Université Grenoble I – Joseph Fourier
Sciences et Géographie

THÈSE

pour obtenir le grade de

DOCTEUR DE L'UNIVERSITÉ GRENOBLE I

Discipline : INFORMATIQUE

préparée au laboratoire **VERIMAG**

présentée et soutenue publiquement
par

Gerardo SCHNEIDER

le 05 juillet 2002

TITRE

Analyse Algorithmique de Systèmes Hybrides Polygonaux.

Directeurs de thèse :

Sergio Yovine et Eugène Asarin

JURY

Jacques Voiron	Président
Rajeev Alur	Rapporteur
Ahmed Bouajjani	Rapporteur
Sergio Yovine	Directeur de thèse
Eugène Asarin	Directeur de thèse
Vincent Blondel	Examinateur
Patrick Saint-Pierre	Examinateur

No te olvidés del pago
si te vas pá la ciudad
cuanti más lejos te vayas
más te tenés que acordar

Alfredo Zitarrosa

A mis padres

Acknowledgements

I give my thanks first to Joseph Sifakis for giving me the opportunity of doing my Ph.D. thesis at Verimag working with very high-qualified and excellent researchers. It was a pleasure to have as members of my jury Professors Rajeev Alur, Ahmed Bouajjani, Vincent Blondel, Patrick Saint–Pierre and Jacques Voiron, and I thank them all for their valuable comments and remarks about my work.

Some months before coming to Verimag I spent some time at UNU/IIST in Macao where I had my first contact with Real-Time systems and temporal logics. There I changed the direction of my research interests and for that I have to thank Prof. Zhou Chaochen and Xu Qiwen. Pablo and Victor are very good friends with whom I shared a lot of nice experiences in Macao both in science and in the life in general. They have also influenced my decision in coming to Verimag. Thanks to all of them. Even before that, Laira V. Toscani and Antônio C. da Rocha Costa have guided me in a M.Sc. course at UFRGS, Brazil. Thank you for your example, friendly guidance and continuous moral support and for what you have taught me about research.

I would especially like to thank my supervisors Sergio Yovine and Eugène Asarin for their moral and excellent technical support during my studies. Sergio and his wife Alejandra helped me a lot in my first year in many practical aspects as well as just being good friends, making me feel at home. I remember their invitations to have dinner with them almost every week and the discussions we use to have about many subjects (including, of course, many times about my research, late at night). I thank him also by his dedication, guidance and support to attend workshops and conferences.

It has been a great pleasure to work with Eugène. Without his permanent guidance and advise this thesis would not be possible. Even though he was (always) busy and sometimes tired I always found him in a good mood, ready to answer my questions and to talk about other interesting subjects (not necessarily so) related to my research topic. I do not have words to thank him for all the things he have taught me and for his optimism at any time. Thank you for all the time you spent reading and correcting many technical proofs and for proposing in many cases more elegant and shorter ones. *Merci, Monsieur!*

My special thanks to Gordon Pace for his friendship and for all the coffees we shared during more than a year during his post-doc here at Verimag and after that. I really enjoyed our discussions about literature, films, books and science at Verimag as well as in many Grenoble's nice *cafés* after watching some "alternative" films. He is also the responsible for introducing me to the programming language Haskell and for encouraging me to implement the algorithm I present in this thesis in Haskell. I debt him not only the time he spent to taught me the language but also the time he dedicated to share with me the implementation of SPeeDI. Thanks also for your invaluable help aiding me with my English.

I am pleased to acknowledge all the researchers, professors, colleagues and staff for making Verimag a very pleasant place to work. Ahmed, Oded, Pascal, Saddek and Yassine have in many opportunities answered technical questions about verification – thanks go to all of them. Thanks to Aurore, Cyril and Jean–Pierre for being good *collègues de bureau* and for their help with my French. Also Michaël and Pascal has helped me with my French: thank you. With Catalin, Conrado, Manuel, Michaël, Stavros, Thao and Yasmina we have shared not only research talks but many other things. It was a pleasure to meet you all as well as Liana and Ana, with whom we shared very nice *pause café*. I have to thank Nicolas for his continuous help with daily computer problems and to Catalin and Christos for their help with technical problems on LaTeX. Thanks to Catalin and Manuel for their friendship and for the helping hand they lent with my *déménagement*.

Outside Verimag, many people had made my life in Grenoble a pleasant one. I cannot give an exhaustive list here, but I would like to mention two of these persons. I specially thanks Cecilia for transmitting me her passion for Tango. *Gracias Cecilia por los hermosos momentos vividos juntos y por todo lo que me enseñastes.* Monique has been a very good friend with whom I have shared a lot of things mainly in the last months. *Merci Monique pour ton amitié.*

Finally, I want to thank my parents, for their example and love and for their constant support since my undergraduate studies.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

In the last decades daily life has been dominated by technological devices using computers or digital controllers. One source of complexity in such systems arises because these computers perform discrete operations while interacting with a physical environment which, in turn, has continuous dynamics.

These systems can be rather simple: a coffee machine, a thermostat, an elevator or a digital alarm, while others can be extremely complicated: air traffic management systems [135], robotic systems [7], manufacturing plants [69], automobiles [31], automated highway systems [126] and chemical plants [33, 32]. *Critical* systems are ones in which errors can have more serious consequences: errors in the behavior of a coffee machine are not comparable to ones on airplanes, for instance. In general, we would like to satisfy a number of properties depending on what the system (device) was built for. The problem of building complex systems with a certain degree of confidence in their correctness under any circumstance gave rise to the techniques of *simulation* and *testing*. These techniques are widely used and are very useful. However, they have the drawback of not being exhaustive, in the sense that whenever the number of cases to validate is high then it is not possible to test or simulate all the possible cases. Thus, these two techniques explore just *some* of the possible behaviors of the system and can only partially guarantee its correctness.

In the past few decades an alternative approach to simulation and testing has been developed, *formal verification*, that makes an *exhaustive exploration* of the system possible behaviors.

There are two approaches to formal verification: deductive and algorithmic.

- The *deductive approach* (or *theorem proving*) is based on the use of axioms and proof rules of a certain theory in order to prove that a certain property holds. The specification of the system as well as the property are written in an axiomatizable theory and then the idea is to prove that the property can be inferred from the system specification using the inference rules. The advantage of this method is that in principle it can handle very complex systems (even infinite-state ones) but the drawback is that it is not fully automatic. Another disadvantage is that a highly qualified and trained user is needed in order to encode the system and to manipulate and guide the proofs. Some well-known theorem provers are Coq [57, 88], Isabelle [120] and PVS [118].

- The *algorithmic approach* (or *model checking*) [54] is a technique for verifying finite state concurrent systems that can be made fully automatic. In general, a model of the system is written in a mathematical formalism (usually *transition systems* or *automata* are used) and the properties to be checked are expressed in a *temporal logic* [9, 61, 134, 125]. Then, verifying that the system satisfies a given property is reduced to verifying that the model of the system is a *model* (in the logical meaning) of the temporal logic formula expressing the property. Temporal logic model checking was introduced in the early eighties [55, 56, 129] but the first use of temporal logic as a formalism to reason about concurrency was in [121]. Many model checkers have been built in these past years based on different techniques and methods. For applications without timing constraints, SMV [110] and SPIN [85, 86] are two such tools. For real-time systems [143] some of the tools are: COSPAN [13], KRONOS [142], RT-SPIN [137] and Uppaal [34].

Some effort has been done in order to combine these two approaches. InVeSt [36, 35] and STeP [107] are two verification tools that do so.

Many mathematical and real problems can be modeled using *transition systems*, that is a basic model for verification. The above techniques and tools are then applied with success, whenever no technological restrictions limit their use. Other problems are better modeled as equations and in certain cases solving them involves finding isolated points - numbers - that satisfy the equation. In many other applications it is often the case that the unknown is itself a function; the equations obtained from this kind of problems are more complex and are known as *functional equations*. One important class of functional equations are *differential equations* [2, 83, 84] in which not only the unknown function appears but also its different order derivatives. The importance of differential equations is that solving various physical and technological problems can be reduced to finding solutions of such equations, which are usually curves in the plane or in space.

One important part of the general theory of differential equations is its *qualitative theory*. In some cases it is necessary to characterize the solution and the properties of a differential equation without solving it. In fact, for only a few simple equations a general solution can be explicitly given. So that gives rise to the problem of investigating the properties of the solutions of differential equations from the equation itself. The kind of properties that are usually interesting are *convergence* or *stability* of the curves around some singular points or closed curves. For example, do they lie in a bounded part of the plane, are there lines that separates the plane into regions that are not “connected”, are some of them closed curves, etc.? Such singular points, closed curves, *separatrix* lines, etc., are important objects giving qualitative information about the differential equation. The set of all such objects constitute what is called the *phase portrait* of the differential equation.

Sometimes it is useful and even desirable to model systems with uncertainties and disturbances and when building controllers it is important to explicitly specify the control variables. One can model such systems using differential equations with additional variables that represent control or disturbance. Other solution is to use *differential inclusions* [27]. Roughly speaking, differential inclusions are similar to differential equations but its right-hand side is a set instead of a single value. Thus, a differential equation is a particular case of a differential inclusion. In some other cases a problem is modeled not only by one differential equation (inclusion) but by a set of such equations (inclusions) that are somehow related via discrete variables.

A *hybrid system* is a system where both continuous and discrete behaviors interact with each other. A typical example is given by a discrete program that interacts with (controls, monitor, supervises) a continuous physical environment. Traditionally, the main preoccupation of computer scientists has been the study of discrete systems, using logic and discrete mathematics as a basis for reasoning. On the other hand, continuous models have been the subject of study of mathematicians and physicists, even though differential equations with discontinuous right hand side has also been considered in these communities (see [71, 72] and reference therein). Later on, control theoreticians developed theories and methods to solve problems on *Control Theory* about "switching systems", in which digital control is applied to switch between continuous laws. Hybrid systems are nowadays studied, from different points of view and using different approaches and methods, in Computer Science, Control Theory and Mathematics.

One widely used formalization for hybrid systems are *hybrid automata* [82], finite-state machines enriched with differential inclusions. Hybrid automata allow to model the discrete part of a hybrid system as transitions between the states of the machine and its continuous part with differential inclusions.

Some existing verification tools for hybrid systems are: CheckMate [52, 53], *d/dt* [62, 64], Hytech [8, 81] and VeriShift [42, 97].

1.2 Contributions

Problem. One of the main research areas in verification in general and in particular in hybrid systems is reachability analysis which comprises two closely related issues, namely, the study of decidability and the development of algorithms. It is important to realize that for many problems there is no algorithm which solves them. Such problems are called *undecidable* and are studied in *computability theory* [87, 101, 119, 131], that talk about limits of what can be decided by an algorithm. Much effort has been spent in identifying classes of decidable and undecidable problems and in drawing the separating border.

In this thesis we are concerned with the formal verification of hybrid systems following the algorithmic approach. In particular, we concentrate on two dimensional non-deterministic hybrid systems, namely *polygonal differential inclusion systems* (SPDIs). SPDIs are a class of nondeterministic systems that correspond to piecewise constant differential inclusions on the plane, for which we study the reachability problem as well as their qualitative behavior via the construction of their phase portrait. We also study some other classes of 2-dimensional hybrid systems for which we present some undecidability results.

Reachability. The first contribution of this thesis is the development of an algorithm for solving exactly the reachability problem of SPDIs. The approach is based on the combination of two techniques: the representation of the two-dimensional continuous-time dynamics as a one-dimensional discrete-time system (due to Poincaré), and the characterization of the set of qualitative behaviors of the latter as a finite set of *types of signatures.* As far as we know, this is the first application of *geometric* methods to non-deterministic systems.

The practical contribution of this thesis is the implementation of the reachability algorithm for SPDIs into a prototype of a tool called SPeeDI, that is a collection of utilities to manipulate and reason mechanically about SPDIs.

Phase Portrait. As a second contribution, we study phase portraits of SPDIs. It is not a priori clear what the phase portraits of such systems exactly are, mainly due to their intrinsic non-determinism. The *viability kernel* [24, 25] of a simple cycle is the set of starting points of trajectories which can keep rotating in the cycle forever. A *controllability kernel* is the set of all the points such that any point is reachable from any other. We give a non-iterative algorithm for computing the viability and controllability kernels of simple cycles and we study convergence properties of such objects. We conclude that controllability kernels yield an analog of Poincaré-Bendixson theorem for simple SPDI trajectories.

Undecidability. The third main contribution of this work is the study of the decidability of the reachability problem for other 2-dimensional hybrid systems. *Piecewise constant derivative systems* (PCDs) are hybrid systems for which the dynamics are defined by constant derivatives and such that the trajectories are continuous. A *hierarchical PCD* (HPCD) is a PCD with a finite number of discontinuities.

We show that the reachability problem for 2-dimensional HPCDs is as hard as the reachability problem for one dimensional *piecewise affine maps* (PAMs), which is still an open problem. On the same lines we prove that subclasses of two dimensional hybrid automata and PCDs defined on 2-dimensional manifolds are also equivalent to PAMs.

On the negative side we prove that adding a counter or "infinite patterns" makes the reachability problem undecidable for HPCDs. Two kinds of "infinite patterns" are considered, the first is the possibility of having infinite number of regions and the second one is the permission of having dynamics that determine periodic behaviors.

1.3 Related Work

Reachability. Hybrid systems have been widely studied in the last decade [10, 14, 15, 16, 67, 74, 103, 104, 136, 141]. In particular, for planar hybrid systems, [76] presents many examples and a general theory for modeling hybrid systems but no decidability issues are discussed. Most of the decidability results on algorithmic verification of hybrid systems proved in the literature are based on the existence of a finite and computable partition of the state space into classes of states which are equivalent with respect to reachability. This is the case for timed automata [6], certain classes of rectangular automata [80] and hybrid automata with linear vector fields [98]. Except for timed automata, these results rely on stringent hypothesis such as the resetting of variables along transitions.

Although analysis techniques based on the construction of a finite partition have been proposed [51], mainly all implemented computational procedures resort to (forward or backward) propagation of constraints, typically (unions of convex) polyhedra or ellipsoids [4, 17, 42, 63, 73, 97]. In general, these techniques provide semi-decision procedures: if the given final set of states is reachable, they will eventually terminate, otherwise they may fail to do so. This is a property of the techniques, not of the problem. In other words, these algorithms may not terminate for certain systems for which the reachability problem is indeed decidable. Nevertheless, they provide tools for computing (approximations of) the reach-set for large classes of hybrid systems with linear and non-linear vector fields.

Maybe the major drawback of set-propagation, reach-set approximation procedures is that little attention is paid to the geometric properties of the specific (class of) systems under analysis. To our knowledge, in the context of hybrid systems there are two lines of work in the direction of developing more "geometric" approaches. One is based on the existence of (enough) integrals and the ability to compute them all [51, 68]. These methods, however, do not necessarily result in decision procedures. The other, applicable to two-dimensional hybrid dynamical systems, relies on the topological properties of the plane, and explicitly focuses on decidability issues. This approach has been first proposed in [105], where, it is shown that the reachability problem for two-dimensional PCDs is decidable. This result has been extended in [139] for planar piecewise Hamiltonian systems.

In this thesis we follow an approach similar to the one proposed in [105]. Our reachability algorithm is not based on the computation of the reach-set but rather on the computation of the limit of individual trajectories. A key idea is the use of one-dimensional affine Poincaré maps [83, 115] for which we can easily compute the fixpoints. The decidability result of [105] fundamentally relies on the determinism of PCDs which implies that planar trajectories do not intersect themselves. This property is no longer true for differential inclusions. What is interesting, both for PCDs as well as for SPDIs, is that in both cases simple cycles can be *accelerated* in most cases. *Acceleration* is a well-known technique in verification that consists in computing, in one step, all the possible (maybe infinite) states that would be reachable in an unbounded number of steps, clearly saving computation time and space. This technique was applied for automata with counters [41] and for automata with queues [1, 40, 44].

SPeeDI is, as far as we know, the only verification tool that implements a "geometric" method for 2-dimensional hybrid systems. Tools exist that, in theory, can handle the same kind of systems as SPeeDI, like d/dt [62, 64] and HyTech [8, 81]. Even though these are "semi-algorithmic" hybrid system verification tools it is interesting to compare SPeeDI, which implements a decision algorithm, with HyTech, for instance. HyTech is a tool capable of treating hybrid linear systems of any dimension, making it much more general than SPeeDI, which is limited to two-dimensional systems without resets. On the other hand, SPeeDI implements acceleration techniques (based on the resolution of fixpoint equations) which yield a complete decision procedure for SPDIs. Also, SPeeDI does not handle arbitrary polyhedra, but only polygons and line segments. For these reasons, comparing the performance of the two tools is meaningless and no fair benchmarking is really possible.

Differential Inclusions and Viability Theory. The use of differential inclusions as a tool for modeling uncertainties has been known for more than 70 years. The first works have been published by Marchaud [108] and Zaremba [144] in the early thirties. An extended survey can be found in [130] where a differential inclusion solver is presented. A more complete treatment is given in the book of Aubin and Cellina [27].

One of the reasons for a renewed interest in differential inclusions was the development of *viability theory* [24, 25]. To quote [24], viability theory can be characterized as follows:

> "The main purpose of viability theory is to explain possible viable evolutions of a system, determined by given nondeterministic dynamics and state constraints, to reveal the concealed feedbacks which allow the system to be regulated and provide selection mechanisms for implementing them."

Viability theory has applications in biology, economics, cognitive sciences, games, control theory, etc., and more recently in hybrid systems [29].

Phase Portrait. Real analysis [2, 94] was the tool used for studying dynamical problems for many years, until the French mathematician H. Poincaré [122, 123, 124] and the Russian mathematician A.M. Lyapunov [102] started doing a qualitative analysis of differential equations [50, 75, 83], more than 100 years ago, from the requirements of mechanics and astronomy.

There have been very few results on the qualitative properties of trajectories of hybrid systems [22, 26, 65, 95, 96, 109, 133]. In particular, the question of defining and constructing phase portraits of hybrid systems has not been directly addressed except in [109], where phase portraits of deterministic systems with piecewise constant derivatives are explored and more recently in [28, 30] where a characterization of viability and invariance kernels were given for impulsive differential inclusions.

Undecidability. Although there has been intense research activity in the last years in the domain of hybrid systems, there is no clear boundary between what is decidable and what is not on such systems. It is well known that for particular cases the reachability question is decidable. In [6] it was shown that reachability is decidable for timed automata (TA) that is a particular case of hybrid automata (all the variables have slope 1) and in [106] the decidability of the same problem for 2-dimensional PCDs was proved. In [5, 116] the decidability of the reachability problem for multi-rate automata (a hybrid automata such that the variables run at any constant slope) was proven and in [80, 127] the same holds for rectangular automata under a number of restrictions. More decidability results were given in [43] for *timed graph* with duration variables and in [45] for 2-dimensional *extended integrator graphs.*

On the other hand and not surprisingly, many undecidability results were given. In [80] it was shown that relaxing some restrictions the reachability problem for some classes of rectangular automata become undecidable. In [4, 5, 6, 11, 90, 138] it was shown that the reachability question for many different extensions of TA with some non-clock variables is undecidable under some restrictions. Some other undecidability results were given for low (three or less) dimensional spaces. In [19] it was shown that the reachability problem for 3-dimensional PCDs is undecidable whereas in [113] it was proved that Turing machines (TMs) can be simulated by dynamical systems with piecewise affine functions (in three dimensional spaces). In [93], two elementary functions were constructed that simulates TMs. See reference therein for other undecidability results. Among other results, in [49] it was shown that smooth ODEs in $\mathbb{R}^2$ can simulate arbitrary TMs and in [92] it was proved that TMs can be simulated by 2-dimensional PAMs and by other variations of PAMs. In [46] the algorithmic complexity of hybrid and dynamical systems was analyzed, in particular it was studied the frontier between decidability and undecidability for low dimensional systems for some problems like the controllability of commuted linear systems in dimension two that are related to the mortality problem for two dimensional matrices and to the reachability problem for one dimensional PAMs. Finally, in [38] the decidability of other problems of hybrid systems different from reachability, like stability and controllability, were analyzed.

1.4 Thesis Outline

This thesis is divided in 10 chapters, including this introductory chapter and a concluding one.

In Chapter 2 we concentrate on *hybrid systems* and in particular on two dimensional ones. We first recall the definition of *hybrid automata* as well as the definition of two other 2-dimensional hybrid systems: *piecewise constant derivative systems* and *planar multi-polynomial systems* (PMP). We define next the class of systems we are going to deal with, namely *polygonal differential inclusion systems* (SPDIs).

In Chapter 3 we present a useful class of functions called *truncated affine multi-valued operators* (TAMF) that serves as a technical basis for characterizing *successors* and *predecessors* operators that are used in the reachability algorithm for SPDIs presented in the fifth Chapter.

In Chapter 4 we present the difficulties that arise when trying to solve the reachability problem for SPDIs and we show how to overcome these difficulties doing a qualitative analysis of trajectory segments. We abstract trajectories into *types of signatures* and we show that the abstraction preserves reachability.

Chapter 5 is concerned with the decision procedure for the reachability problem of SPDIs.

Given, for instance, two points in an SPDI, the reachability question is: Is one point reachable from the other? We show how a case analysis simplifies the treatment of cycles and how to take advantage of the fact that *successors* are TAMF in order to *accelerate* cycles. We finally present our reachability algorithm and we prove its soundness and completeness together with some examples.

In Chapter 6 we study the problem of defining and constructing the phase portrait of SPDIs. We recall the definition of *viability* kernel of a simple cycle and we introduce the notion of *controllability* kernel. We give a non-iterative algorithm for computing them both. Moreover, we prove that any infinite trajectory performing forever the same cyclic pattern converges to the controllability kernel of the cycle. The phase portrait of SPDIs is obtained computing all the viability and controllability kernels.

Chapter 7 is dedicated to the study of the (un)decidability of other classes of 2-dimensional hybrid systems, small variations of PCDs. We show that the reachability problem for 2-dimensional HPCDs is as hard as the reachability problem for *piecewise affine maps* (PAMs) for which the question is still open. We also prove that other similar classes, subclasses of two dimensional hybrid automata and PCDs defined on 2-dimensional manifolds, are also equivalent to PAMs. Besides that, we prove that the reachability problem for HPCDs is undecidable when adding a counter or allowing some kind of "infinite pattern".

In Chapter 8 we present SPeeDI, a tool that implements the reachability algorithm given in the fifth Chapter.

Chapter 9 is concerned with *general SPDIs* (GSPDIs) that are extensions of SPDIs allowing more general trajectory segments. We prove that reachability is decidable for GSPDIs.

Finally, in Chapter 10 we present the conclusions and we discuss about future work.

Chapter 2

Hybrid Systems

In this chapter we introduce the formal background for the rest of the thesis. We recall the definition of *hybrid automata* as a formalism for representing hybrid systems and of linear hybrid and rectangular automata. We also recall the definition and decidability results of two other planar hybrid systems for which the geometric method has already been applied, namely *piecewise constant derivative systems* (PCDs) and *planar multi-polynomial systems* (PMPs) and we define *polygonal differential inclusion systems* (SPDIs), a class of nondeterministic systems that correspond to piecewise constant differential inclusions on the plane.

Organization of the chapter: In the first section we define some basic concepts and in section two we recall the definition of hybrid automata and of two subclasses: linear hybrid automata and rectangular automata. In section three we present PCDs and PMPs and in the fourth section we define our class of hybrid systems: SPDI. In the last section we conclude and we point out some related work.

2.1 Preliminaries

Let $\mathbf{a} = (a_1, a_2), \mathbf{x} = (x_1, x_2) \in \mathbb{R}^2$ and $\lambda \in \mathbb{R}$. We define:

Definition 1 *The* inner product *of two vectors* $\mathbf{a} = (a_1, a_2)$ *and* $\mathbf{x} = (x_1, x_2)$ *is defined as* $\mathbf{a}\,\mathbf{x} = a_1x_1 + a_2x_2$. $\lambda\mathbf{x} = (\lambda x_1, \lambda x_2)$ *is the product of a vector by an scalar* λ *and* $|\mathbf{x}| = \sqrt{\mathbf{x}\,\mathbf{x}}$ *is the* 2-norm. ■

We denote by $\hat{\mathbf{x}}$ the vector $(x_2, -x_1)$ obtained from $\mathbf{x}$ by rotating clockwise by the angle $\pi/2$. Notice that $\mathbf{x}\,\hat{\mathbf{x}} = 0$.

Definition 2 *The* distance *between two points* $\mathbf{x}$ *and* $\mathbf{y}$ *is defined to be* $|\mathbf{x} - \mathbf{y}|$. *For* $\epsilon > 0$, *the* ϵ-neighborhood *of* $\mathbf{x}$ *is* $B_\epsilon(\mathbf{x}) = \{\mathbf{y} \mid |\mathbf{x} - \mathbf{y}| < \epsilon\}$. *The* interior *of* $X \subseteq \mathbb{R}^2$, *denoted by* ***int***(X), *is the set of* $\mathbf{x} \in X$ *for which there exists* $\epsilon > 0$ *such that* $B_\epsilon(\mathbf{x}) \subseteq X$. ■

For $\mathbf{x}_1, \ldots \mathbf{x}_n \in \mathbb{R}^2$ a *linear* combination is a vector $\mathbf{x} = \sum_{i=1}^{n} \lambda_i \mathbf{x}_i$ for some $\lambda_i \in \mathbb{R}$. A *positive combination* is a linear combination with $\lambda_i \geq 0$ for every i. The positive *hull* of a set $X \subseteq \mathbb{R}^2$ is the set of all positive combinations of points in X. A (closed) *half-space* is the set of all points $\mathbf{x}$ satisfying $\mathbf{a}\,\mathbf{x} \leq b$. A *convex closed polygonal set* P is the intersection of finitely many half-spaces. An *edge* e is a segment of line in $\mathbb{R}^2$.

Let S be a finite index set and $\mathbb{P} = \{P_s\}_{s \in S}$ be a finite set of convex closed polygonal sets, called *regions*, such that:

1. For all $s \in S$, $\mathtt{int}(P_s) \neq \emptyset$;
2. For all $s \neq r \in S$, $\mathtt{int}(P_s \cap P_r) = \emptyset$;
3. $\bigcup_{s \in S} P_s = \mathbb{R}^2$.

Condition 1 states that regions are full dimensional. Condition 2 says that the intersection between two regions is either empty, an edge, or a point, whereas the third condition states that the regions covers the whole space. Thus, $\mathbb{P}$ is a *partition* of the plane[1].

Remark. All the above definitions were given in two dimensions since, all the hybrid systems we are working with in this thesis will be 2 dimensional, with the exception of hybrid automata that are defined for n dimensions.

We denote by $E(P)$ the set of edges of the form $e = P \cap P'$ with $P \neq P'$ and by $V(P)$ the set of vertices of the form $v = e \cap e'$ with $e, e' \in E(P)$. Let $\mathtt{int}(E(P)) = \{\mathtt{int}(e) \mid e \in P\}$ be the set of all the *open* edges of P, then let $EV(P) = \mathtt{int}(E(P)) \cup V(P)$ be the set of all the vertex and open edges of P.

2.2 Hybrid Automata

A *hybrid system* is a dynamical system that combines discrete and continuous components. A natural model for hybrid systems is *hybrid automata* [82] that are automata such that at each discrete location the dynamics is governed by differential equations (over continuous variables) and whose transitions (between locations) are enabled by conditions on the values of the variables. There are many (more or less) equivalent definitions of hybrid systems/automata (see for example [4, 82, 133]). We adopt in this chapter the following definition.

Definition 3 *An* n-dimensional hybrid automaton *is a sextuple* $\mathcal{H} = (\mathcal{X}, Q, f, \mathsf{I}_0, \mathsf{Inv}, \delta)$ *where*

- $\mathcal{X} \subseteq \mathbb{R}^n$ *is the* continuous state space. *Elements of* $\mathcal{X}$ *are written as* $\mathbf{x} = (x_1, x_2, \ldots, x_n)$*;*
- Q *is a finite set of* discrete locations*;*

[1] Condition (3) can be relaxed, and then we can consider partition of a subset of the plane.

- $f : Q \to (\mathcal{X} \to \mathbb{R}^n)$ *assigns a continuous vector field on* $\mathcal{X}$ *to each discrete location. While in discrete location* $\ell \in Q$*, the evolution of the continuous variables is governed by the differential equation* $\dot{\mathbf{x}} = f_\ell(\mathbf{x})$*. We say that the differential equation defines the* dynamics *of location* ℓ*;*

- *The* initial condition $\mathsf{I}_0 : Q \to 2^{\mathcal{X}}$ *is a function that for each state defines the initial values of the variables of* $\mathcal{X}$*;*

- *The* invariant $\mathsf{Inv} : Q \to 2^{\mathcal{X}}$*;* $\mathsf{Inv}(\ell)$ *is the condition that must be satisfied by the continuous variables in order to stay in location* $\ell \in Q$*;*

- δ *is a set of* transitions *of the form* $tr = (\ell, g, \gamma, \ell')$ *with* $\ell, \ell' \in Q$*. Such a quadruple means that a transition from* ℓ *to* ℓ' *can be taken whenever the* guard g *is satisfied and then the* reset γ *is applied. The guard is a predicate* $g \in 2^{\mathcal{X}}$ *and the reset is a function* $\gamma : \mathcal{X} \to 2^{\mathcal{X}}$*.* ■

Remark. In the above definition the evolution of the continuous variables are governed by differential equations. The dynamics can be defined in a more general way using differential inclusions instead of differential equations. We have chosen this definition in order to make it simpler, but we will see that for the particular case of SPDIs, differential inclusions will be used (see section 2.4).

We will denote by $\mathsf{V} = \{x_1, x_2, \ldots, x_n\}$ the set of the elements $\mathbf{x} \in \mathcal{X}$ ($\mathbf{x} = x_1, x_2, \ldots, x_n$).

A hybrid automaton $\mathcal{H}$ is said to be *deterministic* if for every location ℓ and any initial condition $\mathbf{x}_0 \in \mathsf{I}_0$ there exists at most one solution to the equation $\dot{\mathbf{x}} = f_\ell(\mathbf{x})$.

A *state* is a pair $(\ell, \mathbf{x})$ consisting of a location $\ell \in Q$ and $\mathbf{x} \in \mathcal{X}$. We write Σ for the set of states. At any time, the state of a hybrid automaton is given by a location and the values for all variables. A state can change in two ways: (1) by *discrete* and *instantaneous* transition that changes both the location and the values of the variables according to the transition relation, and (2) by a *time delay* that changes only the values of the variables according to the dynamics of the current location.

The system may stay at a location only if the invariant is true, and a transition must be taken before the invariant becomes false.

Hybrid automata can be seen as directed graphs whose arcs represent transitions and whose vertices represent discrete locations.

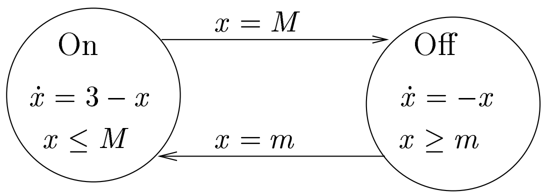


Figure 2.1: A model for the thermostat.

Example 1 Let us consider a thermostat as a simple example of hybrid automata. The thermostat controls the temperature of a room, sensing continuously (via a thermometer) the temperature and turning a heater on and off. The goal is to maintain the temperature of the room between m and M degrees and in order to do that the heater is kept on as long the temperature is below M and it is switched off whenever the temperature is above M. The room temperature and the thermostat can be seen as a hybrid system and can be modeled as a hybrid automata. The temperature is governed by differential equations. The hybrid automaton consists then in two locations On and Off. In location On the heater is on and the temperature rises according to the dynamics $\dot{x} = 3 - x$ while in location Off the temperature goes down following the dynamics $\dot{x} = -x$. Figure 2.1 shows the above automaton. ■

We introduce now the notion of *trajectory segment* on each location of a hybrid automaton.

Definition 4 *Given a location ℓ, an ℓ-*trajectory segment *on some interval $[0,T] \subseteq \mathbb{R}$, is a continuous and derivable function $\xi(\cdot)$ on $(0,T)$ such that for all $t \in (0,T)$, $\xi(t) \in \mathsf{Inv}(\ell)$ and $\dot{\xi}(t) = f_\ell(\xi(t))$.* ■

We can define the notion of *trajectory segment* as a sequence of ℓ-trajectory segments that are "connected" via the reset functions.

Definition 5 *A* trajectory segment *of a hybrid automaton $\mathcal{H}$ is a function $\Theta : [0,T] \to Q \times \mathcal{X}$, $\Theta(t) = (\ell(t), \xi(t))$ such that there exists a sequence of times values $t_0 = 0 < t_1 < \ldots < t_n = T$ for which the following holds for each $1 \leq i \leq n$:*

1. *ℓ is constant on (t_i, t_{i+1}) (we describe its value there by ℓ_i) and ξ is derivable on (t_i, t_{i+1}), it is right continuous and with left limits (cadlag) everywhere;*
2. *There is a transition $(\ell_i, g, \gamma, \ell_{i+1}) \in \delta$ such that $\xi^-(t_{i+1}) \in g(\ell_i, \ell_{i+1})$ and $\xi(t_{i+1}) = \gamma(\xi^-(t_{i+1}))$, where $\xi^-(t)$ is the left limit of $\xi(t)$;*
3. *For any $0 \leq i \leq n$, for any $t \in (t_i, t_{i+1})$, $\dot{\xi}(t) = f_{\ell(t)}(\xi(t))$.* ■

If $T = \infty$, a trajectory segment is called a trajectory. ■

Given $\xi(0) = \mathbf{x}_0$ and $\xi(t) = \mathbf{x}_f$, we say that ξ is a trajectory segment from $\mathbf{x}_0$ to $\mathbf{x}_f$.

Remark. In the literature sometimes the notion of a *run* of a hybrid automata is introduced instead of that of trajectory.

Example 2 Let us consider again the thermostat presented on Example 1. In this example, for a given initial condition $x(0) = K$, the solution of the differential equations for the On and Off locations can be analytically solved: $x(t) = Ke^{-t} + 3(1 - e^{-t})$ and $x(t) = Ke^{-t}$ respectively.

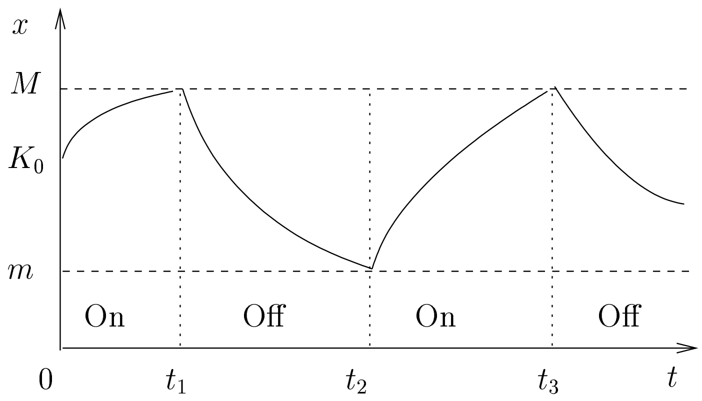


Figure 2.2: A typical trajectory of the room temperature.

Thus a typical trajectory, from an initial condition $x(0) = K_0$ is depicted in Figure 2.2, where starting with a temperature $m \leq K_0 \leq M$ (with the heater on), it evolves following the equation $x(t) = K_0 e^{-t} + 3(1 - e^{-t})$. After t_1 time, the temperature reaches M and the heater is then turned off making the heater change its mode to Off. Then the temperature evolves following the equation $x(t) = Me^{-t}$, until it reaches m at time t_2 and so on and so forth. ■

We recall now two particular subclasses of hybrid automata: linear hybrid automata and rectangular automata.

2.2.1 Linear hybrid automata

A *linear term* over the set V of variables is a linear combination of the variables in V with integer coefficients. If t_1 and t_2 are linear terms, then $t_1 \leq t_2$ is a *linear inequality*.

Definition 6 *A hybrid automaton $\mathcal{H}$ is* linear [4, 82] *if the following restrictions are met.*

1. *The initial and invariant conditions as well as the guard are boolean combinations of linear inequalities;*

2. *The dynamics are defined by differential equations of the form $\dot{x} = k_x$, one for each variable $x \in \mathsf{V}$, where $k_x \in \mathbb{Z}$ is an integer constant. We say that k_x is the* slope *(or* rate*) of the variable x at a given location.* ■

There are some interesting cases of linear hybrid automata.

- We say that a variable x is a *memory cell* (or a *discrete variable*) if it has slope 0 in every location of $\mathcal{H}$. A *discrete system* is a linear hybrid automaton all of whose variables are discrete.

- A discrete variable x is a *proposition* if it can take values on $\{0,1\}$ and x is a *clock* if it has slope 1 in every location. A *timed automaton* [6] is a linear hybrid automaton whose variables are just clocks or propositions and the linear expressions are boolean combinations of inequalities of the form $x\#c$ or $x-y\#c$ where c is a nonnegative integer and $\# \in \{<,\leq,=,>,\geq\}$.

- A variable x is a *skewed clock* if there is a rational $k \in \mathbb{Q}\backslash\{0,1\}$ such that x has slope k in each location. A *multi-rate timed system* is a linear hybrid automaton all of whose variables are propositions and skewed clocks. A *n-rate timed system* is a multi-rate timed system whose skewed clocks proceed at n different rates.

- The variable x is a *two-slope clock* if there is a rational k such that for each location $\dot{x}=k$ or $\dot{x}=1$. A *stopwatch*[2] is a two-slope clock with $k=0$.

Two other special cases of linear hybrid automata are Rectangular automata and PCDs.

2.2.2 Rectangular automata

A *rectangle of dimension* n $R=\prod_{1\leq i\leq n} I_i$ is the product of n intervals $I_i \subseteq \mathbb{R}$ of the real line with rational or infinite extremities. We say that the rectangle is *bounded* if each interval I_i, for $1\leq i\leq n$ is bounded.

Definition 7 *A hybrid automaton is a* rectangular automaton [82, 80, 127] *if the following holds.*

1. *For each location ℓ, the initial condition I_0 has the form $\bigwedge_{1\leq i\leq n} x_i \in I_i$ for each variable $x_i \in \mathsf{V}$ and I_i an interval of a rectangle R_{I_0};*
2. *Idem for the invariant condition Inv (for eventually different intervals I_i of a rectangle R_{Inv});*
3. *For each location ℓ, the dynamics has the form $\dot{x}_i \in I_i$ for each variable $x_i \in \mathsf{V}$ and I_i an interval of a rectangle R_{A};*
4. *For each transition $tr=(\ell,g,\gamma,\ell')$, the guard g has the form $\bigwedge_{1\leq i\leq n} x_i \in I_i$ for each variable $x_i \in \mathsf{V}$ and I_i an interval of a rectangle R_g.* ■

2.3 Some Planar Hybrid Systems

2.3.1 Piecewise constant derivative systems

We recall now a class of hybrid systems called *piecewise constant derivative systems* (PCDs), that are hybrid systems for which the dynamics is defined by constant derivatives and such

[2]Stopwatches are also known in the literature as *integrators*.

that the trajectories are continuous. PCDs are interesting since even though their dynamics is rather simple, their behavior is quite rich.

Let $\mathbb{F} = \{\phi_s\}_{s \in S}$ be a family of constant vectors, i.e. $\phi_s = \mathbf{c}_s$ and $\mathbb{P}$ be a partition of the plane. We have then the following definition.

Definition 8 *A* piecewise constant derivative system (PCD) [19, 106] *is a pair* $\mathcal{H} = (\mathbb{P}, \mathbb{F})$*. Each region* P_s *has dynamics* $\dot{\mathbf{x}} = \mathbf{c}_s$ *for* $\mathbf{x} \in P_s$ *(given a generic region* P *we also use the notation* $\phi(P)$*).* ■

Whenever $\mathbb{P}$ is a partition of a proper subspace of the plane, then we said that the PCD is *bounded.*

Consider a region P and an edge $e \in E(P)$. We say that e is an *entry* of P if for all $\mathbf{x} \in \mathtt{int}(e)$ and for $\mathbf{c} = \phi(P)$, $\mathbf{x} + \mathbf{c}\epsilon \in P$ for some $\epsilon > 0$. We say that e is an *exit* of P if the same condition holds for some $\epsilon < 0$. We denote by $In(P) \subseteq E(P)$ the set of all entries of P and by $Out(P) \subseteq E(P)$ the set of all exits of P.

Let $\mathbf{x} \in V(P)$ be a common vertex of two edges e and e'. $\mathbf{x}$ is an *entry point* to P if both e and e' are entry edges; it is an exit point if both e and e' are exit edges. In fact, vertices can be seen as a particular kind of edges, with exactly one point and in what follows we consider them as edges, hence the term *edge* should be understood as belonging to the set $EV(P)$ for some P. If needed, the difference between edges in V and E will be explicitly specified.

As for hybrid automata, we define now the notion of *trajectory.*

Definition 9 *A* trajectory segment *in some interval* $[0, T] \subseteq \mathbb{R}$*, with initial condition* $\mathbf{x} = \mathbf{x}_0$*, is a continuous and almost-everywhere (everywhere except on finitely many points) derivable function* $\xi(\cdot)$ *such that* $\xi(0) = \mathbf{x}_0$ *and for all* $t \in (0, T)$*:*

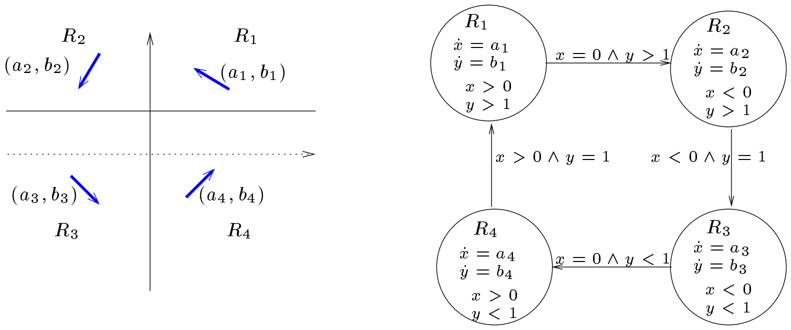
1. *if* $\xi(t) \in \mathtt{int}(P)$ *then* $\dot{\xi}(t)$ *is defined and* $\dot{\xi}(t) = \phi(P)$*;*
2. *if* $\xi(t) \in e$ *and* $e \in In(P)$ *then* $\dot{\xi}^+(t)$ *is defined and* $\dot{\xi}^+(t) = \phi(P)$*, where* $\dot{\xi}^+(t) = \frac{d^+\xi}{dt}$ *is the right derivative of* ξ*.* ■

If $T = \infty$*, a trajectory segment is called a* trajectory*.* ■

Let $\mathsf{V} = \{x, y\}$. Notice that PCDs can be viewed as linear hybrid automata without reset, such that the locations are the regions P_s; the invariant condition Inv is given by the polygon defined by the region P; for each variable $x \in \mathsf{V}$, $f_\ell(x) = \dot{x} = c$, where c is a rational constant; and the guards are the edges.

Example 3 In Figure 2.3 a simple PCD and its corresponding hybrid automata are shown. ■

PCDs are more general than timed automata since each variable has a uniform slope that



(a) (b)

Figure 2.3: (a) A simple PCD; (b) Its corresponding hybrid automata.

can change in each location, and is more restrictive in the sense that no discrete jumps (reset) are allowed. The trajectories of the system are then continuous but not smooth. This requirement makes PCDs to be closer to continuous dynamical systems and then more suitable for a topological and geometrical analysis.

The point-to-point reachability problem for PCDs is the following: *Given a PCD $\mathcal{H}$ and two points $\mathbf{x}_0$ and $\mathbf{x}_f$, is there a trajectory segment starting on $\mathbf{x}_0$ and finishing on $\mathbf{x}_f$?*

It is well known that the above problem (and its edge-to-edge and region-to-region variants) is decidable in two dimensions [19, 105] and undecidable for higher dimensions [19, 18].

2.3.2 Planar multi-polynomial systems

Other interesting class of planar hybrid systems are *planar multi-polynomial systems* [139] (PMPs), that are systems defined in a polygonal partition of the plane (as for PCDs) for which the dynamics is given by a vector field whose solutions can be described by polynomials of arbitrary degree.

The trajectory segments within the regions here can be described by polynomials of arbitrary degree (without singularities in each region P of the polygonal partition), i.e. for some polynomial $H(x, y)$ all trajectory segments satisfy equation $H(x, y) = C$ for some constant $C \in \mathbb{R}$ and such that all trajectories that enter the region P also leave it and vice-versa. It can be shown [139] that those trajectories correspond to the following equations:

$$\begin{cases} \dot{x} = & \frac{\partial H(x,y)}{\partial y} \\ \dot{y} = & -\frac{\partial H(x,y)}{\partial x} \end{cases}$$

The reachability problem for PMPs is decidable [139].

Remark. Notice that a 2-dim PCD is a particular case of a PMP, with $H_i(x, y) = -b_i x + a_i y$, that is, the dynamics on each region R_i is defined by:

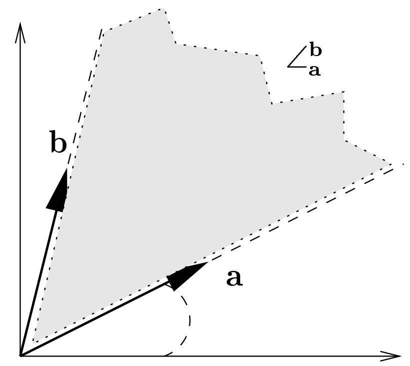


Figure 2.4: Positive hull of $\{\mathbf{a},\mathbf{b}\}$ with $\hat{\mathbf{a}}\ \mathbf{b} < 0$.

$$\begin{cases} \dot{x} = & \frac{\partial H_i(x,y)}{\partial y} & = a_i \\ \dot{y} = & -\frac{\partial H_i(x,y)}{\partial x} & = b_i \end{cases}$$

2.4 Polygonal Differential Inclusions (SPDIs)

In control theory it is useful to model systems where the control variable appears explicitly. Systems with uncertainties and disturbances are also interesting. One can model such systems using differential equations of the form $\dot{x} = f(x,u)$ where $u \in U$ is a control or a disturbance. Other solution is to use differential inclusions. A differential inclusion has the form $\dot{x} \in f(x)$, where f is a set-valued function ($f : \mathbb{R}^n \longrightarrow \wp(\mathbb{R}^n)$). Hence, a differential equation $\dot{x} = f(x,u)$ where $u \in U$ is a control or a disturbance can be represented as a differential inclusion $\dot{x} \in g(x)$ where $g(x) = \{f(x,u) \mid u \in U\}$ [128]. The differential inclusion $\dot{x} \in g(x)$ captures every possible behavior of f.

In this section we define SPDIs, a class of nondeterministic systems that correspond to piecewise constant differential inclusions on the plane. Before defining SPDI we need the following definition.

Definition 10 *An* angle $\angle_{\mathbf{a}}^{\mathbf{b}}$ *on the plane, defined by two non-zero vectors* $\mathbf{a},\mathbf{b}$ *is the set of all positive linear combinations* $\mathbf{x} = \alpha\ \mathbf{a} + \beta\ \mathbf{b}$, *with* $\alpha, \beta \geq 0$, *and* $\alpha + \beta > 0$. *We can always assume that* $\mathbf{b}$ *is situated in the counter-clockwise direction from* $\mathbf{a}$. ∎

That is, given $A = \{\mathbf{a},\mathbf{b}\}$, with $\hat{\mathbf{a}}\ \mathbf{b} < 0$, $\angle_{\mathbf{a}}^{\mathbf{b}}$ is the positive hull of A (Fig. 2.4).

Let $\mathbb{F} = \{\phi_s\}_{s \in S}$ be such that ϕ_s is the positive hull of two vectors $\mathbf{a}_s$ and $\mathbf{b}_s$ with $\hat{\mathbf{a}}_s\ \mathbf{b}_s < 0$ and $\mathbb{P}$ be a partition of the plane.

A *polygonal differential inclusion system* (SPDI) consists of a partition of the plane into convex polygonal regions, together with a differential inclusion associated with each region.

More formally,

Definition 11 *A* polygonal differential inclusion system *(SPDI) is a pair $\mathcal{H} = (\mathbb{P}, \mathbb{F})$. Each region P_s has dynamics $\dot{\mathbf{x}} \in \phi_s$ for $\mathbf{x} \in P_s$ (given a generic region P we also use the notation $\phi(P)$).* ■

As an example consider the problem of a swimmer trying to escape from a whirlpool in a river.

Example 4 The dynamics $\dot{\mathbf{x}}$ of the swimmer around the whirlpool is approximated by the piecewise differential inclusion defined as follows. The zone of the river nearby the whirlpool is divided into 8 regions $R_1, \ldots, R_8$. To each region R_i we associate a pair of vectors $(\mathbf{a}_i, \mathbf{b}_i)$ meaning that $\dot{\mathbf{x}}$ belongs to their positive hull:

- $\mathbf{a}_1 = \mathbf{b}_1 = (1, 5)$,
- $\mathbf{a}_2 = \mathbf{b}_2 = (-1, \frac{1}{2})$,
- $\mathbf{a}_3 = (-1, \frac{11}{60})$ and $\mathbf{b}_3 = (-1, -\frac{1}{4})$,
- $\mathbf{a}_4 = \mathbf{b}_4 = (-1, -1)$,
- $\mathbf{a}_5 = \mathbf{b}_5 = (0, -1)$,
- $\mathbf{a}_6 = \mathbf{b}_6 = (1, -1)$,
- $\mathbf{a}_7 = \mathbf{b}_7 = (1, 0)$,
- $\mathbf{a}_8 = \mathbf{b}_8 = (1, 1)$.

The corresponding SPDI is illustrated in Fig. 2.5-(a). ■

Definition 12 *A* trajectory segment *in some interval $[0, T] \subseteq \mathbb{R}$, with initial condition $\mathbf{x} = \mathbf{x}_0$, is a continuous and almost-everywhere (everywhere except on finitely many points) derivable function $\xi(\cdot)$ such that $\xi(0) = \mathbf{x}_0$ and for all $t \in (0, T)$:*

1. *if $\xi(t) \in \boldsymbol{int}(P)$ then $\dot{\xi}(t)$ is defined and $\dot{\xi}(t) \in \phi(P)$;*
2. *if $\xi(t) \in e$ and $e \in In(P)$ then $\dot{\xi}^+(t)$ is defined and $\dot{\xi}^+(t) = \phi(P)$, where $\dot{\xi}^+(t) = \frac{d^+\xi}{dt}$ is the right derivative of ξ.*

If $T = \infty$, a trajectory segment is called a trajectory*.* ■

Example 5 Figure 2.5-(b) shows a typical trajectory of the SPDI presented in Example 4 from point $\mathbf{x}_0$ to $\mathbf{x}_f$.

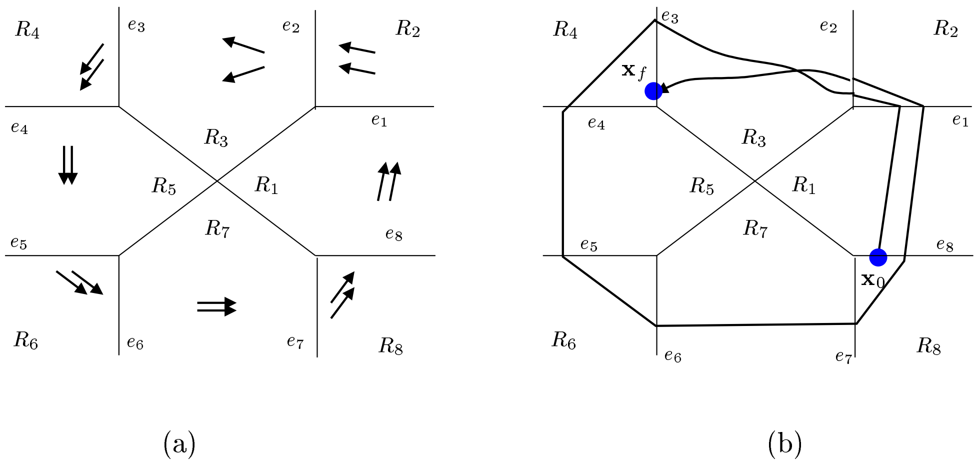


Figure 2.5: (a) The SPDI of the swimmer; (b) A typical trajectory segment.

Edges, vertices, entry edges, exit edges and the corresponding sets are defined as for PCDs. The set of all edges of an SPDI will be denoted by $\mathcal{E}$, i.e.,

$$\mathcal{E} = \bigcup_{s \in S} EV(P_s)$$

In general, $E(P) \neq \mathit{In}(P) \cup \mathit{Out}(P)$. We say that P is a *good* region iff all the edges in $E(P)$ are entries or exits, that is,

Definition 13 *A region P of an SPDI is* good *if and only if*

$$E(P) = \mathit{In}(P) \cup \mathit{Out}(P).$$

■

Notice that, if P is a good region, then for all $e \in E(P)$, $e \notin \phi(P)$. We say that the SPDI has the *goodness* property. Hereinafter, we assume that all regions are good.

Example 6 In Figure 2.6-(a), region P (with $\phi(P) = \angle_{\mathbf{a}}^{\mathbf{b}}$) is good, since all are entry or exit edges. Figure 2.6-(b) shows a region that is not good: edges e_2 and e_5 are not in $\mathit{In}(P) \cup \mathit{Out}(P)$. ■

The reachability problem for an SPDI $\mathcal{H}$ can be defined as a predicate

$$\mathit{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \xi \, \exists t \geq 0 \, . \, (\xi(0) = \mathbf{x}_0 \wedge \xi(t) = \mathbf{x}_f)$$

If such a predicate holds, we say that $\mathbf{x}_f$ is *reachable* from $\mathbf{x}_0$. Let REACH_{SPDI} be the following problem:

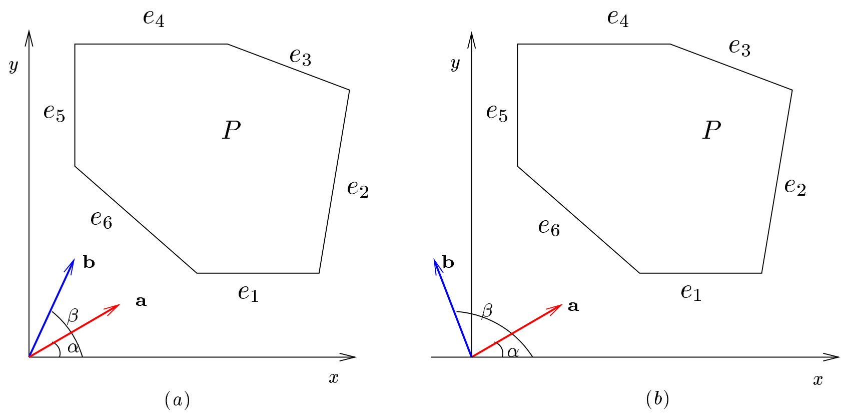


Figure 2.6: a) A good region. b) A bad region.

Problem 1 *Given an SPDI $\mathcal{H}$ and two points $\mathbf{x}_0$ and $\mathbf{x}_f$, is $\mathbf{x}_f$ reachable from $\mathbf{x}_0$?* $\square$

The edge-to-edge reachability problem is the following: Given two edges e and e' of $\mathcal{H}$, is there $\mathbf{x}_0 \in e$ and $\mathbf{x}_f \in e'$ such that $\mathbf{x}_f$ is reachable from $\mathbf{x}_0$? The region-to-region reachability problem is defined straightforwardly.

Remark. Notice that a 2-dim PCD (see section 2.3.1) is an SPDI for which at each region P_s, vectors $\mathbf{a}_s$ and $\mathbf{b}_s$ are equal, that is $\dot{\mathbf{x}} = \mathbf{c}_s$ for a given vector $\mathbf{c}_s$.

2.5 Summary and Related Work

In this chapter we have recalled the definition of hybrid automata, that has been introduced in [5] as well as the definitions of PCDs and PMPs. In [106] it was shown that the reachability problem is decidable for 2 dimensional PCDs and in [140] that the same holds for PMPs. We have presented these two planar hybrid systems since it was precisely in [106] that the "geometric" decision method for reachability was introduced and then it was extended for PMPs [140].

We have also defined SPDIs that are planar nondeterministic systems with piecewise constant differential inclusions and which can be seen as non-deterministic PCDs. Differential inclusions [27] were introduced in [108] and [144] as a tool for modeling uncertainties. They can also be used to model systems with control and disturbances.

Chapter 3

Affine Multivalued Operators

In this chapter we introduce a class of functions called *truncated affine multi-valued functions* (TAMFs) and we study some of its properties. This chapter is rather technical and its main contribution is to serve as a theoretical basis for the notion of *successors* defined in chapter 5. For those interested just in having the basic elements for understanding next chapters, a quick reading of the first section will be enough. Whenever more detailed results of this section be needed in the following chapters, references to those will be given.

Organization of the chapter: We divide the chapter into three sections. In the first one the TAMF class of functions is introduced and some important properties are stated without proving them. In the second section we prove the results presented in the first section as well as many others while in the last section we conclude with a summary and related work.

3.1 Affine Operators (definitions and main results)

In this section we introduce the class of functions we are going to work with and we state some important properties. Let us introduce some useful notions.

Definition 14 *An* affine function $f : \mathbb{R} \to \mathbb{R}$ *is defined by a formula* $f(x) = ax + b$ *with* $a > 0$. ■

Affine functions can be extended to *multi-valued* functions.

Definition 15 *An* affine multi-valued operator (AMF) $F : \mathbb{R} \to 2^{\mathbb{R}}$ *is determined by two affine functions* $f_l(x)$ *and* $f_u(x)$ *that maps* x *to the interval* $\langle f_l(x), f_u(x)\rangle$, *where* $\langle a, b\rangle$ *means* (a, b), $[a, b]$, $(a, b]$ *or* $[a, b)$ *:*

$$F(x) = \langle f_l(x), f_u(x)\rangle$$

with $\mathsf{Dom}(F) = \{x \mid f_l(x) \leq f_u(x)\}$. ■

We use the notation $F = \langle f_l, f_u \rangle$. Such an operator can be naturally extended to subsets of $\mathbb{R}$:

$$F(S) = \bigcup_{x \in S} F(x)$$

In particular, if $S = \langle l, u \rangle$ is an interval, then:

$$F(\langle l, u \rangle) = \langle f_l(l), f_u(u) \rangle$$

where the domain of F is given by $\mathsf{Dom}(F) = \{\langle l, u \rangle \mid f_l(l) \leq f_u(u)\}$ (we consider just well-formed intervals $\langle l, u \rangle$, i.e. with $l \leq u$).

We are interested in considering some kind of restricted affine functions with respect to some intervals. We introduce then the following class of functions.

Definition 16 *A* truncated affine multi-valued operator (TAMF) $\mathcal{F} : \mathbb{R} \to 2^{\mathbb{R}}$ *is determined by an affine multi-valued operator F and intervals $S \subseteq \mathbb{R}^+$ and $J \subseteq \mathbb{R}^+$ as follows:*

$$\mathcal{F}_{F,S,J}(x) = \begin{cases} F(x) \cap J & \textit{if } x \in S \\ \emptyset & \textit{otherwise} \end{cases}$$

■

A TAMF can also be expressed as $\mathcal{F}_{F,S,J}(x) = F(\{x\} \cap S) \cap J$. We use cal style to denote TAMFs operators and in general we will write $\mathcal{F}$ instead of $\mathcal{F}_{F,S,J}$.

Definition 17 *The* intersection $\langle_1 a_1, b_1 \rangle_1 \cap \langle_2 a_2, b_2 \rangle_2 = \langle_3 a_3, b_3 \rangle_3$ *is defined as follows:*

$$\begin{array}{l}
a_3 = \max\{a_1, a_2\} \\
b_3 = \min\{b_1, b_2\} \\
\langle_3 = \begin{cases} \langle_1 & \textit{if } a_1 > a_2 \\ \langle_2 & \textit{if } a_1 < a_2 \\ \max\{\langle_1, \langle_2\} & \textit{if } a_1 = a_2 \end{cases} \\
\rangle_3 = \begin{cases} \rangle_1 & \textit{if } b_1 < b_2 \\ \rangle_2 & \textit{if } b_1 > b_2 \\ \min\{\rangle_1, \rangle_2\} & \textit{if } b_1 = b_2 \end{cases}
\end{array}$$

where $'[' < '('$ *and* $')' < ']'$. ■

Truncated affine multi-valued functions can be also extended to sets and in particular to intervals, as shown in what follows.

$$\begin{array}{lll}
\mathcal{F}(I) & = \bigcup_{x \in I} \mathcal{F}(x) & \text{(by definition)} \\
& = \bigcup_{x \in I} F(\{x\} \cap S) \cap J & \text{(by definition of } \mathcal{F}) \\
& = F(\cup_{x \in I} \{x\} \cap S) \cap J & \\
& = F(I \cap S) \cap J &
\end{array}$$

We define the inverse of an AMF:

Definition 18 *The* inverse *of F is defined by $F^{-1}(x) = \{y \mid x \in F(y)\}$.* ■

It is not difficult to show that $F^{-1} = \langle f_u^{-1}, f_l^{-1} \rangle$ and the inverse of a TAMF $\mathcal{F}$ is given by the following Lemma:

Lemma 1 *Given a $\mathcal{F}(I) = F(I \cap S) \cap J$, then $\mathcal{F}^{-1}(I) = F^{-1}(I \cap J) \cap S$.* □

Definition 19 *A TAMF $\mathcal{F}$ is* normalized *if $S = \mathsf{Dom}(\mathcal{F}) = \{x \mid F(x) \cap J \neq \emptyset\}$ and $J = \mathsf{Im}(\mathcal{F})$.* ■

Notice that we have that for normalized TAMFs, $S \subseteq F^{-1}(J)$ and $J = \mathcal{F}(S)$. In fact, any TAMF can be normalized as stated in the following lemma.

Lemma 2 *Every TAMF $\mathcal{F}$ can be represented in normal form.* □

In what follows, we consider just TAMFs in normal form. The following result shows an important property of affine operators, that is the closure under composition.

Lemma 4 (composition of affine operations)
Affine functions, affine multi-valued operators, and truncated affine multi-valued operators are closed under composition. □

Example 7 Let $x \in J_0$ (where $J_0 = [0, 1]$), and

$$F_1(x) = (2x - \frac{3}{5}, 3x + 5]$$

and

$$F_2(x) = [5x + 2, 7x + 6]$$

be two (non-truncated) affine multi-valued functions, $\mathcal{F}_1 = F_1 \cap J_1$ (with $J_1 = (1, 6]$), and $\mathcal{F}_2 = F_2 \cap J_2$ (with $J_2 = [6, 10)$) their truncated versions. We have that

$$F_1^{-1}(\langle y_1, y_2 \rangle) = \langle \frac{y_1 - 5}{3}, \frac{5y_2 + 3}{10} \rangle$$

and

$$F_2^{-1}(\langle y_1, y_2 \rangle) = \langle \frac{y_1 - 6}{7}, \frac{y_2 - 2}{5} \rangle$$

To obtain $\mathcal{F}_2 \circ \mathcal{F}_1(x)$ we need to compute F', S' and J' as in Lemma 4 but first we compute S_1 and S_2:

$$\begin{aligned} S_1 &= F_1^{-1}(J_1) \cap J_0 \\ &= F_1^{-1}((1,6]) \cap [0,1] \\ &= (-\frac{4}{3}, \frac{33}{10}) \cap [0,1] \\ &= [0,1] \end{aligned}$$

$$\begin{aligned} S_2 &= F_2^{-1}(J_2) \cap J_1 \\ &= F_2^{-1}([6,10)) \cap (1,6] \\ &= [0, \frac{8}{5}) \cap (1,6] \\ &= (1, \frac{8}{5}) \end{aligned}$$

$$\begin{aligned} S' &= S_1 \cap F_1^{-1}(J_1 \cap S_2) \\ &= [0,1] \cap F_1^{-1}((1,6] \cap (1, \frac{8}{5})) \\ &= [0,1] \cap F_1^{-1}((1, \frac{8}{5})) \\ &= [0,1] \cap (-\frac{4}{3}, \frac{11}{10}) \\ &= [0,1] \end{aligned}$$

$$\begin{aligned} J' &= J_2 \cap F_2(J_1 \cap S_2) \\ &= [6,10) \cap F_2((1,6] \cap (1, \frac{8}{5})) \\ &= [6,10) \cap F_2((1, \frac{8}{5})) \\ &= [6,10) \cap (7,10) \\ &= (7,10) \end{aligned}$$

$$\begin{aligned} F'(x) &= F_2 \circ F_1(x) \\ &= (5(2x - \frac{3}{5}) + 2,\ 7(3x+5) + 6] \\ &= (10x - 1,\ 21x + 41] \end{aligned}$$

Hence, the truncated affine multi-valued operator $\mathcal{F}_2 \circ \mathcal{F}_1(x)$ is

$$\mathcal{F}_2 \circ \mathcal{F}_1(x) = \begin{cases} (10x - 1,\ 21x + 41] \cap (7, 10) & \text{if } x \in [0, 1] \\ \emptyset & \text{otherwise} \end{cases}$$

Notice that the result can be extended to any interval $\langle l, u \rangle$. ■

We use the notation $\hat{\mathcal{F}}$ for truncated affine multi-valued operators with $S = J$; i.e. the image and the domain coincide and then $\hat{\mathcal{F}}(I) = F(I \cap H) \cap H$. The following TAMF property will have a key role in the acceleration of cycles when computing successors for the reachability algorithm in section 5.2.

Lemma 11 (Fundamental lemma) Let $\hat{\mathcal{F}}$ be a truncated affine multi-valued operator. Then $\hat{\mathcal{F}}^n(I) = F^n(I \cap H) \ \cap \ H$. □

Intuitively, what the above lemma says is that in order to obtain the iterated truncated affine multi-valued function truncated with an interval H (both the argument and the final result), we only need to iterate the non-truncated function intersecting the argument just once at the beginning and once at the end.

We presented in this section the main definitions and properties to be used in chapter 5.

3.2 Affine Operators (properties)

We will prove in this section the lemmas introduced in the previous section as well as other interesting properties of iterations of affine operations.

3.2.1 Truncated affine multi-valued operators

To start with, we prove that to obtain the inverse of a truncated affine multi-valued function $\mathcal{F}$ we need just to inverse the corresponding non-truncated affine function and truncate it with the domain and co-domain of $\mathcal{F}$ interchanged.

Lemma 1 *Given a* $\mathcal{F}(I) = F(I \cap S) \cap J$, *then* $\mathcal{F}^{-1}(I) = F^{-1}(I \cap J) \cap S$.

Proof: We prove first that $\mathcal{F}^{-1}(I) \subseteq F^{-1}(I \cap J) \cap S$. Let $y \in \mathcal{F}^{-1}(I)$, then it exists $x \in I$ such that $x \in \mathcal{F}(y)$. We have to prove that $y \in S$ and $y \in F^{-1}(\{x\} \cap J))$. Since $x \in \mathcal{F}(y)$, then $x \in F(\{y\} \cap S)$ and $x \in J$, that implies $y \in F^{-1}(x)$ and since $x \in J$ we have that $y \in F^{-1}(\{x\} \cap J)$. On the other hand, from $x \in F(\{y\} \cap S)$ we deduce that $y \in S$. Hence, $y \in F^{-1}(\{x\} \cap J) \cap S$. Given $y \in S$ and $y \in F^{-1}(I \cap J)$ we prove now that $y \in \mathcal{F}^{-1}(x)$. From $y \in F^{-1}(I \cap J)$ we have that $y \in S$ and $y \in F^{-1}(I \cap J)$. Thus, it exists $x \in I \cap J$ s.t.

$y \in y \in F^{-1}(x)$ that implies $x \in F(y)$. From the above results we have then that $x \in J$, $x \in F(y)$ and $y \in S$ and hence $y \in \mathcal{F}^{-1}(I)$. $\Box$

We prove now that every TAMF can be normalized.

Lemma 2 *Every TAMF $\mathcal{F}$ can be represented in normal form.*

Proof: Let $\mathcal{F}(I) = F(I \cap S) \cap J$ be a TAMF. We show that there exists a TAMF $\mathcal{F}'(I) = F'(I \cap S') \cap J'$ such that $\mathcal{F} = \mathcal{F}'$ and $\mathcal{F}'$ is in normal form. Let $\mathcal{F}'$ be the above function with $F' = F$, $S' = S \cap F^{-1}(J)$ and $J' = \mathcal{F}(S)$. Clearly $S' = \mathsf{Dom}(\mathcal{F}')$ and $J' = \mathsf{Im}(\mathcal{F}')$. It remains to show that $\mathcal{F} = \mathcal{F}'$. If $I \cap S = \emptyset$ then the result follows, since $\mathcal{F} = \emptyset$ and $\mathcal{F}' = \emptyset$. Suppose that $I \cap S \neq \emptyset$, we separate the proof into the following cases.

1. $(I \cap S) \cap F^{-1}(J) = \emptyset$: Clearly, $\mathcal{F}' = \emptyset$. On the other hand, we have that for any $x \in I \cap S$ $(I \cap S \neq \emptyset)$, $x \notin F^{-1}(J)$. Thus, $F(x) \cap J = \emptyset$ and $\mathcal{F} = \emptyset$. Hence, $\mathcal{F} = \mathcal{F}'$.

2. $(I \cap S) \cap F^{-1}(J) \neq \emptyset$: Remember that $F' = F$, $S' = S \cap F^{-1}(J)$ and $J' = F(S) \cap J$, we have then:

$$\begin{aligned}
\mathcal{F}'(I) &= F'(I \cap S') \cap J' && \\
&= F(I \cap S \cap F^{-1}(J)) \cap F(S) \cap J && \text{(by def. of } \mathcal{F}'\text{)} \\
&= F(I) \cap F(S) \cap F(F^{-1}(J)) \cap F(S) \cap J && \text{(by Lemma 3)} \\
&= F(I) \cap F(S) \cap F(F^{-1}(J)) \cap J && \text{(by property of } \cap\text{)} \\
&= F(I \cap S) \cap F(F^{-1}(J)) \cap J && \text{(by Lemma 3)} \\
&= F(I \cap S) \cap J && \text{(because } J \subseteq F(F^{-1}(J))\text{)} \\
&= \mathcal{F}(I) &&
\end{aligned}$$

From all the cases above we have then that $\mathcal{F}$ can be represented in normal form. $\Box$

Before showing that the class of functions above defined are closed under composition we prove the following lemma.

Lemma 3 *Let $F = \langle f_l, f_u \rangle$ be a multi-valued affine operator. If $I \cap H \neq \emptyset$ or $I = \emptyset$ or $H = \emptyset$ then $F(I \cap H) = F(I) \cap F(H)$.*

Proof: Let $I = \langle l_I, u_I \rangle$ and $H = \langle l_H, u_H \rangle$. Clearly if $I = \emptyset$ or $H = \emptyset$ then $F(I \cap H) = \emptyset = F(I) \cap F(H)$.
Suppose now that $I \cap H \neq \emptyset$, then

$$\begin{aligned}
F(I \cap H) &= F(\langle \max\{l_I, l_H\}, \min\{u_I, u_H\}\rangle) && \text{(by def. of } \cap\text{)} \\
&= \langle f_l(\max\{l_I, l_H\}), f_u(\min\{u_I, u_H\})\rangle && \text{(by def. of } F\text{)} \\
&= \langle \max\{f_l(l_I), f_l(l_H)\}, \min\{f_u(u_I), f_u(u_H)\}\rangle && \text{(by monotonicity of } F\text{)} \\
&= \langle f_l(l_I), f_u(u_I)\rangle \cap \langle f_l(l_H), f_u(u_H)\rangle && \text{(by def. of intersection)} \\
&= F(\langle l_I, u_I\rangle) \cap F(\langle l_H, u_H\rangle) && \text{(by def. of } F\text{)} \\
&= F(I) \cap F(H) &&
\end{aligned}$$

□

Now we can prove the closure of composition for the three classes of functions introduced before.

Lemma 4 (composition of affine operations) *Affine functions, affine multi-valued operators, and truncated affine multi-valued operators are closed under composition.*

Proof:

Affine functions: For $f(x) = ax + b$ and $g(x) = cx + d$ the composition $g \circ f(x) = c(ax + b) + d = (ca)x + (cb + d)$ has the required form. Notice, that the coefficient ca is positive since c and a are positive.

Affine multi-valued operators For $F = \langle f_l, f_u \rangle$ and $H = \langle h_l, h_u \rangle$, the composition $H \circ F$ is nothing other than $\langle h_l \circ f_l, h_u \circ f_u \rangle$.

Truncated affine multi-valued operators For

$$\mathcal{F}_1(x) = F_1(\{x\} \cap S_1) \cap J_1$$

and

$$\mathcal{F}_2(x) = F_2(\{x\} \cap S_2) \cap J_2$$

we have that

$$\mathcal{F}_2 \circ \mathcal{F}_1(x) = \mathcal{F}_{F',S',J'}(x)$$

with $F' = F_2 \circ F_1$, $J' = J_2 \cap F_2(J_1 \cap S_2)$ and $S' = S_1 \cap F_1^{-1}(J_1 \cap S_2)$:

$$\begin{aligned} \mathcal{F}_2 \circ \mathcal{F}_1(x) &= \mathcal{F}_2(F_1(\{x\} \cap S_1) \cap J_1) \\ &= F_2((F_1(\{x\} \cap S_1) \cap J_1) \cap S_2) \cap J_2; \end{aligned} \tag{3.1}$$

we split the proof into two cases

1. $x \in S_1$: in this case $F_1(\{x\} \cap S_1) = F_1(x)$ and then expression (3.1) is equal to $F_2((F_1(x) \cap J_1) \cap S_2) \cap J_2$ that is equal to

$$F_2(F_1(x) \cap (J_1 \cap S_2)) \cap J_2 \tag{3.2}$$

   We consider two cases

   (a) $F_1(x) \cap (J_1 \cap S_2) \neq \emptyset$: In this case the distributivity holds (see Lemma 3) and expression (3.2) is equal to $F_2(F_1(x)) \cap F_2(J_1 \cap S_2) \cap J_2$. But $F_1(x) \cap (J_1 \cap S_2) \neq \emptyset$ is equivalent to $x \in F_1^{-1}(J_1 \cap S_2)$. Hence,

$$\mathcal{F}_2(\mathcal{F}_1(x)) = F_2 \circ F_1(x) \cap F_2(J_1 \cap S_2) \cap J_2$$

   under the condition $x \in S_1 \wedge x \in F_1^{-1}(J_1 \cap S_2)$, i.e. $x \in S_1 \cap F_1^{-1}(J_1 \cap S_2)$.

   (b) $F_1(x) \cap (J_1 \cap S_2) = \emptyset$: here (3.2) is equal to the empty set and then $\mathcal{F}_2 \circ \mathcal{F}_1 = \emptyset$

2. $x \notin S_1$: that is, $\{x\} \cap S_1 = \emptyset$ and then $F_1(\{x\} \cap S_1) = \emptyset$, that implies $\mathcal{F}_2 \circ \mathcal{F}_1 = \emptyset$.

From the cases above we have

$$\mathcal{F}_2 \circ \mathcal{F}_1(x) = \mathcal{F}_{F',S',J'}(x) = \begin{cases} F'(x) \cap J' & \text{if } x \in S' \\ \emptyset & \text{otherwise} \end{cases}$$

with $F' = F_2 \circ F_1$, $J' = J_2 \cap F_2(J_1 \cap S_2)$ and $S' = S_1 \cap F_1^{-1}(J_1 \cap S_2)$. $\square$

We show next that normalization is preserved by composition.

Lemma 5 *If $\mathcal{F}_1$ and $\mathcal{F}_2$ are normalized, then $\mathcal{F}_2 \circ \mathcal{F}_1$ is also normalized.*

Proof: By Lemma 4,

$$\mathcal{F}_2 \circ \mathcal{F}_1(x) = \mathcal{F}_{F',S',J'}(x)$$

with

$$\begin{array}{l} F' = F_2 \circ F_1, \\ J' = J_2 \cap F_2(J_1 \cap S_2), \quad \text{and} \\ S' = S_1 \cap F_1^{-1}(J_1 \cap S_2). \end{array}$$

Thus, we have to prove that

$$S_1 \cap F_1^{-1}(S_2 \cap J_1) \subseteq [F_2 \circ F_1]^{-1}(J_2 \cap F_2(J_1 \cap S_2)).$$

Suppose that $x \in S_1$ and $x \in F_1^{-1}(S_2 \cap J_1)$, i.e.

$$x \in S_1 \text{ and } F_1(x) \cap (S_2 \cap J_1) \neq \emptyset,$$

then there exists

$$y \in F_1(x) \cap (S_2 \cap J_1),$$

and by normalization of $\mathcal{F}_2$ we have that

$$F_2(F_1(x) \cap (S_2 \cap J_1)) \cap J_2 \neq \emptyset.$$

By Lemma 3 we have that

$$F_2(F_1(x)) \cap F_2(S_2 \cap J_1) \cap J_2 \neq \emptyset,$$

with $x \in S_1$. Hence

$$x \in [F_2 \circ F_1]^{-1}(J_2 \cap F_2(J_1 \cap S_2)).$$

$\square$

The following result shows how to compute fixpoints of affine functions [106].

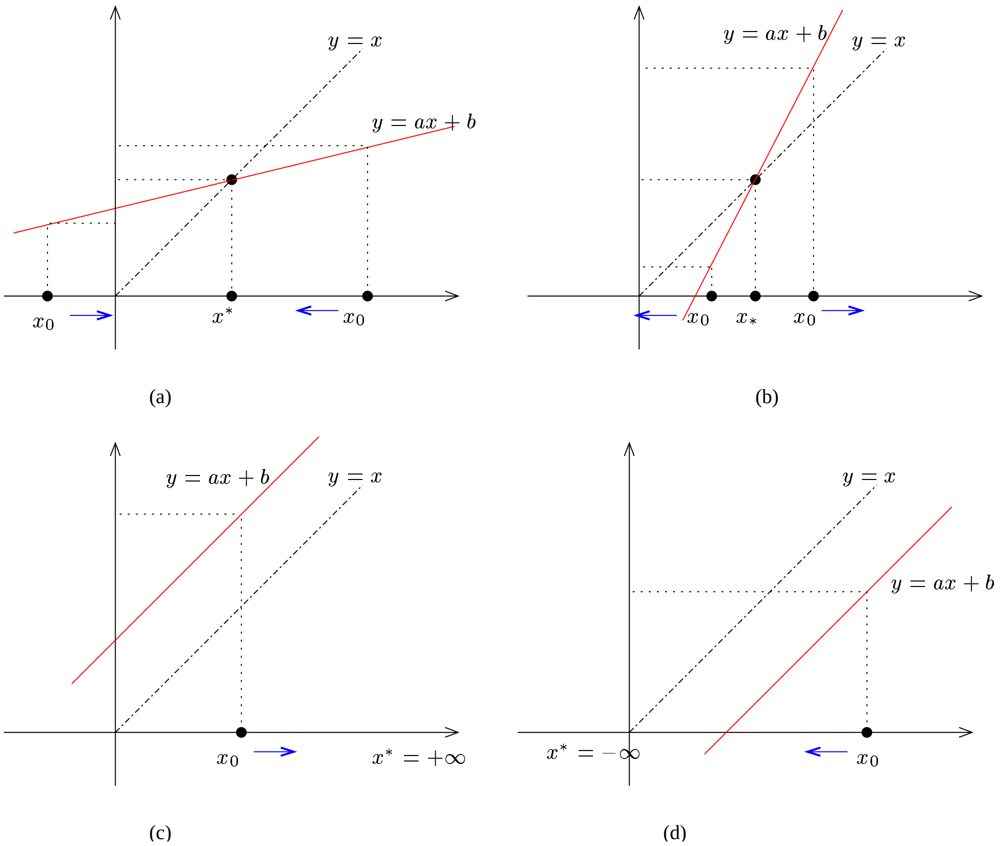


Figure 3.1: (a): $a < 1$, $x^* = b/(1-a)$; (b): $a > 1$, $x^* = -\infty$ if $x_0 < x_*$,$x^* = x_0$ if $x_0 = x_*$, $x^* = +\infty$ if $x_0 > x_*$; (c): $a = 1$ and $b > 0$, $x^* = +\infty$; (d): $a = 1$ and $b < 0$, $x^* = -\infty$

Lemma 6 *Let f be an affine function, x_0 be any initial point and $x_n = f^n(x_0)$. The following properties hold*

1. *The sequence x_n is monotonous;*
2. *It converges to a limit x^* (finite or infinite), which can be effectively computed knowing a, b and x_0.*

Proof: Monotonicity of x_n follows from the identity $x_{n+1} - x_n = a^n(x_1 - x_0)$:

$$\begin{aligned}
x_n = f^n(x_0) &\implies f^n(x_0) = a^n x_0 + a^{n-1}b + \ldots + ab + b \\
&\implies x_{n+1} - x_n = (a^{n+1}x_0 + a^n b + \ldots + ab + b) - (a^n x_0 + a^{n-1}b + \ldots + ab + b) \\
&\implies x_{n+1} - x_n = (a^{n+1}x_0 + a^n b - a^n x_0) \\
&\implies x_{n+1} - x_n = a^n(ax_0 + b - x_0) \\
&\implies x_{n+1} - x_n = a^n(x_1 - x_0)
\end{aligned}$$

Existence of limit is immediate from the monotonicity. To calculate the limit several cases should be considered (see Fig. 3.1):

$a < 1$: In this case the limit is finite and it is the unique fixpoint of the function f: $ax^* + b = x^*$, and hence $x^* = b/(1-a)$.

$a = 1$: In this case

$$x^* = \begin{cases} -\infty & \text{if} \quad b < 0 \\ x_0 & \text{if} \quad b = 0 \\ \infty & \text{if} \quad b > 0 \end{cases}$$

$a > 1$: In this case we should calculate first the (unstable) fixpoint $x_* = b/(1-a)$. However in this case the limit is not necessary equal to x_* . Namely,

$$x^* = \begin{cases} -\infty & \text{if} \quad x_0 < x_* \\ x_0 & \text{if} \quad x_0 = x_* \\ \infty & \text{if} \quad x_0 > x_* \end{cases}$$

□

This result can be easily extended to intervals and affine multi-valued operators.

Lemma 7 *Let $\langle l_0, u_0 \rangle$ be any initial interval and $\langle l_n, u_n \rangle = F^n(\langle l_0, u_0 \rangle)$. The following properties hold*

1. *The sequences l_n and u_n are monotonous;*
2. *They converge to limits l^* and u^* (finite or infinite), which can be effectively computed.*

Proof: Direct consequence of Lemma 6 considering l_n and u_n. □

3.2.2 Some special properties

In this section we prove some other auxiliary results and the main result of this chapter that is the Fundamental Lemma (Lemma 11) and a corollary (Corollary 13) stating that in order to compute the iteration of a TAMF we need to compute the function once at the beginning and once at the end and compose them with the result given by the Fundamental Lemma. We start proving some basic facts about affine functions.

Lemma 8 *Let $f(x) = ax + b$ be an affine function ($a > 0$). Then, the following holds (for $n > 0$):*

1. *If $x \leq y$ then $f^n(x) \leq f^n(y)$;*
2. *If $x \leq f(x)$ then $f(x) \leq f^n(x)$;*

3. *If* $f(y) \leq y$ *then* $f(y)^n \leq f(y)$*;*
4. *If* $f(x) \leq x$ *and* $f(x) \leq f^n(y)$ *then* $x \leq y$*;*
5. *If* $x \leq f(x)$ *and* $x \leq y$ *then* $f(x) \leq f^n(y)$*;*
6. *If* $f(y) \leq y$ *and* $x \leq y$ *then* $f^n(x) \leq f(y)$*;*
7. *If* $y \leq f(y)$ *and* $f^n(x) \leq f(y)$ *then* $x \leq y$*.*

Proof: The result follows directly from simple algebraic properties of partial order and monotonicity. □

The following two lemmas show that AMFs and TAMFs are monotone and continuous.

Lemma 9 *Let* $I = \langle l_I, u_I \rangle$ *and* $H = \langle l_H, u_H \rangle$ *be two intervals and* $F(I) = \langle f_l(l_I), f_u(u_I) \rangle$ *an AMF. Then,*

1. F *is monotone;*
2. F *is continuous.*

Proof:

1. We have to prove that if $I \subseteq H$ then $F(I) \subseteq F(H)$. If $I = \emptyset$ or $H = \emptyset$ the result holds trivially, hence we will consider $I \neq \emptyset$ and $H \neq \emptyset$. Suppose that $I \subseteq H$ (i.e. $l_H \leq l_I \leq u_I \leq u_H$), then

$$\begin{array}{lll} x \in F(I) & \Longrightarrow x \in \langle f_l(l_I), f_u(u_I) \rangle & \text{(by def. of } F) \\ & \Longrightarrow x \in \langle f_l(l_H), f_u(u_H) \rangle & \text{(by hypothesis and monotonicity of } f_l \text{ and } f_u) \\ & \Longrightarrow x \in F(H) & \text{(by def. of } F) \end{array}$$

Thus, $F(I) \subseteq F(H)$, i.e. F is monotone.

2. We have to prove that $F(\cup_i I_i) = \bigcup_i (F(I_i))$. Let $A = \cup_i I_i$.

$\subseteq$: Suppose that $y \in F(A)$, then $y \in \bigcup_{x \in A} F(x)$ (by def. of F applied to sets). Thus, for some $x \in A$, there exist an interval I_j s.t. $x \in I_j$ and then $y \in F(x)$. By definition and monotonicity of F we have that $F(x) \subseteq F(I_j)$, that implies $y \in F(I_j)$ and then $y \in \bigcup_i F(I_i)$. Hence, $F(\cup_i I_i) \subseteq \bigcup_i (F(I_i))$.

$\supseteq$: Suppose that $y \in \bigcup_i (F(I_i))$, then exist an interval I_j s.t. $y \in F(I_j)$ and by monotonicity of F, $y \in F(\cup_i I_i)$. Hence $\bigcup_i (F(I_i)) \subseteq F(\cup_i I_i)$.

From both results we have that $F(\cup_i I_i) = \bigcup_i (F(I_i))$, i.e. F is continuous.

□

We have the same result for TAMFs.

Lemma 10 *Let I and H be two intervals and $\mathcal{F}(I) = F(I \cap S) \cap J$ a TAMF. Then,*

1. *$\mathcal{F}$ is monotone;*
2. *$\mathcal{F}$ is continuous.*

Proof:

1. We have to prove that if $I \subseteq H$ then $\mathcal{F}(I) \subseteq \mathcal{F}(H)$. If $I = \emptyset$ or $H = \emptyset$ the result holds trivially. Suppose that I and H are non-empty and that $I \subseteq H$. Then,

$$\begin{aligned} x \in \mathcal{F}(I) &\Longrightarrow x \in F(I \cap S) \cap J && \text{(by def. of } \mathcal{F}) \\ &\Longrightarrow x \in F(I \cap S) \,\wedge\, x \in J && \\ &\Longrightarrow x \in F(H \cap S) \,\wedge\, x \in J && \text{(by hypothesis and monotonicity of } F) \\ &\Longrightarrow x \in F(H \cap S) \cap J && \text{(by def. of } \mathcal{F}) \\ &\Longrightarrow x \in \mathcal{F}(H) && \end{aligned}$$

   Hence, $\mathcal{F}(I) \subseteq \mathcal{F}(H)$ and $\mathcal{F}$ is monotone.

2. We prove that $\mathcal{F}(\cup_i I_i) = \bigcup_i (\mathcal{F}(I_i))$:

$$\begin{aligned} \mathcal{F}(\cup_i I_i) &= F(\cup_i I_i \cap S) \cap J && \text{(by def. of } \mathcal{F}) \\ &= F(\cup_i (I_i \cap S)) \cap J && \text{(by distrib.)} \\ &= \textstyle\bigcup_i F(I_i \cap S) \cap J && \text{(by monotonicity of } F) \\ &= \textstyle\bigcup_i \mathcal{F}(I_i) && \text{(by definition of } \mathcal{F}) \end{aligned}$$

   We conclude that $\mathcal{F}$ is continuous.

□

In what follows we will use the following notation: given $I = \langle l, u \rangle$, $x < I$ is equivalent to $x < l$ if I is a (left) closed interval or to $x \leq l$ if I is an (left) open interval; dually for $x > I$ (w.r.t. u). Given two intervals $\langle l_1, u_1 \rangle$ and $\langle l_2, u_2 \rangle$, $l_1 < l_2$ it means $l_1 \leq l_2$ if $\langle l_2, u_2 \rangle$ is a (left) closed interval and $\langle l_2, u_2 \rangle$ is an (left) open interval and it means $l_1 < l_2$ otherwise. Similarly for the right extremes of the intervals.

Remember that $\hat{\mathcal{F}}(I) = F(I \cap H) \cap H$. Let $H = \langle l_H, u_H \rangle$ and $I \cap H = \langle l_{I \cap H}, u_{I \cap H} \rangle$. For this kind of functions we have the following result.

Lemma 11 (Fundamental lemma) *Let $\hat{\mathcal{F}}$ be a truncated affine multi-valued operator. Then $\hat{\mathcal{F}}^n(I) = F^n(I \cap H) \,\cap\, H$.*

Proof:

Base case ($n = 1$): By definition $\hat{\mathcal{F}}(I) = F(I \cap H) \cap H$.

Inductive step ($n \geq 2$): We have to prove that $\hat{\mathcal{F}}^{n+1} = F^{n+1}(I \cap H) \cap H$.

$$\begin{array}{rcll} \hat{\mathcal{F}}^{n+1}(I) & = & \hat{\mathcal{F}}(\hat{\mathcal{F}}^n(I)) & \\ & = & \hat{\mathcal{F}}(F^n(I \cap H) \cap H) & \text{(By inductive hypothesis)} \\ & = & F(F^n(I \cap H) \cap H) \cap H & \text{(By definition of } \hat{\mathcal{F}}\text{)} \end{array} \quad (3.3)$$

We consider two cases:

1. $\mathbf{F^n(I \cap H) \subseteq H}$: In this case (3.3) is equal to $F(F^n(I \cap H)) \cap H$ that is the same as $F^{n+1}(I \cap H) \cap H$.
2. $\mathbf{F^n(I \cap H) \not\subseteq H}$: We split the proof into two sub-cases
   (a) $\mathbf{F^n(I \cap H) \cap H = \emptyset}$: By inductive hypothesis, $\hat{\mathcal{F}}^n(I) = \emptyset$ and then $\hat{\mathcal{F}}^{n+1}(I) = \emptyset$. On the other hand, that $F^{n+1}(I \cap H) \cap H = \emptyset$ follows from the supposition by monotonicity.
   (b) $\mathbf{F^n(I \cap H) \cap H \neq \emptyset}$: In this case the distributivity holds (by Lemma 3) and expression (3.3) is equal to $F^{n+1}(I \cap H) \cap F(H) \cap H$, that can be written as
   $$[\max\{F^{n+1}(l_{I \cap H}), F(l_H), l_H\},\ \min\{F^{n+1}(u_{I \cap H}), F(u_H), u_H\}]$$
   We analyze the following cases depending on the relationship between $F(H)$ and H as shown in Figure 3.2
      i. $F(l_H) < l_H \wedge F(u_H) < u_H$: The first condition ($F(l_H) < l_H$) implies $\max\{F^{n+1}(l_{I \cap H}), F(l_H), l_H\} = \max\{F^{n+1}(l_{I \cap H}), l_H\}$. On the other hand, $F(u_H) < u_H$ implies $F^{n+1}(u_{I \cap H}) < F(u_H)$ (from $u_{I \cap H} < u_H$ and Lemma 6) and then $\min\{F^{n+1}(u_{I \cap H}), F(u_H), u_H\} = \min\{F^{n+1}(u_{I \cap H}), u_H\}$.
      ii. $F(l_H) < l_H \wedge u_H < F(u_H)$: The analysis of the first condition is as before, whereas the second one implies that $\min\{F^{n+1}(u_{I \cap H}), F(u_H), u_H\} = \min\{F^{n+1}(u_{I \cap H}), u_H\}$.
      iii. $l_H < F(l_H) \wedge u_H < F(u_H)$: By Lemma 5 and from $l_H < l_{I \cap H}$ we have that $l_H < F(l_H)$ implies $F(l_H) < F^{n+1}(l_{I \cap H})$. The second condition is analyzed as the above case.
      iv. $l_H < F(l_H) \wedge F(u_H) < u_H$: Both cases were already treated before.

   From all the cases above, we have that $F^{n+1}(I \cap H) \cap F(H) \cap H = [\max\{F^{n+1}(l_{I \cap H}), l_H\},\ \min\{F^{n+1}(u_{I \cap H}), u_H\}] = F^{n+1}(I \cap H) \cap H$.

Thus, from cases 1 and 2 we finally obtain that $\hat{\mathcal{F}}^{n+1}(I) = F^{n+1}(I \cap H) \cap H$. □

Let $\mathcal{F}$ be a TAMF and $\hat{\mathcal{F}}$ be equal to $F(I \cap H) \cap H$ with $H = S \cap J$, then the following result holds.

Lemma 12 $\hat{\mathcal{F}}^n(I) = \mathcal{F}^n(I \cap J) \cap S$.

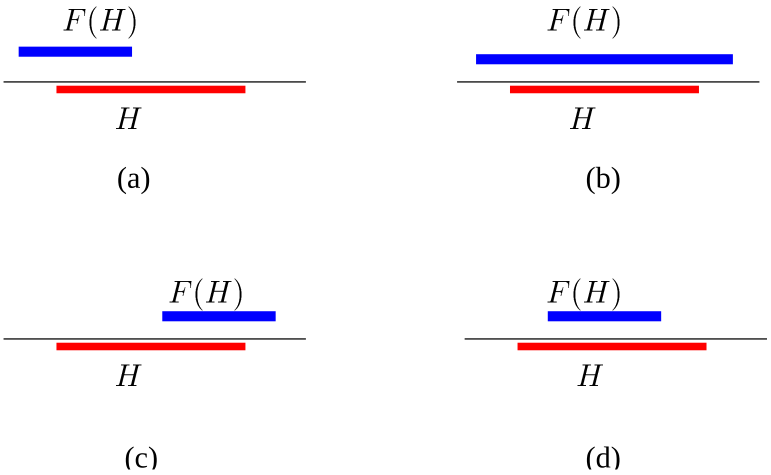


Figure 3.2: Relationship between $F(H)$ and H (case 2b of Lemma 11).

Proof: The proof is done by induction over n.

Base case ($n = 1$):

$$\begin{aligned}
\hat{\mathcal{F}}(I) &= F(I \cap J \cap S) \cap J \cap S && \text{(by def. of } \hat{\mathcal{F}}\text{)}\\
&= (F((I \cap J) \cap S) \cap J) \cap S && \text{(by associativity)}\\
&= \mathcal{F}(I \cap J) \cap S && \text{(by def. of } \mathcal{F}\text{)}
\end{aligned}$$

Inductive Step ($n > 1$):

$$\begin{aligned}
\hat{\mathcal{F}}^{n+1}(I) &= \hat{\mathcal{F}}^{n}(\hat{\mathcal{F}}(I)) && \\
&= F^{n}(\hat{\mathcal{F}}(I) \cap J \cap S) \cap J \cap S && \text{(by Lemma 11)}\\
&= F^{n}((F(I \cap J \cap S) \cap J \cap S) \cap J \cap S) \cap J \cap S && \text{(by def. of } \hat{\mathcal{F}}\text{)}\\
&= F^{n}(F(I \cap J \cap S) \cap J \cap S) \cap J \cap S && \\
&= \hat{\mathcal{F}}^{n}(F(I \cap J \cap S)) && \text{(by Lemma 11)}\\
&= \mathcal{F}^{n}(F(I \cap J \cap S) \cap J) \cap S && \text{(by H.I.)}\\
&= \mathcal{F}^{n}(\mathcal{F}(I \cap J)) \cap S && \text{(by def. of } \mathcal{F}\text{)}\\
&= \mathcal{F}^{n+1}(I \cap J) \cap S &&
\end{aligned}$$

$\square$

As a consequence we have the following corollary that shows that in order to compute the iteration of a TAMF we need to compute the function once at the beginning and once at the end and compose them with the result given by the Fundamental Lemma.

Corollary 13 $\mathcal{F}^{n+2} = \mathcal{F} \circ \hat{\mathcal{F}}^{n} \circ \mathcal{F}$.

Proof:

$$\begin{aligned}
\mathcal{F}^{n+2}(I) &= \mathcal{F}(\mathcal{F}^n(\mathcal{F}(I))) \\
&= \mathcal{F}(\mathcal{F}^n(\mathcal{F}(I) \cap J)) \\
&= \mathcal{F}(\mathcal{F}^n(\mathcal{F}(I) \cap J) \cap S) \\
&= \mathcal{F}(\hat{\mathcal{F}}^n(\mathcal{F}(I))) && \text{(by Lemma 12)} \\
&= \mathcal{F} \circ \hat{\mathcal{F}}^n \circ \mathcal{F}(I).
\end{aligned}$$

□

3.3 Summary and Related Work

We have presented in this chapter a class of functions called *truncated affine multi-valued functions* (TAMFs) and we have studied some of its properties. TAMFs operates on sets in general and in particular on intervals. This class is rather important since it is the base for the definition of the *successor operators* on chapter 5 (see section 5.1). There we will show that successors are TAMFs that operates on intervals (edges of some polygons). In particular, the Fundamental Lemma (Lemma 11) guarantees that when considering iterations we only need to intersect the argument of the function with one special interval just once at the beginning and once at the end and not in each iteration.

As far as we know, there is not formal definition of TAMFs in the literature. There are many interval-based temporal logics where intervals have an important role: Duration Calculus [77], Interval Temporal Logic [114] and the Interval-based Temporal Logic introduced in [3]. In the last work mentioned, a formal axiomatization of “interval-based” time is given. The TAMF class can be formalized in an algebraic way using a simpler axiomatization of intervals than the one defined in [3].

Chapter 4

SPDI - Qualitative Analysis of Trajectory Segments

In this chapter we present the difficulties that arise when trying to solve the reachability problem for polygonal differential inclusion systems (SPDIs) and we show how to overcome these difficulties doing a qualitative analysis of trajectory segments. As a first step, we show that between any two points linked by an arbitrary trajectory segment there always exists a trajectory segment without self-crossings. Even considering trajectory segments of this type, there are infinitely many of them to treat. We find a good abstraction of such trajectory segments considering some kind of "symbolic" trajectories and we show that there exists just finitely many of them and that the abstraction preserves reachability.

Organization of the chapter: In the first section we describe the difficulties of the reachability problem of SPDIs. In the second section we present our first step in order to overcome the difficulties shown in the first section, that is the simplification of trajectory segments. The second step of our approach, that is the abstraction of trajectory segments into *signatures* is presented in section three. In the same section we present the fourth step that is the *representation* of signatures in a convenient way. Section four is dedicated to the abstraction of signatures into *type of signatures*. In section five we study some properties of types of signatures and we present the main results of this chapter namely that there exists finitely many types of signatures and that our abstraction procedure preserves reachability. Finally, we conclude with a summary.

4.1 Reachability Difficulties

In this section we present some of the main difficulties that arise when trying to answer to the reachability question stated in Problem 1. Indeed, there is only one main difficulty which is that, given two points, there are infinitely many trajectory segments from one point to the other, and this is mainly due to the following three reasons:

1. Even locally (restricted to one region) there are infinitely many trajectory segments, due to the differential inclusion;

2. Self-crossings introduce also the possibility of having infinite number of trajectory segments;

3. Suppose that we consider a kind of "*symbolic*" trajectory segments, taking into account just the sequence of traversed edges (we call them *signatures* and will be defined later on section 4.2.1). Let us consider a simple cycle of traversed edges that is repeated k times. Even in this case we obtain an infinite number of trajectory segments since each k gives a different trajectory segment (and k can be eventually infinite).

The following example shows all the three cases above.

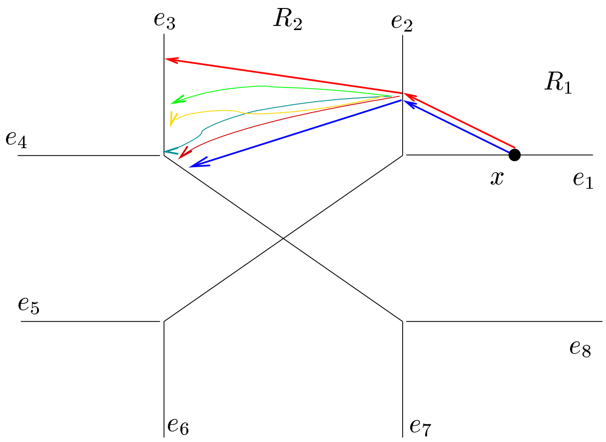


Figure 4.1: Locally infinitely many trajectory segments.

Example 8 Let us consider again the swimmer of Example 4.

1. In Figure 4.1 is shown that in region R_2 there are infinitely many trajectory segments from point x;

2. The second difficulty is due to self-crossing trajectory segments as shown in Figure 4.2;

3. Let us consider the sequence of traversed edges $e_1, e_2, \cdots, e_8, e_1$ as shown in Figure 4.3 and let k be the number of times this sequence is repeated. It is not difficult to see that for $k = 2$ we have $e_1, e_2, \cdots, e_8, e_1, e_2, \cdots, e_8, e_1$, for $k = 3$, e_1, e_2, $\cdots$, e_8, e_1, e_2, $\cdots$, e_8, e_1, e_2, $\cdots$, e_8, e_1 and so on, i.e., there is a potentially unbounded number of iterations. Moreover, we have other trajectory segments with different sequence of traversed edges as the one shown in the figure with signature $e_1, e_2, e_9, e_{10}, e_{11}, e_{12}$, $e_9, \cdots, e_{12}, \cdots$. ■

Our solution to overcome this problem consists in performing the following four steps:

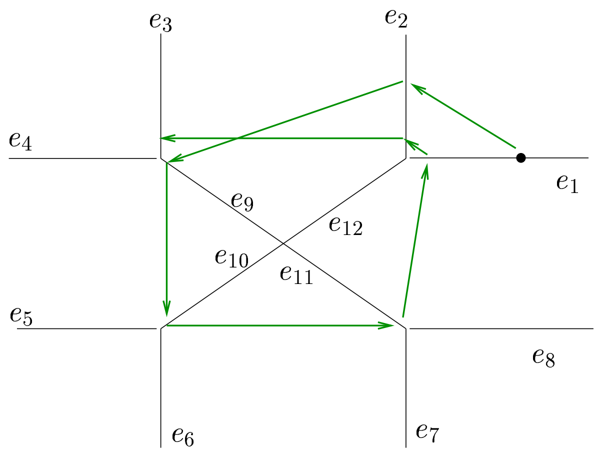


Figure 4.2: A self-crossing trajectory.

1. Simplification of trajectory segments: straightening them and removing self-crossings. Given an arbitrary trajectory segment from one point to another, we show how to get a piecewise constant derivative trajectory segment without self-crossing.

2. Abstraction of trajectory segments into *signatures*, considering the sequence of traversed edges.

3. *Factorization* of signatures in a convenient way, having only sequences of edges and simple cycles.

4. Abstraction of factorized signatures into *types of signatures*, that are signatures without taking into account the number of times each simple cycle is iterated.

In next section we will show how to simplify trajectory segments.

4.2 Simplification of Trajectory Segments

As explained in the last section, here we are going to perform the first step of our solution that consists in two parts: straightening trajectory segments and removing self-crossings. In other words, we will perform an abstraction, from trajectory segments to piecewise constant derivative trajectory segments without self-crossings.

4.2.1 Straightening of trajectory segments

We show here how to transform trajectory segments into rectilinear ones *straightening* them. We also define the notion of *signature*. W.l.o.g. we consider in what follows that $\xi(0) \in e$ for some edge $e \in \mathcal{E}$. We have the following definition.

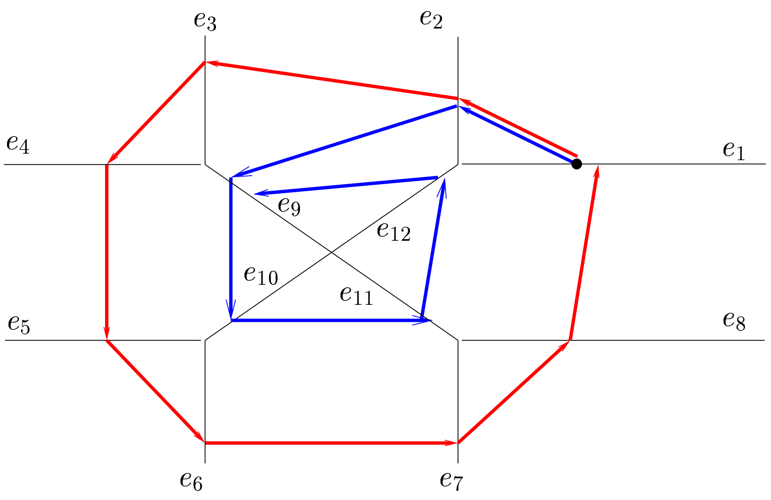


Figure 4.3: Infinitely many signatures.

Definition 20 *The* trace *of a trajectory ξ is the sequence* $\mathtt{trace}(\xi) = \mathbf{x}_0\mathbf{x}_1\ldots$ *of the intersection points of ξ with the set of edges, that is,* $\mathbf{x}_i \in \xi \cap \mathcal{E}$. ■

Example 9 Let us consider the trajectory segment ξ from point $\mathbf{x}_0$ to point $\mathbf{x}_7$ shown in Figure 4.4-(a). The trace of such a trajectory segment is the sequence of points $\mathtt{trace}(\xi) = \mathbf{x}_0\mathbf{x}_1\ldots\mathbf{x}_6\mathbf{x}_7$. ■

Given a trajectory (segment) on an SPDI, we can consider the sequence of the edges traversed by it.

Definition 21 *An* edge signature *(or simply a* signature*) of an SPDI is a sequence of edges. The* edge signature of a trajectory *ξ, $\mathsf{Sig}(\xi)$, is the ordered sequence of traversed edges by the trajectory segment, that is,* $\mathsf{Sig}(\xi) = e_0e_1\ldots$, *with* $\mathtt{trace}(\xi) = \mathbf{x}_0\mathbf{x}_1\ldots$ *and* $\mathbf{x}_i \in e_i$. *The* region signature*of ξ is the sequence* $\mathsf{RSig}(\xi) = P_0P_1\ldots$ *of traversed regions, that is,* $e_i \in \mathit{In}(P_i)$. ■

Definition 22 *A signature* $\mathsf{Sig}(\xi) = e_0e_1\ldots e_n$ *is a* cyclic signature *(or a* signature cycle*) iff* $e_0 = e_n$ *and* $\mathsf{Sig}(\xi)$ *is a* simple edge-cycle *(or a* simple cycle*) if additionally for all* $0 < i \neq j < n$, $e_i \neq e_j$. *A region signature* $\mathsf{RSig}(\xi) = P_0P_1\ldots P_n$ *is a* cyclic-region *(or a* region cycle*) iff* $P_0 = P_n$ *and it is a* simple region-cycle *if in addition for all* $0 < i \neq j < n$, $P_i \neq P_j$. ■

We will say that $\mathsf{Sig}(\xi) = \sigma$ if σ is the edge signature of the trajectory ξ.

Example 10 Let us consider again the trajectory segment ξ from point $\mathbf{x}_0$ to point $\mathbf{x}_7$ of

Figure 4.4-(a). Its edge signature is the sequence of edges $\mathsf{Sig}(\xi) = e_1e_2e_9e_{10}e_{11}e_1e_2e_3$ and its region signature is $\mathsf{RSig}(\xi) = R_1R_2R_4R_6R_8R_1R_2$. ■

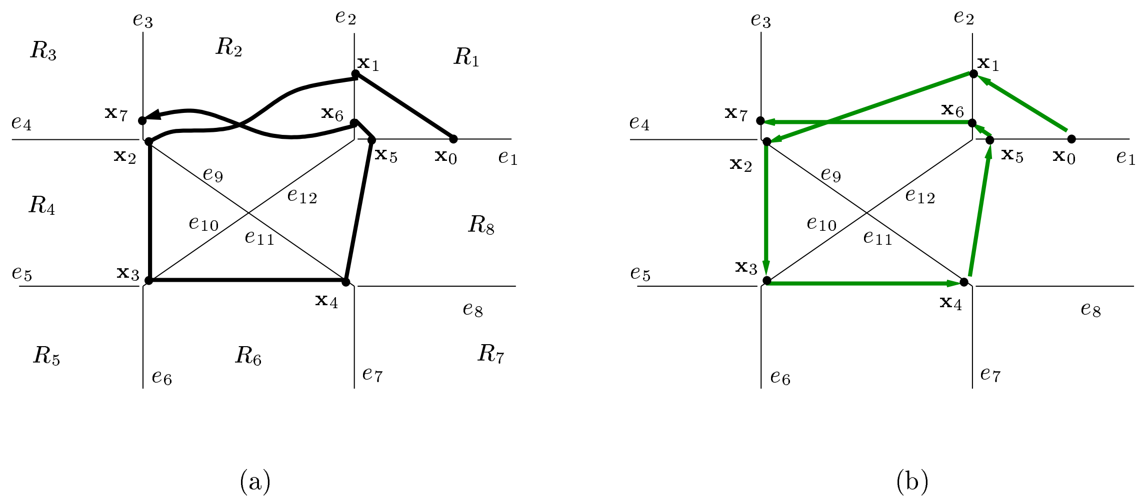


Figure 4.4: (a) A trajectory segment with its trace, (b) The straightened trajectory segment

The following result expresses that any segment of trajectory in a given region can be straightened, preserving its initial and final points (see Fig. 4.5).

Proposition 1 *For every trajectory segment ξ there exists a trajectory segment ξ' with the same initial and final points, and edge and region signatures, such that for each P_i in the region signature, there exists $\mathbf{c}_i \in \phi(P_i)$, such that $\dot{\xi}'(t) = \mathbf{c}_i$ for all $t \in (t_i, t_{i+1})$. Moreover,* $\mathtt{trace}(\xi) = \mathtt{trace}(\xi')$

Proof: Let ξ be a trajectory segment whose trace is $\mathtt{trace}(\xi) = \mathbf{x}_0 \ldots \mathbf{x}_k$. Let $0 = t_0 < t_1 < \ldots < t_k$ be such that $\xi(t_i) = \mathbf{x}_i$. Since ξ is continuous and derivable in the interval (t_i, t_{i+1}), there exists a unique trajectory segment ξ' with $\xi'(t_i) = \xi(t_i)$ for all $i \in [0, k-1]$, such that the derivative $\dot{\xi}'$ is constant in the interval (t_i, t_{i+1}). That is, for each P_i in the region signature, there exists $\mathbf{c}_i \in \phi_i$, such that $\dot{\xi}'(t) = \mathbf{c}_i$ for all $t \in (t_i, t_{i+1})$. □

Example 11 In Figure 4.4-(b) it is shown the straightened trajectory segment of the one given in Figure 4.4-(a). ■

Hence, in order to solve the reachability problem it is enough to consider trajectory segments having piecewise constant slopes. Notice that, however, such slopes need not be the same for each occurrence of the same region in the region signature. Hereinafter, we only consider trajectory segments whose derivatives are piecewise constants.

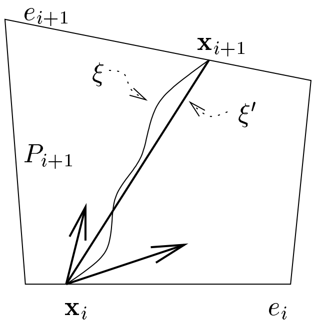


Figure 4.5: Piecewise constant trajectory.

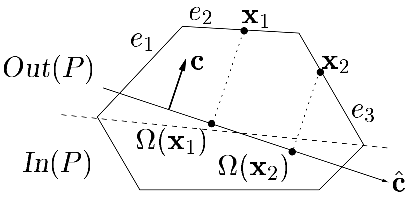


Figure 4.6: Ordering: $\mathbf{x}_1 \preceq \mathbf{x}_2$.

4.2.2 Removing self-crossings

Before proceeding to the removing of self-crossing trajectory segments we need to introduce an order relation which will be intensively used in the sequel.

Consider a region P and let $\mathbf{c} \in \phi(P)$. The mapping $\Omega : \mathbb{R}^2 \to \mathbb{R}$, defined as $\Omega(\mathbf{x}) = \mathbf{x}\ \hat{\mathbf{c}}$, assigns to every $\mathbf{x} \in \mathbb{R}^2$ a value proportional to the length of the projection of the vector $\mathbf{x}$ on the right rotation of $\mathbf{c}$. Indeed, the ordering is given by the direction of $\hat{\mathbf{c}}$ and one can easily see that the relation $\preceq$, defined as $\mathbf{x}_1 \preceq \mathbf{x}_2$ if $\Omega(\mathbf{x}_1) \leq \Omega(\mathbf{x}_2)$, is a dense linear order on $In(P)$ and $Out(P)$ (Fig. 4.6). We use $\prec$ to denote the strict variant of $\preceq$ and say that $e_1 \prec e_2$ iff $e_1 \neq e_2$ and $\mathbf{x}_1 \preceq \mathbf{x}_2$ for every $\mathbf{x}_1 \in e_1, \mathbf{x}_2 \in e_2$. For example, in Fig. 4.6 we have $e_1 \prec e_2 \prec e_3 \prec e_4$. Notice that the order does not depend on the choice of $\mathbf{c} \in \phi(P)$.

We say that a trajectory ξ crosses itself if there exist $t \neq t'$ such that $\xi(t) = \xi(t')$. If a trajectory does not cross itself, the sequence of consecutive intersection points with $In(P)$ or $Out(P)$ is monotone with respect to $\preceq$. That is, for every three points $\mathbf{x}_1$, $\mathbf{x}_2$ and $\mathbf{x}_3$ (visited in this order), if $\mathbf{x}_1 \prec \mathbf{x}_2 \prec \mathbf{x}_3$ the trajectory is a "counterclockwise expanding spiral"(Fig. 4.7(a)) or a "clockwise contracting spiral" (Fig. 4.7(b)) and if $\mathbf{x}_3 \prec \mathbf{x}_2 \prec \mathbf{x}_1$, the trajectory is a "counterclockwise contracting spiral" (Fig. 4.7(c)) or a "clockwise expanding spiral" (Fig. 4.7(d)).

Lemma 14 *For every trajectory ξ, if ξ does not cross itself, then for every edge e, the sequence* $\texttt{trace}(\xi) \cap e$ *is monotone (with respect to $\preceq$).*

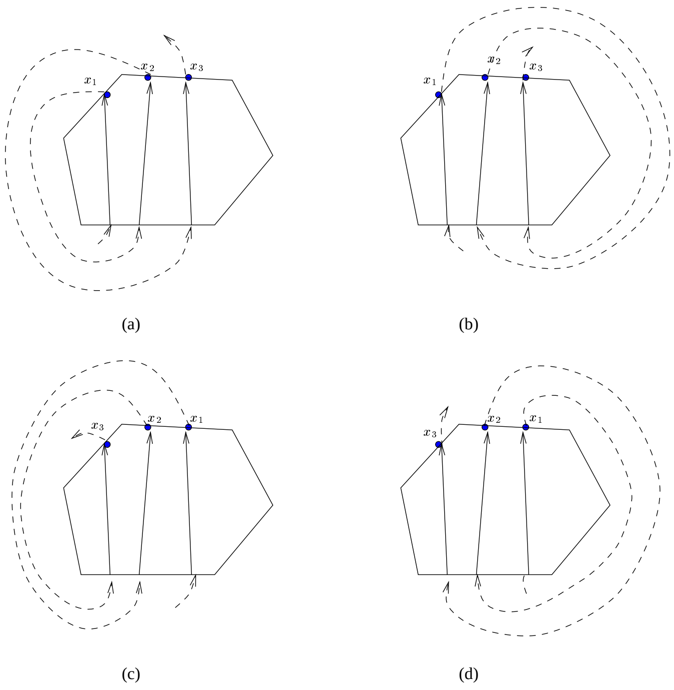


Figure 4.7: (a) $\mathbf{x}_1 \prec \mathbf{x}_2 \prec \mathbf{x}_3$: counterclockwise expanding spiral; (b) $\mathbf{x}_1 \prec \mathbf{x}_2 \prec \mathbf{x}_3$: clockwise contracting spiral; (c) $\mathbf{x}_3 \prec \mathbf{x}_2 \prec \mathbf{x}_1$: counterclockwise contracting spiral; (d) $\mathbf{x}_3 \prec \mathbf{x}_2 \prec \mathbf{x}_1$: clockwise expanding spiral.

Proof: The proof follows directly form Jordan's curve theorem [78]. $\square$

We prove now that self-crossings can be removed from trajectory segments, preserving the reachability problem, but first we prove that we can always diminish the number of self-crossings.

Lemma 15 *For every trajectory segment ξ that crosses itself at least once, there exists a trajectory segment ξ' with the same initial and final points of ξ having a number of self-crossings strictly smaller.*

Proof: Suppose that the trajectory segment ξ with trace $\mathtt{trace}(\xi) = \mathbf{x}_0 \ldots \mathbf{x}_f$ crosses itself *once* inside the region P. Let $e_1, e_2 \in \mathit{In}(P)$ be the input edges and $e'_1, e'_2 \in \mathit{Out}(P)$ be the output ones. Let $\mathbf{x} = \mathbf{x}_i \in e_1$ and $\mathbf{y} = \mathbf{x}_j \in e_2$, with $i < j$, be the points in $\mathtt{trace}(\xi)$ the first and the second times ξ enters P, and let $\mathbf{x}' = \mathbf{x}_{i+1} \in e'_2$ and $\mathbf{y}' = \mathbf{x}_{j+1} \in e'_1$ be the corresponding output points. Let $\mathbf{c}_x, \mathbf{c}_y \in \phi(P) = \angle_{\mathbf{a}}^{\mathbf{b}}$ be the derivatives of ξ in the time intervals (t_i, t_{i+1}) and (t_j, t_{j+1}), respectively. Indeed, $\mathbf{c}_x$ and $\mathbf{c}_y$ are the director vectors of the segments $\overline{\mathbf{x}\mathbf{x}'}$ and $\overline{\mathbf{y}\mathbf{y}'}$, respectively (Fig. 4.8(a)). Consider now the segment $\overline{\mathbf{x}\mathbf{y}'}$. Notice that the director vector $\mathbf{c}'_x$ of this segment can be obtained as a

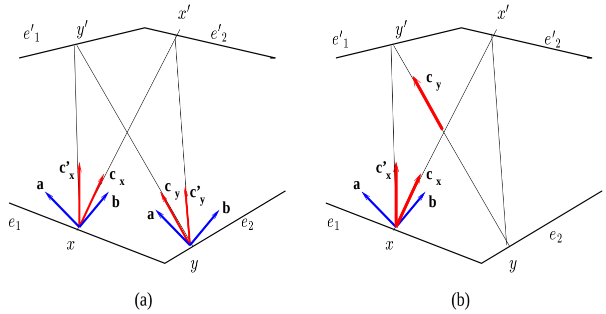


Figure 4.8: A trajectory that crosses itself.

positive combination of the vectors $\mathbf{c}_x$ and $\mathbf{c}_y$. That is, there exist $\alpha_1, \alpha_2 > 0$ such that $\mathbf{c}'_x = \alpha_1 \mathbf{c}_x + \alpha_2 \mathbf{c}_y$ (see Fig. 4.8(b)). Since $\mathbf{c}_x, \mathbf{c}_y \in \phi(P) = \angle_{\mathbf{a}}^{\mathbf{b}}$, there exist $\gamma_1, \gamma_2, \delta_1, \delta_2 > 0$ such that $\mathbf{c}'_x = \alpha_1(\gamma_1 \mathbf{a} + \gamma_2 \mathbf{b}) + \alpha_2(\delta_1 \mathbf{a} + \delta_2 \mathbf{b}) = (\alpha_1\gamma_1 + \alpha_2\delta_1)\mathbf{a} + (\alpha_1\gamma_2 + \alpha_2\delta_2)\mathbf{b}$. Thus, $\mathbf{c}'_x \in \phi(P)$. Similarly we can prove that $\mathbf{c}'_y$ is a positive combination of **a** and **b**. Hence, there exists a trajectory ξ' that does not cross itself in P having a trace $\mathtt{trace}(\xi') = \mathbf{x}_1 \ldots \mathbf{x}\mathbf{y}' \ldots \mathbf{x}_f$ (Fig. 4.9). Notice that the result also works for the *degenerate* case when the trajectory segment crosses itself at an edge (or vertex) (see Fig. 4.10-(a)). If the trajectory segment ξ crosses itself more than once in region P, then the number of times the trajectory segment ξ', obtained by cutting away the loop (Fig. 4.9(c)), crosses itself in P is strictly smaller than the number of times ξ does it (see Fig. 4.11). After replacing $\overline{\mathbf{xx}'}$ and $\overline{\mathbf{yy}'}$ by $\overline{\mathbf{xy}'}$, the intersection q of $\overline{\mathbf{xx}'}$ and $\overline{\mathbf{yy}'}$ disappears. If the new segment of line $\overline{\mathbf{xy}'}$ crosses another segment $\overline{\mathbf{zz}'}$ (say at a point t), then $\overline{\mathbf{zz}'}$ necessarily crosses either $\overline{\mathbf{xx}'}$ (at r) or $\overline{\mathbf{yy}'}$ (at s) -or both-, before the transformation. The above is due to the fact that if $\overline{\mathbf{zz}'}$ crosses one side of the triangle $\mathbf{xy}'q$ then it must also cross one of the other sides of the triangle, say at r. Thus, no new crossing can appear and the number of crossings in the new configuration is always less than in the old one. $\square$

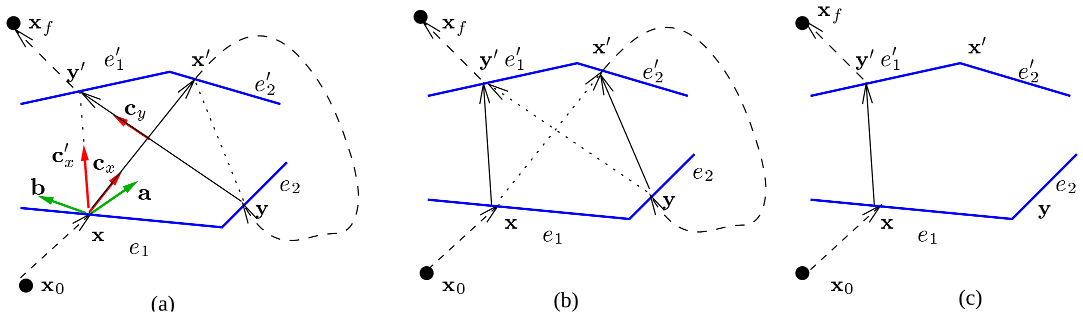


Figure 4.9: Obtaining a non-crossing trajectory

Remark. In the above proof notice that we consider that the trajectory segment may cross itself just in a finite number of points. In fact there is a degenerate case shown in Figure 4.10-(b), where there are infinitely many crossing points. The above result continue to holds, it suffices to consider induction over crossing points and intervals.

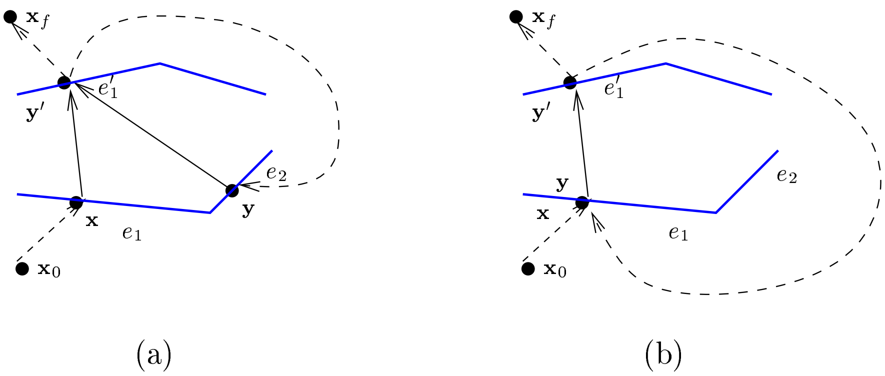


Figure 4.10: "Degenerate" self crossings.

We have then the following proposition.

Proposition 2 (Existence of a non-crossing trajectory) *If there exists an arbitrary trajectory segment from points* $\mathbf{x}_0 \in e_0$ *to* $\mathbf{x}_f \in e_f$ *then there always exists a non-crossing trajectory segment between them.*

Proof: By induction on the number n of times the trajectory segment crosses itself using Lemma 15 in the induction step. □

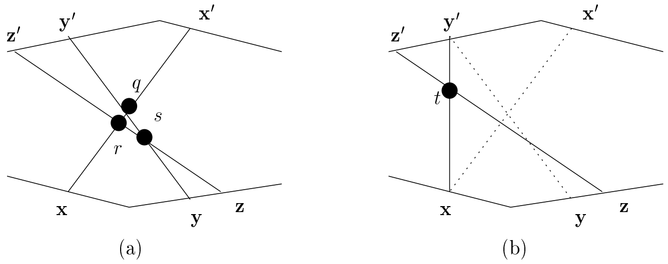


Figure 4.11: The number of crossings decrease after eliminating self-crossings. (a) Before (3 crossings); (b) After (1 crossing).

Example 12 Given the trajectory segment of Figure 4.4-(b), after eliminating the self-crossing we obtain the trajectory segment of Figure 4.12. ■

Hence, in order to solve the reachability problem we only need to consider non-crossing trajectory segments with piecewise constant derivatives. In what follows, we only deal with trajectory segments of this kind.

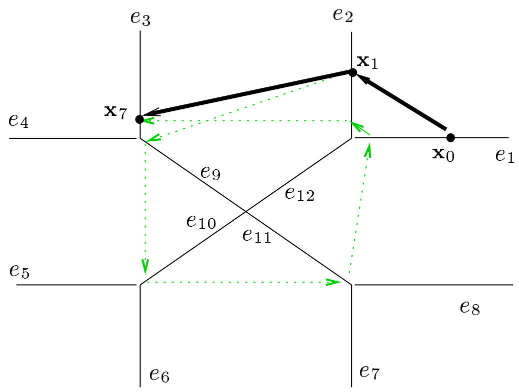


Figure 4.12: A trajectory segment without self-crossing.

4.3 From Simplified Trajectory Segments to *Factorized Signatures*

In this section we present the second and third steps of our solution. We show now how to abstract trajectory segments into *signatures*. Given a trajectory segment of an SPDI considering its trace over the sequence of traversed edges (signature) gives a good abstraction that preserves reachability. We have then the following fact.

Fact 16 *If there exists a trajectory segment from points $\mathbf{x}_0 \in e_0$ to $\mathbf{x}_f \in e_f$, then there exists a sequence of points $\mathbf{x}_0, \ldots, \mathbf{x}_i, \ldots, \mathbf{x}_f$ with edge signature $e_0, \ldots, e_i, \ldots, e_f$.* $\square$

In what follows we present a *representation theorem* that allows to express signatures in a *factorized* way.

Given a sequence w, ε denotes the empty sequence whereas $first(w)$ and $last(w)$ are the first and last elements of the sequence respectively. An edge signature σ can be expressed as a sequence of edges and cycles of the form $r_1 s_1^{k_1} r_2 s_2^{k_2} \ldots r_n s_n^{k_n} r_{n+1}$, where

1. For all $1 \leq i \leq n+1$, r_i is a sequence of pairwise different edges;

2. For all $1 \leq i \leq n$, s_i is a simple cycle (i.e., without repetition of edges) repeated k_i times;

This representation can be obtained as follows.

Algorithm $\mathcal{A}$Let $\sigma = e_1 \dots e_{p-1} e_p$ be an edge signature. Starting from e_{p-1} and traversing backwards, take the first edge that occurs the second time. If there is no such edge, then trivially the signature can be expressed as a sequence of different edges. Otherwise, suppose that the edge e_j occurs again at position i (i.e. $e_i = e_j$ with $i < j$), thus $\sigma_{\mathcal{A}} = wsr$, where w, s and r are obtained as follows, depending on the repeated edge:

$$\begin{aligned} w &= e_0 \dots e_i \\ s &= e_{i+1} \dots e_j \\ r &= e_{j+1} \dots e_{p-1} \end{aligned}$$

Clearly r is not a cycle and s is a simple cycle with no repeated edges. We continue the analysis with w. Let $k_m = \max\{l \mid s^l \text{ is a suffix of } w\}$. Thus, $\sigma_{\mathcal{A}} = w' s^k r$ with $w' = e_0 \dots e_h$ (a prefix of w) and $k = k_m + 1$. We repeat recursively the procedure above with w'. Adding the edge e_p to the last r (at the end) we obtain $\sigma_{\mathcal{A}} = r_1 s_1^{k_1} \dots r_n s_n^{k_n} r_{n+1}$ that is a representation of signature σ. □

Notice that the "preprocessing" (taking away the last edge e_p) is done in order to differentiate edge signatures that end with a cycle from those that do not. There exists many other (maybe easier) ways of decomposing a signature σ (in particular, traversing forward instead of backwards), but the one chosen here permits a clearer and simpler presentation of the reachability algorithm. In fact, using the above representation, the last visited edge in a cycle $e_1 \dots e_k$ is always the last one (e_k). The representation obtained by the above algorithm give rises to the following theorem.

Theorem 17 (Representation Theorem) *Let $\sigma = e_1 \dots e_p$ be an edge signature, then it can always be written as $\sigma_{\mathcal{A}} = r_1 s_1^{k_1} \dots r_n s_n^{k_n} r_{n+1}$, where for any $1 \leq i \leq n+1$, r_i is a sequence of pairwise different edges and for all $1 \leq i \leq n$, s_i is a simple cycle (i.e., without repetition of edges).* □

Each edge signature can then be represented as a sequence of edges and simple cycles. In what follows "cyclic signatures" will mean "simple cycles" unless the contrary be said.

Example 13 Let us consider the following examples. Suppose that

$$\sigma = abcdbcefgefgefgefhi.$$

Then, after applying once the above procedure of the algorithm we obtain that

$$\sigma_{\mathcal{A}} = w(s_2)^3 r_1,$$

with $w = abcdbcef$; $s_2 = gef$; $r_1 = h$. Applying the procedure once more to w we obtain

$$w = w'(s_3)^1 r_2$$

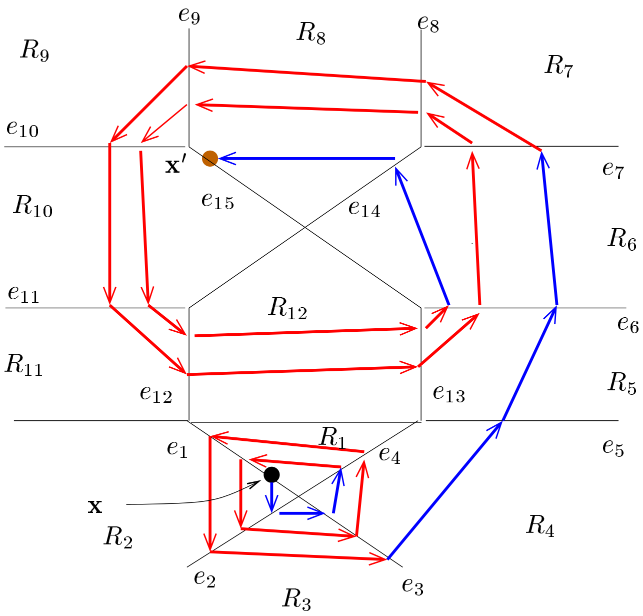


Figure 4.13: A trajectory segment from $\mathbf{x}$ to $\mathbf{x}'$.

with $w' = r_3 = abc$; $s_3 = dbc$; $r_2 = ef$. Putting all together and adding the last edge (i) gives

$$\sigma_{\mathcal{A}} = abc(dbc)^1 ef(gef)^3 hi$$

Suppose now, that the signature ends with a cycle:

$$\sigma = abcdbcefgefgefgefgef.$$

In this case we apply the preprocessing obtaining

$$\sigma_{\mathcal{A}} = w(s_2)^4 r_1$$

with $w = abcdbce$; $s_2 = fge$; $r_1 = \varepsilon$. Applying the procedure to w we finally obtain

$$w = w'(s_3)^1 r_2$$

with $w' = r_3 = abc$; $s_3 = dbc$; $r_2 = e$ and that gives (adding f to the end)

$$\sigma_{\mathcal{A}} = abc(dbc)^1 e(fge)^4 f.$$

■

Example 14 Let us consider an SPDI and the trajectory segment shown in Figure 4.13 from a point $\mathbf{x} \in e_1$ to a point $\mathbf{x}' \in e_{15}$, where for simplicity we do not show the formal definition of the SPDI. The edge signature of the trajectory segment is $\sigma = e_1e_2e_3 \dots e_6e_7 \dots e_{13}e_6e_{14}e_{15}$. Applying algorithm $\mathcal{A}$ above we obtain the following representation:

$$\sigma_{\mathcal{A}} = e_1e_2e_3(e_4e_1e_2e_3)^2e_5e_6(e_7 \cdots e_{13}e_6)^2e_{14}e_{15}.$$

■

Notice that even though the number of trajectory segments from one point to another has been considerably reduced after the simplification of section 4.2, there are still an infinite number of them. Even when considering signatures, the number is still infinite. Our representation theorem simplifies the analysis but does not decrease the number of signatures to be considered. The problem is that in principle all the simple cycles can be iterated an unbounded number of times. Hence, the following natural step is to abstract away the number of times each simple cycle is iterated, that is the subject of next section.

4.4 From Factorized Signatures to *Types of Signatures*

In this section we show how to abstract the signatures obtained in the previous section via the representation theorem to *type of signatures*.

Given a representation of a signature, obtained as before, we have the following definition.

Definition 23 *Let $\sigma = e_1 \dots e_p$ be an edge signature and $\sigma_{\mathcal{A}} = r_1s_1^{k_1} \dots r_ns_n^{k_n}r_{n+1}$ be its representation (obtained by Algorithm $\mathcal{A}$). Then we define the* type of a signature *σ as* $\texttt{type}(\sigma) = r_1, s_1, \dots, r_n, s_n, r_{n+1}$. ■

Moreover, from Theorem 17 we conclude that we can obtain an abstraction of a signature as shown in the following corollary.

Corollary 18 *For each edge signature $\sigma = e_1 \dots e_i \dots e_p$ there exists a type of signature of the form* $\texttt{type}(\sigma) = r_1, s_1, \dots, r_n, s_n, r_{n+1}$.

Proof: The result follows from Theorem 17 constructing $\sigma_{\mathcal{A}} = r_1s_1^{k_1} \dots r_ns_n^{k_n}r_{n+1}$ and then abstracting away the number of times each cycle is iterated. □

Notice that the above corollary really gives an abstraction in the sense of *abstract interpretation* [58, 59] but we consider that our abstraction is quite straightforward and defining the equivalence classes, a quotient set, the abstraction function, etc, would obscure the presentation.

When referring to the type of a signature, we will always mean the type being generated as in Theorem 17 (i.e., by Algorithm $\mathcal{A}$).

Definition 24 *The set of all the types of signatures of an SPDI will be denoted by $\mathcal{T}$, that is,*

$$\mathcal{T} = \{\tau \mid \exists \sigma \in \mathcal{E}^* \ . \ \mathtt{type}(\sigma) = \tau\}$$

where $\mathcal{E}$ is the set of all the edges of the SPDI. ■

The set of types of signatures from one edge e_0 to other edge e_f will be denoted by $\mathcal{T}(e_0, e_f)$.

Example 15 The type of the signature $\sigma_{\mathcal{A}} = abc(dbc)^1 ef(gef)^3 hi$ of Example 13 is

$$\mathtt{type}(\sigma) = abc, (dbc), ef, (gef), hi.$$

And the type of $\sigma_{\mathcal{A}} = abc(dbc)^1 e(fge)^4 f$ is

$$\mathtt{type}(\sigma_{\mathcal{A}}) = abc, (dbc), e, (fge), f.$$

■

Example 16 The type of the signature of Example 14 is

$$\mathtt{type}(\sigma) = e_1e_2e_3, (e_4e_1e_2e_3), e_5e_6, (e_7 \cdots e_{13}e_6), e_{14}e_{15}.$$

■

We have defined signatures as being a sequence of edges but we are particularly interested on signatures that corresponds to trajectory segments. We have then the following definition.

Definition 25 *We say that a signature σ is* feasible *if and only if there exists a trajectory segment ξ with signature σ, i.e.,* $\mathsf{Sig}(\xi) = \sigma$. ■

Let $\mathcal{T}_{feasi}$ denote the set of all the types of feasible signatures, i.e.,

$$\mathcal{T}_{feasi} = \{\tau \mid \exists \sigma \in \mathcal{E}^* \ . \ \sigma \text{ is feasible and } \mathtt{type}(\sigma) = \tau\}$$

Given a type of signature we want to characterize the set of all the signatures with such type, that is the set of signatures that *concretize* the type. This notion is defined in what follows.

Definition 26 *Given a type of signature $\tau = r_1, s_1, \ldots, s_n, r_{n+1}$, the* concretization *(or* realization*) of τ is the set of all the edges signatures with type τ, i.e.,*

$$\mathtt{Concr}(\tau) = \{r_1 s_1^{k_1} \ldots s_n^{k_n} r_{n+1} \mid k_i \in \mathbb{N}^+, \textit{for } 1 \le i \le n\}$$

■

4.5 Properties of Types of Feasible Signatures

We present now some properties of the types of signatures obtained in the previous section and we show the main results of the chapter, namely that the set of type of signatures is finite and that all feasible signatures belongs to a type.

Let ξ be a trajectory segment with edge signature $\mathsf{Sig}(\xi) = e_0 \dots e_p$, and region signature $\mathsf{RSig}(\xi) = P_0 \dots P_p$. We have the following definition.

Definition 27 *An edge e is said to be* abandoned *by ξ after position i, if $e_i = e$ and for some j,k, $i \leq j < k$, $P_j \dots P_k$ forms a region cycle and $e \notin \{e_{i+1}, \dots, e_k\}$. Since trajectory segments are* finite *we should add the trivial case when $e \neq e_j$ for all $j, j > i$.* ∎

An important result concerning abandonment is the following.

Lemma 19 (Abandonment is Irreversible) *For every trajectory segment ξ and edge e, if e is abandoned by ξ after position i, e will not appear in $\mathsf{Sig}(\xi)$ at any position $j > i$.*

Proof: It follows from Lemma 14. See Claim 2 in [19]. □

Intuitively, the abandonment lemma above guarantees that any edge that occurs in a prefix of an edge signature but does not appear in a cycle following this prefix cannot occur anymore in any postfix (starting with the cycle) of the edge signature.

Edge	*Abandonment position*
e_1	9
e_2	10
e_3	11
e_4	8
e_5	12

Table 4.1: Abandonment of edges of Example 17.

Example 17 Let us consider the trajectory segment from $\mathbf{x}$ to $\mathbf{x}'$ of Figure 4.13, with signature $\sigma_{\mathcal{A}} = e_1 e_2 e_3 (e_4 e_1 e_2 e_3)^2 e_5 e_6 (e_7 \cdots e_{13} e_6)^2 e_{14} e_{15}$. In order to visualize the position, we unfold the above signature and we write the occurrence position of each edge as a supra-index[1]:

$$\sigma_{\mathcal{A}} = e_1^1 e_2^2 e_3^3 (e_4^4 e_1^5 e_2^6 e_3^7)(e_4^8 e_1^9 e_2^{10} e_3^{11}) e_5^{12} e_6^{13} (e_7^{14} \cdots e_{13}^{20} e_6^{21})(e_7^{22} \cdots e_{13}^{28} e_6^{29}) e_{14}^{30} e_{15}^{31}$$

[1]We have kept the parenthesis in order to visualize the cycles.

Notice that $R_6R_7\dots R_{11}R_{12}R_5$ forms a region cycle with positions 13, 14, …, 19 and 20 respectively. Edge e_5, for instance, is abandoned after position 12 since it does not belong to the set of edges $\{e_6, e_7, \dots e_{13}\}$ (that have positions 13, 14, …, 20 respectively). Moreover, e_5 cannot appear in any extension of the above trajectory segment from $\mathbf{x}'$. Moreover, edges e_1 to e_4 are also abandoned at positions shown in Table 4.5. ■

We have that the types of feasible signatures obtained by Corollary 18 have the following properties.

Lemma 20 *Let $\sigma = e_0 \dots e_p$ be a feasible signature, then its type,* $\texttt{type}(\sigma) = r_1, s_1, \dots, r_n$, s_n, r_{n+1} *satisfies the following properties.*

$\mathbf{P_1}$ *For every $1 \le i \ne j \le n+1$, r_i and r_j are disjoint;*

$\mathbf{P_2}$ *For every $1 \le i \ne j \le n$, s_i and s_j are different.*

Proof:

$\mathbf{P_1}$ Let $e \in r_i$; we consider two cases:

1. $e \notin s_i$: The result follows immediately from Lemma 19 (e cannot occur in any r_j, $j > i$);
2. $e \in s_i$: Suppose that $e \in r_{i+1}$. Then we have $s_i = e_1 \cdots e_i \cdots e_k$ and $r_{i+1} = e_{k+1} \cdots e_j \cdots e_l$, with $e_i = e_j$, but this is not possible: the construction of σ was doing backwards, and in this case we should have a cycle $s = e_{i+1} \cdots e_k e_{k+1} \cdots e_j$. If $e \in r_j$ (for any $j > i+1$) then again we have two cases: $e \in s_{j-1}$ or $e \notin s_{j-1}$; the first case is not possible by construction and the latter contradicts Lemma 19.

$\mathbf{P_2}$ Let $s_i = e_1, \dots, e_k$ be a simple cycle. After cycling k_i times the cycle is abandoned by edge e_k (by construction of $\sigma_{\mathcal{A}}$). Let P be a region s.t. $e_k \in \mathit{In}(P)$ and consider the unfolding of the last iteration and its continuation:

$$\dots, e_1, e_2, \dots, e_k, e, \dots$$

where, by feasibility, $e = \mathit{first}(r_{i+1})$, $e_k \in \mathit{In}(P)$ and $e_1, e \in \mathit{Out}(P)$ $(e_1 \ne e)$. By the ordering between edges we have that either $e \prec e_1$ or $e_1 \prec e$. By the monotonicity of the trajectory, in both cases e_1 cannot occur after e in σ. Thus, any other cycle s_j, with $i < j$, differs from s_i at least on e_1. Hence, all the cycles are different. □

Notice that by Lemma 20 any type of signature in $\mathcal{T}_{feasi}$ satisfies properties $\mathbf{P_1}$ and $\mathbf{P_2}$. As a direct consequence of Lemma 20 we have the following proposition.

Proposition 3 *The set of types of feasible signatures $\mathcal{T}_{feasi}$ is finite.* □

Moreover, any feasible signature belongs to a type as shown in the following lemma.

Lemma 21 *For any signature σ, if σ is feasible then there exists a type of signature $\tau \in \mathcal{T}$ such that $\sigma \in \texttt{Concr}(\tau)$.*

Proof: The result follows from Theorem 17 , Corollary 18, Lemma 20 and definition of $\texttt{Concr}$. $\square$

Remember that the point-to-point reachability for SPDIs can be stated as:

$$\mathit{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \xi\, \exists t \geq 0 \,.\, (\xi(0) = \mathbf{x}_0 \wedge \xi(t) = \mathbf{x}_f)$$

and for a given ξ, we have the following predicate:

$$\mathit{Reach}_\xi(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists t \geq 0 \,.\, (\xi(0) = \mathbf{x}_0 \wedge \xi(t) = \mathbf{x}_f)$$

Let us define the reachability following a given signature as:

$$\mathit{Reach}_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \xi \,.\, (\mathsf{Sig}(\xi) = \sigma \wedge \mathit{Reach}_\xi(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f))$$

Finally, the following predicate defines the point-to-point reachability for a given type of signature τ:

$$\mathit{Reach}_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \sigma \in \texttt{Concr}(\tau) \,.\, \mathit{Reach}_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f))$$

We prove now that following the steps presented in this chapter, transforming a given arbitrary trajectory segment into its type of signature, reachability is preserved.

Theorem 22 *Given an SPDI $\mathcal{H}$ and two points $\mathbf{x}_0$ and $\mathbf{x}_f$, then the following holds:*

$$\mathit{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \;\; \textit{iff} \;\; \mathit{Reach}_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$$

for some $\tau \in \mathcal{T}_{feasi}$.

Proof:

1. We prove first that if $\mathit{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ then $\exists \tau \,.\, \mathit{Reach}_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$:

$$\begin{array}{rcll}
\mathit{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \Longleftrightarrow & \exists \xi \,.\, \mathit{Reach}_\xi(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \text{(by def. of } \mathit{Reach}\text{)} \\
& \Longrightarrow & \exists \sigma \,.\, \mathsf{Sig}(\sigma) = \xi \wedge \mathit{Reach}_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \text{(by def. of } \mathit{Reach}_\sigma\text{)} \\
& \Longrightarrow & \sigma \text{ is feasible} & \text{(by def. of feasibility)} \\
& \Longrightarrow & \exists \tau \in \mathcal{T}_{feasi} \,.\, \sigma \in \texttt{Concr}(\tau) & \text{(by Lemma 21)} \\
& \Longrightarrow & \exists \sigma \in \texttt{Concr}(\tau) \,.\, \mathit{Reach}_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \\
& \Longleftrightarrow & \mathit{Reach}_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \text{(by def. of } \mathit{Reach}_\tau\text{)}
\end{array}$$

2. We prove now that if for some τ, $Reach_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ then $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$:

$$\begin{array}{rcll} Reach_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \Longleftrightarrow & \exists \sigma \in \mathtt{Concr}(\tau) \ . \ Reach_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \text{(by def. of } Reach_\tau\text{)} \\ & \Longrightarrow & \exists \xi \ . \ \mathsf{Sig}(\sigma) = \xi \wedge Reach_\xi(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \text{(by def. of } Reach_\sigma\text{)} \\ & \Longrightarrow & \exists \xi \exists t \geq 0 \ . \ \xi(0) = \mathbf{x}_0 \wedge \xi(t) = \mathbf{x}_f & \text{(by def. of } Reach_\xi\text{)} \\ & \Longleftrightarrow & Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) & \end{array}$$

From the two cases above we have that $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ iff $\exists \tau \ . \ Reach_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$. $\Box$

Thus, by Proposition 3, to solve the reachability problem we can proceed by examining one by one the types of signatures that guarantee to preserve reachability by the above theorem.

4.6 Summary

In this chapter we have described the main difficulties that arise when considering the reachability problem for SPDIs as well as a solution to overcome them. We have shown that to any arbitrary trajectory segment between two points corresponds a type of signature. The intermediate results are:

1. For any arbitrary trajectory segment, there exists a piecewise constant derivative trajectory segment without self-crossing (Proposition 2);

2. For any trajectory segment without self-crossing and with piecewise constant derivatives, there exists a signature (Fact 16);

3. For each signature there exists a factorized signature (Theorem 17);

4. For each factorized signature there exists a type of signature (Corollary 18).

We have also proved that the set of types of feasible signatures is finite (Proposition 3) and that any feasible signature belongs to a type. Moreover, and as a main result we have proved that the above procedure, from arbitrary trajectory segments to types of signatures, preserves reachability (Theorem 22).

Chapter 5

SPDI - Reachability Analysis

In the previous chapter we have shown how to make an abstraction of trajectories into types of signatures and we have proved that in order to solve the reachability problem we only need to consider the set of types of signatures. This set is indeed finite. The last step of our method consists in giving a decision procedure for each type of signature. In this chapter we develop an algorithm for solving the reachability problem for SPDIs. Our procedure is based on the computation of the limit of individual trajectories. A key idea is the use of one-dimensional affine Poincaré maps for which we can easily compute the fixpoints.

A preliminary version of the results presented in the previous and in this chapter has been published in [22].

Organization of the chapter: In the first section we introduce our basic "tool": successor functions. In the following section we present our main result, that is a decision procedure for the reachability problem of SPDIs (with the restriction of *goodness*) and we show its soundness and termination. Finally, we conclude with a summary and related work.

5.1 Successor Operators

We define in this section our main technical tool for computing reachability, that is the *successor operators* and for that we need an effective representation of (rational) points and intervals on edges.

Let us introduce a one-dimensional coordinate system on each edge. For each edge e we chose a point on it (the origin) with radius-vector $\mathbf{v}$, and a director vector $\mathbf{e}$ going in the positive direction in the sense of the order $\prec$.

To characterize e we need the coordinates of its extreme points: two more numbers $e^l, e^u \in \mathbb{Q} \cup \{-\infty, \infty\}$ such that $e = \{\mathbf{v} + x\mathbf{e} \mid e^l < x < e^u\}$. Clearly, having fixed $\mathbf{v}$ and $\mathbf{e}$ for every edge we can represent every point $\mathbf{x} \in e$ by a pair (e, x) identifying the edge e and the coordinate x (see Fig.5.1(a)). Every interval $\langle \mathbf{x}_1, \mathbf{x}_2 \rangle$ contained in e is represented as $(e, \langle x_1, x_2 \rangle)$, where $\mathbf{x}_1 = (e, x_1)$ and $\mathbf{x}_2 = (e, x_2)$ (see Fig.5.1(b)). Notice that if e is a vertex,

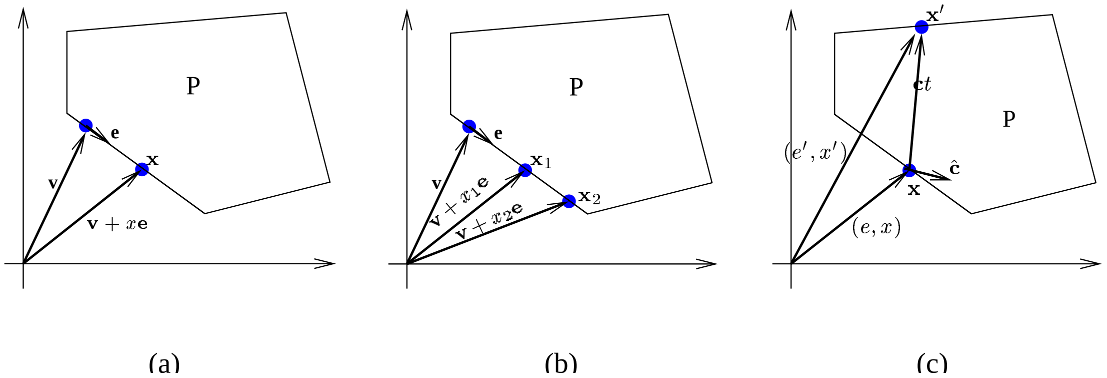


Figure 5.1: (a) Representation of edges; (b) Representation of an interval; (c) One-step successor.

then $e = \{\mathbf{v}\}$, where $\mathbf{v}$ is the only vector that characterizes e. Moreover, all the vertices have local coordinates $x \in [0, 0]$, i.e. a vertex v is represented by a pair $(v, 0)$; hence, whenever e is a vertex, $e = \langle e^l, e^u \rangle$ must be understood as $e = [e^l, e^u]$ whereas if e is a "true" edge then $e = (e^l, e^u)$.

We define the *edge-to-edge successor* $\mathsf{Succ}^{\mathbf{c}}_{ee'}$ following a given vector $\mathbf{c}$.

Definition 28 *Let* $e = \langle e^l, e^u \rangle \in In(P)$ *and* $e' = \langle e'^l, e'^u \rangle \in Out(P)$[1] *be two edges,* $\mathbf{x} = (e, x)$ *and* $\mathbf{x}' = (e', x')$ *be two points and* $\mathbf{c} \in \phi(P)$ *a given vector. The* edge-to-edge successor following a given vector $\mathbf{c}$ *is defined as*

$$\mathsf{Succ}^{\mathbf{c}}_{ee'}(x) = x'$$

such that $\mathbf{x}' = \mathbf{x} + \mathbf{c}t$ *for some* $t > 0$. ■

Notice that $\mathbf{x}'$ is unique. We say that the point (e', x') is the successor of (e, x) in the direction $\mathbf{c}$ (see Fig.5.1(c)). We prove now that successors are TAMFs.

Lemma 23 *The function* $\mathsf{Succ}^{\mathbf{c}}_{ee'}$ *is truncated affine.*

Proof: Expanding $\mathbf{x}' = \mathbf{x} + \mathbf{c}t$, we obtain

$$\mathbf{v}' + x'\mathbf{e}' = \mathbf{v} + x\mathbf{e} + t\mathbf{c}.$$

Multiplying both expressions by $\hat{\mathbf{c}}$ (the right rotation of $\mathbf{c}$) we obtain that

$$(\mathbf{v}' + x'\mathbf{e}')\hat{\mathbf{c}} = \mathbf{v}\hat{\mathbf{c}} + x\mathbf{e}\hat{\mathbf{c}}$$

[1] We will use both e and $\langle e^l, e^u \rangle$ to denote an edge.

i.e. $x'(\mathbf{e}'\hat{\mathbf{c}}) = x(\mathbf{e}\hat{\mathbf{c}}) + (\mathbf{v} - \mathbf{v}')\hat{\mathbf{c}}$. Thus

$$x' = \frac{\mathbf{e}\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}x + \frac{\mathbf{v} - \mathbf{v}'}{\mathbf{e}'\hat{\mathbf{c}}}\hat{\mathbf{c}}$$

and

$$x' \in \langle e'^l, e'^u \rangle.$$

Thus putting $\alpha(\mathbf{c}) = \frac{\mathbf{e}\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}$, $\beta(\mathbf{c}) = \frac{\mathbf{v}-\mathbf{v}'}{\mathbf{e}'\hat{\mathbf{c}}}\hat{\mathbf{c}}$ we have that

$$\mathsf{Succ}^{\mathbf{c}}_{ee'}(x) = \alpha(\mathbf{c})x + \beta(\mathbf{c})$$

with $\alpha > 0$ (which follows from the region's *goodness* property). Notice that we have $x' = \mathsf{Succ}^{\mathbf{c}}_{ee'}(x)$ iff $x \in e$, $x' \in e'$ and $x' = F(x)$. Thus,

$$x' = F(\{x\} \cap S) \cap J$$

with $F(x) = \alpha(\mathbf{c})x + \beta(\mathbf{c})$, $S = \langle e^l, e^u \rangle$ and $J = \langle e'^l, e'^u \rangle$, i.e.

$$x' = \mathcal{F}_{F, \langle e^l, e^u \rangle, \langle e'^l, e'^u \rangle}.$$

□

The notion of *successor* can be extended on all possible directions $\mathbf{c} \in \phi(P)$. Given $P \in \mathbb{P}$, $e \in In(P)$ and $e' \in Out(P)$, for $\mathbf{x} = (e, x)$, $\mathsf{Succ}_{ee'}(x)$ is the set of all points in e' reachable from $\mathbf{x}$ by a trajectory segment $\xi : [0, t] \to \mathbb{R}^2$ in P.

Definition 29 *Let $P \in \mathbb{P}$, $e \in In(P)$ and $e' \in Out(P)$. For $\mathbf{x} = (e, x)$, the* edge-to-edge successor $\mathsf{Succ}_{ee'}(x)$ *is defined as*

$$\mathsf{Succ}_{ee'}(x) = \{x' \mid \mathbf{x}' = (e', x') \wedge \xi(0) = x \wedge \xi(t) = x' \wedge \mathsf{Sig}(\xi) = ee'\}$$

■

$F^{\mathbf{c}}_{ee'}(x)$ will denote the *non-truncated* function $\alpha(\mathbf{c})x + \beta(\mathbf{c})$. The above notion of *successor* can be applied to any subset $A \subseteq \langle e^l, e^u \rangle$ and in particular to intervals $\langle l, u \rangle$:

Lemma 24 *Let $\phi(P) = \angle^{\mathbf{b}}_{\mathbf{a}}$, $\mathbf{x} = (e, x)$ and $\langle l, u \rangle \subseteq \langle e^l, e^u \rangle$. Then:*

1. $\mathsf{Succ}_{ee'}(x) = \bigcup_{\mathbf{c} \in \phi(P)} \mathsf{Succ}^{\mathbf{c}}_{ee'}(x) = \langle F^{\mathbf{b}}_{ee'}(x), F^{\mathbf{a}}_{ee'}(x) \rangle \cap \langle e'^l, e'^u \rangle$;
2. $\mathsf{Succ}_{ee'}(\langle l, u \rangle) = \langle F^{\mathbf{b}}_{ee'}(l), F^{\mathbf{a}}_{ee'}(u) \rangle \cap \langle e'^l, e'^u \rangle$.

Proof: It follows from the results given in chapter 3. □

Therefore, $\mathsf{Succ}_{ee'}$ is truncated affine multivalued:

$$\mathsf{Succ}_{ee'}(\langle l, u \rangle) = F_{ee'}(\langle l, u \rangle \cap \langle e^l, e^u \rangle) \cap \langle e'^l, e'^u \rangle$$

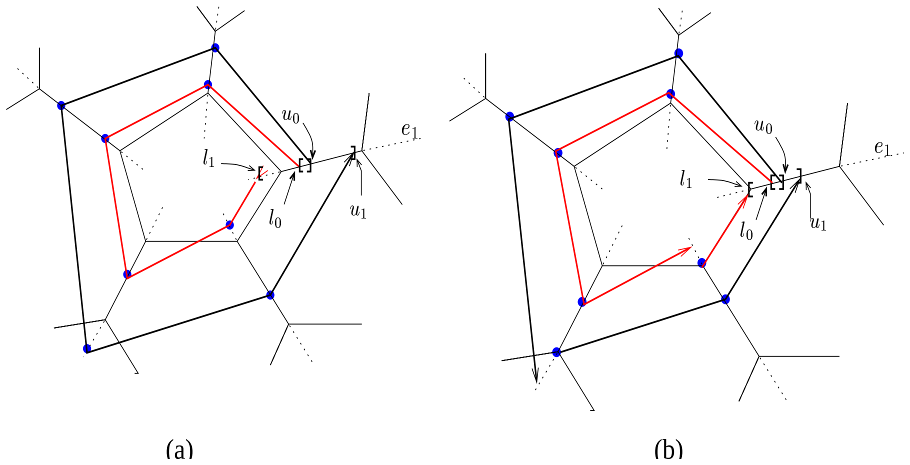


Figure 5.2: (a) Non-truncated operator: $\mathsf{Succ}_{e_1}(l_0, u_0) = \langle l_1, u_1 \rangle$, with $l_1 < e_1^l < u_1 \leq e_1^u$; (b) Truncated successor: $\mathsf{Succ}_{e_1}(l_0, u_0) = \langle l_1, u_1 \rangle \in \langle e_1^l, e_1^u \rangle$.

This lemma shows that in order to find a successor of an interval in an edge e, we should apply the smallest dynamics (**a**) to its right end and the greatest (**b**) to its left end, and intersect the result with the target edge. Fig. 5.2 shows the difference between non-truncated and truncated successors.

The successor operator will be used as a building block in the reachability algorithm. It can be naturally extended on edge signatures: for $\sigma_1 = e_1 e_2 \ldots e_n$ let

$$\mathsf{Succ}_{\sigma_1}(I) = \mathsf{Succ}_{e_{n-1}e_n} \circ \ldots \circ \mathsf{Succ}_{e_2 e_3} \circ \mathsf{Succ}_{e_1 e_2}(I)$$

that by Lemma 4 is truncated affine.

Notice that since we use edge signatures the semi-group property takes the following form.

Lemma 25 *For any edge signatures σ_1 and σ_2 and an edge e*

$$\mathsf{Succ}_{e\sigma_1} \circ \mathsf{Succ}_{\sigma_2 e} = \mathsf{Succ}_{\sigma_2 e \sigma_1}$$

□

It is convenient to define a (trivial) successor Succ_e where e is a single edge. The only way to do it preserving the semi-group property is to put $\mathsf{Succ}_e(x) = x$.

In order to manipulate successor operators we should investigate their algebraic properties. Since one-step successors $\mathsf{Succ}_{e_1 e_2}$ are truncated affine, Lemma 25 guarantees that all the multi-step Succ_u are truncated affine as well. In the sequel we will apply the iteration analysis to their non-truncated versions F_u.

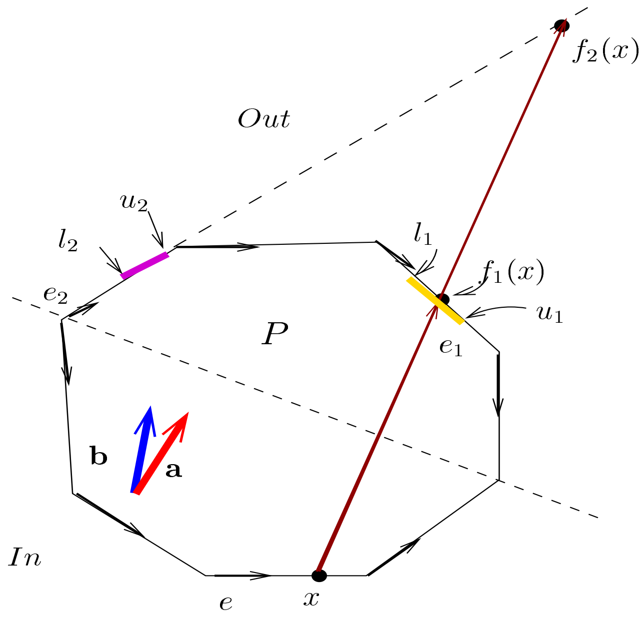


Figure 5.3: Statement of Lemma 26.

Lemma 26 *Let P be a region, $e \in In(P)$, $e_1, e_2 \in Out(P)$, $\langle l_i, u_i \rangle$ be any subinterval of $\langle e_i^l, e_i^u \rangle$ and $f_i(x) = F^{\mathbf{c}}_{ee_i}(x)$ (for any $\mathbf{c} \in \angle^{\mathbf{b}}_{\mathbf{a}}$). If $e_2 \prec e_1$ and $l_1 < f_1(x)$ then $u_2 < f_2(x)$.*

Proof: (See Fig. 5.3). By definition, from $e_2 \prec e_1$, we have that $\mathbf{e}_2 u_2 + \mathbf{v}_2 \preceq \mathbf{e}_1 l_1 + \mathbf{v}_1$ and by definition of $\preceq$ between points

$$\hat{\mathbf{c}}(\mathbf{e}_2 u_2 + \mathbf{v}_2) \leq \hat{\mathbf{c}}(\mathbf{e}_1 l_1 + \mathbf{v}_1).$$

Distributing we obtain that

$$\hat{\mathbf{c}}\mathbf{e}_2 u_2 + \hat{\mathbf{c}}\mathbf{v}_2 \leq \hat{\mathbf{c}}\mathbf{e}_1 l_1 + \hat{\mathbf{c}}\mathbf{v}_1.$$

On the other hand, from the hypothesis we have that

$$l_1 < \frac{\hat{\mathbf{c}}\mathbf{e}}{\hat{\mathbf{c}}\mathbf{e}_1} x + \frac{\hat{\mathbf{c}}(\mathbf{v} - \mathbf{v}_1)}{\hat{\mathbf{c}}\mathbf{e}_1}$$

and doing some algebraic manipulation we obtain

$$\hat{\mathbf{c}}\mathbf{e}_1 l_1 + \hat{\mathbf{c}}\mathbf{v}_1 < \hat{\mathbf{c}}\mathbf{v} + \hat{\mathbf{c}}\mathbf{e}x.$$

From both results we obtain (by transitivity) that

$$\hat{\mathbf{c}}\mathbf{e}_2 u_2 + \hat{\mathbf{c}}\mathbf{v}_2 < \hat{\mathbf{c}}\mathbf{v} + \hat{\mathbf{c}}\mathbf{e}x.$$

By rearranging we have that

$$u_2 < \frac{\hat{\mathbf{c}}\mathbf{e}}{\hat{\mathbf{c}}\mathbf{e}_2} x + \frac{\hat{\mathbf{c}}(\mathbf{v} - \mathbf{v}_2)}{\hat{\mathbf{c}}\mathbf{e}_2},$$

i.e. $u_2 < f_2(x)$. $\square$

As a consequence we have the following result.

Corollary 27 *Let P be a region, $e \in In(P)$, $e_1, e_2 \in Out(P)$, $f_i(x) = F^{\mathbf{c}}_{ee_i}(x)$ be an affine function and $\mathcal{F}_i(\langle x, y\rangle) = F_i(\langle x, y\rangle \cap S_i) \cap J_i$ be a truncated affine multi-valued function (with $F_i = [f_i^l, f_i^u]$ and $J_i = \langle L_i, U_i\rangle$). Given that $e_2 \prec e_1$ we have that*

1. *If $L_1 < f_1^l(x)$ then $\mathcal{F}_2(\langle x, y\rangle) = \emptyset$;*
2. *If $f_2^u(y) < U_2$ then $\mathcal{F}_1(\langle x, y\rangle) = \emptyset$.*

Proof: Direct from lemma 26 and definition of $\mathcal{F}_i(\langle x, y\rangle)$. $\square$

Example 18 Let us come back to the example of the swimmer trying to escape from a whirlpool in a river (see Fig. 2.5). Suppose that the swimmer is following a trajectory with edge signature $(e_1 \dots e_8)^*$. It is not difficult to find a representation of the edges such that for each edge e_i, $(e_i^l, e_i^u) = (0, 1)$. Besides, the (non-truncated) affine successor functions are:

$$\begin{array}{l}
F_{e_1e_2}(x) = \left[\frac{x}{2}, \frac{x}{2}\right] \\
F_{e_2e_3}(x) = \left[x - \frac{1}{4}, x + \frac{11}{60}\right] \\
F_{e_ie_{i+1}}(x) = [x, x], \text{ for all } i \in [3, 7] \\
F_{e_8e_1}(x) = \left[x + \frac{1}{5}, x + \frac{1}{5}\right]
\end{array}$$

The truncated affine version of the functions above (normalized) are

$$\begin{array}{lcl}
\mathsf{Succ}_{e_1e_2}(x) & = & \begin{cases} \left[\frac{x}{2}, \frac{x}{2}\right] \cap (0,1) & \text{if } x \in (0,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_2e_3}(x) & = & \begin{cases} \left[x - \frac{1}{4}, x + \frac{11}{60}\right] \cap (0,1) & \text{if } x \in (0,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_ie_{i+1}}(x) & = & \begin{cases} [x, x] \cap (0,1) & \text{if } x \in (0,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_8e_1}(x) & = & \begin{cases} \left[x + \frac{1}{5}, x + \frac{1}{5}\right] \cap (0,1) & \text{if } x \in (0, \frac{4}{5}) \\ \emptyset & \text{otherwise} \end{cases}
\end{array}$$

The successor function for the loop $s = e_1 \dots e_8$ is obtained by composition of the above functions as follows. Let us first compute

$$\mathsf{Succ}_{e_1e_2e_3}(l, u) = F(\langle l, u\rangle \cap S) \cap J$$

where

$$\begin{array}{ll}
F & = F_{e_2e_3} \circ F_{e_1e_2} \\
S & = S_1 \cap F^{-1}_{e_1e_2}(J_1 \cap S_2) \\
J & = J_2 \cap F_{e_2e_3}(J_1 \cap S_2)
\end{array}$$

with

$$\begin{array}{ll} J_0 & = e_1 = (0,1) \\ J_1 & = e_2 = (0,1) \\ J_2 & = e_3 = (0,1) \\ S_1 & = F^{-1}_{e_1e_2}(J_1) \cap J_0 \\ S_2 & = F^{-1}_{e_2e_3}(J_2) \cap J_1 \end{array}$$

and

$$\begin{array}{ll} F^{-1}_{e_1e_2}(l,u) & = [2l, 2u] \\ F^{-1}_{e_2e_3}(l,u) & = [l - \frac{11}{60}, u + \frac{1}{4}] \end{array}$$

We compute now all the parameters above in order to obtain F, S and J

$$\begin{array}{ll} S_1 & = F^{-1}_{e_1e_2}((0,1)) \cap (0,1) = (0,2) \cap (0,1) = (0,1) \\ S_2 & = F^{-1}_{e_2e_3}((0,1)) \cap (0,1) = (-\frac{11}{60}, \frac{5}{4}) \cap (0,1) = (0,1) \\ & \\ F(l,u) & = [\frac{l}{2} - \frac{1}{4}, \frac{u}{2} + \frac{11}{60}] \\ S & = (0,1) \cap F^{-1}_{e_1e_2}((0,1) \cap (0,1)) = (0,1) \cap (0,2) = (0,1) \\ J & = (0,1) \cap F_{e_2e_3}((0,1) \cap (0,1)) = (0,1) \cap (-\frac{1}{4}, \frac{71}{60}) = (0,1) \end{array}$$

We have then that

$$\mathsf{Succ}_{e_1e_2e_3}(l,u) = \begin{cases} [\frac{l}{2} - \frac{1}{4}, \frac{u}{2} + \frac{11}{60}] \cap (0,1) & \text{if } \langle l,u\rangle \subseteq (0,1) \\ \emptyset & \text{otherwise} \end{cases}$$

Since $F_{e_ie_{i+1}}$ for $i \in [3,7]$ are the identity functions, we have that

$$\mathsf{Succ}_{e_3\ldots e_8}(l,u) = \begin{cases} \langle l,u\rangle \cap (0,1) & \text{if } \langle l,u\rangle \subseteq (0,1) \\ \emptyset & \text{otherwise} \end{cases}$$

and composing the functions above we obtain that $\mathsf{Succ}_{e_1\ldots e_8} = \mathsf{Succ}_{e_1e_2e_3}$. We compute now

$$\mathsf{Succ}_{e_1\ldots e_8e_1}(l,u) = F'(\langle l,u\rangle \cap S') \cap J'$$

where

$$\begin{array}{ll} F' & = F_{e_8e_1} \circ F_{e_1\ldots e_8} \\ S' & = S_1 \cap F^{-1}_{e_1\ldots e_8}(J_1 \cap S_2) \\ J' & = J_2 \cap F_{e_8e_1}(J_1 \cap S_2) \end{array}$$

with

$$\begin{aligned} J_0 &= e_1 = (0,1) \\ J_1 &= J = (0,1) \\ J_2 &= e_1 = (0,1) \\ S_1 &= F^{-1}_{e_1 \ldots e_8}(J_1) \cap J_0 \\ S_2 &= F^{-1}_{e_8 e_1}(J_2) \cap J_1 \end{aligned}$$

and

$$\begin{aligned} F^{-1}_{e_1 \ldots e_8}(l,u) &= [2l - \tfrac{11}{30}, 2u + \tfrac{1}{2}] \\ F^{-1}_{e_8 e_1}(l,u) &= [l - \tfrac{1}{5}, u - \tfrac{1}{5}] \end{aligned}$$

We compute the parameters above to obtain F', S' and J':

$$\begin{aligned} S_1 &= F^{-1}_{e_1 \ldots e_8}((0,1)) \cap (0,1) = (-\tfrac{11}{30}, \tfrac{5}{2}) \cap (0,1) = (0,1) \\ S_2 &= F^{-1}_{e_8 e_1}((0,1)) \cap (0,1) = (-\tfrac{1}{5}, \tfrac{4}{5}) \cap (0,1) = (0, \tfrac{4}{5}) \end{aligned}$$

$$\begin{aligned} F'(l,u) &= [\tfrac{l}{2} - \tfrac{1}{20}, \tfrac{u}{2} + \tfrac{23}{60}] \\ S' &= (0,1) \cap F^{-1}_{e_1 \ldots e_8}((0,1) \cap (0, \tfrac{4}{5})) = (0,1) \cap (-\tfrac{11}{30}, \tfrac{21}{10}) = (0,1) \\ J' &= (0,1) \cap F_{e_8 e_1}((0,1) \cap (0, \tfrac{4}{5})) = (0,1) \cap (\tfrac{1}{5}, 1) = (\tfrac{1}{5}, 1) \end{aligned}$$

Hence,

$$\mathsf{Succ}_{e_1 \ldots e_8 e_1}(l,u) = \begin{cases} [\tfrac{l}{2} - \tfrac{1}{20}, \tfrac{u}{2} + \tfrac{23}{60}] \cap (\tfrac{1}{5}, 1) & \text{if } \langle l,u \rangle \subseteq (0,1) \\ \emptyset & \text{otherwise} \end{cases}$$

Finally, by Lemma 6 we obtain the limits: $l^* = \frac{-\frac{1}{20}}{1-\frac{1}{2}} = -\frac{1}{10}$, and $u^* = \frac{\frac{23}{60}}{1-\frac{1}{2}} = \frac{23}{30}$. ■

The notion of *edge signature* introduced in the previous chapter allows to consider one dimensional discrete systems instead of the two dimensional continuous systems we are leading with, and hence use the Poincaré map. Indeed, the following lemma shows that a successor function computes the Poincaré map of a trajectory segment.

Lemma 28 *Given an SPDI $\mathcal{H}$ and two points $\mathbf{x}_0 = (e_0, x_0)$ and $\mathbf{x}_f = (e_f, x_f)$, $Reach_\sigma(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ iff $x_f \in \mathsf{Succ}_\sigma(x_0)$.*

Proof: The result follows directly from definitions of $Reach_\sigma$ and Succ. □

We have introduced till now all the basic elements needed in order to attack the reachability problem and just remain the last step that is the analysis of each type of signature.

5.2 Reachability Analysis

In this section we present our main result, namely a decision procedure to solve the reachability problem for SPDIs. After showing that there are just five different possible behaviors of cycles analysing the limit trajectories we give then an algorithm that compute the exit set of each sequential part and each cycle and that test whether a given point is reachable or not.

5.2.1 Main algorithm

Given an SPDI $\mathcal{H}$, we are interested in the reachability analysis between two points.

From the previous chapter we know that there exists a finite number of type signatures of the form $r_1, s_1, \ldots, r_n, s_n, r_{n+1}$. Moreover, the type signatures are restricted to those with $e_0 = first(r_1)$ and $e_f \in r_{n+1}$. Given such a set of type signatures $\mathcal{T}(e_0, e_f)$, the following algorithm is guaranteed to terminate, answering YES if $\mathbf{x}_f$ is reachable from $\mathbf{x}_0$ or NO otherwise:

function $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$
  $R = \mathsf{false}$
  for each $\tau \in \mathcal{T}(e_0, e_f)$
    $R = R \vee Reach_{type}(x_0, x_f, \tau)$
  $\longleftarrow R$

Reachability from $\mathbf{x}_0$ to $\mathbf{x}_f$ with fixed type of signature τ is tested by the function

$$Reach_{type}(x_0, x_f, \tau).$$

Let the type τ have the form $\tau = r_1, s_1, \ldots, r_n, s_n, r_{n+1}$. Put $f_i = first(s_i)$ and $e_{x_i} = first(r_{i+1})$ if r_{i+1} is non-empty and f_{i+1} otherwise (i.e. e_{x_i} is the edge to which the trajectory exits from the loop s_i). Let us say that a type signature τ has a $\mathsf{loop}_{\mathsf{end}}$ property if $first(r_{n+1}) = first(s_n)$, i.e. signatures of type τ terminate by several repetitions of the last loop.

function $Reach_{type}(x_0, x_f, \tau)$:
  $Z = \mathsf{Succ}_{r_1 f_1}(x_0)$
  for $i = 1$ *to* $n-1$
    $Z = \mathsf{Succ}_{r_{i+1} f_{i+1}}(Exit(Z, s_i, e_{x_i}))$
  if $\mathsf{loop}_{\mathsf{end}}(\tau)$
    then $\longleftarrow Test(Z, s_n, x_f)$
    else $\longleftarrow x_f \in \mathsf{Succ}_{r_{n+1}}(Exit(Z, s_n, e_{x_n}))$?

This algorithm uses two functions:

1. $Test(Z, s, x)$ that answers whether x is reachable from a set Z (represented as a finite union of intervals) in the loop s. Formally, it checks whether $x \in \mathsf{Succ}_{s^+ first(s)}(I)$, i.e.,

$$\exists k \geq 1 \ . \ x \in \mathsf{Succ}_{s^k first(s)}(I)?$$

2. The function $Exit(Z, s, e)$ that for an initial set Z, a loop s, and an edge e (not in this loop) finds all the points on e reachable by making s several times and then exiting to e. Formally, it computes

$$\mathsf{Succ}_{s+e}(I) = \bigcup_{k\geq 1} \mathsf{Succ}_{s^k e}(I)$$

which is always a finite union of intervals.

Since we know how to calculate the successor of a given interval in one and in several steps ($\mathsf{Succ}_{ee'}(\cdot)$ and $\mathsf{Succ}_r(\cdot)$), in order to implement $Test(\cdot)$ and $Exit(\cdot)$ it remains to show how to analyze the (simple) cycles s_i and eventually their continuation.

Both algorithms $Test(\cdot)$ and $Exit(\cdot)$ start by doing qualitative analysis of the cycle. This analysis proceeds as follows.

Let s be a simple cycle, $f = first(s)$ its first edge, and $I = \langle l, u\rangle \subset f$ an initial interval and $\mathsf{Succ}_{sf}(x) = F_{sf}(\{x\} \cap S) \cap J$. Notice that the above formula is correct (by definition of truncated successor) but if we want to iterate the successor (apply it again) we must guarantee that we can really do it. Thus, the result should be in S and hence we obtain that to be able to iterate, $\mathsf{Succ}_{sf}(x)$ must be included in $S \cap J$, that is then the precondition for cycling. In what follows $\langle L, U\rangle$ will denote $S \cap J$.

The first thing to do is to determine the qualitative behavior of the leftmost and rightmost trajectories of the interval endpoints in the cycle. This can be done without iterating Succ_{sf}. Indeed, by Lemma 6, we can compute the limits $(l^*, u^*) = \lim_{n\to\infty} F^n_{sf}(\langle l, u\rangle)$ (notice that those are limits only for the *non-truncated operator* F), not taking into account that the edges are possible bounded (we use Lemma 11) and compare these limit points corresponding to unrestricted dynamics with L and U. There are five possibilities:

1. **STAY** The cycle is not abandoned by any of the two trajectories[2]: $L \leq l^* \leq u^* \leq U$;

2. **DIE** The right trajectory exits the cycle through the left (consequently the left one also exits) or the left trajectory exits the cycle through the right (consequently the right one also exits). In symbols, $u^* < L \vee l^* > U$, see Fig. 5.4;

3. **EXIT-BOTH** Both trajectories exit the cycle (the left one through the left and the right one through the right): $l^* < L \wedge u^* > U$, see Fig. 5.5;

4. **EXIT-LEFT** The leftmost trajectory exits the cycle but not the other: $l^* < L \leq u^* \leq U$, see Fig. 5.6.

5. **EXIT-RIGHT** The rightmost trajectory exits the cycle but not the other: $L \leq l^* \leq U < u^*$.

[2]See Appendix A for some properties of STAY signatures.

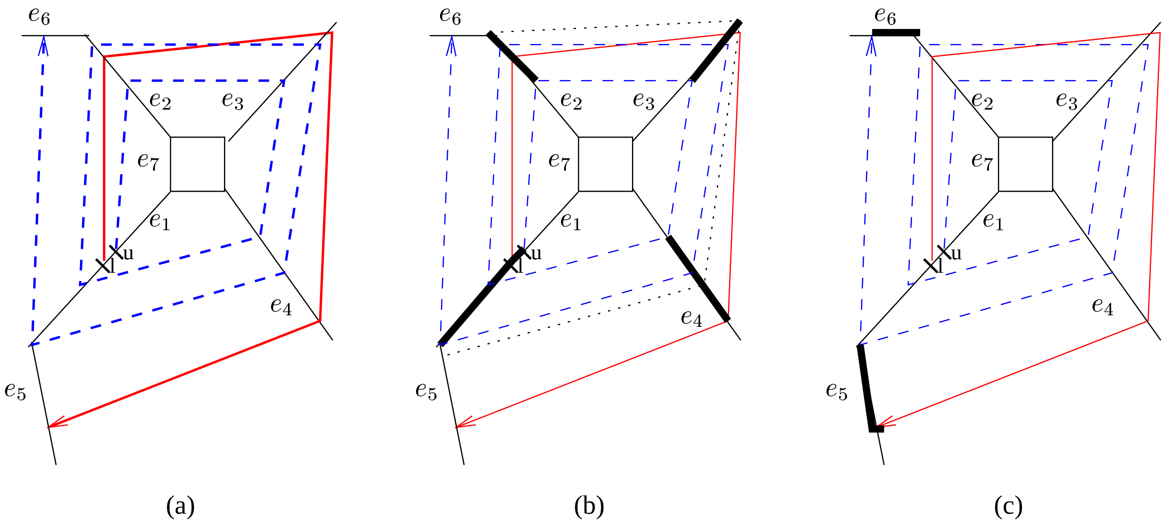


Figure 5.4: [DIE] (a) Both trajectories leave the cycle $(e_1, e_2, e_3, e_4)^*$ through the left; (b) Reachable points on the cycle (in bold); (c) Possible continuation after leaving the cycle (in bold).

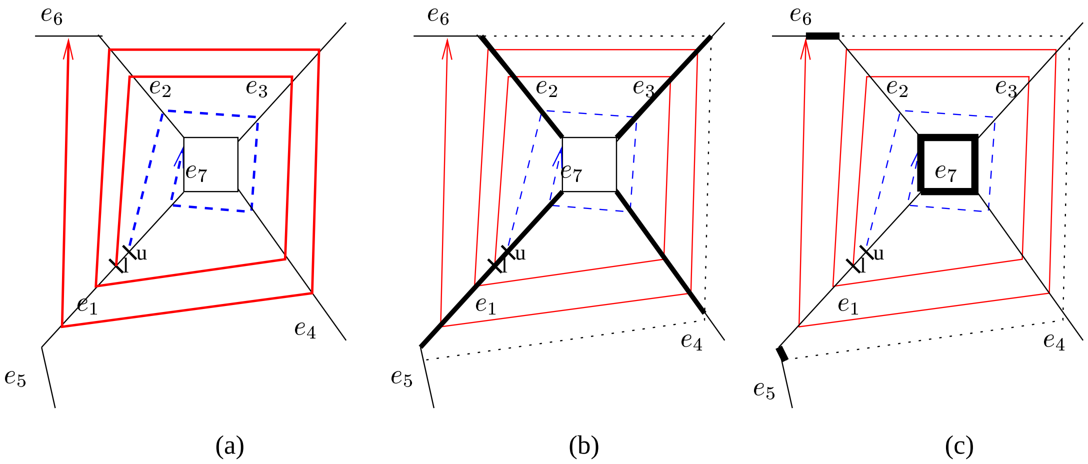


Figure 5.5: [EXIT-BOTH]] (a) Both trajectories leave the cycle $(e_1, e_2, e_3, e_4)^*$; (b) Reachable points on the cycle (in bold); (c) Possible continuation after leaving the cycle (in bold).

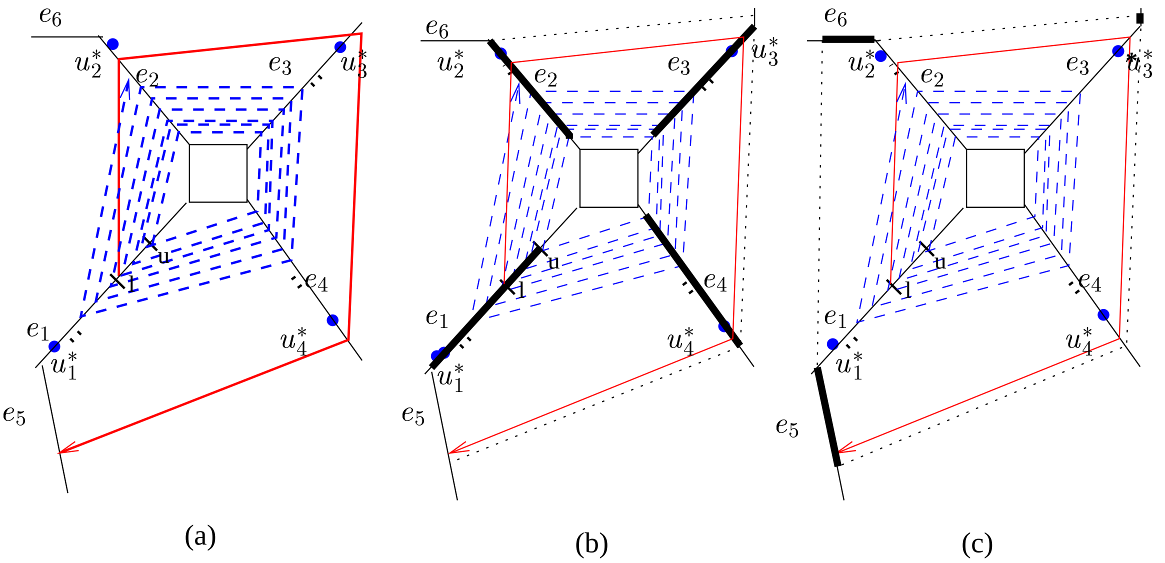


Figure 5.6: [EXIT-LEFT] (a) The left trajectory leave the cycle $(e_1, e_2, e_3, e_4)^*$ through the left, whereas the right one tends to the limit u^*; (b) Reachable points on the cycle (in bold); (c) Possible continuation after leaving the cycle (in bold).

This qualitative analysis is implemented in the function $Analyze(I, s)$ which returns the kind of qualitative behavior of the interval $I = \langle l, u \rangle$ under the loop s.

function $Analyze(I, s)$
  cases
    $L \leq l^* \leq u^* \leq U$: $\longleftarrow$ STAY
    $u^* < L \vee l^* > U$: $\longleftarrow$ DIE
    $l^* < L \wedge u^* > U$: $\longleftarrow$ EXIT-BOTH
    $L \leq l^* \leq U < u^*$: $\longleftarrow$ EXIT-RIGHT
    $l^* < L \leq u^* \leq U$: $\longleftarrow$ EXIT-LEFT
  endcases

Notice that one (or both) of the successor functions can be the identity. In this case we have an infinite number of fixpoints but the analysis above continue to apply.

5.2.2 Exit

In this section we describe the EXIT algorithm and show its soundness and termination. The *exit set* on a given edge e_x after cycling on s, for a given initial interval I, is

$$Ex = \bigcup_{m>0} \mathsf{Succ}_{se_x} \circ \mathsf{Succ}_{sf}^{m}(I).$$

The function $Exit(Z, s, e_x)$ should return $\mathsf{Succ}_{s+e_x}(Z)$. Both the argument Z and the result are finite collections of intervals. The exploration is made for each initial interval separately.

```
function  Exit(Z, s, e_x)
          E = ∅
          for each  I ∈ Z
                    if  Succ_sf(I) ∩ S ≠ ∅
                    then E = E ∪ Exit_Analyze(Succ_sf(I) ∩ S, s, e_x)
                    else E = E ∪ Succ_se_x(Succ_sf(I))
          ←— E
```

Notice that the call $\mathsf{Succ}_{sf}(I)$ ensures that $I \subseteq \langle L, U\rangle$. All the work for each initial interval I is done by the function $Exit_{Analyze}(I, s, e_x)$ which launches the $Analyze(\cdot)$ procedure described above and last, according to the result of this analysis launches one of five specialized procedures $Exit_{STAY}$, $Exit_{LEFT}$, $Exit_{RIGHT}$, $Exit_{BOTH}$, $Exit_{DIE}$ which calculates the exit set.

We consider separately all these five algorithms and state their termination and soundness. This will imply termination and soundness of the $Exit$ function itself.

Notation. We recall the notations introduced before and we introduce others to simplify the proofs. As before, let s be a simple cycle, $f = first(s)$ its first edge and $I = \langle l, u\rangle \subset f$ be the initial interval. Notice that the functions $Exit_*$ are always called with $I \subseteq \langle L, U\rangle$ (in fact this is the precondition for iterating, see Lemma 11). Let $I_i = \langle l_i, u_i\rangle = \mathsf{Succ}_{sf}^{i}(I)$ and $\tilde{I}_i = \langle \tilde{l}_i, \tilde{u}_i\rangle = F_{sf}^{i}(I)$. The Fundamental Lemma (Lemma 11) guarantees that $I_i = \tilde{I}_i \cap \langle L, U\rangle$. Remember that $\mathcal{F}(I) = \mathsf{Succ}_{sf}(I) = F_{sf}(I \cap S) \cap J$ and $\hat{\mathcal{F}}(I) = F_{sf}(I \cap S \cap J) \cap S \cap J$.

Exit-STAY

```
function  Exit_STAY(I, s, e_x)
          ←— ∅
```

Soundness By hypothesis, $L < l^* < u^* < U$. Hence, for all i, $\tilde{I}_i = \langle \tilde{l}_i, \tilde{u}_i\rangle \subseteq \langle L, U\rangle$, hence $I_i = \tilde{I}_i$ and by Corollary 27 we have that $\mathsf{Succ}_{se_x}^{i}(I) = \emptyset$.

Termination Trivial. □

Exit-DIE

```
function  Exit_DIE(I, s, e_x)
          Z = ∅
          repeat
                  I = Succ_sf(I)
                  Z = Z ⋃ Succ_se_x(I)
          until I = ∅
          ⟵ Z
```

Soundness Trivial.

Termination From the hypothesis we know that there exists an n s.t. $\tilde{I}_n \cap \langle L, U\rangle = \emptyset$ (either because $\tilde{u}_n < L$ if $u^* < L$ or because $U < \tilde{l}_n$ if $U < l^*$). Both cases imply that $\mathsf{Succ}^n_{sf}(I) = \emptyset$. □

Exit-BOTH

```
function  Exit_BOTH(I, s, e_x)
          ⟵ Succ_se_x(Succ_sf(⟨L, U⟩))
```

Soundness Notice that we call $Exit_{BOTH}$ with $\mathsf{Succ}_{sf}(I) \cap S = \mathcal{F}(I)$. On the other hand, because the limits are out of $\langle L, U\rangle$, we know that there exists an n such that $\langle L, U\rangle \subset \tilde{I}_n$ and by the Fundamental Lemma (Lemma 11), $\hat{\mathcal{F}}^n(I) = I_n = \langle L, U\rangle$ (i.e. $\hat{\mathcal{F}}^n \circ \mathcal{F}(I) = \langle L, U\rangle$). By Corollary 13 we have that $\mathcal{F}^n(I) = \mathcal{F} \circ \hat{\mathcal{F}}^{n-2} \circ \mathcal{F}(I) = \mathcal{F}(\langle L, U\rangle) = \mathsf{Succ}_{sf}(\langle L, U\rangle)$.

1. We prove first that the algorithm produces just 'exits' (i.e., $\mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(\langle L, U\rangle)) \subseteq Ex$). This follows directly from the fact that $\mathsf{Succ}_{sf}(\langle L, U\rangle) = \mathsf{Succ}^n_{sf}(I) \subseteq \cup_{m>0}\mathsf{Succ}^m_{sf}(I)$;
2. We prove now that all the 'exits' are computed ($Ex \subseteq \mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(\langle L, U\rangle))$). By definition, $Ex = \cup_{m>0}\mathsf{Succ}_{se_x} \circ \mathcal{F}^m(I)$, that can be written as $Ex = \mathsf{Succ}_{se_x} \circ \mathcal{F}(I) \cup \mathsf{Succ}_{se_x} \circ \mathcal{F} \circ \mathcal{F}(\cup_{m\geq 2}\mathcal{F}^{m-2}(I))$. Let A be the set $\cup_{m\geq 2}\mathcal{F}^{m-2}(I)$, thus $\mathcal{F} \circ \mathcal{F}(A) = \mathcal{F}(S \cap \mathcal{F}(A)) \subseteq \mathcal{F}(S \cap J) = \mathcal{F}(\langle L, U\rangle)$. On the other hand, $\mathsf{Succ}_{se_x} \circ \mathcal{F}(I) \subseteq \mathsf{Succ}_{se_x} \circ \mathcal{F}(\langle L, U\rangle)$, since $I \subseteq \langle L, U\rangle$ and by monotonicity of both functions. Hence, $Ex \subseteq \mathsf{Succ}_{se_x} \circ \mathcal{F}(\langle L, U\rangle)$.

Termination Trivial. □

Exit-LEFT

```
function  Exit_LEFT(I, s, e_x)
          ⟵ Succ_se_x(Succ_sf(⟨L, max{u, u*}⟩))
```

Soundness By hypothesis, $l^* < L < u^* \leq U$. Thus, there exists a natural number n s.t. $\tilde{l_n} \leq L$ and for all i, $u_i = \tilde{u}_i \leq U$. Let's consider the following two cases:

1. If $f \prec e_x$ then $Ex = \emptyset$ (by definition of Exit-LEFT) and $\mathsf{Succ}_{se_x}(I_i) = \emptyset$ for any i (by Corollary 27-2), so $\mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(\langle L, \max\{u, u^*\}\rangle)) = \emptyset$;
2. If $e_x \prec f$, we consider two cases:
   (a) If $u < u^*$ then for all i, $u_i = \tilde{u}_i \leq u^*$ and then $\cup_{m>0}\mathsf{Succ}^m_{sf}(I) = \mathsf{Succ}_{sf}(L, u^*)$, thus $Ex = \mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(L, u^*))$;
   (b) If $u^* < u$ then for all i, $u_i = \tilde{u}_i \leq u$ and $\cup_{m>0}\mathsf{Succ}^m_{sf}(I) = \mathsf{Succ}_{sf}(L, u)$. Consequently, $Ex = \mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(L, u))$;

   From both cases we have that $Ex = \mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(\langle L, \max\{u, u^*\}\rangle))$.

Termination Trivial. □

In fact, it can be shown that $Exit_{LEFT} = \mathsf{Succ}_{se_x}(\mathsf{Succ}_{sf}(\langle L, U\rangle))$.

Exit-RIGHT

Similar to the previous case.

5.2.3 Test

In this section we describe the Test function and show its soundness and termination. In what follows, $l \uparrow$ means that the sequence $l, l_1, l_2, \ldots$ of successive successors of l is increasing whereas $l \downarrow$ means that the sequence is decreasing. Similarly for $u \uparrow$ and $u \downarrow$. Notice that detecting whether the sequences l_n and u_n are increasing or decreasing can be easily done at the stage of the preliminary analysis of the loop.

The upper-level structure is the same as for EXIT: each initial interval is treated separately by $Test_{Analyze}$, which makes one turn of the loop, calls $Analyze$ and delegates all the remaining to one of the five specialized functions $Test_{STAY}$, $Test_{LEFT}$, $Test_{RIGHT}$, $Test_{BOTH}$, $Test_{DIE}$.

```
function  Test(Z, s, x)
          R = false
          for each  I ∈ Z such that Succ_sf(I) ∩ S ≠ ∅
                    R = R ∨ Test_Analyze(Succ_sf(I), s, x)
          ⟵ R
```

The five specialized $Test$ functions use the following two procedures: The function $Found(I, x)$ determines, if the current interval I contains x (YES), does not contain x and moves in the opposite direction (NO), or none of both these cases (NOTYET). The function $Search(I, x)$

iterates the loop s until the previous function $Found$ gives a definite answer YES or NO. Special measures will be taken to guarantee termination.

$$
\begin{array}{l}
\textbf{function}\ \ Found(I,x)\\
\qquad\qquad \textbf{cases}\\
\qquad\qquad\qquad x \in I : \qquad\quad \longleftarrow \text{YES}\\
\qquad\qquad\qquad I = \emptyset : \qquad\quad \longleftarrow \text{NO}\\
\qquad\qquad\qquad x < I \wedge l \uparrow : \longleftarrow \text{NO}\\
\qquad\qquad\qquad x > I \wedge u \downarrow : \longleftarrow \text{NO}\\
\qquad\qquad\qquad \textbf{else} : \qquad\quad \longleftarrow \text{NOTYET}\\
\qquad\qquad \textbf{endcases}
\end{array}
$$

$$
\begin{array}{l}
\textbf{function}\ \ Search(I,x)\\
\qquad\qquad \textbf{while}\ Found(I,x) = \text{NOTYET}\\
\qquad\qquad\qquad I = \mathsf{Succ}_{sf}(I)\\
\qquad\qquad \longleftarrow Found(I,x)
\end{array}
$$

Test-STAY

$$
\begin{array}{l}
\textbf{function}\ \ Test_{STAY}(I,s,x)\\
\qquad\qquad \textbf{cases}\\
\qquad\qquad\qquad l^* < x < u^* : \longleftarrow \text{YES}\\
\qquad\qquad\qquad x \leq l^* \wedge l \downarrow : \longleftarrow \text{NO}\\
\qquad\qquad\qquad x \geq u^* \wedge u \uparrow : \longleftarrow \text{NO}\\
\qquad\qquad\qquad \textbf{else} : \qquad\quad \longleftarrow Search(I,x)\\
\qquad\qquad \textbf{endcases}
\end{array}
$$

Soundness We prove the soundness considering each case separately:

1. We have to prove that if $l^* < x < u^*$ then $x \in Reach(I)$. By hypothesis $l^* < x_f < u^*$, then there exists a positive real number ϵ such that $l^* + \epsilon < x_f < u^* - \epsilon$. It's not difficult to see that exists two real numbers N_1 and N_2 such that for all n greater (or equal) than N_1, $u_n > u^* - \epsilon$ and for all n greater (or equal) than N_2, $l_n < l^* + \epsilon$. Let N be equal to the maximum between N_1 and N_2, then it follows that $l_N < l^* + \epsilon$ and $u^* - \epsilon < u_N$. Thus, $l_N < x_f < u_N$ and x_f is reachable.
2. We have to prove that if $x \leq l^* \wedge l \downarrow$ then $x \notin Reach(I)$. Trivial, by definition of limit and monotonicity of the sequence.
3. We have to prove that if $u^* \leq x \wedge u \uparrow$ then $x \notin Reach(I)$. Trivial, by definition of limit and monotonicity of the sequence.
4. We have to prove that if $(x < l^* \wedge l \uparrow) \vee (u^* < x \wedge u \downarrow)$ then $Search(I,x) \equiv (x \in Reach(I)?)$. Computing $Search(I,x)$ gives a sequence of intervals $I, I_1, \ldots, I_n$

s.t. $Reach(I) = \bigcup_i I_i$. If $Search(I,x)$ terminates then $\exists i \cdot (Found(I_i,x) =$ YES $\vee Found(I_i,x) =$ NO) and $\forall j < i \cdot Found(I_i,x) =$ NOTYET. We analyze then each of the cases of $Found(I,x)$:

(a) If $x \in I$ then $Found(I_i,x) =$ YES and $x \in I_i$, i.e. $x \in Reach(I)$.

(b) If $I = \emptyset$ then $Found(I_i,x) =$ NO and $\forall k \geq i \cdot I_k = \emptyset$ and $x \notin I_j$. Thus $x \notin Reach(I)$.

(c) If $x < I \wedge l \uparrow$ then $Found(I_i,x) =$ NO and $\forall k \geq i \cdot x < l_i < l_k$ and because $x \notin I_j$ then $x \notin I_k$ and hence $x \notin Reach(I)$.

(d) If $I < x \wedge u \downarrow$ then $Found(I_i,x) =$ NO and $\forall k \geq i \cdot u_k < u_i < x$ and because $x \notin I_j$ then $x \notin I_k$ and hence $x \notin Reach(I)$.

Termination We have to show termination just when $(x < l^* \wedge l \uparrow) \vee (u^* < x \wedge u \downarrow)$. If $x < l^* \wedge l \uparrow$ then $\exists i \cdot (x < l_i < l^* \wedge Found(I_i,x) = NO)$. Thus, it terminates. Similarly for the other case. □

Test-DIE

function $Test_{DIE}(I,s,x)$ $\longleftarrow Search(I,x)$

Soundness Trivial.

Termination Eventually I becomes empty. Hence, at this stage $Found(I,x) =$ NO and $Search$ terminates. □

Test-BOTH

function $Test_{BOTH}(I,s,x)$ $\longleftarrow x \in \mathsf{Succ}_{sf}(\langle L,U\rangle)?$

Soundness Immediate from the proof of soundness of the Exit algorithm for EXIT-BOTH.

Termination Trivial. □

Test-LEFT

$$
\begin{array}{l}
\textbf{function } Test_{LEFT}(I, s, x) \\
\quad \textbf{cases} \\
\quad\quad x \in \mathsf{Succ}_{sf}(\langle L, u^* \rangle) : \quad \longleftarrow \text{YES} \\
\quad\quad x < \mathsf{Succ}_{sf}(\langle L, u^* \rangle) : \quad \longleftarrow \text{NO} \\
\quad\quad \mathsf{Succ}_{sf}(\langle L, u^* \rangle) < x \wedge u \uparrow : \longleftarrow \text{NO} \\
\quad\quad \textbf{else} : \quad \longleftarrow Search(I, x) \\
\quad \textbf{endcases}
\end{array}
$$

Soundness The proof is similar to the STAY case.

Termination We have to consider just the case when $u \downarrow$ and $\mathsf{Succ}_{sf}(\langle L, u^* \rangle) < x$. In this case we know that $\exists i \cdot u^* < u_i < x \;\wedge\; Found(I_i, x) = \text{NO}$. Thus the algorithm terminates. □

Test-RIGHT

The algorithm and its correctness proof are similar to the previous case. □

5.2.4 Examples

In this section we present two examples of the application of the reachability algorithm for SPDIs.

Example 19 Consider again the swimmer of Figure 2.5 defined in section 2.4. Let $\mathbf{x}_0 = (e_1, \frac{1}{2})$ be her initial position. We want to decide whether she is able to escape from the whirlpool and reach the final position $\mathbf{x}_f = (e_1, \frac{3}{4})$. Recall that $(L, U) = S \cap J = (\frac{1}{5}, 1)$ and

$$l^* = \frac{-\frac{1}{20}}{1 - \frac{1}{2}} = -\frac{1}{10}$$

and

$$u^* = \frac{\frac{23}{60}}{1 - \frac{1}{2}} = \frac{23}{30}$$

Thus, by the *Analyze* function we know that the cycle behaves as an Exit-LEFT and applying the function $Test_{LEFT}$ we obtain that $\mathbf{x}_f = (e_1, \frac{3}{4})$ is reachable from $\mathbf{x}_0 = (e_1, \frac{1}{2})$ because we have that

$$\mathsf{Succ}_{e_1 e_8 \cdots e_1}((L, u^*)) = \mathsf{Succ}_{e_1 e_8 \cdots e_1}((\frac{1}{5}, \frac{23}{30})) = (\frac{1}{20}, \frac{23}{30})$$

and

$$\frac{3}{4} \in (\frac{1}{20}, \frac{23}{30})$$

See Figure 5.7. ■

Example 20 Let's change the above example in order to show another behavior. For simplicity we consider the same partition of the swimmer but with the following differential inclusion dynamics:

- $R_1 : \mathbf{a} = (1, \frac{10}{3}), \mathbf{b} = (1, 5)$;
- $R_2 : \mathbf{a} = \mathbf{b} = (-1, 1)$;
- $R_3 : \mathbf{a} = \mathbf{b} = (-1, 0)$;
- $R_4 : \mathbf{a} = \mathbf{b} = (-1, -1)$;
- $R_5 : \mathbf{a} = \mathbf{b} = (0, -1)$;
- $R_6 : \mathbf{a} = \mathbf{b} = (1, -1)$;
- $R_7 : \mathbf{a} = \mathbf{b} = (1, 0)$;
- $R_8 : \mathbf{a} = \mathbf{b} = (1, 1)$.

We are interested in the edge signature $e_0(e_1 \dots e_8)^* e_9$, and what matters for computing the reachable points of e_9 starting from $x_0 \in e_0$ are the following edge-to-edge successor functions:

$$\begin{aligned}
\mathsf{Succ}_{e_0e_1}(x) &= \begin{cases} \left[\frac{1}{5}x, \frac{3}{10}x\right] \cap (0,1) & \text{if } x \in (0,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_ie_{i+1}}(x) &= \begin{cases} [x, x] \cap (0,1) & \text{if } x \in (0,1) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_8e_1}(x) &= \begin{cases} \left[x + \frac{1}{5}, x + \frac{3}{10}\right] \cap (0,1) & \text{if } x \in (0, \frac{4}{5}) \\ \emptyset & \text{otherwise} \end{cases} \\
\mathsf{Succ}_{e_8e_9}(x) &= \begin{cases} \left[5x - 4, \frac{10}{3}x - \frac{7}{3}\right] \cap (0,1) & \text{if } x \in (\frac{7}{10}, 1) \\ \emptyset & \text{otherwise} \end{cases}
\end{aligned}$$

Let x_0 be equal to $\frac{1}{2}$ on edge e_0 and x_f be $\frac{3}{10}$ on e_9; deciding whether exists a trajectory from $(e_0, \frac{1}{2})$ to $(e_9, \frac{3}{10})$ can be done following the steps:

1. Compute the "enter interval" to the loop: $\mathsf{Succ}_{e_0e_1}(\frac{1}{2}) = [\frac{1}{10}, \frac{3}{20}]$;
2. Compute the successor function of the loop $(e_1 \dots e_8)^{*3}$:

$$\mathsf{Succ}_{e_1 \dots e_8 e_1}[l, u] = \begin{cases} \left[l + \frac{1}{5}, u + \frac{3}{10}\right] \cap (\frac{1}{5}, 1) & \text{if } [l, u] \subseteq (0, \frac{4}{5}) \\ \emptyset & \text{otherwise} \end{cases}$$

[3] Notice that in fact this function is the same as $\mathsf{Succ}_{e_8e_1}$ since the other functions are the identity.

Iteration	I	Z
0	$[\frac{1}{10}, \frac{3}{20}]$	$\emptyset$
1	$[\frac{3}{10}, \frac{9}{20}]$	$\emptyset$
2	$[\frac{1}{2}, \frac{3}{4}]$	$\{[0, \frac{1}{6}]\}$
3	$[\frac{7}{10}, 1]$	$\{[0, 1]\}$
4	$[\frac{9}{10}, 1]$	$\{[0, 1]\}$

Table 5.1: Execution trace of the cycle $e_1 \dots e_8$ starting from $[0.1, 0.15] \in e_1$. I represents the current interval (in e_1) and Z is the set of exit intervals (in e_9).

3. Compute the limits of the loop signature: By Lemma 6 we have that $u^* = l^* = \infty$ for both affine functions. We can then conclude that the trajectories will be counterclockwise expanding spirals and the *Analyze* function gives that the loop will behave as a DIE (see section 5.2.1);

4. Execute the function $Exit_{DIE}([\frac{1}{10}, \frac{3}{20}], e_1 \dots e_8, e_9) = \{[0, 1]\}$.

The execution trace is given in Table 5.1, where in the Z column we can see the set of (truncated) exit intervals over the edge e_9: in the third iteration the exit interval is the whole edge e_9. From the above we conclude that $(e_9, \frac{3}{10})$ is reachable from $(e_0, \frac{1}{2})$.

As an example of a non reachable point, consider the edge signature $(e_1 \dots e_8)^*$ with $[\frac{9}{10}, \frac{19}{20}] \in e_1$ as initial interval and $x_f = \frac{3}{10}$ in e_9 as before. After computing the corresponding functions we obtain that in the first iteration the loop is left and the exit interval on edge e_9 is $[\frac{1}{2}, \frac{5}{6}]$, from where we can conclude that $(e_9, \frac{3}{10})$ is not reachable from $(e_1, [\frac{9}{10}, \frac{19}{20}])$. ■

5.2.5 Main result

Notice that the function $Reach_{type}(x_0, x_f, \tau)$ of the previous section computes $Reach_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ and hence the algorithm $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ computes the following:

$$Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \forall \tau \in \mathcal{T}_{feasi} \; . \; Reach_\tau(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f).$$

From the previous section and the main result of chapter 4 we have the following theorem.

Theorem 29 *The algorithm $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ is sound and complete.*

Proof: Soundness follows from the soundness of all the functions used in the algorithm that has already been proved. We have to prove that $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ computes the good result for all the existing trajectory segments from $\mathbf{x}_0$ to $\mathbf{x}_f$, but this follows from Theorem 22 and the fact that all the types of feasible signatures are considered. □

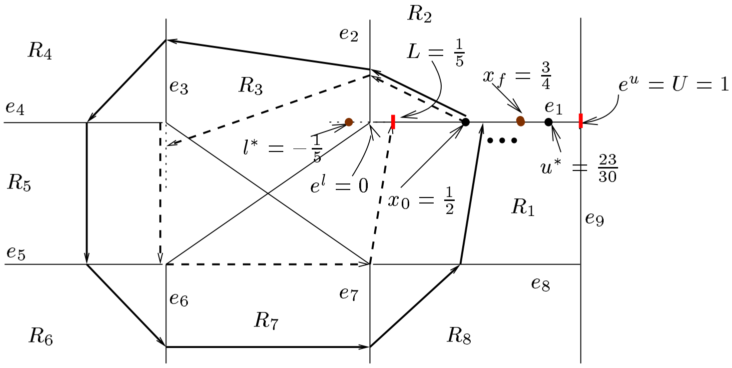


Figure 5.7: Example: $\mathbf{x}_f = (e_1, \frac{3}{4})$ is reachable from $\mathbf{x}_0 = (e_1, \frac{1}{2})$ ($\frac{3}{4} \in \mathsf{Succ}_{e_1 e_8 \cdots e_1}(L, u^*)$).

From all the above we have the main result of this chapter that is a positive answer to the reachability problem REACH_{SPDI} (Problem 1).

Theorem 30 (Point–to–Point Reachability) *The problem* REACH_{SPDI} *is decidable.* □

It is not difficult to see that the result also holds for edge–to–edge and region–to–region reachability.

5.3 Summary and Related Work

In this chapter we have presented an algorithm for solving the reachability problem for SPDIs. The novelty of the approach for the domain of Hybrid System is the combination of two techniques, namely, the representation of the two-dimensional continuous dynamics as a one-dimensional discrete system (due to Poincaré, see for example [83, 115]), and the characterization of the set of qualitative behaviors of the latter as a finite set of types of signatures.

Most of the proved decidability results in the literature are based on the existence of a finite and computable partition of the state space into classes of states which are equivalent with respect to reachability. Even though analysis techniques based on the construction of a finite partition have been proposed [51], mainly all implemented computational procedures resort to (forward or backward) propagation of constraints, typically (unions of convex) polyhedra or ellipsoids [4, 17, 42, 63, 73, 97]. The geometric approach used for our decision procedure has been presented in [105] where it is shown that the reachability problem for two-dimensional systems with piece-wise constant derivatives (PCD) is decidable. This result has been extended in [139] for planar piecewise Hamiltonian systems. In [19] it has been shown that the reachability problem for PCD is undecidable for dimensions higher than two (see chapter 2) what is really bad news, since no extension of the method can be thinkable beyond two dimension, at least as an exact decision procedure for reachability.

Chapter 6

SPDI - Phase Portrait

In this chapter we study phase portraits of SPDIs. It is not a priori clear what the phase portraits of such systems exactly are, mainly due to the inherent non-determinism. To begin with, we concentrate on the qualitative behavior of sets of trajectories having the same cyclic pattern, based on the classification of cyclic behaviors given in section 5.2. We compute the *controllability* and *viability* kernels of SPDI and we show that they can be seen as limit cycles and their basins of attraction respectively. We also prove some convergence properties of SPDI trajectories without self-crossings. The phase portrait of an SPDI is then obtained computing all the viability and controllability kernels.

This chapter is an extended version of the results presented in [23].

The chapter is organized as follows: In the first section we recall some definitions and we explicit some assumptions we need. In the second section we introduce the notion of viability kernel for a simple cycle and we show how to compute it. Controllability kernels are introduced in section 3 and a non-iterating algorithm for computing them for simple cycles is presented. In the fourth section we study some properties of controllability kernels. In section 5, using the result of the previous sections, we give an algorithm to build the phase portrait of SPDI and in section 6 we mention some work of how to compute limit cycles for PCDs. We finish with a last section that summarizes the results presented in this chapter and that revises the related work.

6.1 Preliminaries

In this section we recall some definitions through examples from the previous chapter and we explicit some assumptions we need in order to built the phase portrait of SPDI.

Let $F : \mathbb{R} \to 2^{\mathbb{R}}$ be an affine multi-valued function $F = \langle f_l, f_u \rangle$ and $F^{-1} = \langle f_u^{-1}, f_l^{-1} \rangle$ its inverse function. Remember that for an interval I, a truncated affine multi-valued function $\mathcal{F} : \mathbb{R} \to 2^{\mathbb{R}}$ is defined as $\mathcal{F}(I) = F(I \cap S) \cap J$. Let $\mathcal{H} = (\mathbb{P}, \mathbb{F})$ be a SPDI. Given a *trajectory segment* ξ of an SPDI we denote its *signature* by $\mathsf{Sig}(\xi)$. For each region $P \in \mathbb{P}$,

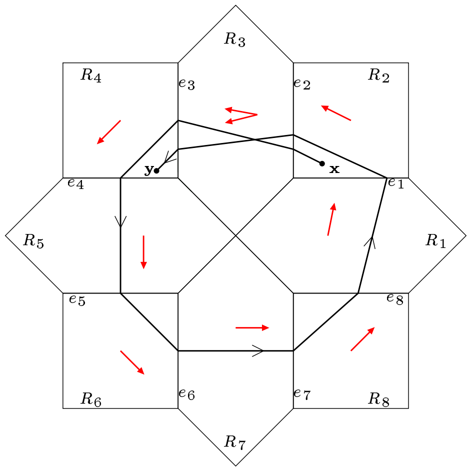


Figure 6.1: An SPDI and its trajectory segment.

let $In(P) \subseteq E(P)$ be the set of all entries of P and $Out(P) \subseteq E(P)$ be the set of all exits of P.

We have the following assumption.

Assumption 1 *All the edges in $E(P)$ are either entries or exits, that is, $E(P) = In(P) \cup Out(P)$.*

That is, we are going to consider just *good* regions.

Example 21 Consider the SPDI illustrated in Fig. 6.1. For each region R_i, $1 \leq i \leq 8$, there is a pair of vectors that are the same as the swimmer example on section 2.4 but where the regions are slightly different. ■

In order to built the phase portrait, we assume the following.

Assumption 2 *We will only consider trajectories with infinite signatures.*

Given an SPDI, we fix a one-dimensional coordinate system on each edge to represent points laying on edges (see section 5.1). For notational convenience, we indistinctly use letter e to denote the edge or its one-dimensional representation. Accordingly, we write $\mathbf{x} \in e$ or $x \in e$, to mean "point $\mathbf{x}$ in edge e with coordinate x in the one-dimensional coordinate system of e". The same convention is applied to sets of points of e represented as intervals (e.g., $\mathbf{x} \in I$ or $x \in I$, where $I \subseteq e$) and to trajectories (e.g., "ξ starting in x" or "ξ starting in $\mathbf{x}$").

Now, let $P \in \mathbb{P}$, $e \in In(P)$ and $e' \in Out(P)$. For $I \subseteq e$, remember that $\mathsf{Succ}_{ee'}(I)$ is the set of all points in e' reachable from some point in I by a trajectory segment $\xi : [0, t] \to \mathbb{R}^2$ in

P. We have shown (see Lemma 24) that $\mathsf{Succ}_{ee'}$ is a TAMF.

Example 22 Let $e_1, \ldots, e_8$ be as in Fig. 6.1 and $I = [l, u]$. We assume a one-dimensional coordinate system such that $e_i = S_i = J_i = (0, 1)$. We have that:

$$F_{e_1e_2}(I) = \left[\frac{l}{2}, \frac{u}{2}\right] \qquad F_{e_2e_3}(I) = \left[l - \frac{1}{4}, u + \frac{11}{60}\right]$$
$$F_{e_ie_{i+1}}(I) = I \quad 3 \leq i \leq 7 \qquad F_{e_8e_1}(I) = \left[l + \frac{1}{5}, u + \frac{1}{5}\right]$$

with $\mathsf{Succ}_{e_ie_{i+1}}(I) = F_{e_ie_{i+1}}(I \cap S_i) \cap J_{i+1}$, for $1 \leq i \leq 7$, and $\mathsf{Succ}_{e_8e_1}(I) = F_{e_8e_1}(I \cap S_8) \cap J_1$. ■

Given a sequence $\sigma = e_1, e_2, \ldots, e_n$, Lemma 4 implies that the successor of I along σ defined as $\mathsf{Succ}_\sigma(I) = \mathsf{Succ}_{e_{n-1}e_n} \circ \ldots \circ \mathsf{Succ}_{e_1e_2}(I)$ is a TAMF.

Example 23 Let $\sigma = e_1 \cdots e_8 e_1$. We have that $\mathsf{Succ}_\sigma(I) = F(I \cap S) \cap J$, where:

$$F(I) \quad = \quad \left[\frac{l}{2} - \frac{1}{20}, \frac{u}{2} + \frac{23}{60}\right] \tag{6.1}$$

$S = (0, 1)$ and $J = (\frac{1}{5}, 1)$ are computed using Lemma 4. ■

For $I \subseteq e'$, $\mathsf{Pre}_{ee'}(I)$ is the set of points in e that can reach a point in I by a trajectory segment in P. We can define the *predecessor operator* as

Definition 30 *The* edge-to-edge predecessor operator *is defined as*

$$\mathsf{Pre}_{ee'} = \mathsf{Succ}_{ee'}^{-1}$$

Given a signature σ, the σ-predecessor operator *is defined as*

$$\mathsf{Pre}_\sigma = \mathsf{Succ}_\sigma^{-1}.$$

Remark. We are not going to prove the fact that predecessors are TAMFs. It follows directly from Lemma 1.

Example 24 Let $\sigma = e_1 \ldots e_8 e_1$ be as in Fig. 6.1 and $I = [l, u]$. We have that $\mathsf{Pre}_{e_ie_{i+1}}(I) = F^{-1}_{e_ie_{i+1}}(I \cap J_{i+1}) \cap S_i$, for $1 \leq i \leq 7$, and $\mathsf{Pre}_{e_8e_1}(I) = F^{-1}_{e_8e_1}(I \cap J_1) \cap S_8$, where:

$$F^{-1}_{e_1e_2}(I) = [2l, 2u] \qquad F^{-1}_{e_2e_3}(I) = \left[l - \frac{11}{60}, u + \frac{1}{4}\right]$$
$$F^{-1}_{e_ie_{i+1}}(I) = I \quad 3 \leq i \leq 7 \qquad F^{-1}_{e_8e_1}(I) = \left[l - \frac{1}{5}, u - \frac{1}{5}\right]$$

Besides, $\mathsf{Pre}_\sigma(I) = F^{-1}(I \cap J) \cap S$, where $F^{-1}(I) = [2l - \frac{23}{30}, 2u + \frac{1}{10}]$. ■

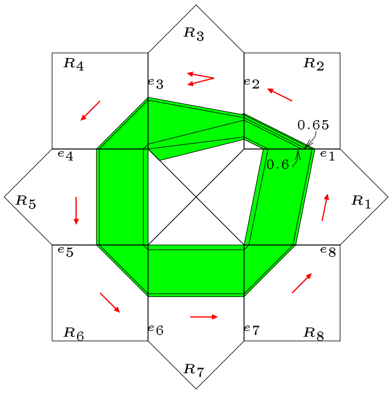


Figure 6.2: Reachability analysis.

Let $\sigma = e_1 \cdots e_k e_1$ be a simple edge-cycle, i.e., $e_i \neq e_j$ for all $1 \leq i \neq j \leq k$. Let $\mathsf{Succ}_\sigma(I) = F(I \cap S) \cap J$ with $F = \langle f_l, f_u \rangle$ (we suppose that this representation is normalized).

We denote by $\mathcal{D}_\sigma$ the one-dimensional discrete-time dynamical system defined by Succ_σ, that is $x_{n+1} \in \mathsf{Succ}_\sigma(x_n)$.

Assumption 3 *None of the two functions f_l, f_u is the identity.*

This assumption is needed in order to avoid having an infinite number of fixpoints. This will become clearer later on section 6.2.

Let l^* and u^* be the fixpoints of f_l and f_u, respectively, and $S \cap J = \langle L, U \rangle$. We have shown in section 5.2.1 that a simple cycle is of one of the following types: STAY, DIE, EXIT-BOTH, EXIT-LEFT or EXIT-RIGHT.

The classification above gives some information about the qualitative behavior of trajectories. Any trajectory that enters a cycle of type DIE will eventually quit it after a finite number of turns. If the cycle is of type STAY, all trajectories that happen to enter it will keep turning inside it forever. In all other cases, some trajectories will turn for a while and then exit, and others will continue turning forever. Remember that this information is very useful for solving the reachability problem (see section 5.2.1).

Example 25 Consider again the cycle $\sigma = e_1 \cdots e_8 e_1$. Fig. 6.2 shows part of the reach set of the interval $[0.6, 0.65] \subset e_1$. Notice that the leftmost trajectory exits the cycle in the third turn while the rightmost one shifts to the right and "converges to" the limit $u^* = \frac{23}{30}$. Clearly, no point in $[0.6, 0.65]$ will ever reach a point of e_1 smaller than $L = \frac{1}{5}$ or bigger than u^*. Fig. 6.2 has been automatically generated by the SPeeDI toolbox we have developed for reachability analysis of SPDIs (see chapter 8). ∎

The above result does not allow to directly answer other questions about the behavior of the

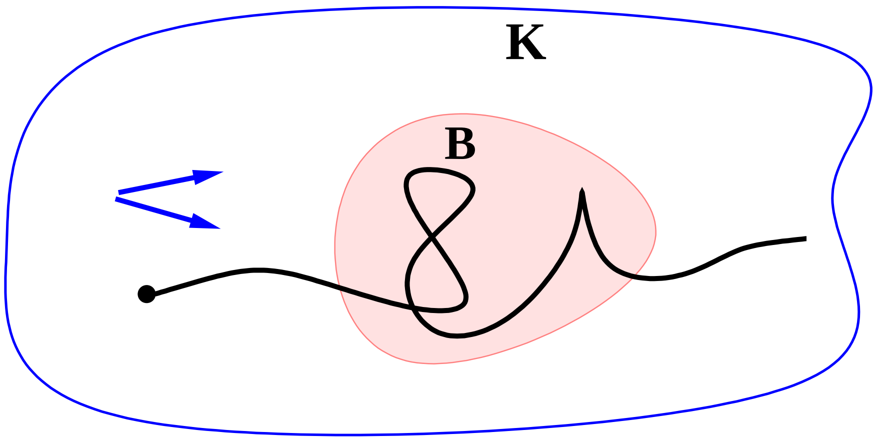


Figure 6.3: Example.

SPDI such as, determine for a given point (or set of points) whether: (a) there exists (at least) one trajectory that remains in the cycle, and (b) it is possible to control the system to reach any other point. In order to do this, we need to further study the properties of the system around simple edge-cycles.

6.2 Viability Kernel

In this section and the following two, we are going to concentrate on studying the qualitative behavior of sets of trajectories having the same cyclic pattern, that is we consider only cyclic signatures. We rely on the information given by the classification recalled in the previous section (STAY, DIE, etc. cycles) to more deeply study the qualitative behavior of the system. In this first part we introduce *viability kernel* [24, 27] and we show how to compute it.

In general, a *viability domain* is a set of points such that for any point in the set, there exists at least one trajectory that remains in the set forever and the *viability kernel* is the largest of such sets.

Example 26 In Figure 6.3, we can see that the set B divides the space into two regions: the part that is inside B and the rest ($K \setminus B$). The dynamics in B is given by a differential inclusion that allows the first derivative to be any value (i.e., $\angle_{\mathbf{a}}^{\mathbf{b}}$ is such that $\mathbf{a} = 0°$ and $\mathbf{b} = 360°$) whereas outside B, the dynamics is given by the two drawn vectors. Let us consider region A as in Figure 6.4. Notice that for any point in A, there is a trajectory segment to a point in B from where it can remain for ever in B. On the other hand, outside A (and outside B), for example points y and z, are not starting points of infinite trajectories. Then, the viability kernel is given by $A \cup B$. ■

In particular, for SPDI, given a cyclic signature, the *viability domain* is a set of points which

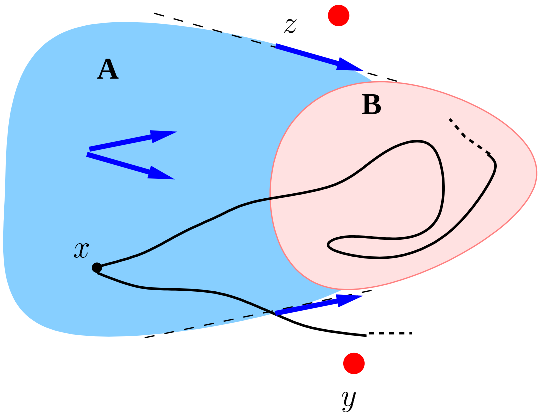


Figure 6.4: Viability kernel.

can keep rotating in the cycle forever and the *viability kernel* is the largest of such sets. We show that this kernel is a non-convex polygon (often with a hole in the middle) and we give a non-iterative algorithm for computing the coordinates of its vertices and edges.

In what follows, let $K \subset \mathbb{R}^2$.

Definition 31 *A trajectory ξ is* viable *in K if $\xi(t) \in K$ for all $t \geq 0$. K is a* viability domain *if for every $\mathbf{x} \in K$, there exists at least one trajectory ξ, with $\xi(0) = \mathbf{x}$, which is viable in K. The* viability kernel *of K, denoted $\mathsf{Viab}(K)$, is the largest viability domain contained in K.* ■

Remark. We do not define the viability kernel to be closed as in [24].

6.2.1 One dimensional discrete-time system

The same concepts can be defined for $\mathcal{D}_\sigma$, by setting that a trajectory $x_0 x_1 \ldots$ of $\mathcal{D}_\sigma$ is viable in an interval $I \subseteq \mathbb{R}$, if $x_i \in I$ for all $i \geq 0$.

Theorem 31 *For $\mathcal{D}_\sigma$, if σ is not DIE then* $\mathsf{Viab}(e_1) = S$, *else* $\mathsf{Viab}(e_1) = \emptyset$.[1]

Proof. If σ is DIE, $\mathcal{D}_\sigma$ has no trajectories. Therefore, $\mathsf{Viab}(e_1) = \emptyset$.
Let σ be not DIE. We first prove that any viability domain is a subset of S: Let I be a viability domain. Then, for all $x \in I$, there exists a trajectory starting in x which is viable in I. Then, $x \in \mathsf{Dom}(\mathsf{Succ}_\sigma) = S$. Thus, $I \subseteq S$.
Now, let us prove that S is a viability domain: It suffices to show that for all $x \in S$,

[1] Notice that this theorem can be used to compute $\mathsf{Viab}(I)$ for any $I \subseteq e_1$.

$\mathsf{Succ}_\sigma(x) \cap S \neq \emptyset$.
Let $x \in S$.
If σ is STAY, we have that both l^* and u^* belong to $S \cap J$. It follows that both $f_l(x)$ and $f_u(x)$ are in S.
If σ is EXIT-LEFT, we have that $l^* < S \cap J$ and $u^* \in S \cap J$. Then, $f_u(x) \in S$.
If σ is EXIT-RIGHT, we have that $l^* \in S \cap J$ and $u^* > S \cap J$. Then, $f_l(x) \in S$.
If σ is EXIT-BOTH, we have that $l^* < S \cap J$ and $u^* > S \cap J$. If $x \in J$: then $x \in F(x)$. If $x < J$: then $f_l(x) < x < S \cap J$, and either $f_u(x) \in S \cap J$ or $f_u(x) > S \cap J$ (the other case yields a contradiction). If $x > J$: similar to the previous case.
Thus, for all $x \in S$, $\mathsf{Succ}_\sigma(x) \cap S \neq \emptyset$.
Hence, $\mathsf{Viab}(e_1) = S$. $\Box$

The following lemma will be useful when proving some results about convergence in the next section.

Lemma 32 *For $\mathcal{D}_\sigma$, if the trace $x_1 x_2 \ldots$ of ξ is viable in S then $\forall n > 1 \,.\, x_n \in S \cap J$.*

Proof. By Theorem 31, $x_1 \in S$ and since $x_{n+1} \in \mathsf{Succ}_\sigma(x_n)$ we have that $x_n \in \mathsf{Dom}(\mathsf{Succ}_\sigma)$, i.e. $x_n \in S$. On the other hand, $x_n \in \mathsf{Succ}_\sigma(x_{n-1})$ that is included in $\mathsf{Im}(\mathsf{Succ}_\sigma)$, hence $x_n \in J$. $\Box$

6.2.2 Continuous-time system

The viability kernel for the continuous-time system can be now found by propagating S from e_1 using the following operator. The *extended predecessor* of an output edge e of a region R is the set of points in R such that there exists a trajectory segment that reaches e without traversing any other edge. More formally,

Definition 32 *Let R be a region and e be an edge in $Out(R)$. The e-*extended predecessor* of I, $\overline{\mathsf{Pre}}_e(I)$ is defined as:*

$$\overline{\mathsf{Pre}}_e(I) \quad = \quad \{\mathbf{x} \mid \exists \xi : [0,t] \to \mathbb{R}^2, t > 0 \,.\, \xi(0) = \mathbf{x} \wedge \xi(t) \in I \wedge \mathsf{Sig}(\xi) = e\}$$

■

The above notion can be extended to cyclic signatures (and so to edge-signatures) as follows. Let $\sigma = e_1, \ldots, e_k$ be a cyclic signature. For $I \subseteq e_1$, the *σ-extended predecessor* of I, $\overline{\mathsf{Pre}}_\sigma(I)$ is the set of all $\mathbf{x} \in \mathbb{R}^2$ for which there exists a trajectory segment ξ starting in $\mathbf{x}$, that reaches some point in I, such that $\mathsf{Sig}(\xi)$ is a suffix of $e_2 \ldots e_k e_1$.

It is easy to see that $\overline{\mathsf{Pre}}_\sigma(I)$ is a polygonal subset of the plane which can be calculated using the following procedure. First compute $\overline{\mathsf{Pre}}_{e_i}(I)$ for all $1 \leq i \leq n$ and then apply this operation k times:

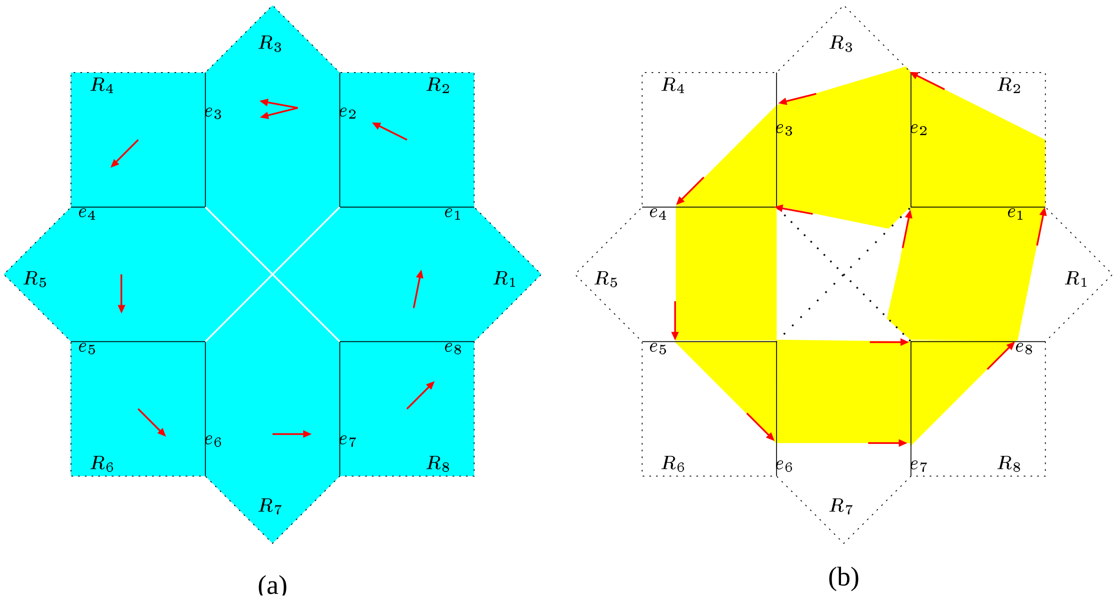


Figure 6.5: Viability kernel.

$$\overline{\mathsf{Pre}}_\sigma(I) = \bigcup_{i=1}^{k} \overline{\mathsf{Pre}}_{e_i}(I_i)$$

with $I_1 = I$, $I_k = \mathsf{Pre}_{e_k e_1}(I_1)$ and $I_i = \mathsf{Pre}_{e_i e_{i+1}}(I_{i+1})$, for $2 \leq i \leq k-1$.

Now, let define the following set:

$$K_\sigma \quad = \quad \bigcup_{i=1}^{k} (\mathtt{int}(P_i) \cup e_i) \tag{6.2}$$

where P_i is such that $e_{i-1} \in \mathit{In}(P_i)$, $e_i \in \mathit{Out}(P_i)$ and $\mathtt{int}(P_i)$ is the interior of P_i.
We can now compute the viability kernel of K_σ.

Theorem 33 *If σ is not DIE,* $\mathsf{Viab}(K_\sigma) = \overline{\mathsf{Pre}}_\sigma(S)$, *otherwise* $\mathsf{Viab}(K_\sigma) = \emptyset$.

Proof: If σ is DIE, trivially $\mathsf{Viab}(K_\sigma) = \emptyset$.
Let σ be not DIE. We first prove that any viability domain K, with $K \subseteq K_\sigma$, is a subset of $\overline{\mathsf{Pre}}_\sigma(S)$: Let $\mathbf{x} \in K$. Then, there exists a trajectory ξ such that $\xi(0) = \mathbf{x}$ and for all $t \geq 0$, $\xi(t) \in K$. Clearly, the sequence $x_1 x_2 \ldots$ of the intersections of ξ with e_1 is a trajectory of $\mathcal{D}_\sigma$. Then, by Theorem 31, $x_i \in S$ for all $i \geq 1$. Thus, $\mathbf{x} \in \overline{\mathsf{Pre}}_\sigma(S)$.
It remains to prove that $\overline{\mathsf{Pre}}_\sigma(S)$ is a viability domain. Let $\mathbf{x} \in \overline{\mathsf{Pre}}_\sigma(S)$. Then, there exists

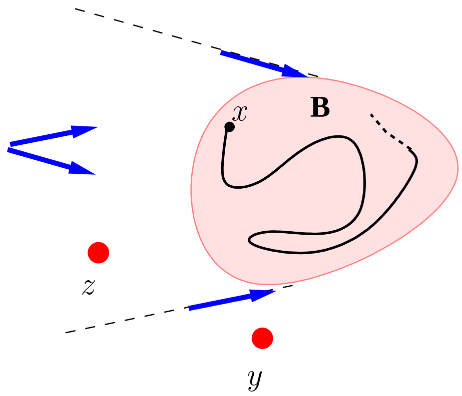


Figure 6.6: Controllability kernel.

a trajectory segment $\bar{\xi} : [0,T] \to \mathbb{R}^2$ such that $\bar{\xi}(T) \in S$ and $\mathsf{Sig}(\bar{\xi})$ is a suffix of σ. Theorem 31 implies that $\bar{\xi}(T)$ is the initial state of some trajectory ξ with $\mathsf{Sig}(\xi) = \sigma^\omega$. It is straightforward to show that for all $t \geq 0$, $\xi(t) \in \overline{\mathsf{Pre}}_\sigma(S)$. Concatenating $\bar{\xi}$ and ξ, we obtain a viable trajectory starting in $\mathbf{x}$.
Hence, $\mathsf{Viab}(K_\sigma) = \overline{\mathsf{Pre}}_\sigma(S)$. □

This result provides a non-iterative algorithmic procedure for computing the viability kernel of K_σ.

Example 27 Let $\sigma = e_1 \ldots e_8 e_1$. Fig. 6.5 depicts: (a) K_σ, and (b) $\overline{\mathsf{Pre}}_\sigma(S)$. ■

6.3 Controllability Kernel

In this section we define and we show how to compute the *controllability kernel* of a simple cycle.

We say $K \subset \mathbb{R}^2$ is *controllable* if for any two points $\mathbf{x}$ and $\mathbf{y}$ in K there exists a trajectory segment ξ starting in $\mathbf{x}$ that reaches an arbitrarily small neighborhood of $\mathbf{y}$ without leaving K.

Example 28 Let consider the same example as for viability domains (see Figure 6.3): the set B divides the space into two regions: the part inside B and the rest. The dynamics in B is given by a differential inclusion that allows the first derivative to be any value (i.e., $\angle_{\mathbf{a}}^{\mathbf{b}}$ is such that $\mathbf{a} = 0°$ and $\mathbf{b} = 360°$) whereas outside B, the dynamics is given by the two drawn vectors. Notice that any point x in B is the starting point of a trajectory that reach any other point in B as shown in Figure 6.6. Outside B points are not reachable one from the other, y is reachable from x but not vice-versa, for instance. Then, B is the controllability kernel. ■

For SPDIs and considering cyclic signatures, the controllability kernel is a cyclic polygonal stripe within which a trajectory can reach any point from any point. More formally,

Definition 33 *K is* controllable *iff* $\forall \mathbf{x}, \mathbf{y} \in K, \forall \delta > 0, \exists \xi : [0,t] \to \mathbb{R}^2, t > 0 \,.\, (\xi(0) = \mathbf{x} \wedge |\xi(t) - \mathbf{y}| < \delta \wedge \forall t' \in [0,t] \,.\, \xi(t') \in K)$. *The* controllability kernel *of K, denoted* $\mathsf{Cntr}(K)$, *is the largest controllable subset of K.* ■

6.3.1 One dimensional discrete-time system

The same notions can be defined for the discrete dynamical system $\mathcal{D}_\sigma$. Define

$$\mathcal{C}_{\mathcal{D}}(\sigma) = \begin{cases} \langle L, U \rangle & \text{if } \sigma \text{ is EXIT-BOTH} \\ \langle L, u^* \rangle & \text{if } \sigma \text{ is EXIT-LEFT} \\ \langle l^*, U \rangle & \text{if } \sigma \text{ is EXIT-RIGHT} \\ \langle l^*, u^* \rangle & \text{if } \sigma \text{ is STAY} \\ \emptyset & \text{if } \sigma \text{ is DIE} \end{cases} \tag{6.3}$$

Theorem 34 *For $\mathcal{D}_\sigma$, $\mathcal{C}_{\mathcal{D}}(\sigma) = \mathsf{Cntr}(S)$.*

Proof. Controllability of $\mathcal{C}_{\mathcal{D}}(\sigma)$ follows from the reachability result given in the previous chapter. To prove that $\mathcal{C}_{\mathcal{D}}(\sigma)$ is maximal we reason by contradiction. Suppose it is not. Then, there should exist a controllable set $C \supset \mathcal{C}_{\mathcal{D}}(\sigma)$. Since $C \subseteq S \cap J$, there should exist $y \in C$ such that either $y < l^*$, or $y > u^*$. In any case, controllability implies that for all $l^* < x < u^*$, there exists a trajectory segment starting in x that reaches an arbitrarily small neighborhood of y. From the reachability algorithm given in section 5.2.1 we know that $Reach(x) \subset (l^*, u^*)$, which yields a contradiction. Hence, $\mathcal{C}_{\mathcal{D}}(\sigma)$ is the controllability kernel of S. □

6.3.2 Continuous-time system

For $I \subseteq e_1$ let us define $\overline{\mathsf{Succ}}_\sigma(I)$ as the set of all points $\mathbf{y} \in \mathbb{R}^2$ for which there exists a trajectory segment ξ starting in some point $x \in I$, that reaches $\mathbf{y}$, such that $\mathsf{Sig}(\xi)$ is a prefix of $e_1 \ldots e_k$. The successor $\overline{\mathsf{Succ}}_\sigma(I)$ is a polygonal subset of the plane which can be computed similarly to $\overline{\mathsf{Pre}}_\sigma(I)$, that is,

Definition 34 *Let R be a region and e be an edge in $In(R)$. The e-*extended successor *of I, $\overline{\mathsf{Succ}}_e(I)$ is defined as:*

$$\overline{\mathsf{Succ}}_e(I) \quad = \quad \{\mathbf{x} \mid \exists \xi, \mathbf{y} \in I, t > 0 \,.\, \xi(0) = \mathbf{y} \wedge \xi(t) = \mathbf{x} \wedge \mathsf{Sig}(\xi) = e\}$$

■

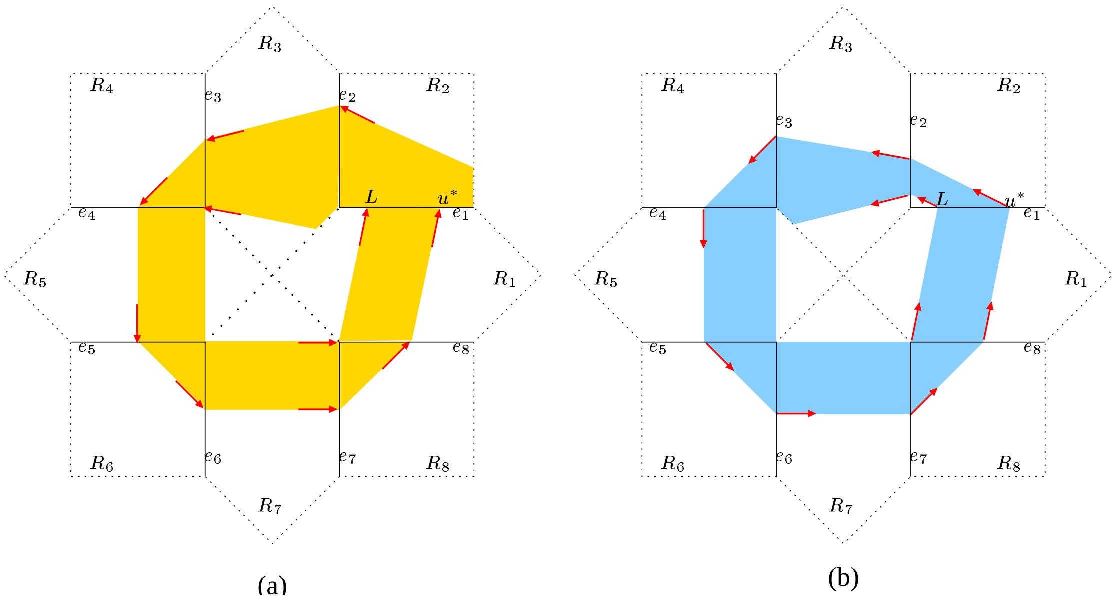


Figure 6.7: Predecessors and successors of a simple cycle.

The extended successors for cyclic signatures (and for edge-signatures) can be defined as follows. Let $\sigma = e_1, \ldots, e_k$ be a cyclic signature. For $I \subseteq e_1$, the σ-*extended successor* of I, $\overline{\mathsf{Succ}}_\sigma(I)$ is the set of all reachable points $\mathbf{x} \in \mathbb{R}^2$ via a trajectory segment ξ starting in $\mathbf{y} \in e_1$, such that $\mathsf{Sig}(\xi)$ is a prefix of $e_1 \ldots e_k$.

As for extended predecessors, $\overline{\mathsf{Succ}}_\sigma(I)$ is a polygonal subset of the plane which can be calculated using the following procedure. First compute $\overline{\mathsf{Succ}}_{e_i}(I)$ for all $1 \leq i \leq n$ and then apply this operation k times:

$$\overline{\mathsf{Succ}}_\sigma(I) = \bigcup_{i=1}^{k} \overline{\mathsf{Succ}}_{e_i}(I_i)$$

where $I_1 = I$ and $I_{i+1} = \mathsf{Succ}_{e_i e_{i+1}}(I)$ for $1 \leq i \leq k-1$.

Example 29 Let $\sigma = e_1 \cdots e_8 e_1$. Fig. 6.7 depicts: (a) $\overline{\mathsf{Pre}}_\sigma(L, u^*)$, (b) $\overline{\mathsf{Succ}}_\sigma(L, u^*)$, with $L = \frac{1}{5} < u^* = \frac{23}{30}$. ■

Define

$$\mathcal{C}(\sigma) \quad = \quad (\overline{\mathsf{Succ}}_\sigma \cap \overline{\mathsf{Pre}}_\sigma)(\mathcal{C}_\mathcal{D}(\sigma)) \tag{6.4}$$

We show in the following theorem how to compute controllability kernels.

Theorem 35 *$\mathcal{C}(\sigma) = \mathsf{Cntr}(K_\sigma)$.*

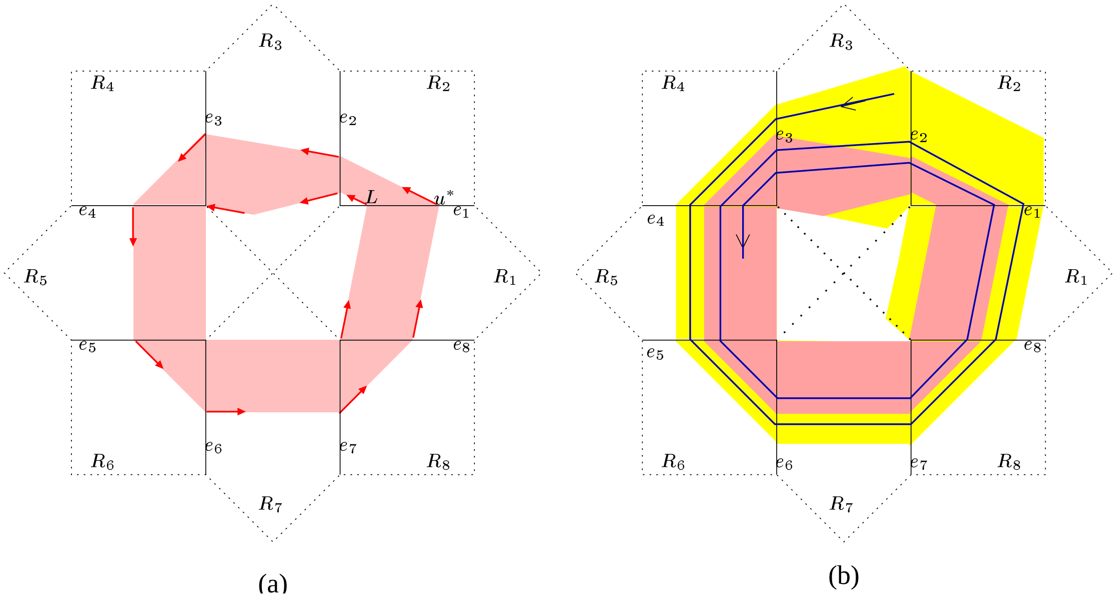


Figure 6.8: Controllability kernel of a simple cycle.

Proof. Let $\mathbf{x}, \mathbf{y} \in \mathcal{C}(\sigma)$. Since $\mathbf{y} \in \overline{\mathsf{Succ}}_\sigma(\mathcal{C}_\mathcal{D}(\sigma))$, there exists a trajectory segment starting in some point $w \in \mathcal{C}_\mathcal{D}(\sigma)$ and ending in $\mathbf{y}$. Let ϵ be an arbitrarily small number and $B_\epsilon(\mathbf{y})$ be the set of all points $\mathbf{y}'$ such that $|\mathbf{y} - \mathbf{y}'| < \epsilon$. Clearly, $w \in \overline{\mathsf{Pre}}_\sigma(B_\epsilon(\mathbf{y})) \cap \mathcal{C}_\mathcal{D}(\sigma)$. Now, since $\mathbf{x} \in \overline{\mathsf{Pre}}_\sigma(\mathcal{C}_\mathcal{D}(\sigma))$, there exists a trajectory segment starting in $\mathbf{x}$ and ending in some point $z \in \mathcal{C}_\mathcal{D}(\sigma)$. Since $\mathcal{C}_\mathcal{D}(\sigma)$ is controllable, there exists a trajectory segment starting in z that reaches a point in $\overline{\mathsf{Pre}}_\sigma(B_\epsilon(\mathbf{y})) \cap \mathcal{C}_\mathcal{D}(\sigma)$. Thus, there is a trajectory segment that starts in $\mathbf{x}$ and ends in $B_\epsilon(\mathbf{y})$. Therefore, $\mathcal{C}(\sigma)$ is controllable. Maximality follows from the maximality of $\mathcal{C}_\mathcal{D}(\sigma)$ (Theorem 34) and the definition of $\overline{\mathsf{Succ}}_\sigma$ and $\overline{\mathsf{Pre}}_\sigma$. Hence, $\mathcal{C}(\sigma)$ is the controllability kernel of K_σ. □

This result provides a non-iterative algorithmic procedure for computing the controllability kernel of K_σ.

Example 30 Let $\sigma = e_1 \cdots e_8 e_1$. Recall that σ is EXIT-LEFT with $L = \frac{1}{5} < u^* = \frac{23}{30}$. Fig. 6.8(a) depicts $\mathsf{Cntr}(K_\sigma)$. ■

6.4 Properties of Controllability Kernels

In this section we present some properties of controllability kernels, regarding convergence and its relation to fixpoints in general. In particular, for STAY cycles we have stronger limit cycle properties.

6.4.1 Convergence

Definition 35 *A trajectory ξ* converges *to a set $K \subset \mathbb{R}^2$ if* $\lim_{t\to\infty} \mathrm{dist}(\xi(t), K) = 0$. ■

For $\mathcal{D}_\sigma$, convergence is defined as $\lim_{n\to\infty} \mathrm{dist}(\xi_n, I) = 0$. The following result says that the controllability kernel $\mathcal{C}_\mathcal{D}(\sigma)$ can be considered to be a kind of (weak) limit cycle of $\mathcal{D}_\sigma$.

Theorem 36 *For $\mathcal{D}_\sigma$, any viable trajectory in S converges to $\mathcal{C}_\mathcal{D}(\sigma)$.*

Proof. Let $x_1 x_2 \ldots$ a viable trajectory. By Lemma 32, $x_i \in S \cap J$ for all $i \geq 2$. Recall that $\mathcal{C}_\mathcal{D}(\sigma) \subseteq S \cap J$. There are three cases: (1) There exists $N \geq 2$ such that $x_N \in \mathcal{C}_\mathcal{D}(\sigma)$. Then, for all $n \geq N$, $x_n \in \mathcal{C}_\mathcal{D}(\sigma)$. (2) For all n, $x_n < \mathcal{C}_\mathcal{D}(\sigma)$. Therefore, $x_n < l^*$. Let $\hat{x}_n$ be such that $\hat{x}_1 = x_1$ and for all $n \geq 1$, $\hat{x}_{n+1} = f_l(\hat{x}_n)$. Clearly, for all n, $\hat{x}_n \leq x_n < l^*$, and $\lim_{n\to\infty} \hat{x}_n = l^*$, which implies $\lim_{n\to\infty} x_n = l^*$. (3) For all n, $x_n > \mathcal{C}_\mathcal{D}(\sigma)$. Therefore, $u^* < x_n$. Let $\hat{x}_n$ be such that $\hat{x}_1 = x_1$ and for all $n \geq 1$, $\hat{x}_{n+1} = f_u(\hat{x}_n)$. Clearly, for all n, $u^* < x_n \leq \hat{x}_n$, and $\lim_{n\to\infty} \hat{x}_n = u^*$, which implies $\lim_{n\to\infty} x_n = u^*$. Hence, $x_1 x_2 \ldots$ converges to $\mathcal{C}(\sigma)$. □

Furthermore, $\mathcal{C}(\sigma)$ can be regarded as a (weak) limit cycle of the SPDI. The following result is a direct consequence of Theorem 33 and Theorem 36.

Theorem 37 *Any viable trajectory in K_σ converges to $\mathcal{C}(\sigma)$.* □

Example 31 Fig. 6.8(b) shows a trajectory with signature $\sigma = e_1 \cdots e_8 e_1$ which is viable in K_σ and converges to $\mathcal{C}(\sigma)$. ■

6.4.2 STAY cycles

The controllability kernels of STAY-cycles have stronger limit cycle properties. We define the notion of *invariant.*

Definition 36 *We say that K is* invariant *if for any $x \in K$, every trajectory starting in x is viable in K.* ■

The following result is a corollary of the previous theorems.

Theorem 38 *Let σ be STAY. Then,*
(1) $\mathcal{C}(\sigma)$ is invariant.
(2) There exists a neighborhood K of $\mathcal{C}(\sigma)$ such that any viable trajectory starting in K converges to $\mathcal{C}(\sigma)$.

Proof.

1. Suppose that $\mathcal{C}(\sigma)$ is not invariant, then it exists $x \in \mathcal{C}(\sigma)$ and a trajectory ξ starting on x (i.e. $x = \xi(0)$) s.t. ξ is not viable. By definition of $\mathcal{C}(\sigma)$, exists $x' \in \langle l^*, u^* \rangle$ and $t \geq 0$ such that $x' = \xi(t)$. On the other hand, by our assumption of non invariance, it exists $T > t$ such that $\xi(T) \notin \mathcal{C}(\sigma)$, that means $\xi(T) \notin \overline{\mathsf{Pre}}(l^*, u^*)$ and then x' has a successor not in $\langle l^*, u^* \rangle$, contradicting the hypothesis that σ is STAY. Hence $\mathcal{C}(\sigma)$ must be invariant;

2. It follows directly from Theorem 37.□

6.4.3 Fixpoints

Here we give an alternative characterization of the controllability kernel of a cycle in SPDI. As in [95], we define *fixpoints* and *periodic points.*

Definition 37 *A point x in e_1 a* fixpoint *iff $x \in \mathsf{Succ}_\sigma(x)$. We call a point $\mathbf{x} \in K_\sigma$ a* periodic point *iff there exists a trajectory segment ξ starting and ending in $\mathbf{x}$, such that $\mathsf{Sig}(\xi)$ is a cyclic shift of σ.* ■

If $\mathbf{x} \in K_\sigma$ is a periodic point then there exists also an infinite periodic trajectory passing through some $x \in e_1$. The following result characterizes the set of fixpoints and of periodic points for SPDIs.

Theorem 39 *For SPDIs,*
(1) $\mathcal{C}_\mathcal{D}(\sigma)$ is the set of all the fixpoints in e_1.
(2) $\mathcal{C}(\sigma)$ is the set of all the periodic points in K_σ.

Proof.

1. Let $\sigma = e_1, \cdots e_k$ be a cycle signature, $\langle L, U \rangle = S \cap J$ as before and x a fixpoint of e_1.
If σ is DIE, trivial.
If σ is STAY, any fixpoint of e_1 must be in $\langle l^*, u^* \rangle$, hence $x \in \langle l^*, u^* \rangle$.
If σ is EXIT-BOTH, notice that if x is a fixpoint in e_1, then it exists a viable trajectory ξ starting on x such that for all $n > 1$, $x_n = x$, but by Lemma 32, $x_n = S \cap J$, i.e. any fixpoint of e_1 must be in $S \cap J$.
If σ is EXIT-LEFT, from the above results any fixpoint must be in $\langle L, U \rangle \cap \langle l^*, u^* \rangle$, hence $x \in \langle L, u^* \rangle$.
If σ is EXIT-RIGHT, as for EXIT-LEFT, we obtain that $x \in \langle l^*, U \rangle$.

2. Let $\mathbf{x} \in K_\sigma$ be a periodic point, then any trajectory starting on $\mathbf{x}$ must intersect e_1 in a point x that is a fixpoint, but by 1., $x \in \mathcal{C}_\mathcal{D}(\sigma)$, then $\mathbf{x} \in \overline{\mathsf{Pre}}(x)$ that implies $\mathbf{x} \in \mathcal{C}(\sigma)$. □

As a direct consequence of the above theorem, the following result holds.

Corollary 40 *Given a cyclic signature $\sigma = e_1, \cdots e_k$, all the fixpoints in e_1 are included in $\langle L, U\rangle \cap \langle l^*, u^*\rangle$.* □

6.5 Phase Portrait

Let ξ be any trajectory without self-crossings. Recall that ξ is assumed to have an infinite signature. An immediate consequence of Lemma 20 (see section 4.5) is that $\mathsf{Sig}(\xi)$ can be canonically expressed as a sequence of edges and cycles of the form $r_1 s_1^* \ldots r_n s_n^\omega$, with (among others) the following properties:

1. For all $1 \leq i \leq n$, r_i is a sequence of pairwise different edges, and s_i is a simple cycle.
2. For all $1 \leq i \neq j \leq n$, r_i and r_j are disjoint, and s_i and s_j are different.
3. For all $1 \leq i \leq n-1$, s_i is repeated a finite number of times.
4. s_n is repeated forever.

Hence,

Theorem 41 *Every trajectory with infinite signature which does not have self-crossings converges to the controllability kernel of some simple edge-cycle.*

Proof. It follows directly from the above properties and from Theorem 37. □

We define now the notions of *limit set* and *limit points* of a given trajectory.

Definition 38 *Given a trajectory ξ such that $\xi(0) = \mathbf{x}$, a point $\mathbf{y}$ is a* limit point *of $\mathbf{x}$ if* $\lim_{t\to\infty} \xi(t) = \mathbf{y}$. *The set of all the limits points of $\mathbf{x}$ is its* limit set, $\text{limit}(\xi)$. ■

Corollary 42
1. *Any trajectory ξ with infinite signature without self-crossings is such that its limit set* $\text{limit}(\xi)$ *is a subset of the controllability kernel $\mathcal{C}(\sigma)$ of a simple edge-cycle σ.*
2. *Any point in $\mathcal{C}(\sigma)$ is a limit point of a trajectory ξ with infinite signature without self-crossings*

Proof. The result is a direct consequence of Theorem 41. □

We conclude that controllability kernels are important elements of the phase portrait of an SPDI yielding an analog of Poincaré-Bendixson theorem (see for example [83]) for simple

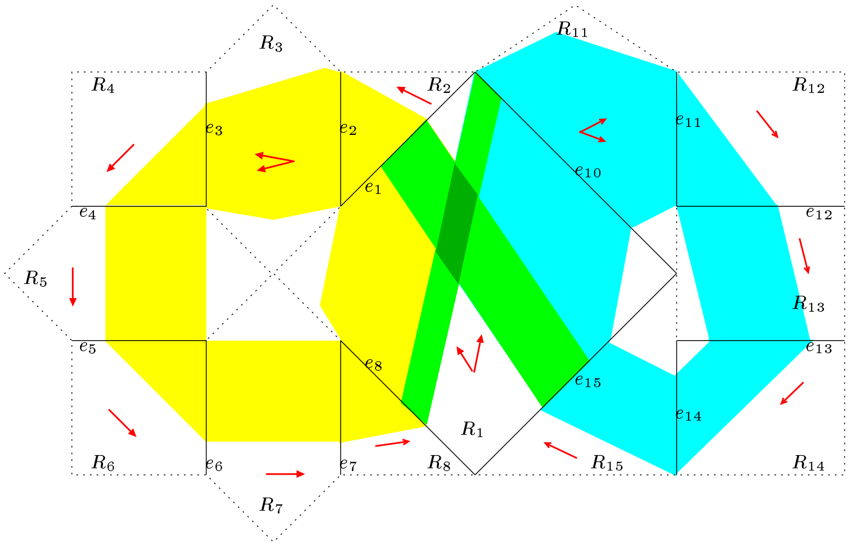


Figure 6.9: Another SPDI and its "phase-portrait".

trajectories. Moreover, all such components of the phase portrait can be algorithmically constructed. Indeed, since there are finitely many simple cycles, the following algorithm computes all the limit sets and their attraction basins for such kind of trajectories:

$$\textbf{for each } \text{simple cycle } \sigma \textbf{ compute } \mathcal{C}(\sigma),\ \overline{\mathsf{Pre}_\sigma}(S)$$

Example 32 Fig. 6.9 shows an SPDI with two edge cycles $\sigma_1 = e_1, \cdots, e_8, e_1$ and $\sigma_2 = e_{10}, \cdots, e_{15}, e_{10}$, and their respective controllability kernels. Every simple trajectory eventually arrives (or converges) to one of the two limit sets and rotates therein forever. ■

6.6 Limit Cycles for PCD

Limit cycles are very important in control theory as we have seen. In this section we describe how to count (determine) limit cycles of PCD systems using our reachability algorithm, that is a much easier task than for SPDI.

We have already seen that there exists a finite number of type of signatures and of course there is also a finite number of simple cycles. The first thing to do is to list all the feasible simple cycles. Then for each simple cycle $e_1, \cdots, e_n$ we compute the fixpoint for each edge as described in the reachability algorithm. That gives a periodic orbit. In order to know whether this periodic orbit is already a limit cycle we analyse the Poincaré map of the cycle (that is an affine function $y = ax + b$ with $a > 0$): if $a < 1$ then the periodic orbit is a (stable) limit cycle; if $a > 1$ then the periodic orbit is an unstable limit cycle; if $a = 1$ there

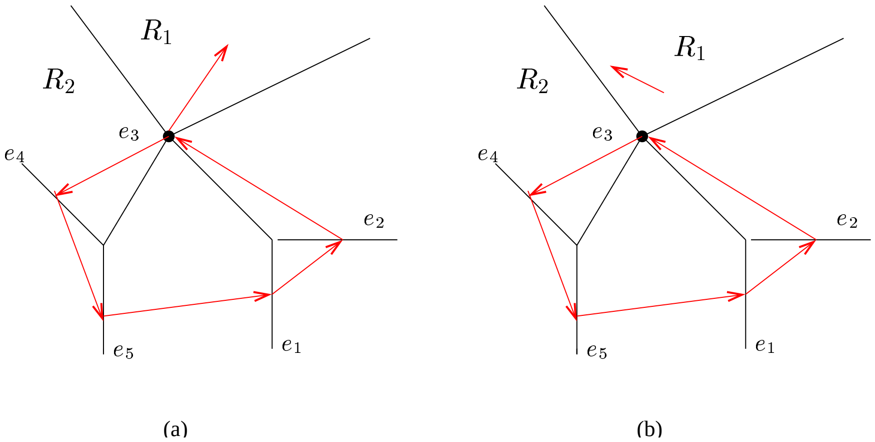


Figure 6.10: (a): Edge (vertex) e_3 is of input to regions R_1 and R_2; (b): Edge (vertex) e_3 is of input just of R_2.

is no limit cycle. See Fig. 3.1 and proof of Lemma 6.

The above procedure works well for the general case but not when one (ore more) of the edges on the simple cycle is a vertex. In this case we have to take into account a possible non-determinism due to the fact that the vertex can be an input edge for more than one region (see Fig. 6.10). Hence we have the following lemma.

Lemma 43 *Given a simple cycle $e_1, \cdots, e_i, \cdots, e_n$ where e_i is a vertex, then we have that e_i is an input edge exactly for one region if and only if $e_1, \cdots, e_i, \cdots, e_n$ is a periodic orbit. Moreover, this periodic orbit is not a limit cycle.*

Proof: Suppose that e_i is an input edge exactly for one region, that means that the trajectory is deterministic and clearly $e_1, \cdots, e_i, \cdots, e_n$ is a periodic orbit (by definition of simple cycle). On the other hand, if e_i is an input edge for more than one region then once in e_i we can decide to leave the simple cycle at any time, that implies that the cycle is not a periodic orbit. That the periodic orbit is not a limit cycle follows directly from the fact that there is no neighborhood for e_i. $\square$

Then, the following algorithm compute all the limit cycles for PCDs:

for each simple cycle σ **compute** its fixpoint.

See [109] for a deeper treatment of limit cycles and of phase portrait construction for PCD.

6.7 Summary and Related Work

The reachability question has been an important and extensively studied research problem in the hybrid systems community, however, there have been very few results on the qualitative properties of trajectories of hybrid systems [26, 65, 95, 96, 109, 133]. In particular, the question of defining and constructing phase portraits of hybrid systems has not been directly addressed except in [109], where phase portraits of deterministic systems with piecewise constant derivatives are explored and more recently in [30] a characterisation of viability and invariance kernels were given for implusive differential inclusions based on [28, 29].

The contribution of this chapter was an automatic procedure to analyze the qualitative behavior of non-deterministic planar hybrid systems. Our algorithm enumerates all the "limit cycles" (i.e., controllability kernels) and their local basins of attraction (i.e., viability kernels [24, 27]). We have also shown that the distance between any infinite trajectory performing forever the same cyclic pattern and the controllability kernel always converges to zero.

Our analysis technique for a single cycle is very similar to the one used in [95] for n-dimensional systems. However, for polygonal systems, we are able to prove further properties such as controllability of and convergence to the set of fixpoints, and that there are only a finite number of them. These results are the analog of Poincaré-Bendixson for polygonal differential inclusions. The difference with [109] is that our results hold for non-deterministic systems.

Chapter 7

(Un)decidability Results for Extensions of PCD

In this chapter we discuss some issues related to the (un)decidability of lower dimensional systems (for planar and 3-dim systems). The models considered here are hierarchical PCDs (HPCDs) that are hybrid automata such that at each location the dynamics is given by a PCD, PCDs on 2-dimensional manifolds, (restricted) 2-dimensional rectangular automata (RA) and linear hybrid automata (LA), HPCDs with one counter, HPCDs with infinite partition and rate dependent initial state HPCDs. On the negative side, we show that the reachability problem for the last three classes of systems is undecidable proving that the halting problem for Turing machines can be reduced to them. On the other hand, we show that we cannot assert whether the reachability problem for the other models is decidable or not since this problem is equivalent to a (not so) well known open problem: is the reachability question decidable for piecewise affine maps (PAMs)? PAMs are one dimensional systems defined by a (real-valued) affine function $f(x) = a_i x + b_i$ if $x \in I_i$ (for $1 \leq i \leq k$). This is an open problem even whenever the parameters of the map as well as the extremities of the intervals are rational. A preliminary version of the content of this chapter will appear in [21].

Organization of the chapter: In the first section we recall the definition of two dimensional manifolds and of our two reference models: Turing machines (TMs) and piecewise affine maps (PAMs). In the second section we introduce hierarchical PCDs (HPCDs) and PCDs on 2-dimensional manifolds ($\mathrm{PCD_{2m}}$) and we show that the reachability problem for HPCDs, $\mathrm{PCD_{2m}}$, RA and LA (the last two with some restrictions) is as hard as the reachability for PAMs. In the following section we show that enriching HPCDs with one counter leads to the undecidability of the reachability question as well as for others generalization of HPCDs, namely adding to it infinite partition or having origin-dependent rate dynamics. We conclude in the last section with a summary and related work.

7.1 Preliminaries

In this section we introduce two dimensional manifolds and we recall the definitions of our two reference models: Turing machine and piecewise affine maps. We also discuss briefly about the notion of *simulation*.

See section 2.2 for a definition of linear hybrid and rectangular automata and section 2.3.1 for the definition of PCDs.

7.1.1 Two dimensional manifolds

All the (topological) definitions, examples and results of this section follow the *combinatorial method* [78].

Let $\mathcal{P}$ be a collection of polygons, and $e_1, \ldots, e_n$ be a set of edges from these polygons.

Definition 39 *We say that the edges are* identified *when a new topology is defined on $\mathcal{P}$ as follows: (1) each edge is assigned a direction from one endpoint to the other and placed in topological correspondence with the unit interval in such a way that the initial points of all edges correspond to 0 and the final points correspond to 1; (2) the points on the edges $e_1, \ldots, e_n$ that all correspond to the same value from the unit interval are treated as a single point; (3) the neighborhoods of the new topology on $\mathcal{P}$ are the disks entirely contained in a single polygon plus the unions of half disks whose diameters are matching intervals around corresponding points on the edges $e_1, \ldots, e_n$.* ■

Definition 40 *Given a collection of vertices $v_1, \ldots, v_n$ of $\mathcal{P}$ we say that they are* identified *when a new topology is defined on $\mathcal{P}$ in which this collection of vertices is treated as a single point and the neighborhoods are defined to be disks completely contained in a single polygon plus the unions of portions of disks around each of the points $v_1, \ldots, v_n$.* ■

In any case, any of the edges meeting at one of these vertices is also identified, the sectors forming a neighborhood at the "point" $\{v_1, \ldots, v_n\}$ must contain matching intervals from these edges. See Figure 7.2.

Definition 41 *A topological space is* triangulable *if it can be obtained from a set of triangles by the identification of edges and vertices subject to the restriction that any two triangles are identified either along a single edge or at a single vertex, or are completely disjoint.* ■

Definition 42 *A* surface *(or* 2-dimensional manifold*) is a triangulable space for which in addition:*

1. *each edge is identified with exactly one other edge; and*
2. *the triangles identified at each vertex can always be arranged in a cycle $T_1, \ldots, T_k, T_1$ so that adjacent triangles are identified along an edge.* ■

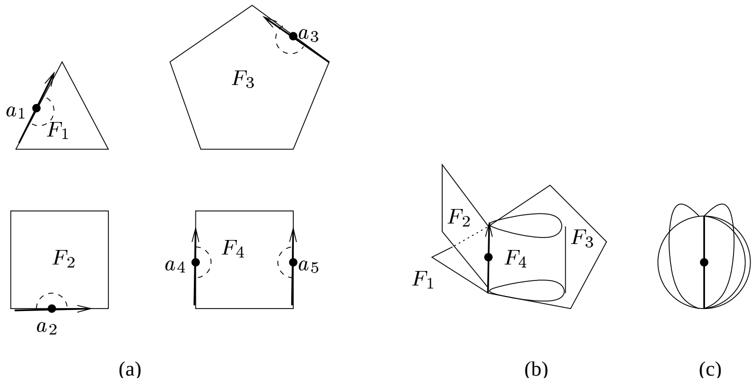


Figure 7.1: Identification at edges. (a) Four polygons identified at five edges; (b) The identification on the space; (c) A typical edge-neighborhood.

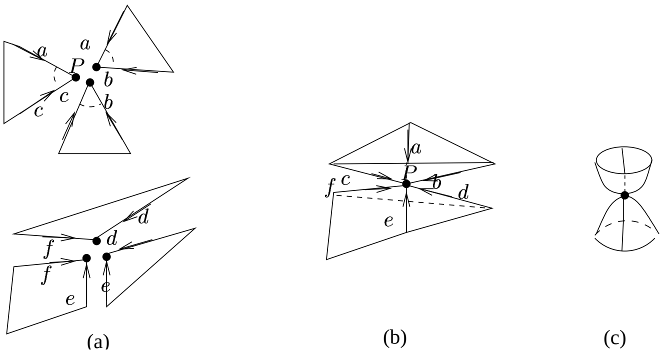


Figure 7.2: Identification at vertices. (a) Six polygons identified at a vertex; (b) The identification on the space; (c) A neighborhood of the vertex.

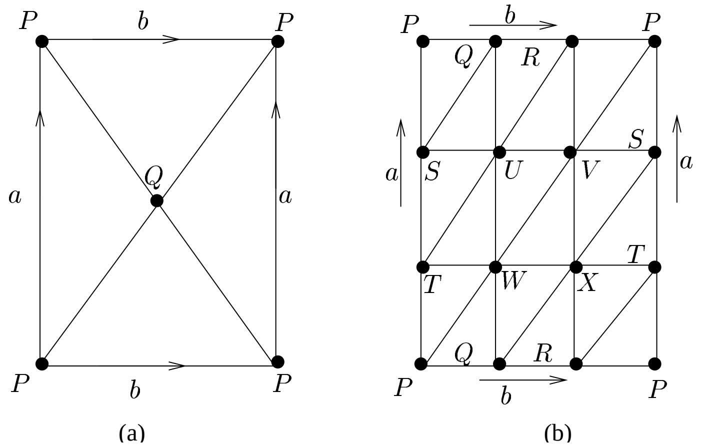


Figure 7.3: Triangulations for a Torus: (a) A false triangulation; (b) A good triangulation.

We say that a triangulation of a topological space is *good* if it satisfies the above definition. See Figure 7.3.

We state now, without proving it (see [78], p.122) an important theorem in the topological theory of surfaces:

Theorem 44 (Classification theorem) *Every compact, connected surface is topologically equivalent to a sphere, or a connected sum of tori, or a connected sum of projective planes.* □

The classification theorem for compact surfaces does not extend to non-compact surfaces. In fact, there are many interesting non-compact surfaces and that leads to the following definition.

Definition 43 *A* surface with boundary *is a topological space obtained by identifying edges and vertices of a set of triangles according to all the requirements of the definition of surface except that certain edges may not be identified with another edge. These edges, which violate the definition of a surface, are called* boundary edges*, and their vertices, which also violate the definition of surface, are called* boundary vertices*.* ■

Intuitively, a surface with boundary is a topological space in which every point either has a neighborhood equivalent to a disk or has a neighborhood equivalent to a half disk. Typical examples of surfaces with boundary are the cylinder and the Möbius strip. Indeed, the cylinder is equivalent to a sphere with two disks cut out. For this kind of surfaces there is also a classification theorem:

Theorem 45 (Classification theorem for Surfaces with Boundary) *Every compact,* *connected surface with boundary is equivalent to either a sphere, or a connected sum of tori, or a connected sum of projective planes, in any case with some finite number of disks removed.* □

We introduce now the last two models, that are important since they are our "reference" models: Turing machines and piecewise affine maps (PAMs).

7.1.2 Turing Machines

We define (for completeness) a well known computing device, called Turing machine.

Definition 44 *A* Turing machine *[119] is a quadruple $\mathcal{M} = (Q, \Sigma, \delta, s_0)$ where Q is a finite set of* states*, $s_0 \in Q$ is the* initial state*, Σ is a finite set of* symbols *called the* alphabet *of $\mathcal{M}$. We assume that Q and Σ are disjoint sets and that Σ contains a special symbol $\not b$ that is the* blank *symbol. Finally, δ is the* transition function *which maps $Q \times \Sigma$ to $(Q \cup \{\mathsf{Halt}\}) \times \Sigma \times \{R, L, -\}$. We assume that the* halting state Halt *and the cursor directions R (for "right"), L (for "left") and $-$ (for "stay"), are not in $Q \cup \Sigma$.* ■

Notice that δ defined as $\delta(q, s) = (q', s', D)$, is the "program" of the Turing machine, specifying, for each combination of current state $q \in Q$ and current symbol $s \in \Sigma$, the next state $q' \in Q$, the new symbol $s' \in \Sigma$ and the direction $D \in \{R, L, -\}$ in which the *head* will move. At each moment, the Turing machine is in a given state q, with the head reading the current symbol s. A *configuration* of a Turing machine is triple (w_L, q, w_R) such that $q \in Q$ is the current state, w_L is string that is to the left of the head position and w_R is the right string, including the current symbol s. We denote by ε the string that contains just the blank symbol. An *input* of a Turing machine is a string x of symbols in Σ. An initial configuration for an input x is (ε, q_0, x).

We define Halt_{TM} [119] to be the following problem:

Problem 2 *Given the description of a Turing machine $\mathcal{M}$ and its input x, will $\mathcal{M}$ halt on x?* □

This is a well known undecidable problem and can be posed as a reachability problem: *Given the description of a Turing machine $\mathcal{M}$ and its input x, will configuration (w'_L, q', w'_R) be reachable from configuration (w_L, q, w_R)?*

7.1.3 Piecewise affine maps (PAMs)

We define in this section one dimensional piecewise affine maps (PAMs) [46, 91, 92][1].

[1]In [92] PAMs are called *piecewise-linear continuous functions.*

Definition 45 *We say that a function $f : \mathbb{R} \to \mathbb{R}$ is* piecewise affine *(PAM) if f is of the form $f(x) = a_i x + b_i$ for $x \in I_i$, where $I_i = [l_i, u_i]$ is an interval with $l_i, u_i \in \mathbb{R}$.* ■

Whenever a_i, b_i and the extremities of I_i are rational we say that f is a *rational* PAM. In the same obvious way we can define *piecewise-analytic* and *piecewise-monotone* functions. If the set of I_i's is countably infinite rather than finite we say that f is a *countably* PAM. If f is injective we say that the PAM is *injective* ($\mathrm{PAM_{inj}}$).

The reachability problem for a PAM $\mathcal{A}$ can be defined as a predicate

$$Reach(\mathcal{A}, x, y) \equiv \exists k \,.\, y = f^k(x).$$

Let REACH_{PAM} be the following problem.

Problem 3 *Given a PAM $\mathcal{A}$ and two points x and y, is y reachable from x?* □

Even for a function f with just two linear pieces, there is no known decision algorithm for the above problem. The same problem is known to be undecidable in dimension 2 and if piecewise affine maps are replaced by polynomials, the problem is open for any dimension [46, 91, 92].

7.1.4 About the notion of simulation

We say that two classes of systems are *equivalents* if and only if each system of one class can be *simulated* by a system of the other and vice versa. In what follows the term "equivalent" will be used in this sense. Even though the idea of *simulation* (*abstraction* or *realization*) is more or less understood (and uniform) in the Computer Science community as "machines that perform the same computation" [39, 111, 112], in dynamical systems, simulation is captured by the notions of topological equivalence and homomorphism [66, 75, 132]. That rises the problem of having a good notion of simulation for systems that combine discrete with continuous dynamics like hybrid systems. Some attempts to overcome this problem are [18] and [49]. In any case, simulation is defined ad-hoc for some classes of models, for example in [19] a notion of simulation of a transition system by a PCD is given.

In the following sections we are not going to use a formal notion of simulation but just to use it with its intuitive (and usual) meaning knowing that if a system A simulates another system B and if a given problem $P(B)$ is undecidable, then $P(A)$ is undecidable.

7.2 Between Decidability and Undecidability

We show in this section that slight generalizations of 2-dimensional PCDs lead to classes of systems for which the reachability problem is as as hard as for PAMs, for which such problem is known to be open. We will consider hierarchical PCDs (HPCDs), PCDs in 2-dimensional manifolds, rectangular hybrid automata with one clock x and one positive n-skewed clock y (under some restrictions) and linear hybrid automata with two stopwatches (with some restrictions). Recall that the reachability problem is decidable [106] for 2-dimensional PCDs and undecidable for dimensions greater than two [18].

7.2.1 HPCD

Hierarchical piecewise constant derivative systems (HPCDs) can be seen as hybrid automata such that at each location the dynamics is given by a PCD. More formally,

Definition 46 *A* hierarchical PCD *(HPCD) is a hybrid automaton $H_{PCD} = (\mathcal{X}, Q, f, \mathsf{I}_0, \mathsf{Inv}, \delta)$ such that Q and I_0 are as before while the dynamics is not anymore a single differential equation but a PCD and each transition $tr = (\ell, g, \gamma, \ell')$ is such that (1) its guard g is a predicate of the form $P(x, y) \equiv (ax + by + c = 0 \wedge x \in I \wedge y \in J)$ where I and J are intervals and a, b, c and the extremities of I and J are rational-valued and (2) the reset functions γ are affine functions: $\gamma(\mathbf{x}) = A\mathbf{x} + \mathbf{b}$. Last, Inv is defined as the negation of the union of the guards, i.e. we can stay in location ℓ as long as no guard is satisfied.* ■

If all the PCDs are bounded, then the HPCD is said to be *bounded*.

Notation. In section 5.1 we have introduced a 1-dim coordinate system on each edge e: a point $\mathbf{x}$ on edge e with local coordinates λ is denoted by (e, λ) or whenever no confusion may arise, just as λ. If the dynamics of ℓ is given by PCD and that of ℓ' is given by PCD', for convenience and w.l.o.g. we will consider that a guard of a transition $tr = (\ell, g, \gamma, \ell')$ is a segment of an edge e of PCD and that reset functions assign points on edges of PCD to points on edges of PCD'. Given a point $\mathbf{x} = (x, y)$, we will write $\gamma(e, x, y) = (e', x', y')$, where $e \in EV(\ell)$ is an edge, $e' \in EV(\ell')$ is the target edge of PCD', and x' and y' are the new values of the variables after the reset. In some cases we will use the local coordinates of the point on the edge and in this case we will write $\gamma(e, \lambda) = (e', \lambda')$

Notice that the notions of *ℓ-trajectory segment* and *trajectory* are the same as for hybrid automata (see section 2.2) and that the notion of edge-to-edge *successor* and its relation with trajectories is defined on the same lines as for PCDs (see section 2.3.1) except that in HPCDs there are resets and in this case the successor is given by the reset function.

It can be argued that the term *hierarchical* in the above definition is superfluous and that in fact HPCDs are just 2 dimensional linear hybrid automata. Even though this is true, the definition is intentional since we want to emphasize the fact that there are just "few" real discontinuities due to jumps and reset and that in general the trajectory behaves like a PCD.

Given two points $\mathbf{x}_0 = (e, \lambda_0)$ and $\mathbf{x}_f = (e', \lambda_f)$, the reachability problem for a HPCD $\mathcal{H}$ can be defined as a predicate $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \sigma \,.\, \lambda_f = \mathsf{Succ}_\sigma(\lambda_0)$. Let REACH_{HPCD} be the following problem:

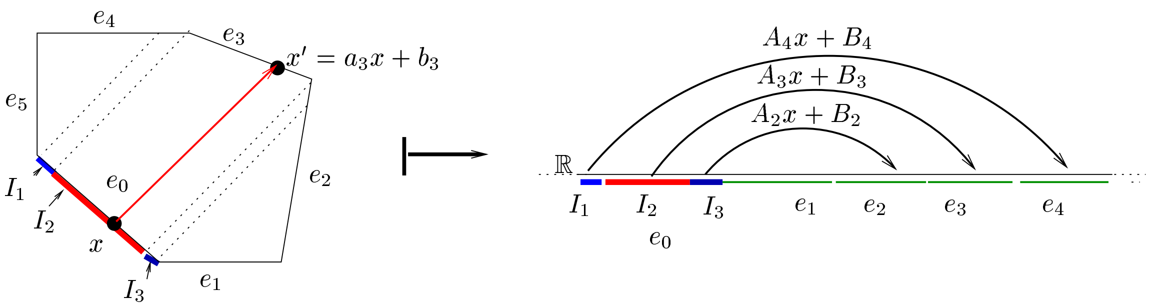


Figure 7.4: Sketch of the simulation of a HPCD by a PAM.

Problem 4 *Given a HPCD $\mathcal{H}$ and two points $\mathbf{x}_0$ and $\mathbf{x}_f$, is $\mathbf{x}_f$ reachable from $\mathbf{x}_0$?* □

We will prove that HPCDs are equivalent to PAMs, in the sense of section 7.1.4. For that we show first that each HPCD $\mathcal{H}$ is simulated by a PAM $\mathcal{A}$ and that for each PAM $\mathcal{A}$ there is a HPCD $\mathcal{H}$ such that $\mathcal{H}$ simulates $\mathcal{A}$. For proving the first, we should: (1) Encode an initial and final point of $\mathcal{H}$ by points on some intervals of $\mathcal{A}$; (2) Represent a configuration of $\mathcal{H}$ by a configuration of $\mathcal{A}$; (3) Simulate an edge-to-edge transition of $\mathcal{H}$ by some function application on $\mathcal{A}$.

Lemma 46 (PAMs simulate HPCDs) *For every 2-dimensional HPCD $\mathcal{H}$ there is a PAM $\mathcal{A}$ such that $\mathcal{A}$ simulates $\mathcal{H}$.*

Sketch of the proof: The idea of the proof is the following. We arrange all the edges of $\mathcal{H}$ in the Real line (in an arbitrary order) and we represent each edge-to-edge successor function and each reset function by an affine map (restricted to an interval). Assembling all those affine maps together yields the PAM $\mathcal{A}$ simulating $\mathcal{H}$. See Figure 7.4 and Appendix B for details. □

Remark. Notice that to not have problem in the above simulation, the HPCD $\mathcal{H}$ must be bounded.

In order to prove that HPCDs simulate PAMs we should: (1) Encode an initial and final point of $\mathcal{A}$ by points on some edges on $\mathcal{H}$; (2) Represent a configuration of $\mathcal{A}$ by a configuration of $\mathcal{H}$; (3) Simulate one-step computation of $\mathcal{A}$ by some trajectory segment (many-steps successor) on $\mathcal{H}$.

Lemma 47 (HPCDs simulate PAMs) *For every PAM $\mathcal{A}$ there is a 2-dimensional HPCD $\mathcal{H}$ such that $\mathcal{H}$ simulates $\mathcal{A}$.*

Sketch of the proof: Let $\mathcal{A}$ be defined by $f(z) = a_iz + b_i$ if $z \in I_i$ for $i \in \{1, \cdots, k\}$ where $I_i = [l_i, u_i]$ are rational intervals. We define a one-location HPCD with a one-region PCD

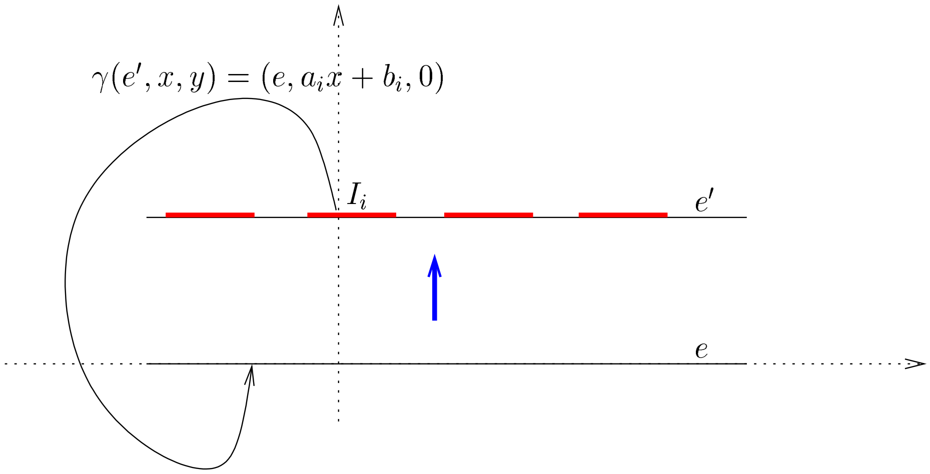


Figure 7.5: The HPCD that simulates a PAM.

defined by $y \geq 0 \wedge y \leq 1$, i.e. there are two edges $e \equiv y = 0$ and $e' \equiv y = 1$, and dynamics defined by vector $(0, 1)$ as shown in figure 7.5. There are as many transitions as intervals I_i of the PAM. The guards are of the form $y \in e \wedge x \in I_i$ and the reset functions associated with these guards are of the form $\gamma(e', x, y) = (e, a_i x + b_i, 0)$. The initial point z_0 of the PAM is encoded as a point $(x_0, y_0) \in e$ with local coordinate $\lambda_0 = x_0 = z_0$. Hence, it is easy to see that $z_f = f(z_0)$ iff $\lambda_f = \gamma(e', \lambda')$ where $\lambda' = \mathsf{Succ}_{ee'}(\lambda_0)$. $\square$

From the above two lemmas, we have then the following theorem.

Theorem 48 (HPCDs are equivalent to PAMs) REACH_{HPCD} *is decidable iff* REACH_{PAM} *is.* $\square$

Remark. It can be said that encoding everything in reset functions is not fair. Indeed, the simulation works for less general resets. In Lemma 47' (see Appendix B) it is shown that any PAM can be simulated by an HPCD with reset functions of the form $\gamma(x, y) = (y + d, 0)$. Let us denote the corresponding HPCD by $\mathrm{HPCD_{iso}}$ and its reachability problem by $\mathsf{REACH}_{\mathrm{HPCD_{iso}}}$. Hence we have the following theorem.

Theorem 49 ($\mathrm{HPCD_{iso}}$ **are equivalent to PAMs)** $\mathsf{REACH}_{\mathrm{HPCD_{iso}}}$ *is decidable iff* REACH_{PAM} *is.* $\square$

7.2.2 About rectangular and linear 2-dimensional hybrid automata

In this section we prove some corollaries of Theorem 48 and Theorem 49.

We start proving that $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$ and $\mathrm{RA}_{2\mathrm{cl}}$ are equivalent to PAMs, as a consequence of Theorem 48.

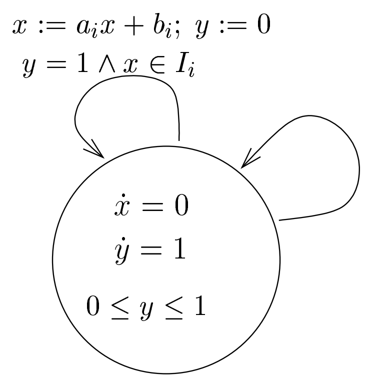


Figure 7.6: A $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$ equivalent to HPCD of Figure 7.5.

The class of one-state rectangular hybrid automata with one clock y, one memory cell x, invariants of the form $C \leq y \leq D$, guards of the form $y = D$ and resets of the form $\gamma(x, y) = (ax + b, 0)$ will be denoted as $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$.

Corollary 50 ($\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$ are equivalent to PAMs) *Reachability for* $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$ *is decidable iff reachability for PAMs is.*

Sketch of the proof: The proof consists in taking the HPCD defined for simulating a PAM (see Figure 7.5) and building an equivalent $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$. The corresponding $\mathrm{RA}_{1\mathrm{cl}1\mathrm{mc}}$ is pictured in Figure 7.6. □

The class of one-state rectangular hybrid automata with two clocks x and y, invariants of the form $C \leq y \leq D$, guards of the form $y = D$ and resets of the form $\gamma(x, y) = (ax + b, 0)$ will be denoted as $\mathrm{RA}_{2\mathrm{cl}}$.

Corollary 51 ($\mathrm{RA}_{2\mathrm{cl}}$ are equivalent to PAMs) *Reachability for* $\mathrm{RA}_{2\mathrm{cl}}$ *is decidable iff reachability for PAMs is.*

Sketch of the proof: In Lemma 47 an HPCD $\mathcal{H}$ (see Figure 7.5) that simulates a PAM was built. We obtain another HPCD $\mathcal{H}'$ applying an affine transformation to $\mathcal{H}$, where the e edge remains unchanged whereas e' is translated by one unit to the right. $\mathcal{H}'$ is represented in Figure 7.7-(a), where given $I = [l, u]$ $I + 1$ is a short for $[l + 1, u + 1]$. It is not difficult to see that the automaton of Figure 7.7-(b) is a $\mathrm{RA}_{2\mathrm{cl}}$ equivalent to $\mathcal{H}'$. □

Notice that $\mathrm{RA}_{2\mathrm{cl}}$ automata can be considered as a generalization of *updatable* timed au-

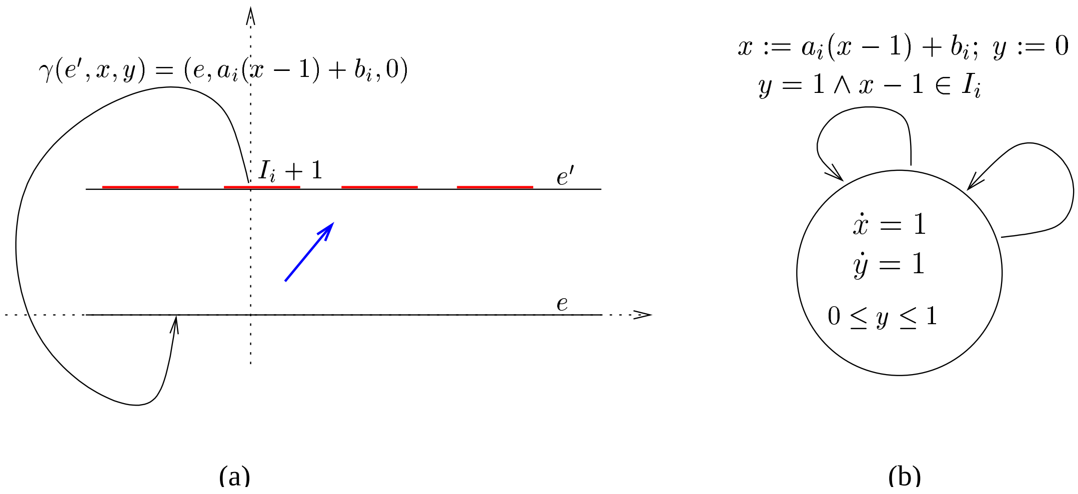


Figure 7.7: (a) A HPCD that simulates a PAM; (b) The corresponding $\mathrm{RA}_{2\mathrm{cl}}$.

tomata [47, 48] since some resets on $\mathrm{RA}_{2\mathrm{cl}}$ are more general (of the form $y := ax + b$).

In what follows we prove two corollaries that are consequence of Theorem 49.

We denote by $\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$, the class of rectangular hybrid automata with one two-slope clock x (taking values on $\{-1, 1\}$) and one positive n-skewed clock y with the following restrictions: (1) on each transition, x is reset to function of y of the form $x := y + d$ and y is reinitialized with a constant value c, where c is the inferior bound of y in ℓ'; (2) the values of the two variables are never compared, and (3) the guard of a transition from location ℓ to ℓ' is of the form $x = A$, where A is one of the bounds of x in the invariant of location ℓ.

Corollary 52 ($\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$ are equivalent to PAMs) *Reachability for* $\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$ *is decidable iff reachability for PAMs is.*

Sketch of the proof: Given a PAM, we construct a HPCD which simulates it as in the proof of Lemma 47' (see Figure 7.8-(a) for a fragment of this HPCD). It is easy to see that this is in fact a $\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$ (see Figure 7.8-(b)). □

Let $\mathcal{H}$ be a linear hybrid automaton with just two (mutually exclusive) stopwatches x and y with the following restriction: (1') whenever a transition is taken, x and y remain unchanged or the new value of x is a function of y of the form $x := y + d$ and y is reinitialized with a constant value c, where c is the inferior bound of y in ℓ'; (2') the guard of a transition from ℓ to ℓ' is of the form $x = A$ or $ax + by + c = 0$, where A is one of the bounds of x in the invariant of location ℓ and a, b and c are rational constants. We denote these class by $\mathrm{LA}_{\mathrm{st}}$.

Corollary 53 ($\mathrm{LA}_{\mathrm{st}}$ are equivalent to PAMs) *Reachability for* $\mathrm{LA}_{\mathrm{st}}$ *is decidable iff reachability for PAMs is.*

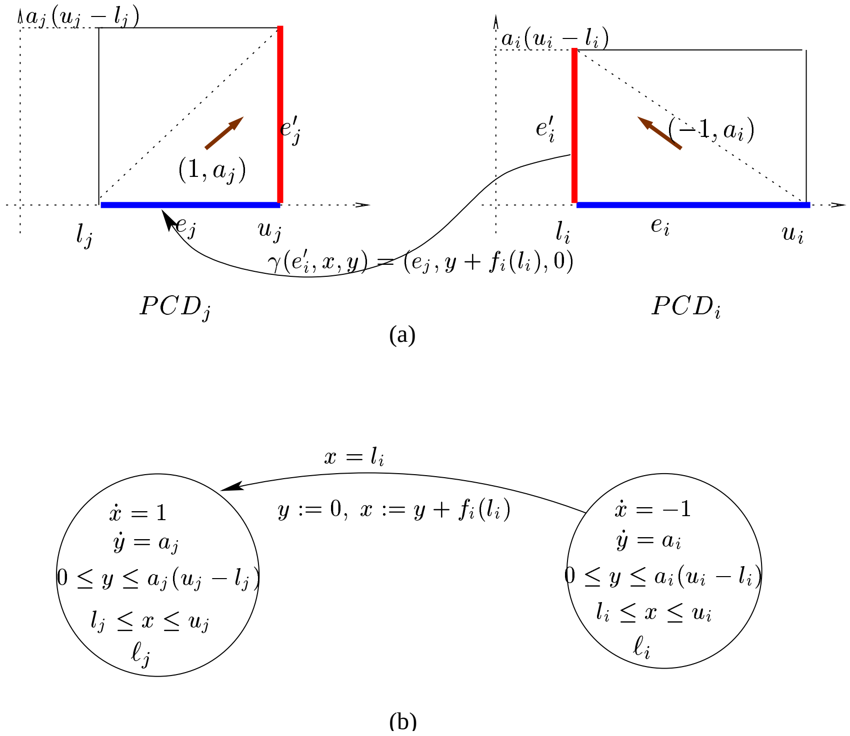


Figure 7.8: (a) A HPCD with two locations; (b) The corresponding $\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$.

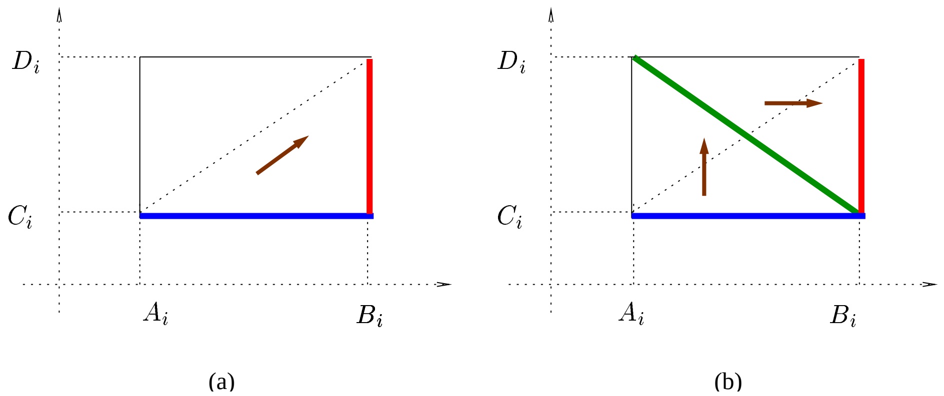
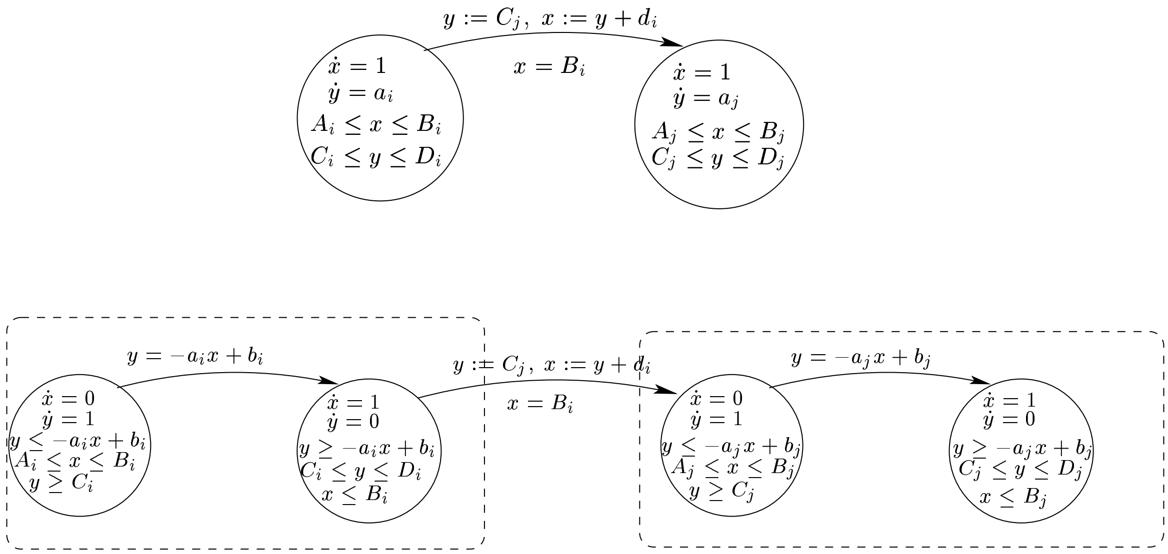


Figure 7.9: (a) A one-location $\mathrm{RA}_{1\mathrm{sk}1\mathrm{sl}}$; (b) An equivalent two-location $\mathrm{LA}_{\mathrm{st}}$.

Sketch of the proof: We use the previous corollary to build a $\mathrm{RA}_{\mathrm{1sk1sl}}$ that simulates a given PAM. It is well known that rectangular hybrid automata with both negative and positive skewed clocks are equivalent to similar automata with only positive values on the skewed clocks [117]. Given a $\mathrm{RA}_{\mathrm{1sk1sl}}$ $\mathcal{H}_R$ we obtain the corresponding timed automaton $\mathcal{H}_P$ using the procedure of [117] and we obtain a $\mathrm{LA}_{\mathrm{st}}$ $\mathcal{H}_L$ just decomposing each location of $\mathcal{H}_T$ into two as shown in Figure 7.9. In Figure 7.10 we show a two-location $\mathrm{RA}_{\mathrm{1sk1sl}}$ with one clock and one skewed clock and its corresponding $\mathrm{LA}_{\mathrm{st}}$. □



where $b_i = C_i + B_i a_i$ and $b_j = C_j + B_j a_j$

Figure 7.10: From $\mathrm{RA}_{\mathrm{1sk1sl}}$ to $\mathrm{LA}_{\mathrm{st}}$.

7.2.3 $\mathrm{PCD}_{\mathrm{2m}}$: PCDs on 2-dimensional manifolds

Surfaces (or 2-dimensional manifolds) were introduced in section 7.1.1. In this section we introduce *PCDs on 2-dimensional manifolds* and we show that the reachability problem for this class of systems is equivalent to a subproblem of REACH_{PAM} (Problem 3). By definition a 2-dimensional manifold must be triangulable and to define a PCD on a (triangulated) *2-dimensional manifold* we have to define a PCD on each of its triangles.

Definition 47 *A* PCD on 2-dimensional manifolds *(*$\mathrm{PCD}_{\mathrm{2m}}$*) is a PCD defined on a surface.* ■

Notice that a PCD can have any kind of polygonal regions, but w.l.o.g. we will consider here triangles as regions, since all the regions of the original PCD can always be triangulated as shown in Figure 7.11 where a PCD on a torus is defined.

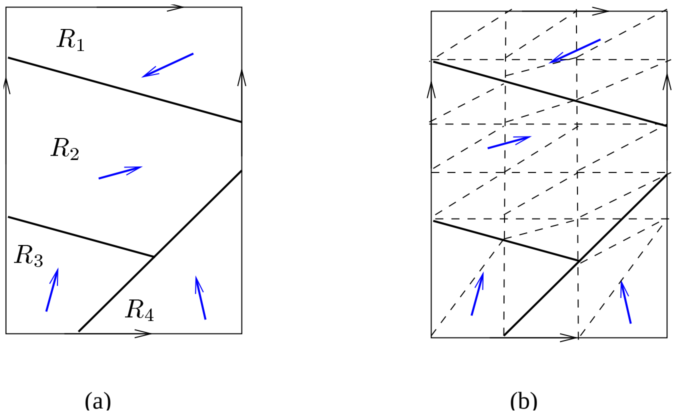


Figure 7.11: (a) A $\mathrm{PCD}_{2\mathrm{m}}$-Torus T; (b) A triangulation of T.

Given two points $\mathbf{x}_0 = (e, \lambda_0)$ and $\mathbf{x}_f = (e', \lambda_f)$, the reachability problem for a $\mathrm{PCD}_{2\mathrm{m}}$ $\mathcal{H}$ can be defined as a predicate $Reach(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f) \equiv \exists \sigma \ . \ \lambda_f = \mathsf{Succ}_\sigma(\lambda_0)$. We define $\mathsf{REACH}_{\mathrm{PCD}_{2\mathrm{m}}}$ to be the following problem:

Problem 5 *Given a* $\mathrm{PCD}_{2\mathrm{m}}$ $\mathcal{H}$ *and two points* $\mathbf{x}_0$ *and* $\mathbf{x}_f$*, is* $\mathbf{x}_f$ *reachable from* $\mathbf{x}_0$*?* $\square$

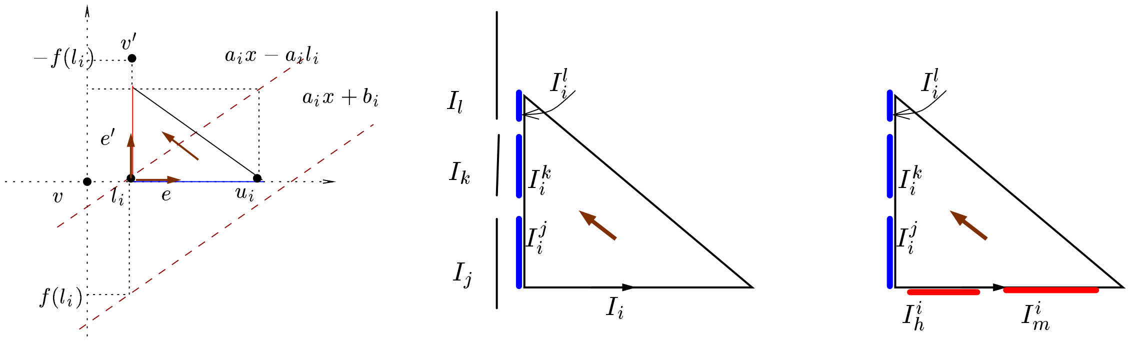
We show that indeed the decidability of Problem 5 is an open problem, showing as before, that $\mathsf{REACH}_{\mathrm{PCD}_{2\mathrm{m}}}$ is equivalent to REACH_{PAM} for injective PAMs.

Lemma 54 ($\mathrm{PAM}_{\mathrm{inj}}$ **simulate** $\mathrm{PCD}_{2\mathrm{m}}$**)** *Every* $\mathrm{PCD}_{2\mathrm{m}}$ *can be simulated by an injective PAM.*

Sketch of the proof: Let $\mathcal{H}$ be a $\mathrm{PCD}_{2\mathrm{m}}$. The reduction consists in two parts. First, refine $\mathcal{H}$ by triangulation. Second, we proceed as for the simulation of HPCDs by PAMs. Notice that $\mathcal{H}$ is in fact an HPCD where a jump is produced each time we reach an identified edge and the resets are the identity on the local coordinate of the identified edges. We will not reproduce the proof here, see Lemma 46. The requirement that each edge is identified with exactly one other edge ensures injectivity. $\square$

Lemma 55 ($\mathrm{PCD}_{2\mathrm{m}}$ **simulate** $\mathrm{PAM}_{\mathrm{inj}}$**)** *Every injective PAM can be simulated by a* $\mathrm{PCD}_{2\mathrm{m}}$*.*

Sketch of the proof: Let $\mathcal{A}$ be an injective PAM defined as $f(z) = a_i z + b_i$ if $z \in I_i$ for $1 \leq i \leq n$. We obtain a $\mathrm{PCD}_{2\mathrm{m}}$ in the following way. First define a point on an edge e with local coordinate λ equal to z and then define a PCD such that $\mathsf{Succ}_{ee'}(\lambda) = a_i\lambda + b_i$ (see Figure 7.12-(a)). Consider then the line defined by edge e'. This edge is partitioned into intervals I_i^k if $f(I_i) \cap I_k \neq \emptyset$, as shown in Figure 7.12-(b). In fact, $I_i^k = f(I_i) \cap I_k$. Each



(a) (b) (c)

Figure 7.12: Simulation of a $\mathrm{PCD}_{2\mathrm{m}}$ by a $\mathrm{PAM}_{\mathrm{inj}}$: (a) First definition of PCD_i; (b) Decomposition of edge e_i'; (c) Partition of edge I_i.

segment I_i^k of this edge is identified with the corresponding interval on the oriented edges I_k. Finally, edge I_i is partitioned into intervals I_i^h if $f(I_h) \cap I_i \neq \emptyset$, again $I_i^h = f(I_h) \cap I_i$ (see Figure 7.12-(c)). We obtain in this way (doing the same construction for every interval I_i, $1 \leq i \leq n$) a surface with boundary. It is important to emphasize that each $I_i^j \in e_i'$ (for all $1 \leq i \neq j \leq n$) is identified with exactly one segment of edge I_j. By the Classification Theorem for Surfaces with Boundary (see Theorem 45) we have that this surface is equivalent to a sphere with some disks removed and we obtain then a $\mathrm{PCD}_{2\mathrm{m}}$ just "sewing" the disks. We associate with these disks a zero dynamics, i.e. the dynamics on these regions are given by the vector $(0,0)$. See Appendix B for a detailed proof. □

From the above two lemmas we have that $z_f = f^*(z_0)$ iff $\mathrm{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$, where $\mathbf{x}_0$ has local coordinate $\lambda_0 = z_0$ on a given edge e and $\mathbf{x}_f$ has local coordinate $\lambda_f = z_f$ on an edge e'. Then the following theorem holds.

Theorem 56 ($\mathrm{PCD}_{2\mathrm{m}}$ **are equivalent to** $\mathrm{PAM}_{\mathrm{inj}}$) *Reachability for* $\mathrm{PCD}_{2\mathrm{m}}$ *is decidable iff reachability for injective PAMs is.* □

We have proved in this section that the decidability of problems REACH_{HPCD} and $\mathsf{REACH}_{\mathrm{PCD}_{2\mathrm{m}}}$ are as hard as the decidability of REACH_{PAM}.

7.3 Undecidability Results

The importance of this section does not lie in the results themselves (since they are quite straightforward) but on the fact that the class of simple systems like HPCDs reveals to really be in the boundary between decidable and undecidable hybrid systems. Indeed, modifying

HPCDs slightly we obtain other classes of systems for which the reachability problem become undecidable. We consider two kinds of modifications: the first is the addition of a simple counter that gives rise to HPCD_{1c} and the second is the addition of an *infinite pattern*. We will consider two kinds of "infinite patterns", the first is the possibility of having infinite number of regions (HPCD_{∞}) and the second one is the possibility of having dynamics with some kind of periodic behavior (HPCD_{x}).

7.3.1 HPCDs with one counter (HPCD_{1c})

We introduce in what follows HPCDs with an additional element: a *counter*. We next show that these systems simulate Turing machines.

Definition 48 *A* HPCD with one counter HPCD_{1c} *is a HPCD* $\mathcal{H} = (\mathcal{X}, Q, f, \mathsf{I}_0, \mathsf{Inv}, \delta)$ *where:* $\mathcal{X} \subseteq \mathbb{R}^2 \times \mathbb{N}$ *is augmented with a memory cell c called a* counter*;* Q *and* Inv *are as before;* I_0 *is as before, initializing also the counter c; the dynamics of each location* ℓ *is a PCD (by definition,* $f_\ell(c) = 0$*); transitions are such that: (1) guards g are of the form* $P(x,y) \wedge Q(c)$ *where* $P(x,y)$ *is as for HPCDs and* $Q(c) \equiv c = 0 \mid c > 0 \mid \mathsf{true}$*, and (2) the reset function, for a given point* $\mathbf{x} = (x,y) \in e$ *is defined as* $\gamma(e,x,y,c) = (e', 0, f_2(x), f_3(c))$*, where* f_2 *is of the form* $ax + b$ *and* $f_3(c) = c + 1$ *or* $f_3(c) = c - 1$*.* ■

Informally, a HPCD_{1c} is similar to a HPCD[2] augmented with a counter that can just be incremented or decremented at each transition and that can be tested against zero.

Let us consider an HPCD with one counter (HPCD_{1c}). We prove that the reachability problem for HPCD_{1c} is undecidable showing that a HPCD_{1c} $\mathcal{H}$ can simulate a Turing machine $\mathcal{M}$. In order to do that we first encode a configuration (w_L, q, w_R) (we consider that the current symbol is the first character of w_R) of $\mathcal{M}$ as a point $\mathbf{x}$ on a given edge on a location ℓ of $\mathcal{H}$with (rational) local coordinate λ. We show then that each basic operation of $\mathcal{M}$ can be simulated by an algebraic operation on x performing some computation on $\mathcal{H}$.

It is well known that the string $w_L w_R$ can be represented as a positive rational number z as follows. Let $w_L = \ldots a_i, \ldots, a_1, a_0$ and $w_R = a_{-1}, a_{-2}, \ldots, a_{-j} \ldots$ (w.l.o.g. we suppose that $\mathcal{M}$ has at least one '1'). Then we define

$$z = \sum_{i=-\infty}^{\infty} a_i 2^i$$

Notice that w_L and w_R can be obtained by taking int_z (the integer part of z) and frac_z (its fractional part). If $\mathrm{frac}_z < \frac{1}{2}$ then the current symbol is 0, otherwise it is 1.

Proposition 4 (HPCD_{1c} simulate TMs) *For every TM* $\mathcal{M}$ *there is a 2-dimensional HPCD with one counter* $\mathcal{H}$ *such that* $\mathcal{H}$ *simulates* $\mathcal{M}$*.*

[2]Notice that it is not exactly a HPCD, since the reset are different

Sketch of the proof: We associate with each TM-state q_i three locations ℓ_i, ℓ'_i and ℓ''_i of HPCD_{1c} defined as (see Figure 7.13 and 7.14). A configuration of the TM is encoded as a point $\mathbf{x}$ on edge e_1 with local coordinate λ. The dynamics is defined in such a way that a kind of spiral is produced in order to reach edges e_2 or e_3. The counter is used to count the number of cycles of the spiral, thus it counts the 'integer' part of λ. Notice that edge e_2 or e_3 is reached depending on whether $\mathrm{frac}_\lambda \geq \frac{1}{2}$ (current symbol is 1) or $\mathrm{frac}_\lambda < \frac{1}{2}$ (current symbol is 0) and at this moment the value of c is $c = \mathrm{int}_\lambda$. If edge e_2 is reached, then guard g_2 is satisfied and the system jumps to edge e_2 on location ℓ'_i with $\lambda' = \mathrm{frac}_\lambda$ and $c = \mathrm{int}_\lambda$. At ℓ'_i, λ' is incremented at the same time c is decremented and whenever $c = 0$ (and hence $\lambda' = \lambda$) a jump to edge e_1 on location ℓ_j is produced where $c = 0$ and $\lambda_f = f'(\lambda')$. The proof is similar for edge e_3 on location ℓ_i, arriving at location ℓ_k with $c = 0$ and $\lambda_f = f''(\lambda')$. Each TM-instruction is encoded then as follows:

$q_i\, s \longrightarrow s\, q_j\, R$: Take $f'(\lambda) = f''(\lambda) = 2\lambda$.

$q_i\, 0 \longrightarrow 1\, q_j\, R$: Define $f''(\lambda) = 2\lambda + 1$.

$q_i\, 1 \longrightarrow 0\, q_j\, R$: In this case we take $f'(\lambda) = 2\lambda - 1$.

$q_i\, s \longrightarrow s\, q_j\, L$: Take $f'(\lambda) = f''(\lambda) = \frac{1}{2}\lambda$.

$q_i\, 0 \longrightarrow 1\, q_j\, L$: Define $f''(\lambda) = \frac{1}{2}\lambda - \frac{1}{2}$.

$q_i\, 1 \longrightarrow 0\, q_j\, L$: Take $f'(\lambda) = \frac{1}{2}\lambda + \frac{1}{2}$.

We have then that $\mathrm{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ iff a TM stops in a configuration (w_{L_f}, q_f, w_{R_f}) starting at a given configuration (w_{L_0}, q_0, w_{R_0}).

See appendix for a detailed proof. □

7.3.2 HPCDs with infinite partition (HPCD_∞)

We will consider in this section HPCDs for which we relax the condition of having a finite number of regions. We call this class of systems, *HPCDs with infinite partition* (HPCD_∞). We are not going to define this class formally, since we are just interested in showing that this additional feature (having an infinite partition) leads immediately to the undecidability of the reachability problem for HPCD_∞ ($\mathsf{REACH}_{\mathrm{HPCD}_\infty}$):

From the previous section we can conclude that the difficulty in simulating a TM is that on PCDs (HPCDs) it cannot be distinguished whether the current symbol is 0 or 1. Indeed, for HPCD_{1c} we can emulate that adding one counter to memorize the integer part of the TM-string representation in order to obtain the fractional part to decide whether the current symbol is 0 or 1. The following proposition shows that this test can be easily done with an infinite partition.

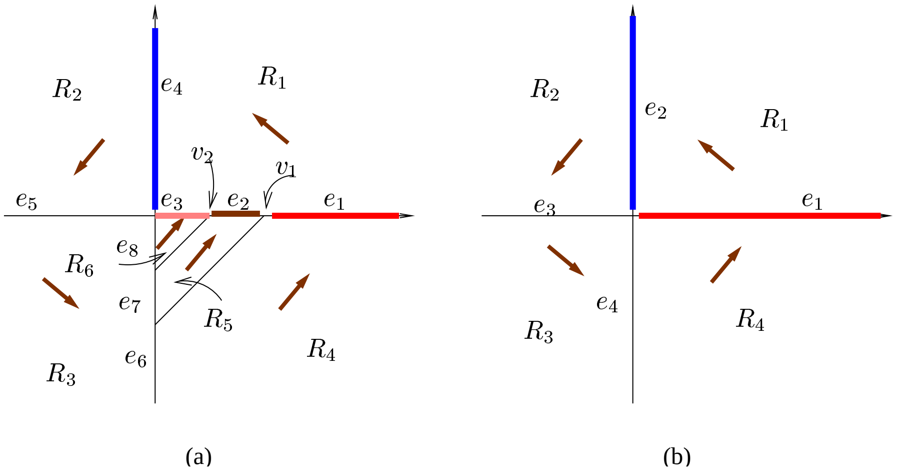


Figure 7.13: Sketch of the simulation of a TM by a HPCD_{1c}: (a) PCD_i; (b) PCD'_i and PCD''_i.

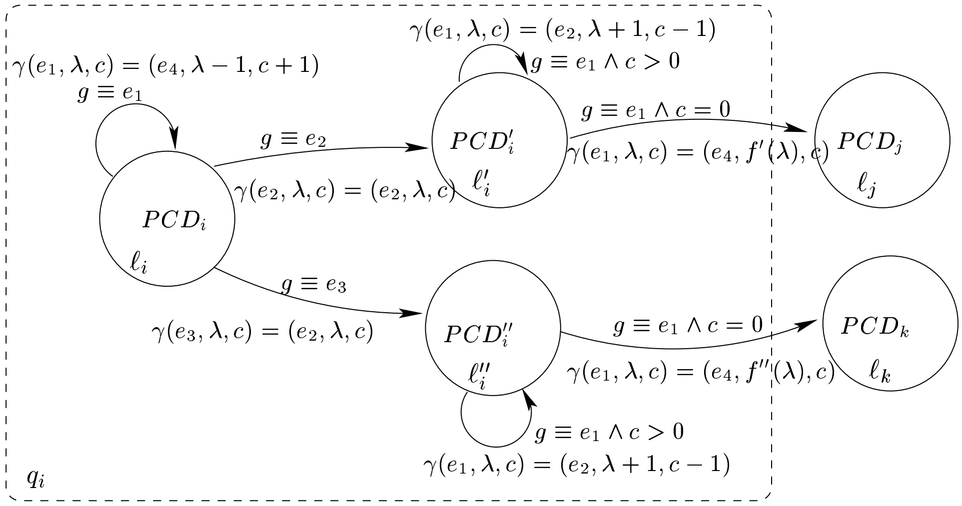


Figure 7.14: Sketch of the simulation of a TM by a HPCD_{1c}: Representation of TM-state q_i.

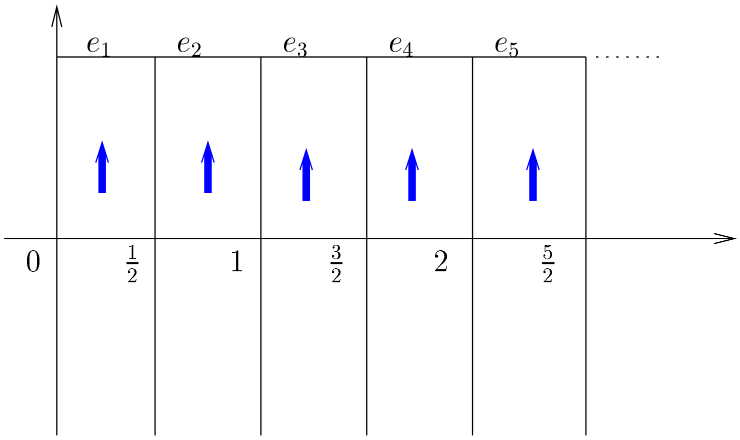


Figure 7.15: Sketch of the simulation of a TM by a HPCD_∞.

Proposition 5 **(HPCD_∞ simulate TMs)** *For every TM $\mathcal{M}$ there is a 2-dimensional HPCD with infinite partition $\mathcal{H}$ such that $\mathcal{H}$ simulates $\mathcal{M}$.*

Sketch of the proof: We represent the TM tape contents by a point on the x-axis with the abscissa $x = \sum_{i=-\infty}^{\infty} a_i 2^i$ in a HPCD with infinite partition as in Figure 7.15. In this case it is not necessary to test whether the current symbol is 0 or 1: whenever an "even" edge is reached (e_i with $i = 2k$ for $k \in \mathbb{N}$), that corresponds to $\text{frac}_\text{x} > \frac{1}{2}$, a jump is produced to a "1-state", whereas the "odd" edges are sent to "0-states". □

7.3.3 Origin-dependent rate HPCDs (HPCD_x)

Another way of introducing "infinite patterns" is allowing continuous dynamics with some periodic behavior that depends on the initial points after a reset is done. We define first *origin-dependent rate PCDs* to extend then the definition to HPCDs (HPCD_x).

Definition 49 *An* origin-dependent rate PCD *is a PCD $\mathcal{H} = (\mathbb{P}, \mathbb{F})$ such that each region P_s has dynamics $\dot{\mathbf{x}} = \phi_s(\mathbf{x}_0)$ (as before, given a generic region P we will also use the notation $\phi(P, \mathbf{x}_0)$).* ■

In the construction of Proposition 6 we will use rather particular ϕ_s functions.

A *trajectory segment* in some interval $[0, T] \subseteq \mathbb{R}$, with initial condition $\mathbf{x} = \mathbf{x}_0$, is a continuous and almost-everywhere (everywhere except on finitely many points) derivable function $\xi(\cdot)$ such that $\xi(0) = \mathbf{x}_0$ and $\exists t_0, t_1, \ldots, t_n$ such that $t_0 = 0$ and $t_n = T$, and $\mathbf{x}_i = \xi(t_i) \in e$ for some edge $e \in In(P)$ and for all $t \in [t_i, t_{i+1})$, $\dot{\xi}(t) = \phi(P, \mathbf{x}_i)$ is defined and is equal to $\dot{\xi}(t_i)$.

After reaching an edge, the system evolves according to a fixed rate that depends on the initial value of the variables when entering the region. The idea of having flows (dynamics)

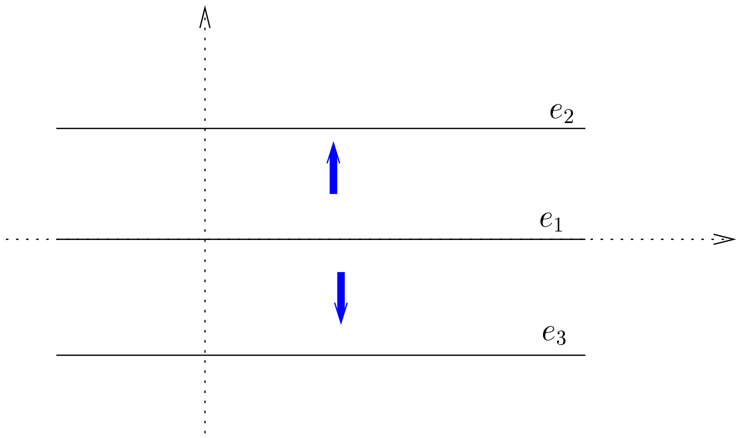


Figure 7.16: Sketch of the simulation of a TM by a $\mathrm{HPCD_x}$.

that depend on initial sates has been taken from [12]. We extend the above definition to HPCDs:

Definition 50 *An* origin-dependent rate HPCD ($\mathrm{HPCD_x}$) *is a HPCD such that each PCD is an origin-dependent rate PCD.* ■

Let $\mathsf{REACH}_{\mathrm{HPCD_x}}$ be the reachability problem for $\mathrm{HPCD_x}$, stated in a similar way as for HPCDs. We show next that $\mathsf{REACH}_{\mathrm{HPCD_x}}$ is undecidable showing that $\mathrm{HPCD_x}$ simulate Turing machines.

Proposition 6 ($\mathrm{HPCD_x}$ simulate TMs) *For every TM $\mathcal{M}$ there is a 2-dimensional* $\mathrm{HPCD_x}$ *$\mathcal{H}$ such that $\mathcal{H}$ simulates $\mathcal{M}$.*

Proof: We associate with each TM-state q_i a location ℓ_i, where the PCD_i is defined as:

Region	*Defining conditions*	*Vector*
R_1	$(y > 0) \wedge (y < 1)$	$(0, f(x_0))$
R_2	$(y < 0) \wedge (y > -1)$	$(0, f(x_0))$
R_3	$y < -1$	(0,1)
R_4	$y > 1$	(0,1)

See Figure 7.16. Let e_1, e_2 and e_3 be defined as: $e_1 \stackrel{\text{def}}{=} (y = 0)$, $e_2 \stackrel{\text{def}}{=} (y = 1)$ and $e_3 \stackrel{\text{def}}{=} (y = -1)$ and $f(x_0) = (-1)^{\lfloor 2x_0 \rfloor}$, where x_0 is the first coordinate on edge e_0 of the initial point $\mathbf{x}_0$. Notice that f is in fact the function

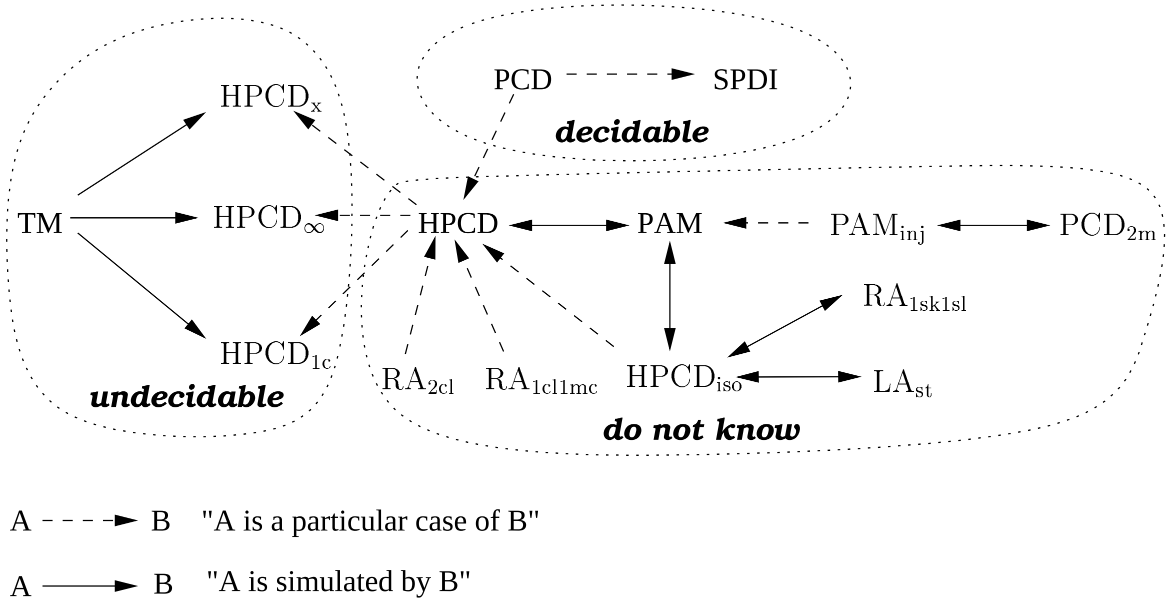


Figure 7.17: Relation between the classes of 2-dimensional hybrid systems considered in this work.

$$f(x) = \begin{cases} 1 & \text{if } \mathrm{frac_x} < \frac{1}{2} \\ -1 & \text{otherwise} \end{cases}$$

There are two transitions from ℓ_i: (1) $tr_1 = (\ell_i, g_1, \gamma_1, \ell_j)$ where $g_1 \equiv e_2$ and $\gamma_1(e_2, x) = (e_1', f'(x))$; (2) $tr_2 = (\ell_i, g_2, \gamma_2, \ell_h)$ where $g_2 \equiv e_3$ and $\gamma_2(e_3, x) = (e_1, f''(x))$. Transitions tr_1 and tr_2 allow the trajectory to continue in locations ℓ_j and ℓ_h with a reset function that implement the instructions of the Turing machine as before. $\square$

Notice that the above definition allows the dynamics to be defined by any function of the initial point, but in order to simulate a TM we need very particular kind of functions, those that have a periodic pattern. We could have chosen any periodic function like sinus or cosinus. In any case, the key idea is to obtain an "infinite pattern" as before.

7.4 Summary and Related Work

As a summary of the result of this chapter, we picture in Figure 7.17 the relation of the main hybrid models we have considered in this thesis.

Although many intense research activity in the last years have been done in the domain of hybrid systems, there is no clear boundary between what is decidable or not on such systems. The contribution of this chapter is twofold. First, we have shown that between 2 dimensional PCDs (for which the reachability problem is decidable [106]) and 3 dimensional PCDs (reachability is undecidable [19]) there exist an interesting class, *2-dimensional HPCD*, for which the reachability question is still open. We have also shown that the same is true for other similar classes, namely 2-dimensional rectangular automata and 2-dimensional linear hybrid automata with some restrictions as well as for PCDs on 2-dimensional manifolds. Second, we have proved that 2-dimensional HPCDs are really in the boundary between decidability and undecidability, since adding a simple counter or allowing some kind of "infinite pattern" to these systems, makes the reachability problem undecidable.

It is well known that for particular cases the reachability question is decidable. In [6] it was shown that reachability is decidable for timed automata (TA), that are particular case of hybrid automata (all the variables have slope 1) and in [106] the decidability of the same problem for 2-dimensional PCDs was proved. In [47] some syntactic extensions of TA are considered (*updatable timed automata*) for which the decidability of the emptiness problem is studied and in [48] the expressive power of such automata are considered. Consider now the following restrictions: (1) whenever the dynamics change, its value is (nondeterministically) reinitialized, and (2) the value of two variables with different dynamics are never compared. Under (1) and (2), it was shown that the reachability problem for multirate automata (a hybrid automata such that the variables run at any constant slope) (see [5, 116]) and rectangular automata [80, 127] is decidable. Some more decidability results were given for subclasses of linear hybrid systems, *extended integrator graphs*, in [45] and for timed graphs with one stopwatch in [43].

On the other hand and not surprisingly, many undecidability results were given. In [80] it was shown that relaxing restriction (1), the reachability problem for rectangular automata with at least 5 clocks and one two-slope variable (with rational slopes $k_1 \neq k_2$) is undecidable and that relaxing (2) leads to undecidability of the reachability problem for rectangular automata with at least 5 clocks and one skewed clock. In [138] it was shown that the reachability question for TA with 3 stopwatches is undecidable (under restriction (2)) as well as for TA with 1 memory cell and allowing assignments between variables. Other undecidability results (always for the reachability problem) were given for TA with 6 memory cells without assignment between variables [11], for TA with two three-slope variables under restriction (1) [90], for TA with two non-clock constant slope variables [4], for TA with additive clock constraints [6] and for TA with two skewed clocks [5].

Some other undecidability results were given for low (three or less) dimensional spaces. In [19] it was shown that the reachability problem for 3-dim PCDs is undecidable. In [113] it was proved that Turing machines can be simulated by dynamical systems with piecewise affine functions (in 3 dimension spaces). In [93], two elementary functions are constructed: one in one dimension that simulates Turing machines (TMs) with an exponential slowdown and one in two dimensions that simulate TMs in real time. See references therein for other undecidability results. Among other results, in [49] it is shown that smooth ODEs in $\mathbb{R}^2$ can simulate arbitrary Turing machine and in [92] it is proved that TMs can be simulated by 2-dimensional PAMs, by one dimensional countable PAM (PAMs with an infinite number of intervals) and by a continuous piecewise-monotone functions in linear time. As a relevant result in the same work it is also shown that there exist TMs that cannot be simulated by a one dimensional PAM. Finally, in [46] the algorithmic complexity of hybrid and dynamical systems is analyzed, in particular in its fourth chapter it is studied the frontier between decidability and undecidability for low dimensional systems for some problems like the controllability of commuted linear systems in dimension two that are related to the mortality problem for two dimensional matrices and to the reachability problem for one dimensional PAMs. In [38] the decidability of other problems of hybrid systems different from reachability, like stability and controllability, are analyzed.

HPCD_∞ and HPCD_x are not O-minimal hybrid systems [99].

Chapter 8

SPeeDI

In this chapter we discuss some issues related to the prototype of the tool SPeeDI, that implements the reachability algorithm for SPDI presented in chapter 5. We present some theoretical results first and then we describe SPeeDI. The tool SPeeDI is a collection of utilities to manipulate and reason mechanically about SPDIs, completely implemented in 5000 lines of Haskell, a general-purpose, lazy, functional language.

A description of the tool will appear in [20].

Organization of the chapter: In section 1 we study some properties of types of signatures that are used by the SPeeDI algorithm. In the second section we describe the architecture of the tool and its main functionalities. In section 3 we discuss some implementation issues and in section 4 we present an example. Finally, the last section resume the chapter and future work.

8.1 Characterization of Types of Signatures

In this section we define the *underlying graph* of an SPDI to be a kind of "*symbolic*" graph of the SPDI and we show the relation between them. This is presented in the first part where one of the interesting relation is between paths on the graph and *feasible* signatures in the SPDI. In the second part we show some important properties of the signatures generated by Algorithm $\mathcal{A}$, in addition to the ones proved on Lemma 20, and we conjecture that in fact these properties are necessary and sufficient conditions for characterizing the set of types of signatures.

8.1.1 The graph of an SPDI

Given an SPDI $\mathcal{H}$, let $\mathcal{E}$ be the set of edges of $\mathcal{H}$, then we can define a graph $\mathcal{G}_{\mathcal{H}}$ where nodes correspond to edges of $\mathcal{H}$ and such that there exists an arc from one node to another if there exists a trajectory segment from the first edge to the second one without traversing

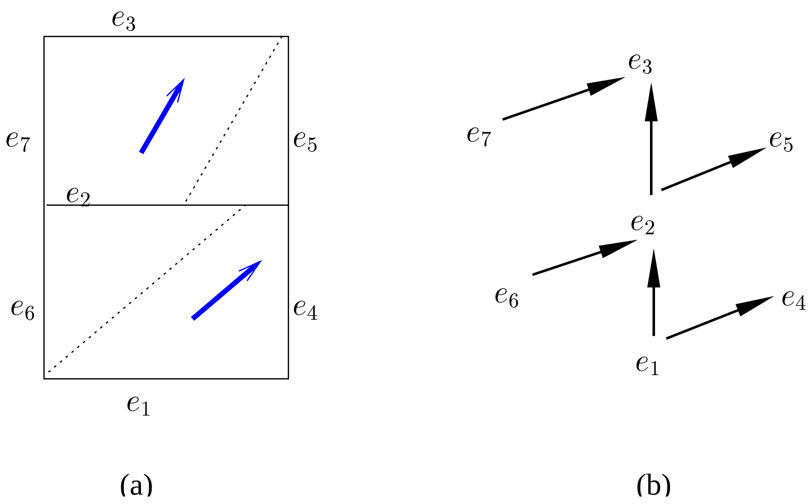


Figure 8.1: Path in $\mathcal{G}_{\mathcal{H}}$ does not imply existence of trajectory segments: (a) An SPDI; (b) Its graph.

any other edge. More formally,

Definition 51 *Given an SPDI $\mathcal{H}$, the* underlying graph of $\mathcal{H}$ *(or simply the* graph of $\mathcal{H}$*), is a graph $\mathcal{G}_{\mathcal{H}} = (N_{\mathcal{G}}, A_{\mathcal{G}})$, with $N_{\mathcal{G}} = \{e \mid e \in \mathcal{E}\}$ and $A_{\mathcal{G}} = \{(e, e') \mid \exists \xi, t \,.\, \xi(0) \in e \wedge \xi(t) \in e' \wedge \mathsf{Sig}(\xi) = ee'\}$. We say that a sequence $e_0 e_1 \ldots e_k$ of nodes in $\mathcal{G}_{\mathcal{H}}$ is a* path *whenever $(e_i, e_{i+1}) \in A_{\mathcal{G}}$ for $0 \leq i \leq k-1$.* ■

The following lemma shows the relation between edge signatures in an SPDI and paths in its corresponding graph.

Lemma 57 *Let ξ be a trajectory segment of $\mathcal{H}$ with edge signature $\mathsf{Sig}(\xi) = \sigma = e_0 \ldots e_p$, then σ is a path in $\mathcal{G}_{\mathcal{H}}$.*

Proof: Trivial. □

Remark. Notice that the converse of the above lemma is not true in general. See counter-example of Figure 8.1, where there exists a path from node e_1 to e_3, but it does not exist a trajectory segment form edge e_1 to edge e_3 on the SPDI.

Lemma 58 *Let $\sigma = e_0 \ldots e_p$ be a feasible signature, then σ is a path in $\mathcal{G}_{\mathcal{H}}$.*

Proof: Trivial. □

8.1.2 Properties of types of feasible signatures

On section 4.5 we have proved that types of feasible signatures, generated by Algorithm $\mathcal{A}$, satisfy properties $\mathbf{P_1}$ and $\mathbf{P_2}$. These properties were important since they guarantee the finitness of the set $\mathcal{T}_{feasi}$ of type of feasible signatures.

Given a graph $\mathcal{G}$, we define the following two predicates:

$$\mathsf{Seq}(e_1, \cdots, e_k) \text{ iff } (e_i, e_{i+1}) \in A_{\mathcal{G}} \,\wedge\, e_i \neq e_j$$

for $1 \leq i \neq j \leq k$, which determines that $e_1, \cdots, e_k$ is a sequential path on $\mathcal{G}$ and

$$\mathsf{Loop}(w) \text{ iff } \mathsf{Seq}(w) \,\wedge\, (last(w), first(w)) \in A_{\mathcal{G}} \,\wedge\, |w| > 3$$

that defines the simple loops of $\mathcal{G}$.

In the following lemma we prove more properties of types of feasible signatures. In fact, having shown that for any feasible signature of an SPDI there exists a path on the corresponding underlying graph (Lemma 58) we will abuse notation and in what follows the expressions "feasible signatures" and "paths" will be overlapped[1]. This overlapping of notions is done intensionally since we want to apply Algorithm $\mathcal{A}$ (see section 4.3) not to feasible signatures but for paths on the graph that correspond to feasible signatures. We have then that the following holds.

Lemma 59 *Let $\sigma = e_0 \ldots e_p$ be path on a graph $\mathcal{H}_{\mathcal{G}}$ (corresponding to a feasible signature on $\mathcal{H}$), then the type of signature $\mathtt{type}(\sigma) = r_1, s_1, \ldots, r_n, s_n, r_{n+1}$ (generated by algorithm $\mathcal{A}$) satisfies the following properties.*

$\mathbf{P_1}$ *For every $1 \leq i \neq j \leq n+1$, r_i and r_j are disjoint;*

$\mathbf{P_2}$ *For every $1 \leq i \neq j \leq n$, s_i and s_j are different;*

$\mathbf{P_3}$
1. *For $1 \leq i \leq n+1$, $\mathsf{Seq}(r_i)$;*
2. *For $1 \leq i \leq n$, $\mathsf{Loop}(s_i)$;*

$\mathbf{P_4}$
1. *For $1 \leq i < n$, $(last(s_i r_{i+1}), first(s_{i+1})) \in A_{\mathcal{G}}$ and if $r_1 \neq \varepsilon$ then $(last(r_1), first(s_1)) \in A_{\mathcal{G}}$;*
2. *For $1 \leq i < n$, $(last(s_i), first(r_{i+1} s_{i+1})) \in A_{\mathcal{G}}$;*

$\mathbf{P_5}$ *For every $1 \leq i < n$, s_i and r_{i+1} are disjoint;*

$\mathbf{P_6}$ $r_{n+1} \neq \varepsilon$;

$\mathbf{P_7}$ *For every $1 \leq i \leq n$, s_i is never a suffix of r_i;*

$\mathbf{P_8}$ *For $1 < i \leq n$, if r_i is a suffix of s_i then any suffix of s_{i-1} is different from any prefix of s_i;*

[1] Obviously, paths that corresponds to feasible signatures must be considered.

$\mathbf{P_9}$ *If v is a vertex appearing in* $\texttt{type}(\sigma)$*, then it can only occur exactly once in r_i for some $1 \leq i \leq n+1$ in σ.*

Proof: Properties $\mathbf{P_1}$ and $\mathbf{P_2}$ were already proved in Lemma 20.

$\mathbf{P_3}$ These properties are valid by construction;

$\mathbf{P_4}$ The result holds by definition of SPDI and of the corresponding graph $\mathcal{G}_{\mathcal{H}}$;

$\mathbf{P_5}$ Suppose that there exists $e \in r_{i+1}$ such that $e \in s_i$. But this is not possible by construction of σ;

$\mathbf{P_6}$ By construction, r_{n+1} has at least one edge;

$\mathbf{P_7}$ Again, the result follows by construction;

$\mathbf{P_8}$ Given $\sigma = e_0 \ldots e_p$ we consider $\sigma_{\mathcal{A}} = r_1 s_1^{k_1} \ldots r_n s_n^{k_n} r_{n+1}$ (obviously with type $r_1\ s_1\ \ldots\ r_n\ s_n\ r_{n+1}$). Suppose that for some $1 < i \leq n$ there exists a suffix of s_{i-1} equal to a prefix of s_i and such that r_i is equal to a suffix of s_i, then s_i is equal to a suffix of s_{i-1} followed by r_i ($s_i = \texttt{suffix}(s_{i-1}) r_i$). But this contradicts the construction of s_i since k_i is not the maximum number of times s_i is repeated.

$\mathbf{P_9}$ Notice first that v cannot occur in s_i for any $1 \leq i \leq n$ since this would mean that the trajectory segment crosses itself. Suppose now that v occurs at r_i for some $1 \leq i \leq n$, then by Lemma 19 it cannot appear later on the signature. □

We claim that the decomposition obtained by algorithm $\mathcal{A}$ is *canonical* in the sense that all these properties really characterize the set of types of feasible signatures generated by Algorithm $\mathcal{A}$.

Conjecture 1 *If $\sigma = e_0 \ldots e_n$ is a feasible signature then it admits a unique decomposition satisfying properties* $\mathbf{P_1}$ *to* $\mathbf{P_9}$. □

If the above conjecture is valid, this canonicity would guarantee that any other algorithm that generates a set of type of signatures satisfying properties $\mathbf{P_1}$ to $\mathbf{P_9}$ is equivalent to $\mathcal{A}$. This would be interesting from the practical point of view, since the construction given by $\mathcal{A}$ is not an efficient one, and certainly more efficient ways of generating the set of type of feasible signature could be found. Indeed, SPeeDI implements a different algorithm from $\mathcal{A}$ and the types of signatures generated satisfy properties $\mathbf{P_1}$-$\mathbf{P_7}$ (see section 8.3.4).

8.2 Description of the Tool

In this section we outline the main features of the tool and we describe its main utilities.

Visualization aids: To help visualize systems, the tool can generate graphical representations of the SPDI, and particular trajectories and signatures within it.

Information gathering: SPeeDI calculates edge-to-edge successor function composition and enlist signatures going from one edge to another.

Verification: The most important facet of the tool suite is that of verification. At the lowest level, the user may request whether, given a signature (with a possibly restricted initial and final edge), it is a feasible one or not. At a more general, and useful level, the user may simply give a restricted initial edge and restricted final edge, and the tool attempts to answer whether the latter is reachable from the former.

Trace generation: Whenever reachability succeeds SPeeDI generates stripes of feasible trajectories using different strategies and graphical representation of them.

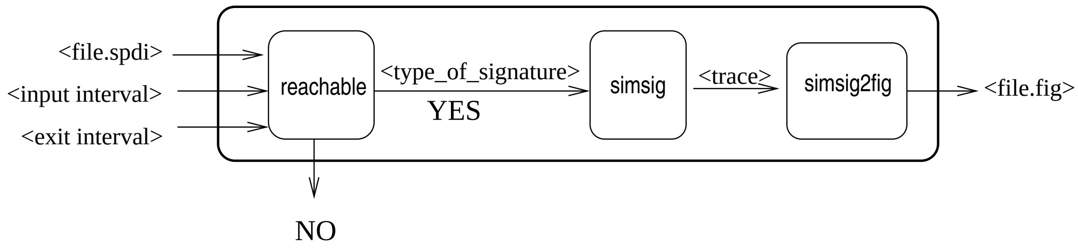


Figure 8.2: Workflow of the tool.

This typical usage sequence of the tool suite is captured in Figure 8.2.

Figure 8.3 illustrates a typical session of the tool on an example SPDI composed of 63 regions. The left part of the diagram shows selected portions of the input file, defining vectors, named points on the x-y plane, and regions (as sequences of point names, and pairs of differential inclusion vectors). The lower right-hand panel shows the signature generated by the tool `reachable` which satisfies the user's demand. The signature has two loops which are expressed with the star symbol. A trace is then generated from the signature using `simsig`. It traverses three times the first loop and two times the second one. The graphical representation of the SPDI and the trace is generated automatically using `simsig2fig`. The execution time for this example is a few seconds.

See Figure 8.5 for a succinct description of the different utilities of the tool. A more detailed explanation is given in Appendix C.

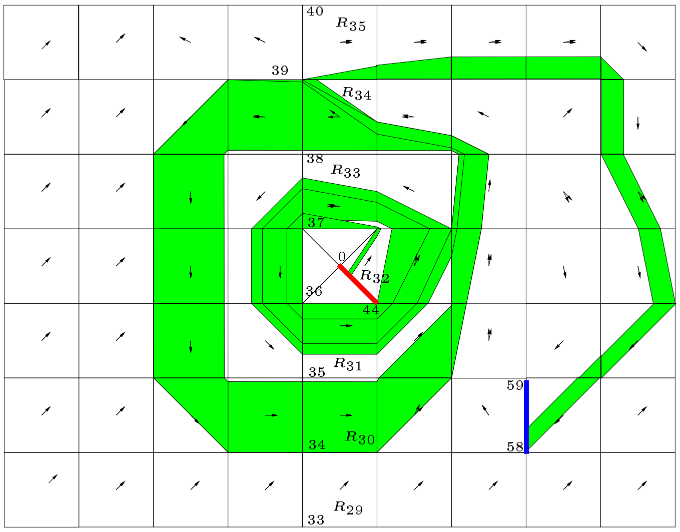
8.3 Implementation Issues

SPeeDI is implemented in Haskell and consist of the utilities described in the previous section plus a library for intervals, vectors and truncated affine multi-valued functions.

Input file

```
Points:
0. 0.0, 0.0
* ...
33. -5.0, -35.0
34. -5.0, -25.0
35. -5.0, -15.0
36. -5.0, -5.0
37. -5.0, 5.0
38. -5.0, 15.0
39. -5.0, 25.0
* ...
Vectors:
* ...
v3. -1,0.1833333333
v8. 1,0
v9. 1,1
v12. 1, 1.5
v20. -1, 0.001
v22. 1,-0.001
v25. -1,0.7
v28. 1, 0.001
*...
Regions:
* ...
* R29
33 ? 41 ! 42 ! 34 ? 33, v9, v9
* R30
34 ! 42 ! 43 ? 35 ? 34, v22, v22
* R31
35 ? 36 ? 0 ! 44 ! 43 ! 35, v8, v8
* R32
44 ! 45 ! 0 ? 44, v12, v12
* R33
0 ? 45 ? 46 ! 38 ! 37 ! 0, v3, v20
* R34
38 ? 46 ? 47 ! 39 ! 38, v25, v20
* ...
```

Generated Figure



Session log

```
% reachable example.spdi "[1,2]" "[0,10]" 0-44 59-60
Generating and trying signatures from edge 0-44 to 58-59
Starting interval:[1.0,2.0] Finishing interval:[0.0,10.0]

(0-44,45-44)  (45-53,45-46,37-38,...,36-35,44-43,44-52)*
(53-52,53-61,54-62,54-55,46-47)(38-39,..., 46-47)* (39-47,
...,67-59,58-59)    <REACHABLE>
```

Figure 8.3: Example

8.3.1 Input language

As shown in Figure 8.3, the input file consists of three parts: description of points, description of vectors and description of regions. For a more detailed syntax of the input file, see Appendix D.

8.3.2 Input Validation

Besides the obvious syntax validation, SPeeDI has the following consistency validations on the input file:

1. Regions must be well defined polygons;
2. Vectors corresponding to a region differential inclusion must respect the fact that the

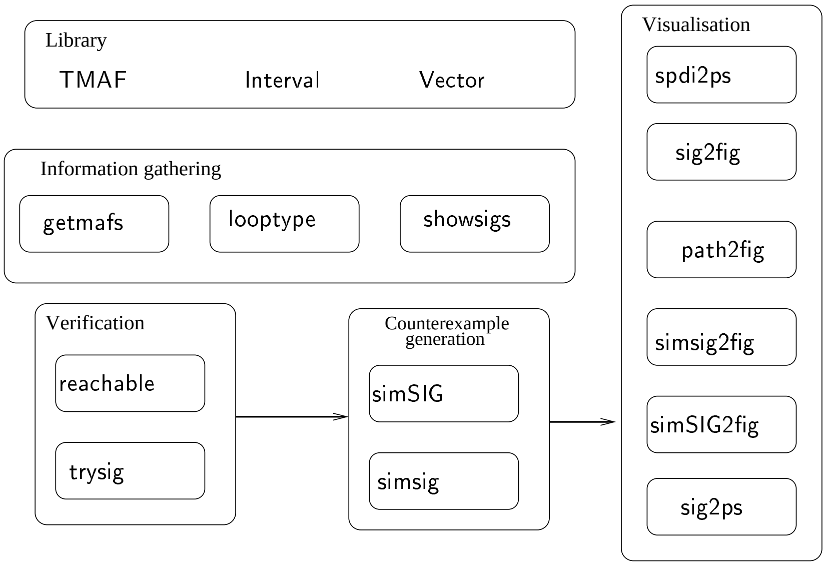


Figure 8.4: General architecture of the tool.

`<a-vector>` corresponds to **a** and `<b-vector>` corresponds to **b**, i.e., that $\hat{\mathbf{a}}\,\mathbf{b} < 0$;

3. Each region is *good* (see section 2.4);

8.3.3 Data structures

In functional languages, the underlying model of computation is the notion of function. In Haskell [60] the built-in types are integers, floating point numbers, characters, booleans, functions, lists, strings and tuples. All of them are used in our implementation and in what follows we explain the data structure used in order to define SPDI.

An SPDI $\mathcal{H}$ can be represented as a graph $\mathcal{G}_{\mathcal{H}}$. Indeed, given $\mathcal{H}$, we can define a graph $\mathcal{G}_{\mathcal{H}}$ where nodes correspond to edges of $\mathcal{H}$ and such that there exists an arc from one node to another if there exists a trajectory segment from the first edge to the second one without traversing any other edge. $\mathcal{G}_{\mathcal{H}}$ is defined in Haskell as a list of edges identifiers and a transition function that associate to each pair of edges its TAMF if it exists or “Nothing” otherwise.

The graph is defined then in SPeeDI as:

```
data Graph =                                                (1)
  Graph {                                                   (2)
    transitionFunction :: EdgeId -> EdgeId -> Maybe TAMF,  (3)
    domain :: [EdgeId]                                      (4)
  }
```

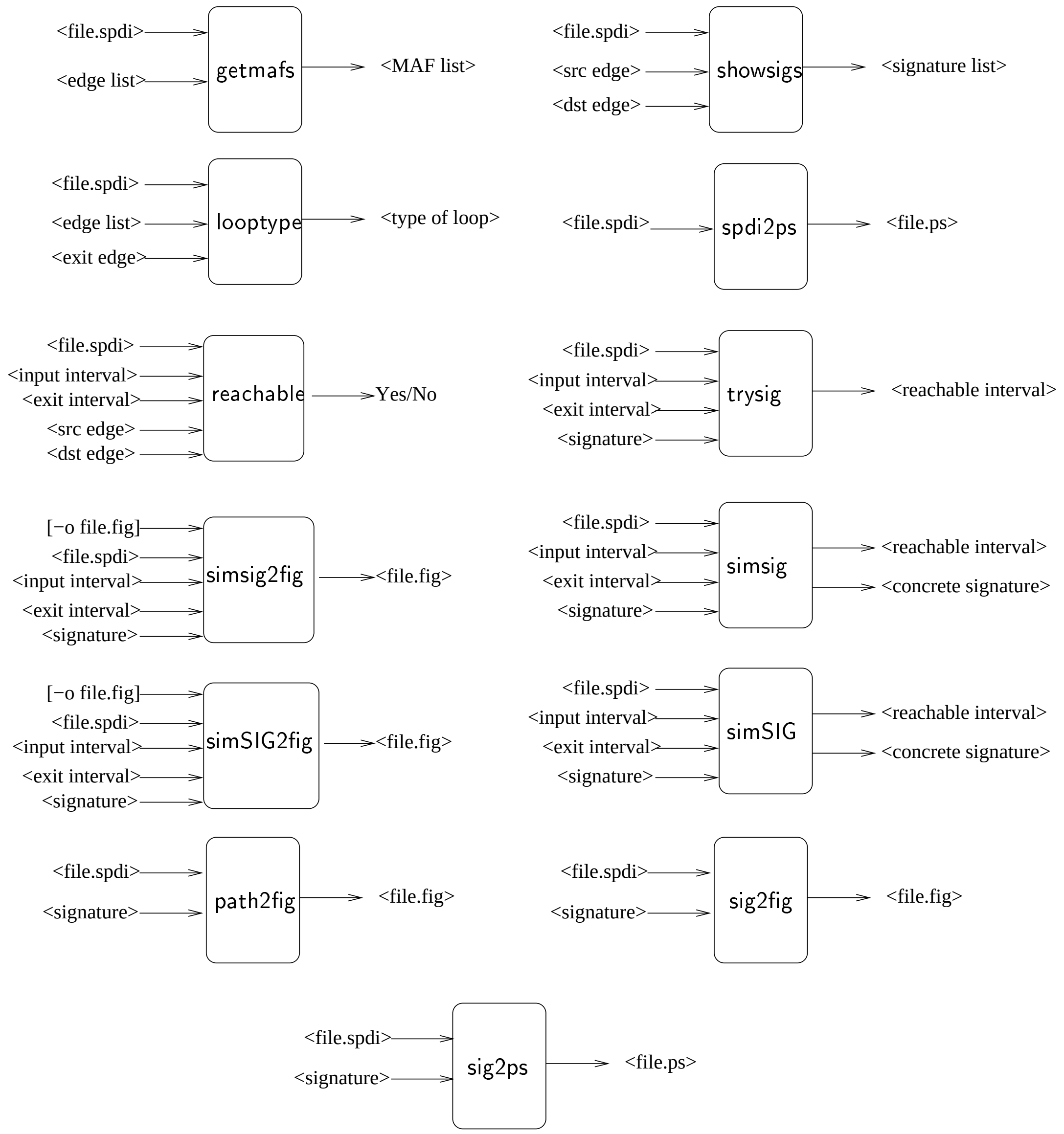


Figure 8.5: Description of the utilities.

Line (1) is the name of the data-type whereas in line (2) `Graph` is the type constructor (we chose it to be equal to the data-type name). As explained before, the graph then by a `domain` that is a list of edges identifiers and a `transitionFunction` that is a function that given two edges identifiers gives the corresponding TAMF if it exists (i.e. if the second edge can be reached in one step from the first one in the SPDI) and `Nothing` otherwise. This is defined by the type `Maybe`.

8.3.4 Generation of Types of signatures

Given two intervals $I_0 \in e_0$ and $I_f \in e_f$ SPeeDI generates all the types of signatures $r_1, s_1, \cdots, r_n, s_n, r_{n+1}$ that satisfy the following properties:

1. $first(r_1) = e_0$ and $last(r_{n+1}) = e_f$;
2. For every $1 \leq i \neq j \leq n+1$, r_i is a path on the graph;
3. For every $1 \leq i \neq j \leq n$, s_i is a simple loop on the graph;
4. For every $1 \leq i \neq j \leq n+1$, r_i and r_j are disjoint;
5. For every $1 \leq i \neq j \leq n$, s_i and s_j are different;
6. For every $1 \leq i < n$, s_i and r_{i+1} are disjoint;
7. For every $1 \leq i \leq n$, s_i is never a suffix of r_i;

The first property guarantees that only signatures from the initial edge to the final one are generated. The next two properties are natural properties: only types of signatures of the form obtained by $\mathcal{A}$ are generated. These properties correspond to $\mathbf{P_1}$ of Lemma 59. The fourth and fifth conditions are properties $\mathbf{P_1}$ and $\mathbf{P_2}$ of Lemma 20 (see section 4.5). The last two conditions are in fact properties $\mathbf{P_5}$ and $\mathbf{P_7}$ respectively described in Lemma 59.

8.3.5 Optimizations

In this section we describe the optimizations done in order to minimize the number of types of signatures analyzed for reachability. The following optimizations are implemented on the current version of the tool.

1. Elimination of some types of infeasible signatures: we just consider trajectories that have a TAMF. It can be the case that there is no trajectory segment from one edge to other of the same region even though there is a *path* on the graph. This is detected on SPeeDI checking that the `transitionFunction` for the two given edges gives a TAMF and not `Nothing`;

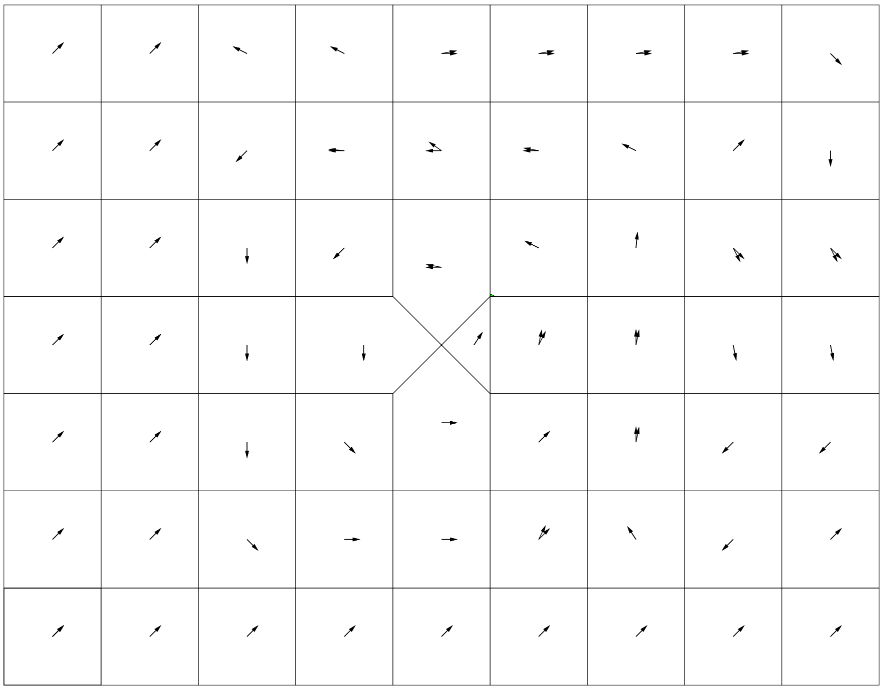


Figure 8.6: The SPDI of Example 8.4.

2. (Recursive) elimination of the minimal elements of the graph different from the source node src. When considering reachability from edge e to edge e', clearly any minimal element on the graph cannot be reachable from e;

3. As in the previous point, we do the same for the maximal elements and the destination node dst.

8.4 Example

In this section we present an example of an SPDI and the application of the different utilities explained before.

8.4.1 Presentation of the example

The SPDI we are going to consider has 63 regions and 162 edges as shown in Figure 8.6. We are not going to write down the details of the definition of the example; the input file .spdi is shown in Appendix E.

8.4.2 Information gathering

To see the affine functions (edge-to-edge successors) of a sequence of edges and the composed successors, we use getmafs as follows:

getmafs example1.spdi "0-44,44-45,45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52"

We present here just the output for the first three edges:

```
The requested AMFs:
From edge 84 (0-44) to edge 86 (44-45):
AMF is [1.7677669529663687x-2.5, 1.7677669529663687x-2.5]
   (accumulated AMF is [1.7677669529663687x-2.5, 1.7677669529663687x-2.5])
From edge 86 (44-45) to edge 103 (45-53):
AMF is [0.2x, 0.5x]
   (accumulated AMF is [0.35355339059327373x-0.5, 0.8838834764831843x-1.25])
From edge 103 (45-53) to edge 88 (45-46):
AMF is [0.5x, 0.5x]
   (accumulated AMF is [0.17677669529663687x-0.25, 0.44194173824159216x-0.625])
```

See Appendix F.2 for the complete answer as well as execution time.

Sometimes it can be useful to know the qualitative behavior of a cycle (see section 5.2). For example for the cycle shown in Figure 8.7,

looptype example1.spdi "45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52"

gives as a result 'Loop type: Exit right'.

A type of signature from one edge e_0 to another e_f is said to be feasible is there exists a path from e_0 to e_f on the symbolic graph. To see all the feasible type of signatures from edge 0-44 to edge 58-59 we execute

showsigs example1.spdi 0-44 58-59

that gives 36 feasible type of signatures, the first being

```
Generating normal signatures from edge 0-44 to 58-59 ...
   1. (0-44,45-44)(45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52)*
(53-52,53-61,54-62,54-55,46-4738-39,30-31,30-22,29-21,28-20,27-19,27-26,
35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,46-47)*(-47,48-47,56-55,
```

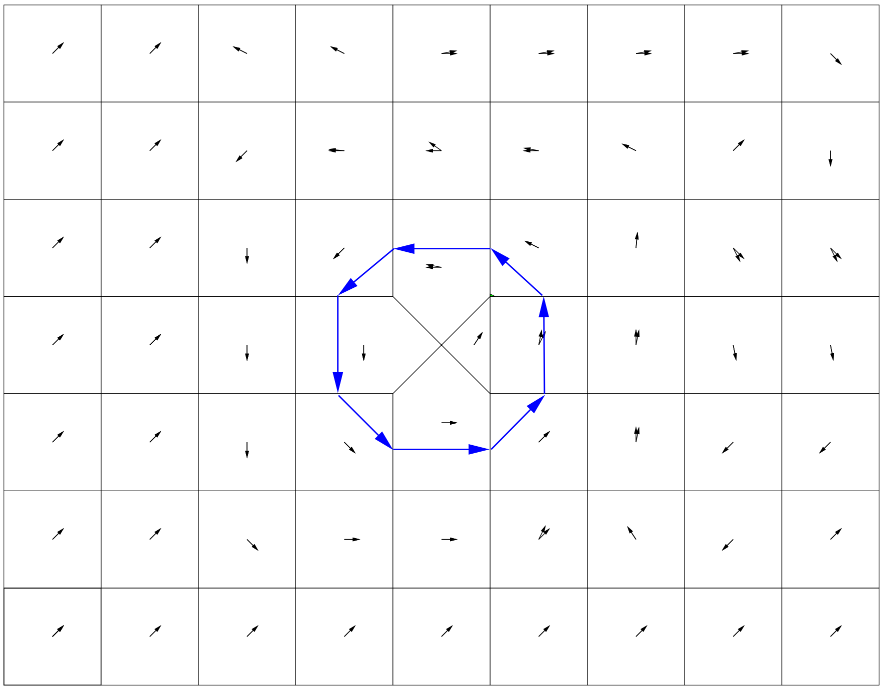


Figure 8.7: An edge cycle.

```
64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)
...
```

The full output is given in Appendix F.3.

8.4.3 Reachability

The type of signatures listed by showsigs are the candidates for the reachability question: Is $I_f \subseteq e_f$ reachable from $I_0 \subseteq e_0$? For example if we want to know whether edge 58-59 is reachable from interval $[1, 2]$ in edge 0-44 we execute

reachable example1.spdi "[1,2]" "[0,10]" 0-44 58-59

We obtain 'REACHABLE' as answer and the same signature as the first shown by showsigs. See Appendix F.4 to see the detailed output.

Given a type of signature (for example the answer of reachable) we can obtain the reachable interval on its last edge using trysig as follows:

```
trysig example1.spdi "[1,2]" "[0,10]"
        "0-44,45-44 [45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52]
```

```
53-52,53-61,54-62,54-55,46-47 [38-39,30-31,30-22,29-21,28-20,
27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,
46-47] 39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,
67-68,67-59,58-59"
```

that gives the interval [0.0,2.9519033333328584] on edge 58-59 as answer. Notice that in this type of signature we have two cycles but in order to compute trysig we don't need to iterate, since we use the algorithm of section 5.2 (that use acceleration of loops).

If we want to see a concrete signature (i.e. the corresponding signature with an unfolding of the cycles), we can apply simSIG that gives a list of pairs `<edges, interval>` that is the "trace" of the type of signature:

```
simSIG example1.spdi "[1,2]" "[0,10]"
         "0-44,45-44 [45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52]
          53-52,53-61,54-62,54-55,46-47 [38-39,30-31,30-22,29-21,28-20,
          27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,
          46-47] 39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,
          67-68,67-59,58-59"
```

The complete output of the simulation is presented in Appendix F.6 and it is shown in Figure 8.3. This picture was generated with simSIG2fig[2].

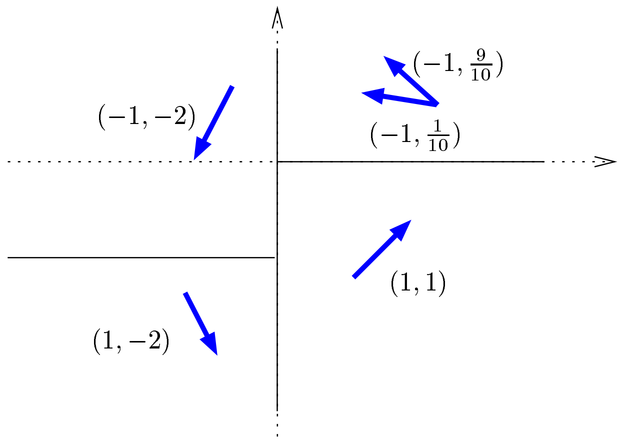


Figure 8.8: SPDI of Example 8.5.1.

[2]Numbers of edges and regions were added for pedagogical reasons.

8.5 Comparison with HyTech

While SPeeDI is, as far as we know, the only verification tool for hybrid systems implementing a decision algorithm (with the exception of timed automata), it is interesting to compare it to "semi-algorithmic" hybrid system verification tools such as HyTech [81, 79]. HyTech is a tool capable of treating hybrid linear systems of any dimension, making it much more general than SPeeDI, which is limited to two-dimensional systems without resets. On the other hand, SPeeDI implements acceleration techniques (based on the resolution of fix-point equations) which yield a complete decision procedure for SPDIs. Also, SPeeDI does not handle arbitrary polyhedra, but only polygons and line segments. For these reasons, comparing the performance of the two tools is meaningless and no fair benchmarking is really possible. However, we have explored a simple illustrative example.

8.5.1 Example

Consider the SPDI defined as follows (see Figure 8.8) with $I_0 \equiv (y = 0 \wedge x \in [3; 4])$ as initial region.

Region	*Defining conditions*	*Vector*
R_0	$(x \geq 0) \wedge (y \geq 0)$	$\mathbf{a} = (-1, \frac{9}{10}), \mathbf{b} = (-1, \frac{1}{10})$
R_1	$(x \leq 0) \wedge (y \geq -10)$	$\mathbf{a} = \mathbf{b} = (-1, -2)$
R_2	$(x \leq 0) \wedge (y \leq -10)$	$\mathbf{a} = \mathbf{b} = (1, -2)$
R_3	$(x \geq 0) \wedge (y \leq 0)$	$\mathbf{a} = \mathbf{b} = (1, 1)$

We consider different final points x_f on the x axis and try to answer the question: Is x_f reachable from I_0?

The experimental results are given in Table 8.5.1.

All the results above of HyTech were using the `reach backward` command. In all the cases the `reach forward` gives "`Library overflow error in multiplication`".

In order to understand these results, notice that as shows the exact analysis, the system, starting from the initial interval I_0 spirals as shown on Figure 8.9, that simulates the case whenever $x_f = \frac{201}{9}$ (the picture was obtained using `simsig2fig`). The intersection of the spiral with the x axis converges to the "fixpoint interval" $I^* = (\frac{200}{9}; 200)$. SPeeDI in fact computes this interval I^*, and whenever $x_f \in I^*$ it gives immediately the positive answer to the reachability question. If $x_f \geq 200$ SPeeDI says "no". The only case when it really computes successors is when x_f lies between I_0 and I^*.

Notice that the problems with HyTech are mainly whenever the final point I_f is close to the fixpoints ($l^* = \frac{200}{9}$ and $u^* = 200$) and of course whenever I_f is located in between the fixpoints or when $x_f \geq u^*$.

Final Point	*HyTech*	*SPeeDI*	*Reachable*
199	`overflow`	`0.05 sec`	Yes
200	`overflow`	`0.05 sec`	No
201	`overflow`	`0.01 sec`	No
210	`overflow`	`0.05 sec`	No
5	`0.04 sec`	`0.05 sec`	No
20	`0.07 sec`	`0.05 sec`	No
$\frac{200}{9}$	`0.10 sec`	`0.05 sec`	Yes
$\frac{201}{9}$	`overflow`	`0.03 sec`	Yes
$\frac{199}{9}$	`0.07 sec`	`0.04 sec`	Yes
$\frac{1}{2}$	`0.06 sec`	`0.05 sec`	No

Table 8.1: Comparison results with HyTech.

8.6 Summary

We have presented a prototype tool for solving the reachability problem for the class of polygonal differential inclusions. The tool implements the algorithm presented in chapter 5 which is based on the analysis of a finite number of qualitative behaviors generated by a discrete dynamical system characterized by positive affine Poincaré maps. Since the number of such behaviors may be extremely big, the tool uses several powerful heuristics that exploit the topological properties of planar trajectories for considerably reducing the set of actually explored signatures. When reachability is successful, the tool outputs a visual representation (in the form of an Xfig file) of the stripe of trajectories that go from the initial point (edge, polygon) to the final one.

SPeeDI was implemented in Haskell [89], a general-purpose, lazy, functional language [37, 70]. Despite the fact that functional languages, especially lazy ones, have a rather bad reputation regarding performance (see for example [100] for a report on the experiences of writing verification tools in functional languages), we found that the performance we obtained was more than adequate for the magnitude of examples we had in mind. Furthermore, we feel that with the gain in the level of abstraction of the code, we have much more confidence in the correctness of our tool had we used a lower level language. We found laziness particularly useful in separating control and data considerations. Quite frequently, optimizations dictated that we evaluate certain complex expressions at most once, if at all. In most strict languages, this would have led to complex code which mixes data computations (which use the values of the expressions) with control computation (to decide whether this is the first time we are using the expression and, if so, evaluate it). Thanks to shared expressions and laziness, all this came for free — resulting in cleaner code, where the complex control is not done by the programmer.

From some experiments we have run comparing HyTech with SPeeDI, we have reached a

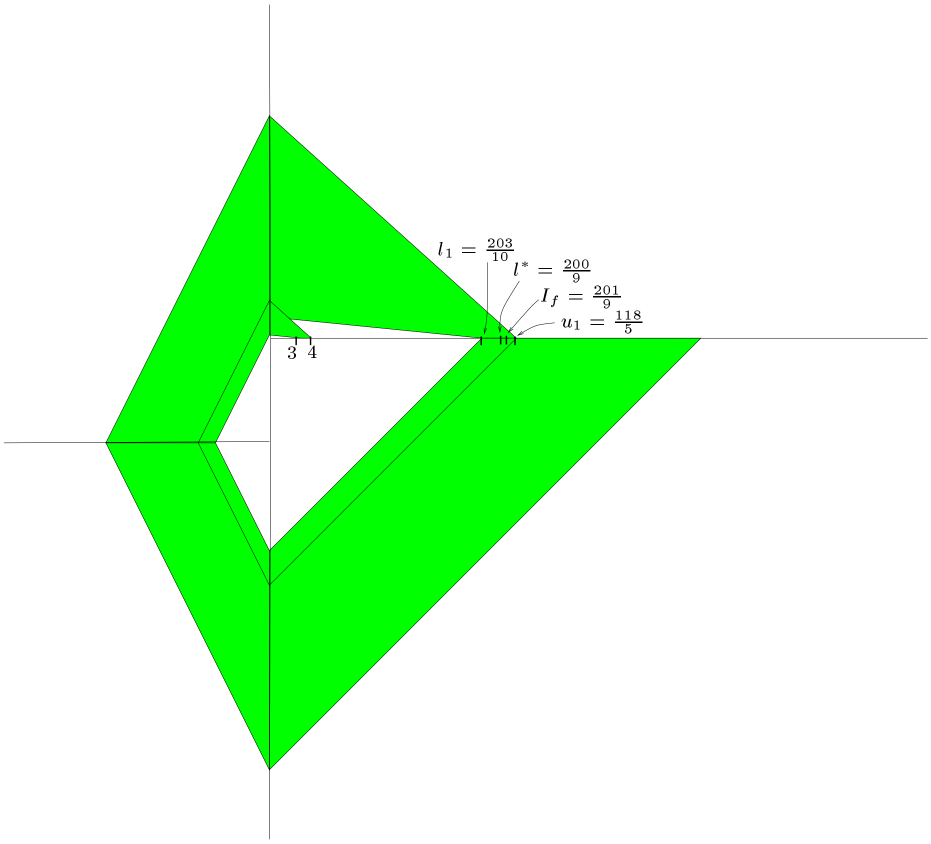


Figure 8.9: Simulation of reachability for $x_f = \frac{201}{9}$.

number of qualitative conclusions:

- It is well known that since HyTech uses exact rational arithmetic, it can easily run into overflow problems. This is particularly an issue when the path to the target passes through a large number of regions. This makes verification of non-trivial sized SPDIs (eg the one in figure 8.3) impossible.
- In the case of loops, SPeeDI calculates the limit interval without repeatedly iterating the loop. It makes use of this interval to accelerate the reachability analysis, avoiding time consuming loop traversals. In contrast, HyTech performs these iterations. Following the loops explicitly, easily leads to overflow problems, and, more seriously, in certain (even simple) configurations, this analysis never terminates.

While the first issue is limited to HyTech, the second is inherent to any tool based on non-accelerated reachability analysis. On examples which HyTech can handle, the two tools take approximately the same amount of time (a fraction of a second) to reach the result. SPeeDI, however, can handle much larger examples.

Chapter 9

Relaxing "Goodness"

In this chapter we show that in fact our *goodness* hypothesis is unnecessary and thus reachability is decidable for SPDIs without this condition. *General SPDIs* (GSPDIs) are SPDIs without the goodness restriction.

Organization of the chapter: In the first section we present the problem and we define some basic concepts. In the second section we prove that reachability is preserved after eliminating Input/Output edges and we make an informal discussion about the importance of the ordering between edges. In section three we give a "topological" proof of the results of chapters 4 and 5 that rely on the goodness condition and we show that reachability is decidable for GSPDIs. In the last section we summarize the chapter.

9.1 Preliminaries

The *goodness* restriction (see Definition 13 in section 2.4) was introduced in order to simplify treatment of trajectories since it guarantees, among other things, that each region can be partitioned into *entry* and *exit* edges in an ordered way. This good property can be lost when relaxing goodness. Indeed, without this condition there are edges that are neither of entry nor of exit as shown in the following example.

Example 33 In Figure 9.1, $In(R) = \{e_1, e_6\}$, $Out(R) = \{e_3, e_4\}$. Edges e_2 and e_5 are neither of entry nor of exit of R. ■

We define now *inout* edges.

Definition 52 *An edge $e \in P$ is an* inout *edge of P if e is neither an entry nor an exit edge of P.* ■

In what follows we define *general SPDIs*.

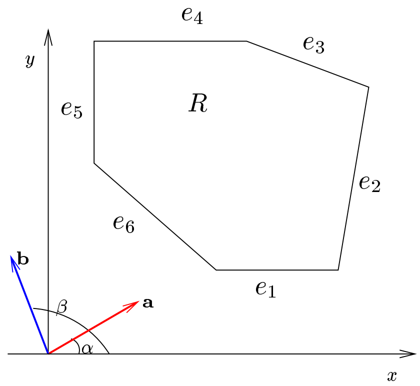


Figure 9.1: A region that does not satisfy goodness.

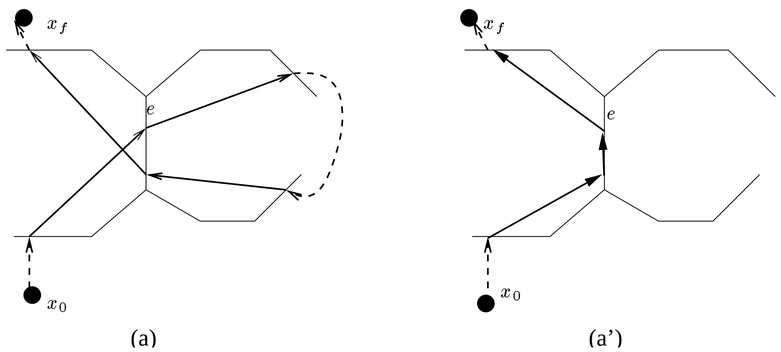


Figure 9.2: (a): A proper inout edge; (b): A sliding edge.

Definition 53 *An SPDI without the goodness restriction is called a* general SPDI (GSPDI). ■

Thus, in GSPDIs there are three kinds of edges: inouts, entries and exits.

We have seen how self-crossing of trajectory segments of SPDIs can be eliminated in section 4.2.2. The result still holds for GSPDIs, thus in what follows we will consider only trajectory segments without self-crossings.

Notice that on GSPDIs a trajectory can "intersect" an edge at an infinite number of points because it can *slide* at it. Thus, a trace is not anymore a sequence of points but rather a sequence of intervals.

Definition 54 *The* trace *of a trajectory ξ is the sequence* $\mathtt{trace}(\xi) = I_0 I_1 \ldots$ *of the intersection intervals of ξ with the set of edges, that is, $I_i \subseteq (\xi \cap \mathcal{E})$.* ■

A point interval $I = [\mathbf{x}, \mathbf{x}]$ will be sometimes written as $\mathbf{x}$ whenever no confusion might arise.

Definition 55 *An* edge signature *(or simply a* signature*) of a GSPDI is a sequence of edges. The* edge signature of a trajectory *ξ, $\mathsf{Sig}(\xi)$, is the ordered sequence of traversed edges by the trajectory segment, that is, $\mathsf{Sig}(\xi) = e_0 e_1 \ldots$, with $\mathtt{trace}(\xi) = I_0 I_1 \ldots$ and $I_i \subseteq e_i$. The* region signature *of ξ is the sequence $\mathsf{RSig}(\xi) = P_0 P_1 \ldots$ of traversed regions, that is, $e_i \in In(P_i)$.* ■

Notice that in many cases the intervals of a trace are in fact points. We say that a trajectory with edge signature $\mathsf{Sig}(\xi) = e_0 e_1 \ldots e_i \ldots$ and trace $\mathtt{trace}(\xi) = I_0 I_1 \ldots I_i \ldots$ *interval-crosses* edge e_i if I_i is not a point.

Given a trajectory segment, we will make the difference between *proper inout* edges and *sliding* edges.

Definition 56 *Let ξ be a trajectory segment from point $\mathbf{x}_0 \in e_0$ to $\mathbf{x}_f \in e_f$, with edge signature $\mathsf{Sig}(\xi) = e_0 \ldots e_i \ldots e_n$, and $e_i \in E(P)$ be an edge of P. We say that e_i is a* sliding edge of P for ξ *if ξ interval-crosses e_i, otherwise e is said to be a* properinout edge of P for *ξ.* ■

We say that a trajectory segment ξ *slides* on an edge e if e is a sliding edge of P for ξ and ξ is said to be a *sliding trajectory* if there is at least one sliding edge $e \in \mathsf{Sig}(\xi)$.

Example 34 In Figure 9.2-(a), e is a proper inout edge. Edge e on Figure 9.2-(b) is a sliding edge. ■

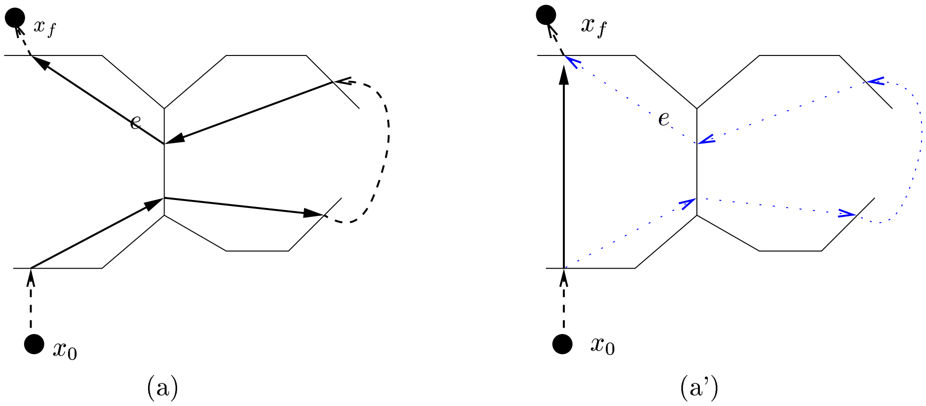


Figure 9.3: Inout case.

9.2 Simplification of Trajectory Segments

In this section we show how to simplify trajectory segments eliminating inout edges. We start by eliminating proper inout edges.

Lemma 60 *Let ξ be a trajectory segment $\mathbf{x}_0 \in e_0$ to $\mathbf{x}_f \in e_f$ with edge signature* $\mathsf{Sig}(\xi) = e_0 \ldots e_i \ldots e_n$. *If e_i is a proper inout edge then there exists a trajectory segment ξ' from $\mathbf{x}_0$ to $\mathbf{x}_f$ such that*

that traverses e_i in at most one sense (that is, e_i is either an entry or an exit, but no both).

Sketch of the Proof: There is only one case of proper inout edges for trajectory segments without self-crossings. This is illustrated in Fig. 9.3-(a) where edge e_i is a proper inout edge. After a straightforward algebraic vector manipulation, on the same lines of elimination of self-crossings - section 4.2.2, the trajectory segment shown in Fig. 9.3-(a') is obtained. $\square$

As a consequence we have the following proposition.

Proposition 7 (Existence of a trajectory without proper inout edges) *If there exists a trajectory segment from points $\mathbf{x}_0 \in e_0$ to $\mathbf{x}_f \in e_f$ then there always exists a trajectory segment, whose edge signature contains no proper inout edges, between them.*

Proof: By induction on the number n of proper inout edges of the signature of the trajectory segment using Lemma 60 in the induction step. $\square$

We show now how to eliminate sliding edges.

Lemma 61 *Let ξ be a trajectory segment $\mathbf{x}_0 \in e_0$ to $\mathbf{x}_f \in e_f$ with edge signature* $\mathsf{Sig}(\xi) = e_0 \ldots e_i \ldots e_n$. *If e_i is a sliding edge for ξ then there exists a trajectory segment ξ' from $\mathbf{x}_0$*

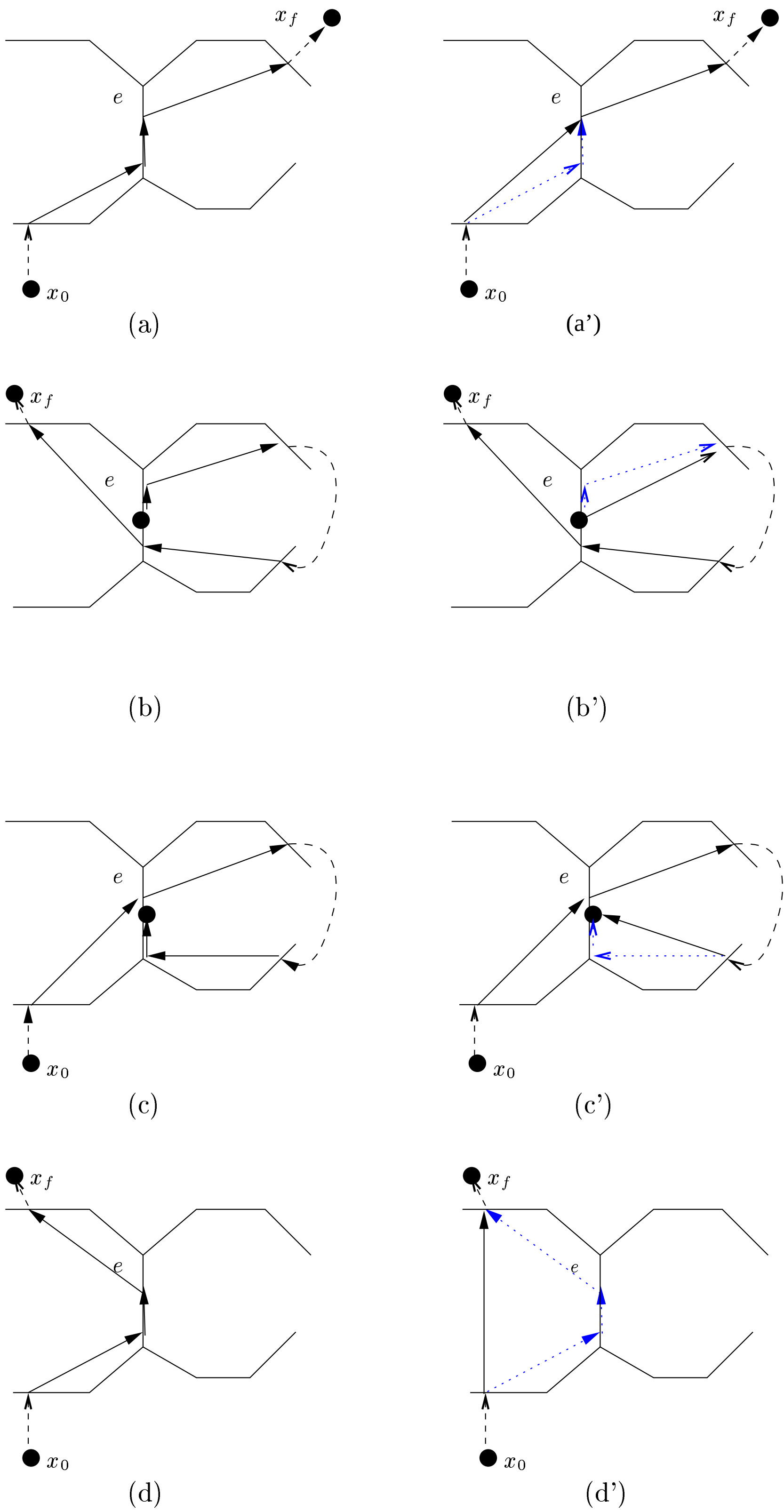


Figure 9.4: Sliding cases.

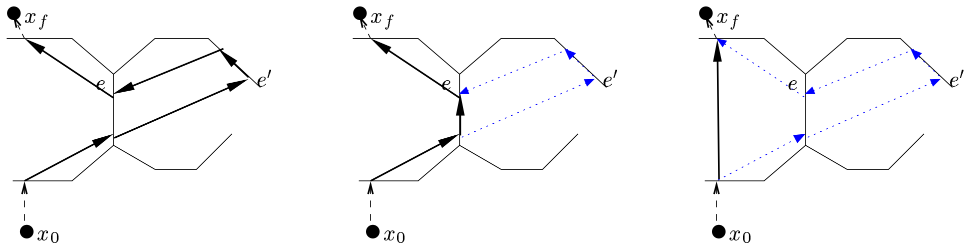


Figure 9.5: Elimination order of inout edges.

to $\mathbf{x}_f$ *that does not slide on edge* e_i.

Sketch of the Proof: Sliding edges can arise in four different cases; they are shown in Fig. 9.4-(a) to (d). The corresponding primed figures (Fig. 9.4-(a') to (d')) show the transformation done in order to avoid sliding on edge e. □

As a consequence we have the following result.

Proposition 8 (Existence of a non-sliding trajectory) *If there exists a sliding trajectory segment from points* $\mathbf{x}_0 \in e_0$ *to* $\mathbf{x}_f \in e_f$ *then there always exists a non-sliding trajectory segment between them.*

Proof: By induction on the number n of sliding edges of the signature of the trajectory segment using Lemma 61 in the induction step. □

Remark. Notice that the choose of eliminating first proper inout edges and next sliding, is not arbitrary. In fact, the number of sliding edges is not guaranteed to decrease if sliding edges are eliminated before proper inout edges. See the following example.

Example 35 In Figure 9.5-(a) a trajectory segment that slides at edge e' is shown. After eliminating the sliding at edge e', a new sliding edge is introduced (e). This is shown in Figure 9.5-(b). However, if proper inout edges are eliminated first, we do not introduce new proper inout edges as shown in part (c) of the same Figure. ■

Remark. We are not going to prove formally that the simplification of trajectory segments given before preserves reachability since this is done on the same lines as in chapter 4 for SPDIs.

About the ordering between edges. We finish this section with an informal discussion about the importance of the "contiguous" order between entry and exit edges on SPDIs.

We have seen that in SPDIs edges of a region can be bipartitioned into entry and exit edges in a "contiguous" way (see Fig. 9.9) having as a consequence an ordering between edges. As explained above, this is not longer the case in GSPDIs.

First of all, notice that the ordering of edges on an SPDI were chosen in order to preserve the "positive affinity" (and hence the monotonicity) of the successor functions. Given a region R with differential inclusion $\angle_{\mathbf{a}}^{\mathbf{b}}$, let e be an entry edge and e_1 and e_2 two exit edges of R. For e we chose the direction (given by a director vector $\mathbf{e}$) that satisfies the inequality $\hat{\mathbf{a}}\ \mathbf{e} > 0$ (see Figure 9.8). The same for e_1 and e_2. As a consequence we obtain an ordering like in Figure 9.9.

Notice that on a GSPDI (see Figure 9.6(a)), the property that for any edge e, $\hat{\mathbf{a}}\ \mathbf{e} > 0$ is not longer valid since an edge can be of entry and of exit and then the ordering can change. In spite of that, once an inout edge is "converted" into an entry (or exit) then we can have the notation of considering the ordering of entry edges going counter-clockwise and clockwise for exit edges (see Figure 9.6(b)).

It is important to notice that even though the definition of edge and region signatures as well as edge cycle continue to hold, it is not the case for region cycle. In fact we can have a region signature $P_1 \cdots P_i \cdots P_k P_1$ that it is not a region cycle. The reason is that in GSPDIs a trajectory can enter a region through two different edges without forming a cycle.

Thus we have that a region signature $P_1 \cdots P_i \cdots P_k P_1$ is a *region cycle* if the edge signature $e_1 \cdots e_k e_1$, with $e_i \in Out(P_i)$ for all $1 \leq i \leq k$, forms an edge cycle.

In Figure 9.7 the following is a region cycle: $P_1 P_2 P_3 P_4 P_2 P_5 P_1$. Notice that $P_2 P_3 P_4 P_2$ is region cycle for SPDIs but not for the given GSPDI.

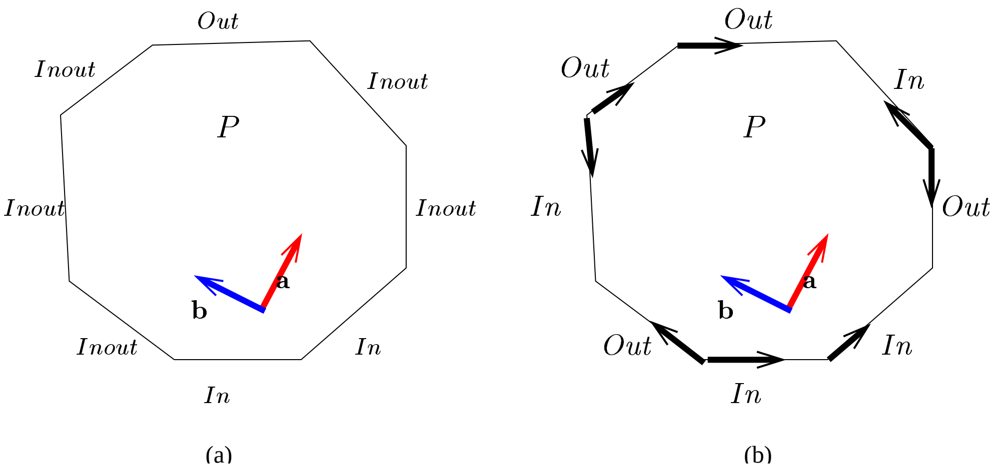


Figure 9.6: (a) A GSPDI; (b) Ordering after fixing input and output edges.

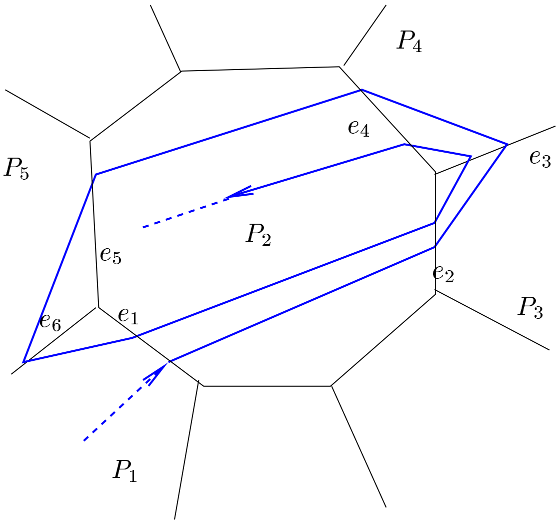


Figure 9.7: A region cycle.

9.3 Reachability Analysis for GSPDIs

In this section we rephrase “topologically” the results of chapters 4 and 5, that used the “contiguity” between entry and exit edges in their proofs, and we prove them. We also re-prove soundness of Exit-LEFT and Exit-STAY algorithm and at the end we show that reachability is decidable for GSPDIs.

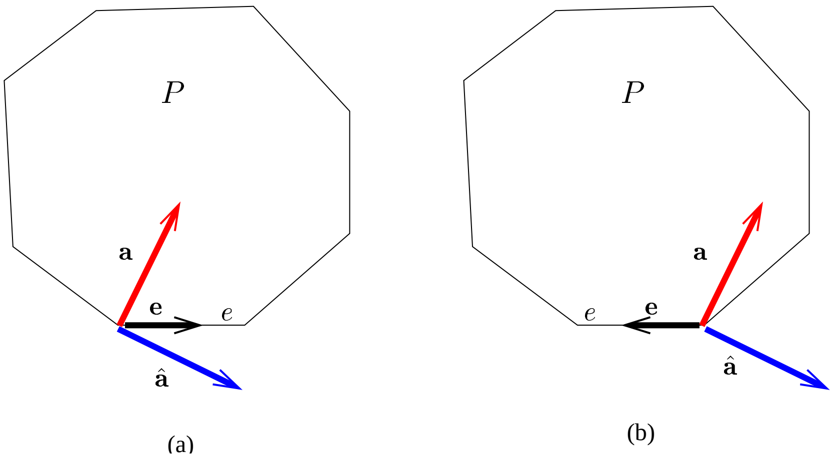


Figure 9.8: (a) $\hat{\mathbf{a}}\,\mathbf{e} > 0$; (b) $\hat{\mathbf{a}}\,\mathbf{e} < 0$.

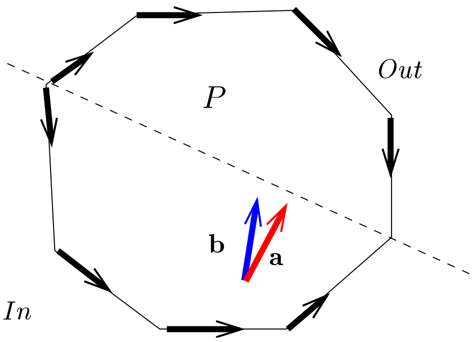


Figure 9.9: Ordering of edges on an SPDI (all the edges e satisfy $\hat{\mathbf{a}}\ \mathbf{e} > 0$).

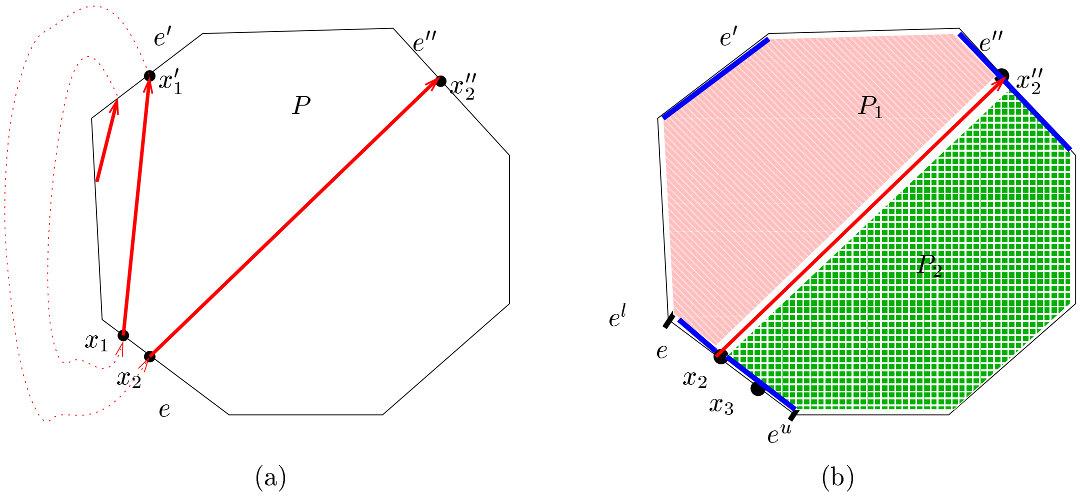


Figure 9.10: (a): Simple cycle s_i and its continuation through edge e; (b) Edge e' cannot be reached from point x_3 without intersecting $\overline{\mathbf{x}_2\mathbf{x}_2''}$

9.3.1 Proof of Lemma 20, Lemma 26 and Corollary 27

The only results that use the "contiguity" order between entry and exit edges are Lemmas 20 (see section 4.5), 26 and Corollary 27 (see section 5.1). We prove these results without using the order between entry and exit edges.

Recall Lemma 20 for property $\mathbf{P_2}$:

Lemma 20' *Let $\sigma = e_0 \ldots e_p$ be a feasible signature, then its type, $\mathtt{type}(\sigma) = r_1, s_1, \ldots, r_n, s_n, r_{n+1}$ satisfies the following property, $\mathbf{P_2}$: For every $1 \leq i \neq j \leq n$, s_i and s_j are different.*

Proof: In order to prove property $\mathbf{P_2}$ we prove that, given a simple cycle $s_i = e', \ldots, e$, the sequence of edges ee' cannot occur after leaving s_i (hence it cannot occur in any other simple cycle s_j, with $1 \le i < j \le n$). After cycling k_i times cycle s_i is abandoned by edge e (guaranteed by construction). Let P be a region s.t. $e \in In(P)$ and consider the unfolding of the last iteration and its continuation (see Fig. 9.10-(a)):

$$\ldots, e, e', \ldots, e, e'', \ldots$$

where $e'' = first(r_{i+1})$, $e \in In(P)$ and $e', e'' \in Out(P)$ ($e' \neq e''$). Let x_2 be the last point visited on edge e before leaving cycle s_i and x_2'' be the first point on edge e'' after leaving s_i (see Fig. 9.10-(b)). Segment $\overline{\mathbf{x}_2\mathbf{x}_2''}$ of the trajectory segment divides region P into two subregions P_1 and P_2 and edge e into two segments $\overline{e^l x_2}$ and $\overline{x_2 e^u}$. By the non-crossing hypothesis (and monotonicity on edges) after leaving s_i the only accessible part of edge e is the segment $\overline{x_2 e^u} \in e$. By Jordan's curve theorem the only way to reach edge e' from any point in $\overline{x_2 e^u} \in e$ is by crossing $\overline{\mathbf{x}_2\mathbf{x}_2''}$ or by crossing one of the edges of region P_2. The first case is not possible since it would contradict the hypothesis of non-crossing trajectory and in the second one the sequence ee' would not belong to the trajectory segment. □

In what follows we use the following notation: whenever we partition the space into two regions P_L and P_R by the line defined by a segment of line $\overline{xy}$, P_L is the semi-space of all the points that are a left rotation of $\vec{xy}$ and P_R is the semi-space corresponding to the points that are a right rotation of the same vector. With $f(x) \downarrow$ we mean that f is defined at x and $f(x) \uparrow$ will mean that f is undefined at x.

Next we will (topologically) rephrase Lemma 26 and Corollary 27 and we prove them both.

Lemma 26' *Let P be a region, $e \in In(P)$, $e_1, e_2 \in Out(P)$, $\langle l_i, u_i \rangle$ be any subinterval of $\langle e_i^l, e_i^u \rangle$ and $f_i(x) = F^{\mathbf{c}}_{e,e_i}(x)$.*

1. *Let P be partitioned into two regions P_L and P_R by the line defined by $\overline{xl_1}$, then the following holds: if $e_2 \in P_L$, $f_2(x) \downarrow$ and $l_1 < f_1(x)$ then $u_2 < f_2(x)$;*

2. *Let the plane be partitioned into two subspaces P_L and P_R by the line defined by $\overline{xl_2}$, then the following holds: if $e_1 \in P_R$, $f_1(x) \downarrow$ and $f_2(x) < u_2$ then $f_1(x) < l_1$.*

Proof:

1. Remember that the line defined by e_2 is ordered and that u_2, A and $f_2(x)$ belongs to it. We have then that $e_2 \in P_L$ (and hence $u_2 \in P_L$) and that $f_2(x) \in P_R$ (by construction of the partition). We have then that $u_2 < A$ and $A < f_2(x)$, that implies $u_2 < f_2(x)$. See Figure 9.11(a).

2. This case is symmetric to the previous one. □

Corollary 27' *Let P be a region, $e \in In(P)$, $e_1, e_2 \in Out(P)$, $f_i(x) = F^{\mathbf{c}}_{e,e_i}(x)$ be an affine function and $\mathcal{F}_i(\langle x, y \rangle) = F_i(\langle x, y \rangle \cap S_i) \cap J_i$ be a truncated affine multi-valued function (with $F_i = [f_i^l, f_i^u]$ and $J_i = \langle L_i, U_i \rangle$).*

1. *Let P be partitioned into two regions P_L and P_R by the line defined by $\overline{xL_1}$, then the following holds: If $e_2 \in P_L$ and $L_1 < f_1^l(x)$ then $\mathcal{F}_2(\langle x, y \rangle) = \emptyset$;*

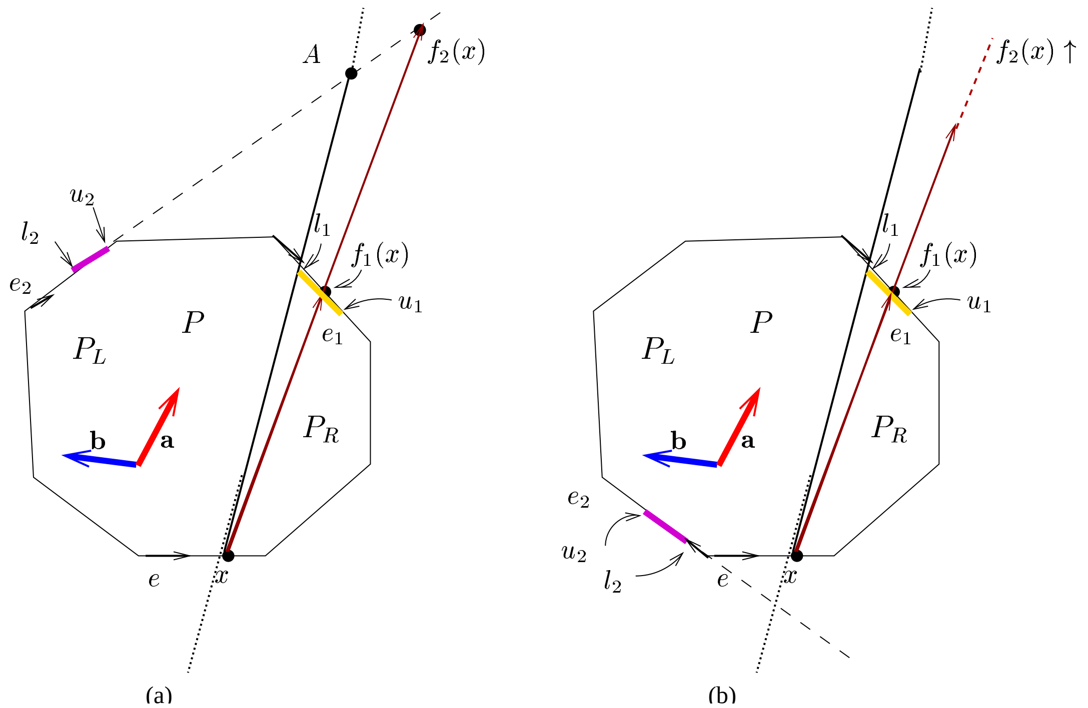


Figure 9.11: Lemma 26'-1. (a) When $f_2^l(x) \downarrow$; (b) The case $f_2^l(x) \uparrow$.

2. *Let P be partitioned into two regions P_L and P_R by the line defined by $\overline{xL_2}$, then the following holds: if $e_1 \in P_R$ and $f_2^u(y) < U_2$ then $\mathcal{F}_1(\langle x, y\rangle) = \emptyset$.*

Proof:

1. If $f_2^l(x)$ is undefined, then it is obvious that $\mathcal{F}_2(\langle x, y\rangle) = \emptyset$. If $f_1^l(x)$ is defined, then the result follows directly from Lemma **??**-1 and definition of $\mathcal{F}_i(\langle x, y\rangle)$.

2. Symmetric to the above case using Lemma **??**-2. □

9.3.2 Soundness of Exit-STAY and Exit-LEFT

We prove now soundness of the Exit-STAY and Exit-LEFT algorithm whose proofs rely on the results proved in the previous section.

Let $A = \mathsf{Succ}_s^{\mathbf{b}}(L)$ and consider the line defined by $\overline{AL}$. This line partition the space into P_L and P_R as before.

Exit-STAY

function $Exit_{STAY}(I, s, ex)$ $\longleftarrow \emptyset$

Soundness By hypothesis, $L < l^* < u^* < U$. Hence, for all i, $\tilde{I}_i = \langle \tilde{l}_i, \tilde{u}_i \rangle \subseteq \langle L, U \rangle$, hence $I_i = \tilde{I}_i$ and by Corollary 9.3.1 we have that $\mathsf{Succ}^i_{s,ex}(I) = \emptyset$.

Termination Trivial. $\Box$

Exit-LEFT:

function $Exit_{LEFT}(I, s, ex)$ $\longleftarrow \mathsf{Succ}_{s,ex}(\mathsf{Succ}_{s,f}(\langle L, \max\{u, u^*\}\rangle))$

Soundness By hypothesis, $l^* < L < u^* \leq U$. Thus, there exists a natural number n s.t. $\tilde{l_n} \leq L$ and for all i, $u_i = \tilde{u}_i \leq U$. Let's consider the following two cases:

1. If $ex \in P_R$ then $Ex = \emptyset$ (by definition of Exit-LEFT) and $\mathsf{Succ}_{s,ex}(I_i) = \emptyset$ for any i (by Corollary 9.3.1-2), so $\mathsf{Succ}_{s,ex}(\mathsf{Succ}_{s,f}(\langle L, \max\{u, u^*\}\rangle)) = \emptyset$;
2. If $ex \in P_L$, we consider two cases:
   (a) If $u < u^*$ then for all i, $u_i = \tilde{u}_i \leq u^*$ and then $\cup_{m>0}\mathsf{Succ}^m_{s,f}(I) = \mathsf{Succ}_{s,f}(L, u^*)$, thus $Ex = \mathsf{Succ}_{s,ex}(\mathsf{Succ}_{s,f}(L, u^*))$;
   (b) If $u^* < u$ then for all i, $u_i = \tilde{u}_i \leq u$ and $\cup_{m>0}\mathsf{Succ}^m_{s,f}(I) = \mathsf{Succ}_{s,f}(L, u)$. Consequently, $Ex = \mathsf{Succ}_{s,ex}(\mathsf{Succ}_{s,f}(L, u))$;

   From both cases we have that $Ex = \mathsf{Succ}_{s,ex}(\mathsf{Succ}_{s,f}(\langle L, \max\{u, u^*\}\rangle))$.

Termination Trivial. $\Box$

From the above results we have that the main algorithm for reachability is still valid for GSPDIs after doing the following pre-processing steps:

1. Detect all the inout edges;
2. Generate all the types of signatures fixing inout edges as entry and exit;
3. Apply the reachability algorithm for SPDIs given in chapter 5.

Notice that even though the set of type of signatures grows exponentially, it continues to be finite, hence we have that

Theorem 62 *The problem* $\mathbf{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ *is decidable for GSPDIs.* $\Box$

9.4 Conclusion

In this chapter we have defined *general SPDIs* (GSPDIs) that are SPDIs without the *goodness* restriction and we have proved that reachability is decidable for GSPDIs.

Chapter 10

Conclusion

10.1 Achievements

We have presented an algorithm for solving the reachability problem for polygonal differential inclusion systems (SPDIs). The novelty of the approach for the domain of Hybrid System is the combination of two techniques, namely, the representation of the two-dimensional continuous dynamics as a one-dimensional discrete system (due to Poincaré), and the characterization of "symbolic" trajectories of the latter as a finite set of types of signatures. The importance of this result lies not only on the result itself (the decidability of the reachability problem) but also on the method used. This is the first application of a "geometric" method to non-deterministic systems with the possibility of accelerating simples cycles in many cases.

As an application of the above algorithm, we have also given an automatic procedure to analyze the qualitative behavior of SPDIs. Our algorithm enumerates all the "limit cycles" (i.e., controllability kernels) and their local basins of attraction (i.e., viability kernels). Our analysis technique for a single cycle is very similar to the one used in [95] for n-dimensional systems. However, for polygonal systems, we are able to prove further properties such as controllability of and convergence to the set of fixpoints, and that there are only a finite number of them. These results are the analog of Poincaré-Bendixson for polygonal differential inclusions. The difference with [109] is that our results hold for non-deterministic systems.

We have not restricted our analysis only to SPDIs but to other two dimensional hybrid systems. Although many intense research activity in the last years have been done in the domain of hybrid systems, there is no clear boundary between what is decidable or not on such systems. In this line, we have two kind of results. First, we have shown that between 2-dimensional PCDs (for which the reachability problem is decidable [106]) and 3 dimensional PCDs (reachability is undecidable [19]) there exists an interesting class, *2-dimensional HPCDs*, for which the reachability question is still open. We have also shown that the same is true for other similar systems, namely 2-dimensional rectangular automata and 2-dimensional linear hybrid automata with some restrictions as well as for PCDs on 2-dimensional manifolds. This result was obtained showing that the decidability of the reachability problem for the above systems is equivalent to the decidability of the same question for piecewise affine maps (PAMs), that is a well known open problem. Second, we have proved that 2-dimensional HPCDs are really in the boundary between decidability and undecidability since the addition of a simple counter or the permission of having some kind of "infinite pattern" to these systems makes the reachability problem undecidable. The method used here was the reduction of the halting problem of Turing machines to the reachability problem for the above classes of systems.

From the practical point of view, we have implemented the reachability algorithm for SPDIs in the functional language Haskell [89] in a prototype of a tool called SPeeDI. Even though there is no specific tool for handling 2-dimensional non-deterministic systems using a decision algorithm, thus any comparison with other "semi-algorithmic" hybrid system verification tools (like d/dt [64] or HyTech [81]) could be meaningless, we have explored a simple example in order to compare SPeeDI with HyTech. From this comparison we can conclude that the geometric method (using Poincaré maps and acceleration techniques) used by SPeeDI has an important impact practically.

Summarizing, the main contributions of this thesis are the following:

- Reachability analysis: A decision procedure for solving exactly the reachability problem of *Polygonal Differential Inclusion Systems* (SPDIs);
  - First application of *geometric* methods for non-deterministic systems;
  - Application of *acceleration* of cycles;
- Algorithmic phase portrait construction of SPDIs;
  - Identification of important elements of the phase portrait, namely the viability and controllability kernels;
  - Exploration of properties of controllability and convergence to the set of limit cycles (Poincaré-Bendixson's like theorem);
  - A non-iterative algorithm for computing (exactly) the *viability* and *controllability kernels*;
- Undecidability analysis for 2-dimensional hybrid systems;
  - Identification of classes of 2-dimensional hybrid systems for which the decidability of the reachability question is equivalent to a well-known open problem;
  - Identification of features to cross the decidability boundary;
- Implementation of the reachability algorithm for SPDIs into a prototype of a tool called SPeeDI.

10.2 Research Directions

TAMF. The TAMF class could be formalized in an algebraic way using a simpler axiomatization of intervals than the one defined in [3].

Reachability. One question that naturally arises is decidability of the reachability problem for hybrid automata whose locations are equipped with SPDIs (hierarchical SPDIs). We have shown that the decidability of the reachability problem for HPCDs is an open question. We can certainly find (stringent) conditions, such as planarity of the automaton, "memory-less" resets, etc., under which decidability follows almost straightforwardly from the decidability of SPDIs.

Phase Portrait. This work is a first step in the direction of finding an algorithm for automatically constructing the complete phase portrait of an SPDI. This would require identifying and analyzing other useful structures such as stable and unstable manifolds, orbits (generated by identity Poincaré maps), bifurcation points (resulting of the non-deterministic qualitative behavior at the vertices of the polygons), limit behaviors of self-intersecting trajectories, etc.

Actually, Example 32 illustrates the difficulties that arise when exploring the limit behavior of self-crossing trajectories of an SPDI. Figure 6.9 shows that there exist infinite self-crossing (and even periodic) trajectories that keep switching between the two cycles forever. In this particular case, it can be shown that all trajectories converge to the "joint controllability kernel" $\mathsf{Cntr}(K_{\sigma_1} \cup K_{\sigma_2})$ which turns out to be $\mathcal{C}(\sigma_1) \cup \mathcal{C}(\sigma_2)$ (the cross-shaped region in the Figure is the bridge between the two cycles). However, the analysis of limit behaviors of self-cutting trajectories in the general case is considerably more difficult and challenging.

SPeeDI. From the practical point of view, we intend to improve SPeeDI including some "topological" optimizations. There are some intrinsic topological properties of SPDIs that can be exploited in order to optimize the generation of type of signatures. These optimizations are not implemented in the current version of SPeeDI.

1. We can explore the topological restriction that types of feasible signatures must respect the non-crossing property. For example, given $e_1, e_2, e_3 \in \mathit{In}(R)$, with $e_1 \prec e_2 \prec e_3$ and $e_4, e_5, e_6 \in \mathit{Out}(R)$, with $e_4 \prec e_5 \prec e_6$ and a signature $r_1 s_1 \cdots s_i r_i \cdots$ with $s_i = e_5 \cdots e_1$ and $r_i = e_6 \cdots$, then no continuation of the signature can contain e_4 and hence none of the successor of this edge. This is true for the other regions of the loop;

2. Another topological restriction comes from the fact that after getting into an EXIT-LEFT cycle, it is not possible to visit output edges that are at the right of the signature as well as all their continuation. The same kind of analysis can be done for EXIT-RIGHT and DIE loops;

3. A STAY cycle can appear just at the end of a signature.

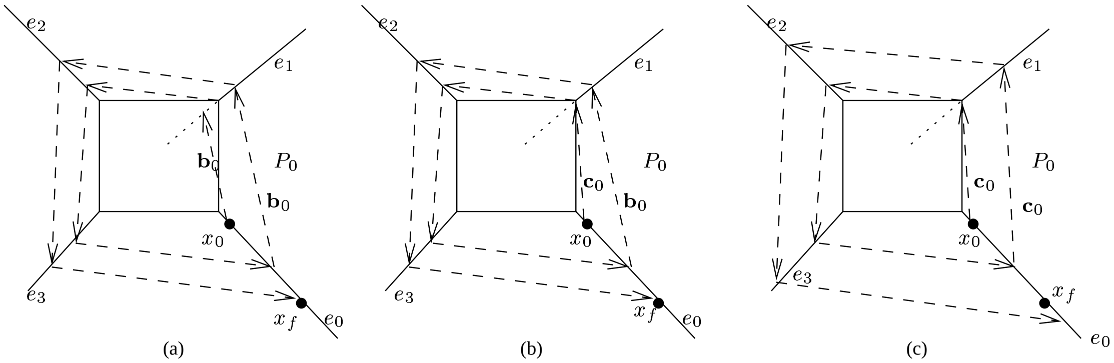


Figure 10.1: (a) x_f is reachable from x_0 following the leftmost (truncated) trajectory; (b) The dynamics in region P_0 is different the second time the trajectory enters it; (c) Taking $\mathbf{c}_0$ makes x_f unreachable.

Future work previews the integration of SPeeDI into a large tool suite for qualitative analysis of hybrid systems. We also plan to extend its functionality beyond reachability verification. In particular, we are currently working on the implementation of the algorithm developed in Chapter 6 for constructing the phase portrait of an SPDI which is composed of viability and controllability kernels.

Practical Applications. In this thesis we have not presented practical (real) examples and we intend to find out some interesting systems which can be modeled by SPDIs. The use of SPDIs for approximating non-linear planar differential equations seems to be a a very interesting application for further research. Given a non-linear differential equation we can find a (good) partition of the state space in order to be able to approximate it with an SPDI. The relevance of the approach is not only as a method for approximating reachability but also as a way of finding over-approximations of limit cycles and their basins of attraction.

Parametric Analysis. On other possible direction of future work is the application of the same method used for reachability for solving the the *parameter synthesis problem* for SPDIs, that is, for any two points, $\mathbf{x}_0$ and $\mathbf{x}_f$, assign a constant slope $\mathbf{c}_P \in \phi(P)$ to every region P such that $\mathbf{x}_f$ is reachable from $\mathbf{x}_0$, or conclude that such an assignment does not exist. Actually, the existence of a trajectory segment does not imply the existence of one with uniquely (in each region) defined slope. That is, the decidability of the reachability problem for SPDIs does not imply the decidability of the parameter synthesis one. As a simple counter-example just think of a cycle signature $\sigma = e_0e_1e_2e_3$ with $\mathbf{x}_0, \mathbf{x}_f \in e_0$ (i.e. $\mathbf{x}_0 = (e_0, x_0)$ and $\mathbf{x}_f = (e_0, x_f)$) with differential inclusions $\dot{\mathbf{x}} \in \angle_{\mathbf{a}_0}^{\mathbf{b}_0}$ such that the leftmost truncated successor is different from the non-truncated one for the first iteration ($\mathsf{Succ}^{\mathbf{b}}_{\sigma e_0}(x_0) \neq F^{\mathbf{b}}_{\sigma e_0}(x_0)$) and they are equal therein after. Suppose also that x_f is reachable from x_0 following always the leftmost (truncated) trajectory segment cycling three times, that is $x_f = \mathsf{Succ}^3_{\sigma e_0}(x_0)$ (see Fig. 10.1-(a)). Clearly, x_f is reachable from x_0, but there does not exist a unique value $\mathbf{c}_0 \in \angle_{\mathbf{a}_0}^{\mathbf{b}_0}$ that gives a positive answer to the reachability problem. In Fig. 10.1-(b) we can see that the dynamics in region P_0 is different the second time the trajectory segment enters it (suppose $\mathbf{c}_0 = \mathbf{a}_0 < \mathbf{b}_0$) whereas Fig. 10.1-(c) shows that the truncated successor restricts the following choice of possible values of the dynamics for P_0 to be at most equal to $\mathbf{c}_0 \in \angle_{\mathbf{a}_0}^{\mathbf{b}_0}$, therefore fixing the slope to be at least $\mathbf{c}_0$ in P_0, makes x_f unreachable (independently of the slopes chosen in the other regions).

Appendix A

Some properties of STAY cycles

We prove here two results about STAY cycles.

Lemma 1 *If σ is STAY then $S \cap J \subseteq F^{-1}(S \cap J)$.*

Proof: By hypothesis, $[l^*, u^*] \subseteq S \cap J$, that means that for any $I \subseteq S \cap J$, $F(I) \subseteq S \cap J$. By monotonicity of inverse function, $F^{-1} \circ F(I) \subseteq F^{-1}(S \cap J)$ from which we obtain that $I \subseteq F^{-1}(S \cap J)$ (since for any I, $I \subseteq F^{-1} \circ F(I)$). $\Box$

Lemma 2 *If σ is STAY then $S \cap J \subseteq \mathsf{Pre}_\sigma(S)$.*

Proof: By definition $\mathsf{Pre}_\sigma(S) = \mathcal{F}^{-1}(S) = F^{-1}(S \cap J) \cap S$. By lemma 1 we have that $S \cap J \subseteq F^{-1}(S \cap J)$ and then $S \cap J \subseteq F^{-1}(S \cap J) \cap S$. Hence, $S \cap J \subseteq \mathcal{F}^{-1}(S)$. $\Box$

Appendix B

Proofs of Lemmas of Chapter 7

Lemma 46 (PAM simulates HPCD) *For every 2-dim HPCD $\mathcal{H}$ there is a PAM $\mathcal{A}$ such that $\mathcal{A}$ simulates $\mathcal{H}$.*

Proof: Let $\mathcal{H}$ be a HPCD and PCD_i the PCD of location ℓ_i. We show how to encode each region of PCD_i by parts of a PAM $\mathcal{A}$. Let e_0 be an input edge of region R and $e_1, \cdots, e_k$ be output edges of R and reachable from e_0 by the one-step successor $\mathsf{Succ}_{e_0 e_i}(\lambda) = a_i\lambda + b_i$ $(1 \leq i \leq k)$ (see Figure 7.4). We partition edge e_0 into intervals $I_1, \cdots I_k$ in the following way: $I_i = \mathsf{Pre}_{e_0 e_i}(e_i)$. Suppose that each edge e_i $(0 \leq i \leq k)$ has local coordinates $0..d_i$. We dispose sequentially all the edges of R in the positive Real line starting for example at position p, i.e. $e_i = (l_i, u_i]$ with $l_0 = p$, $u_0 = p + d_0$ and for all $1 \leq i \leq k$, $l_i = u_{i-1}$ and $u_i = l_i + d_i$. Hence a point on edge e_i with local coordinates λ will be situated on the Real line $\mathbb{R}$ in position $l_i + \lambda$ (see Figure 7.4). We proceed in the same way for the other regions of PCD_i.

Let $\mathsf{Succ}_{e_i e_j}(\lambda) = a_i\lambda + b_i$ be a one-step successor, we define a function f as follows:

$$f(z) = A_i z + B_i \quad \text{if } z \in I_i$$

where $A_i = a_i$ and $B_i = b_i + l_j - a_i l_i$

We show now that for $\lambda_0 \in e_i$ and $\lambda_f \in e_j$, $\mathsf{Succ}_{e_i e_j}(\lambda_0) = \lambda_f$ iff $z_f = f(z_0)$. Let $\mathsf{Succ}_{e_i e_j}(\lambda_0) = \lambda_f = a_i\lambda_0 + b_i$ such that λ_0 and λ_f have coordinates $z_0 = l_i + \lambda_0$ and $z_f = l_j + \lambda_f$ on $\mathbb{R}$. Thus

$$\begin{aligned} \lambda_f = a_i\lambda_0 + b_i \quad & \text{iff } z_f - l_j = a_i(z_0 - l_i) + b_i \\ & \text{iff } z_f = a_i z_0 + (b_i + l_j - a_i l_i) \\ & \text{iff } z_f = A_i z_0 + B_i \\ & \text{iff } z_f = f(z_0) \end{aligned}$$

We have then constructed a function f for each one-step successor. The PAM $\mathcal{A}$ corresponding to the PCD of location ℓ_i is defined then as the function that consists of the body of all

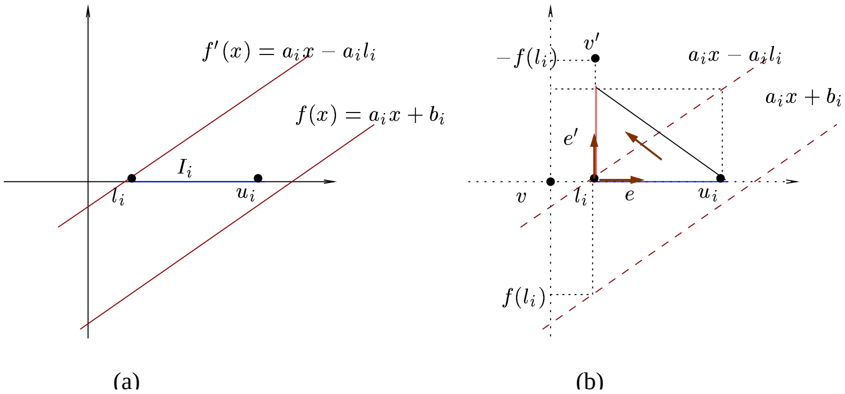


Figure B.1: (a) Translation of f to f'; (b) Basic elements for the computation of $\mathsf{Succ}_{ee'}$.

the functions f above. Up to now we have encoded just a simple PCD, it remains to encode the jumps from location ℓ_i to location ℓ_j in order to simulate a HPCD by a PAM. This is done in the same way as before, since the reset are edge-to-edge affine functions.

From the above results we have that Reach$(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ iff $z_f = f^*(z_0)$. $\square$

Lemma 47' (HPCD$_{\text{iso}}$ simulates PAM) *For every PAM $\mathcal{A}$ there is a 2-dim HPCD $\mathcal{H}$ with resets of the form $\gamma(x, y) = (y + d, 0)$ such that $\mathcal{H}$ simulates $\mathcal{A}$.*

Proof: Let $\mathcal{A}$ be defined by $f(z) = a_i z + b_i$ if $z \in I_i$ for $i \in \{1, \cdots, n\}$ where $I_i = [l_i, u_i]$ are rational intervals. For simplifying the presentation of the proof, we suppose that $a_i > 0$. We will discuss at the end how to do the simulation whenever $a_i < 0$.

Notice first that given a function $f(x) = a_i x + b_i$ (with $a_i > 0$) and an interval I_i on the x-axe, we can always define a function $f'(x)$ parallel to $f(x)$ such that $f'(l_i) = 0$: $f'(x) = a_i x - a_i l_i$ (see Figure B.1-(a)).

We show how to obtain a n-locations HPCD $\mathcal{H}$ that simulates $\mathcal{A}$. We associate with each interval I_i of $\mathcal{A}$, such that $a_i > 0$, a location ℓ_i defined with the following PCD (see Figure **??**):

Region	*Defining conditions*	*Vector*
R_1	$(x < u_i) \wedge (x > l_i) \wedge (y > 0) \wedge (y < -a_i x + a_i u_i)$	$(-1, a_i)$
R_2	$(x > l_i) \wedge (y > 0) \wedge (y > -a_i x + a_i u_i)$	$(1, 1)$
R_3	$(x < l_i)$	$(1, 1)$
R_4	$(x > l_i) \wedge (y < 0)$	$(1, 1)$

Let e_i and e'_i be the two edges of the above PCD defined as

$$\begin{aligned} e_i &\stackrel{\text{def}}{=} (x < u_i) \wedge (x > l_i) \wedge (y = 0) \\ e'_i &\stackrel{\text{def}}{=} (x = l_i) \wedge (y > 0) \wedge (y < a_i(u_i - l_i)) \end{aligned}$$

Notice that on region R_1, taking $\mathbf{e} = (1, 0)$, $\mathbf{e}' = (0, 1)$, $\mathbf{v} = (0, 0)$, $\mathbf{v}' = (l_i, -(a_i l_i + b_i))$ and $\mathbf{c} = (-1, a_i)$ (see Figure B.1-(b)) and applying the formula

$$\mathsf{Succ}_{e_i e'_i}(\lambda) = \frac{\mathbf{e}\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}\lambda + \frac{(\mathbf{v} - \mathbf{v}')\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}$$

we obtain

$$\mathsf{Succ}_{e_i e'_i}(\lambda) = a_i \lambda + b_i.$$

Notice that any point $\mathbf{x}' \in e'_i$ has local coordinates λ' in the interval $[a_i l_i + b_i, a_i u_i + b_i]$ and any point $\mathbf{x} = (x, y) \in e_i$ has local coordinates $\lambda \in [l_i, u_i]$ such that $\lambda = x$.

We encode a point $z \in I_i$ in $\mathcal{A}$ as a point $\mathbf{x} = (e_i, \lambda)$ on location ℓ_i, where $\lambda = z$.

Let e_f be an edge of PCD_f (on location ℓ_f) defined by

$$e_f \stackrel{\text{def}}{=} (x > l_j) \wedge (x < u_j) \wedge (y = 0)$$

Each location ℓ_i has one transition $tr = (\ell_i, g, \gamma, \ell_f)$ where the guard is defined as

$$g \stackrel{\text{def}}{=} (e'_i \wedge \lambda \in I_f)$$

and the reset function is defined as

$$\gamma(e'_i, \lambda) = (e_f, \lambda)^1.$$

Let $z_0 \in I_i$ and $z_f \in I_f$ be two points of the PAM $\mathcal{A}$ s.t. $f(z_0) = a_i z_0 + b_i = z_f$. These points are encoded on $\mathcal{H}$ as $\mathbf{x}_0 = (e_i, \lambda_0)$ and $\mathbf{x}_f = (e_f, \lambda_f)$ as explained before.

We prove now that given $z_0 \in I_i$ and $z_f \in I_f$ on $\mathcal{A}$,

$$z_f = f(z_0) \text{ iff } \lambda_f = \mathsf{Succ}_\sigma(\lambda_0)$$

where $\sigma = e_i e'_i e_f$.

First, suppose that $z_0 \in I_i$, $z_f \in I_f$ and $z_f = f(z_0) = a_i z_0 + b_i$, then on location ℓ_i we have $\mathbf{x}_0 = (e_i, \lambda_0)$ such that

$$\mathsf{Succ}_{e_i e'_i}(\lambda_0) = a_i \lambda_0 + b_i = \lambda'_0.$$

[1] By the above considerations about local coordinates, a reset of the form $\gamma(e'_i, \lambda) = (e_f, \lambda)$ corresponds in fact to the reset $\gamma(e'_i, x, y) = (e_f, y + d, 0)$ where $d = a_i l_i + b_i$. The result is obtained doing some algebraic manipulation transforming local coordinates into real coordinates (see Figure B.1)

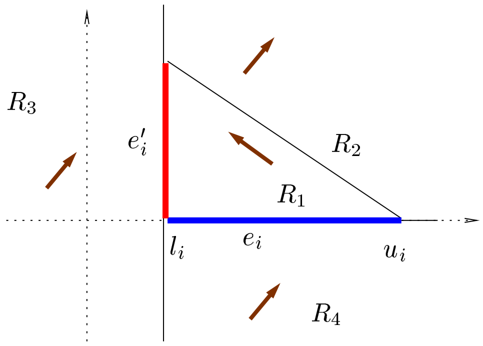


Figure B.2: PCD_i

But $\lambda'_0 = z_f \in I_f$ and hence the guard g is satisfied and the reset is applied:

$$\gamma(e'_i, \lambda'_0) = (e_f, \lambda'_0) = \mathbf{x}_f$$

Thus,

$$\lambda_f = \mathsf{Succ}_\sigma(\lambda_0).$$

Suppose now that $\lambda_f = \mathsf{Succ}_\sigma(\lambda_0)$, for $\sigma = e_i e'_i e_f$. We have that

$$\begin{aligned} \mathsf{Succ}_\sigma(\lambda_0) &= \mathsf{Succ}_{e'_i e_f} \circ \mathsf{Succ}_{e_i e'_i}(\lambda_0) \\ &= \mathsf{Succ}_{e'_i e_f}(a_i\lambda_0 + b_i) \\ &= a_i\lambda_0 + b_i \\ &= \lambda_f \end{aligned}$$

Hence $z_f = a_i z_0 + b_i = f(z_0)$.

Whenever $a_i < 0$, we define region R_1 by $(x < u_i) \wedge (x > l_i) \wedge (y > 0) \wedge (y < -a_i x + a_i l_i)$ (the other regions are then determined). Taking $\mathbf{e} = (1,0)$, $\mathbf{e}' = (0,1)$, $\mathbf{v} = (0,0)$, $\mathbf{v}' = (u_i, -(a_i u_i + b_i))$ and $\mathbf{c} = (1, -a_i)$ we have that $\mathsf{Succ}_{ee'}(\lambda) = a_i\lambda + b_i$. The rest of the proof is as before.

As a consequence we have then that $z_f = f^*(z_0)$ iff Reach$(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$. $\square$

Lemma 55 (PCD$_{2\text{m}}$ **simulates** PAM$_{\text{inj}}$) *Every injective PAM system can be simulated by a* PCD$_{2\text{m}}$.

Proof: Let $\mathcal{A}$ be an injective PAM defined as $f(z) = a_i z + b_i$ if $z \in I_i$ for $1 \leq i \leq n$. For simplifying the presentation of the proof, we suppose that $a_i > 0$. Whenever $a_i < 0$, a transformation as the one shown in the proof of Lemma 47' is done.

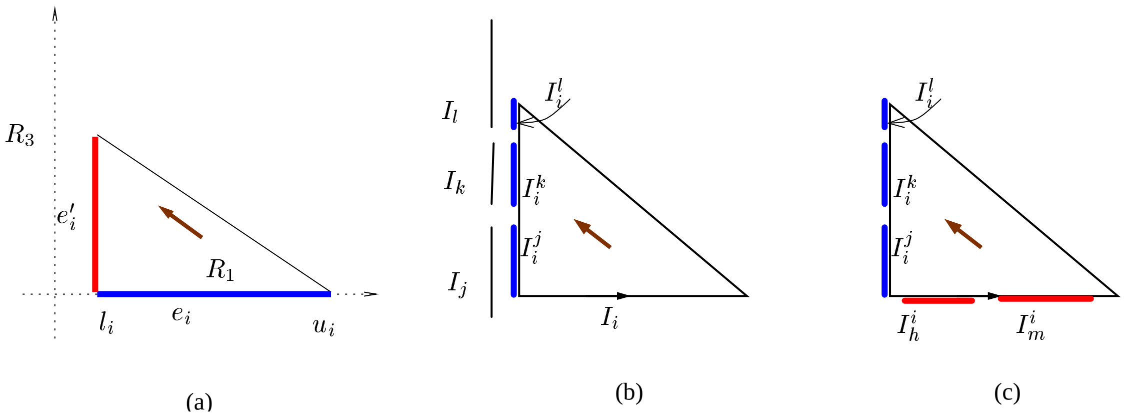


Figure B.3: Simulation of a $\mathrm{PCD_{2m}}$ by a $\mathrm{PAM_{inj}}$: (a) First definition of PCD_i; (b) Decomposition of edge e'_i; (c) Partition of edge I_i.

We obtain a $\mathrm{PCD_{2m}}$ $\mathcal{M}$ in the following way.

First, for each interval I_i, such that $a_i > 0$ we define first the following one-region PCD (see Figure B.3-(a)):

Region	*Defining conditions*	*Vector*
R_i	$(x < u_i) \wedge (x > l_i) \wedge (y > 0) \wedge (y < -a_i x + a_i u_i)$	$(-1, a_i)$

Let e_i and e'_i be the two edges of the above PCD defined as

$$\begin{aligned} e_i &\stackrel{\text{def}}{=} (x < u_i) \wedge (x > l_i) \wedge (y = 0) \\ e'_i &\stackrel{\text{def}}{=} (x = l_i) \wedge (y > 0) \wedge (y < a_i(u_i - l_i)) \end{aligned}$$

Notice that on region R_1, taking $\mathbf{e} = (1, 0)$, $\mathbf{e}' = (0, 1)$, $\mathbf{v} = (0, 0)$, $\mathbf{v}' = (l_i, -(a_i l_i + b_i))$ and $\mathbf{c} = (-1, a_i)$ (see Figure B.1-(b)) and applying the formula

$$\mathsf{Succ}_{e_i e'_i}(\lambda) = \frac{\mathbf{e}\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}\lambda + \frac{(\mathbf{v} - \mathbf{v}')\hat{\mathbf{c}}}{\mathbf{e}'\hat{\mathbf{c}}}$$

we obtain

$$\mathsf{Succ}_{e_i e'_i}(\lambda) = a_i \lambda + b_i.$$

Notice that, as for the $\mathrm{HPCD_{iso}}$ (see proof of Lemma 47') any point $\mathbf{x}' \in e'_i$ has local coordinates λ' in the interval $[a_i l_i + b_i, a_i u_i + b_i]$ and any point $\mathbf{x} = (x, y) \in e_i$ has local coordinates $\lambda \in [l_i, u_i]$ such that $\lambda = x$.

We encode a point $z \in I_i$ in $\mathcal{A}$ as a point $\mathbf{x} = (e_i, \lambda)$ on location ℓ_i, where $\lambda = z$.

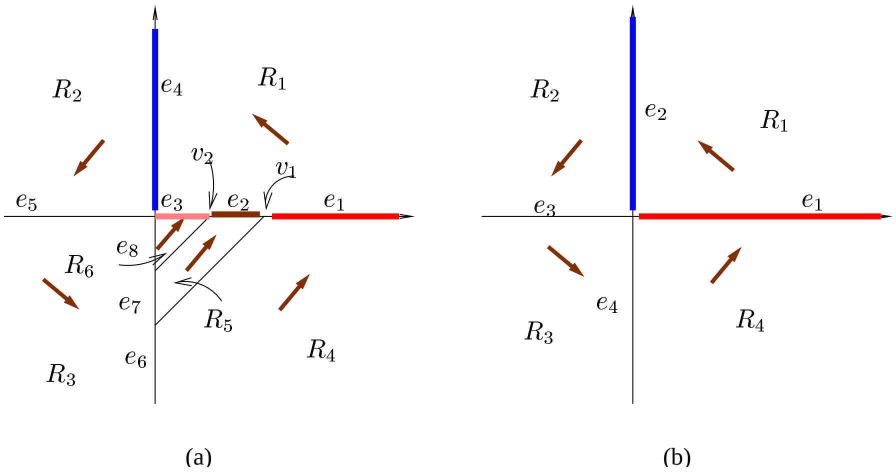


Figure B.4: Sketch of the simulation of a TM by a $\mathrm{HPCD_{1c}}$: (a) PCD_i; (b) PCD'_i and PCD''_i.

Second, we partition edge e'_i (on location ℓ_i) into intervals I_i^k if $\mathsf{Succ}_{e_ie'_i}(I_i) \cap I_k \neq \emptyset$, as shown in Figure B.3-(b). In fact, $I_i^k = \mathsf{Succ}_{e_ie'_i}(I_i) \cap I_k$. Each segment I_i^k of this edge is identified with the corresponding interval on the oriented edges I_k. Finally, edge I_i is partitioned into intervals I_h^i if $\mathsf{Succ}_{e_ie'_i}(I_h) \cap I_i \neq \emptyset$ (on location ℓ_h), again $I_h^i = \mathsf{Succ}_{e_ie'_i}(I_h) \cap I_i$ (see Figure B.3-(c)). We obtain in this way (doing the same construction for every interval I_i, $1 \leq i \leq n$) a surface with boundary. It is important to emphasize that each $I_i^j \in e'_i$ (for all $1 \leq i \neq j \leq n$) is identified with exactly one segment of edge I_j. By the Classification Theorem for Surfaces with Boundary (see Theorem 45) we have that this surface is equivalent to a sphere with some disks removed and we obtain then a $\mathrm{PCD_{2m}}$ just "sewing" the disks. We associate with these disks a zero dynamics, i.e. the dynamics on these regions are given by the vector $(0, 0)$. The above construction allows to prove that $z_f = f(z_0)$ iff $\lambda_f = \mathsf{Succ}_\sigma(\lambda_0)$ on the same lines as for $\mathrm{HPCD_{iso}}$. □

Proposition 4 ($\mathrm{HPCD_{1c}}$ simulates TM) *For every TM $\mathcal{M}$ there is a 2-dim HPCD with one counter $\mathcal{H}$ such that $\mathcal{H}$ simulates $\mathcal{M}$.*

Proof: Let (w_L, q, w_R) be a TM-configuration. We encode each TM-state q_i by three ocations ℓ_i, ℓ'_i and ℓ''_i of $\mathcal{H}$ defined as (see Figure B.4 and B.5):

1. Location ℓ_i. PCD_i is defined as:

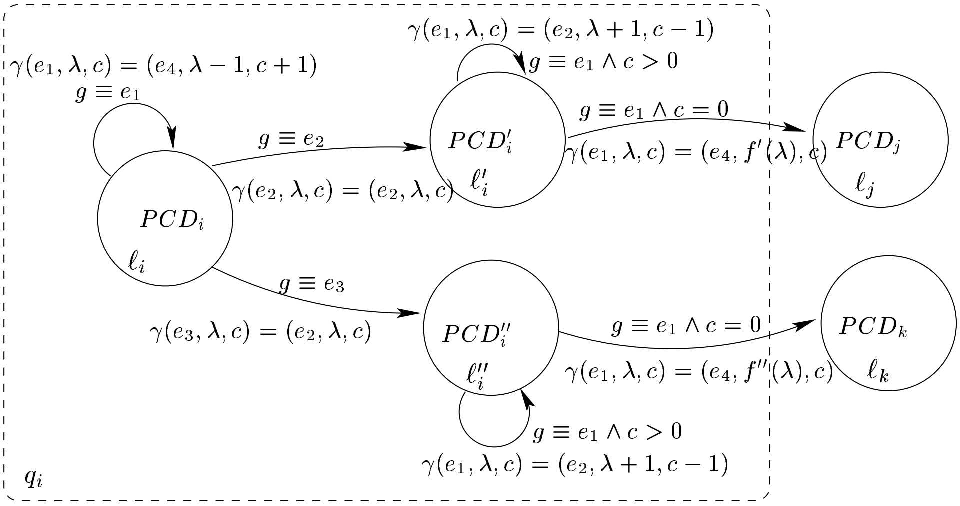


Figure B.5: Sketch of the simulation of a TM by a HPCD_{1c}: Representation of TM-state q_i.

Region	*Defining conditions*	*Vector*
R_1	$(x>0)\wedge(y>0)$	(-1,1)
R_2	$(x<0)\wedge(y>0)$	(-1,-1)
R_3	$(x<0)\wedge(y<0)$	(1,-1)
R_4	$(x>0)\wedge(y<0)\wedge(y<x-1)$	(1,1)
R_5	$(x>0)\wedge(y<0)\wedge(y>x-1)\wedge(y<x-\frac{1}{2})$	(1,1)
R_6	$(x>0)\wedge(y<0)\wedge(y>x-\frac{1}{2})$	(1,1)

Let $e_1,\dots e_8$ be the following edges:

$$
\begin{aligned}
e_1 &\stackrel{\text{def}}{=} (y=0)\wedge(x>1)\\
e_2 &\stackrel{\text{def}}{=} (y=0)\wedge(x<1)\wedge(x>\tfrac{1}{2})\\
e_3 &\stackrel{\text{def}}{=} (y=0)\wedge(x>0)\wedge(x<\tfrac{1}{2})\\
e_4 &\stackrel{\text{def}}{=} (x=0)\wedge(y>0)\\
e_5 &\stackrel{\text{def}}{=} (y=0)\wedge(x<0)\\
e_6 &\stackrel{\text{def}}{=} (x=0)\wedge(y<1)\\
e_7 &\stackrel{\text{def}}{=} (x=0)\wedge(y>-\tfrac{1}{2})\wedge(y<-1)\\
e_8 &\stackrel{\text{def}}{=} (x=0)\wedge(y<0)\wedge(y>-\tfrac{1}{2})
\end{aligned}
$$

We consider also the two vertex (that are edges) $v_1 \stackrel{\text{def}}{=} (y=0)\wedge(x=1)$ and $v_2 \stackrel{\text{def}}{=} (y=0)\wedge(x=\frac{1}{2})$.

There are three transitions from ℓ_i[2]:

[2] As was already mentioned, a point $\mathbf{x}=(x,y)\in e$ can be represented by edge e and its local coordinate:

(a) $tr_1 = (\ell_i, g_1, \gamma_1, \ell_h)$ where $g_1 \equiv e_1$, $\ell_h = \ell_i$ and $\gamma_1(e_1, \lambda, c) = (e_4, \lambda - 1, c + 1)$;

(b) $tr_2 = (\ell_i, g_2, \gamma_2, \ell_g)$ where $g_2 \equiv (e_2 \vee v_2)$, $\ell_g = \ell_i'$, $\gamma_2(e_2, \lambda, c) = (e_2, \lambda, c)$ and $\gamma_2(v_2, \lambda, c) = (e_2, \lambda, c)$

(c) $tr_3 = (\ell_i, g_3, \gamma_3, \ell_k)$ where $g_3 \equiv (e_3 \vee v_1)$, $\ell_k = \ell_i''$, $\gamma_3(e_3, \lambda, c) = (e_2, \lambda, c)$ and $\gamma_3(v_1, \lambda, c) = (e_2, \lambda, c)$.[3]

In what follows we do not take into account vertex v_1 and v_2 for sake of simplicity. Their treatment is like for edges e_2 and e_3.

Let $\sigma = e_4 e_5 e_6 e_1$. It's not difficult to see that $\mathsf{Succ}_\sigma = Id$ and that $\mathsf{Succ}_{\sigma e_4}(\lambda) = \lambda - 1$ (since when edge e_1 is reached, there is a transition to edge e_4 with a reset function that takes λ into $\lambda - 1$). Notice also that $\mathsf{Succ}_{e_4 e_5 e_7 e_2} = \mathsf{Succ}_{e_4 e_5 e_8 e_3} = Id$.

2. Location ℓ_i'. PCD_i' is defined as:

Region	*Defining conditions*	*Vector*
R_1	$(x > 0) \wedge (y > 0)$	(-1,1)
R_2	$(x < 0) \wedge (y > 0)$	(-1,-1)
R_3	$(x < 0) \wedge (y < 0)$	(1,-1)
R_4	$(x > 0) \wedge (y < 0)$	(1,1)

Let e_1, e_2, e_3 and e_4 be defined as:

$$\begin{aligned} e_1 &\stackrel{\text{def}}{=} (y = 0) \wedge (x > 0) \\ e_2 &\stackrel{\text{def}}{=} (x = 0) \wedge (y > 0) \\ e_3 &\stackrel{\text{def}}{=} (y = 0) \wedge (x < 0) \\ e_4 &\stackrel{\text{def}}{=} (x = 0) \wedge (y < 0) \end{aligned}$$

There are two transitions from ℓ_i':

(a) $tr_1 = (\ell_i', g_1, \gamma_1, \ell_h)$ where $g_1 \equiv (e_1 \wedge c > 0)$, $\ell_h = \ell_i'$ and $\gamma_1(e_1, \lambda, c) = (e_2, \lambda + 1, c - 1)$;

(b) $tr_2 = (\ell_i', g_2, \gamma_2, \ell_j)$ where $g_2 \equiv (e_1 \wedge c = 0)$ and $\gamma_2(e_1, \lambda, c) = (e_4, f'(\lambda), c)$ (ℓ_j and $f'(\lambda)$ will be defined later);

Let $\sigma = e_2 e_3 e_4 e_1$. It's not difficult to see that $\mathsf{Succ}_\sigma = Id$ and that $\mathsf{Succ}_{\sigma e_2}(\lambda) = \lambda + 1$ (as before, there is a reset function from edge e_1, while $c > 0$, to edge e_2, but that increments λ instead of decrementing it)

3. Location ℓ_i'' is defined as ℓ_i' with the only difference that transition tr_2 it's from ℓ_i'' to ℓ_k that will be in general different from location ℓ_j and that $f''(\lambda)$ will be also different of $f'(\lambda)$:

$\mathbf{x} = (e, \lambda)$. In this proof we will use λ instead of (x, y).

[3] By the above considerations about local coordinates, a reset of the form $\gamma(e_1, \lambda) = (e_4, \lambda - 1)$ for instance, corresponds in fact to the reset $\gamma(e_1, x, y) = (e_4, 0, x - 1)$.

(a) $tr_1 = (\ell_i'', g_1, \gamma_1, \ell_h)$ where $g_1 \equiv (e_1 \wedge c > 0)$, $\ell_h = \ell_i''$ and $\gamma_1(e_1, \lambda, c) = (e_2, \lambda + 1, c - 1)$;

(b) $tr_2 = (\ell_i'', g_2, \gamma_2, \ell_k)$ where $g_2 \equiv (e_1 \wedge c = 0)$ and $\gamma_2(e_1, \lambda, c) = (e_4, f''(\lambda), \lambda, c)$ (ℓ_k and $f''(\lambda)$ will be defined later);

Remember that string $w_L w_R$ can be represented as a positive rational number z as follows. Let $w_L = \ldots a_i, \ldots, a_1, a_0$ and $w_R = a_{-1}, a_{-2}, \ldots, a_{-j} \ldots$ w.l.o.g. we suppose that $\mathcal{M}$ has at least one '1'). Then we define[4]

$$z = \sum_{i=-\infty}^{\infty} a_i 2^i$$

We encode z as a point, in $\mathcal{H}$, $\mathbf{x} = (e_4, \lambda)$ on location ℓ.

Thus, a TM-configuration (w_{L_i}, q_i, w_{R_i}) is encoded as follows: q_i is encoded as the 3 locations ℓ_i, ℓ_i' and ℓ_i'' as described above and the string $w_{L_i} w_{R_i}$ is encoded as a point $\mathbf{x}_i = (e_4, \lambda_i)$ on location ℓ_i.

Before explaining how to encode each TM-instruction we show that for a given $\mathbf{x} = (e_4, \lambda)$ (on location l_i) we reach a point $\mathbf{x}_f = (e_1, \lambda_f)$ on location l_j (on location l_k) if $\text{frac}_\lambda < \frac{1}{2}$ ($\text{frac}_\lambda \geq \frac{1}{2}$). Initially, $c := 0$.

Let $\sigma = e_4 e_5 e_6 e_1$, $\beta_i = \mathsf{Succ}_\sigma(\lambda)$ the i-th time a trajectory starting at $\mathbf{x} = (e_4, \lambda)$ visit edge e_1 and $\alpha_i = \mathsf{Succ}_{\sigma e_4}(\lambda) = \beta_i - 1$ idem for edge e_4. Then $\beta_1 = Id(\lambda) = \lambda$ and after taking transition tr_1 we have $\alpha_1 = \mathsf{Succ}_{\sigma e_4}(\lambda) = \lambda - 1$ and $c = 1$. We have then that $\beta_2 = \lambda - 1$ and $\alpha_2 = \mathsf{Succ}_{\sigma e_4}(\lambda - 1) = \lambda - 2$ and $c = 2$. In general (by an easy induction proof), we have that $\beta_i = \lambda - (i-1)$, $\alpha_i = \lambda - i$ and $c = i$, but whenever $\beta_i \leq 1$ then $\mathsf{Succ}^i_\sigma(\lambda) = \emptyset$. We prove that $\mathsf{Succ}^i_\sigma(\lambda) = \emptyset$ after int_λ iterations: let $i = \text{int}_\lambda$, then $\alpha_i = \lambda - \text{int}_\lambda = \text{frac}_\lambda$ and $\alpha_i < 1$, hence $\mathsf{Succ}^i_\sigma(\lambda) = \emptyset$. Notice that at this moment we reach edge e_2 or e_3 depending on whether $\text{frac}_\lambda \geq \frac{1}{2}$ or $\text{frac}_\lambda < \frac{1}{2}$ and $c = \text{int}_\lambda$. If edge e_2 is reached, then guard g_2 is satisfied and the system jumps to edge e_2 on location ℓ_i' with $\lambda' = \text{frac}_\lambda$ and $c = \text{int}_\lambda$. At ℓ_i', λ' is incremented at the same time c is decremented and whenever $c = 0$ (and hence $\lambda' = \lambda$) a jump to edge e_4 on location ℓ_j is produced where $c = 0$ and $\lambda_f = f'(\lambda')$. The proof is similar for edge e_3 on location ℓ_i, arriving at location ℓ_k with $c = 0$ and $\lambda_f = f''(\lambda')$.

We show now how to encode each TM-instruction by some computation on $\mathcal{H}$. Let $s \in \{0, 1\}$. Notice that in order to simulate each TM-instruction we need to obtain the current symbol and depending on its value to take an action. In fact we have already shown that testing whether the current symbol is 0 or 1 is done just testing frac_λ and then the only things that remain to define are functions f' and f''.

$q_i\, s \longrightarrow s\, q_j\, R$: Take $f'(\lambda) = f''(\lambda) = 2\lambda$.

$q_i\, 0 \longrightarrow 1\, q_j\, R$: Define $f''(\lambda) = 2\lambda + 1$.

[4] w_L and w_R can be obtained by taking int_z (the integer part of z) and frac_z (its fractional part). If $\text{frac}_z < \frac{1}{2}$ then the current symbol is 0, otherwise it is 1.

$q_i\,1 \longrightarrow 0\,q_j\,R$: In this case we take $f'(\lambda) = 2\lambda - 1$.

$q_i\,s \longrightarrow s\,q_j\,L$: Take $f'(\lambda) = f''(\lambda) = \frac{1}{2}\lambda$.

$q_i\,0 \longrightarrow 1\,q_j\,L$: Define $f''(\lambda) = \frac{1}{2}\lambda - \frac{1}{2}$.

$q_i\,1 \longrightarrow 0\,q_j\,L$: Take $f'(\lambda) = \frac{1}{2}\lambda + \frac{1}{2}$.

Thus, λ_f encodes z_f.

From the above results we have that $\mathrm{Reach}(\mathcal{H}, \mathbf{x}_0, \mathbf{x}_f)$ iff a TM stops in a configuration (w_{L_f}, q_f, w_{R_f}) starting at a given configuration (w_{L_0}, q_0, w_{R_0}). $\square$

Appendix C

Main Utilities of SPeeDI

The following is more detailed description of the tools. All the utilities described next takes a ⟨file.spdi⟩ as input. We describe briefly the different utilities of SPeeDI (see Figure 8.5):

getmafs: It takes an edge list and it returns a list of the MAFs corresponding to pairs of successive edge in the list, as well as the accumulated MAF (accumulated composition of the edge-to-edge successors);

looptype: Given a list of edges (a cycle signature) and an exit edge, it returns the (qualitative) type of the cycle (DIE, STAY, EXIT-BOTH, EXIT-RIGHT or EXIT-LEFT);

reachable: Given an input interval (I_0), a source edge (e_0), an output interval (I_f) and a final edge (e_f), it answer to the question: Is $I_f \subseteq e_f$ reachable from $I_0 \subseteq e_0$? If it is the case, then the answer is **Yes** and it gives a signature that realizes it, otherwise it says **No**;

showsigs: It takes a source and a destination edge (e_0 and e_f respectively) and it gives a list of all the feasible type of signatures from e_0 to e_f;

simSIG: Given an input interval (I_0), an output interval (I_f) and a signature it returns a concrete signature: list of pairs of edges and intervals, that is the concrete execution of the signature for the input I_0. It leaves a cycle as late as possible;

simsig: Idem simsignature but it leaves a cycle as soon as possible;

trysig: It gives the reachable interval for a given input interval and signature;

simsig2fig: It generates a picture in FIG format (file.fig) from the simulation execution of simsig.

simSIG2fig: It generates a picture in FIG format (file.fig) from the simulation execution of simSIG.

spdi2ps: It generate a postscript file that is the graphic representation of the input SPDI file;

sig2ps: Generate a PostScript file with the type of a signature — a plain arrow showing the path followed.

sig2fig: As sig2ps, but generates a FIG file.

path2fig: Given a path (a sequence of edges to be followed sequentially) produce a FIG file showing the explored regions.

Appendix D

File Syntax for SPeeDI

As shown in Figure 8.3, the input file consists of three parts. The first part is the description of the regions vertices that starts with the keyword "`Points:`". The syntax is the following:

```
<point-label>. <x-coordinate>, <y-coordinate>
```

White spaces are allowed anywhere and each point description is written ina separate line. The same is valid in general for the other parts that follow.

The second part starts with the keyword "`Vectors:`" and corresponds to the description of the differential inclusions vectors:

```
<vector-label>. <x-coordinate>, <y-coordinate>
```

The last part begin after the keyword "`Regions:`" where the regions are described as a sequence of vertices separated by "?" or "!" depending on the character of the edge defined by the two successive vertex. If the edge is of Input, then a "?" is used as separator of the corresponding vertices. If it is an Output edge, then "!" is written. The differential inclusion of the region is input as a pair of vectors (defined in the `Vector` part). Syntax:

```
<list-of-edges>, <a-vector>, <b-vector>
```

Where <a-vector> and <b-vector> are <vector-label>s and they correspond to the differential inclusion given by $\angle_{\mathbf{a}}^{\mathbf{b}}$; <list-of-edges> is given by the following syntax:

```
<point-label> <inout> <point-label> ... <inout> <point-label>
```

where <inout> can be "?" or "!" and the first <point-label> must be equal to the last one.

Input File

input–file	::=	**Points:** `<EOL>`
		point–list
		Vectors: `<EOL>`
		vector–list
		Regions: `<EOL>`
		region–list
comment	::=	‘*’ *line*
line	::=	*string*
int–constant	::=	$[0-9]^+$
point	::=	*point–label* ‘.’ *x–coordinate* ‘,’ *y–coordinate*
point–list	::=	*point* `<EOL>` *point–list* | ε
vector	::=	*vector–label* ‘.’ *x–coordinate* ‘,’ *y–coordinate*
vector–list	::=	*vector* `<EOL>` *vector–list* | ε
region	::=	*region–def* ‘,’ *a–vector* ‘,’ *b–vector*
region–def	::=	*point–label inout region–def*
		| *point–label*
region–list	::=	*region* `<EOL>` *region–list* | ε
inout	::=	‘?’ | ‘!’
a–vector	::=	*vector–label*
b–vector	::=	*vector–label*
x–coordinate	::=	*coordinate*
y–coordinate	::=	*coordinate*
coordinate	::=	*rational* ‘,’ *rational*
rational	::=	*int–constant* ‘.’ *int–constant*
point–label	::=	*string*
vector–label	::=	*string*

Table D.1: Syntax of the Input file for SPeeDI.

Comments are written as lines starting with an asterisk ‘*’ and can be put at any moment on the input file.

In Table D.1 we present the syntax of SPeeDI input file and in Table D.2 the syntax of the command-lines. We denote a blanc space by ‘␣’. We define *string* to be the sequenece of any character with the exception of the ones in the following set: {‘␣’, ‘,’, ‘.’, ‘?’, ‘!’, ‘*’}. `<EOL>` is the *end-of-line* character.

In table D.2 “file.*” will denote the name of the input/output file, where $* \in \{\mathrm{fig}, \mathrm{ps}, \mathrm{spdi}\}$.

Commands

command–line	::=	*getmafs*
		$\mid$ *looptype*
		$\mid$ *path2fig*
		$\mid$ *reachable*
		$\mid$ *showsigs*
		$\mid$ *sig2fig*
		$\mid$ *sig2ps*
		$\mid$ *simsig*
		$\mid$ *simSIG*
		$\mid$ *simsig2fig*
		$\mid$ *simSIG2fig*
		$\mid$ *spdi2ps*
		$\mid$ *trysig*
getmafs	::=	**getmafs** ‘file.spdi’ *edge–list*
looptype	::=	**looptype** ‘file.spdi’ *edge–list edge*
path2fig	::=	**path2fig** [-o ‘file.fig’] *file–spdi interval edgelist*
reachable	::=	**reachable** ‘file.spdi’ *initial–interval final–interval* *initial-edge final–edge*
showsigs	::=	**showsigs** ‘file.spdi’ *initial-edge final–edge*
sig2fig	::=	**sig2fig** [-o file.fig] ‘file.spdi’ ‘“’ *signature* ‘”’
sig2ps	::=	**sig2ps** [-o file.ps] ‘file.spdi’ ‘“’ *signature* ‘”’
simsig	::=	**simsig** ‘file.spdi’ *initial–interval final–interval* ‘“’ *signature* ‘”’
simSIG	::=	**simSIG** ‘file.spdi’ *initial–interval final–interval* ‘“’ *signature*‘”’
simsig2fig	::=	**simsig2fig** [-o file.fig] ‘file.spdi’ *initial–interval* *final–interval* ‘“’ *signature* ‘”’
simSIG2fig	::=	**simSIG2fig** [-o file.fig] ‘file.spdi’ *initial–interval* *final–interval* ‘“’ *signature* ‘”’
spdi2ps	::=	**spdi2ps** ‘file.spdi’ [‘file.ps’]
trysig	::=	**trysig** ‘file.spdi’ *initial–interval final–interval* ‘“’ *signature* ‘”’
edge	::=	*point* ‘-’ *point*
initial–edge	::=	*edge*
final–edge	::=	*edge*
edge–list	::=	*edge* ‘,’ *edge–list* $\mid$ ε
initial–interval	::=	*interval*
final–interval	::=	*interval*
interval	::=	*left–extremity rational* ‘,’ *rational right–extremity*
left–extremity	::=	‘[’ $\mid$ ‘(’
right–extremity	::=	‘]’ $\mid$ ‘)’
signature	::=	*sequential loop signature* $\mid$ ε
loop	::=	‘[’ *edge–list* ‘]’ $\mid$ ε
sequential	::=	*edge–list* $\mid$ ε

Table D.2: Syntax of the command-lines.

Appendix E

Input file for Example 8.4

The input file, example1.spdi is shown in what follows:

```
Points:
0. 0.0, 0.0
1. -45.0, -35.0
2. -45.0, -25.0
3. -45.0, -15.0
4. -45.0, -5.0
5. -45.0, 5.0
6. -45.0, 15.0
7. -45.0, 25.0
8. -45.0, 35.0
9. -35.0, -35.0
10. -35.0, -25.0
11. -35.0, -15.0
12. -35.0, -5.0
13. -35.0, 5.0
14. -35.0, 15.0
15. -35.0, 25.0
16. -35.0, 35.0
17. -25.0, -35.0
18. -25.0, -25.0
19. -25.0, -15.0
20. -25.0, -5.0
21. -25.0, 5.0
22. -25.0, 15.0
23. -25.0, 25.0
24. -25.0, 35.0
25. -15.0, -35.0
26. -15.0, -25.0
27. -15.0, -15.0
28. -15.0, -5.0
29. -15.0, 5.0
30. -15.0, 15.0
31. -15.0, 25.0
32. -15.0, 35.0
33. -5.0, -35.0
34. -5.0, -25.0
35. -5.0, -15.0
36. -5.0, -5.0
37. -5.0, 5.0
38. -5.0, 15.0
39. -5.0, 25.0
40. -5.0, 35.0
41. 5.0, -35.0
42. 5.0, -25.0
43. 5.0, -15.0
44. 5.0, -5.0
45. 5.0, 5.0
46. 5.0, 15.0
47. 5.0, 25.0
48. 5.0, 35.0
49. 15.0, -35.0
50. 15.0, -25.0
51. 15.0, -15.0
52. 15.0, -5.0
53. 15.0, 5.0
54. 15.0, 15.0
55. 15.0, 25.0
56. 15.0, 35.0
57. 25.0, -35.0
58. 25.0, -25.0
59. 25.0, -15.0
60. 25.0, -5.0
61. 25.0, 5.0
62. 25.0, 15.0
63. 25.0, 25.0
64. 25.0, 35.0
65. 35.0, -35.0
66. 35.0, -25.0
67. 35.0, -15.0
68. 35.0, -5.0
69. 35.0, 5.0
70. 35.0, 15.0
71. 35.0, 25.0
72. 35.0, 35.0
73. 45.0, -35.0
74. 45.0, -25.0
75. 45.0, -15.0
76. 45.0, -5.0
77. 45.0, 5.0
78. 45.0, 15.0
79. 45.0, 25.0
80. 45.0, 35.0

Vectors:
v1. 1,5
v2. -1,0.5
```

```
v3. -1,0.1833333333
v4. -1,-0.25
v5. -1,-1
v6. 0,-1
v7. 1,-1
v8. 1,0
v9. 1,1
v10. 1,2
v11. -1,1
v12. 1, 1.5
v13. 5,2
v14. 1,-5
v15. -1,-5
v16. 1,-2
v17. 1,6
v18. 0,1
v19. 0.001,1
v20. -1, 0.001
v21. 0.001,-1
v22. 1,-0.001
v23. 0.01,1
v24. 0.1,1
v25. -1,0.7
v26. -1,0.1
v27. 1,0.1833333333
v28. 1, 0.001
v29. -1,1.5

Regions:
* R1
1 ? 9 ! 10 ! 2 ? 1, v9, v9
* R2
2 ? 10 ! 11 ! 3 ? 2, v9, v9
* R3
3 ? 11 ! 12 ! 4 ? 3, v9, v9
* R4
4 ? 12 ! 13 ! 5 ? 4, v9, v9
* R5
5 ? 13 ! 14 ! 6 ? 5, v9, v9
* R6
6 ? 14 ! 15 ! 7 ? 6, v9, v9
* R7
7 ? 15 ! 16 ! 8 ? 7, v9, v9
* R8
9 ? 17 ! 18 ! 10 ? 9, v9, v9
* R9
10 ? 18 ! 19 ! 11 ? 10, v9, v9
* R10
11 ? 19 ! 20 ! 12 ? 11, v9, v9
* R11
12 ? 20 ! 21 ! 13 ? 12, v9, v9
* R12
13 ? 21 ! 22 ! 14 ? 13, v9, v9
* R13
14 ? 22 ! 23 ! 15 ? 14, v9, v9
* R14
15 ? 23 ! 24 ! 16 ? 15, v9, v9

* R15
17 ? 25 ! 26 ! 18 ? 17, v9, v9
* R16
18 ! 26 ! 27 ? 19 ? 18, v7, v7
* R17
19 ! 27 ? 28 ? 20 ! 19, v21, v21
* R18
20 ! 28 ? 29 ? 21 ! 20, v21, v21
* R19
21 ! 29 ? 30 ? 22 ! 21, v21, v21
* R20
22 ! 30 ? 31 ? 23 ! 22, v5, v5
* R21
23 ? 31 ? 32 ! 24 ! 23, v2, v2
* R22
25 ? 33 ! 34 ! 26 ? 25, v9, v9
* R23
26 ! 34 ! 35 ? 27 ? 26, v22, v22
* R24
27 ? 28 ? 36 ! 35 ! 27, v7, v7
* R25
29 ? 37 ? 0 ! 36 ! 28 ! 29, v6, v6
* R26
37 ? 38 ? 30 ! 29 ! 37, v5, v5

* R27
30 ? 38 ? 39 ! 31 ! 30, v26, v20
* R28
```

```
31 ? 39 ? 40 ! 32 ! 31, v2, v2
* R29
33 ? 41 ! 42 ! 34 ? 33, v9, v9
* R30
34 ! 42 ! 43 ? 35 ? 34, v22, v22

* R31
35 ? 36 ? 0 ! 44 ! 43 ! 35, v8, v8

* R33
44 ! 45 ! 0 ? 44, v12, v12


* R36
*0 ? 45 ? 46 ! 38 ! 37 ! 0, v3, v4
0 ? 45 ? 46 ! 38 ! 37 ! 0, v3, v20
* R37
*38 ? 46 ? 47 ! 39 ! 38, v2, v20
38 ? 46 ? 47 ! 39 ! 38, v25, v20

* R38
39 ? 47 ! 48 ! 40 ? 39, v28, v27

* R39
41 ? 49 ! 50 ! 42 ? 41, v9, v9
* R40
42 ? 50 ! 51 ! 43 ? 42, v9, v10
* R41
43 ? 44 ! 52 ! 51 ? 43, v9, v9
* R42
45 ? 44 ? 52 ! 53 ! 45, v10, v1
* R43
45 ? 53 ? 54 ! 46 ! 45, v2, v2
* R44
46 ? 54 ? 55 ! 47 ! 46, v3, v20
* R45
47 ? 55 ! 56 ! 48 ? 47, v28, v27
* R46
49 ? 57 ! 58 ! 50 ? 49, v9, v9
* R47
50 ? 58 ? 59 ! 51 ! 50, v29, v29
* R48
51 ? 59 ! 60 ! 52 ? 51, v1, v19
* R49
53 ? 52 ? 60 ! 61 ! 53, v1, v19

* R50
*54 ? 53 ? 61 ! 62 ! 54, v23, v23
54 ? 53 ? 61 ! 62 ! 54, v24, v24
* R51
54 ? 62 ? 63 ! 55 ! 54, v2, v2

* R52
55 ? 63 ! 64 ! 56 ? 55, v28, v27
* R53
57 ? 65 ! 66 ! 58 ? 57, v9, v9
* R54
58 ! 66 ? 67 ? 59 ! 58, v5, v5
* R55
60 ? 68 ? 67 ! 59 ! 60, v5, v5
* R56
60 ? 61 ? 69 ! 68 ! 60, v14, v14
* R57
61 ? 62 ? 70 ! 69 ! 61, v16, v7
* R58
62 ? 70 ! 71 ! 63 ? 62, v9, v9
* R59
63 ? 71 ! 72 ! 64 ? 63, v28, v27
* R60
65 ? 73 ! 74 ! 66 ? 65, v9, v9
* R61
66 ? 74 ! 75 ! 67 ? 66, v9, v9
* R62
68 ! 67 ! 75 ? 76 ? 68, v5, v5
* R63
68 ? 69 ? 77 ! 76 ! 68, v14, v14
* R64
69 ? 70 ? 78 ! 77 ! 69, v16, v7
* R65
70 ! 78 ! 79 ? 71 ? 70, v21, v21
* R66
71 ! 79 ! 80 ? 72 ? 71, v7, v7
```

Appendix F

Complete output of Example 8.4

F.1 Output of spdi2ps

Executing 'spdi2ps example1.spdi' we obtain a 67 pages poscript file in 10.37 sec (CPU use: 90.9 %) with the partition of the plane and the vertex number of each region in the first page. In the following pages of 'example1.ps', each region is highlighted and some additional information about the edges (input and output edges and their ordering) as well as its vectors (the differential inclusion) are shown. In Figure ?? the first and second pages of the file are shown.

F.2 Output of getmafs

The application of 'getmafs example1.spdi "0-44,44-45,45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52"' gives as output:

```
The requested MAFs:
Show MAFs from SPDI file v1.0
The requested MAFs:
From edge 84 (0-44) to edge 86 (44-45):
 MAF is [1.7677669529663687x-2.5, 1.7677669529663687x-2.5]
   (accumulated MAF is
     [1.7677669529663687x-2.5, 1.7677669529663687x-2.5])
From edge 86 (44-45) to edge 103 (45-53):
 MAF is [0.2x, 0.5x]
   (accumulated MAF is
     [0.35355339059327373x-0.5, 0.8838834764831843x-1.25])
From edge 103 (45-53) to edge 88 (45-46):
 MAF is [0.5x, 0.5x]
```

```
   (accumulated MAF is
     [0.17677669529663687x-0.25, 0.44194173824159216x-0.625])
From edge 88 (45-46) to edge 71 (37-38):
 MAF is [x+1.0e-2, x+1.8333333330000001]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])
From edge 71 (37-38) to edge 68 (37-29):
 MAF is [x, x]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])
From edge 68 (37-29) to edge 66 (36-28):
 MAF is [x, x]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])
From edge 66 (36-28) to edge 67 (36-35):
 MAF is [x, x]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])
From edge 67 (36-35) to edge 85 (44-43):
 MAF is [x, x]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])
From edge 85 (44-43) to edge 100 (44-52):
 MAF is [x, x]
   (accumulated MAF is
     [0.17677669529663687x-0.24, 0.44194173824159216x+1.2083333330000001])

User time: 0.27 sec.
CPU percentage: 93.7 \%.
```

F.3 Output of showsigs

showsigs example1.spdi 0-44 58-59

```
Closure loops which may be added to the end of signatures:
   1. 58-59,51-59,52-60,53-61,62-61,69-61,69-68,76-68,67-68,67-59
   2. 58-59,51-59,52-60,53-61,62-61,70-69,77-69,76-68,67-68,67-59
   3. 58-59,51-59,52-60,53-61,54-62,54-55,46-47,39-47,48-47,56-55,64-63,
      72-71,79-71,78-70,77-8,67-68,67-59

Generating normal signatures from edge 0-44 to 58-59 ...
   1. (0-44,45-44)(45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52)*
```

```
(53-52,53-61,54-62,54-55,46-4738-39,30-31,30-22,29-21,28-20,27-19,27-26,
35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,46-47)*(-47,48-47,56-55,
64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)
   2. (0-44)(0-45,0-37,0-36,0-44)*(45-44)(45-53,45-46,37-38,37-29,36-28,
36-35,44-43,44-52)*(53-52,-61,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,
28-20,27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-,54-62,54-55,46-47)*
(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)
   3. (0-44,0-45,37-38,37-29,36-28,36-35,44-43,44-52)(45-53,45-46,37-38,
37-29,36-28,36-35,44-43,442)*(53-52,53-61,54-62,54-55,46-47)(38-39,30-31,
30-22,29-21,28-20,27-19,27-26,35-34,43-42,43-51,52-,52-60,53-61,54-62,
54-55,46-47)*(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,
67-598-59)
   4. (0-44,0-45)(0-37,0-36,0-44,0-45)*(37-38,37-29,36-28,36-35,44-43,
44-52)(45-53,45-46,37-38,37-,36-28,36-35,44-43,44-52)*(53-52,53-61,54-62,
54-55,46-47)(38-39,30-31,30-22,29-21,28-20,27-19,27-235-34,43-42,43-51,
52-51,52-60,53-61,54-62,54-55,46-47)*(39-47,48-47,56-55,64-63,72-71,79-71,
78-70,-69,76-68,67-68,67-59,58-59)
   5. (0-44,0-45,0-37,36-28,36-35,44-43,44-52)(45-53,45-46,37-38,37-29,
36-28,36-35,44-43,44-52)*(552,53-61,54-62,54-55,46-47)(38-39,30-31,30-22,
29-21,28-20,27-19,27-26,35-34,43-42,43-51,52-51,52-653-61,54-62,54-55,
46-47)*(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,
58-59)
   6. (0-44,0-45,0-37)(0-36,0-44,0-45,0-37)*(36-28,36-35,44-43,44-52)
(45-53,45-46,37-38,37-29,36-236-35,44-43,44-52)*(53-52,53-61,54-62,54-55,
46-47)(38-39,30-31,30-22,29-21,28-20,27-19,27-26,35-343-42,43-51,52-51,
52-60,53-61,54-62,54-55,46-47)*(39-47,48-47,56-55,64-63,72-71,79-71,78-70,
77-69,768,67-68,67-59,58-59)
   7. (0-44,0-45,37-38,37-29,0-36,44-43,44-52)(45-53,45-46,37-38,37-29,
36-28,36-35,44-43,44-52)*(552,53-61,54-62,54-55,46-47)(38-39,30-31,30-22,
29-21,28-20,27-19,27-26,35-34,43-42,43-51,52-51,52-653-61,54-62,54-55,
46-47)*(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,
58-59)
   8. (0-44,0-45,0-37,0-36,44-43,44-52)(45-53,45-46,37-38,37-29,36-28,
36-35,44-43,44-52)*(53-52,531,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,
28-20,27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-614-62,54-55,46-47)*
(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)
   9. ()(0-44,0-45,0-37,0-36)*(44-43,44-52)(45-53,45-46,37-38,37-29,36-28,
36-35,44-43,44-52)*(53-553-61,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,
28-20,27-19,27-26,35-34,43-42,43-51,52-51,52-60,561,54-62,54-55,46-47)*
(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)

Total number of combined signatures: 36
```

```
User time: 7.73 sec.
CPU percentage: 97.2 \%.
```

F.4 Output of reachable

reachable example1.spdi "[1,2]" "[0,10]" 0-44 58-59

```
Show feasability of going from one edge to another in an SPDI v1.0
Generating and trying signatures from edge 0-44 to 58-59 ...
Starting interval: [1.0,2.0]
Finishing interval: [0.0,10.0]
Resulting reduced graph has 69 transitions.
   1. (0-44,45-44)(45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52)*
(53-52,53-61,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,28-20,27-19,27-26,
35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,46-47)*(39-47,48-47,56-55,
64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)   <reachable>

REACHABLE

User time: 6.77 sec.
CPU percentage: 86.2 \%.
```

F.5 Output of trysig

```
   trysig example1.spdi "[1,2]" "[0,10]"
           "0-44,45-44 [45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52]
            53-52,53-61,54-62,54-55,46-47 [38-39,30-31,30-22,29-21,28-20,
            27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,
            46-47] 39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,
            67-68,67-59,58-59"

Show whether a given signature is feasible in an SPDI v1.0.

Requested signature:
  (0-44,45-44)(45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52)*
(53-52,53-61,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,28-20,27-19,
27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,46-47)*
(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)

Starting interval: [1.0,2.0]
Finishing interval: [0.0,10.0]
```

```
Reachable ([0.0,2.9519033333328584])

User time: 0.42 sec.
CPU percentage: 71.4 \%.
```

F.6 Output of simsig

```
simsig ejemplo_SPeeDI_1.spdi "[1,2]" "[0,10]"
   "0-44,45-44 [45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52]
    53-52,53-61,54-62,54-55,46-47 [38-39,30-31,30-22,29-21,28-20,
    27-19,27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,
    46-47] 39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,
    67-68,67-59,58-59"

Generate a concrete trace from an abstract signature v1.0
Requested signature:
  (0-44,45-44)(45-53,45-46,37-38,37-29,36-28,36-35,44-43,44-52)*
(53-52,53-61,54-62,54-55,46-47)(38-39,30-31,30-22,29-21,28-20,27-19,
27-26,35-34,43-42,43-51,52-51,52-60,53-61,54-62,54-55,46-47)*
(39-47,48-47,56-55,64-63,72-71,79-71,78-70,77-69,76-68,67-68,67-59,58-59)

Starting interval: [1.0,2.0]
Finishing interval: [0.0,10.0]
0-44 [1.4142135623730951,2.0]
45-44 [0.0,1.0355339059327373]
45-53 [0.0,0.5177669529663687]
45-46 [0.0,0.2588834764831843]
37-38 [1.0e-2,2.0922168094831846]
37-29 [1.0e-2,2.0922168094831846]
36-28 [1.0e-2,2.0922168094831846]
36-35 [1.0e-2,2.0922168094831846]
44-43 [1.0e-2,2.0922168094831846]
44-52 [1.0e-2,2.0922168094831846]
45-53 [2.01,7.092216809483185]
45-46 [1.005,3.5461084047415925]
37-38 [1.3333333339999998,5.379441737741592]
37-29 [1.3333333339999998,5.379441737741592]
36-28 [1.3333333339999998,5.379441737741592]
36-35 [1.3333333339999998,5.379441737741592]
44-43 [1.3333333339999998,5.379441737741592]
44-52 [1.3333333339999998,5.379441737741592]
45-53 [6.333333334,10.0]
```

```
45-46 [3.166666667,5.0]
37-38 [5.0,6.833333333000001]
37-29 [5.0,6.833333333000001]
36-28 [5.0,6.833333333000001]
36-35 [5.0,6.833333333000001]
44-43 [5.0,6.833333333000001]
44-52 [5.0,6.833333333000001]
53-52 [0.0,3.666666666000001]
53-61 [0.0,0.7333333332000003]
54-62 [1.0,1.7333333332000003]
54-55 [0.5,0.8666666666000001]
46-47 [0.51,2.6999999996]
38-39 [0.52,9.6999999996]
30-31 [0.53,10.0]
30-22 [0.53,10.0]
29-21 [0.52,9.99]
28-20 [0.51,9.98]
27-19 [0.5,9.97]
27-26 [0.5,9.97]
35-34 [0.51,9.98]
43-42 [0.52,9.99]
43-51 [0.26,9.99]
52-51 [0.26,9.99]
52-60 [2.6000000000000003e-4,1.9980000000000002]
53-61 [1.026e-2,3.998]
54-62 [1.01026,4.998]
54-55 [0.50513,2.499]
46-47 [0.51513,4.332333333]
38-39 [0.52513,9.99]
30-31 [0.53513,10.0]
30-22 [0.53513,10.0]
29-21 [0.52513,9.99]
28-20 [0.51513,9.98]
27-19 [0.50513,9.97]
27-26 [0.50513,9.97]
35-34 [0.51513,9.98]
43-42 [0.52513,9.99]
43-51 [0.262565,9.99]
52-51 [0.262565,9.99]
52-60 [2.62565e-4,1.9980000000000002]
53-61 [1.3333333339999998,3.998]
54-62 [2.3333333339999998,4.998]
54-55 [1.1666666669999999,2.499]
```

```
46-47 [3.0,4.332333333]
39-47 [0.0,1.9033333328571436]
48-47 [8.166666667,9.991903333332857]
56-55 [7.029999999999999,9.981903333332857]
64-63 [7.02,9.971903333332857]
72-71 [7.01,9.961903333332858]
79-71 [7.01,9.961903333332858]
78-70 [7.0,9.951903333332858]
77-69 [2.0,4.951903333332858]
76-68 [0.0,2.951903333332858]
67-68 [0.0,2.951903333332858]
67-59 [0.0,2.951903333332858]
58-59 [0.0,2.951903333332858]

User time: 0.40 sec.
CPU percentage: 93.4 \%.
```

Bibliography

[1] P. Abdulla, A. Annichini, and A. Bouajjani. Symbolic verification of lossy channel systems: Application to the bounded retransmission protocol. volume 1579 of *LNCS*, pages 208–222, 1999.

[2] A.D. Aleksandrov, A.N. Kolmogorov, and M.A. Lavrent'ev, editors. *Mathematics: Its content, methods and meaning*, volume 1,2,3. The MIT Press, 1963.

[3] J.F. Allen and P.J. Hayes. Moments and points in an interval-based temporal logic. *Comput. Intell.*, 5:225–238, 1990.

[4] R. Alur, C. Courcoubetis, N. Halbwachs, T.A. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138:3–34, 1995.

[5] R. Alur, C. Courcoubetis, T.A. Henzinger, and P.-H. Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, number 736 in LNCS, pages 209–229. Springer-Verlag, 1993.

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

[7] R. Alur, R. Groshu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. In N. Lynch and B.H. Krogh, editors, *HSCC'00*, volume 1790 of *LNCS*, pages 6–19. Springer-Verlag, 2000.

[8] R. Alur, T.A. Henzinger, , and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 1996.

[9] R. Alur and T.A. Henzinger. Logics and Models of Real Time: a Survey. In J. W. de Bakker, C. Huizing, W. P. de Roever, and G. Rozenberg, editors, *Real Time: Theory in Practice*, volume 600 of *LNCS*, pages 74–106. Springer-Verlag, 1992.

[10] R. Alur, T.A. Henzinger, and E.D. Sontag, editors. *Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, Rutgers University in New Brunswick, NJ, USA, October 1996. Springer.

[11] R. Alur, T.A. Henzinger, and M.Y. Vardi. Parametric real-time reasoning. In *25th ACM STOC*, pages 592–601, 1993.

[12] R. Alur, S. Kannan, and S. La Torre. Polyhedral flows in hybrid automata. In F.W. Vaandrager and J.H.van Schuppen, editors, *HSCC'99*, number 1569 in LNCS, pages 5–18. Springer-Verlag, 1999.

[13] R. Alur and R.P. Kurshan. Timing analysis in COSPAN. In R. Alur, T.A. Henzinger, and E.D. Sontag, editors, *Hybrid Systems III*, volume 1066 of *LNCS*, pages 220–231. Springer, 1996.

[14] P.J. Antsaklis, W. Kohn, Kirkland, M. Lemmon, A. Nerode, and S. Sastry, editors. *Hybrid Systems V*, volume 1567 of *Lecture Notes in Computer Science*, Notre Dame, Indiana, USA, September 1998. Springer.

[15] P.J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, Ithaca, NY, USA, October 1995. Springer.

[16] P.J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors. *Hybrid Systems IV*, volume 1273 of *Lecture Notes in Computer Science*, Ithaca, NY, USA, October 1997. Springer.

[17] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise-linear dynamical systems. In *HSCC'00*, number 1790 in LNCS. Springer Verlag, 2000.

[18] E. Asarin and O. Maler. On some relations between dynamical systems and transition systems. In S. Abiteboul and E. Shamir, editors, *ICALP'94*, number 820 in LNCS. Springer, 1994.

[19] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138:35–65, 1995.

[20] E. Asarin, G. Pace, G. Schneider, and S. Yovine. Speedi: a verification tool for polygonal hybrid systems. In *CAV'2002*, To be published in LNCS, Copenhagen, Denmark, July 2002. Springer-Verlag.

[21] E. Asarin and G. Schneider. Widening the boundary between decidable and undecidable hybrid systems. In *CONCUR'2002*, To be published in LNCS, Brno, Czech Republic, August 2002. Springer-Verlag.

[22] E. Asarin, G. Schneider, and S. Yovine. On the decidability of the reachability problem for planar differential inclusions. In M.D. di Benedetto and A. Sangiovanni-Vincentelli, editors, *HSCC'2001 (Hybrid Systems: Computation and Control)*, number 2034 in LNCS, pages 89–104, Rome, Italy, 2001. Springer-Verlag.

[23] E. Asarin, G. Schneider, and S. Yovine. Towards computing phase portraits of polygonal differential inclusions. In C.J. Tomlin and M.R. Greenstreet, editors, *HSCC'2002 (Hybrid Systems: Computation and Control)*, number 2289 in LNCS, pages 49–61, Stanford, USA, March 2002. Springer-Verlag.

[24] J.-P. Aubin. A survey on viability theory. *SIAM J. Control and Optimization*, 28(4):749–789, July 1990.

[25] J.-P. Aubin. *Viability Theory.* Birkhäuser, Berlin, 1991.

[26] J.-P. Aubin. The substratum of impulse and hybrid control systems. In *HSCC'01*, volume 2034 of *LNCS*, pages 105–118. Springer, 2001.

[27] J-P. Aubin and Arrigo Cellina. *Differential Inclusions.* Number 264 in A Series of Comprehensive Studies in Mathematics. Springer-Verlag, 1984.

[28] J.-P. Aubin, J. Lygeros, M. Quincampoix, and S. Sastry. Impulse differential inclusions: A viability approach to hybrid systems. *IEEE Transactions on Automatic Control*, 2002.

[29] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Towards a viability theory for hybrid systems. In *European Control Conference*, 2001.

[30] J.-P. Aubin, J. Lygeros, M. Quincampoix, S. Sastry, and N. Seube. Viability and invariance kernels of impulse differential inclusions. In *Conference on Decision and Control*, volume 40 of *IEEE*, pages 340–345, December 2001.

[31] A. Balluchi, L. Benvenuti, G.M. Miconi, U. Pozzi, T. Villa, M.D. Di Benedetto, H. Wong-Toi, and A.L. Sangiovanni-Vincentelli. Maximal safe set computation for idle speed control of an automative engine. In N. Lynch and B.H. Krogh, editors, *HSCC'00*, volume 1790 of *LNCS*. Springer-Verlag, 2000.

[32] N. Bauer, S. Kowalewski, and G. Sand. A case study: Multi product batch plant for the demonstration of control and scheduling problems. In *accepted to ADPM 2000*, 2000.

[33] Nanette Bauer. A demonstration plant for the control and scheduling of multi-product batch operations. VHS Case Study 7, 2000.

[34] J. Bengtsson, K.G. Larsen, F. Larsson, P. Pettersson, and Wang Yi. UPPAAL — a Tool Suite for Automatic Verification of Real–Time Systems. In *Proc. of Workshop on Verification and Control of Hybrid Systems III*, number 1066 in LNCS, pages 232–243. Springer–Verlag, October 1995.

[35] S. Bensalem, Y. Lakhnech, and S. Owre. Computing abstractions of infinite state systems compositionally and automatically. In Alan J. Hu and Moshe Y. Vardi, editors, *Computer-Aided Verification, CAV '98*, volume 1427, pages 319–331, Vancouver, Canada, 1998. Springer-Verlag.

[36] S. Bensalem, Y. Lakhnech, and S. Owre. InVeSt: A tool for the verification of invariants. In A.J. Hu and M.Y. Vardi, editors, *Computer-Aided Verification, CAV '98*, volume 1427, pages 505–510, Vancouver, Canada, 1998. Springer-Verlag.

[37] R. Bird and P. Wadler. *Introduction to Functional Programming.* Prentice Hall International, New York, 1988.

[38] V.D. Blondel and J.N. Tsitsilkis. Complexity of stability and controllability of elementary hybrid systems. *Automatica*, 35:479–489, 1999.

[39] L.S. Bobrow and M.A. Arbib. *Discrete Mathematics.* W.B. Saunders, 1974.

[40] B. Boigelot, P. Godefroid, B. Willems, , and P. Wolper. The power of QDDs. In *SAS'97*, September 1997.

[41] B. Boigelot and P. Wolper. Symbolic verification with periodic sets. In *Proceedings of the 6th International Conference on Computer Aided Verification*, volume 818 of *LNCS*, pages 55–67, 1994.

[42] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *HSCC'00*, number 1790 in LNCS. Springer Verlag, 2000.

[43] A. Bouajjani, R. Echahed, and R. Robbana. Verifying invariance properties of timed systems with duration variables. In *Intern. Symp. on Formal Techniques in Real Time and Fault Tolerant Systems (FTRTFT'94)*, number 863 in LNCS, pages 193–210, Lübeck (Germany), September 1994.

[44] A. Bouajjani and P. Habermehl. Symbolic Reachability Analysis of FIFO Channel Systems with Nonregular Sets of Configurations (extended abstract). In *Automata, Languages and Programming, 24th International Colloquium*, volume 1256 of *LNCS*, pages 560–570. Springer-Verlag, July 1997.

[45] A. Bouajjani and R. Robbana. Verifying omega-regular properties for a subclass of linear hybrid systems. In *7th Intern. Conf. on Computer Aided Verification (CAV'95)*, number 939 in LNCS, pages 437–450, Liege (Belgium), July 1995.

[46] Olivier Bournez. *Complexité algorithmique des systèmes dynamiques continus et hybrides.* PhD thesis, ENS de Lyon, 1999.

[47] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Are timed automata updatable? In *12th Int. Conf. Computer Aided Verification (CAV'2000), Chicago, IL, USA*, volume 1855 of *LNCS*, pages 464–479. Springer, July 2000.

[48] P. Bouyer, C. Dufourd, E. Fleury, and A. Petit. Expressiveness of updatable timed automata. In *25th Int. Symp. Math. Found. Comp. Sci. (MFCS'2000), Bratislava, Slovakia*, volume 1893 of *LNCS*, pages 232–242. Springer, Aug. 2000.

[49] Michael S. Branicky. Universal computation and other capabilities of hybrid and continuous dynamical systems. *Theoretical Computer Science*, 138(1), 1995.

[50] F. Brauer and J.A. Nohel. *The qualitative theory of ordinary differential equations: an introduction.* Dover, 1969.

[51] M. Broucke. A geometric approach to bisimulation and verification of hybrid systems. In *HSCC'99*, number 1569 in LNCS. Springer Verlag, 1999.

[52] A. Chutinan. *Hybrid System Verification Using Discrete Model Approximations*. PhD thesis, Carnegie Mellon University, Department of Electrical and Computer Engineering, May 1999.

[53] A. Chutinan and B.H. Krogh. Computing polyhedral approximations to dynamic flow pipes. In *Proc. of the 37th Annual International Conference on Decision and Control, CDC'98*. IEEE, 1998.

[54] Edmund M. Clarke, Orna Grumberg, and Doron A. Peled. *Model checking*. The MIT Press, 1999.

[55] E.M. Clarke and E.A. Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Proceedings of the Workshop on Logics of Programs*, volume 131 of *LNCS*, pages 52–71. Springer-Verlag, May 1981.

[56] E.M. Clarke, E.A. Emerson, and A.P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specification. *ACM Trans. on Programming Languages and Systems*, 8(2):244–263, April 1986.

[57] T. Coquand and G. Huet. The calculus of construction. *Information and Computation*, 76(2), 1988.

[58] P. Cousot. Abstract interpretation. *Symposium on Models of Programming Languages and Computation, ACM Computing Surveys*, 28(2):324–328, jun 1996.

[59] P. Cousot and R. Cousot. Abstract interpretation: a unified lattice model for static analysis of programs by construction or approximation of fixpoints. In *Conference Record of the Fourth Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 238–252, Los Angeles, California, 1977. ACM Press, New York, NY.

[60] H.C. Cunningham. Notes on functional programming with gofer. Technical Report UMCIS–1995–01, Department of Computer and Information Science, University of Mississippi, 1995.

[61] Mads Dam. Temporal Logic, Automata and Classical Theories: an Introduction, 1994. Notes for the Sixth European Summer School in Logic, Language and Information. Copenhagen.

[62] T. Dang. d/dt manual. Technical report, Verimag, Grenoble, 2000.

[63] T. Dang and O. Maler. Reachability analysis via face lifting. In *HSCC'98*, number 1386 in LNCS, pages 96–109. Springer Verlag, 1998.

[64] Thao Dang. *Vérification et synthèse des systèmes hybrides*. PhD thesis, VERIMAG, INPG, October 2000.

[65] A. Deshpande and P. Varaiya. Viable control of hybrid systems. In *Hybrid Systems II*, number 999 in LNCS, pages 128–147, 1995.

[66] R.L. Devaney. *An Introduction to Chaotic Dynamical Systems*. Addison-Wesley, Redwood City, 2nd edition, 1989.

[67] M.D. di Benedetto and A. Sangiovanni-Vincentelli, editors. *Hybrid Systems: Computation and Control*, volume 2034 of *Lecture Notes in Computer Science*, Rome, Italy, March 2001. Springer.

[68] J. Della Dora and S. Yovine. Looking for a methodology for analyzing hybrid systems. In *Submitted to ECC 2001*, 2001.

[69] J.J.H. Fey and J.H. van Schuppen. VHS case study 4 - modeling and control of a juice processing plant. draft, 1999.

[70] A.J. Field and P.G. Harrison. *Functional Programming*. Addison Wesley, Reading, Massachusetts, 1988.

[71] A.F. Filippov. Differential equations with discontinuous right-hand side. *Sbornik*, 5:99–127, 1960. (In Russian). English trans. in Amer. Math. Soc. Translations,42:199-231, 1964.

[72] A.F. Filippov. *Differential equations with discontinuous righthand sides*. Kluwer Acaemic, 1988.

[73] M. R. Greenstreet and I. Mitchell. Reachability analysis using polygonal projections. In *HSCC'99*, number 1569 in LNCS, pages 103–116. Springer Verlag, 1999.

[74] R.L. Grossman, A. Nerode, A.P. Ravn, and H. Rischel, editors. *Hybrid Systems*, volume 736 of *Lecture Notes in Computer Science*. Springer-Verlag, 1993.

[75] J. Guckenheimer and P. Holmes. *Nonlinear Oscillations, Dynamical Systems and Linear Algebra*. Springer-Verlag, New York, 1990.

[76] J. Guckenheimer and S. Johnson. Planar hybrid systems. In *Hybrid Systems and Autonomous Control Workshop*, pages 202–225, 1994.

[77] Michael R. Hansen and Zhou Chaochen. Lecture Notes on Logical Foundations for the Duration Calculus. Lecture Notes, 13, UNU/IIST, P.O.Box 3058, Macau, August 1993.

[78] Michael Henle. *A combinatorial introduction to topology*. Dover publications, Inc., 1979.

[79] T.A. Henzinger, P-H. Ho, and H. Wong-Toi. Hytech: The next generation. In *Proc. IEEE Real-Time Systems Symposium RTSS'95*, Pisa, Italy, December 1995.

[80] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *27th Annual Symposium on Theory of Computing*, pages 373–382. ACM Press, 1995.

[81] T.A. Henzinger, P.-H.Ho, and H.Wong-toi. Hytech: A model checker for hybrid systems. *Software Tools for Technology Transfer*, 1(1), 1997.

[82] Thomas A. Henzinger. The theory of hybrid automata. In *11th Annual IEEE Symposium on Logic in Computer Science (LICS 1996)*, pages 278–292, 1996.

[83] Morris W. Hirsch and Stephen Smale. *Differential Equations, Dynamical Systems and Linear Algebra.* Academic Press Inc., 1974.

[84] H. Hochstadt. *Differential equations: a modern approach.* Dover, 1963.

[85] G.J. Holzmann. *Design and Validation of Computer Protocols.* Prentice Hall, 1991.

[86] G.J. Holzmann and D. Peled. The state of spin. In *CAV'96: 8th International Conference on Computer Aided Verification*, volume 1102 of *LNCS*, pages 385–389, 1996.

[87] J.E. Hopcroft and J.D. Ullman. *Introduction to Automata Theory, Languages and Computation.* Addison-Wesley, 1979.

[88] G. Huet, G. Kahn, and C. Paulin-Mohring. The coq proof assistant - a tutorial, version 6.1. Technical report, INRIA, 1997.

[89] Simon Peyton Jones and John Hughes. Report on Haskell 98: A non-strict, purely functional language, 1999. available from `http://www.haskell.org`.

[90] Y. Kesten, A. Pnueli, J. Sifakis, and S. Yovine. Integration graphs: a class of decidable hybrid systems. In *Hybrid Systems*, number 736 in LNCS, pages 179–208. Springer, 1993.

[91] P. Koiran. My favourite problems. http://www.ens-lyon.fr/˜koiran/problems.html.

[92] P. Koiran, M. Cosnard, and M. Garzon. Computability with low-dimensional dynamical systems. *Theoretical Computer Science*, 132(1):113–128, 1994.

[93] P. Koiran and C. Moore. Closed-form analytic maps in one and two dimensions can simulate universal Turing machines. *TCS (Special Issue on Real Numbers)*, 210:217–223, 1999.

[94] A.N. Kolmogorov and S.V. Fomin. *Introductory Real Analysis.* Dover, 1975.

[95] M. Kourjanski and P. Varaiya. Stability of hybrid systems. In *Hybrid Systems III*, number 1066 in LNCS, pages 413–423. Springer, 1996.

[96] P. Kowalczyk and M. di Bernardo. On a novel class of bifurcations in hybrid dynamical systems. In *HSCC'01*, number 2034 in LNCS. Springer, 2001.

[97] A.B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *HSCC'00*, number 1790 in LNCS. Springer Verlag, 2000.

[98] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *Hybrid Systems : Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 137–151. Springer Verlag, 1999.

[99] G. Lafferriere, G.J. Pappas, and S. Sastry. O–Minimal hybrid systems. Technical Report UCB/ERL M98/29, Electronics Research Laboratory, University of California, Berkeley, May 1998.

[100] M. Leucker, T. Noll, P. Stevens, and M. Weber. Functional programming languages for verification tools: Experiences with ML and Haskell. In *Proceedings of the Scottish Functional Programming Workshop (SFPW'01)*, 2001.

[101] H.R. Lewis and C.H. Papadimitriou. *Elements of the Theory of Computation*. Prentice-Hall, 1981.

[102] A.M. Lyapunov. *The general problem of stability of motion*. PhD thesis, Kharkov Mathematical Society, Kharkov, 1892. (In Russian).

[103] N. Lynch and B.H. Krogh, editors. *Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*. Springer-Verlag, 2000.

[104] O. Maler, editor. *Hybrid and Real-Time Systems, HART'97*, volume 1201 of *Lecture Notes in Computer Science*. Springer-Verlag, 1997.

[105] O. Maler and A. Pnueli. Reachability analysis of planar multi-linear systems. In C. Courcoubetis, editor, *Proceedings of the 4th Computer-Aided Verification*. LNCS 697, Springer Verlag, July 1993.

[106] O. Maler and A. Pnueli. Reachability analysis of planar multi-linear systems. In C. Courcoubetis, editor, *Proc. of the 5th Workshop on Computer-Aided Verification*, number 697 in LNCS, pages 194–209. Springer-Verlag, 1993.

[107] Zohar Manna, Nikolaj Bjorner, Anca Browne, Edward Y. Chang, Michael Colon, Luca de Alfaro, Harish Devarajan, Arjun Kapur, Jaejin Lee, Henny Sipma, and Tomas E. Uribe. Step: The stanford temporal prover. In *TAPSOFT*, pages 793–794, 1995.

[108] A. Marchaud. Sur les champs de demi-cones et les équations differentielles du premier ordre. *Bull. Soc. Math. France*, 62, 1934.

[109] A. Matveev and A. Savkin. *Qualitative theory of hybrid dynamical systems*. Birkhäuser Boston, 2000.

[110] Kenneth L. McMillan. *Symbolic Model Checking*. Kluwer Academic Publisher, 1993.

[111] R. Milner. *Communication and Concurrency*. Prentice Hall Int., 1989.

[112] M.L. Minsky. *Computation: Finite and Infinite Machines*. Prentice-Hall, Englewood Cliffs, 1967.

[113] Cristopher Moore. Unpredictability and undecidability in dynamical systems. *Physical Review Letters*, 64(20), 1990.

[114] B. Moszkowski. A temporal logic for multi-level reasoning about hardware. *IEEE Computer*, 2(18), 1985.

[115] V.V. Nemytskii and V.V. Stepanov. *Qualitative theory of differential equations.* Princeton University Press, 1960.

[116] X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. An approach to the description and analysis of hybrid systems. In *Hybrid Systems*, number 736 in LNCS, pages 149–178. Springer, 1993.

[117] A. Olivero, J. Sifakis, and S. Yovine. Using abstractions for the verification of linear hybrid systems. In *6th Computer-Aided Verification, CAV'94*, number 818 in LCNS, pages 81–94, July 1994.

[118] S. Owre, J. Rushby, and N. Shankar. PVS : a prototype verification system. In *11th Conf. on Automated Deduction*, volume 83 of *LNCS*, pages 748–752, 1992.

[119] Christos H. Papadimitriou. *Computational complexity.* Addison-Wesley, 1994.

[120] L.C. Paulson. *Isabelle: A Generic Theorem Prover*, volume 828 of *LNCS*. Springer, 1994. (With contributions by Tobias Nipkow).

[121] A. Pnueli. The temporal logic of programs. In *Proceedings of the 18th IEEE Symposium on the Foundations of Computer Science (FOCS-77)*, pages 46–57, 1977.

[122] H. Poincaré. *Mémoire sur les courbes définies par les équations différentielles I-IV, Ouvre I.* Gauthier-Villar: Paris, 1880–1890.

[123] H. Poincaré. Sur les équations de la dynamique et le problème des trois corps. *Acta Math.*, 13:1–270, 1890.

[124] H. Poincaré. *Les méthodes nouvelles de la mécanique céleste, 3 Vols.* Gauthier-Villars: Paris, 1899.

[125] A.N. Prior. *Time and Modality.* Oxford: Clarendon Press, 1957.

[126] PATH Project. http://paleale.eecs.berkeley.edu/.

[127] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In *CAV 94*, number 818 in LNCS, pages 95–104. Springer, 1994.

[128] A. Puri, P. Varaiya, and V. Borkar. Epsilon approximations of differential inclusions. In *Hybrid Systems III*, volume 1066 of *LNCS*, pages 362–376. Springer, 1996.

[129] J.P. Queille and J. Sifakis. A temporal logic to deal with fairness in transition systems. In *23rd Annual Symposium Foundations of Computer Science*, pages 217–225, 1982.

[130] S. Raczynski. Differential inclusion in systems simulation. *Trans. of the Society for Computer Simulation*, 13(1), 1996.

[131] Hartley Rogers. *Theory of recursive functions and effective computability*. The MIT Press, 1987.

[132] K.S. Sibirsky. *Introduction to Topological Dynamics*. Noordhoff International Publishing, Leyden, 1975.

[133] S. Simić, K. Johansson, S. Sastry, and J. Lygeros. Towards a geometric theory of hybrid systems. In *HSCC'00*, number 1790 in LNCS. Springer, 2000.

[134] Colin Stirling. Modal and Temporal Logics. In S. Abramsky, Dov M. Gabbay, and T.S.E. Maibaum, editors, *Handbook of Logic in Computer Science. Background: Computational Structures*, volume 2 of *Handbook of Logic in Computer Science*, pages 477–563. Clarendon Press - Oxford, 1992.

[135] C. Tomlin, J. Lygeros, and S. Sastry. Conflict resolution for air traffic management: A study in multi-agent hybrid systems. In *Trans. on Automatic Control*, volume 43 of *IEEE*, 1998.

[136] C.J. Tomlin and M.R. Greenstreet, editors. *Hybrid Systems: Computation and Control*, volume 2289 of *Lecture Notes in Computer Science*, Stanford, CA, USA, March 2002. Springer.

[137] S. Tripakis and C. Courcoubetis. Extending Promela and SPIN for real time. In *TACAS'96*, volume 1055 of *LNCS*, Passau, Germany, 1996. Springer-Verlag.

[138] K. Čerāns. *Algorithmic problems in analysis of real-time systems specifications*. PhD thesis, Univ. of Latvia, 1992.

[139] K. Čerāns and J. Vīksna. Deciding reachability for planar multi-polynomial systems. In *Hybrid Systems III*, number 1066 in LNCS. Springer Verlag, 1996.

[140] K. Čerāns and J. Vīksna. Deciding reachability for planar multi-polynomial systems. In *Hybrid Systems III*, number 1066 in LNCS. Springer Verlag, 1996.

[141] F.W. Vaandrager and J.H. van Schuppen, editors. *Hybrid Systems : Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, Berg en Dal near Nijmegen, The Netherlands, March 1999. Springer-Verlag.

[142] S. Yovine. Kronos: A verification tool for real-time systems. *International Journal of Software Tools for Technology Transfer*, 1(1/2):123–133, October 1997.

[143] S. Yovine. Model-checking timed automata. In G. Rozenberg and F. Vaandrager, editors, *Lectures on Embedded Systems*, volume 1494 of *LNCS*. Springer-Verlag, 1998.

[144] S.K. Zaremba. Sur les équations au paratingent. *Bull. Sc. Math. France*, 60, 1936.

Index

Glossary

$EV(P)$, set of vertex and open edges of P, 26
$\mathcal{D}_\sigma$, one-dimensional discrete-time dynamical system, 96
$\mathcal{E}$, set of edges, 35
$\mathcal{G}_\mathcal{H}$, underlying graph of $\mathcal{H}$, 136
$\mathtt{int}(\cdot)$, interior, 25
$In(P)$, set of entry edges of P, 31
$Out(P)$, set of exit edges of P, 31
$\mathbb{P}$, partition of the plane, 26
$\overline{\mathsf{Pre}}_e(\cdot)$, e-extended predecessor, 99
$\overline{\mathsf{Succ}}_e(\cdot)$, e-extended successor, 102
$\mathsf{Succ}_{ee'}$, edge-to-edge successor, 73
$\mathsf{Succ}^{\mathbf{c}}_{ee'}$, edge-to-edge successor following a given vector $\mathbf{c}$, 72
$\mathcal{T}$, set of types of signatures, 66
$\mathcal{T}(e_0, e_f)$, set of types of signatures from e_0 to e_f, 66
$\mathcal{T}_{feasi}$, set of types of feasible signatures, 66
$\angle_{\mathbf{a}}^{\mathbf{b}}$, angle defined by two vectors $\mathbf{a}, \mathbf{b}$, 33
$\mathsf{Cntr}(\cdot)$, controllability kernel, 102
$\mathtt{Concr}(\tau)$, concretization of τ, 66
$\dot{\xi}^+(\cdot)$, right derivative of ξ, 31
$\hat{\mathbf{x}}$, clockwise rotation of vector $\mathbf{x}$ by the angle $\pi/2$, 25
$|\cdot|$, 2-norm, 25
$\prec$, order on edges, 58
$\mathsf{RSig}(\xi)$, region signature of ξ, 56
$\mathsf{Sig}(\xi)$, edge signature of ξ, 56
σ, edge signature, 56
$\mathtt{trace}(\xi)$, trace of ξ, 56
τ, type of signature, 66
$\mathtt{type}(\sigma)$, type of the signature σ, 65
$\mathbf{a}\,\mathbf{b}$, inner product of vectors $\mathbf{a}$ and $\mathbf{b}$, 25
ε, empty sequence, 62
$\mathsf{Viab}(\cdot)$, viability kernel, 98
$\xi^-(\cdot)$, left limit of ξ, 28

RÉSUMÉ : Les *systèmes polygonaux à inclusions différentielles* (SPDIs) sont des systèmes planaires non déterministes qui peuvent être représentés par des inclusions différentielles constantes par morceaux. Cette thèse porte sur les aspects théoriques et pratiques des SPDIs tels que le problème de l'atteignabilité et de la construction du portrait de phase. Nous montrons que le problème de l'atteignabilité est décidable pour les SPDIs. Notre procédure est basée sur le calcul des limites des trajectoires individuelles : l'idée sous-jacente est l'utilisation de fonctions de Poincaré unidimensionelles, pour lequelles on peut facilement calculer les points fixes et qui permettent dans la plupart des cas d'accélérer les cycles. Nous avons implanté cet algorithme d'atteignabilité dans l'outil SPeeDI. Ensuite, nous construisons le portrait de phase des SPDIs. Nous savons identifier les *noyaux de viabilité* des boucles simples. Il s'agit des ensembles de points initiaux de trajectoires restant dans la boucle. Nous introduisons la notion de *noyau de controlabilité* de boucles simples comme l'ensemble des points atteignables les uns à partir des autres par des trajectoires qui restent dans le noyau. Nous proposons un algorithme non itératif pour calculer ces deux noyaux, qui nous permet ensuite de construire le portrait de phase des SPDIs. Enfin, nous étudions la décidabilité du problème de l'atteignabilité pour d'autres classes de systèmes hybrides à deux dimensions : les *systèmes hiérarchiques constants par morceaux* (HPCDs) et les *systèmes constants par morceaux, définis sur les surfaces*. Nous montrons que le problème de l'atteignabilité pour ces deux classes de systèmes est équivalent à l'atteignabilité pour des systèmes affines par morceaux, dont la décidabilité est un problème ouvert. Nous montrons enfin que le problème de l'atteignabilité pour quelques extensions de HPCDs est indécidable.

MOTS CLES : SPDI, Inclusions Différentielles, Systèmes Hybrides, Vérification, Décidabilité, Analyse d'atteignabilité, Portrait de Phase, Controllabilité, Viabilité.

TITLE: Algorithmic Analysis of Polygonal Hybrid Systems

ABSTRACT: A *polygonal differential inclusion system* (SPDI) is a non-deterministic planar hybrid system which can be represented by piecewise constant differential inclusions. In this thesis we are concerned with several theoretical and practical questions related to SPDIs such as reachability analysis and phase portrait construction. First we show that the reachability question for SPDIs is indeed decidable. Our procedure is not based on the computation of the reach-set but rather on the computation of the limit of individual trajectories. A key idea is the use of edge-to-edge one-dimensional affine Poincaré maps, the fix-points of which are easily computed. By taking advantage of this information, cycles can be *accelerated* in most cases. The above reachability algorithm has been implemented in a tool called SPeeDI. We next build the phase portrait of such systems. In particular, we identify the *viability* kernels of simple cycles. Such kernels are the set of starting points of trajectories that can keep rotating in the cycles forever. We also introduce the notion of *controllability* kernel of simple cycles as the set of points such that any two points of the set are reachable from each other *via* trajectories that remain on the set. We give non-iterative algorithms to compute both kernels. We obtain the SPDI phase portrait computing all the viability and controllability kernels. We finally study the decidability of the reachability problem for other 2-dimensional hybrid systems. We introduce *hierarchical piecewise constant derivative systems* (HPCDs) and *2-dimensional manifolds with piecewise constant derivative systems*. We show that the reachability problem for the above two classes of systems is as hard as the reachability problem for *piecewise affine maps* that is known to be an open problem. We also show that the reachability question for slight extensions of HPCDs are undecidable.

KEYWORDS: SPDI, Differential Inclusions, Hybrid Systems, Verification, Decidability, Reachability Analysis, Phase Portrait, Controllability, Viability.

DISCIPLINE : Informatique

Thèse préparée au laboratoire **VERIMAG**, Centre Équation, 2 avenue de Vignate, 38610 Gières, France.
http://www-verimag.imag.fr