag, 1993.

[94] 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*, Lecture Notes in Computer Science 1569, pages 29–31. Springer-Verlag, 1999.

[95] N.B.O.L. Pettit. *Analysis of Piecewise Linear Dynamical Systems*. Research Studies Press LTD, 1995.

[96] M. Phillips, S. Levy, and T. Munzner. Geomview: An interactive geometry viewer. Notices of the American Mathematical Society, October 1993.

[97] F.P. Preparata and M.I. Shamos. *Computational Geometry*. Springer-Verlag, 1985.

[98] A. Puri, V. Borkar, and P. Varaiya. ϵ-approximation of differential inclusions. In T.A. Henzinger R. Alur and E.D. Sontag, editors, *Hybrid Systems III*, Lecture Notes in Computer Science 1066, pages 362–376. Springer-Verlag, 1996.

[99] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In *Computer Aided Verification, CAV'94*, Lecture Notes in Computer Science 816, pages 54–104. Springer-Verlag, 1994.

[100] A. Puri and P. Varaiya. Driving safely in smart cars. Technical Report UBC-ITS-PRR-95-24, California PATH, University of California in Berkeley, July 1995.

[101] A. Puri and P. Varaiya. Verification of hybrid systems using abstraction. In A. Nerode P. Antsaklis, W. Kohn and S. Sastry, editors, *Hybrid Systems II*, Lecture Notes in Computer Science 999. Springer-Verlag, 1995.

[102] P.J. Ramadge and W.M. Wonham. The control of discrete event systems. *Proc. of the IEEE*, 77, 1989.

[103] H.J. Samet. *Design and Analysis of Spatial Data Structure: Quadtree, Octree, and Other Hierachical Methods*. Addison-Wesley, 1989.

[104] A.V.D. Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Lecture Notes in Control and Information Sciences 251. Springer, 1999.

[105] J.A. Sethian. *Level Set Methods : Evolving Interfaces in Geometry, Fluid Mechanics, Computer Vision, and Materials Science*. Cambridge, 1996.

[106] O. Shakernia, G.J. Pappas, and S. Sastry. Decidable controller synthesis for classes of linear systems. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 407–420. Springer-Verlag, 2000.

[107] O. Stursberg and S. Kowalewski. Approximating switched continuous systems by rectangular automata. In *Proc. ECC'99*, 1999.

[108] O. Stursberg, S. Kowalewski, and S. Engell. Generating timed discrete models of continuous systems. In *Proc. 2nd IMACS Symposium on Mathematical Modelling, MATHMOD'97*, pages 203–209, 1997.

[109] O. Stursberg, S. Kowalewski, and S. Engell. On the generation of timed approximations for continuous systems. *Mathematical and Computer Modelling of Dynamical Systems*, 6(1):51–70, 2000.

[110] M. Tittus and B. Egardt. Controllability and control-law synthesis of linear hybrid systems. In G. Cohen and J.-P. Quadrat, editors, *Proc. International Conference on Analysis and Optimization of Systems*, Lecture Notes in Computer Science 199, pages 377–383. Springer-Verlag, 1994.

[111] C. Tomlin, J. Lygeros, and S. Sastry. Conflict resolution for air traffic management: A study in multi-agent hybrid systems. *IEEE Trans. on Automatic Control*, 43(4), 1998.

[112] C. Tomlin, J. Lygeros, and S. Sastry. A game-theoretic approach to controller design for hybrid systems. *Proc. of the IEEE*, 2000.

[113] V.I. Utkin. *Sliding Modes and their Application in Variable Structure Systems.* Mir, Moscow, 1978.

[114] P. Varaiya. Reach set computation using optimal control. In *Proc. KIT Workshop, Verimag, Grenoble*, pages 377–383, 1998.

[115] V. Veliov. On the time-discretization of control systems. *SIAM journal on Control and Optimization*, 35(5):1470–1486, 1997.

[116] C. Walnum. *3D Graphics Programming with OpenGL.* Que Corp, 1995.

[117] K. Weiler. *Topological Structures for Geometric Modeling.* PhD thesis, Dept. Comput. Syst. Engr., Rensselaer Polytechnic Inst., Troy, 1986.

[118] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proc. of the 36th International Conference on Decision and Control, CDC'97*, 1997.

[119] H. Wong-Toi and G. Hoffmann. The control of dense real-time discrete event systems. Technical report, Stanford University, 1992. Technical report STAN-CS-92-1411.

[120] S. Yovine. Kronos: A verification tool for real-time systems. *Software Tools for Technology Transfer*, 1(1):123–133, 1997.

[121] J. Zhang, K.H. Johansson, and S. Sastry. Dynamical systems revisited: Hybrid systems with zeno executions. In B. Krogh and N. Lynch, editors, *Hybrid Systems: Computation and Control*, Lecture Notes in Computer Science 1790, pages 451–464. Springer-Verlag, 2000.

[122] F. Zhao. *Automatic Analysis and Synthesis of Controllers for Dynamical Systems Based on Phase-space Knowledge*. PhD thesis, MIT, Artificial Intelligence Laboratory, August 1992.

RÉSUMÉ : Les systèmes hybrides sont des systèmes qui combinent des dynamiques discrètes et continues. Cette thèse propose des techniques algorithmiques de vérification et de synthèse pour ces systèmes. Le manque de méthodes pour calculer les ensembles atteignables par des dynamiques continues est l'obstacle principal vers une méthodologie algorithmique de vérification. Nous développons deux techniques d'atteignabilité approximatives pour les systèmes continus basées sur une méthode efficace pour représenter des ensembles et une combinaison des techniques de la simulation, de la géométrie algorithmique, de l'optimisation et de la commande optimale. La première technique est spécialisée pour les systèmes linéaires et étendue aux systèmes avec entrée incertaine. La seconde peut être appliquée aux systèmes non-linéaires. En appliquant ces techniques nous développons un algorithme de vérification des propriétés de sûreté pour des systèmes hybrides avec des dynamiques continues et discrètes générales. Nous étudions ensuite le problème de la synthèse de contrôleurs de sûreté pour les systèmes hybrides. Nous présentons un algorithme de synthèse des contrôleurs par commutation basé sur le calcul de l'ensemble d'invariance maximal et les techniques d'analyse d'atteignabilité. Nous décrivons l'outil *d/dt* qui permets la vérification et la synthèse automatique pour les systèmes hybrides avec des inclusions différentielles linéaires. Nous avons appliqué avec succès l'outil pour analyser quelques systèmes pratiques.

ABSTRACT: This thesis proposes a practical framework for the verification and synthesis of hybrid systems, that is, systems combining continuous and discrete dynamics. The lack of methods for computing reachable sets of continuous dynamics has been the main obstacle towards an algorithmic verification methodology for hybrid systems. We develop two effective approximate *reachability techniques* for *continuous systems* based on an efficient representation of sets and a combination of techniques from simulation, computational geometry, optimization, and optimal control. One is specialized for *linear* systems and extended to systems with *uncertain input*, and the other can be applied for *non-linear* systems. Using these reachability techniques we develop a *safety verification algorithm* which can work for a broad class of hybrid systems with arbitrary continuous dynamics and rather general switching behavior. We next study the problem of synthesizing *switching controllers* for hybrid systems with respect to a *safety property.* We present an effective synthesis algorithm based on the calculation of the maximal invariant set and the approximate reachability techniques. Finally, we describe the *tool d/dt* which provides automatic safety verification and controller synthesis for hybrid systems with linear differential inclusions. Besides numerous academic examples, we have successfully applied the tool to verify some practical systems.
Keywords: Hybrid automata, differential equations, reachability analysis, formal verification, controller synthesis.

MOTS-CLÉS : Automates hybrides, équations différentielles, analyse d'atteignabilité, vérification formelle, synthèse de contrôleurs.

SPÉCIALITÉ : Automatique, Productique
VERIMAG, 2, avenue de Vignate, 38610 Gières, France, http://www-verimag.imag.fr