Lecture Notes on Hybrid Systems

John Lygeros

Department of Electrical and Computer Engineering
University of Patras
Rio, Patras, GR-26500, Greece
lygeros@ee.upatras.gr

ENSIETA
2-6/2/2004

Abstract

The aim of this course is to introduce some fundamental concepts from the area of hybrid systems, that is dynamical systems that involve the interaction of continuous (real valued) states and discrete (finite valued) states. Applications where these types of dynamics play a prominent role will be highlighted. We will introduce general methods for investigating properties such as existence of solutions, reachability and decidability of hybrid systems. The methods will be demonstrated on the motivating applications. Students who successfully complete the course should be able to appreciate the diversity of phenomena that arise in hybrid systems and how discrete "discrete" entities and concepts such as automata, decidability and bisimulation can coexist with continuous entities and concepts, such as differential equations.

Contents

List of Figures

Chapter 1

Dynamical Systems: an Overview

1.1 Notation & Background

- $\mathbb{R}^n$ denotes the n-dimensional Euclidean space. This is a finite dimensional **vector space** (also known as a linear space). If $n = 1$, we will drop the superscript and write just $\mathbb{R}$ (the set of real numbers or "the real line"). I will make no distinction between vectors and real numbers in the notation (no arrows over the letters, bold fond, etc.). Both vectors and real numbers will be denoted by lower case letters.
- $\|x\| = \sqrt{x_1^2 + x_2^2 + \ldots + x_n^2}$ denotes the standard (Euclidean) **norm** in $\mathbb{R}^n$.
- $\mathbb{Z}$ denotes the set of integers, $\ldots, -2, -1, 0, 1, 2, \ldots$.
- $x \in A$ is a shorthand for "x belongs to a set A", e.g. $x \in \mathbb{R}^n$ means that x is an n-dimensional vector.
- Given a set X, $P(X)$ denotes the **power set** of X, i.e. the set of all subsets of X. In other words, $A \in P(X)$ means that $A \subseteq X$. By definition, $X \in P(X)$ for all sets X.
- $\emptyset$ denotes the **empty set** (a set containing nothing). By definition $\emptyset \in P(X)$ for all sets X.

  Exercise 1.1 Consider a set containing only 3 elements, say $Q = \{q_1, q_2, q_3\}$. Write down all the elements of $P(Q)$. There should be 8 of them. In mathematics, the power set, $P(X)$, of a set X is sometimes denoted by 2^X. Can you guess the reason for this?

- $f(\cdot) : A \to B$ is a shorthand for a function mapping every element $x \in A$ to an element $f(x) \in B$. For example the function $\sin(\cdot) : \mathbb{R} \to \mathbb{R}$ maps a real number x to its sine, $\sin(x)$.
- In logic
  - $\forall$ is a shorthand for "for all", as in "$\forall x \in \mathbb{R}, x^2 \geq 0$".
  - $\exists$ is a shorthand for "there exists", as in "$\exists x \in \mathbb{R}$ such that $\sin(x) = 0$".
  - $\wedge$ is a shorthand for "and", $\vee$ stands for "or", and $\neg$ stands for "not".
- Logic expressions can be used to define sets by enumerating properties of their elements. For example, the following expression defines a subset of $\mathbb{R}^2$

  $$\{x \in \mathbb{R}^2 \mid (x_1^2 + x_2^2 = 1) \wedge (x_1 \geq 0) \wedge (x_2 \leq 0)\},$$

  namely the part of the unit circle that falls in the 4^{th} quadrant.
- ∞ denotes "infinity".

- Given two real numbers $a \leq b$,

$$[a, b] = \{x \in \mathbb{R} \mid a \leq x \leq b\}$$

denotes the **closed interval** from a to b, while

$$[a, b) = \{x \in \mathbb{R} \mid a \leq x < b\}$$

denotes the **right-open** interval from a to b. Notice that if $a = b$, then $[a, b] = [a, a] = \{a\}$, whereas $[a, b) = [a, a) = \emptyset$. We also define $[a, \infty)$ as the set of all real numbers greater than or equal to a.

- Given two sets Q and X, $Q \times X$ denotes the **product** of the two sets. This is the set of all pairs (q, x) with $q \in Q$ and $x \in X$, i.e.

$$Q \times X = \{(q, x) \mid q \in Q \text{ and } x \in X\}$$

I will assume that people are somewhat familiar with the concepts of vector space, state space, differential, etc. A brief review of these topics will be given in Chapter 2

Lecture notes will be self contained but, by necessity, terse. I recommend that students consult the references to fully appreciate the subject. A number of exercises have been added throughout the notes, to enhance the learning experience and highlight subtle points. They should all be easy (if not trivial!) to solve if you understand the notes. **You are not required to turn in solutions for the exercises in the notes**, they are there to help you understand the material, not to test your understanding. You are of course welcome to discuss the solutions with the instructor.

1.2 Dynamical System Classification

Roughly speaking, a **dynamical system** describes the **evolution** of a **state** over **time**. To make this notion more precise we need to specify what we mean by the terms "evolution", "state" and "time".

Certain dynamical systems can also be influenced by external inputs, which may represent either uncontrollable disturbances (e.g. wind affecting the motion of an aircraft) or control signals (e.g. the commands of the pilot to the aircraft control surfaces and engines). Some dynamical systems may also have outputs, which may represent either quantities that can be measured, or quantities that need to be regulated. Dynamical systems with inputs and outputs are sometimes referred to as **control systems**.

Based on the type of their state, dynamical systems can be classified into:

1. **Continuous**, if the state takes values in Euclidean space $\mathbb{R}^n$ for some $n \geq 1$. We will use $x \in \mathbb{R}^n$ to denote the state of a continuous dynamical system.
2. **Discrete**, if the state takes values in a countable or finite set $\{q_1, q_2, \ldots\}$. We will use q to denote the state of a discrete system. For example, a light switch is a dynamical system whose state takes on two values, $q \in \{ON, OFF\}$. A computer is also a dynamical system whose state takes on a finite (albeit very large) number of values.
3. **Hybrid**, if part of the state takes values in $\mathbb{R}^n$ while another part takes values in a finite set. For example, the closed loop system we obtain when we use a computer to control an inverted pendulum is hybrid: part of the state (namely the state of the pendulum) is continuous, while another part (namely the state of the computer) is discrete.

Based on the set of times over which the state evolves, dynamical systems can be classified as:

1. **Continuous time**, if the set of times is a subset of the real line. We will use $t \in \mathbb{R}$ to denote continuous time. Typically, the evolution of the state of a continuous time system is described by an **ordinary differential equation** (ODE). Think of the linear, continuous time system in state space form
$$\dot{x} = Ax.$$
2. **Discrete time**, if the set of times is a subset of the integers. We will use $k \in \mathbb{Z}$ to denote discrete time. Typically, the evolution of the state of a discrete time system is described by a **difference equation**. Think of the linear discrete time system in state space form
$$x_{k+1} = Ax_k.$$
3. **Hybrid time**, when the evolution is over continuous time but there are also discrete "instants" where something "special" happens. More on this in Chapter 3.

Continuous state systems can be further classified according to the equations used to describe the evolution of their state

1. **Linear**, if the evolution is governed by a linear differential equation (continuous time) or difference equation (discrete time).
2. **Nonlinear**, if the evolution is governed by a nonlinear differential equation (continuous time) or difference equation (discrete time).

Exercise 1.2 The linear vs nonlinear classification generally does not apply to discrete state or hybrid systems. Why?

In this class we will start by giving some examples of the the following classes of systems:

1. Nonlinear (continuous state), continuous time systems.
2. Nonlinear, discrete time systems.
3. Discrete state, discrete time systems.

We will then concentrate on hybrid state, hybrid time systems and highlight the differences from the other classes. Classes of systems that will not be treated at all include:

- Infinite dimensional continuous state systems described, for example, by partial differential equations (PDE).
- Discrete state systems with an infinite number of states, e.g. Petri nets, push down automata, Turing machines.
- Stochastic systems, i.e. systems with probabilistic dynamics.

1.3 Examples

1.3.1 Pendulum: A Nonlinear, Continuous Time System

Consider a pendulum hanging from a weight-less solid rod and moving under gravity (Figure 1.1). Let θ denote the angle the pendulum makes with the downward vertical, l the length of the pendulum, m its mass, and d the dissipation constant. The evolution of θ is governed by
$$ml\ddot{\theta} + dl\dot{\theta} + mg\sin(\theta) = 0$$
This is a nonlinear, second order, ordinary differential equation (ODE).

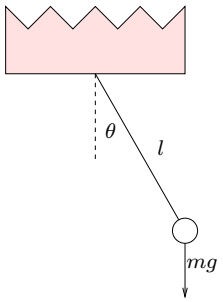

Figure 1.1: The pendulum

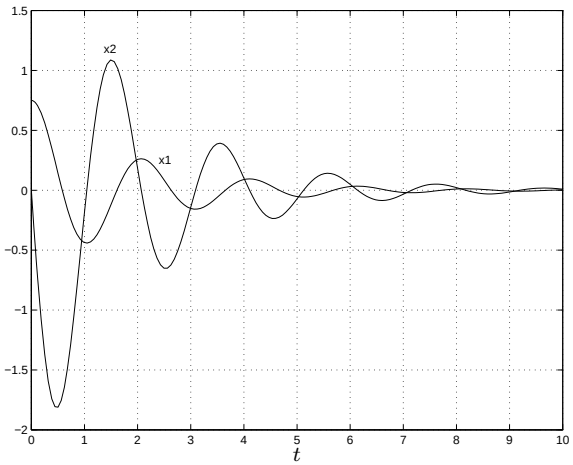

Figure 1.2: Trajectory of the pendulum.

Exercise 1.3 Derive this equation from Newton's laws. Why is this ODE called nonlinear?

To determine how the pendulum is going to move, i.e. determine θ as a function of time, we would like to find a solution to this ODE . Assuming that at time $t = 0$ the pendulum starts as some initial position θ_0 and with some initial velocity $\dot{\theta}_0$, "solving the ODE" means finding a function of time

$$\theta(\cdot) : \mathbb{R} \to \mathbb{R}$$

such that

$$\begin{aligned} \theta(0) &= \theta_0 \\ \dot{\theta}(0) &= \dot{\theta}_0 \\ ml\ddot{\theta}(t) + dl\dot{\theta}(t) + mg\sin(\theta(t)) &= 0,\ \forall t \in \mathbb{R} \end{aligned}$$

Such a function is known as a **trajectory** (or **solution**) of the system. At this stage it is unclear if one, none or multiple trajectories exist for this initial condition. **Existence** and **uniqueness** of trajectories are both desirable properties for ODE that are used to model physical systems.

For nonlinear systems, even if a unique trajectory exists for the given initial condition, it is usually difficult to construct explicitly. Frequently solutions of ODE can only be approximated by **simulation**. Figure 1.2 shows a simulated trajectory of the pendulum for $l = 1$, $m = 1$, $d = 1$, $g = 9.8$, $\theta(0) = 0.75$ and $\dot{\theta}(0) = 0$.

To simplify the notation we typically write dynamical system ODE in **state space** form

$$\dot{x} = f(x)$$

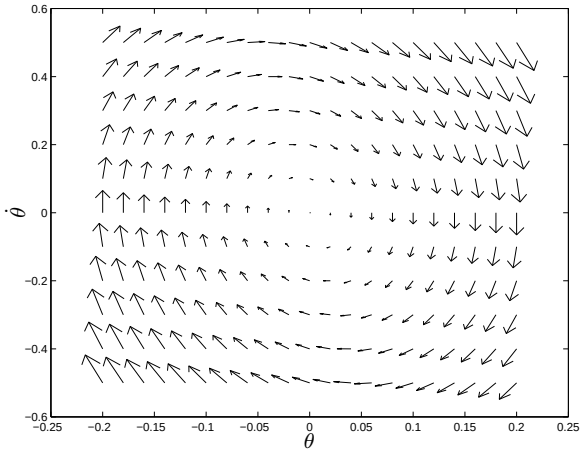

Figure 1.3: The pendulum vector field.

where x is now a vector in $\mathbb{R}^n$ for some appropriate $n \geq 1$. The easiest way to do this for the pendulum is to set

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} \theta \\ \dot{\theta} \end{bmatrix}$$

which gives rise to the state space equations

$$\dot{x} = \begin{bmatrix} \dot{x}_1 \\ \dot{x}_2 \end{bmatrix} = \begin{bmatrix} x_2 \\ -\frac{g}{l}\sin(x_1) - \frac{d}{m}x_2 \end{bmatrix} = f(x)$$

The vector

$$x \in \mathbb{R}^2$$

is called the **state** of the system. The size of the state vector (in this case $n = 2$) is called the **dimension** of the system. Notice that the dimension is the same as the order of the original ODE. The function

$$f(\cdot) : \mathbb{R}^2 \to \mathbb{R}^2$$

which describes the dynamics is called a **vector field**, because is assigns a "velocity" vector to each state vector. Figure 1.3 shows the vector field of the pendulum.

Exercise 1.4 Other choices are possible for the state vector. For example, for the pendulum one can use $x_1 = \theta^3 + \dot{\theta}$ and $x_2 = \dot{\theta}$. What would the vector field be for this choice of state?

Solving the ODE for θ is equivalent to finding a function

$$x(\cdot) : \mathbb{R} \to \mathbb{R}^2$$

such that

$$x(0) = \begin{bmatrix} x_1(0) \\ x_2(0) \end{bmatrix} = \begin{bmatrix} \theta_0 \\ \dot{\theta}_0 \end{bmatrix}$$
$$\dot{x}(t) = f(x(t)),\ \forall t \in \mathbb{R}.$$

For two dimensional systems like the pendulum it is very convenient to visualise the solutions by **phase plane** plots. These are plots of $x_1(t)$ vs $x_2(t)$ parameterised by time (Figure 1.4).

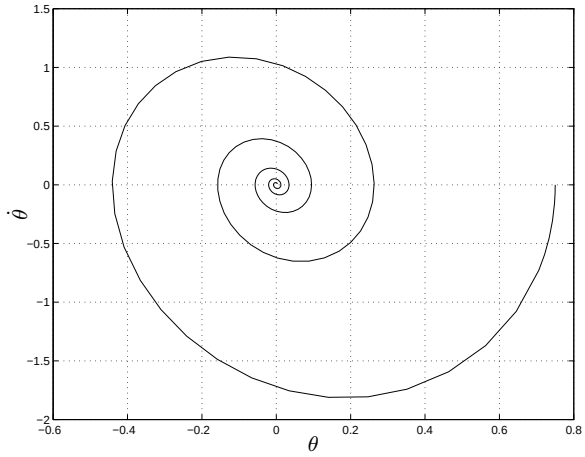

Figure 1.4: Phase plane plot of the trajectory of Figure 1.2.

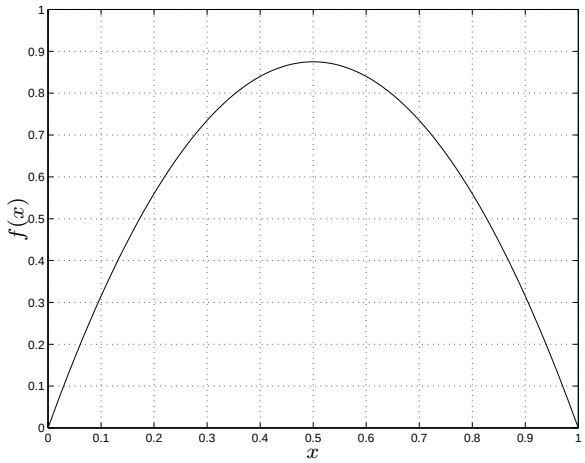

Figure 1.5: The logistic map.

1.3.2 Logistic Map: A Nonlinear Discrete Time System

The logistic map

$$x_{k+1} = ax_k(1 - x_k) = f(x_k) \tag{1.1}$$

is a nonlinear, discrete time dynamical system that has been proposed as a model for the fluctuations in the population of fruit flies in a closed container with constant food supply [78]. We assume that the population is measured at discrete times (e.g. generations) and that it is large enough to be assumed to be a continuous variable. In the terminology of the previous section, this is a one dimensional system with state $x_k \in \mathbb{R}$, whose evolution is governed by the difference equation (1.1) given above.

The shape of the function f (Figure 1.5) reflects the fact that when the population is small it tends to increase due to abundance of food and living space, whereas when the population is large it tends to decrease, due to competition for food and the increased likelihood of epidemics. Assume that $a \leq 4$ and that the initial population is such that $0 \leq x_0 \leq 1$.

Exercise 1.5 Show that under these assumptions $0 \leq x_k \leq 1$ for all $k \in \mathbb{Z}$ with $k \geq 0$.

The behaviour of x_k as a function of k depends on the value of a.

1. If $0 \leq a < 1$, x_k decays to 0 for all initial conditions $x_0 \in [0, 1]$. This corresponds to a situation where there is inadequate food supply to support the population.
2. If $1 \leq a \leq 3$, x_k tends to a steady state value. In this case the population eventually stabilises.
3. If $3 < a \leq 1 + \sqrt{6} = 3.449$, x_k tends to a 2-periodic state. This corresponds to the population alternating between two values from one generation to the next.

As a increases further more and more complicated patterns are obtained: 4-periodic points, 3-periodic points, and even chaotic situations, where the trajectory of x_k is a-periodic (i.e. never meets itself).

1.3.3 Manufacturing Machine: A Discrete System

Consider a machine in a manufacturing plant that processes parts of type p one at a time. The machine can be in one of three states: Idle (I), Working (W) or Down (D). The machine can transition between the states depending on certain events. For example, if the machine is idle and a part p arrives it will start working. While the machine is working it may break down. While the machine is down it may be repaired, etc.

Abstractly, such a machine can be modelled as a dynamical system with a **discrete state**, q, taking three values

$$q \in Q = \{I, W, D\}$$

The state "jumps" from one value to another whenever one of the **events**, σ occurs, where

$$\sigma \in \Sigma = \{p, c, f, r\}$$

(p for "part arrives", c for "complete processing", f for "failure" and r for "repair"). The state after the event occurs is given by a **transition relation**

$$\delta : Q \times \Sigma \to Q$$

Since both Q and Σ are finite sets, one can specify δ by enumeration.

$$\begin{aligned} \delta(I, p) &= W \\ \delta(W, c) &= I \\ \delta(W, f) &= D \\ \delta(D, r) &= I \end{aligned}$$

δ is undefined for the rest of the combinations of q and σ. This reflects the fact that certain events may be impossible in certain states. For example, it is impossible for the machine to start processing a part while it is down, hence $\delta(D, p)$ is undefined.

Exercise 1.6 If the discrete state can take n values and there are m possible events, what is the maximum number of lines one may have to write down to specify δ?

Such a dynamical system is called an **automaton**, or a **finite state machine**. Automata are special cases of **discrete event systems**. Discrete event systems are dynamical systems whose state also jumps depending on a finite set of events but can take on an infinite number of values. The dynamics of a finite state machine can be represented compactly by a **directed graph** (Figure 1.6). This is a graph whose **nodes** represent the possible values of the state (in this case I, W, D). The **arcs** of the graph represent possible transitions between the state values and are labelled by the events.

Exercise 1.7 What is the relation between the number of arcs of the graph and the number of lines one needs to write down in order to specify δ?

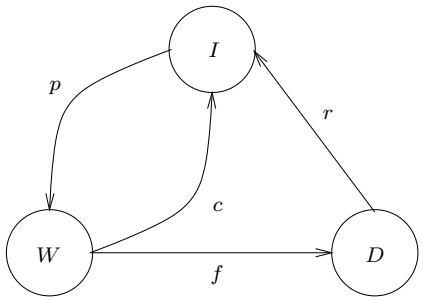

Figure 1.6: The directed graph of the manufacturing machine automaton.

Assume that the machine starts in the idle state $q_0 = I$. What are the sequences of events the machine can experience? Clearly some sequences are possible while others are not. For example, the sequence *pcp* is possible: the machine successfully processes one part and subsequently starts processing a second one. The sequence *ppc*, on the other hand is not possible: the machine can not start processing a second part before the previous one is complete. More generally, any sequence that consists of an arbitrary number of *pc*'s (possibly followed by a single *p*) is an acceptable sequence. In the discrete event literature this set of sequences is compactly denoted as

$$(pc)^*(1+p)$$

where * denotes an arbitrary number (possibly zero) of *pc*'s, 1 denotes the empty sequence (no event takes place), and + denotes "or".

Likewise, pfr is a possible sequence of events (the machine starts processing a part, breaks down and then gets repaired) while pfp is not (the machine can not start processing a part while it is down). More generally, any sequence that consists of an arbitrary number of pfr's (possibly followed by a p or a pf) is an acceptable sequence.

Exercise 1.8 Write this set of sequences in the discrete event notation given above.

The set of all sequences that the automaton can experience is called the **language** of the automaton. The above discussion suggests that the language of the machine automaton is

$$(pc+pfr)^*(1+p+pf)$$

It is important to understand the properties of these languages, for example to determine how to schedule the work in a manufacturing plant.

1.3.4 Thermostat: A Hybrid System

Consider a room being heated by a radiator controlled by a thermostat. Assume that when the radiator is off the temperature, $x \in \mathbb{R}$, of the room decreases exponentially towards 0 degrees according to the differential equation

$$\dot{x} = -ax \tag{1.2}$$

for some $a > 0$.

Exercise 1.9 Verify that the trajectories of (1.2) decrease to 0 exponentially.

When the thermostat turns the heater on the temperature increases exponentially towards 30 degrees, according to the differential equation

$$\dot{x} = -a(x-30). \tag{1.3}$$

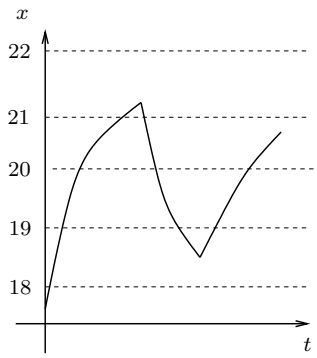

Figure 1.7: A trajectory of the thermostat system.

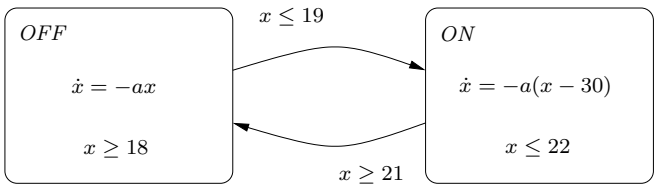

Figure 1.8: Directed graph notation for the thermostat system.

Exercise 1.10 Verify that the trajectories of (1.3) increase towards 30 exponentially.

Assume that the thermostat is trying to keep the temperature at around 20 degrees. To avoid "chattering" (i.e. switching the radiator on an off all the time) the thermostat does not attempt to turn the heater on until the temperature falls below 19 degrees. Due to some uncertainty in the radiator dynamics, the temperature may fall further, to 18 degrees, before the room starts getting heated. Likewise, the thermostat does not attempt to turn the heater on until the temperature rises above 21 degrees. Due to some uncertainty in the radiator dynamics the temperature may rise further, to 22 degrees, before the room starts to cool down. A trajectory of the thermostat system is shown in Figure 1.7. Notice that in this case multiple trajectories may be obtained for the same initial conditions, as for certain values of the temperature there is a choice between switching the radiator on/off or not. Systems for which such a choice exists are known as **non-deterministic**.

Notice that this system has both a continuous and a discrete state. The continuous state is the temperature in the room $x \in \mathbb{R}$. The discrete state, $q \in \{ON, OFF\}$ reflects whether the radiator is on or off. The evolution of x is governed by a differential equation (as was the case with the pendulum), while the evolution of q is through jumps (as was the case with the manufacturing machine). The evolution of the two types of state is coupled. When $q = ON$, x rises according to differential equation (1.3), while when $q = OFF$, x decays according to differential equation (1.2). Likewise, q *can not jump* from ON to OFF unless $x \geq 21$. q *must jump* from ON to OFF if $x \geq 22$. Etc.

It is very convenient to compactly describe such **hybrid systems** by mixing the differential equation with the directed graph notation (Figure 1.8).

1.4 Bibliography and Further Reading

Continuous state systems and their properties have been studied extensively in mathematics engineering, economics, biology, etc. The literature is vast and there are a number of excellent textbooks available (see for example [100, 103, 51, 88]).

Discrete state systems have also been studied for many years, especially in computer science. Good textbooks include [43, 58, 25].

By comparison, the study of hybrid systems is relatively recent. The few books that have appeared on the subject to date [77, 98] have a research monograph "flavour" and address specific topics and classes of systems. [98] contains a substantial textbook style overview. Another good source of material are the special issues devoted by a number of journals on the topic of hybrid systems [85, 9, 91, 90, 84].

Chapter 2

Review of Continuous Systems

2.1 State Space Form

All continuous nonlinear systems considered in this class can be reduced to the standard **state space** form. It is usual to denote

- the **states** of the system by $x_i \in \mathbb{R}$, $i = 1, \ldots, n$,
- the **inputs** by $u_j \in \mathbb{R}$, $j = 1, \ldots, m$, and
- the **outputs** by $y_k \in \mathbb{R}$, $k = 1, \ldots, p$.

The number of states, n, is called the **dimension** (or **order**) of the system. The evolution of the states, inputs and outputs is governed by a set of functions

$$f_i : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}, \text{ for } i = 1, \ldots, n$$
$$h_j : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}, \text{ for } j = 1, \ldots, p$$

Roughly speaking, at a given time $t \in \mathbb{R}$ and for given values of all the states and inputs these functions determine in what direction the state will move, and what the output is going to be.

$$\begin{aligned}
\dot{x}_1 &= f_1(x_1, \ldots, x_n, u_1, \ldots, u_m, t) \\
&\vdots \\
\dot{x}_n &= f_n(x_1, \ldots, x_n, u_1, \ldots, u_m, t) \\
y_1 &= h_1(x_1, \ldots, x_n, u_1, \ldots, u_m, t) \\
&\vdots \\
y_p &= h_p(x_1, \ldots, x_n, u_1, \ldots, u_m, t)
\end{aligned}$$

Exercise 2.1 What is the dimension of the pendulum example? What are the functions f_i?

It is usually convenient to simplify the equations somewhat by introducing vector notation. Let

$$x = \begin{bmatrix} x_1 \\ \vdots \\ x_n \end{bmatrix} \in \mathbb{R}^n, \; u = \begin{bmatrix} u_1 \\ \vdots \\ u_m \end{bmatrix} \in \mathbb{R}^m, \; y = \begin{bmatrix} y_1 \\ \vdots \\ y_p \end{bmatrix} \in \mathbb{R}^p,$$

and define

$$f : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^n$$
$$h : \mathbb{R}^n \times \mathbb{R}^m \times \mathbb{R} \to \mathbb{R}^p$$

by

$$f(x,u,t) = \begin{bmatrix} f_1(x_1,\ldots,x_n,u_1,\ldots,u_m,t) \\ \vdots \\ f_n(x_1,\ldots,x_n,u_1,\ldots,u_m,t) \end{bmatrix}, \; h(x,u,t) = \begin{bmatrix} h_1(x_1,\ldots,x_n,u_1,\ldots,u_m,t) \\ \vdots \\ h_p(x_1,\ldots,x_n,u_1,\ldots,u_m,t) \end{bmatrix}.$$

Then the system equations simplify to

$$\left.\begin{aligned} \dot{x} &= f(x,u,t) \\ y &= h(x,u,t) \end{aligned}\right\} \tag{2.1}$$

Equations (2.1) are known as the **state space form** of the system. The vector space $\mathbb{R}^n$ in which the state of the system takes values is known as the **state space** of the system. If the system is of dimension 2, the state space is also referred to as the **phase plane**. The function f that determines the direction in which the state will move is known as the **vector field**.

Notice that the differential equation for x is first order, i.e. involves $\dot{x}$ but no higher derivatives of x. Sometimes the system dynamics are given to us in the form of higher order differential equations, i.e. equations involving a variable $\theta \in \mathbb{R}$ and its derivatives with respect to time up to $\frac{d^r\theta}{dt^r}$ for some integer $r \geq 1$. Such systems can be easily transformed to state space form by setting $x_1 = \theta$, $x_2 = \dot{\theta}$, $\ldots$, $x_{r-1} = \frac{d^{r-1}\theta}{dt^{r-1}}$.

Exercise 2.2 Consider the system

$$\frac{d^r\theta}{dt^r} + g(\theta, \frac{d\theta}{dt}, \ldots, \frac{d^{r-1}\theta}{dt^{r-1}}) = 0$$

Write this system in state space form.

It may of course happen in certain examples that there are no inputs or outputs, or that there is no explicit dependence of the dynamics on time. Systems of the form

$$\dot{x} = f(x)$$

(i.e. without inputs or outputs and with no explicit dependence on time) are called **autonomous** systems.

Exercise 2.3 Is the pendulum an autonomous system?

Exercise 2.4 Consider a non-autonomous system of the form $\dot{x} = f(x,t)$, of dimension n. Show that it can be transformed to an autonomous system of dimension $n+1$. (Hint: append t to the state).

2.2 Existence and Uniqueness of Solutions

Consider an autonomous dynamical system in state space form

$$\dot{x} = f(x)$$

and assume that at time $t = 0$ the state is equal to x_0, i.e.

$$x(0) = x_0$$

We would like to "solve" the dynamics of the system to determine how the state will evolve in the future (i.e. for $t \geq 0$). More precisely, given some $T > 0$ we would like to determine a function

$$x(\cdot) : [0, T] \to \mathbb{R}^n$$

such that

$$\begin{aligned} x(0) &= x_0 \\ \dot{x}(t) &= f(x(t)), \forall t \in [0, T]. \end{aligned}$$

Such a function $x(\cdot)$ is called a **trajectory** (or **solution**) of the system. Notice that given a candidate trajectory $x(\cdot) : [0, T] \to \mathbb{R}^n$ one needs to verify both the differential condition and the initial condition to ensure that $x(\cdot)$ is indeed a solution of the differential equation.

Exercise 2.5 Assume instead of $x(0) = x_0$ it is required that $x(t_0) = x_0$ for some $t_0 \neq 0$. Show how one can construct solutions to the system

$$x(t_0) = x_0, \ \dot{x} = f(x)$$

from solutions to

$$x(0) = x_0, \ \dot{x} = f(x)$$

by appropriately redefining t. Could you do this with a non-autonomous system?

Without any additional information, it is unclear whether one can find a function $x(\cdot)$ solving the differential equation. A number of things can go wrong.

Example (No solutions) Consider the one dimensional system

$$\dot{x} = -sign(x), \ x(0) = 0$$

A solution to this differential equation does not exist for any $T \geq 0$.

Exercise 2.6 Assume that $x(0) = 1$. Show that solutions to the system exist for all $T \leq 1$ but not for $T > 1$.

Incidentally, something similar would happen with the radiator system if the thermostat insisted on switching the radiator on and off exactly at 20 degrees. ■

Example (Multiple Solutions) Consider the one dimensional system

$$\dot{x} = 3x^{2/3}, \ x(0) = 0$$

All functions of the form

$$x(t) = \begin{cases} (t-a)^3 & t \geq a \\ 0 & t \leq a \end{cases}$$

for any $a \geq 0$ are solutions of this differential equation.

Exercise 2.7 Verify this.

Notice that in this case the solution is not unique. In fact there are infinitely many solutions, one for each $a \geq 0$. ■

Example (Finite Escape Time) Consider the one dimensional system

$$\dot{x} = 1 + x^2,\ x(0) = 0$$

The function

$$x(t) = tan(t)$$

is a solution of this differential equation.

Exercise 2.8 Verify this. What happens at $t = \pi/2$?

Notice that the solution is defined for $T < \pi/2$ but not for $T \geq \pi/2$. ■

To eliminate such pathological cases we need to impose some assumptions on f.

Definition 2.1 *A function $f : \mathbb{R}^n \to \mathbb{R}^n$ is called* **Lipschitz continuous** *if there exists $\lambda > 0$ such that for all $x, \hat{x} \in \mathbb{R}^n$*

$$\|f(x) - f(\hat{x})\| < \lambda \|x - \hat{x}\|$$

λ is known as the Lipschitz constant. A Lipschitz continuous function is continuous, but not necessarily differentiable. All differentiable functions with bounded derivatives are Lipschitz continuous.

Exercise 2.9 Show that for $x \in \mathbb{R}$ the function $f(x) = |x|$ that returns the absolute value of x is Lipschitz continuous. What is the Lipschitz constant? Is f continuous? Is it differentiable?

Theorem 2.1 (Existence & Uniqueness of Solutions) *If f is Lipschitz continuous, then the differential equation*

$$\dot{x} = f(x)$$
$$x(0) = x_0$$

has a unique solution $x(\cdot) : [0, T] \to \mathbb{R}^n$ for all $T \geq 0$ and all $x_0 \in \mathbb{R}^n$.

Exercise 2.10 Three examples of dynamical systems that do not have unique solutions were given above. Why do these systems fail to meet the conditions of the theorem? (The details are not easy to get right.)

This theorem allows us to check whether the differential equation models we develop make sense. It also allows us to spot potential problems with proposed solutions. For example, uniqueness implies that solutions can not cross.

Exercise 2.11 Why does uniqueness imply that trajectories can not cross? (Hint: what would happen at the crossing point?).

2.3 Continuity with Respect to Initial Condition and Simulation

Theorem 2.2 (Continuity with Initial State) *Assume f is Lipschitz continuous with Lipschitz constant λ. Let $x(\cdot) : [0, T] \to \mathbb{R}^n$ and $\hat{x}(\cdot) : [0, T] \to \mathbb{R}^n$ be solutions to $\dot{x} = f(x)$ with $x(0) = x_0$ and $\hat{x}(0) = \hat{x}_0$ respectively. Then for all $t \in [0, T]$*

$$\|x(t) - \hat{x}(t)\| \leq \|x_0 - \hat{x}_0\| e^{\lambda t}$$

In other words, solutions that start close to one another remain close to one another.

This theorem provides another indication that dynamical systems with Lipschitz continuous vector fields are well behaved. For example, it provides theoretical justification for simulation algorithms. Most nonlinear differential equations are impossible to solve by hand. One can however approximate the solution on a computer, using numerical algorithms for computing integrals (Euler, Runge-Kutta, etc.). This is a process known as **simulation**.

Powerful computer packages, such as Matlab, make the simulation of most systems relatively straight forward. For example, the code used to generate the pendulum trajectories is based on a Matlab function

```
function [xprime] = pendulum(t,x)
xprime=[0; 0];
l = 1;
m=1;
d=1;
g=9.8;
xprime(1) = x(2);
xprime(2) = -sin(x(1))*g/l-x(2)*d/m;
```

The simulation code is then simply

```
>> x=[0.75 0];
>> [T,X]=ode45('pendulum', [0 10], x');
>> plot(T,X);
>> grid;
```

The continuity property ensures that the numerical approximation to the solution computed by the simulation algorithms and the actual solution remain close.

When one studies hybrid systems, many of these nice properties unfortunately vanish. As the non-deterministic thermostat system suggests, existence and uniqueness of solutions are much more difficult to guarantee. Continuity is usually impossible.

2.4 Bibliography and Further Reading

The material in this chapter is thoroughly covered in any good textbook on nonlinear dynamical systems, see for example [100, 103, 51, 88].

Chapter 3

Hybrid Automata & Executions

3.1 Examples of Hybrid Systems

Roughly speaking, hybrid systems are dynamical systems that involve the interaction of different types of dynamics. In this class we are interested in hybrid dynamics that arise out of the interaction of continuous state dynamics and discrete state dynamics. Recall that a state variable is called discrete if it takes on a finite (or countable) number of values and continuous if it takes values in Euclidean space $\mathbb{R}^n$ for some $n \geq 1$. By their nature, discrete states can change value only through a discrete "jump" (c.f. the machining example in Chapter 1). Continuous states can change values either through a jump (c.f. the logistic map example in Chapter 1), or by "flowing" in continuous time according to a differential equation (c.f. the pendulum example in Chapter 1). Hybrid systems involve both these types of dynamics: discrete jumps and continuous flows. The analysis and design of hybrid systems is in general more difficult than that of purely discrete or purely continuous systems, because the discrete dynamics may affect the continuous evolution and vice versa.

Hybrid dynamics provide a convenient framework for modelling systems in a wide range of engineering applications:

- In **mechanical systems** continuous motion may be interrupted by collisions.
- In **electrical circuits** continuous phenomena such as the charging of capacitors, etc. are interrupted by switches opening and closing, or diodes going on or off.
- In **chemical process control** the continuous evolution of chemical reactions is controlled by valves and pumps.
- In **embedded computation** systems a digital computer interacts with a mostly analogue environment.

In all these systems it is convenient (and usually fairly accurate) to model the "discrete" components (switches, valves, computers, etc.) as introducing instantaneous changes in the "continuous" components (charging of capacitors, chemical reactions, etc.).

We start our study of hybrid systems by providing a number of examples of hybrid behaviour.

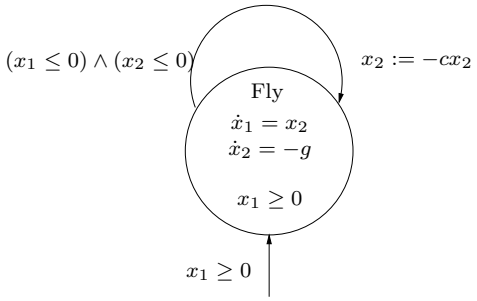

Figure 3.1: Bouncing ball

3.1.1 The Bouncing Ball

A model for a bouncing ball can be represented as a simple hybrid system (Figure 3.1) with single discrete state and a continuous state of dimension two

$$x = \begin{bmatrix} x_1 \\ x_2 \end{bmatrix},$$

where x_1 denotes the vertical position of the ball and x_2 its vertical velocity.

The continuous motion of the ball is governed by Newton's laws of motion. This is indicated by the differential equation that appears in the *vertex* (box), where g denotes the gravitational acceleration. This differential equation is only valid as long as $x_1 \geq 0$, i.e., as long as the ball is above the ground. This is indicated by the logical expression $x_1 \geq 0$ that appears in the vertex below the differential equation.

The ball bounces when $x_1 = 0$ and $x_2 \leq 0$. This is indicated by the logical expression that appears near the beginning of the edge (arrow). At each bounce, the ball loses a fraction of its energy. This is indicated by the equation $x_2 := -cx_2$ (with $c \in [0, 1]$) that appears near the end of the edge. This is an assignment statement, which means that after the bounce the speed of the ball will be c times the speed of the ball before the bounce, and in the opposite direction.

Exercise 3.1 Show that energy is preserved during continuous evolution. What fraction of the energy of the ball is lost at each bounce? What is the time interval that elapses between two bounces as a function of the energy of the ball?

Starting at an initial state with $x_1 \geq 0$ (as indicated by the logical condition next to the arrow pointing to the vertex), the continuous state flows according to the differential equation as long as the condition $x_1 \geq 0$ is fulfilled. When $x_1 = 0$ and $x_2 \leq 0$, a discrete transition takes place and the continuous state is reset to $x_2 := -cx_2$ (x_1 remains constant). Subsequently, the state resumes flowing according to the vector field, and so on. Such a trajectory is called an *execution* (and sometimes a *run* or a *solution*) of the hybrid system.

3.1.2 Gear Shift Control

The gear shift example describes a control design problem where both the continuous and the discrete controls need to be determined. Figure 3.2 shows a model of a car with a gear box having four gears.

The longitudinal position of the car along the road is denoted by x_1 and its velocity by x_2 (lateral dynamics are ignored). The model has two control signals: the gear denoted $\texttt{gear} \in \{1, \ldots, 4\}$ and the throttle position denoted $u \in [u_{\min}, u_{\max}]$. Gear shifting is necessary because little power can be generated by the engine at very low or very high engine speed. The function α_i represents the efficiency of gear i. Typical shapes of the functions α_i are shown in Figure 3.3.

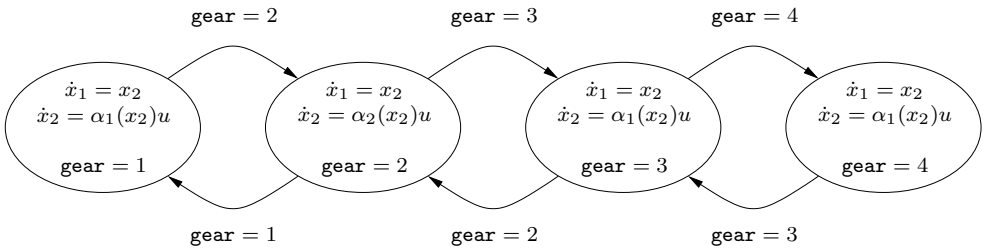

Figure 3.2: A hybrid system modelling a car with four gears.

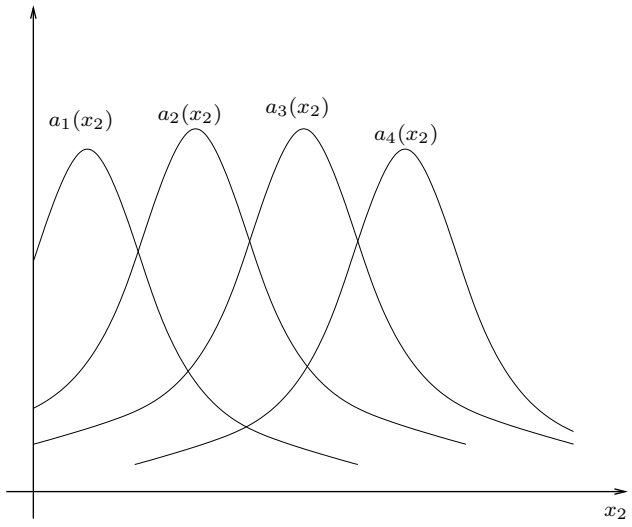

Figure 3.3: The efficiency functions of the different gears.

Exercise 3.2 How many real valued continuous states does this model have? How many discrete states?

Several interesting control problems can be posed for this simple car model. For example, what is the optimal control strategy to drive from $(a, 0)$ to $(b, 0)$ in minimum time? The problem is not trivial if we include the reasonable assumption that each gear shift takes a certain amount of time. The optimal controller, which can be modelled as a hybrid system, may be derived using the theory of optimal control of hybrid systems.

3.1.3 Computer-Controlled System

Hybrid systems are natural models for computer-controlled systems (Figure 3.4), since they involve a physical process (which often can be modelled as continuous-time system) and a computer (which is fundamentally a finite state machine). The classical approach to computer-controlled systems has been using sampled-data theory, where it is assumed that measurements and control actions are taken at a fixed sampling rate. Such a scheme is easily encoded using a hybrid model. The hybrid model also captures a more general formulation, where measurements are taken based on computer interrupts. This is sometimes closer to real-time implementations, for example, in embedded control systems.

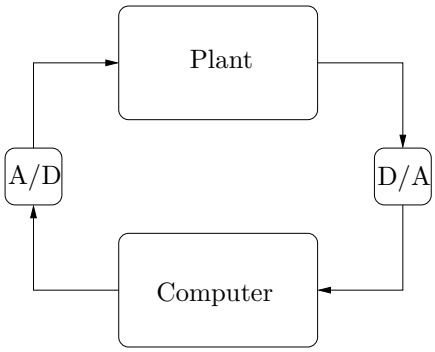

Figure 3.4: Computer-controlled system.

3.1.4 Automated Highway System

Highway congestion is an increasing problem, especially in and around urban areas. One of the promising solutions considered for this problem is traffic automation, either partial or full. The use of an automated system that performs some or all of the tasks of the driver may reduce or eliminate human errors and hence improve safety. Moreover, as the automatic controller can react to disturbances faster than a human driver, automation may also decrease the average inter-vehicle spacing and hence increase throughput and reduce congestion and delays.

The design of an Automated Highway System (AHS) is an extremely challenging control problem, and a number of alternatives have been proposed for addressing it. One of the most forward-looking AHS designs involves a fully automated highway system that supports platooning of vehicles. The platooning concept [99] assumes that traffic on the highway is organised in groups of tightly spaced vehicles (platoons). The first vehicle of a platoon is called the leader, while the remaining vehicles are called followers. The platooning structure achieves a balance between safety and throughput: it is assumed that the system is safe even if in emergency situations (for example, as a result of a failure) collisions do occur, as long as the relative velocity at impact is low. Of course no collisions should take place during normal operation. This gives rise to two safe spacing policies. The obvious one is that of the leaders, who are assumed to maintain a large inter-platoon spacing (of the order of 30-60 meters). The idea is that the leader has enough time to stop without colliding with the last vehicle of the platoon ahead. The more unintuitive spacing policy is that of the followers, who are assumed to maintain tight intra-platoon spacing (of the order of 1-5 meters). In case of emergency, collisions among the followers of a platoon may take place, but, because of the tight spacing, they are expected to be at low relative velocities. Recent theoretical, computational and experimental studies have shown that an AHS that supports platooning is not only technologically feasible but, if designed properly, may lead to an improvement of both the safety and the throughput of the highway system, under normal operation.

Implementation of the platooning concept requires automatic vehicle control, since human drivers are not fast and reliable enough to produce the necessary inputs. To manage the complexity of the design process a hierarchical controller is used. The controller is organised in four layers (Figure 3.5). The top two layers, called network and link, reside on the roadside and are primarily concerned with throughput maximisation, while the bottom two, called coordination and regulation, reside on the vehicles and are primarily concerned with safety. The physical layer is not part of the controller. It contains the "plant", i.e. the vehicles and highway, with their sensors, actuators and communication equipment.

The network layer is responsible for the flow of traffic on the entire highway system, for example, several highways around an urban area. Its task is to prevent congestion and maximise throughput

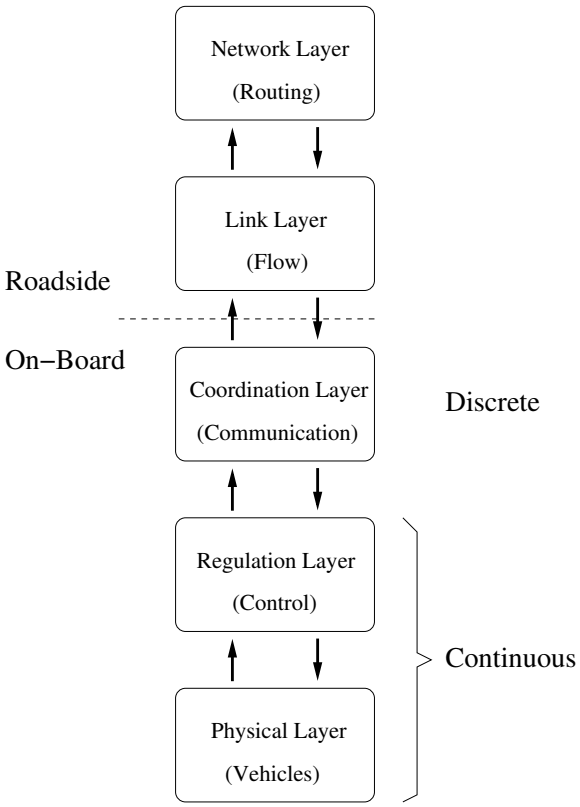

Figure 3.5: The AHS control hierarchy.

by dynamically routing traffic. The link layer coordinates the operation of sections (links) of the highway (for example the highway segment between two exits). Its primary concern is to maximise the throughput of the link. With these criteria in mind, it calculates an optimum platoon size and an optimum velocity and decides which lanes the vehicles should follow. It also monitors incidents and diverts traffic away from them, in an attempt to minimise their impact on traffic flow.

The coordination layer coordinates the operation of neighbouring platoons by choosing manoeuvres that the platoons need to carry out. For normal operation, these manoeuvres are *join* to join two platoons into one, *split* to break up one platoon into two, *lane change*, *entry* and *exit*. The coordination layer is primarily a discrete controller. It uses communication protocols, in the form of finite state machines, to coordinate the execution of these manoeuvres between neighbouring vehicles.

The regulation layer receives the coordination layer commands and readings from the vehicle sensors and generates throttle, steering and braking commands for the vehicle actuators. For this purpose it utilises a number of continuous time feedback control laws that use the readings provided by the sensors to calculate the actuator inputs required for a particular manoeuvre. In addition to the control laws needed for the manoeuvres, the regulation layer makes use of two *default* controllers, one for leader and one for follower operation.

The interaction between the coordination layer (which is primarily discrete) and the regulation layer (which is primarily continuous) gives rise to interesting hybrid dynamics. To ensure the safety of the AHS, one needs to verify that the closed loop hybrid system does not enter a bad region of its state space (e.g. does not allow any two vehicles to collide at high relative velocity). This issue can be addressed by posing the problem as a game between the control applied by one vehicle and the disturbance generated by neighbouring vehicles. It can be shown that information available through

discrete coordination can be used together with appropriate continuous controllers to ensure the safety of the closed loop hybrid system.

3.2 Hybrid Automata

To model all these diverse phenomena one needs a modelling language that is

- *descriptive*, to allow one to capture different types of continuous and discrete dynamics, be capable of modelling different ways in which discrete evolution affects and is affected by continuous evolution, allow non-deterministic models (e.g. the thermostat) to capture uncertainty, etc.
- *composable*, to allow one to build large models by composing models of simple components (e.g. for the AHS application).
- *abstractable*, to allow one to refine design problems for composite models down to design problems for individual components and, conversely, compose results about the performance of individual components to study the performance for the overall system.

Modelling languages that possess at least some subset of these properties have been developed in the hybrid systems literature. Different languages place more emphasis on different aspects, depending on the applications and problems they are designed to address. In this class we will concentrate on one such language, called *hybrid automata*. The hybrid automata we will study are fairly rich (in terms of descriptive power), but are autonomous, i.e. have no inputs and outputs. They are therefore unsuitable for studying composition and abstraction properties.

3.2.1 Hybrid Automata

A hybrid automaton is a dynamical system that describes the evolution in time of the values of a set of discrete and continuous state variables.

Definition 3.1 (Hybrid Automaton) *A hybrid automaton H is a collection $H = (Q, X, f, \mathrm{I}nit, D, E, G, R)$, where*

- $Q = \{q_1, q_2, \ldots\}$ *is a set of* **discrete states**;
- $X = \mathbb{R}^n$ *is a set of* **continuous states**;
- $f(\cdot,\cdot) : Q \times X \to \mathbb{R}^n$ *is a* **vector field**;
- $\mathrm{I}nit \subseteq Q \times X$ *is a set of* **initial states**;
- $\mathrm{D}om(\cdot) : Q \to P(X)$ *is a* **domain**;
- $E \subseteq Q \times Q$ *is a set of* **edges**;
- $G(\cdot) : E \to P(X)$ *is a* **guard condition**;
- $R(\cdot,\cdot) : E \times X \to P(X)$ *is a* **reset map**.

Recall that $P(X)$ denotes the power set (set of all subsets) of X. The notation of Definition 3.1 suggests, for example, that the function $\mathrm{D}om$ assigns a set of continuous states $\mathrm{D}om(q) \subseteq \mathbb{R}^n$ to to each discrete state $q \in Q$. We refer to $(q, x) \in Q \times X$ as the *state* of H.

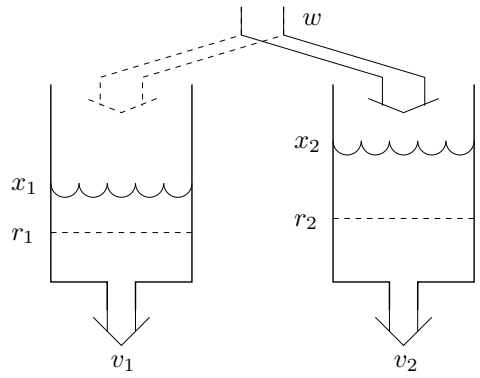

Figure 3.6: The water tank system.

Hybrid automata define possible evolutions for their state. Roughly speaking, starting from an initial value $(q_0, x_0) \in Init$, the continuous state x flows according to the differential equation

$$\dot{x} = f(q_0, x),$$
$$x(0) = x_0,$$

while the discrete state q remains constant

$$q(t) = q_0.$$

Continuous evolution can go on as long as x remains in $Dom(q_0)$. If at some point the continuous state x reaches the guard $G(q_0, q_1) \subseteq \mathbb{R}^n$ of some edge $(q_0, q_1) \in E$, the discrete state may change value to q_1. At the same time the continuous state gets reset to some value in $R(q_0, q_1, x) \subseteq \mathbb{R}^n$. After this discrete transition, continuous evolution resumes and the whole process is repeated.

To simplify the discussion, we assume from now on that the number of discrete states is finite, and that for all $q \in Q$, the vector field $f(q, \cdot)$ is Lipschitz continuous. Recall that this ensures that the solutions of the differential equation $\dot{x} = f(q, x)$ are well defined (Chapter 2). Finally, we assume that for all $e \in E$, $G(e) \neq \emptyset$, and for all $x \in G(e)$, $R(e, x) \neq \emptyset$. This assumption eliminates some pathological cases and in fact be imposed without loss of generality.

As we saw in Chapter 1 and in the examples discussed above, it is often convenient to visualise hybrid automata as directed graphs (Q, E) with vertices Q and edges E. With each vertex $q \in Q$, we associate a set of initial states $\{x \in \mathbf{X} \mid (q, x) \in Init\}$, a vector field $f(q, \cdot) : \mathbb{R}^n \to \mathbb{R}^n$ and a domain $Dom(q) \subseteq \mathbb{R}^n$. An edge $(q, q') \in E$ starts at $q \in Q$ and ends at $q' \in Q$. With each edge $(q, q') \in E$, we associate a guard $G(q, q') \subseteq \mathbb{R}^n$ and a reset function $R(q, q', \cdot) : \mathbb{R}^n \to P(\mathbb{R}^n)$.

Example (Water Tank System) The two tank system, shown in Figure 3.6, consists of two tanks containing water. Both tanks are leaking at a constant rate. Water is added to the system at a constant rate through a hose, which at any point in time is dedicated to either one tank or the other. It is assumed that the hose can switch between the tanks instantaneously.

For $i = 1, 2$, let x_i denote the volume of water in Tank i and $v_i > 0$ denote the constant flow of water out of Tank i. Let w denote the constant flow of water into the system. The objective is to keep the water volumes above r_1 and r_2, respectively, assuming that the water volumes are above r_1 and r_2 initially. This is to be achieved by a controller that switches the inflow to Tank 1 whenever $x_1 \leq r_1$ and to Tank 2 whenever $x_2 \leq r_2$.

It is straight forward to define a hybrid automaton, to describe this process:

- $Q = \{q_1, q_2\}$ (two discrete states, inflow going left and inflow going right);
- $X = \mathbb{R}^2$ (two continuous states, the level of water in the two tanks);

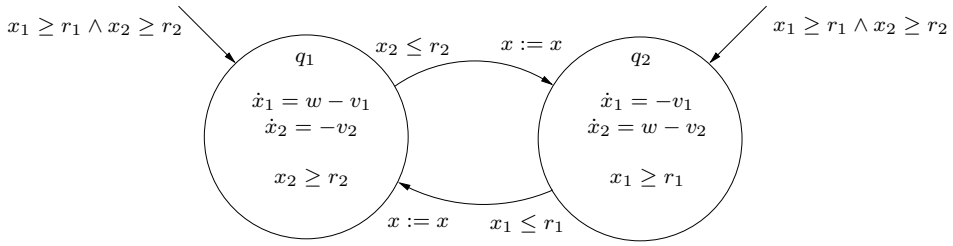

Figure 3.7: Graphical representation of the water tank hybrid automaton.

- (when the inflow is going to the tank on the right, the water level in the left tank goes down while the water level in right tank goes up, and vice verse)

$$f(q_1, x) = \begin{bmatrix} w - v_1 \\ -v_2 \end{bmatrix}, \text{ and } f(q_2, x) = \begin{bmatrix} -v_1 \\ w - v_2 \end{bmatrix};$$

- $Init = \{q_1, q_2\} \times \{x \in \mathbb{R}^2 \mid x_1 \geq r_1 \wedge x_2 \geq r_2\}$ (start with both water levels above the low level marks r_1 and r_2);
- $Dom(q_1) = \{x \in \mathbb{R}^2 \mid x_2 \geq r_2\}$ and $Dom(q_2) = \{x \in \mathbb{R}^2 \mid x_1 \geq r_1\}$ (put water in the current tank as long as the level in the other tank is above the low level mark);
- $E = \{(q_1, q_2), (q_2, q_1)\}$ (possible to switch inflow from left to right and vice versa);
- $G(q_1, q_2) = \{x \in \mathbb{R}^2 \mid x_2 \leq r_2\}$ and $G(q_2, q_1) = \{x \in \mathbb{R}^2 \mid x_1 \leq r_1\}$ (switch the inflow to the other tanks as soon as the water there reaches the low level mark);
- $R(q_1, q_2, x) = R(q_2, q_1, x) = \{x\}$ (the continuous state does not change as a result of switching the inflow).

The directed graph corresponding to this hybrid automaton is shown in Figure 3.7. ■

The directed graphs contain exactly the same information as Definition 3.1. They can therefore be treated as informal definitions of hybrid automata. It is common to remove the assignment $x := x$ from an edge of the graph when the continuous state does not change as a result of the discrete transition corresponding to that edge.

3.2.2 Hybrid Time Sets & Executions

Hybrid automata involve both continuous "flow" determined by differential equations and discrete "jumps" determined by a directed graph (like an automaton). To characterise the evolution of the state of a hybrid automaton one has to think of a set of times that contains both continuous intervals (over which continuous evolution takes place) and distinguished discrete points in time, when discrete transitions happen. Such a set of times is called a *hybrid time set*.

Definition 3.2 (Hybrid Time Set) *A hybrid time set is a sequence of intervals $\tau = \{I_0, I_1, \ldots, I_N\} = \{I_i\}_{i=0}^N$, finite or infinite (i.e. $N = \infty$ is allowed) such that*

- *$I_i = [\tau_i, \tau_i']$ for all $i < N$;*
- *if $N < \infty$ then either $I_N = [\tau_N, \tau_N']$ or $I_N = [\tau_N, \tau_N')$; and*
- *$\tau_i \leq \tau_i' = \tau_{i+1}$ for all i.*

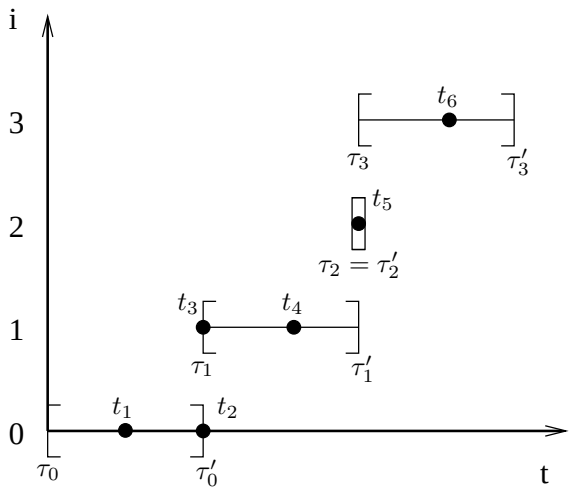

Figure 3.8: A hybrid time set $\tau = \{[\tau_i, \tau_i']\}_{i=0}^{3}$.

An example of a hybrid time set is given in Figure 3.8. Notice that the right endpoint, τ_i', of the interval I_i coincides with the left endpoint, τ_{i+1} of the interval I_{i+1} (c.f. the time instants labelled t_2 and t_3 in Figure 3.8). The interpretation is that these are the times at which discrete transitions of the hybrid system take place. τ_i' corresponds to the time instant just before a discrete transition, whereas τ_{i+1} corresponds to the time instant just after the discrete transition. Discrete transitions are assumed to be instantaneous, therefore $\tau_i' = \tau_{i+1}$. The advantage of this convention is that it allows one to model situations where multiple discrete transitions take place one after the other at the same time instant, in which case $\tau_{i-1}' = \tau_i = \tau_i' = \tau_{i+1}$ (c.f. the interval $I_2 = [\tau_2, \tau_2]'$ in Figure 3.8).

Despite its somewhat complicated nature, a hybrid time set, τ, is a rather well behaved mathematical object. For example, there is a natural way in which the elements of the hybrid time set can be ordered. For $t_1 \in [\tau_i, \tau_i'] \in \tau$ and $t_2 \in [\tau_j, \tau_j'] \in \tau$ we say that t_1 *precedes* t_2 (denoted by $t_1 \prec t_2$) if $t_1 < t_2$ (i.e. if the real number t_1 is less that the real number t_2) or if $i < j$ (i.e. if t_1 belongs to an earlier interval than t_2). In Figure 3.8, we have $t_1 \prec t_2 \prec t_3 \prec t_4 \prec t_5 \prec t_6$. In general, given any two distinct time instants, t_1 and t_2, belonging to some τ we have that either $t_1 \prec t_2$ or $t_2 \prec t_2$ (c.f. given any two distinct real numbers x and y, either $x < y$ or $y < x$). Using mathematical terminology, one would say that each hybrid time set τ is linearly ordered by the relation $\prec$.

Given two hybrid time sets τ and $\hat{\tau}$ there is also a natural way to define if one is "shorter" than the other (τ is called a *prefix* of $\hat{\tau}$ if it is "shorter"). More formally, we say that $\tau = \{I_i\}_{i=0}^{N}$ is a prefix of $\hat{\tau} = \{\hat{I}_i\}_{i=0}^{M}$ (and write $\tau \sqsubseteq \hat{\tau}$) if either they are identical, or τ is a finite sequence, $N \leq M$ (notice that M can be infinite), $I_i = \hat{I}_i$ for all $i = 0, \ldots, N-1$, and $I_N \subseteq \hat{I}_N$. We say that τ is a *strict prefix* of $\hat{\tau}$ (and write $\tau \sqsubset \hat{\tau}$) if $\tau \sqsubseteq \hat{\tau}$ and $\tau \neq \hat{\tau}$. In Figure 3.9, τ is a strict prefix of both $\hat{\tau}$ and $\tilde{\tau}$, but $\hat{\tau}$ is not a prefix of $\tilde{\tau}$ and $\tilde{\tau}$ is not a prefix of $\hat{\tau}$. Notice that given τ and $\hat{\tau}$ we may have neither $\hat{\tau} \sqsubseteq \tau$ nor $\tau \sqsubseteq \hat{\tau}$ (c.f. given two sets of real numbers $A \subseteq \mathbb{R}$ and $B \subseteq \mathbb{R}$ it is possible to have neither $A \subseteq B$ nor $B \subseteq A$). Using mathematical terminology, one would say that the set of all hybrid time sets is partially ordered by the relation $\sqsubseteq$.

Hybrid time sets will be used to define the time horizon over which the states of hybrid systems evolve. What does it mean for the state to "evolve" over a hybrid time set? For continuous systems with state $x \in \mathbb{R}^n$ such an evolution was a function, $x(\cdot) : [0, T] \to \mathbb{R}^n$, mapping a time interval $[0, T]$ to the set $\mathbb{R}^n$ where the state lives (see Chapter 2). For discrete systems (like the manufacturing machine example of Chapter 1) whose state takes values in a finite set $q \in \{q_1, \ldots, q_n\}$ such an evolution was a sequence of states. For hybrid systems, where the state has both a continuous component $x \in \mathbb{R}^n$ and a discrete component $q \in \{q_1, \ldots, q_n\}$ we need to come up with a mixture of these two notions.

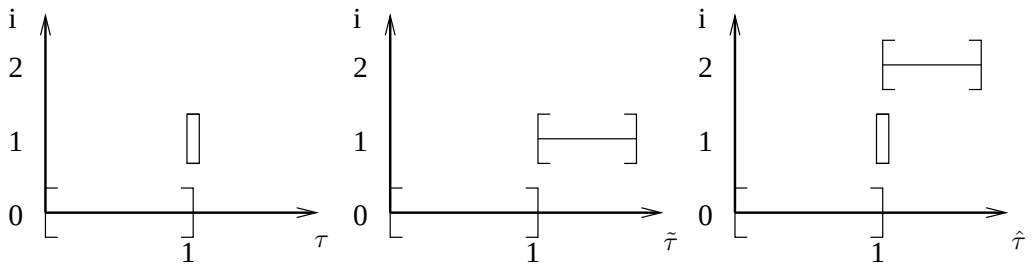

Figure 3.9: $\tau \sqsubset \hat{\tau}$ and $\tau \sqsubset \tilde{\tau}$.

Definition 3.3 (Hybrid Trajectory) *A hybrid trajectory is a triple (τ, q, x) consisting of a hybrid time set $\tau = \{I_i\}_0^N$ and two sequences of functions $q = \{q_i(\cdot)\}_0^N$ and $x = \{x_i(\cdot)\}_0^N$ with $q_i(\cdot) : I_i \to Q$ and $x(\cdot) : I_i \to \mathbb{R}^n$.*

An execution of an autonomous hybrid automaton is a hybrid trajectory, (τ, q, x) of its state variables. The elements listed in Definition 3.1 impose restrictions on the types of hybrid trajectories that the hybrid automaton finds "acceptable".

Definition 3.4 (Execution) *An execution of a hybrid automaton H is a hybrid trajectory, (τ, q, x), which satisfies the following conditions:*

- Initial condition*: $(q_0(0), x_0(0)) \in \mathrm{I}nit$.*
- Discrete evolution*: for all i, $(q_i(\tau_i'), q_{i+1}(\tau_{i+1})) \in E$, $x_i(\tau_i') \in G(q_i(\tau_i'), q_{i+1}(\tau_{i+1}))$, and $x_{i+1}(\tau_{i+1}) \in R(q_i(\tau_i'), q_{i+1}(\tau_{i+1}), x_i(\tau_i'))$.*
- Continuous evolution*: for all i,*
  1. *$q_i(\cdot) : I_i \to Q$ is constant over $t \in I_i$, i.e. $q_i(t) = q_i(\tau_i)$ for all $t \in I_i$;*
  2. *$x_i(\cdot) : I_i \to X$ is the solution to the differential equation*

$$\frac{dx_i}{dt} = f(q_i(t), x_i(t))$$

  over I_i starting at $x_i(\tau_i)$; and,
  3. *for all $t \in [\tau_i, \tau_i')$, $x_i(t) \in \mathrm{D}om(q_i(t))$.*

Definition 3.4 specifies which of the hybrid trajectories are executions of H and which are not by imposing a number of restrictions. The first restriction dictates that the executions should start at an acceptable initial state in Init. For simplicity, we will use $(q_0, x_0) = (q_0(\tau_0), x_0(\tau_0)) \in \mathrm{I}nit$ to denote the initial state of an execution (τ, q, x). As for continuous systems, we can assume that $\tau_0 = 0$ without loss of generality. The second restriction determines when discrete transitions can take place and what the state after discrete transitions can be. The requirements relate the state before the discrete transition $(q_i(\tau_i'), x_i(\tau_i'))$ to the state after the discrete transition $(q_{i+1}(\tau_{i+1}), x_{i+1}(\tau_{i+1}))$: they should be such that $(q_i(\tau_i'), q_{i+1}(\tau_{i+1}))$ is an edge of the graph, $x_i(\tau_i')$ belongs to the guard of this edge and $x_{i+1}(\tau_{i+1})$ belongs the the reset map of this edge. In this context, it is convenient to think of the guard $G(e)$ as *enabling* a discrete transition $e \in E$: the execution *may* take a discrete transition $e \in E$ from a state x as long as $x \in G(e)$. The third restriction determines what happens along continuous evolution, and when continuous evolution must give way to a discrete transition. The first part dictates that along continuous evolution the discrete state remains constant. The second part requires that along continuous evolution the continuous state flows according to the differential equation $\dot{x} = f(q, x)$. Notice that the differential equation depends on the discrete state we are currently in (which is constant along continuous evolution). The third part requires that

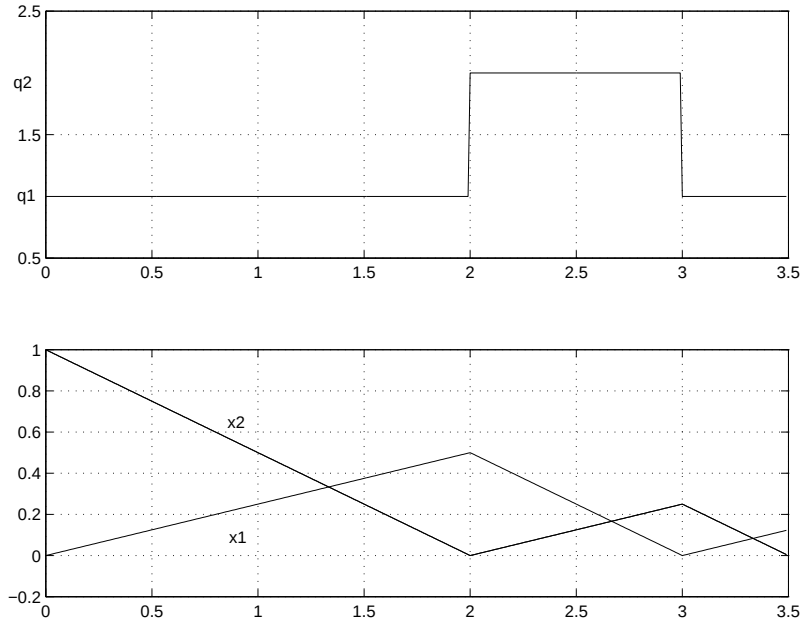

Figure 3.10: Example of an execution of the water tank hybrid automaton.

along continuous evolution the state must remain in the domain, D$om(q)$, of the discrete state. In this context, it is convenient to think of D$om(q)$ as *forcing* discrete transitions: the execution *must* take a transition if the state is about to leave the domain.

Example (Water Tank (cont.)) Figure 3.10 shows an execution of the water tank automaton. The hybrid time set τ of the execution consists of three intervals, $\tau = \{[0,2],[2,3],[3,3.5]\}$. The evolution of the discrete state is shown in the upper plot, and the evolution of the continuous state is shown in the lower plot. The values chosen for the constants are $r_1 = r_2 = 0$, $v_1 = v_2 = 1/2$ and $w = 3/4$. The initial state is $q = q_1$, $x_1 = 0$, $x_2 = 1$. ■

A convenient interpretation is that the hybrid automaton *accepts* (as opposed to generates) executions. This perspective allows one to consider, for example, hybrid automata that accept multiple executions for some initial states, a property that can prove very useful when modelling uncertain system (as illustrated by the thermostat example of Chapter 1).

Definition 3.5 (Classification of executions) *An execution (τ, q, x) is called:*

- **Finite**, *if τ is a finite sequence and the last interval in τ is closed.*
- **Infinite**, *if τ is an infinite sequence, or if the sum of the time intervals in τ is infinite, i.e.*

$$\sum_{i=0}^{N}(\tau_i' - \tau_i) = \infty.$$

- **Zeno**, *if it is infinite but $\sum_{i=0}^{\infty}(\tau_i' - \tau_i) < \infty$.*
- **Maximal** *if it is not a strict prefix of any other execution of H.*

Figure 3.11 shows examples of hybrid time sets of finite, infinite and Zeno executions.

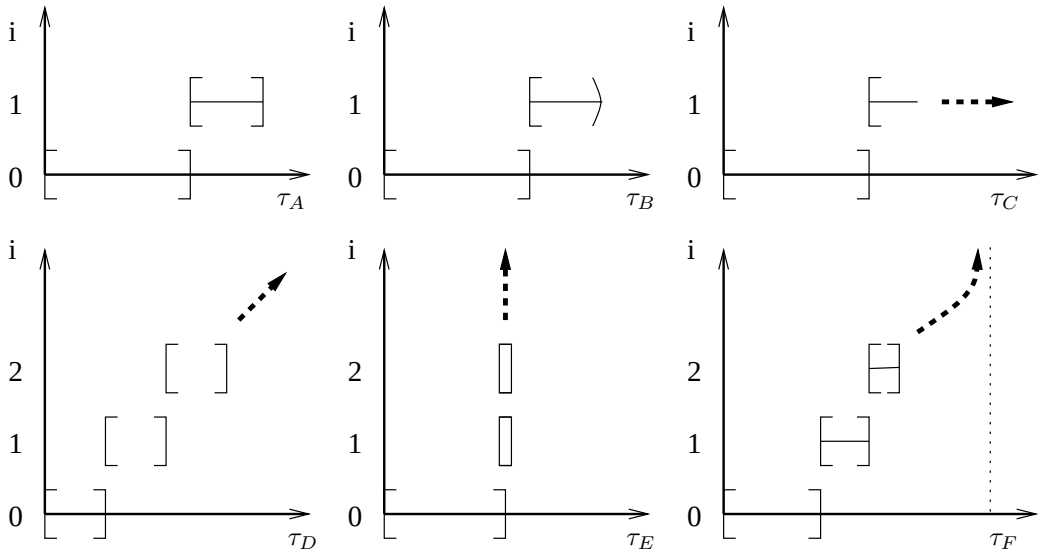

Figure 3.11: τ_A finite, τ_C and τ_D infinite, τ_E and τ_F Zeno.

Exercise 3.3 Show that an execution is Zeno if and only if it takes an infinite number of discrete transitions in a finite amount of time. Does an execution definer over the hybrid time set τ_B of Figure 3.11 belong to any of the classes of Definition 3.5?

3.3 Bibliography and Further Reading

Hybrid systems arise naturally in a number of engineering applications. In addition to the applications mentioned above, the hybrid paradigm has also been used successfully to address problems in air traffic control [95], automotive control [15], bioengineering [23], chemical process control [57, 33], highway systems [99, 44] and manufacturing [83].

The formal definition of hybrid automata is based on a fairly standard class of autonomous hybrid systems. The notation used here comes from [64, 47]. This class of systems has been studied extensively in the literature in a number of variations, for a number of purposes, and by a number of authors. Special cases of the class of systems considered here include switched systems [82], complementarity systems [97], mixed logic dynamic systems [39], and piecewise linear systems [49] (the autonomous versions of these, to be more precise). The hybrid automata considered here are a special case of the hybrid automata of [5] and the impulse differential inclusions of [14] (discussed in Chapter 7 of these notes), both of which allow differential inclusions to model the continuous dynamics. They are a special case of the General Hybrid Dynamical Systems of [21], which allow the continuous state to take values in manifolds (different ones for each discrete state). They are also a special case of hybrid input/output automata of [68], which, among other things, allow infinite-dimensional continuous state.

Chapter 4

Existence of Executions

4.1 Modelling Issues

Powerful modelling languages, such as hybrid automata, allow one to model a very wide variety of physical phenomena, but also make it possible to produce models that are unreasonable, either physically or mathematically. A common danger in hybrid modelling is lack of existence of solutions. In most of the hybrid languages one can easily construct models that admit no solutions for certain initial states. Such systems are known as *blocking* hybrid systems. This is an undesirable property when modelling physical systems, since it suggests that the mathematical model provides an incomplete picture of the physical reality: the evolution of the physical system is likely to continue despite the fact that the evolution of the mathematical model is undefined.

Even if a hybrid system accepts executions for all initial states, it does not necessarily accept executions with infinite execution times. For example, the executions of hybrid models can take an infinite number of discrete transitions in finite time. Such executions are known as *Zeno* executions. One can argue that physical systems do not exhibit Zeno behaviour. However, modelling abstraction can often lead to Zeno models of physical systems. Since abstraction is crucial for handling complex systems, understanding when it leads to Zeno hybrid systems is important.

Another issue that arises in hybrid modelling is lack of uniqueness of solutions. Hybrid systems that accept multiple executions for a single initial state are known as *non-deterministic*. It is often desirable to retain some level of non-determinism in a hybrid model, since it can be used to model uncertainty (recall the thermostat example of Chapter 1). This, however, requires additional care when designing controllers for such systems, or when developing arguments about their performance. A common practice in continuous dynamical systems is to base proofs on arguments about *the solution* of the system. This is motivated by the fact that, under fairly general assumptions (Lipschitz continuity, recall Theorem 2.1), continuous dynamical system have unique solutions. This proof technique is inadequate for non-deterministic systems. Instead one needs to develop arguments that hold for *all solutions* of the system.

Finally, hybrid systems are especially challenging from the point of view of simulation. The problems faced by the developers of simulation algorithms are intimately related to the modelling problems discussed so far.

- *Existence*: simulation algorithms may run into trouble if the simulated model has no solutions. Incorporating tests for existence in the simulation packages can alleviate this problem. More challenging is the case of Zeno executions. In this case, unless special care is taken, the simulation may grind to a halt, or produce spurious results.
- *Uniqueness*: Non-determinism introduces further complications from the point of view of sim-

ulation. Here the simulation algorithm may be called upon to decide between different alternatives. When a choice between continuous evolution and discrete transition is possible, a common approach is to take transitions the moment they are enabled (*as-soon-as* semantics). Probabilistic methods have also been proposed for dealing with non-determinism in the context of simulation.

- *Discontinuity*: Lack of continuity of the solution with respect to initial conditions, an inherent characteristic of hybrid systems, can also lead to problems, both theoretical and practical. The most common problem is event detection (guard crossing).
- *Composability*: When simulating large scale systems (e.g. the Automated Highway System of Chapter 3), one would like to be able to build up the simulation by composing different components (e.g. models for the motion of each vehicle). It may also be desirable to add components to the simulation on-line (e.g. to model vehicles joining the highway), eliminate components (e.g. to model vehicles leaving the highway), or redefine the interactions between components (e.g. to model vehicles changing lanes). Object oriented modelling languages have been developed to address these needs.

4.2 Two Fundamental Concepts

Reachability is a fundamental concept in the study of hybrid systems (and dynamical systems in general). Roughly speaking, a state, $(\hat{q}, \hat{x}) \in Q \times X$ of a hybrid automaton H is called reachable if the hybrid automaton can find its way to $(\hat{q}, \hat{x})$ while moving along one of its executions. The importance of the concept of reachability is difficult to overstate. In the next section we will show how reachability plays a central role in the derivation of existence and uniqueness conditions for executions. Reachability will also turn out to be a key concept in the study of safety properties for hybrid systems.

More formally,

Definition 4.1 (Reachable State) *A state $(\hat{q}, \hat{x}) \in Q \times X$ of a hybrid automaton H is called reachable if there exists a finite execution (τ, q, x) ending in $(\hat{q}, \hat{x})$, i.e. $\tau = \{[\tau_i, \tau_i']\}_0^N$, $N < \infty$, and $(q_N(\tau_N'), x_N(\tau_N')) = (\hat{q}, \hat{x})$.*

We will use R$each \subseteq Q \times X$ to denote the set of all states reachable by H. Clearly, I$nit \subseteq$ R$each$.

Exercise 4.1 Why are all initial states reachable?

Another important concept in the study of existence of executions for hybrid automata is the set of states from which continuous evolution is impossible. We will call these states *transition states*. For $(\hat{q}, \hat{x}) \in Q \times X$ and some $\epsilon > 0$, consider the solution, $x(\cdot) : [0, \epsilon) \to \mathbb{R}^n$ of the differential equation

$$\frac{dx}{dt} = f(\hat{q}, x) \text{ with } x(0) = \hat{x}. \tag{4.1}$$

Notice that, under the assumption that f is Lipschitz continuous in x, the solution to equation (4.1) exists and is unique (Theorem 2.1). The states, T$rans \subseteq Q \times X$, from which continuous evolution is impossible are

$$\text{T}rans = \{(\hat{q}, \hat{x}) \in Q \times X \mid \forall \epsilon > 0 \; \exists t \in [0, \epsilon) \text{ such that } (\hat{q}, x(t)) \notin \text{D}om(\hat{q})\}.$$

In words, T$rans$ is the set of states for which continuous evolution along the differential equation forces the system to exit the domain instantaneously.

The exact characterisation of the set T$rans$ may be quite involved. Clearly, continuous evolution is impossible for all states outside the domain (refer to Definition 3.4). Therefore, for each discrete

state $q \in Q$, states in the complement of the domain of q (i.e. the set of all x outside D$om(q)$, denoted by D$om(q)^c$) must belong to T$rans$. Mathematically this can be written as

$$\bigcup_{q \in Q} \{q\} \times \mathrm{D}om(q)^c \subseteq \mathrm{T}rans$$

If D$om(q)$ is a closed set (i.e. it contains its boundary), then T$rans$ may also contain pieces of the boundary of the domain. In the examples considered in this class, it is usually straight forward to figure out what these parts are going to be.

Example (Water Tank (continued)) Consider again the water tank automaton, and assume that

$$0 < v_1, v_2 < w.$$

We will show how to compute the sets R$each$ and T$rans$ for this system.

First of all R$each$ must contain all initial states. Therefore

$$\mathrm{R}each \supseteq \{q_1, q_2\} \times \{x \in \mathbb{R}^2 \mid (x_1 \geq r_1) \wedge (x_2 \geq r_2)\} \tag{4.2}$$

Can it contain any other states? It turns out that it can not. To see why, we will show using induction that the state remains in the set Init.

Consider an arbitrary initial state $(\hat{q}, \hat{x}) \in \mathrm{I}nit$ and an arbitrary execution (τ, q, x) starting at $(\hat{q}, \hat{x})$. The fact that $(\hat{q}, \hat{x}) \in \mathrm{I}nit$ provides the base case for the induction argument. Assume that for some i, $(q_i(\tau_i), x_i(\tau_i)) \in \mathrm{I}nit$, and consider the case where $q_i(\tau_i) = q_1$ (the case $q_i(\tau_i) = q_2$ is similar). If $\tau_i' > \tau_i$, then continuous evolution takes place from $(q_i(\tau_i), x_i(\tau_i))$. Along this evolution, the first component of the continuous state increases (because $q_i(\tau_i) = q_1$, therefore $\dot{x}_1 = w - v_1$ and $v_1 < w$). The second component of the continuous state, on the other hand, decreases, but remains above r_2. This is because, by the definition of an execution,

$$x_i(t) \in \mathrm{D}om(q_1) = \{x \in \mathbb{R}^2 \mid x_2 \geq r_2\}$$

for all $t \in [\tau_i, \tau_i']$. Therefore, $(q_i(t), x_i(t))$ remains in Init along continuous evolution.

If $\tau_i' = \infty$, or if $[\tau_i, \tau_i']$ is the last interval in τ we are done! Otherwise, a discrete transition takes place from $(q_i(\tau_i'), x_i(\tau_i'))$. But the reset relation, R, leaves x unaffected, therefore,

$$(q_{i+1}(\tau_{i+1}), x_{i+1}(\tau_{i+1})) \in \mathrm{I}nit$$

The last statement is true even if $\tau_i = \tau_i'$.

Summarising, if $(q_i(\tau_i), x_i(\tau_i)) \in \mathrm{I}nit$, then $(q_i(t), x_i(t)) \in \mathrm{I}nit$ for all $t \in [\tau_i, \tau_i']$. Moreover, $(q_{i+1}(\tau_{i+1}), x_{i+1}(\tau_{i+1})) \in \mathrm{I}nit$. Therefore, by induction on i, $(q_i(t), x_i(t)) \in \mathrm{I}nit$ for all i and all t and

$$\mathrm{R}each \subseteq \{q_1, q_2\} \times \{x \in \mathbb{R}^2 \mid (x_1 \geq r_1) \wedge (x_2 \geq r_2)\} \tag{4.3}$$

Equations (4.2) and (4.3) together imply that

$$\mathrm{R}each = \{q_1, q_2\} \times \{x \in \mathbb{R}^2 \mid (x_1 \geq r_1) \wedge (x_2 \geq r_2)\}$$

To establish the set T$rans$ for the water tank system, notice that continuous evolution is impossible if $q = q_1$ and $x_2 < r_2$ (the inflow will get immediately switched to tank 2) or if $q = q_2$ and $x_1 < r_1$. Therefore,

$$\mathrm{T}rans \supseteq \big(\{q_1\} \times \{x \in \mathbb{R}^2 \mid x_2 < r_2\}\big) \cup \big(\{q_2\} \times \{x \in \mathbb{R}^2 \mid x_1 < r_1\}\big)$$

On the other hand, continuous evolution is possible if $q = q_1$ and $x_2 > r_2$, or if $q = q_2$ and $x_1 > r_1$. Therefore

$$\mathrm{T}rans \subseteq \big(\{q_1\} \times \{x \in \mathbb{R}^2 \mid x_2 \leq r_2\}\big) \cup \big(\{q_2\} \times \{x \in \mathbb{R}^2 \mid x_1 \leq r_1\}\big)$$

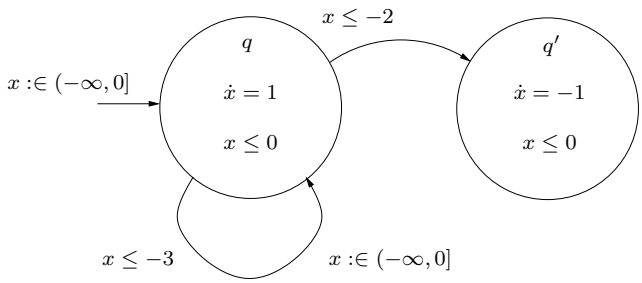

Figure 4.1: Examples of blocking and non-determinism.

How about if $q = q_1$ and $x_2 = r_2$? If continuous evolution was to take place from this state, x_2 would immediately go below r_2. This is because $q = q_1$ implies that $\dot{x}_2 = -v_2 < 0$ (recall that $v_2 > 0$). This, however, would imply that the state would leave the domain $\mathrm{D}om(q_1)$, which is impossible along continuous evolution. Therefore, continuous evolution is also impossible from states where $q = q_1$ and $x_2 = r_2$ (and, by a symmetric argument, states where $q = q_2$ and $x_1 = r_1$). Overall,

$$\mathrm{T}rans = \left(\{q_1\} \times \{x \in \mathbb{R}^2 \mid x_2 \leq r_2\}\right) \cup \left(\{q_2\} \times \{x \in \mathbb{R}^2 \mid x_1 \leq r_1\}\right).$$

■

4.3 Local Existence and Uniqueness

Next, we turn our attention to questions of existence of executions. We give some conditions under which infinite executions exist for all initial states, and conditions under which these executions are unique.

Definition 4.2 (Non-Blocking and Deterministic) *A hybrid automaton H is called non-blocking if for all initial states $(\hat{q}, \hat{x}) \in \mathrm{I}nit$ there exists an infinite execution starting at $(\hat{q}, \hat{x})$. It is called deterministic if for all initial states $(\hat{q}, \hat{x}) \in \mathrm{I}nit$ there exists at most one maximal execution starting at $(\hat{q}, \hat{x})$.*

Roughly speaking, the non-blocking property implies that infinite executions exist for all initial states, while the deterministic property implies that the infinite executions (if they exist) are unique. As we have seen, continuous dynamical systems described by differential equations have both these properties if the vector field f is assumed to be Lipschitz continuous (Theorem 2.1). In hybrid systems, however, more things can go wrong.

Consider, for example, the hybrid automaton of Figure 4.1. Let $(\hat{q}, \hat{x})$ denote the initial state, and notice that $\hat{q} = q$. If $\hat{x} = -3$, executions starting at $(\hat{q}, \hat{x})$ can either flow along the vector field $\dot{x} = 1$, or jump back to q reseting x anywhere in $(-\infty, 0]$, or jump to q' leaving x unchanged. If $\hat{x} = -2$ executions starting at $(\hat{q}, \hat{x})$ can either flow along the vector field, or jump to q'. If $\hat{x} = -1$ executions stating at $(\hat{q}, \hat{x})$ can only flow along the vector field. Finally, if $\hat{x} = 0$ there are no executions starting at $(\hat{q}, \hat{x})$, other than the trivial execution defined over $[\tau_0, \tau_0']$ with $\tau_0 = \tau_0'$. Therefore, the hybrid automaton of Figure 4.1 accepts no infinite executions for some initial states and multiple infinite executions for others.

Intuitively, a hybrid automaton is non-blocking if for all reachable states for which continuous evolution is impossible a discrete transition is possible. This fact is stated more formally in the following lemma.

Lemma 4.1 *A hybrid automaton, H, is non-blocking if for all $(\hat{q}, \hat{x}) \in \mathrm{R}each \cap \mathrm{T}rans$, there exists $\hat{q}' \in Q$ such that $(\hat{q}, \hat{q}') \in E$ and $\hat{x} \in G(\hat{q}, \hat{q}')$. If H is deterministic, then it is non-blocking if and only if this condition holds.*

Proof: Consider an initial state $(q_0, x_0) \in Init$ and assume, for the sake of contradiction, that there does not exist an infinite execution starting at (q_0, x_0). Let $\chi = (\tau, q, x)$ denote a maximal execution starting at (q_0, x_0), and note that τ is a finite sequence.

First consider the case $\tau = \{[\tau_i, \tau_i']\}_{i=0}^{N-1}[\tau_N, \tau_N')$ and let $(q_N, x_N) = \lim_{t \to \tau_N'}(q_N(t), x_N(t))$. Note that, by the definition of execution and a standard existence argument for continuous dynamical systems, the limit exists and χ can be extended to $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ with $\widehat{\tau} = \{[\tau_i, \tau_i']\}_{i=0}^{N}$, $\widehat{q}_N(\tau_N') = q_N$, and $\widehat{x}_N(\tau_N') = x_N$. This contradicts the assumption that χ is maximal.

Now consider the case $\tau = \{[\tau_i, \tau_i']\}_{i=0}^{N}$, and let $(q_N, x_N) = (q_N(\tau_N'), x_N(\tau_N'))$. Clearly, $(q_N, x_N) \in Reach$. If $(q_N, x_N) \notin Trans$, then there exists $\epsilon > 0$ such that χ can be extended to $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ with $\widehat{\tau} = \{[\tau_i, \tau_i']\}_{i=0}^{N-1}[\tau_N, \tau_N' + \epsilon)$, by continuous evolution. If, on the other hand $(q_N, x_N) \in Trans$, then by assumption there exists $(q', x') \in Q \times X$ such that $(q_N, q') \in E$, $x_N \in G(q_N, q')$ and $x' \in R(q_N, q', x_N)$. Therefore, χ can be extended to $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ with $\widehat{\tau} = \{[\tau_i, \tau_i']\}_{i=0}^{N+1}$, $\tau_{N+1} = \tau_{N+1}' = \tau_N'$, $q_{N+1}(\tau_{N+1}) = q'$, $x_{N+1}(\tau_{N+1}) = x'$ by a discrete transition. In both cases the assumption that χ is maximal is contradicted.

This argument also establishes the "if" of the second part. For the "only if", consider a deterministic hybrid automaton that violates the conditions, i.e., there exists $(q', x') \in Reach$ such that $(q', x') \in Trans$, but there is no $\widehat{q}' \in Q$ with $(q', \widehat{q}') \in E$ and $x' \in G(q', \widehat{q}')$. Since $(q', x') \in Reach$, there exists $(q_0, x_0) \in Init$ and a finite execution, $\chi = (\tau, q, x)$ starting at (q_0, x_0) such that $\tau = \{[\tau_i, \tau_i']\}_{i=0}^{N}$ and $(q', x') = (q_N(\tau_N'), x_N(\tau_N'))$.

We first show that χ is maximal. Assume first that there exists $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ with $\widehat{\tau} = \{[\tau_i, \tau_i']\}_{i=0}^{N-1}$ $[\tau_N, \tau_N' + \epsilon)$ for some $\epsilon > 0$. This would violate the assumption that $(q', x') \in Trans$. Next assume that there exists $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ with $\widehat{\tau} = \tau[\tau_{N+1}, \tau_{N+1}']$ with $\tau_{N+1} = \tau_N'$. This requires that the execution can be extended beyond (q', x') by a discrete transition, i.e., there exists $(\widehat{q}', \widehat{x}') \in Q \times X$ such that $(q', \widehat{q}') \in E$, $x' \in G(q', \widehat{q}')$ and $\widehat{x}' \in R(q', \widehat{q}', x')$. This would contradict our original assumptions. Overall, χ is maximal.

Now assume, for the sake of contradiction that H is non-blocking. Then, there exists an infinite (and therefore maximal) χ' starting at (q_0, x_0). But $\chi \neq \chi'$ (as the former is finite and the latter infinite). This contradicts the assumption that H is deterministic. ∎

Intuitively, a hybrid automaton may be non-deterministic if either there is a choice between continuous evolution and discrete transition, or if a discrete transition can lead to multiple destinations (recall that continuous evolution is unique by Theorem 2.1). More specifically, the following lemma states that a hybrid automaton is deterministic if and only if (1) whenever a discrete transition is possible continuous evolution is impossible, and (2) discrete transitions have unique destinations.

Lemma 4.2 *A hybrid automaton, H, is deterministic if and only if for all $(\hat{q}, \hat{x}) \in Reach$*

1. *if $\hat{x} \in G(\hat{q}, \hat{q}')$ for some $(\hat{q}, \hat{q}') \in E$, then $(\hat{q}, \hat{x}) \in Trans$;*
2. *if $(\hat{q}, \hat{q}') \in E$ and $(\hat{q}, \hat{q}'') \in E$ with $\hat{q}' \neq \hat{q}''$ then $\hat{x} \notin G(\hat{q}, \hat{q}') \cap G(\hat{q}, \hat{q}'')$; and,*
3. *if $(\hat{q}, \hat{q}') \in E$ and $x \in G(\hat{q}, \hat{q}')$ then $R(\hat{q}, \hat{q}', \hat{x}) = \{\hat{x}'\}$, i.e. the set contains a single element, $\hat{x}'$.*

Proof: For the "if" part, assume, for the sake of contradiction, that there exists an initial state $(q_0, x_0) \in Init$ and two maximal executions $\chi = (\tau, q, x)$ and $\widehat{\chi} = (\widehat{\tau}, \widehat{q}, \widehat{x})$ starting at (q_0, x_0) with $\chi \neq \widehat{\chi}$. Let $\bar{\chi} = (\bar{\tau}, \bar{q}, \bar{x})$ denote the maximal common prefix of χ and $\widehat{\chi}$. Such a prefix exists as the executions start at the same initial state. Moreover, $\bar{\chi}$ is not infinite, as $\chi \neq \widehat{\chi}$. As in the proof of Lemma 4.1, $\bar{\tau}$ can be assumed to be of the form $\bar{\tau} = \{[\bar{\tau}_i, \bar{\tau}_i']\}_{i=0}^{N}$. Let $(q_N, x_N) = (q_N(\bar{\tau}_N'), x_N(\bar{\tau}_N')) = (\widehat{q}_N(\bar{\tau}_N'), \widehat{x}_N(\bar{\tau}_N'))$. Clearly, $(q_N, x_N) \in Reach$. We distinguish the following four cases:

Case 1: $\bar{\tau}'_N \notin \{\tau'_i\}$ and $\bar{\tau}'_N \notin \{\widehat{\tau}'_i\}$, i.e., $\bar{\tau}'_N$ is not a time when a discrete transition takes place in either χ or $\widehat{\chi}$. Then, by the definition of execution and a standard existence and uniqueness argument for continuous dynamical systems, there exists $\epsilon > 0$ such that the prefixes of χ and $\widehat{\chi}$ are defined over $\{[\bar{\tau}_i, \bar{\tau}'_i]\}_{i=0}^{N-1}[\bar{\tau}_N, \bar{\tau}'_N + \epsilon)$ and are identical. This contradicts the fact that $\bar{\chi}$ is maximal.

Case 2: $\bar{\tau}'_N \in \{\tau'_i\}$ and $\bar{\tau}'_N \notin \{\widehat{\tau}'_i\}$, i.e., $\bar{\tau}'_N$ is a time when a discrete transition takes place in χ but not in $\widehat{\chi}$. The fact that a discrete transition takes place from (q_N, x_N) in χ indicates that there exists $q' \in Q$ such that $(q_N, q') \in E$ and $x_N \in G(q_N, q')$. The fact that no discrete transition takes place from (q_N, x_N) in $\widehat{\chi}$ indicates that there exists $\epsilon > 0$ such that $\widehat{\chi}$ is defined over $\{[\bar{\tau}_i, \bar{\tau}'_i]\}_{i=0}^{N-1}[\bar{\tau}_N, \bar{\tau}'_N + \epsilon)$. A necessary condition for this is that $(q_N, x_N) \notin \mathrm{T}rans$. This contradicts condition 1 of the lemma.

Case 3: $\bar{\tau}'_N \notin \{\tau'_i\}$ and $\bar{\tau}'_N \in \{\widehat{\tau}'_i\}$, symmetric to Case 2.

Case 4: $\bar{\tau}'_N \in \{\tau'_i\}$ and $\bar{\tau}'_N \in \{\widehat{\tau}'_i\}$, i.e., $\bar{\tau}'_N$ is a time when a discrete transition takes place in both χ and $\widehat{\chi}$. The fact that a discrete transition takes place from (q_N, x_N) in both χ and $\widehat{\chi}$ indicates that there exist (q', x') and $(\widehat{q}', \widehat{x}')$ such that $(q_N, q') \in E$, $(q_N, \widehat{q}') \in E$, $x_N \in G(q_N, q')$, $x_N \in G(q_N, \widehat{q}')$, $x' \in R(q_N, q', x_N)$, and $\widehat{x}' \in R(q_N, \widehat{q}', x_N)$. Note that by condition 2 of the lemma, $q' = \widehat{q}'$, hence, by condition 3, $x' = \widehat{x}'$. Therefore, the prefixes of χ and $\widehat{\chi}$ are defined over $\{[\bar{\tau}_i, \bar{\tau}'_i]\}_{i=0}^{N}[\bar{\tau}_{N+1}, \bar{\tau}'_{N+1}]$, with $\bar{\tau}_{N+1} = \bar{\tau}'_{N+1} = \bar{\tau}'_N$, and are identical. This contradicts the fact that $\bar{\chi}$ is maximal and concludes the proof of the "if" part.

For the "only if" part, assume that there exists $(q', x') \in \mathrm{R}each$ such that at least one of the conditions of the lemma is violated. Since $(q', x') \in \mathrm{R}each$, there exists $(q_0, x_0) \in \mathrm{I}nit$ and a finite execution, $\chi = (\tau, q, x)$ starting at (q_0, x_0) such that $\tau = \{[\tau_i, \tau'_i]\}_{i=0}^{N}$ and $(q', x') = (q_N(\tau'_N), x_N(\tau'_N))$. If condition 1 is violated, then there exist $\widehat{\chi}$ and $\tilde{\chi}$ with $\widehat{\tau} = \{[\tau_i, \tau'_i]\}_{i=0}^{N-1}[\tau_N, \tau'_N + \epsilon)$, $\epsilon > 0$, and $\tilde{\tau} = \tau[\tau_{N+1}, \tau'_{N+1}]$, $\tau_{N+1} = \tau'_N$, such that $\chi \sqsubset \widehat{\chi}$ and $\chi \sqsubset \tilde{\chi}$. If condition 2 is violated, there exist $\widehat{\chi}$ and $\tilde{\chi}$ with $\widehat{\tau} = \tilde{\tau} = \tau[\tau_{N+1}, \tau'_{N+1}]$, $\tau_{N+1} = \tau'_{N+1} = \tau'_N$, and $\widehat{q}_{N+1}(\tau_{N+1}) \neq \tilde{q}_N(\tau_{N+1})$, such that $\chi \sqsubset \widehat{\chi}$, $\chi \sqsubset \tilde{\chi}$. Finally, if condition 3 is violated, then there exist $\widehat{\chi}$ and $\tilde{\chi}$ with $\widehat{\tau} = \tilde{\tau} = \tau[\tau_{N+1}, \tau'_{N+1}]$, $\tau_{N+1} = \tau'_{N+1} = \tau'_N$, and $\widehat{x}_{N+1}(\tau_{N+1}) \neq \tilde{x}_{N+1}(\tau_{N+1})$, such that $\chi \sqsubset \widehat{\chi}$, $\chi \sqsubset \tilde{\chi}$. In all three cases, let $\overline{\widehat{\chi}}$ and $\overline{\tilde{\chi}}$ denote maximal executions of which $\widehat{\chi}$ and $\tilde{\chi}$ are prefixes, respectively. Since $\widehat{\chi} \neq \tilde{\chi}$, it follows that $\overline{\widehat{\chi}} \neq \overline{\tilde{\chi}}$. Therefore, there are at least two maximal executions starting at (q_0, x_0) and thus H is non-deterministic. ■

The following theorem is a direct consequence of Lemmas 4.1 and 4.2.

Theorem 4.1 (Existence and Uniqueness) *A hybrid automaton H accepts a unique infinite execution for each initial state if it satisfies all the conditions of Lemmas 4.1 and 4.2.*

Important Note: The conditions of the lemmas involve the set of reachable states, R*each*. This is needed only to make the conditions necessary. If all we are interested in is establishing that a hybrid automaton accepts an infinite executions for all initial states, or that infinite executions are unique, it suffices to show that the conditions of the lemmas hold for all states (as opposed to all *reachable* states). This can make our life considerably easier, since calculating the set of reachable states is sometimes hard.

Example (Water Tank (continued)) Consider again the water tank automaton with $0 < v_1, v_2 < w$. Recall that

$$\begin{aligned}
\mathrm{R}each &= \{(q, x) \in \mathbf{Q} \times \mathbb{R}^2 \mid x_1 \geq r_1 \wedge x_2 \geq r_2\},\\
\mathrm{T}rans &= \{q_1\} \times \{x \in \mathbb{R}^2 \mid x_2 \leq r_2\} \cup \{q_2\} \times \{x \in \mathbb{R}^2 \mid x_1 \leq r_1\}.
\end{aligned}$$

Therefore,

$$\begin{aligned}
\mathrm{R}each \cap \mathrm{T}rans = &\{q_1\} \times \{x \in \mathbb{R}^2 \mid x_1 \geq r_1 \wedge x_2 = r_2\} \cup\\
&\{q_2\} \times \{x \in \mathbb{R}^2 \mid x_2 \geq r_2 \wedge x_1 = r_1\}.
\end{aligned}$$

Consider an arbitrary state $(\hat{q}, \hat{x}) \in \mathrm{R}each \cap \mathrm{T}rans$ (in fact the argument holds for any state $(\hat{q}, \hat{x}) \in \mathrm{T}rans$, see "important note" above). Notice that, if $\hat{q} = q_1$, then

$$\hat{x} \in \{x \in \mathbb{R}^2 \mid (x_1 \geq r_1) \wedge (x_2 = r_2)\} \subseteq G(q_1, q_2).$$

Likewise, if $q = q_2$, then $x \in G(q_1, q_2)$. Therefore, the condition of Lemma 4.1 is satisfied, and the water tank system is non-blocking.

Next, consider an arbitrary reachable state $(\hat{q}, \hat{x}) \in \mathrm{R}each$ (in fact the argument holds for any state $(\hat{q}, \hat{x}) \in Q \times X$). Assume that $\hat{q} = q_1$ (a similar argument holds if $\hat{q} = q_2$).

1. If $\hat{x} \in G(q_1, q_2) = \{x \in \mathbb{R}^2 \mid x_2 \leq r_2\}$, then $x_2 \leq r_2$. Therefore $(\hat{q}, \hat{x}) \in \mathrm{T}rans$.
2. Only one discrete transition is possible from q_1 (namely $(q_1, q_2) \in E$).
3. $R(q_1, q_2, \hat{x}) = \{\hat{x}\}$ contains one element.

Therefore, the conditions of Lemma 4.2 are also satisfied. By Theorem 4.1, the water tank automaton accepts a unique infinite execution for each initial state. ■

4.4 Zeno Executions

The conditions of Theorem 4.1 ensure that a hybrid automaton accepts infinite executions for all initial states. They do not, however, ensure that the automaton accepts executions defined over arbitrarily long time horizons. The Lipschitz assumption on f eliminates the possibility of escape to infinity in finite time along continuous evolution (c.f. finite escape example in Chapter 2). However, the infinite executions may be such that the state takes an infinite number of discrete transitions in finite time. Executions with this property are known as Zeno executions.

The name "Zeno" comes from the ancient Greek philosopher, Zeno of Elea. Born around 490BC, Zeno was a philosopher and one of the founders of the Eleatic school. He was a student of Parmenides, whose teachings rejected the ideas of plurality and transition as illusions generated by our senses. The main contribution of Zeno was a series of paradoxes designed to support the view of his mentor by showing that accepting plurality and motion leads to logical contradictions. One of the better known ones is the race of Achilles and the turtle.

> Achilles, a renowned runner, was challenged by the turtle to a race. Being a fair sportsman, Achilles decided to give the turtle a 100 meter head-start. To overtake the turtle, Achilles will have to first cover half the distance separating them, i.e. 50 meters. To cover the remaining 50 meters, he will first have to cover half that distance, i.e. 25 meters, and so on. There are an infinite number of such segments and to cover each one of them Achilles needs a non zero amount of time. Therefore, Achilles will never overtake the turtle.

This paradox may seem simple minded, but it was not until the beginning of the 20^{th} century that it was resolved satisfactorily by mathematicians and philosophers. And it was not until the end of the 20^{th} century that it turned out to be a practical problem, in the area of hybrid systems.

The Zeno phenomenon is notoriously difficult to characterise and eliminate in hybrid systems. In this class we will not examine the properties of Zeno executions in detail. We will only give some examples of hybrid automata that admit Zeno behaviour.

Example (Chattering System) Consider the hybrid automaton of Figure 4.2. It is easy to show that this hybrid automaton accepts a unique infinite execution for all initial states. However, all

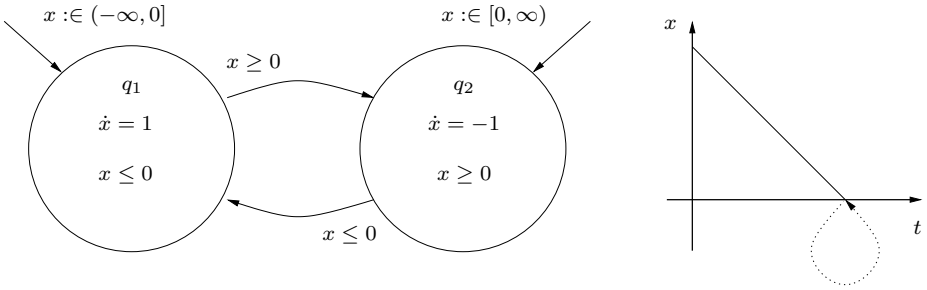

Figure 4.2: Chattering system.

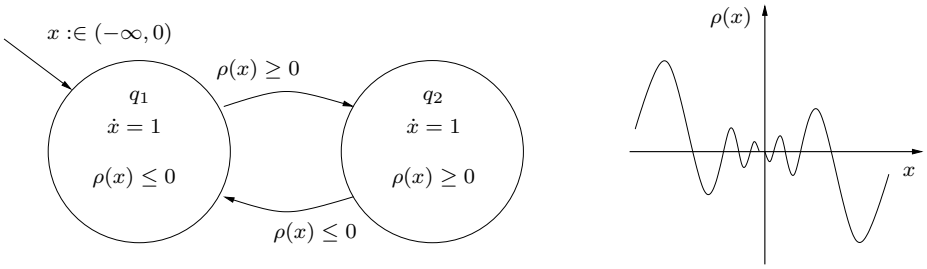

Figure 4.3: System with a smooth, non-analytic domain.

infinite executions are Zeno. An execution starting in x_0 at time τ_0 reaches $x = 0$ in finite time $\tau_0' = \tau_0 + |x_0|$ and takes an infinite number of transitions from then on, without time progressing further.

This is a phenomenon known in continuous dynamical system as *chattering*. A bit of thought in fact reveals that this system is the same as the example used to demonstrate absence of solutions in Chapter 2. In the control literature the "Zenoness" of such chattering systems is sometimes eliminated by allowing weaker solution concepts, such as sliding solutions (also known as Filippov solutions). A more thorough treatment of this topic can be found in [34, 96]. ■

Example (Non-analytic Domain) Consider the hybrid automaton of Figure 4.3. Assume that the function $\rho : \mathbb{R} \to \mathbb{R}$ that determines the boundary of the domain is of the form

$$\rho(x) = \begin{cases} \sin\left(\frac{1}{x^2}\right) \exp\left(-\frac{1}{x^2}\right) & \text{if } x \neq 0 \\ 0 & \text{if } x = 0 \end{cases}$$

It is easy to check that the automaton is non-blocking and deterministic.

For any $\epsilon > 0$, ρ has an infinite number of zero crossings in the interval $(-\epsilon, 0]$. Therefore, the execution of the hybrid automaton with initial state (q_1, x_0) will take an infinite number of discrete transitions before time $\tau_0 + |x_0|$ (notice that $x_0 < 0$). ■

Example (Water Tank (continued)) We have already shown that the water tank hybrid automaton accepts a unique infinite execution for each initial state if $0 < v_1, v_2 < w$. If in addition the inflow is less than the sum of the outflows ($w < v_1 + v_2$), then all infinite executions are Zeno. It is easy to show that the execution starting at time τ_0 takes an infinite number of transitions by time

$$\tau_0 + \frac{x_1(\tau_0) + x_2(\tau_0) - r_1 - r_2}{v_1 + v_2 - w}$$

■

4.5 Bibliography and Further Reading

The simulation of hybrid systems presents special challenges, that need particular attention. Nowadays general purpose simulation packages such as Matlab and Simulink can deal adequately with most complications (this was not always the case!) Specialised packages have also been developed that allow accurate simulation of hybrid systems (at least to the extent that this is possible in theory). For references see [8, 76, 75, 7]. See also [31] for a compositional language for hybrid system simulation.

The fundamental properties of existence and uniqueness of solutions, continuity with respect to initial conditions, etc. naturally attracted the attention of researchers in hybrid systems from fairly early on. The majority of the work in this area concentrated on developing conditions for well-posedness (existence and uniqueness of solutions) for special classes of hybrid systems: piecewise linear systems [46, 56], complementarity systems [97, 37], mixed logic dynamical systems [39], etc. The discussion in these notes is based on [64, 63, 47].

Continuity of the solutions with respect to initial conditions and parameters has also been studied, but somewhat less extensively. Motivated by questions of simulation, Tavernini [93] established a class of hybrid systems that have the property of continuous dependence of solutions for almost every initial condition. More recently, an approach to the study of continuous dependence on initial conditions based on the Skorohod topology was proposed [24]. The Skorohod topology, used in stochastic processes for the space of cadlag functions [18], is mathematically appealing, but tends to be cumbersome to work with in practice. [64] presents a more practical (but still limited) approach to the question of continuity.

Zeno executions are treated in [3, 17, 4] from a computer science perspective and [47, 48, 38, 92, 107] from a control theory perspective. [47, 74] attempt to define extensions of Zeno execution beyond the Zeno time, motivated by the classical definition of "sliding" flows for discontinuous vector fields.

Chapter 5

Analysis and Synthesis

5.1 Specifications

The reason why we are interested in modelling hybrid systems is that we would like to be able to analyse the resulting models, and infer some properties of the real system from them. If control inputs are available, we would also like to be able to design controllers such that the closed loop hybrid system satisfies certain desirable properties.

Given a hybrid automaton modelling a physical system and some desirable property that we would like this system to possess (**specification**), we would like to be able answer the following two questions:

1. **Verification**: does the hybrid automaton *meet the specification* (satisfy the desirable property).
2. **Synthesis**: if there are some design choices to be made (e.g. the system has control inputs and a controller needs to be designed) can the design be done in such a way that the resulting system meets the specification.

For real systems, verification and synthesis are usually followed by a process of **validation**: the theoretical hybrid design is implemented on a prototype of the real system and tests are performed to determine whether it meets the specification. It is not uncommon for a design to fail the validation test, due to factors that were omitted in the hybrid automaton model. The design then needs to be tuned further and the process of synthesis-verification-validation needs to be repeated.

What kinds of specifications may one want to impose on a hybrid system? A common specification is stability: one would like to determine whether a hybrid system is stable or asymptotically stable. If control inputs are available, the problem becomes one of stabilisation: Can one choose a controller such that the closed loop system is stable or asymptotically stable? Just as for purely continuous systems, Lyapunov methods are very useful in this case. Both the stability definitions and Lyapunov theorems need to be appropriately modified to account for the presence of discrete states. This topic will not be addressed further in this class. For more details, interested students may want to refer to [30].

In this class we will concentrate primarily on properties that can be encoded as sets of hybrid trajectories. Recall that a hybrid trajectory (Definition 3.3) is a triple, (τ, q, x) consisting of a hybrid time set τ and two sequences of functions, q and x mapping each interval of τ to the discrete states Q and the continuous states $\mathbb{R}^n$ respectively. A specification can be encoded as a set of desirable hybrid trajectories, $\mathcal{H}$. A hybrid automaton is then said to meet the specification if all the executions of the hybrid automaton belong to this set $\mathcal{H}$. (Recall that the executions of a

hybrid automaton are also hybrid trajectories, in particular the ones that meet the conditions of Definition 3.4).

What kind of properties can be encoded like this? The most common are properties that have to do with reachability. For example, the property

$$\text{“The state } (q,x) \text{ always remains in a set of states } F \subseteq Q \times X\text{”}$$

is one such property, and so is the dual property

$$\text{“The state } (q,x) \text{ eventually reaches a set of states } F \subseteq Q \times X\text{”}$$

The first property is known as a **safety property**, because the set F can be used to encode “good” or “safe” states. For example, when analysing the behaviour of two cars following one another on an automated highway, F can be the set of states for which the two cars have not collided (i.e. the spacing between them is greater than zero); we would like to ensure that the cars always remain in the set F (i.e. do not collide). Using notation from **Temporal Logic** this safety property can be written as

$$\Box((q,x) \in F)$$

$\Box$ stands for “always”; the way to read the above formula is “always (q,x) is in F”.

The second property (“The state (q,x) eventually reaches F”) is known as a **liveness property**. It reflects the fact that something good should eventually happen to our system. For example, cars on an automated highway not only want to avoid collisions with other cars, but also want to reach their destination. In the temporal logic notation this property can be written as

$$\Diamond((q,x) \in F)$$

$\Diamond$ stands for “eventually”; the way to read the above formula is “eventually (q,x) is in F”.

Using concepts like these one can encode arbitrarily complex properties. For example the property

$$\Box\Diamond((q,x) \in F)$$

stands for “always, eventually (q,x) is in F”, or in other words, the state visits the set F “infinitely often”. Another example is

$$\Diamond\Box((q,x) \in F)$$

which stands for “eventually always (q,x) is in F”, or in other words, (q,x) reaches F at some point and stays there for ever after. And so on.

How can one check that a hybrid automaton meets such a specification? Roughly speaking there are three different approaches to this problem:

1. **Model Checking**: For certain classes of hybrid systems the process can be carried out completely automatically. The system and the specification are encoded in an appropriate programming language, and given to a computer to analyse. After a few minutes/hours/days/weeks the computer either runs out of memory, or comes back with an answer: “The system satisfies the specification”, or “The system does not satisfy the specification”. In the latter case the computer also generates an execution of the hybrid automaton that fails to meet the specification; such an execution is known as a *witness*. The witness is useful for redesigning the system. The basics of this approach will be given in Chapter 6.

2. **Deductive**: Induction arguments, progress variables, etc. are used to develop a proof that the system meets the specification. Most of the work in structuring the proof has to be done my hand. Computational theorem proving tools may then be used to check the individual steps of the proof. The basics of this approach will be the topic of the rest of this handout.

3. **Optimal Control and Viability Theory**: Reachability problems can also be encoded in an optimal control or viability theory setting. The optimal control approach requires some rather advanced machinery from optimal control theory and will not be covered in this class. The viability theory approach will be introduced in Chapter 7. Most of the work with these approaches has to be done analytically (by hand!) Because optimal control problems are notoriously difficult to solve analytically one often has to resort to numerical tools (PDE solvers, etc.) to approximate the solution.

5.2 Deductive Methods

In the rest of this chapter we will introduce some basic principles of deductive analysis, motivated by the reachability problem. First of all notice that

Proposition 5.1 *A hybrid automaton H satisfies a specification $\Box((q,x) \in F)$ if and only if* R*each* $\subseteq F$.

Exercise 5.1 Prove Proposition 5.1.

Deductive arguments aim to establish bounds on R*each* through invariant sets. The definition of "invariant set" for hybrid automata is a direct generalisation of the definition for continuous dynamical systems: a set of states is called invariant if all executions of the hybrid automaton starting in that set remain in that set for ever. More formally,

Definition 5.1 (Invariant Set) *A set of states, $M \subseteq Q \times X$, of a hybrid automaton, H, is called invariant if for all $(\hat{q},\hat{x}) \in M$, all executions (τ,q,x) starting at $(\hat{q},\hat{x})$, all $I_i \in \tau$ and all $t \in I_i$ we have that $(q_i(t),x_i(t)) \in M$.*

In the above statement "execution (τ,q,x) starting at $(\hat{q},\hat{x})$" refers to a hybrid trajectory with $(q_0(\tau_0),x_0(\tau_0)) = (\hat{q},\hat{x})$ that satisfies the discrete and continuous evolution conditions of Definition 3.4, but not necessarily the initial condition (i.e. we allow $(\hat{q},\hat{x}) \notin$ I*nit* in Definition 5.1). Students are asked to forgive this slight abuse of the terminology.

The following fact is a direct consequence of the definition.

Proposition 5.2 *Consider a hybrid automaton H.*

1. *The union and intersection of two invariant sets of H are also invariant sets of H.*
2. *If M is an invariant set and* I*nit* $\subseteq M$, *then* R*each* $\subseteq M$.

Exercise 5.2 Prove Proposition 5.2.

The two parts of the proposition can be used together to provide progressively tighter bounds on R*each*, by figuring out invariant sets that contain I*nit* and then taking their intersection. Given a specification of the form $\Box((q,x) \in F)$, the idea is to find an invariant set that contains I*nit* (and hence R*each*) and is contained in F, i.e.

$$\text{I}nit \subseteq M \subseteq F.$$

How does one prove that a set is invariant? Usually by induction. Assume we suspect that a certain set of states, $M \subseteq Q \times \mathbb{R}^n$, may be invariant. First of all we may want to check that the initial states are contained in the set M (otherwise M may turn out to be useless for proving safety properties). Then we check that if continuous evolution starts from a state in M then it remains in M throughout. In other words, we check that for all $T \geq 0$, if

- $(\hat{q}, \hat{x}) \in M$, and
- $x : [0, T] \to \mathbb{R}^n$ is the solution to $\frac{dx}{dt} = f(\hat{q}, x)$ starting at $x(0) = \hat{x}$, and
- $x(t) \in \mathrm{D}om(\hat{q})$ for all $t \in [0, T)$,

then

- $(\hat{q}, x(t)) \in M$ for all $t \in [0, T]$

Exercise 5.3 Show that it is sufficient that $(\hat{q}, x(T)) \in M$ (i.e. we do not need to check $(\hat{q}, x(t)) \in M$ at intermediate times). This is not difficult, but requires a bit of thought.

Finally, we check that if a discrete transition is possible from somewhere in M, then the state after the discrete transition is also in M. In other words, if

- $(\hat{q}, \hat{x}) \in M$, and
- $(\hat{q}, \hat{q}') \in E$, and
- $\hat{x} \in G(\hat{q}, \hat{q}')$

then

- $R(\hat{q}, \hat{q}', \hat{x}) \subseteq M$

We have actually seen this procedure in practice already: these were the steps we followed to determine the set of states reachable by the water tank system.

5.3 Bibliography and Further Reading

Of the analysis questions listed in this chapter, the one that has arguably attracted the most attention if the question of stability of equilibria and invariant sets. Most of the work in this area has concentrated on extensions of Lyapunov's Direct Method to the hybrid domain [106, 20, 79]. The work of [50] provided effective computational tools, based on Linear Matrix Inequalities, for applying these results to a class of piecewise linear systems. [64] discuss extensions of LaSalle's Method and Lyapunov's Indirect Method to the hybrid domain. Related to the latter is also the work of [45, 105], where a direct study of the stability of piecewise linear systems is developed. For an excellent overview of the literature in this area the reader is referred to [30].

The corresponding synthesis problem of stabilisation has been somewhat less extensively studied. Much of the work in this area deals with switched systems (usually linear and/or planar). The proposed stabilisation schemes typically involve selecting appropriate times for switching between a set of given systems [82, 102, 60, 59, 105]. In some cases this approach has been extended to robust stabilisation schemes for systems that involve certain types of uncertainty [89, 45].

Temporal logic is widely used in computer to encode properties given as sets of trajectories (safety properties, etc.) as well as dynamics for discrete systems. A very thorough treatment can be found in [70, 71].

Deductive methods are commonly used with discrete systems; see [70, 71, 67] for an overview. One way of formally extending deductive arguments to the hybrid domain is presented in [22, 68, 72]; the approach of [22, 68] has been applied to a number of examples, primarily to establish the safety of transportation systems [40, 101, 32, 65, 62, 61].

Deductive arguments can be automated (at least partly) using theorem provers. One tool that provides computational support for deductive arguments for hybrid systems is STeP [73].

Chapter 6

Model Checking and Timed Automata

Finite state systems are relatively easy to work with because one can investigate their properties by systematically exploring their states. For example, one can decide if a finite state system will eventually visit a particular set of states by following the system trajectories one by one. This is tedious to do by hand, but is easy to implement on a computer. Moreover, the process is guaranteed to terminate: since the number of states is finite, sooner or later we will find ourselves visiting the same states over and over again. At this point either the desired set has already been reached, or, if not, it will never be reached.

With hybrid systems it is in general impossible to do this. Because the number of states is infinite, it is impossible to enumerate them and try to explore them one by one. However, there are hybrid systems for which one can find a finite state system which is, in some sense, equivalent to the hybrid system. This is usually done by partitioning the state space into a finite number of sets with the property that any two states in a give set exhibit similar behaviour. Then, to decide whether the hybrid system has certain properties, one has to work with the finite sets of the partition, as opposed to the infinite states of the original hybrid system. Moreover, the generation and analysis of the finite partition can be carried out automatically by a computer.

The process of automatically analysing the properties of systems by exploring their state space is known as *model checking*. In this handout we discuss some fundamental ideas behind model checking, and introduce a class of hybrid systems, known as *timed automata*, that are amenable to this approach. As with deductive methods the discussion will be motivated by safety (reachability) analysis. Because of the complexity of the material we will not develop the results in their full beauty and generality. A good starting point for a deeper study is [5].

6.1 Transition Systems

We first introduce a very general class of dynamical systems, known as transition systems, on which the above questions can be formulated.

Definition 6.1 (Transition System) *A transition system, $T = (S, \delta, S_0, S_F)$ consists of*

- *A set of states S (finite or infinite);*
- *A transition relation $\delta : S \to P(S)$;*
- *A set of initial states $S_0 \subseteq S$;*

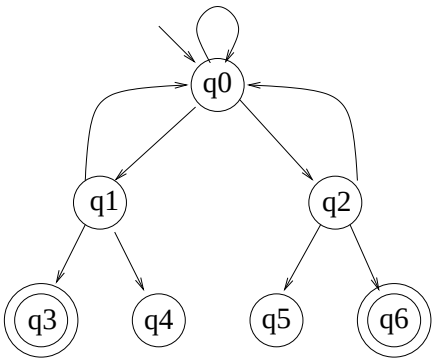

Figure 6.1: Finite state transition system.

- *A set of final states* $S_F \subseteq S$.

The set of final states is included because we are interested in reachability type specifications. We will use it to encode desired final states, sets of states in which we want to stay, etc.

A *trajectory* of a transition system is finite or infinite sequence of states $\{s_i\}_{i=0}^{N}$ such that

1. $s_0 \in S_0$; and,
2. $s_{i+1} \in \delta(s_i)$ for all $i = 0, 1, \ldots, N-1$.

Example (Finite State Transition System) A transition system with finite states is shown in Figure 6.1. The formal definition of the system is

1. $S = \{q_0, \ldots, q_6\}$;
2. $\delta(q_0) = \{q_0, q_1, q_2\}$, $\delta(q_1) = \{q_0, q_3, q_4\}$, $\delta(q_2) = \{q_0, q_5, q_6\}$, $\delta(q_3) = \delta(q_4) = \delta(q_5) = \delta(q_6) = \emptyset$;
3. $S_0 = \{q_0\}$ (denoted by an arrow pointing to q_0);
4. $S_F = \{q_3, q_6\}$ (denoted by a double circle).

■

Example (Transition System of a Hybrid Automaton) Hybrid automata can be transformed into transition systems by abstracting away time. Consider a hybrid automaton $H = (Q, X, \mathrm{I}nit, f, \mathrm{D}om, E, G, R)$ together with a distinguished "final" set of states $F \subseteq Q \times X$. We will define a transition system for the hybrid automaton. Start with

- $S = Q \times X$, i.e. $s = (q, x)$
- $S_0 = \mathrm{I}nit$
- $S_F = F$.

The transition relation δ can be defined in many parts: a discrete transition relation $\delta_e : S \to P(S)$ for each edge $e \in E$ of the graph and a continuous transition relation $\delta_C : S \to P(S)$ to model the passage of time. For each $e = (q, q') \in E$ define

$$\delta_e(\hat{q}, \hat{x}) = \begin{cases} \{q'\} \times R(e, \hat{x}) & \text{if } (\hat{q} = q) \text{ and } (\hat{x} \in G(e)) \\ \emptyset & \text{if } (\hat{q} \neq q) \text{ or } (x \notin G(e)) \end{cases}$$

For the continuous transition relation let

$$\delta_C(\hat{q},\hat{x}) = \{(\hat{q}',\hat{x}') \in Q \times X \mid [\hat{q}' = \hat{q}] \wedge [\exists T \geq 0,\ (x(T) = \hat{x}') \wedge (\forall t \in [0,T], x(t) \in \mathrm{D}om(\hat{q}))]\}$$

where $x(\cdot)$ denotes the solution of the differential equation

$$\dot{x} = f(\hat{q},x) \text{ starting at } x(0) = \hat{x}$$

The overall transition relation is then

$$\delta(s) = \delta_C(s) \cup \bigcup_{e \in E} \delta_e(s)$$

In words, a transition from $s \in S$ to $s' \in S$ is possible in the transition system if either a discrete transition $e \in E$ of the hybrid system will bring s to s', or s can flow continuously to s' after some time. Notice that in the last statement time has been abstracted away. We do not care how long it takes to get from s to s', we only care whether it is possible to get there eventually. The transition system captures the sequence of "events" that the hybrid system may experience, but not the timing of these events. ■

Transition systems are designed to allow one to answer reachability (and other) questions algorithmically. For example, say we would like to answer the question

"Does there exist a trajectory of T such that $s_i \in S_F$ for some i?"

If this is the case we say that S_F *is reachable*. We can approach this question using the predecessor operator

$$\mathrm{P}re : P(S) \to P(S)$$

which for each set of states $\hat{S} \subseteq S$ is defined as

$$\mathrm{P}re(\hat{S}) = \{s \in S \mid \exists \hat{s} \in \hat{S} \text{ with } \hat{s} \in \delta(s)\}$$

The operator Pre takes a set of states, $\hat{S}$, and returns the states that can reach $\hat{S}$ in one transition. The following algorithm can then be used to determine if S_F is reachable.

Algorithm 1 (Backwards Reachability)
initialisation: $W_0 = S_F$, $i = 0$
repeat
  if $W_i \cap S_0 \neq \emptyset$
    return "S_F reachable"
  end if
  $W_{i+1} = \mathrm{P}re(W_i) \cup W_i$
  $i = i + 1$
until $W_i = W_{i-1}$
return "S_F not reachable"

Using an induction argument it is easy to show that if the algorithm *terminates* (i.e. at some point it returns "S_F reachable" or "S_F not reachable") then it produces the right answer.

Exercise 6.1 Show this.

Exercise 6.2 Define an appropriate operator Post : $P(S) \to P(S)$ that for each set of states $\hat{S} \subseteq S$ returns the set of states that can be reached from $\hat{S}$ in one transition. Hence develop a forward reachability algorithm.

This algorithm is written in what is known as *pseudo-code*. It is conceptually useful, but is still a long way from being implementable on a computer. To effectively implement the algorithm one needs to figure out a way to explain to the computer how to

1. store sets of states,
2. compute the Pre of a set of states,
3. take the union and intersection of sets of states,
4. check whether a set of states is empty, and
5. check whether two sets of states are equal.

If the number of states S is finite all of these are relatively easy to do by enumerating the states. None of these steps are completely straight forward, however, if the state has real valued components (as is the case with hybrid systems).

Even if one was able to perform all of these operations using a computer program, it is still unclear whether the program would always produce an answer to the reachability question. The above algorithm may come up with the answer "S_F reachable", the answer "S_F not reachable", but it may also come up with no answer at all. This will be the case if new states get added to W_i each time we go around the repeat-until loop (hence $W_{i+1} \neq W_i$) but none of these states belongs to S_0 (hence $W_{i+1} \cap S_0 = \emptyset$).

Example (Non-Terminating System) Consider the transition system $T = (S, \delta, S_0, S_F)$ with $S = \mathbb{R}$,

$$\delta(x) = 2x$$

$S_0 = \{-1\}$, $S_F = \{1\}$. The Backwards Reachability algorithm produces the following sequence of sets:

$$W_0 = \{1\}, W_1 = \{1, \frac{1}{2}\}, \ldots, W_i = \{1, \frac{1}{2}, \ldots, \left(\frac{1}{2}\right)^i\}, \ldots$$

Notice that W_{i+1} contains one more element that W_i, therefore we will never have $W_i \neq W_{i+1}$. Moreover, -1 will never be in W_i, therefore $W_i \cap S_0 = \emptyset$. Hence the algorithm will not terminate. ■

With finite state systems termination is not a problem: the set W_i will sooner or later stop growing.

Example (Finite State Transition System (cont.)) When applied to the finite state system of Figure 6.1 the Backwards Reachability Algorithm produces the following sequence of sets:

$$W_0 = \{q_3, q_6\}, W_1 = \{q_1, q_2, q_3, q_6\}, W_2 = \{q_0, q_1, q_2, q_3, q_6\}$$

Notice that $W_2 \cap S_0 = \{q_0\} \neq \emptyset$. Therefore, after two steps the algorithm terminates with the answer "S_F reachable". ■

Exercise 6.3 Assume the set S is finite and contains M states. Give an upper bound on the number of times the algorithm will have to perform the "repeat-until" loop.

6.2 Bisimulation

Since finite state systems are so much easier to work with, we are going to try to turn our infinite state systems into finite state ones, by grouping together states that have "similar" behaviour. Such a grouping of states is called a *partition* of the state space. A partition is a collection of sets of states, $\{S_i\}_{i \in I}$, with $S_i \subseteq S$, such that

1. Any two sets, S_i and S_j, in the partition are disjoint, i.e. $S_i \cap S_j = \emptyset$ for all $i, j \in I$ with $i \neq j$. (A family of sets with this property is called *mutually disjoint*).
2. The union of all sets in the partition is the entire state space, i.e.
$$\bigcup_{i \in I} S_i = S$$
(A family of sets with this property is said to *cover* the state space).

The index set, I, of the partition may be either finite or infinite. If I is a finite set (e.g. $I = \{1, 2, \ldots, M\}$ for $M < \infty$) then we say that the partition $\{S_i\}_{i \in I}$ is a *finite partition*.

Example (Finite State Transition System (cont.)) The collection of sets
$$\{q_0\}, \{q_1, q_2\}, \{q_3, q_6\}, \{q_4, q_5\}$$
is a partition of the state space S of the finite system of Figure 6.1. The collection
$$\{q_0\}, \{q_1, q_3, q_4\}, \{q_2, q_5, q_6\}$$
is also a partition. However, the collection
$$\{q_1, q_3, q_4\}, \{q_2, q_5, q_6\}$$
is not a partition, and neither is
$$\{q_0, q_1, q_3, q_4\}, \{q_0, q_2, q_5, q_6\}$$
■

Given a transition system, $T = (S, \delta, S_0, S_F)$ and a partition of the state space $\{S_i\}_{i \in I}$ we can define a transition system whose states are the elements of the partition $S_i \subseteq S$, rather than individual states $s \in S$. This transition system $\hat{T} = (\hat{S}, \hat{\delta}, \hat{S}_0, \hat{S}_F)$ is defined as

- $\hat{S} = \{S_i\}_{i \in I}$, i.e. the states are the sets of the partition;
- $\hat{\delta}$ allows a transition from one set in the partition (say S_i) to another (say S_j) if and only if δ allows a transition from some state in S_i (say $s \in S_i$) to some state in S_j (say $s' \in S_j$). In mathematical notation,
$$\hat{\delta}(S_i) = \{S_j \mid \exists s \in S_i, \exists s' \in S_j \text{ such that } s' \in \delta(s)\}$$
- A set in the partition (say S_i) is in the initial set of $\hat{T}$ (i.e. $S_i \in \hat{S}_0$) if and only if some element of S_i (say $s \in S_i$) is an initial state of the original transition system (i.e. $s \in S_0$).
- A set in the partition (say S_i) is a final set of $\hat{T}$ (i.e. $S_i \in \hat{S}_F$) if and only if some element of S_i (say $s \in S_i$) is a final state of the original transition system (i.e. $s \in S_F$).

Exercise 6.4 Show that the above definitions are equivalent to $\hat{\delta}(S_i) = \{S_j \mid \delta(S_i) \cap S_j \neq \emptyset\}$, $S_i \in \hat{S}_0 \Leftrightarrow S_i \cap S_0 \neq \emptyset$, $S_i \in \hat{S}_F \Leftrightarrow S_i \cap S_F \neq \emptyset$. You need to define $\delta(S_i)$ appropriately, but otherwise this is a tautology.

The transition system generated by the partition is known as the *quotient transition system*. Notice that if the partition is finite, then the quotient transition system $\hat{T}$ is a finite state system and therefore can be easily analysed.

Using this method we can in principle construct finite state systems out of any transition system. The problem is that for most partitions the properties of the quotient transition system do not allow us to draw any useful conclusions about the properties of the original system. However, there is a special type of partition for which the quotient system $\hat{T}$ is in a sense equivalent to the original transition system, T. This type of partition is known as a *bisimulation*.

Definition 6.2 (Bisimulation) *A bisimulation of a transition system $T = (S, \delta, S_0, S_F)$ is a partition $\{S_i\}_{i \in I}$ of the state space S of T such that*

1. *S_0 is a union of elements of the partition,*
2. *S_F is a union of elements of the partition,*
3. *if one state (say s) in some set of the partition (say S_i) can transition to another set in the partition (say S_j), then all other states, $\hat{s}$ in S_i must be able to transition to some state in S_j. More formally, for all $i, j \in I$ and for all states $s, \hat{s} \in S_i$, if $\delta(s) \cap S_j \neq \emptyset$, then $\delta(\hat{s}) \cap S_j \neq \emptyset$.*

Exercise 6.5 Show that a partition $\{S_i\}_{i \in I}$ is a bisimulation if and only if conditions 1 and 2 above hold and 3 is replaced by "for all i, $Pre(S_i)$ is a union of elements of $\{S_i\}_{i \in I}$".

Example (Finite Transition System (cont.)) The partition

$$\{q_0\}, \{q_1, q_2\}, \{q_3, q_6\}, \{q_4, q_5\}$$

is a bisimulation of the finite system of Figure 6.1. Let us test this:

1. $S_0 = \{q_0\}$ which is an element of the partition;
2. $S_F = \{q_3, q_6\}$ which is also an element of the partition;
3. Let us study the third condition for the set $\{q_1, q_2\}$. From q_1 one can jump to the following sets in the partition
$$\{q_0\}, \{q_3, q_6\}, \{q_4, q_5\}$$
From q_2 one can jump to exactly the same sets in the partition. Therefore the third condition is satisfied for set $\{q_1, q_2\}$. It is also easy to check this condition for the remaining sets (for the set $\{q_0\}$ the third condition is trivially satisfied).

The partition

$$\{q_0\}, \{q_1, q_3, q_4\}, \{q_2, q_5, q_6\}$$

on the other hand, is not a bisimulation. For example, S_F is not a union of elements of the partition. Also, from q_1 one can transition to q_0, whereas from q_3 and q_4 (the other elements of the set $\{q_1, q_3, q_4\}$) one cannot transition to q_0. ■

Bisimulations are important because of the following property.

Theorem 6.1 *Let $\{S_i\}_{i \in I}$ be a bisimulation of a transition system, T, and let $\hat{T}$ be the quotient transition system. S_F is reachable by T if and only if $\hat{S}_F$ is reachable by $\hat{T}$.*

(NB. This is a simplified version of a much deeper theorem. In fact bisimulations preserve not only reachability properties, but any kind of property that can be written as a formula in a temporal logic known as the Computation Tree Logic (CTL).)

This is a very important and useful result. For finite state systems its implications are mostly in terms of computational efficiency. If we can find a bisimulation of the finite state system (like we did in the finite state example discussed above) we can study reachability in the quotient system instead of the original system. The advantage is that the quotient system will in general be much smaller that the original system. In the above example, the quotient system had 4 states whereas the original system had 7.

The implications are much more profound for infinite state systems. Even when the original transition system has an infinite number of states, sometimes we may be able to find a bisimulation that

consists of a finite number of sets. Then we will be able to answer reachability questions for the infinite state system by studying an equivalent finite state system. Since finite state systems are so much easier to work with this could be a very big advantage. A class of hybrid systems for which we can always find finite bisimulations will be introduced in the next section.

The following algorithm can be used to find a bisimulation of a transition system $T = (S, \delta, S_0, S_F)$.

Algorithm 2 (Bisimulation)
initialisation: $\mathcal{P} = \{S_0, S_F, S \setminus (S_0 \cup S_F)\}$
while there exists $S_i, S_j \in \mathcal{P}$ such that $S_i \cap \mathrm{P}re(S_j) \neq \emptyset$ and $S_i \cap \mathrm{P}re(S_j) \neq S_i$ **do**
$S_i' = S_i \cap \mathrm{P}re(S_j)$
$S_i'' = S_i \setminus \mathrm{P}re(S_j)$
$\mathcal{P} = (\mathcal{P} \setminus S_i) \cup \{S_i', S_i''\}$
end while
return $\mathcal{P}$

The symbol $\setminus$ in the algorithm stands for set difference: $S_i \setminus \mathrm{P}re(S_j)$ is the set that contains all elements of S_i that are not elements of $\mathrm{P}re(S_j)$, in other words

$$S_i \setminus \mathrm{P}re(S_j) = \{s \in S \mid (s \in S_i) \wedge (s \notin \mathrm{P}re(S_j))\}$$

The algorithm maintains a partition of the state space, denoted by $\mathcal{P}$, which gets refined progressively so that it looks more and more like a bisimulation. The definition of the bisimulation suggests that if $\mathcal{P}$ is to be a bisimulation then it must at least allow us to "distinguish" the initial and final states. We therefore start with a partition that contains three sets: S_0, S_F and everything else $S \setminus (S_0 \cup S_F)$, i.e.

$$\mathcal{P} = \{S_0, S_F, S \setminus (S_0 \cup S_F)\}$$

At each step of the algorithm, we examine the sets of the candidate partition. Assume we can find two sets $S_i, S_j \in \mathcal{P}$ such that $\mathrm{P}re(S_j)$ contains some elements of S_i (i.e. $S_i \cap \mathrm{P}re(S_j) \neq \emptyset$) but not all of them (i.e. $S_i \not\subseteq \mathrm{P}re(S_j)$, or, equivalently, $S_i \cap \mathrm{P}re(S_j) \neq S_i$). Then some states $s \in S_i$ may find themselves in S_j after one transition (namely those with $s \in S_i \cap \mathrm{P}re(S_j)$), while others cannot do the same (namely those with $s \in S_i \setminus \mathrm{P}re(S_j)$). This is not allowed if $\mathcal{P}$ is to be a bisimulation. We therefore replace S_i by two sets: the states in S_i that can transition to S_j ($S_i' = S_i \cap \mathrm{P}re(S_j)$) and the states in S_i that cannot transition to S_j ($S_i'' = S_i \setminus \mathrm{P}re(S_j)$). Notice that after the replacement $\mathcal{P}$ has one more set than it did before. The process is repeated until for all sets $S_i, S_j \in \mathcal{P}$ either $S_i \cap \mathrm{P}re(S_j) \neq \emptyset$ or $S_i \cap \mathrm{P}re(S_j) \neq S_i$. The resulting collection of sets, $\mathcal{P}$ is a bisimulation. In fact:

Theorem 6.2 *If the Bisimulation Algorithm terminates it will produce the coarsest bisimulation of the transition system (i.e. the bisimulation containing the smallest number of sets).*

For finite state systems this algorithm is easy to implement (by enumerating the states) and will always terminate.

Example (Finite State Transition System (cont.)) Let us apply the bisimulation algorithm to the finite system of Figure 6.1. Initially

$$\mathcal{P} = \{S_0, S_F, S \setminus (S_0 \cup S_F)\} = \{\{q_0\}, \{q_3, q_6\}, \{q_1, q_2, q_4, q_5\}\}$$

Notice that $\mathrm{P}re(\{q_3, q_6\}) = \{q_1, q_2\}$. This is not an element of the partition $\mathcal{P}$. It intersects $\{q_1, q_2, q_4, q_5\}$ but is not equal to it. We therefore split $\{q_1, q_2, q_4, q_5\}$ into two sets

$$\{q_1, q_2, q_4, q_5\} \cap \mathrm{P}re(\{q_3, q_6\}) = \{q_1, q_2\}$$

and

$$\{q_1, q_2, q_4, q_5\} \setminus \mathrm{P}re(\{q_3, q_6\}) = \{q_4, q_5\}$$

After one iteration of the algorithm the partition becomes

$$\mathcal{P} = \{\{q_0\}, \{q_3, q_6\}, \{q_1, q_2\}, \{q_4, q_5\}\}$$

It is easy to check that this is a bisimulation. Clearly S_0 and S_F are elements of the partition. Moreover,

$$\mathrm{P}re(\{q_0\}) = \{q_0, q_1, q_2\}$$

which is a union of elements of the partition, and so on. ■

The problem with using the bisimulation algorithm on finite state systems is that it may be more work to find a bisimulation than to investigate the reachability of the original system. Sometimes bisimulations can be computed by "inspection", by taking advantage of symmetries of the transition structure. In the above example, we can immediately see that the left sub-tree is a mirror image of the right sub-tree. This should make us suspect that there is a bisimulation lurking somewhere in the system. There is an entire community in computer science that develops methods for detecting and exploiting such symmetries.

When we try to apply the bisimulation algorithm to infinite state systems we face the same problems we did with the Backward Reachability algorithm: how to store sets of states in the computer, how to compute Pre, etc. Moreover, even in cases where we can do all these, the algorithm may never terminate. The reason is that not all infinite state transition systems have finite bisimulations. In the next section we will introduce a class of (infinite state) hybrid systems for which we can not only implement the above algorithm in a computer, but also ensure that it will terminate in a finite number of steps.

6.3 Timed Automata

Timed automata are a class of hybrid systems that involve particularly simple continuous dynamics: all differential equations are of the form $\dot{x} = 1$ and all the domains, guards, etc. involve comparison of the real valued states with constants ($x = 1$, $x < 2$, $x \geq 0$, etc.). Clearly timed automata are somewhat limited when it comes to modelling physical systems. They are very good however for encoding timing constraints ("event A must take place at least 2 seconds after event B and not more than 5 seconds before event C", etc.). For some applications, such as multimedia, Internet and audio protocol verification, etc. this type of description is sufficient for both the dynamics of the system and the properties that we want the system to satisfy. We conclude this chapter with a brief discussion of the properties of timed automata. Because complicated mathematical notation is necessary to formally introduce the topic we will give a rather informal exposition. Students interested in the details are referred to the (rather technical but classic) paper [2].

Consider $x \in \mathbb{R}^n$. A subset of $\mathbb{R}^n$ set is called *rectangular* if it can be written as a finite boolean combination of constraints of the form $x_i \leq a$, $x_i < b$, $x_i = c$, $x_i \geq d$, and $x_i > e$, where a, b, c, d, e are rational numbers. Roughly speaking, rectangular sets are "rectangles" in $\mathbb{R}^n$ whose sides are aligned with the axes, or unions of such rectangles. For example, in $\mathbb{R}^2$ the set

$$(x_1 \geq 0) \wedge (x_1 \leq 2) \wedge (x_2 \geq 1) \wedge (x_2 \leq 2)$$

is rectangular, and so is the set

$$((x_1 \geq 0) \wedge (x_2 = 0)) \vee ((x_1 = 0) \wedge (x_2 \geq 0))$$

The empty set is also a rectangular set (e.g. $\emptyset = (x_1 \geq 1) \wedge (x_1 \leq 0)$). However the set

$$\{x \in \mathbb{R}^2 \mid x_1 = 2x_2\}$$

is not rectangular.

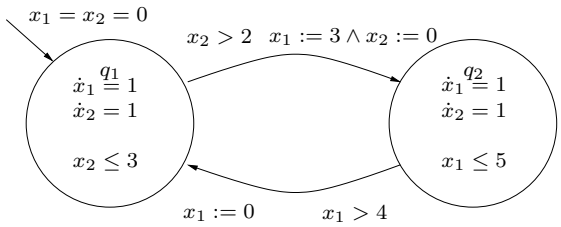

Figure 6.2: Example of a timed automaton.

Exercise 6.6 Draw these sets in $\mathbb{R}^2$. You should immediately be able to see why rectangular sets are called rectangular.

Notice that rectangular sets are easy to encode in a computer. Instead of storing the set itself (which is impossible since the set is infinite) we can store and manipulate the list of constraints used to generate the set (which is finite).

Roughly speaking, a timed automaton is a hybrid automaton which

- involves differential equations of the form $\dot{x}_i = 1$. Continuous variables governed by this differential equation are known as *clocks*.
- the sets involved in the definition of the initial states, guards and domain are rectangular sets
- the reset is either a rectangular set, or may leave certain states unchanged.

Example (Timed Automaton) An example of a timed automaton is given in Figure 6.2. We have

- $Q = \{q_1, q_2\}$;
- $X = \mathbb{R}^2$;
- $f(q_1, x) = f(q_1, x) = \begin{bmatrix} 1 \\ 1 \end{bmatrix}$;
- $Init = \{(q_1, (0, 0))\}$;
- $Dom(q_1) = \{x \in \mathbb{R}^2 \mid x_2 \leq 3\}$, $Dom(q_2) = \{x \in \mathbb{R}^2 \mid x_1 \leq 5\}$;
- $E = \{(q_1, q_2), (q_2, q_1)\}$;
- $G(q_1, q_2) = \{x \in \mathbb{R}^2 \mid x_2 > 2\}$, $G(q_2, q_1) = \{x \in \mathbb{R}^2 \mid x_1 > 4\}$;
- $R(q_1, q_2, x) = \{(3, 0)\}$, $R(q_2, q_1, x) = \{(0, x_2)\}$

■

Exercise 6.7 Is this timed automaton non-blocking? Is it deterministic?

Notice that in the timed automaton of the example all the constants that appear in the definition are non-negative integers. It turns out that we can in general assume that this is the case: given any timed automaton whose definition involves rational and/or negative constants we can define an equivalent timed automaton whose definition involves only non-negative integers. This is done

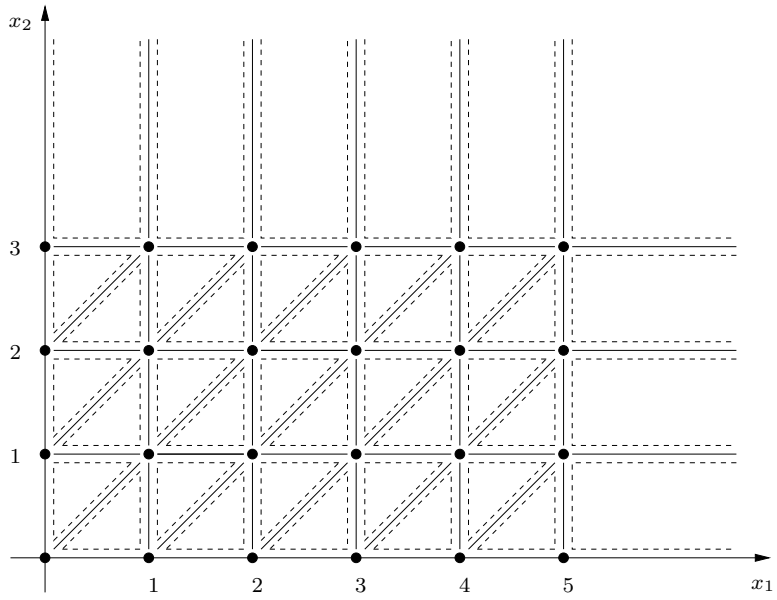

Figure 6.3: Region graph for the timed automaton of Figure 6.2.

by "scaling" (multiplying by an appropriate integer) and "shifting" (adding an appropriate integer) some of the states.

We can turn timed automata into transition systems by "abstracting away" time, just like we did for general hybrid systems above. It turns out that the transition system corresponding to a timed automaton always has a finite bisimulation. One standard bisimulation that works for all timed automata is the *region graph*. The region graph for the timed automaton of Figure 6.2 is shown in Figure 6.3.

We will briefly describe the way the region graph is constructed. Assume that all the constants that appear in the definition of the timed automaton are non-negative integers (this can be done without loss of generality as noted above). As usual, let us label the continuous variables (clocks) as $x_1, \ldots, x_n$. Let C_i be the largest constant with which x_i is compared in any of the sets used in the definition of the timed automaton (initial sets, guards, etc.). In the example $C_1 = 5$ and $C_2 = 3$. If all we know about the timed automaton is these bounds C_i, x_i could be compared with any integer M with $0 \leq M \leq C_i$ in some guard, reset or initial condition set. Therefore, the discrete transitions of the timed automaton may be able to "distinguish" states with $x_i < M$ from states with $x_i = M$ and from states with $x_i > M$, for all $0 \leq M \leq C_i$. Distinguish means, for example, that a discrete transition may be possible from a state with $x_i < M$ but not from a state with $x_i > M$ (because the guard contains the condition $x_i < M$). Because these sets may be distinguished by the discrete transitions we add them to our candidate bisimulation. In the example this gives rise to the sets

$$\begin{aligned}
\text{for } x_1 : & \; x_1 \in (0,1), x_1 \in (1,2), x_1 \in (2,3), x_1 \in (3,4), x_1 \in (4,5), x_1 \in (5,\infty) \\
& \; x_1 = 0, x_1 = 1, x_1 = 2, x_1 = 3, x_1 = 4, x_1 = 5, \\
\text{for } x_2 : & \; x_2 \in (0,1), x_2 \in (1,2), x_2 \in (2,3), x_2 \in (3,\infty) \\
& \; x_2 = 0, x_2 = 1, x_2 = 2, x_2 = 3.
\end{aligned}$$

The product of all these sets, i.e. the sets

$$\{x \in \mathbb{R}^2 \mid x_1 \in (0,1) \wedge x_2 \in (0,1)\}$$
$$\{x \in \mathbb{R}^2 \mid x_1 \in (0,1) \wedge x_2 = 1\}$$
$$\{x \in \mathbb{R}^2 \mid x_1 = 1 \wedge x_2 \in (0,1)\}$$
$$\{x \in \mathbb{R}^2 \mid x_1 = 1 \wedge x_2 = 1\}$$
$$\{x \in \mathbb{R}^2 \mid x_1 \in (1,2) \wedge x_2 \in (3\infty)\}, \textit{etc.}$$

define all the sets in $\mathbb{R}^2$ that the discrete dynamics (initial states, guards, domain and reset relations) can distinguish. Notice that in the continuous state space $\mathbb{R}^2$ these product sets are open squares (squares without their boundary), open horizontal and vertical line segments (line segments without their end points), points on the integer grid and open, unbounded rectangles (when some $x_i > C_i$).

Since $\dot{x}_1 = \dot{x}_2 = 1$, time flow makes the continuous state move diagonally up along 45° lines. By allowing time to flow the timed automaton may therefore distinguish points below the diagonal of each square, points above the diagonal and points on the diagonal. For example, points above the diagonal of the square

$$\{x \in \mathbb{R}^2 \mid x_1 \in (0,1) \wedge x_2 \in (0,1)\}$$

will leave the square through the line

$$\{x \in \mathbb{R}^2 \mid x_1 \in (0,1) \wedge x_2 = 1\}$$

points below the diagonal will leave the square through the line

$$\{x \in \mathbb{R}^2 \mid x_1 = 1 \wedge x_2 \in (0,1)\}$$

while points on the diagonal will leave the square through the point

$$\{x \in \mathbb{R}^2 \mid x_1 = 1 \wedge x_2 = 1\}$$

This leads us to split each open square in three: two open triangles and an open diagonal line segment.

It can in fact be shown that this is enough to generate a bisimulation.

Theorem 6.3 *The region graph is a finite bisimulation of the timed automaton.*

It should be stressed that the region graph is not necessarily the coarsest bisimulation. One may be able to find bisimulations with fewer sets; the elements of these bisimulation will be unions of sets of the region graph. It is generally considered bad form to try to construct the entire region graph to investigate properties of a timed automaton. Usually, one either uses the bisimulation algorithm to construct a coarse bisimulation, or uses the reachability algorithm directly to investigate reachability. The point of Theorem 6.3 is that it guarantees that there is at least one finite bisimulation for each timed automaton, which in turn guarantees that the Bisimulation and Reachability algorithms can be implemented and will terminate.

A counting exercise reveals that the total number of regions in the region graph is of the order of

$$n!\, 2^n \prod_{i=1}^{n} (2C_i + 2)$$

(! denotes factorial.) Even for relatively small n this is a very large number! (! denotes exclamation point) What is worse, the number grows very quickly (exponentially) as n increases. (In addition to $n!$ and 2^n, there is another hidden exponential in this formula. Because on a computer numbers are stored in binary, C_i is exponential in the number of bits used to store it).

This is bad news. It implies that a relatively simple timed automaton can give rise to a region graph with a very large number of sets, which will be a nightmare to analyse. It turns out that this is a general property of timed automata and has nothing to do with the way the region graph was constructed. Because timed automata have finite bisimulations, we know that they can be automatically analysed by a computer. However, in the worst case, the number of operations that the computer will have to perform to analyse the system will be exponential in the size of the problem (e.g. the length of the input file we need to define our timed automaton for the computer). This is irrespective of how well we write the programme. In computational complexity terminology, model checking for timed automata turns out to be *PSPACE Complete*.

A second bit of bad news about the model checking method is that it does not work for more complicated systems. For example, one can show that simple variants of timed automata do not have finite bisimulations. This is the case for example if we allow $\dot{x}_i = c_i$ for some constant $c_i \neq 1$ (*skewed clocks*, leading to *multi-rate automata*), or allow comparisons between clocks (terms of the form $x_i \leq x_j$), or reseting one clock to another ($x_i := x_j$). In computer theory terminology, the reachability question for such systems is *undecidable*, i.e. there does not exist an algorithm that will answer the question in finite time.

6.4 Bibliography and Further Reading

Timed automata were the first class of hybrid systems that were shown to be amenable to model checking methods [2]. Since then a number of other classes of hybrid systems with this property have been established: classes of multi-rate automata [1], classes of systems with continuous dynamics governed by constant differential inclusions [41] and classes of systems with continuous dynamics governed by linear differential equations [55]. It has also been shown that a very wide class of hybrid systems can be approximated arbitrarily closely by such "decidable" hybrid systems [86] (albeit at the cost of exponential computational complexity). For an excellent overview of the developments in this area see [5].

In the case of hybrid control systems, related methods have been developed for automatically synthesising controllers to satisfy specifications (e.g. given in temporal logic) whenever possible [69, 104, 11].

Based on the theoretical results, computational tools been developed to automatically perform verification or synthesis tasks [42, 6, 29, 16].

Chapter 7

Reachability with Inputs: A Viability Theory Perspective

7.1 Reachability with Inputs

In addition to discrete and continuous state variables, many hybrid systems also contain input variables U, possibly divided into discrete (U_D) and continuous (U_C). Depending on the application, inputs may influence

1. Continuous evolution, through the vector field
$$f(\cdot,\cdot,\cdot) : Q \times X \times U \to \mathbb{R}^n.$$
2. When discrete transitions take place, through the domain
$$\mathrm{D}om(\cdot) : Q \to 2^{X \times U}.$$
3. The destination of discrete transitions, through the reset map
$$R(\cdot,\cdot,\cdot) : E \times X \times U \to 2^X.$$

One can pause a number of interesting problems for hybrid systems with inputs, that make no sense for autonomous hybrid systems. For example, one can study stabilisation, optimal control, reachability specifications, etc. As before we will restrict our attention to reachability specifications. Depending on what the inputs are supposed to represent, reachability questions for hybrid systems with inputs can take a number of forms:

1. **Viability**. Roughly speaking, this involves answering the question "Does there exist a choice for the inputs u such that the executions of the hybrid system remain in a given set?". In this case one can think of the inputs as controls, that can be used to steer the executions of the system.
2. **Invariance**. This involves answering the question "Do the executions of the system remain in a given set for all choices of u?". In this case one can think of the inputs as uncontrollable disturbances that can steer the system outside the desired set.
3. **Gaming**. In this case some of the input variables play the role of controls, while others play the role of disturbances. The relevant question in this case is "Does there exist a choice for the controls, such that despite the action of the disturbances the execution of the system remain is a given set?"

Working with hybrid systems with inputs is considerably more complicated. Even defining the precise semantics of an execution of the system is far from straight forward. Additional complications arise when one considers gaming problems, since one has to introduce assumptions about information exchange among the players, appropriate notions of strategy, etc. Here we will restrict our attention to a special case of the general problem for which precise mathematical answers can be formulated for these questions. In particular, we will study questions of viability and invariance for a class of hybrid systems known as *impulse differential inclusions*. The proofs are rather technical and are included in the notes only for the sake of completeness. For more details please refer to [14].

7.2 Impulse Differential Inclusions

Definition 7.1 (Impulse Differential Inclusion) *An impulse differential inclusion is a collection $H = (X, F, R, J)$, consisting of a finite dimensional vector space X, a set valued map $F : X \to 2^X$, regarded as a* differential inclusion $\dot{x} \in F(x)$*, a set valued map $R : X \to 2^X$, regarded as a* reset map*, and a set $J \subseteq X$, regarded as a* forced transition set*.*

We call $x \in X$ the *state* of the impulse differential inclusion. Subsequently, $I = X \setminus J$ will be used to denote the complement of J. Comparing Definition 7.1 with Definition 3.1, we see that the set I plays a similar role for the impulse differential inclusion that D*om* played for hybrid automata. The differential inclusion, $F(\cdot)$, plays a similar role to the vector field, $f(\cdot, \cdot)$. Finally, the domain of the reset map R plays the same role as the guards G of a hybrid automaton, and the image of the reset map R plays the same role as the reset relation (also denoted by R) of the hybrid automaton. Note that the initial states are not explicitly mentioned, and that there are no discrete states. Discrete states, q, can be introduced into the model, by embedding them in $\mathbb{R}$ (e.g., $q = 1, 2, 3, \ldots$) and introducing a trivial differential inclusion $\dot{q} \in \{0\}$ to capture their continuous evolution. This is somewhat cumbersome to do, so we will state the results without this additional complication.

Impulse differential inclusions are extensions of differential inclusions and discrete time systems over finite dimensional vector spaces. A differential inclusion,

$$\dot{x} \in F(x),$$

over a finite dimensional vector space X can be thought of as an impulse differential inclusion, (X, F, R, J), with $R(x) = \emptyset$ for all $x \in X$ and $J = \emptyset$. Likewise, a discrete time system,

$$x_{k+1} \in R(x_k),$$

can be thought of as an impulse differential inclusion, $H = (X, F, R, J)$, with $F(x) = \{0\}$ for all $x \in X$ and $J = X$. In the control literature, differential inclusions and discrete time systems are frequently used to model continuous and discrete control systems. The continuous control system

$$\dot{x} = f(x, u), \quad u \in U(x)$$

with $x \in \mathbb{R}^n$, $u \in \mathbb{R}^m$, $f : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $U : \mathbb{R}^n \to 2^{\mathbb{R}^m}$ can be though of as a differential inclusion

$$\dot{x} \in F(x) = \{f(x, u) \mid u \in U(x)\}.$$

Likewise, the discrete time control system

$$x_{k+1} = r(x_k, u_k), \quad u_k \in U(x_k)$$

with $x_k \in \mathbb{R}^n$, $u_k \in \mathbb{R}^m$, $r : \mathbb{R}^n \times \mathbb{R}^m \to \mathbb{R}^n$ and $U : \mathbb{R}^n \to 2^{\mathbb{R}^m}$ can be though of as

$$x_{k+1} \in R(x_k) = \{r(x_k, u) \mid u \in U(x_k)\}.$$

Therefore, impulse differential inclusions can be thought of as hybrid control systems, with controls both on the continuous evolutions and on the discrete transitions.

Impulse differential inclusions can be used to describe hybrid phenomena. As for the executions of a hybrid automaton, the state of an impulse differential inclusion evolves over hybrid time sets.

Definition 7.2 (Run of an Impulse Differential Inclusion) *a run of an impulse differential inclusion, $H = (X, F, R, J)$, is a hybrid trajectory, (τ, x), consisting of a hybrid time set $\tau = \{I_i\}_{i=0}^N$ and a sequence of maps $x = \{x_i\}_{i=0}^N$, $x_i(\cdot) : I_i \to X$, that satisfies:*

- Discrete Evolution*: for all i, $x_{i+1}(\tau_{i+1}) \in R(x_i(\tau_i'))$*
- Continuous Evolution*: if $\tau_i < \tau_i'$, $x_i(\cdot)$ is a solution to the differential inclusion $\dot{x} \in F(x)$ over the interval $[\tau_i, \tau_i']$ starting at $x_i(\tau_i)$, with $x_i(t) \notin J$ for all $t \in [\tau_i, \tau_i'[$.*

Recall that a *solution* to the differential inclusion $\dot{x} \in F(x)$ over an interval $[0, T]$ starting at $x_0 \in X$ is an absolutely continuous function $x : [0, T] \to X$, such that $x(0) = x_0$ and almost everywhere $\dot{x}(t) \in F(x(t))$. As for hybrid automata, runs of impulse differential inclusions can be classified into finite, infinite, Zeno, etc. (cf. Definition 3.5).

Definition 7.2 dictates that, along a run the state can evolve continuously according to the differential inclusion $\dot{x} \in F(x)$ until the set J is reached. Moreover, whenever $R(x) \neq \emptyset$, a discrete transition from state x to some state in $R(x)$ may take place. In other words R *enables* discrete transitions (transitions may happen when $R(x) \neq \emptyset$ but do not have to), while J *forces* discrete transitions (transitions must happen when $x \in J$).

Notice that if at a state $x \in X$ a transition must happen ($x \in J$) but is not able to ($R(x) = \emptyset$) the system *blocks*, in the sense that there does not exist a run of the impulse differential inclusion starting at x (other than the trivial run $([0, 0], x)$). This is similar to the blocking situations (lack of runs) that we encountered for hybrid automata. To prevent such behaviour we introduce the following assumption.

Assumption 7.1 *An impulse differential inclusion (X, F, R, J) is said to satisfy* Assumption 7.1 *if $J \subseteq R^{-1}(X)$ and, if J is open (hence $I = X \setminus J$ is closed), $F(x) \cap T_I(x) \neq \emptyset$, for all $x \in I \setminus R^{-1}(X)$.*

As an example, consider again the bouncing ball system. The vertical motion of the ball can be captured by an impulse differential inclusion, $H_B = (X_B, F_B, R_B, J_B)$ with two state variables, the height of the ball, x_1 and its velocity in the vertical direction, x_2. Therefore, $X_B = \mathbb{R}^2$ and

$$\begin{aligned} F_B(x_1, x_2) =& (x_2, -g) \\ R_B(x_1, x_2) =& \begin{cases} (x_1, -cx_2) & \text{if } x_1 \leq 0 \text{ and } x_2 \leq 0 \\ \emptyset & \text{otherwise} \end{cases} \\ J_B =& \{x \in X_B \mid x_1 \leq 0 \text{ and } x_2 \leq 0\}, \end{aligned}$$

where g represents the acceleration due to gravity and $c^2 \in [0, 1]$ the fraction of energy lost with each bounce.

7.3 Viability and Invariance Definitions

For impulse differential inclusions, reachability questions can be characterised by viability constraints.

Definition 7.3 (Viable Run) *a run, (τ, x) of an impulse differential inclusion, $H = (X, F, R, J)$, is called viable in a set $K \subseteq X$ if $x(t) \in K$ for all $I_i \in \tau$ and all $t \in I_i$.*

Notice that the definition of a viable run requires the state to remain in the set K throughout, along continuous evolution up until and including the state before discrete transitions, as well as after discrete transitions. Based on the notion of a viable run, one can define two different classes of sets.

Definition 7.4 (Viable and Invariant Set) *A set $K \subseteq X$ is called viable under an impulse differential inclusion, $H = (X, F, R, J)$, if for all $x_0 \in K$ there exists an infinite run starting at x_0 that is viable in K. K is called invariant under the impulse differential inclusion, if for all $x_0 \in K$ all runs starting at x_0 are viable in K.*

In the cases where an impulse differential inclusion fails to satisfy a given viability or invariance requirement, one would like to establish sets of initial conditions (if any) for which the requirement will be satisfied. This notion can be characterised in terms of viability and invariance kernels.

Definition 7.5 (Viability and Invariance Kernel) *The viability kernel, $\mathrm{V}iab_H(K)$ of a set $K \subseteq X$ under an impulse differential inclusion $H = (X, F, R, J)$ is the set of states $x_0 \in X$ for which there exists an infinite run viable in K. The invariance kernel, $\mathrm{I}nv_H(K)$ of $K \subseteq X$ under $H = (X, F, R, J)$ is the set of states $x_0 \in X$ for which all runs are viable in K.*

Notice that by definition $\mathrm{V}iab_H(K) \subseteq K$ and $\mathrm{I}nv_H(K) \subseteq K$, but in general the two sets are incomparable.

To state viability and invariance results for impulse differential inclusions we need to introduce some technical definitions from set valued analysis. For more details see [12, 13].

For a set valued map $R : X \to 2^Y$ and a set $K \subseteq Y$ we use $R^{-1}(K)$ to denote the *inverse image* of K under R and $R^{\ominus 1}(K)$ to denote the *extended core* of K under R, defined by

$$R^{-1}(K) = \{x \in X \mid R(x) \cap K \neq \emptyset\}, \quad \text{and}$$
$$R^{\ominus 1}(K) = \{x \in X \mid R(x) \subseteq K\} \cup \{x \in X \mid R(x) = \emptyset\}.$$

Notice that $R^{-1}(Y)$ is the set of $x \in X$ such that $R(x) \neq \emptyset$. We call the set $R^{-1}(Y)$ the *domain* of R and the set $\{(x, y) \in X \times Y \mid y \in R(x)\}$ the *graph* of R.

A set valued map $R : X \to 2^X$ is called *upper semicontinuous at $x \in X$* if for every $\epsilon > 0$ there exists $\delta > 0$ such that

$$\forall\, x' \in B(x, \delta), \;\; R(x') \subseteq B(R(x), \epsilon).$$

R is called *lower semicontinuous at $x \in X$* if for all $x' \in R(x)$ and for all sequences x_n converging to x, there exists a sequence $x'_n \in R(x_n)$ converging to x'. R is called *upper semicontinuous* (respectively *lower semicontinuous*) if it is upper semicontinuous (respectively lower semicontinuous) at all $x \in X$. It should be noted that, unlike single valued functions, these two notions of continuity are not equivalent for set valued maps. It can be shown that if R is upper semicontinuous with closed domain and $K \subseteq X$ is a closed set, then $R^{-1}(K)$ is closed, whereas if R is lower semicontinuous and $U \subseteq X$ is an open set, then $R^{-1}(U)$ is open. Notice that the last statement also implies that if R is lower semicontinuous and $K \subseteq X$ is closed, $R^{\ominus 1}(K)$ is closed, since $R^{\ominus 1}(K) = X \setminus R^{-1}(X \setminus K)$.

For a closed subset, $K \subseteq X$, of a finite dimensional vector space, and a point $x \in K$, we use $T_K(x)$ to denote the *contingent cone* to K at x, i.e. the set of $v \in X$ such that there exists a sequence of real numbers $h_n > 0$ converging to 0 and a sequence of $v_n \in X$ converging to v satisfying

$$\forall\, n \geq 0, \;\; x + h_n v_n \;\in\; K.$$

Notice that, if x is in the interior of K, $T_K(x) = X$.

Subsequently we will be dealing with differential inclusions of the form $\dot{x} \in F(x)$, where $F : X \to 2^X$. To ensure existence of solutions we will need to impose some standard regularity assumptions on the map F, for example require F to be Marchaud and/or Lipschitz. We say that a map $F : X \to 2^X$ is *Marchaud* if and only if

1. the graph and the domain of F are nonempty and closed;
2. for all $x \in X$, $F(x)$ is convex, compact and nonempty; and,

3. the growth of F is linear, that is there exists $c > 0$ such that for all $x \in X$

$$\sup\{\|v\| \mid v \in F(x)\} \ \leq \ c(\|x\| + 1).$$

We say F is *Lipschitz* if and only if there exists a constant $\lambda > 0$ (known as the *Lipschitz constant*) such that for all $x, x' \in X$

$$F(x) \subseteq F(x') + \lambda\|x - x'\|B(0,1).$$

7.4 Viability Conditions

The viability conditions for impulse differential inclusions involve the notion of "viability with target". Viability with target provides conditions under which solutions of $\dot{x} \in F(x)$ that remain in a set K until they reach a target set C exist.

Lemma 7.1 *Consider a Marchaud map $F : X \to 2^X$ and two closed sets $K \subseteq X$ and $C \subseteq X$. For all $x_0 \in K$, there exists a solution of $\dot{x} \in F(x)$ starting at x_0 which is either*

1. *defined over $[0, \infty[$ with $x(t) \in K$ for all $t \geq 0$, or*
2. *defined over $[0, T]$ for some $T \geq 0$, with $x(T) \in C$ and $x(t) \in K$ for all $t \in [0, T]$,*

if and only if for all $x \in K \setminus C$, $F(x) \cap T_K(x) \neq \emptyset$.

Proof:

Necessity: Consider $x_0 \in K \backslash C$ and $x(\cdot)$ a trajectory starting from x_0 which stays in K on some interval $[0, \sigma]$ (and which does not reach C in this time interval). By application of Proposition 3.4.1 of [12], we obtain

$$F(x_0) \cap T_K(x_0) \neq \emptyset.$$

Sufficiency: Let $x_0 \in K \backslash C$. Because C is closed, some $r > 0$ exists such that $B(x_0, r) \cap C \neq \emptyset$. In the set $B_K(x_0, r) := K \cap B(x_0, r)$, one can imitate the proof of Proposition 3.4.2 of [12] and obtain the existence of $T > 0$ and of a solution to $\dot{x} \in F(x)$ starting at x_0 which remains in $B_K(x_0, r)$ on $[0, T]$.

Using an argument (Zorn's Lemma) classical in differential equation theory, it is possible to extend $x(\cdot)$ to a maximal trajectory - again denoted $x(\cdot)$ - on some $[0, \tilde{T}]$ viable in K and such that $C \cap [0, \hat{T}) = \emptyset$. Either $\hat{T} = +\infty$ and the proof is complete, or $\hat{T} < +\infty$ and then $x(\hat{T}) \in C$ (if not one could extend a little the trajectory to a viable one, this would be a contradiction with the maximality of $x(\cdot)$). ■

The conditions characterising viable sets depend on whether the set J is open or closed. In the case where J is closed we have the following.

Theorem 7.1 (Viability Conditions, J Closed) *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud, R is upper semicontinuous with closed domain and J is closed. A closed set $K \subseteq X$ is viable under H if and only if*

1. *$K \cap J \subseteq R^{-1}(K)$, and*
2. *$\forall\, x \in K \backslash R^{-1}(K)$, $F(x) \cap T_K(x) \neq \emptyset$*

In words, the conditions of the theorem require that for any state $x \in K$, whenever a discrete transition has to take place ($x \in K \cap J$), a transition back into K is possible ($R(x) \cap K \neq \emptyset$), and whenever a discrete transition to another point in K is not possible ($R(x) \cap K = \emptyset$) continuous evolution that remains in K has to be possible (encoded by the local viability condition $F(x) \cap T_K(x) \neq \emptyset$). **Proof**: Notice that, since R is upper semicontinuous with closed domain and K is closed, $R^{-1}(K)$ is also closed.

Necessity: Assume that K is viable under (X, F, R, J) and consider an arbitrary $x_0 \in K$. To show the first condition is necessary assume $x_0 \in K \cap J$. Then continuous evolution is impossible at x_0. Assume, for the sake of contradiction, that $x_0 \notin R^{-1}(K)$. Then either $R(x) = \emptyset$ (in which case the system blocks and no infinite runs start at x_0) or all runs starting at x_0 leave K through a discrete transition to some $x_1 \in R(x_0)$. In either case, the assumption that K is viable is contradicted. To show the second condition is necessary, assume $x_0 \in K \setminus R^{-1}(K)$. Since an infinite run viable in K starts at x_0, there exists a solution to the differential inclusion $\dot{x} \in F(x)$ starting at x_0 which is either

1. defined on $[0, \infty[$ with $x(t) \in K \setminus J$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in R^{-1}(K)$ and $x(t) \in K \setminus J$ for all $t \in [0, t'[$.

This implies, in particular, that there is a solution to the differential inclusion $\dot{x} \in F(x)$ starting at x_0 which is either

1. defined on $[0, \infty[$ with $x(t) \in K$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in R^{-1}(K)$ and $x(t) \in K$ for all $t \in [0, t']$.

By the necessary part of Lemma 7.1, this implies that for all $x_0 \in K \setminus R^{-1}(K)$, $F(x) \cap T_K(x) \neq \emptyset$.

Sufficiency: Assume the conditions of the theorem are satisfied and consider an arbitrary $x_0 \in K$. We construct an infinite run of (X, F, R, J) starting at x_0 and viable in K by induction. We distinguish two cases, $x_0 \in K \setminus R^{-1}(K)$ and $x_0 \in K \cap R^{-1}(K)$. In the first case, by the sufficient part of Lemma 7.1, there exists a solution to the differential inclusion $\dot{x} \in F(x)$ starting at x_0 which is either

1. defined on $[0, \infty[$ with $x(t) \in K$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in R^{-1}(K)$ and $x(t) \in K$ for all $t \in [0, t']$.

Notice that, since by the first assumption of the theorem, $K \cap J \subseteq R^{-1}(K)$ there must also be a solution to the differential inclusion $\dot{x} \in F(x)$ starting at x_0 which is either

1. defined on $[0, \infty[$ with $x(t) \in K \setminus J$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in R^{-1}(K)$ and $x(t) \in K \setminus J$ for all $t \in [0, t'[$

(i.e. either the solution stays in K forever and never reaches J, or the solution stays in K and reaches $R^{-1}(K)$ by the time it reaches J). In the former case, consider the infinite run $([0, \infty[, x)$; this is clearly a run of (X, F, R, J), viable in K. In the latter case, let $\tau_0 = 0$, $\tau_0' = t'$, and $\tau_1 = \tau_0'$. Since $x(\tau_0') \in R^{-1}(K)$, $x(\tau_1)$ can be chosen such that $x(\tau_1) \in K$. Notice that this argument also covers the case where $x_0 \in K \cap R^{-1}(K)$, with $x(\tau_0')$ playing the role of x_0. An infinite run viable in K can now be constructed inductively, by substituting x_0 by $x(\tau_1)$ and repeating the process. ■

Similar conditions characterise viability when the set J is open, or, in other words, the set $I = X \setminus J$ is closed.

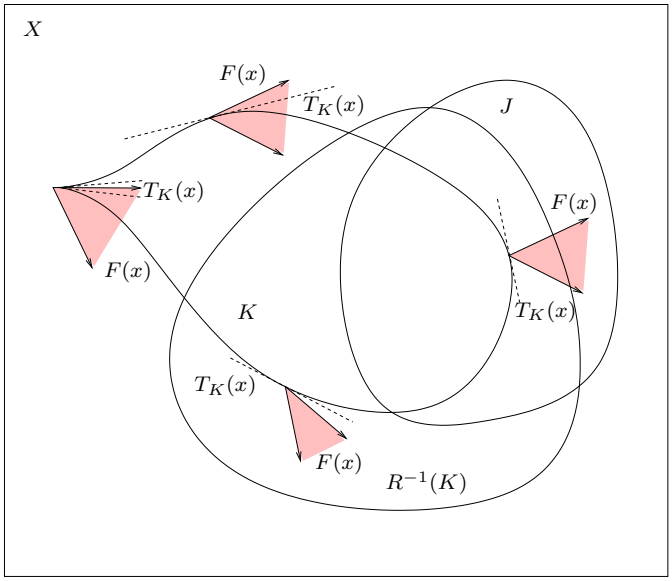

Figure 7.1: K viable under $H = (X, F, R, J)$

Theorem 7.2 (Viability Conditions, J Open) *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud, R is upper semicontinuous with closed domain and J is open. A closed set $K \subseteq X$ is viable under H if and only if*

1. $K \cap J \subseteq R^{-1}(K)$, *and*
2. $\forall\, x \in (K \cap I) \backslash R^{-1}(K), \quad F(x) \cap T_{K \cap I}(x) \neq \emptyset$

Figure 7.1 suggests how the conditions of Theorems 7.1 and 7.2 can be interpreted pictorially.

Notice that Assumption 7.1 does not need to be added explicitly to Theorems 7.1 and 7.2, since the part of it that is essential to guarantee the existence of a run viable in K is implied by the conditions of the theorems. Conditions that guarantee the existence of runs for impulse differential inclusions can be deduced as a corollary of Theorems 7.1 and 7.2.

Corollary 7.1 *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud, and R is upper semicontinuous with closed domain and J is either open or closed. Every finite run of H can be extended to an infinite run if and only if H satisfies Assumption 7.1.*

7.5 Invariance Conditions

The conditions for invariance make use of the notion of "invariance with target" for continuous differential inclusions. Invariance with target involves conditions ensuring that all solutions of $\dot{x} \in F(x)$ remain in a set K until they reach a target set, C.

Lemma 7.2 *Consider a Marchaud and Lipschitz map $F : X \to 2^X$ and two closed sets K and C. All solutions of $\dot{x} \in F(x)$ starting at some $x_0 \in K$ are either*

1. *defined over $[0, \infty[$ with $x(t) \in K$ for all $t \geq 0$, or*
2. *defined over $[0, T]$ with $x(T) \in C$ and $x(t) \in K$ for all $t \in [0, T]$,*

if and only if for all $x \in K \setminus C$, $F(x) \subseteq T_K(x)$.

Proof:

Necessity: Assume that all solutions starting in K stay in K until they reach C. Consider $x_0 \in K \setminus C$ and $v_0 \in F(x_0)$. Then (see for example [12] Corollary 5.3.2) there exists a trajectory $x(\cdot)$ of $\dot{x} \in F(x)$ starting at x_0 such that $\frac{d}{dt}x(0) = v_0$. Since x is a solution to $\dot{x} \in F(x)$ it remains in K until it reaches C. But $x_0 \in K \setminus C$ and C is closed, therefore, there exists $\alpha > 0$ such that $x(t) \in K$ for all $t \in [0, \alpha]$. Since x is absolutely continuous, for all $t \in [0, \alpha[$ where $\frac{d}{dt}x(t)$ is defined, $\frac{d}{dt}x(t) \in T_K(x(t))$ (see for example [12]). In particular, for $t = 0$, $v_0 = \frac{d}{dt}x(0) \in T_K(x_0)$. Hence, for all $x_0 \in K \setminus C$ and for all $v_0 \in F(x_0)$, $v_0 \in T_K(x_0)$, or, in other words, $F(x_0) \subseteq T_K(x_0)$.

Sufficiency: Let λ be the Lipschitz constant of F. Consider $x_0 \in K$ and a solution $x(\cdot)$ of $\dot{x} \in F(x)$ starting at x_0, and show that x remains in K until it reaches C. If $x_0 \in C$ then there is nothing to prove. If $x_0 \in K \setminus C$ consider

$$\theta = \sup\{t \mid \forall\, t' \in [0, t[,\ x(t') \in K \setminus C\}.$$

If $\theta = \infty$ or $x(\theta) \in C$ we are done. We show that $x(\theta) \in K \setminus C$ leads to a contradiction. Indeed, consider $\alpha > 0$ such that $B(x(\theta), \alpha) \cap C = \emptyset$ (which exists since $x(\theta) \notin C$ and C is closed), and $\theta' > \theta$, such that for all $t \in [\theta, \theta']$, $x(t) \in B(x(\theta), \alpha)$ (which exists since x is continuous). For $t \in [\theta, \theta']$, let $\Pi_K(x(t))$ denote a point of $B(x(\theta), \alpha) \cap K$ such that

$$d(x(t), K) = d(x(t), \Pi_K(x(t)))$$

(a projection of $x(t)$ onto K). Then (see for example [12], Lemma 5.1.2) for almost every $t \in [\theta, \theta']$,

$$\begin{aligned}
\frac{d}{dt}d(x(t), K) \leq & d\left(\frac{d}{dt}x(t), T_K(\Pi_K(x(t)))\right) \\
\leq & d\left(\frac{d}{dt}x(t), F(\Pi_K(x(t)))\right) \\
\leq & d\left(\frac{d}{dt}x(t), F(x(t))\right) + \lambda d(x(t), \Pi_K(x(t))) \\
\leq & 0 + d(x(t), K)
\end{aligned}$$

since x is a solution to $\dot{x} \in F(x)$ and by definition of Π. By the Gronwall lemma, $d(x(t), K) = 0$ for all $t \in [\theta, \theta']$, which contradicts the definition of θ. Summarising, if $F(x) \subseteq T_K(x)$ for all $x \in K \setminus C$, then all solutions starting in K either stay for ever in $K \setminus C$ or reach C before they leave K. ■

Lemma 7.2 allows us to prove the following invariance theorem for impulse differential inclusions.

Theorem 7.3 (Invariance Conditions) *Consider an impulse differential inclusion* $H = (X, F, R, J)$ *such that* F *is Marchaud and Lipschitz and* J *is closed. A closed set* $K \subseteq X$ *is invariant under* H *if and only if*

1. $R(K) \subseteq K$, *and*
2. $\forall\, x \in K \setminus J,\ \ F(x) \subseteq T_K(x)$.

In words, the conditions of the theorem require that for all $x \in K$, if a discrete transition is possible ($x \in R^{-1}(X)$), then all states after the transition are also in K ($R(x) \subseteq K$), whereas if continuous evolution is possible ($x \notin J$) then all possible solutions of the differential inclusion $\dot{x} \in F(x)$ remain in K (characterised here by the invariance condition $F(x) \subseteq T_K(x)$). Figure 7.2 suggests how the conditions of Theorem 7.3 can be interpreted pictorially.

Proof:

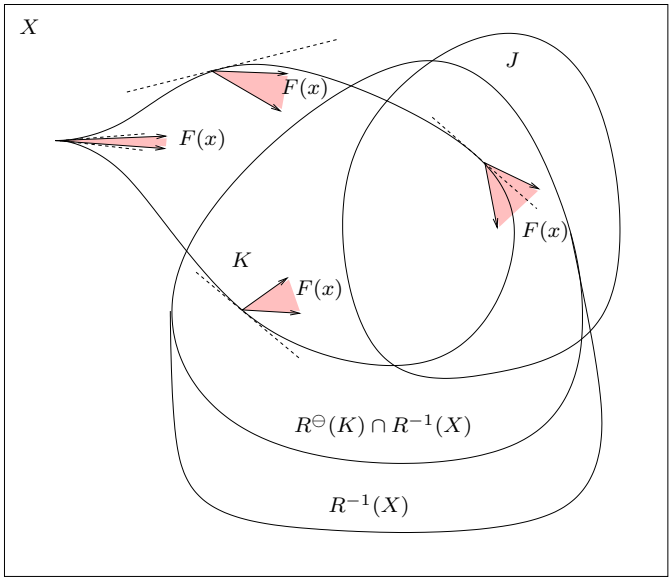

Figure 7.2: K invariant under (X, F, R, J)

Necessity: Assume that K is invariant under (X, F, R, J). If the first condition is violated, then there exists $x_0 \in K$ and $x_1 \in R(x_0)$ with $x_1 \notin K$. Therefore, there exists a run starting at x_0 that leaves K through a discrete transition to some x_1 and the assumption that K is invariant is contradicted. To show the second condition is necessary, notice that since all runs of (X, F, R, J) starting in K are viable in K, then all solutions to $\dot{x} \in F(x)$ starting in K are either

1. defined on $[0, \infty[$ with $x(t) \in K \setminus J$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in J$ and $x(t) \in K$ for all $t \in [0, t']$.

Otherwise, there would exist a solution of $\dot{x} \in F(x)$ which leaves K before reaching J. This solution would be a run of (X, F, R, J) that is not viable in K, which would contradict the assumption that K is invariant. By the necessary part of Lemma 7.2, 1 and 2 imply that for all $x_0 \in K \setminus J$, $F(x) \subseteq T_K(x)$.

Sufficiency: Assume the conditions of the theorem are satisfied and consider an arbitrary $x_0 \in K$ and an arbitrary run, (τ, x), of (X, F, R, J) starting at x_0. Notice that $x(\tau_0) = x_0 \in K$ by assumption. Assume $x(\tau_i) \in K$ and show $x(t) \in K$ until τ_{i+1}; the claim then follows by induction. If $t = \tau_i$ we are done. If $\tau_i \prec t \preceq \tau_i'$, then $x(\tau_i) \in K \setminus J$ since continuous evolution is possible from $x(\tau_i)$. By the second condition of the theorem and the sufficient part of Lemma 7.2, all solutions to the differential inclusion $\dot{x} \in F(x)$ starting at $x(\tau_i)$ are either

1. defined on $[0, \infty[$ with $x(t) \in K$ for all $t \geq 0$; or,
2. defined on $[0, t']$ with $x(t') \in J$ and $x(t) \in K$ for all $t \in [0, t']$.

In the first case, the run is viable in K and we are done. In the second case, $\tau_i' \preceq t'$ and therefore for all $t \in [\tau_i, \tau_i']$, $x(t) \in K$. If $x(\tau_i') \in R^{-1}(K)$, $x(\tau_{i+1}) \in R(x(\tau_i)) \subseteq K$ by the first condition of the theorem. If, on the other hand, $x(\tau_i') \in J$, but $R(x(\tau_i')) = \emptyset$, then the execution blocks at τ_i, and therefore is viable in K. ■

Notice that no assumptions need to be imposed on R. Strictly speaking, Theorem 7.3 remains true even without Assumption 7.1; if the impulse differential inclusion has no runs for certain initial

conditions in K, then, vacuously, all runs that start at these initial conditions are viable in K. In practice, it may be prudent to impose Assumption 7.1, to ensure the results are meaningful.

7.6 Viability Kernels

If K is not viable under an impulse differential inclusion H, one would like to characterise the largest subset of K which is viable under H. This set turns out to be the viability kernel of K under the impulse differential inclusion. The viability kernel of an impulse differential inclusion can be characterised in terms of the notion of the viability kernel with target for a continuous differential inclusion. For a differential inclusion $\dot{x} \in F(x)$, the viability kernel of a set K with target C, $\mathrm{V}iab_F(K, C)$, is defined as the set of states for which there exists a solution to the differential inclusion that remains in K either forever, or until it reaches C.

Lemma 7.3 *Consider a Marchaud map $F : X \to 2^X$ and two closed subsets of X, K and C. $\mathrm{V}iab_F(K, C)$ is the largest closed subset of K satisfying the conditions of Lemma 7.1.*

Proof: Let $D \subseteq K$ a closed set satisfying assumptions of Lemma 7.1. Clearly $D \subseteq \mathrm{V}iab_F(K, C)$.

We claim that $\mathrm{V}iab_F(K, C)$ is closed. Consider a sequence $x_n \in \mathrm{V}iab_F(K, C)$ converging to some $x \in X$. Since K is closed, $x \in K$. We show that $x \in \mathrm{V}iab_F(K, C)$. If $x \in C$, the proof is done. Else, there exists an $r > 0$ with $K \cap B(x, r) \neq \emptyset$. For n large enough $x_n \in B(x, \frac{r}{2})$. For any such n, consider $x_n(\cdot)$ a solution to the differential inclusion starting from x_n, viable in K until it reaches C. Such a solution exists, since $x_n \in \mathrm{V}iab_F(K, C)$.

The graph of the solution map of the differential inclusion restricted to the compact set

$$\{x\} \cup \{x_n,\ n > 0\}.$$

is compact (Theorem 3.5.2 in [12]). Hence, there exists a subsequence to $x_n(\cdot)$ - again denoted $x_n(\cdot)$ - converging to a solution $x(\cdot)$ of the differential inclusion starting at x uniformly on compact intervals.

Let $\sigma > 0$ such that $x[0, \sigma) \cap C = \emptyset$. Such a σ exists since $x \notin C$, C is closed, and $x(\cdot)$ is continuous. Fix $0 \leq t < \sigma$. For n large enough, $x_n[0, t] \cap C = \emptyset$ because C is closed and $x_n(\cdot)$ converges uniformly to $x(\cdot)$ on $[0, t]$. Since $x_n[0, t]$ is contained in K so is $x[0, t]$. Because σ and t are arbitrary, we can deduce that $x(\cdot)$ is viable in K until it reaches C. So $x \in \mathrm{V}iab_F(K, C)$, and therefore $\mathrm{V}iab_F(K, C)$ is closed.

It remains to prove that $\mathrm{V}iab_F(K, C)$ satisfies conditions of Lemma 7.1 (i.e., that it is itself viable with target C). Let $x_0 \in \mathrm{V}iab_F(K, C)$. By the very definition of the viability kernel some trajectory $x(\cdot)$ starting from x exists which is viable in K until it reaches C. Suppose by contradiction that some $s > 0$ exists such $x(s) \notin \mathrm{V}iab_F(K, C)$ and $x[0, s] \cap C = \emptyset$. Then any trajectory starting from $x(s)$ leaves K before reaching C. But $t \mapsto x(s + t)$ is such a trajectory which is viable in K until it reaches C, a contradiction. ∎

Exercise 7.1 Show that $K \cap C \subseteq \mathrm{V}iab_F(K, C) \subseteq K$.

Using this notion, one can give an alternative characterisation of the sets that are viable under an impulse differential inclusion, as fixed points of an appropriate operator. For an impulse differential inclusion $H = (X, F, R, J)$, consider the operator $\mathrm{P}re^{\exists}_H : 2^X \to 2^X$ defined by

$$\mathrm{P}re^{\exists}_H(K) = \mathrm{V}iab_F(K \cap I, R^{-1}(K)) \cup (K \cap R^{-1}(K))$$

Recall that $I = X \setminus J$.

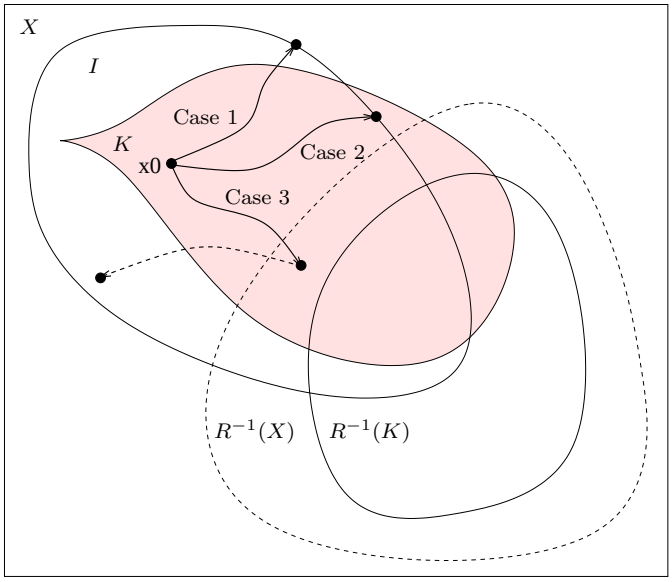

Figure 7.3: Three possible evolutions for $x_0 \notin \mathrm{V}iab_F(K \cap I, R^{-1}(K)) \cup (K \cap R^{-1}(K))$.

Lemma 7.4 *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud, R is upper semicontinuous with closed domain, and J is open. A closed set $K \subseteq X$ is viable under H if and only if it is a fixed point of the operator $\mathrm{P}re^{\exists}_H$.*

Proof:

Necessity: We first show that for every closed set K viable under $H = (X, F, R, J)$, $\mathrm{P}re^{\exists}_H(K) = K$. $\mathrm{P}re^{\exists}_H(K)$ is clearly a subset of K, since $\mathrm{V}iab_F(K \cap I, R^{-1}(K)) \subseteq K \cap I \subseteq K$. Conversely, consider an arbitrary $x_0 \in K$. Assume, for the sake of contradiction, that $x_0 \notin \mathrm{V}iab_F(K \cap I, R^{-1}(K)) \cup (K \cap R^{-1}(K))$. Consider an arbitrary infinite run (τ, x) viable in K and starting at x_0. Then $x(\tau_0) \notin R^{-1}(K)$ and $x(\tau_0) \notin \mathrm{V}iab_F(K \cap I, R^{-1}(K))$. If $\tau_0 = \tau_0'$, x starts by a discrete transition to some $x(\tau_1) \in R(x(\tau_0))$. Since $x(\tau_0) \notin R^{-1}(K)$, $x(\tau_1) \notin K$, which contradicts the assumption that (τ, x) is viable in K. If $\tau_0 < \tau_0'$, then (τ, x) starts by continuous evolution. Since $x_0 = x(\tau_0) \notin \mathrm{V}iab_F(K \cap I, R^{-1}(K))$, the run either

1. leaves K (at time $t \prec \tau_0'$) before it reaches $R^{-1}(K)$, or
2. leaves I (at time τ_0') before it reaches $R^{-1}(K)$, or
3. takes a transition from some $x(\tau_0') \in K \cap I \setminus R^{-1}(K)$

(see Figure 7.3). The first case contradicts the assumption that (τ, x) is viable in K. In the remaining cases, $x(\tau_0') \notin R^{-1}(K)$ and since $x(\tau_1) \in R(x(\tau_0'))$, we have $x(\tau_1) \notin K$. This also contradicts the assumption that (τ, x) is viable in K.

Sufficiency: Next, we show that every closed set K such that $K = \mathrm{P}re^{\exists}_H(K)$ is viable. Consider an arbitrary $x_0 \in K$; we construct by induction an infinite run, (τ, x) that starts at x_0 and is viable in K. By assumption $x_0 = x(\tau_0) \in K$. Assume that we have constructed a run viable in K defined over a finite sequence $[\tau_0, \tau_0'], [\tau_1, \tau_1'], \ldots, [\tau_i, \tau_i]$. Since K is a fixed point of $\mathrm{P}re^{\exists}_H$ and the run is viable in K, $x(\tau_i) \in \mathrm{V}iab_F(K \cap I, R^{-1}(K)) \cup (K \cap R^{-1}(K))$. If $x(\tau_i) \in K \cap R^{-1}(K)$, let $\tau_i' = \tau_i$ and chose $x(\tau_{i+1}) \in R(x(\tau_i')) \cap K$. If $x(\tau_i) \in \mathrm{V}iab_F(K \cap I, R^{-1}(K))$, then there exists a solution to the differential inclusion $\dot{x} \in F(x)$ which is either:

1. defined over $[0, \infty[$ with $x(t) \in K \cap I$ for all $t \geq 0$; or,
2. defined over $[0, t']$ with $x(t') \in R^{-1}(K)$ and $x(t) \in K \cap I$ for all $t \in [0, t']$.

In the former case, set $\tau_i' = \infty$ and the construction of the infinite run is complete. In the latter case, let $\tau_i' = \tau_i + t'$ and choose $x(\tau_{i+1}) \in R(x(\tau_i')) \cap K$. The claim follows by induction. ∎

Theorem 7.4 (Viability Kernel) *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud, R is upper semicontinuous with closed domain and compact images, and J is open. The viability kernel of a closed set $K \subseteq X$ under H is the largest closed subset of K viable under H, that is, the largest closed fixed point of $\mathrm{Pre}_H^{\exists}$ contained in K.*

Proof: The proof makes use of the sequence of sets $\{K_i\}$ constructed by the Viability Kernel Approximation algorithm, that is the sequence $K_0 = K$, $K_{i+1} = \mathrm{Pre}_H^{\exists}(K_i)$. Let

$$K_\infty = \bigcap_{i=0}^{\infty} K_i.$$

The proof proceeds in a sequence of steps. We show that:

1. For every viable set $L \subseteq K$, $L \subseteq \mathrm{Viab}_H(K)$.
2. K_∞ is closed.
3. $\mathrm{Viab}_H(K) \subseteq K_\infty$.
4. $K_\infty \subseteq \mathrm{Viab}_H(K)$.
5. $\mathrm{Viab}_H(K)$ is viable.

Step 1: Every set $L \subseteq K$ which viable under $H = (X, F, R, J)$ must be contained in $\mathrm{Viab}_H(K)$, since for all $x_0 \in L$ there exists an infinite run starting at x_0 that stays in L, and therefore in K.

Step 2: Since $\mathrm{Viab}_F(K_i \cap I, R^{-1}(K_i)) \subseteq K_i \cap I \subseteq K_i$, $K_{i+1} \subseteq K_i$ for all i. Since K is closed, K_0 is closed. Moreover, if K_i is closed, then $R^{-1}(K_i)$ is closed (since R is upper semicontinuous with closed domain), and $\mathrm{Viab}_F(K_i \cap I, R^{-1}(K_i))$ is closed (by Lemma 7.3, since I and $R^{-1}(K_i)$ are closed), and, therefore, K_{i+1} is closed. By induction, K_i form a sequence of nested closed sets, and therefore K_∞ is closed (possibly the empty set).

Step 3: Consider a point $x_0 \in \mathrm{Viab}_H(K)$ and show that $x_0 \in K_\infty$. Assume, for the sake of contradiction, that $x_0 \notin K_\infty$. Then there exists $N \geq 0$ such that $x_0 \notin K_N$. If $N = 0$, then $x_0 \notin K_0 = K$, therefore all runs starting at x_0 that are not viable in K (trivially). This contradicts the assumption that $x_0 \in \mathrm{Viab}_H(K)$. If $N > 0$, we show that for all infinite runs (τ, x) starting at x_0 (which exist since $x_0 \in \mathrm{Viab}_H(K)$), there exists a $t \preceq \tau_1$ such that[1] $x(t) \notin K_{N-1}$. The claim then follows by induction. Indeed, since $x_0 \notin K_N$ we must have have $x_0 \notin \mathrm{Viab}_F(K_{N-1} \cap I, R^{-1}(K_{N-1})) \cup (K_{N-1} \cap R^{-1}(K_{N-1}))$. If $\tau_0 < \tau_0'$, then (τ, x) starts by continuous evolution. Since $x_0 = x(\tau_0) \notin \mathrm{Viab}_F(K_{N-1} \cap I, R^{-1}(K_{N-1}))$, then all solutions to $\dot{x} \in F(x)$ either

1. leave K_{N-1} (at some $t \preceq \tau_0'$) before they reach $R^{-1}(K_{N-1})$, or
2. leave I (at time τ_0') before they reach $R^{-1}(K_{N-1})$, or
3. take a transition from some $x(\tau_0') \in (K_{N-1} \cap I) \setminus R^{-1}(K_{N-1})$.

[1] If $\tau = [\tau_0, \infty)$, $t \preceq \tau_1$ is replaced by $t \prec \tau_0' = \infty$.

(refer to Figure 7.3). In the first case we are done. In the remaining cases, $x(\tau_0') \notin R^{-1}(K_{N-1})$ and since $x(\tau_1) \in R(x(\tau_0'))$, we have $x(\tau_1) \notin K_{N-1}$. The last argument also subsumes the case $\tau_0 = \tau_0'$, since $x_0 \notin K_{N-1} \cap R^{-1}(K_{N-1})$.

Step 4: Consider an arbitrary $x_0 \in K_\infty$. To show that $x_0 \in Viab_H(K)$, we construct an infinite run $(\tau, x) \in \mathcal{R}_H^\infty(x_0)$ viable in K. More specifically, since $x_0 \in K_k$ for all k, there exists a sequence of runs $(\tau^{(k)}, x^{(k)}) \in \mathcal{R}_H(x_0)$, which remain in K for at least k jumps. We will show that the sequence $(\tau^{(k)}, x^{(k)})$ has a cluster point $(\bar\tau, \bar x) \in \mathcal{R}_H^\infty(x_0)$, which is an infinite run of (X, F, R, J), starting at x_0, viable in K.

Let $[\tau_i^{(k)}, \tau_i^{(k)'}]$ (or $[\tau_i^{(k)}, \tau_i^{(k)'}[$ if i is the last interval) denote the sequence of intervals $\tau^{(k)}$. Recall that, without loss of generality, we can assume that $\tau_0^{(k)} = 0$ for all k. Let $\bar\tau_0 = 0$ and define

$$\bar\tau_0' = \lim_{k\to\infty} \inf \tau_0^{(k)'}.$$

Then there exists a subsequence of $\tau_0^{(k)'}$, denoted by $\tau_0^{\sigma(k)'}$, such that

$$\lim_{k\to\infty} \tau_0^{\sigma(k)'} = \bar\tau_0'.$$

We distinguish three cases:

1. $\bar\tau_0' = +\infty$;
2. $\bar\tau_0' \in]0, +\infty[$; and,
3. $\bar\tau_0' = 0$.

Case 1 will lead to a run $(\bar\tau, \bar x) \in \mathcal{R}_H^\infty(x_0)$ that is viable in K and makes no jumps. Case 2 will lead to a run $(\bar\tau, \bar x) \in \mathcal{R}_H^\infty(x_0)$ that is viable in K, whose first jump comes after an interval of continuous evolution. Finally, Case 3 will lead a run $(\bar\tau, \bar x) \in \mathcal{R}_H^\infty(x_0)$ viable in K, that takes its first jump immediately.

Case 1: Consider a sequence $y^{\sigma(k)}(\cdot)$ of solutions to the differential inclusion

$$\dot x \in F(x), \quad x(0) = x_0, \tag{7.1}$$

that coincide with $x^{\sigma(k)}$ on $[0, \tau_0^{\sigma(k)'}[$. Because the set of solutions of (7.1) is compact (see Theorem 3.5.2 of [12]), there exists a subsequence $y^{\phi(k)}(\cdot)$ of the sequence $y^{\sigma(k)}(\cdot)$ that converges to a solution $\bar y(\cdot)$ of (7.1). Moreover, since $\lim_{k\to\infty} \tau_0^{\sigma(k)'} = +\infty$, the sequence $y^{\phi(k)}(\cdot)$ (and hence the sequence $x^{\phi(k)}(\cdot)$) converges to $\bar y(\cdot)$ uniformly over $[0, T]$, for all $T > 0$.

Now, $(\tau^{\phi(k)}, x^{\phi(k)})$ is a run of (X, F, R, J) viable in K for at least k jumps. Therefore, $x^{\phi(k)}(t) \in K \cap I$ for all $t \in [0, \tau_0^{\phi(k)'}[$, and hence, for sufficiently large k, $x^{\phi(k)}(t) \in K \cap I$ for all $t \in [0, T]$. Since $K \cap I$ is closed, $\bar y(t) \in K \cap I$ for all $t \in [0, T]$. Since T is arbitrary, $([0, \infty[, \bar y)$ is an infinite run of (X, F, R, J) (with no jumps) starting at x_0 and viable in K. The proof is complete.

Case 2: We can restrict attention to $k \geq 1$. As for case 1, define the sequence $y^{\sigma(k)}(\cdot)$ of solutions of (7.1) that coincide with $x^{\sigma(k)}$ on $[0, \tau_0^{\sigma(k)'}[$ and the subsequence $y^{\phi(k)}(\cdot)$ converging (uniformly over compact intervals) to a solution $\bar y(\cdot)$ of (7.1). As before, $(\tau^{\phi(k)}, x^{\phi(k)})$ is a run of (X, F, R, J) viable in K for at least $k > 0$ jumps. Therefore, $x^{\phi(k)}(t) \in K \cap I$ for all $t \in [0, \tau_0^{\phi(k)'}[$. Since $K \cap I$ is closed, $\bar y(t) \in K \cap I$ for all $t \in [0, \bar\tau_0']$. Therefore, $([\bar\tau_0, \bar\tau_0'], \bar y)$ is a finite run of (X, F, R, J) (with no jumps) starting at x_0 and viable in K.

Since $y^{\phi(k)}(\cdot)$ converges to $\bar y(\cdot)$ and $\tau_0^{\phi(k)'}$ converges to $\bar\tau_0'$, $x^{\phi(k)}(\tau_0^{\phi(k)'})$ converges to $\bar y(\bar\tau_0')$. Recall that $(\tau^{\phi(k)}, x^{\phi(k)})$ is a run of (X, F, R, J) viable in K for at least $k > 0$ jumps, therefore $x^{\phi(k)}(\tau_1^{\phi(k)}) \in R(x^{\phi(k)}(\tau_0^{\phi(k)'})) \cap K$. Since R is upper semicontinuous with closed domain and compact images,

there exists a subsequence of $x^{\phi(k)}(\tau_1^{\phi(k)})$ converging to some point $\bar{y}_1 \in R(\bar{y}(\bar{\tau}_0')) \cap K$. Therefore, $([0,\bar{\tau}_0'][\bar{\tau}_1,\bar{\tau}_1'],\bar{y})$ with $\bar{\tau}_1 = \bar{\tau}_1' = \bar{\tau}_0'$ and $\bar{y}(\bar{\tau}_1) = \bar{y}_1$ defined as above is a finite run of (X,F,R,J) (with one jump) starting at x_0 and viable in K.

Case 3: The second part of the argument for Case 2 shows that, since $x^{\phi(k)}(\tau_0^{\sigma(k)'})$ converge to x_0, there exists $\bar{y}_1 \in R(x_0) \cap K$. Therefore, $([0,\bar{\tau}_0'][\bar{\tau}_1,\bar{\tau}_1'],\bar{y})$ with $\bar{\tau}_0' = \bar{\tau}_1 = \bar{\tau}_1' = 0$, $\bar{y}(\tau_0') = x_0$ and $\bar{y}(\bar{\tau}_1) = \bar{y}_1$ is a finite run of (X,F,R,J) (with one instantaneous jump) starting at x_0 and viable in K.

To complete the proof for Cases 2 and 3, we repeat the argument starting at $\bar{y}(\bar{\tau}_1)$ (discarding the initial part of the sequences accordingly). We generate $\bar{\tau}_1' = \liminf_{k\to\infty} \tau_1^{(k)'}$ and construct a run of (X,F,R,J) viable in K, defined either over $[0,\bar{\tau}_0'][\bar{\tau}_1,\bar{\tau}_1'[$ (if $\bar{\tau}_1' = +\infty$, in which case the proof is complete) or over $[0,\bar{\tau}_0'][\bar{\tau}_1,\bar{\tau}_1'][\bar{\tau}_2,\bar{\tau}_2']$ with $\bar{\tau}_2 = \bar{\tau}_2' = \bar{\tau}_1'$ (if $\bar{\tau}_1'$ is finite). The claim follows by induction.

Step 5: Finally, we show $\mathrm{V}iab_H(K)$ is viable by showing that it is a fixed point of $\mathrm{P}re_H^{\exists}$. Recall that $\mathrm{P}re_H^{\exists}(\mathrm{V}iab_H(K)) \subseteq \mathrm{V}iab_H(K)$. Consider an arbitrary $x_0 \in \mathrm{V}iab_H(K)$ and assume, for the sake of contradiction, that $x_0 \notin \mathrm{P}re_H^{\exists}(\mathrm{V}iab_H(K))$. Consider an arbitrary infinite run (τ,x) viable in K and starting at x_0 (which exists since $x_0 \in \mathrm{V}iab_H(K)$). If $\tau_0 = \tau_0'$, x starts by a discrete transition to some $x(\tau_1) \in R(x_0)$. Since $x_0 \notin R^{-1}(\mathrm{V}iab_H(K))$, $x(\tau_1) \notin \mathrm{V}iab_H(K)$. If $\tau_0 < \tau_0'$, then (τ,x) starts by continuous evolution. Since $x_0 \notin \mathrm{V}iab_F(\mathrm{V}iab_H(K) \cap I, R^{-1}(\mathrm{V}iab_H(K)))$, the execution either

1. leaves $\mathrm{V}iab_H(K)$ (at time $t \prec \tau_0'$) before it reaches $R^{-1}(\mathrm{V}iab_H(K))$; or,
2. leaves I (at time τ_0') before it reaches $R^{-1}(\mathrm{V}iab_H(K))$; or,
3. takes a transition from some $x(\tau_0') \in \mathrm{V}iab_H(K) \cap I \setminus R^{-1}(\mathrm{V}iab_H(K))$

(see Figure 7.3). In all cases, (τ,x) either blocks or leaves $\mathrm{V}iab_H(K)$ at some $t \in \tau$ with $t \preceq \tau_1$. But if $x(t) \notin \mathrm{V}iab_H(K)$ there is no infinite run of $H = (X,F,R,J)$ starting at $x(t)$ and viable in K. Therefore, (τ,x) either blocks or is not viable in K. This contradicts the assumption that $x_0 \in \mathrm{V}iab_H(K)$. ∎

It should be stressed that the conditions of Theorem 7.4 ensure that for all initial conditions in the viability kernel infinite runs of the impulse differential inclusion exist, but do not ensure that these runs will extend over an infinite time horizon; all runs starting at certain initial conditions in the viability kernel may turn out to be Zeno.

The proof of Theorem 7.4 is based on the following abstract algorithm.

Algorithm 3 (Viability Kernel Approximation)
initialization: $K_0 = K$, $i = 0$
repeat
$K_{i+1} = \mathrm{P}re_H^{\exists}(K_i)$
$i = i + 1$
until $K_i = K_{i-1}$

The algorithm generates a sequence of nested, closed sets K_i that "converge" to the viability kernel. In addition to being useful in the proof of the theorem, the algorithm can therefore also be used to provide progressively better estimates of the viability kernel. This is, of course, provided one can compute $\mathrm{P}re^{\exists}(K_i)$ at each step. Numerical algorithms for approximating this computation have been developed see, for example, [87].

7.7 Invariance Kernels

If K is not invariant under an impulse differential inclusion H, one would like to characterise the largest subset of K which is invariant under H. This turns out to be the invariance kernel of K under the impulse differential inclusion. The invariance kernel can be characterised using the notion of the invariance kernel with target for continuous differential inclusions. For a differential inclusion $\dot{x} \in F(x)$, the invariance kernel of a set K with target C, $Inv_F(K, C)$ is defined as the set of states for which all solutions to the differential inclusion remain in K either for ever, of until they reach C.

Lemma 7.5 *Consider a Marchaud and Lipschitz map $F : X \to 2^X$ and two closed subsets of X, K and C. $Inv_F(K, C)$ is the largest closed subset of K satisfying the conditions of Lemma 7.2.*

Proof: By definition, $Inv_F(K, C)$ is the set of $x_0 \in K$ such that for all solutions $x(\cdot)$ of $\dot{x} \in F(x)$ starting at x_0 either

1. $x(t) \in K$ for all $t \geq 0$, or,
2. there exists $t' \geq 0$ such that $x(t') \in C$ and $x(t) \in K$ for all $t \in [0, t']$.

Therefore, $Inv_F(K, C)$ satisfies the conditions of Lemma 7.2. Moreover, every subset $L \subseteq K$ which satisfies the conditions of Lemma 7.2 must be contained in $Inv_F(K, C)$, since all runs starting in L stay in L (and therefore in K) until they reach C.

It remains to show that $Inv_F(K, C)$ is closed. Consider a sequence $x_n \in Inv_F(K, C)$ converging to x_0 and show that $x_0 \in Inv_F(K, C)$. Since by definition $Inv_F(K, C) \subseteq K$ and K is assumed to be closed, $x_0 \in K$. If $x_0 \in K \cap C$ there is nothing to prove, since by definition $K \cap C \subseteq Inv_F(K, C)$. If $x_0 \in K \setminus C$, let $x(\cdot)$ be any solution of $\dot{x} \in F(x)$ starting at x_0. Let

$$\theta = \sup\{t \mid \forall\, t' \in [0, t[,\ x(t') \in K \setminus C\}.$$

If $\theta = \infty$ or if $x(\theta) \in C$, then $x_0 \in Inv_F(K, C)$, and the proof is complete.

Let λ be the Lipschitz constant of F, and assume, for the sake of contradiction, that $\theta < \infty$ and $x(\theta) \in K \setminus C$. Then, by the definition of θ and the assumption that K and C are closed, there exists $\theta' > \theta$ such that $x(\theta') \notin K$ and for all $t \in [\theta, \theta']$, $x(t) \notin C$. Choose ϵ such that

$$d(x(\theta'), K) > \epsilon e^{\lambda \theta'} \tag{7.2}$$

(possible since K is closed and $x(\theta') \notin K$) and for all $t \in [0, \theta']$

$$\left\{x(t) + \epsilon B(0, 1) e^{\lambda t}\right\} \cap C = \emptyset \tag{7.3}$$

(possible since C is closed and for all $t \in [0, \theta']$, $x(t) \notin C$).

Since $x_n \to x_0$ there exists n large enough such that $\|x_n - x_0\| < \epsilon$. By Filippovs theorem (see for example [12], Theorem 5.3.1) there exists a solution $x_n(\cdot)$ of $\dot{x} \in F(x)$ starting at x_n such that for all $t \in [0, \theta']$

$$\|x_n(t) - x(t)\| \leq \|x_n - x_0\| e^{\lambda t},$$

or, in other words, for all $t \in [0, \theta']$

$$x_n(t) \in B(x(t), \|x_n - x_0\| e^{\lambda t}) \subseteq B(x(t), \epsilon e^{\lambda t}).$$

Therefore, by equation (7.3), for all $t \in [0, \theta']$, $x_n(t) \notin C$, while, by equation (7.2) $x_n(\theta') \notin K$. This contradicts the assumption that $x_n \in Inv_F(K, C)$. Hence, every converging sequence has its limit in $Inv_F(K, C)$, and therefore $Inv_F(K, C)$ is closed. ∎

Exercise 7.2 Show that $K \cap C \subseteq Inv_F(K, C) \subseteq K$.

Using the notion of invariance kernel with target, one can give an alternative characterisation of the sets that are invariant under an impulse differential inclusion, as fixed points of an operator. Given an impulse differential inclusion $H = (X, F, R, J)$, consider the operator $Pre^{\forall}_H : 2^X \to 2^X$ defined by

$$Pre^{\forall}_H(K) = Inv_F(K, J) \cap R^{\ominus 1}(K)$$

Lemma 7.6 *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud and Lipschitz, R is lower semicontinuous, and J is closed. A closed set $K \subseteq X$ is invariant under H if and only if it is a fixed point of the operator $Pre^{\forall}_H$.*

Proof:

Necessity: We first show that for every closed, invariant set K, $K = Pre^{\forall}_H(K)$. Clearly $Pre^{\forall}_H(K) \subseteq K$, since $Inv_F(K, J) \subseteq K$. Conversely, consider an arbitrary point $x_0 \in K$ and show that $x_0 \in Inv_F(K, J) \cap R^{\ominus 1}(K)$. Assume, for the sake of contradiction that this is not the case. Then, either $x_0 \notin Inv_F(K, J)$, or $x_0 \notin R^{\ominus 1}(K)$. If $x_0 \notin R^{\ominus 1}(K)$, there exists $x_1 \in R(x_0)$ such that $x_1 \notin K$; in other words, there exists a run of the impulse differential inclusion starting at x_0 that leaves K by a discrete transition. This contradicts the assumption that K is invariant. If, on the other hand, $x_0 \notin Inv_F(K, J)$ then, in particular, $x_0 \notin J \cap K$ (since $J \cap K \subseteq Inv_F(K, J)$); but $x_0 \in K$, so we must have $x_0 \notin J$, and therefore continuous evolution starting at x_0 is possible. Since $x_0 \notin Inv_F(K, J)$, there exists a solution to $\dot{x} \in F(x)$ starting at x_0 that leaves K before reaching J. This solution is a run (X, F, R, J) that starts in K but is not viable in K. This also contradicts the assumption that K is invariant.

Sufficiency: Next, we show that every closed set K such that $K = Pre^{\forall}_H(K)$ is invariant. Consider an arbitrary run (τ, x) starting at some $x_0 \in K$. We show that (τ, x) is viable in K by induction. Assume that we have shown that $x(t) \in K$ for all $t \in [\tau_0, \tau_0'], [\tau_1, \tau_1'], \ldots, [\tau_i, \tau_i]$. Then, since $K = Pre^{\forall}_H(K)$, $x(\tau_i) \in Inv_F(K, J) \cap R^{\ominus 1}(K)$. If $\tau_i = \tau_i'$ the system takes a discrete transition to some $x(\tau_{i+1}) \in R(x(\tau_i')) \subseteq K$, since $x(\tau_i') = x(\tau_i) \in R^{\ominus 1}(K)$. If $\tau_i < \tau_i'$ the run progresses by continuous evolution. Since $x(\tau_i) \in Inv_F(K, J)$, then either

1. $\tau_i' = \infty$ and $x(t) \in K$ for all $t \geq \tau_i$; or,
2. $\tau_i' < \infty$, $x(\tau_i') \in J$ and $x(t) \in K$ for all $t \in [\tau_i, \tau_i']$.

Notice that $x(\tau_i') \in K = Pre^{\forall}_H(K)$, and, in particular, $x(\tau_i') \in R^{\ominus 1}(K)$. Therefore, $x(\tau_{i+1}) \in R(x(\tau_i')) \subseteq K$. The claim follows by induction.

Notice that in the last argument $R(x(\tau_i'))$ may, in fact, be empty. In this case the run "blocks", in the sense that there exist no infinite runs starting at $x(\tau_i')$. The conclusion that all runs starting at x_0 are viable in K is still true however. To preclude this somewhat unrealistic situation, one can add Assumption 7.1 to the lemma and subsequent Theorem 7.5. ∎

Theorem 7.5 (Invariance Kernel) *Consider an impulse differential inclusion $H = (X, F, R, J)$ such that F is Marchaud and Lipschitz, R is lower semicontinuous and J is closed. The invariance kernel of a closed set $K \subseteq X$ under H is the largest closed subset of K invariant under H, that is, the largest, closed fixed point of $Pre^{\forall}_H$ contained in K.*

Proof: The proof makes use of the sequence of sets generated by the Invariance Kernel Algorithm given below, that is, the sets $K_0 = K$, $K_{i+1} = Pre^{\forall}_H(K_i)$. Let

$$K_\infty = \bigcap_{i=0}^{\infty} K_i.$$

The proof proceeds in a sequence of steps. We show that

1. For every invariant set $L \subseteq K$, $L \subseteq Inv_H(K)$.
2. K_∞ is closed.
3. $Inv_H(K) \subseteq K_\infty$.
4. $K_\infty = Pre^{\forall}_H(K_\infty)$.

Step 2, step 4 and Lemma 7.6 imply that K_∞ is invariant. Therefore, by step 1, $K_\infty \subseteq Inv_H(K)$, and, by step 3, $K_\infty = Inv_H(K)$. Summarising, $Inv_H(K)$ is the largest (by step 1), closed (by step 2), invariant (by step 4) subset of K.

Step 1: Every set $L \subseteq K$ which invariant under (X, F, R, J) must be contained in $Inv_H(K)$, since all runs starting in L stay in L, and therefore in K.

Step 2: Clearly, for all i, $K_{i+1} \subseteq Inv_F(K_i, J) \subseteq K_i$. Since K is closed, K_0 is closed. Moreover, if K_i is closed, then $Inv_F(K_i, J)$ is closed (by Lemma 7.5, since J is closed), $R^{\ominus 1}(K_i)$ is closed (since R is lower semicontinuous), and, therefore, K_{i+1} is closed. By induction, the K_i form a sequence of nested closed sets, and therefore K_∞ is closed (or the empty set).

Step 3: Consider a point $x_0 \in Inv_H(K)$ and show that $x_0 \in K_\infty$. Assume, for the sake of contradiction, that $x_0 \notin K_\infty$. Then there exists $N \geq 0$ such that $x \notin K_N$. If $N = 0$, then $x_0 \notin K_0 = K$, therefore there exists a (trivial) run starting at x_0 that is not viable in K. This contradicts the assumption that $x_0 \in Inv_H(K)$. If $N > 0$, we show that there exists a run stating at x_0 that after at most one discrete transition finds itself outside K_{N-1}. The claim then follows by induction. Indeed, since $x_0 \notin K_N$ we must either have $x_0 \notin Inv_F(K_{N-1}, J)$, or $x_0 \notin R^{\ominus 1}(K_{N-1})$. If $x_0 \notin R^{\ominus 1}(K_{N-1})$, there exists $x_1 \in R(x_0)$ such that $x_1 \notin K_{N-1}$, i.e., there exists a run starting at x_0 that transitions outside K_{N-1}. If, on the other hand, $x_0 \notin Inv_F(K_{N-1}, J)$, then $x_0 \notin J \cap K_{N-1}$. Therefore either $x_0 \notin K_{N-1}$ (and the proof is complete), or $x_0 \notin J$ and continuous evolution is possible. In the latter case, since $x_0 \notin Inv_F(K_{N-1}, J)$, by Lemma 7.5 there exists a solution to $\dot{x} \in F(x)$ starting at x_0 that leaves K_{N-1} before reaching J. This solution is a run of (X, F, R, J) that leaves K_{N-1}.

Step 4: Recall that $Pre^{\forall}_H(K_\infty) \subseteq K_\infty$. Consider an arbitrary $x_0 \in K_\infty$ and show that $x_0 \in Pre^{\forall}_H(K_\infty)$. Assume, for the sake of contradiction, that $x_0 \notin Inv_F(K_\infty, J) \cap R^{\ominus 1}(K_\infty)$. Then there exists a run (τ, x) starting at x_0 and a $t \preceq \tau_1$ such that[2] $x(t) \notin K_\infty$, or, in other words, there exists a run (τ, x), a $t \preceq \tau_1$ and a $N \geq 0$ such that $x(t) \notin K_N$. To see this notice that either $x(\tau_0) \notin R^{\ominus 1}(K_\infty)$ (in which case we can take $\tau_0' = \tau_0$, $x(\tau_1) \notin K_\infty$ and $t = \tau_1$) or $x(\tau_0) \notin Inv_F(K_\infty, J)$ (in which case there exists a solution to $\dot{x} \in F(x)$ that leaves K before reaching J). The same argument, however, also shows that $x(\tau_0) = x_0 \notin K_{N+1}$, which contradicts the assumption that $x_0 \in K_\infty$. ■

Algorithm 4 (Invariance Kernel Approximation)

initialisation: $K_0 = K$, $i = 0$
repeat
  $K_{i+1} = Pre^{\forall}_H(K_i)$
  $i = i + 1$
until $K_i = K_{i-1}$

At each step, the algorithm computes the set of states for which all solution of the differential inclusion $\dot{x} \in F(x)$ stay in the K_i until they reach J. K_{i+1} is then the subset of those states for which if a transition is possible, the state after the transition is also in K_i.

[2] If $\tau = [\tau_0, \tau_0']$ or $\tau = [\tau_0, \tau_0'[$, $t \preceq \tau_1$ should be replaced by $t \preceq \tau_0$ or, respectively, $t \prec \tau_0$.

7.8 The Bouncing Ball Example

It is easy to check that F_B is both Marchaud and Lipschitz and that R_B is upper and lower semi-continuous and has closed domain. Moreover, H_B also satisfies Assumption 7.1, since $R^{-1}(X) = J$. Therefore, we can immediately draw the following conclusion.

Proposition 7.1 *Infinite runs exist for all $x_0 \in X_T$.*

The proposition suggests that the impulse differential inclusion H_B does not deadlock. However, it is easy to show that for all $x_0 \in X_T$ all infinite runs are Zeno.

Despite the Zeno behaviour, H_B is in many ways a reasonable model of the bouncing ball system. For example, one can show that the ball never falls below the surface on which it bounces, and that the system dissipates energy.

Proposition 7.2 *The set $K = \{x \in X_T \mid x_1 \geq 0\}$ is viable and invariant. For all $C > 0$ the set $L = \{x \in X_T \mid gx_1 + x_2^2/2 \leq C\}$ is invariant.*

For the first part, notice that $K \cap J_B = \{x \in X_T \mid x_1 = 0 \text{ and } x_2 \leq 0\}$. Since R_B does not affect x_1, $K \cap J_B \subseteq R^{-1}(K)$ and $R(K) \subseteq K$. Moreover, $K \setminus J_B = \{x \in X_T \mid x_1 > 0 \text{ or } x_1 = 0 \text{ and } x_2 > 0\}$. For x such that $x_1 > 0$, $F_B(x) \subseteq T_K(x) = X_B$. For x such that $x_1 = 0$ and $x_2 > 0$, $F_B(x) \subseteq \{v \in X \mid v_1 > 0\} = T_K(x)$. Therefore, K is viable by Theorem 7.1 and invariant by Theorem 7.3.

For the second part, R leaves x_1 unchanged and maps x_2 to αx_2. Therefore $R(L) \subseteq L$ since $\alpha \in [0, 1]$. Moreover

$$\begin{aligned} L \setminus J = &\{x \in X_T \mid x_1 > 0 \text{ or } x_2 > 0\} \\ &\cap \{x \in X_T \mid gx_1 + x_2^2/2 \leq C\} \end{aligned}$$

For $x \in L \setminus J$ such that $gx_1 + x_2^2/2 < C$, $F_B(x) \subseteq T_K(x) = X_B$. For $x \in L \setminus J$ such that $gx_1 + x_2^2/2 = C$,

$$\begin{aligned} T_K(x) &= \{v \in X_B \mid v_1 g + v_2 x_2 \leq 0\} \\ &\supseteq \{v \in X_B \mid v_1 g + v_2 x_2 = 0\} \supseteq F_B(x) \end{aligned}$$

The claim follows by Theorem 7.3.

7.9 Bibliography and Further Reading

Reachability with inputs has also been studied in the context of optimal control and differential games [66, 94] and using ellipsoidal calculus [54, 53].

Other classes of control problems that have been studied for hybrid systems include supervisory control [52, 11, 28] and optimal control [21, 35].

Because the analytical study of the resulting equations is rarely possible computational tools have been developed to approximate the solutions numerically [27, 36, 26, 19, 81, 10, 80, 87].

Bibliography

[1] R. Alur, C. Courcoubetis, N. Halbawachs, 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.

[2] R. Alur and D. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[3] R. Alur and D. L. Dill. Automata for modeling real-time systems. In *Proceedings of ICALP '90*, volume 443 of *Lecture Notes in Computer Science*, pages 322–335. Springer-Verlag, 1990.

[4] R. Alur and T.A. Henzinger. Modularity for timed and hybrid systems. In *Proceedings of the Eighth International Conference on Concurrency Theory (CONCUR 1997)*, number 1243 in LNCS, pages 74–88. Springer Verlag, 1997.

[5] R. Alur, T.A. Henzinger, G. Lafferriere, and G.J. Pappas. Discrete abstractions of hybrid systems. *Proceedings of the IEEE*, 88(7):971–984, July 2000.

[6] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In *Hybrid Systems III*, number 1066 in LNCS, pages 220–231. Springer Verlag, 1996.

[7] M. Anderson, D. Bruck, S. E. Mattsson, and T. Schonthal. Omsim- an integrated interactive environment for object-oriented modeling and simulation. In *IEEE/IFAC joint symposium on computer aided control system design*, pages 285–290, 1994.

[8] M. Andersson. *Object-Oriented Modeling and Simulation of Hybrid Systems*. PhD thesis, Lund Institute of Technology, Lund, Sweden, December 1994.

[9] P.J. Antsaklis and A. Nerode, Editors. Special issue on hybrid control systems. *IEEE Transactions on Automatic Control*, 43(4), April 1998.

[10] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS. 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*, 88(7):1011–1025, July 2000.

[12] J.-P. Aubin. *Viability Theory*. Birkhäuser, Boston, 1991.

[13] J.-P. Aubin and H. Frankowska. *Set Valued Analysis*. Birkhäuser, Boston, 1990.

[14] J.-P. Aubin, J. Lygeros, M. Quincampoix, S.S. Sastry, and N. Seube. Impulse differential inclusions: A viability approach to hybrid systems. *IEEE Transactions on Automatic Control*, 47(1):2–20, January 2002.

[15] A. Balluchi, L. Benvenuti, M.D. Di Benedetto, C. Pinello, and A.L. Sangiovanni-Vincentelli. Automotive engine control and hybrid systems: Challenges and opportunities. *Proceedings of the IEEE*, 88(7):888–912, July 2000.

[16] J. Bengtsson, K.G. Larsen, F. Larsson, P. Petterson, and W. Yi. UPAAL: a tool suit for automatic verification of real-time systems. In *Hybrid Systems III*, number 1066 in LNCS, pages 232–243. Springer Verlag, 1996.

[17] B. Bérard, P. Gastin, and A. Petit. On the power of non observable actions in timed automata. In *Actes du STACS '96*, Lecture Notes in Computer Science 1046, pages 257–268. Springer-Verlag, 1996.

[18] V.S. Borkar. *Probability theory: an advanced course*. Springer Verlag, New York, 1995.

[19] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS, pages 73–88. Springer Verlag, 2000.

[20] M.S. Branicky. Multiple Lyapunov functions and other analysis tools for switched and hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):475–482, 1998.

[21] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE Transactions on Automatic Control*, 43(1):31–45, 1998.

[22] M.S. Branicky, E. Dolginova, and N. Lynch. A toolbox for proving and maintaining hybrid specifications. In A. Nerode P. Antsaklis, W. Kohn and S. Sastry, editors, *Hybrid Systems IV*, number 1273 in LNCS, pages 18–30. Springer Verlag, 1997.

[23] R.W. Brockett. Hybrid models for motion control systems. In H.L. Trentelman and J.C. Willems, editors, *Perspectives in Control*. Birkhäuser, 1993.

[24] M. Broucke. Regularity of solutions and homotopy equivalence for hybrid systems. In *IEEE Conference on Decision and Control*, Tampa, FL, 1998.

[25] C.G. Cassandras and S. Lafortune. *Introduction to Discrete Event Systems*. Kluwer Academic Publishers, 1999.

[26] A. Chutinam and B. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In Frits W. Vaandrager and Jan H. van Schuppen, editors, *Hybrid Systems: Computation and Control*, number 1569 in LNCS, pages 76–90. Springer Verlag, 1999.

[27] T. Dang and O. Maler. Reachability analysis via face lifting. In S. Sastry and T.A. Henzinger, editors, *Hybrid Systems: Computation and Control*, number 1386 in LNCS, pages 96–109. Springer Verlag, 1998.

[28] J.M. Davoren and A. Nerode. Logics for hybrid systems. *Proceedings of the IEEE*, 88(7):985–1010, July 2000.

[29] C. Daws, A. Olivero, S. Trypakis, and S. Yovine. The tool KRONOS. In R. Alur, T. Henzinger, and E. Sontag, editors, *Hybrid Systems III*, number 1066 in LNCS, pages 208–219. Springer Verlag, 1996.

[30] R. De Carlo, M. Branicky, S. Pettersson, and B. Lennarston. Perspectives and results on the stability and stabilizability of hybrid systems. *Proceedings of the IEEE*, 88(7):1069–1082, July 2000.

[31] A. Deshpande, A. Gollu, and L. Semenzato. The SHIFT programming language for dynamic networks of hybrid automata. *IEEE Transactions on Automatic Control*, 43(4):584–587, April 1998.

[32] E. Dolginova and N. Lynch. Safety verification for automated platoon maneuvers: a case study. In Oded Maler, editor, *Proceedings of HART97*, number 1201 in LNCS, pages 154–170. Springer Verlag, Berlin, 1997.

[33] S. Engell, S. Kowalewski, C. Schulz, and O. Stursberg. Continuous-discrete interactions in chemical processing plants. *Proceedings of the IEEE*, 88(7):1050–1068, July 2000.

[34] A. F. Filippov. *Differential equations with discontinuous right-hand sides*. Kluwer Academic Publishers, 1988.

[35] G. Grammel. Maximum principle for a hybrid system via singular pertrubations. *SIAM Journal of Control and Optimization*, 37(4):1162–1175, 1999.

[36] M.R. Greenstreet and I. Mitchell. Integrating projections. In S. Sastry and T.A. Henzinger, editors, *Hybrid Systems: Computation and Control*, number 1386 in LNCS, pages 159–174. Springer Verlag, 1998.

[37] M. Heemels. *Linear Complementarity Systems: a Study in Hybrid Dynamics*. PhD thesis, Technische Universiteit Eindhoven, 1999.

[38] M. Heemels, H. Schumacher, and S. Weiland. Well-posedness of linear complementarity sysetms. In *Proc. 38th IEEE Conference on Decision and Control*, Phoenix, AZ, 1999.

[39] W. P. M. Heemels, B. De Schutter, and A. Bemporad. Equivalence of hybrid dynamical models. *Automatica*, 37(7):1085–1091, 2001.

[40] C. Heitmeyer and N. Lynch. The generalized railroad crossing: A case study in formal verification of real-time systems. In *Proc. ICCC Real-Time Systems Symposium*, San Juan, Puerto Rico, 1994.

[41] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata. In 27^{th} *Annual Symposium on the Theory of Computing, STOC'95*, pages 373–382. ACM Press, 1995.

[42] T. A. Henzinger, P. H. Ho, and H. Wong Toi. A user guide to HYTECH. In E. Brinksma, W. Cleaveland, K. Larsen, T. Margaria, and B. Steffen, editors, *TACAS 95: Tools and Algorithms for the Construction and Analysis of Systems*, number 1019 in LNCS, pages 41–71. Springer Verlag, 1995.

[43] J.E. Hopcroft, R. Motwani, and J.D. Ullman. *Introduction to automata theory, languages and computation*. Addison-Wesley Publishing, second edition, 2000.

[44] R. Horowitz and P. Varaiya. Control design of an automated highway system. *Proceedings of the IEEE*, 88(7):913–925, July 2000.

[45] B. Hu, X. Xu, A.N. Michel, and P.J. Antsaklis. Robust stabilizing control laws for a class of second order switched systems. *Systems & Control Letters*, 38:197–207, 1999.

[46] J. Imura and A. J. van der Schaft. Characterization of well-posedness of piecewise linear systems. *IEEE Transactions on Automatic Control*, 45(9):1600–1619, September 2000.

[47] K.H. Johansson, M. Egerstedt, J. Lygeros, and S.S. Sastry. On the regularization of Zeno hybrid automata. *Systems and Control Letters*, 38:141–150, 1999.

[48] K.H. Johansson, J. Lygeros, S.S. Sastry, and M. Egerstedt. Simulation of Zeno hybrid automata. In *IEEE Conference on Decision and Control*, pages 3538–3543, Phoenix, Arizona, U.S.A., December 7-10 1999.

[49] M. Johansson. *Piecewise linear control systems*. PhD thesis, Department of Automatic Control, Lund Institute of Technology, Sweden, March 1999.

[50] M. Johansson and A. Rantzer. Computation of piecewise quadratic lyapunov functions for hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):555–559, 1998.

[51] H.B. Khalil. *Nonlinear Systems*. Prentice Hall, third edition, 2001.

[52] X.D. Koutsoukos, P.J. Antsaklis, J.A. Stiver, and M.D. Lemmon. Supervisory control of hybrid systems. *Proceedings of the IEEE*, 88(7):1026–1049, July 2000.

[53] A. B. Kurzhanski and P. Varaiya. On reachability under uncertainty. *SIAM Journal of Control and Optimization*, 41(1):181–216, 2002.

[54] A.B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS, pages 202–214. Springer Verlag, 2000.

[55] G. Lafferriere, G.J. Pappas, and S.S. Sastry. O-minimal hybrid systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, March 2000.

[56] M. Lemmon. On the existence of solutions to controlled hybrid automata. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS, pages 229–242. Springer Verlag, New York, 2000.

[57] B. Lennartsson, M. Tittus, B. Egardt, and S. Pettersson. Hybrid systems in process control. *Control Systems Magazine*, 16(5):45–56, 1996.

[58] H.R. Lewis and C. Papadimitriou. *Elements of the Theory of Computation*. Prentice Hall, second edition, 1997.

[59] D. Liberzon, J.P. Hespanha, and A.S. Morse. Stability of switched systems: A Lie algebraic approach. *Systems and Control Letters*, 37(3):117–122, 1999.

[60] D. Liberzon and A.S. Morse. Basic problems in stability and design of switched systems. *IEEE Control Systems Magazine*, 19:59–70, October 1999.

[61] C. Livadas, J. Lygeros, and N.A. Lynch. High-level modeling and analysis of the traffic alert and collision avoidance system (TCAS). *Proceedings of the IEEE*, 88(7):926–948, July 2000.

[62] C. Livadas and N. Lynch. Formal verification of safety-critical hybrid systems. In S. Sastry and T.A. Henzinger, editors, *Hybrid Systems: Computation and Control*, number 1386 in LNCS, pages 253–272. Springer Verlag, 1998.

[63] J. Lygeros, K.H. Johansson, S.S. Sastry, and M. Egerstedt. On the existence of executions of hybrid automata. In *IEEE Conference on Decision and Control*, pages 2249–2254, Phoenix, Arizona, U.S.A., December 7-10 1999.

[64] J. Lygeros, K.H. Johansson, S.N. Simić, J. Zhang, and S.S. Sastry. Dynamical properties of hybrid automata. *IEEE Transactions on Automatic Control*, 48(1):2–17, January 2003.

[65] J. Lygeros and N. Lynch. Strings of vehicles: Modeling and safety conditions. In S. Sastry and T.A. Henzinger, editors, *Hybrid Systems: Computation and Control*, number 1386 in LNCS, pages 273–288. Springer Verlag, 1998.

[66] J. Lygeros, C.J. Tomlin, and S.S. Sastry. Controllers for reachability specifications for hybrid systems. *Automatica*, pages 349–370, March 1999.

[67] N. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[68] N. Lynch, R. Segala, F. Vaandrager, and H.B. Weinberg. Hybrid I/O automata. In *Hybrid Systems III*, number 1066 in LNCS, pages 496–510. Springer Verlag, 1996.

[69] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Theoretical Aspects of Computer Science*, number 900 in LNCS, pages 229–242. Springer Verlag, 1995.

[70] Z. Manna and A. Pnueli. *The Temporal Logic of Reactive and Concurrent Systems: specification*. Springer-Verlag, Berlin, 1992.

[71] Z. Manna and A. Pnueli. *Temporal Verification of Reactive Systems: Safety*. Springer-Verlag, New York, 1995.

[72] Zohar Manna and Henny Sipma. Deductive verification of hybrid systems using STeP. In S. Sastry and T.A. Henzinger, editors, *Hybrid Systems: Computation and Control*, number 1386 in LNCS, pages 305–318. Springer Verlag, 1998.

[73] Zohar Manna and the STeP group. STeP: The Stanford Temporal Prover. Technical Report STAN-CS-TR-94-1518, Computer Science Department, Stanford University, July 1994.

[74] S. E. Mattsson. On object-oriented modeling of relays and sliding mode behaviour. In *Proc. 13th IFAC World Congress*, volume F, pages 259–264, San Francisco, CA, 1996.

[75] S. E. Mattsson, M. Andersson, and K. J. Åström. Object-oriented modelling and simulation. In D. A. Linkens, editor, *CAD for Control Systems*, chapter 2, pages 31–69. Marcel Dekker Inc., New York, 1993.

[76] S. E. Mattsson, M. Otter, and H. Elmqvist. Modelica hybrid modeling and efficient simulation. In *IEEE Conference on Decision and Control*, Phoenix, AZ, 1999.

[77] A.S. Matveev and A.V. Savkin. *Qualitative theory of hybrid dynamical systems*. Birkhauser, Boston, 2000.

[78] R. May. *Stability and Complexity of Model Ecosystems*. Princeton University Press, Princeton, NJ, 1973.

[79] A.N. Michel and B. Hu. Towards a stability theory for hybrid dynamical systems. *Automatica*, 35:371–384, 1999.

[80] I. Mitchell, A.M. Bayen, and C.J. Tomlin. Validating a hamilton-jacobi approximation to hybrid system reachable sets. In M. Di Benedetto and A. Sangiovanni-Vincentelli, editors, *Hybrid Systems: Computation and Control*, number 2034 in LNCS, pages 418–432. Springer Verlag, 2001.

[81] I. Mitchell and C.J. Tomlin. Level set methods for computation in hybrid systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS, pages 310–323. Springer Verlag, 2000.

[82] A.S. Morse. Control using logic based switching. In A. Isidori, editor, *Trends in Control*, pages 69–114. Springer Verlag, 1995.

[83] D.L. Pepyne and C.G. Cassandras. Optimal control of hybrid systems in manufacturing. *Proceedings of the IEEE*, 88(7):1108–1123, July 2000.

[84] P.J. Antsaklis, Editor. Special issue on hybrid systems: Theory and applications. *Proceedings of the IEEE*, 88(7), July 2000.

[85] A. Pnueli and J. Sifakis Editors. Hybrid systems. *Theoretical Computer Science*, 138(1), 1995.

[86] A. Puri, P. Varaiya, and V. Borkar. ϵ-approximation of differential inclusions. In *IEEE Conference on Decision and Control*, pages 2892–2897, New Orleans, LA, 1995.

[87] P. Saint-Pierre. Approximation of viability kernels and capture basins for hybrid systems. In *European Control Conference*, pages 2776–2783, Porto, Portugal, September 4-7 2001.

[88] S.S. Sastry. *Nonlinear Systems: Analysis, Stability and Control*. Springer Verlag, New York, 1999.

[89] A. V. Savkin, E. Skafidas, and R.J. Evans. Robust output feedback stabilizability via controller switching. *Automatica*, 35:69–74, 1999.

[90] A. Savkin, Editor. Special issue on hybrid systems. *Systems and Control Letters*, 38(3), October 1999.

[91] J.M. Schumacher, A.S. Morse, C.C. Pandelides, and S. Sastry, Editors. Special issue on hybrid systems. *Automatica*, 35(3), March 1999.

[92] S. Simic, K.H. Johansson, S.S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In Nancy Lynch and Bruce H. Krogh, editors, *Hybrid Systems: Computation and Control*, number 1790 in LNCS, pages 421–436. Springer Verlag, 2000.

[93] L. Tavernini. Differential automata and their simulators. *Nonlinear Analysis, Theory, Methods and Applications*, 11(6):665–683, 1987.

[94] C.J. Tomlin, J. Lygeros, and S.S. Sastry. A game theoretic approach to controller design for hybrid systems. *Proceedings of the IEEE*, 88(7):949–969, July 2000.

[95] C.J. Tomlin, G.J. Pappas, and S.S. Sastry. Conflict resolution for air traffic management: a case study in multi-agent hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):509–521, 1998.

[96] V. I. Utkin. *Sliding Modes in Control and Optimization*. Springer-Verlag, Berlin, 1992.

[97] A.J. van der Schaft and H. Schumacher. Complementarity modeling of hybrid systems. *IEEE Transactions on Automatic Control*, 43(4):483–490, 1998.

[98] A.J. van der Schaft and H. Schumacher. *An Introduction to Hybrid Dynamical Systems*. Number 251 in Lecture Notes in Control and Information Sciences. Springer-Verlag, 1999.

[99] P. Varaiya. Smart cars on smart roads: problems of control. *IEEE Transactions on Automatic Control*, AC-38(2):195–207, 1993.

[100] M. Vidyasagar. *Nonlinear Systems Analysis*. Prentice Hall, second edition, 1992.

[101] H.B. Weinberg, Nancy Lynch, and Norman Delisle. Verification of automated vehicle protection systems. In *Hybrid Systems III*, number 1066 in LNCS, pages 101–113. Springer Verlag, 1996.

[102] M. Wicks, P. Peleties, and R. De Carlo. Switched controller synthesis for the quadratic stabilization of a pair of unstable linear systems. *European Journal of Control*, 4:140–147, 1998.

[103] S. Wiggins. *Introduction to Applied Nonlinear Dynamical Systems and Chaos*. Springer-Verlag, third edition, 1997.

[104] Howard Wong-Toi. The synthesis of controllers for linear hybrid automata. In *IEEE Conference on Decision and Control*, pages 4607–4613, San Diego, California, USA, December 10-12 1997.

[105] X. Xu and P.J. Antsaklis. Stabilization of second order LTI switched systems. *International Journal of Control*, 73:1261–1279, 2000.

[106] H. Ye, A. Michel, and L. Hou. Stability theory for hybrid dynamical systems. *IEEE Transactions on Automatic Control*, 43(4):461–474, 1998.

[107] J. Zhang, K.H. Johansson, J. Lygeros, and S.S. Sastry. Zeno hybrid systems. *International Journal of Robust and Nonlinear Control*, 11:435–451, 2001.