Dottorato di Ricerca
Scienze Computazionali ed Informatiche
Ciclo XXIV

Università degli Studi di Napoli - Federico II

Synthesis of Switching Controllers for Linear Hybrid Systems

Stefano MINOPOLI

Novembre 2011

Coordinatore:
Prof. Ernesto BURATTINI

Relatori:
Prof. Massimo BENERECETTI
Dott. Marco FAELLA

2

.

To my Family

Abstract

This thesis is focused on the problem of automatically generating switching controllers for the class of Linear Hybrid Game, with respect to safety and reachability objectives.

In order to solve the safety control problem, a sound and complete symbolic fix-point procedure on the so-called controllable predecessor operator for safety, called $CPre^{\mathrm{S}}$ and based on polyhedral abstractions of the state space, are provided and the termination of each iteration is proved.

Some inaccuracies contained in previous characterizations of the problem are identified and solved. In particular, this work shows that the algorithm provided by Wong-Toi [WT97] does not work in some case which are very likely to occur in practice. The error is identified in the heart of this algorithm (the operator $flow\_avoid$), and is here fixed by proposing a sound and complete procedure, based on a novel algorithm for computing, within a given location of the automaton, the *may reach while avoiding operator* RWA^{m}, that is the set of states that may reach a given polyhedral region while avoiding another one.

The reachability control problem for hybrid games was never considered in the literature, and the task is not trivially due the fact that, unlike classical results for discrete and real-timed case, the reachability control problem is not dual to the safety control problem. Hence, in order to solve this problem an entirely new study was necessary. This thesis proposed a sound and complete fix-point procedure based on a novel algorithm for computing, within a given location of the automaton, the set of *must reach while avoiding operator* RWA^{M}, that is the set of states that must reach a given polyhedral region while avoiding another one.

The theoretical results of this thesis are then effectively and efficiently implemented on the top of the open source tool PHAVer. The obtained tool, called PHAVer+, is used to present some promising experimental results at the end of this thesis.

Contents

List of Figures

List of Tables

Introduction

This thesis is focused on the problem of automatically synthesizing a switching controller for linear hybrid systems with respect to safety and reachability objectives. In Computer Science, this kind of problem is studied in the context of *synthesis*, where the considered systems are "open", in the sense that they cooperate with an *environment*.

This is the main difference with respect to other context, like *verification* (e.g. *model checking*), where the considered systems are generally "closed" by the environment and need a complete system model which is verified toward satisfying a set of properties.

Open systems can be seen as compositions of elements, whose relations and interactions are governed by known laws (e.g. a mechanical system governed by Newton's laws), where the environment generates input events triggering actions on the system that cause a change of the system state, which may in turn produce output events that affect the environment.

The model of this kind of systems can be considered "incomplete" in the sense that it describes a more liberal behavior and usually the question arises of restricting the choices of the system so that some desired goals are met, by means of appropriate control. The control of the system is governed by an appropriate device, called *controller*.

Based on the characteristics of the system components and their interactions, systems can be classified as discrete or continuous, time-invariant or time-varying, linear or nonlinear, deterministic or nondeterministic etc. Also, the controller can be classified as discrete or continuous, depending on the type of device (e.g. *actuators*) on which the controller acts.

This work is focused on continuous open systems (namely, hybrid systems whose definition will be given later) and digital controllers.

This category of systems is one of the most studied in the field of Control Theory, where the system together with the environment in which it is located can be referred to a *plant* (e.g. the subject of control). Its correct behavior

is achieved by designing the controller that will interact with the plant, whose behaviors are modeled by means of differential equations over variables that describe the plant properties.

The control diagram representing this situation is shown in Figure 1. Since the open loop behavior of the plant is often unsatisfactory, the role of the feedback after determining the current state of the plant is to take corrective action by issuing appropriate control commands. It is of fundamental importance to take into account the environment in which our system operates in order to deal with unknown actions (or disturbances) caused by the environment. These disturbances may have disastrous consequences when they are not taken into account.

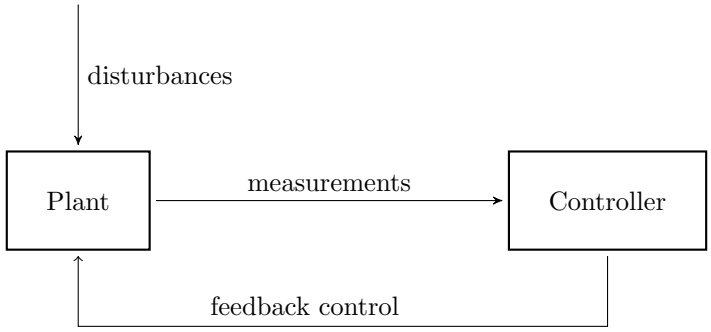


Figure 1: Schema of a control diagram.

In everyday life it is easy to find such systems, where the controller interact with discrete actuators (digital controller). Digital controllers represent in fact a pervasive technology in our societies. Civil and industrial automation, as well as transportation systems are the main domains where such devices are employed. For instance, in a modern car, up to 50 different controllers may be operating at the same time, directing crucial, potentially life-saving car features like braking, anti-lock braking system (ABS), electronic stability program (ESP), etc. The job of each of these controllers is to generate a signal that guides the evolution of the physical system toward a desired goal. The controller may base its decisions on a set of signals received by sensors or by the user (the meta-controller, in a sense).

Open systems as the above are loosely defined as hybrid systems. From an abstract point of view, a hybrid system is an open system whose state variables are partitioned into discrete and continuous ones. Typically, continuous variables represent physical quantities like temperature, speed, etc., while discrete ones represent *control modes*, e.g., states of the controller.

For example, consider the case of anti-lock braking system (ABS), that is a system governed by a controller whose objective is to avoid the blocking of the wheels, regardless of the behavior of the driver, that can be viewed as the external environment. Figure 2 shows the schema of an ABS: several speed sensors (e.g. phonic wheels) read the instantaneous speed of each wheel. The controller reads these values (dotted arrows in the figure) and compares them. In this way it may evaluate the level of locking of each single wheel. If some wheel is close to the locked state, the controller acts (thicked arrows in the figure) immediately on the actuators, e.g. by reducing the pressure on the brake of the almost locked wheel. In this way the controller avoids the locking of the wheels and when the behavior of the vehicle returns to the ideal condition it restores the normal pressure of the brake involved. The cycle "reduce pressure-maintain the right pressure-restore pressure" is repeated several times each second and, in emergency braking, allows to keep all wheels on the right speed, ensuring the optimal braking. Notice that this goal is achieved regardless of the behavior of the driver (represented by the dashed arrows in the figure): the controller may reduce the brake power even if the driver is performing a brake.

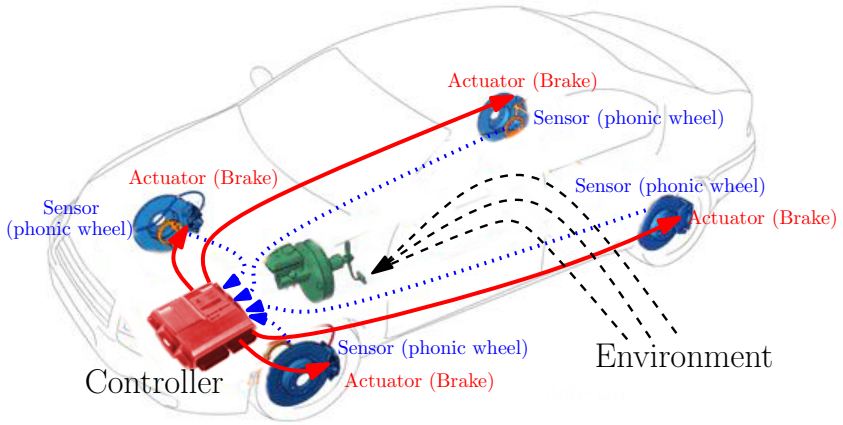


Figure 2: The anti-lock braking system (ABS).

Notice that the global actions of the whole systems can be divided into those that the controller can govern, e.g. reducing the related pressure of a brake (controllable actions) and those that are not under its responsibility, e.g. the pressure on the brake pedal (uncontrollable actions, or environment actions). In addition, the variables that describe the state of the systems can be divided into discrete ones, e.g. on-off status of a LED indicating the entry into operation of the ABS, and into continuous ones, e.g. the values of the wheels speed.

While in control theory discrete variables are hard to model as differential equations, computer science in the opposite is typically specialized on discrete systems. In particular, in order to model also continuous time dynamics, computer science proposes the formalism of *hybrid automata* [Hen96], that is the most common syntactic variety of hybrid system: a finite set of locations, similar to the states of a finite automaton, represents the value of the discrete variables, used to model modes of operation of the system, such as braking mode in a car equipped with ABS. Change of location happens via discrete transitions (change of control modes) that can model the evolution of the discrete variables, while the evolution of the continuous variables is governed by differential equations attached to each location and, for example, can model the continuous response of the car wheels to the force of brake pedal (corresponding to the handle of sensors and actuators in control theory).

Depending on which kind of differential equations are allowed, it is possible to distinguish several classes and related subclasses of hybrid automaton.

The instantaneous description of an hybrid automata is formed by the current location together with the current value of the (continuous) variables.

In work to date, a number of problems for hybrid automata have been studied:

1. *Optimal Control*: roughly speaking, the optimal control problem is to drive the system to a desirable state while minimizing a cost function that depends on the path followed. It typically involves a terminal cost (depending on the terminal state), an integral cost accumulated along continuous evolution, and a series of jump costs associated with discrete transitions. This is a classical problem for continuous systems, extended more recently to discrete systems [SL98], and to classes of hybrid systems with simple continuous dynamics [AM99]. The approach has been extended to general hybrid systems both for the dynamic programming formulation [BBM98] and for the variational formulation, extending the maximum principle [Gra99].

2. *Hierarchical Control*: this describes the systematic decomposition of control tasks so that the resulting hierarchical controller guarantees a certain performance [CW98, PLS00];

3. *Distributed, Multiagent Control*: here, optimal control problems are decomposed so that they can be solved in a distributed way by a collection of agents with a specified communication and information architecture [KN93].

4. *Least Restrictive Controllers for Specifications Such as Safety and Reachability*: here it is required that all trajectories of the system satisfy certain goals, as safety (for example, requiring that the system remains in a certain set of safe states) and reachability (requiring that the system eventually enter in a certain set of target states).

This thesis is focused on the last problem for the subclass of *Linear Hybrid Automaton (LHA)*, where the allowed differential equations are in fact differential inclusions of the type $\dot{\mathbf{x}} \in P$, where $\dot{\mathbf{x}}$ is the vector of the first derivatives of all variables and $P \subseteq \mathbb{R}^n$ is a convex polyhedron. Notice that differential inclusions are non-deterministic, allowing for infinitely many solutions.

Due the fact that the considered systems are open systems, it is needed to extend the formalism of LHAs, in order to take into account also the actions of the environment. To this aim the formalism of *Linear Hybrid Games (LHGs)*, defined as LHAs whose discrete transitions are partitioned into controllable and uncontrollable ones, is introduced.

The main goals of this work is to compute the set of states from which the controller can ensure a given goal, regardless the behavior of the environment that may affect the trajectory followed by the continuous variables and may choose to issue any enabled uncontrollable discrete transition. Hence, the problem can be viewed as a *two player game* [TLSS00]: on one side the controller, who can only issue controllable transitions, on the other side the environment, who can choose the trajectory of the variables and can take uncontrollable transitions whenever they are enabled. The considered goals are safety and reachability. The safety goal consists in the objective of keeping the systems within a given set T of so-called "safe" states. While the reachability goal consists in the objective of reaching a given set T of so-called "target" states. Both problems are known to be undecidable, being at least hard as the standard reachability verification (i.e., 1-player reachability) for triangular hybrid automata [HKPV95], a special case of LHAs.

While the reachability control problem for linear hybrid games was never considered in the literature, for the safety control problem, there is an extensive literature describing approximate solutions [ABD+00], or solutions claimed to be exact [WT97], based on a fix-point procedure [1] on the so-called *controllable predecessor operator* (*CPre*).

The main contributions of this thesis may be listed below:

1. Previous solutions proposed (e.g. Wong Toi et other) for the safety control

[1] In this work the term "procedure" means a step-by-step procedure that may or may not terminate.

problem of linear hybrid games, are showed not to not work in several cases which are very likely to occur in practice.

2. A sound a complete procedure is proposed that fixes inaccuracies of the Wong Toi procedure, that solve the safety control problem for linear hybrid games.

3. A sound a complete procedure is proposed that solves the reachability control problem for linear hybrid games.

4. The algorithms are implemented on the top of the verification tool PHAVer [Fre05], based on the polyhedral abstraction provided by the Parma Polyhedra Library [BHZ08]. The obtained tool is called PHAVer+.

5. Several techniques required to efficiently implement the above algorithms, are discussed.

6. Some need operators on polyhedra are introduced, and some existing PPL operators are or refining, in order to exact implement the above algorithms.

At the end of the thesis, several meaningful example of linear hybrid games are defined and tested by the tool PHAVER+, and the corresponding experimental evaluation is showed.

Summary

The rest of this thesis is divided into three parts: Part I describes the states of the art, first for the class of discrete systems and then for the real-time systems. The last chapter of Part I introduces the formalism used to model hybrid systems. Part II introduces the original contributions of this research: the solutions of the controller synthesis problem for linear hybrid games, w.r.t. safety and reachability goals. Part III shows the implementations of the proposed algorithms and then shows the results and the performance of the implemented tool on some examples.

Part 1. The topic of Chapter 1 is to introduce some basic notions about two-players games, in order to correctly address the reader to the representation of an open system as a game. The considered systems are discrete, and the formalism of the *Game Graphs (GGs)* is introduced in order to model this kind of systems. Notions such as strategy, winning strategy and memoryless strategy are formally defined. Then the control problem is defined as the problem of finding a winning

strategy for the corresponding game. Finally, fix-point algorithms that solve the control problems, based on the controllable predecessor operator, are showed.

Chapter 2 is an introduction to the control problems for open continuous systems, where the considered continuous systems are in fact real-time systems. Several notions are given and extended in order to correctly handle the real-time properties of these systems, modeled by the formalism of *Timed Games (TG)* (an extension of the formalism of the *Timed Automata*). For example, it is defined the notion of *timed* strategy and the controllable predecessor operator is redefined in order to take into account the real-time properties of a timed game.

The core of Chapter 3 is to fully explain a more powerful kind of continuous systems than real-time, called hybrid systems, that allow to express more complete interaction between the environment and the system. The formalism of *Hybrid Games (HGs)* is introduced by giving the syntax and the semantics, and then some proper subclasses are identified and classified. Also, it is shown how the problem of *Zeno* is managed. At the end of the chapter, the safety and the reachability control problem are formally defined.

Part 2. Chapter 4 and Chapter 5 describe solutions of the control problem for linear hybrid games w.r.t. safety goals and reachability goals, respectively. In order to implement the proposed algorithms, several operators on polyhedra are introduced, whose implementation is proved to be sound and complete.

Part 3. Techniques required to efficiently implement these algorithms on the top of the tool PHAVer are showed in Chapter 6, while Chapter 7 shows some experimental results, divided into two parts: the first one shows performance of the obtained tool (PHAVer+) on examples for safety and reachability. The second one shows performance of single calls to the different implementations of the basic operator used by the algorithms.

Part I

Control Problems for Several Classes of Games

Chapter 1

Control Problems for Discrete Systems

This chapter is focused on open discrete systems and on the computation of a controller that achieves some goal regardless of the behavior of the environment. This is the *controller synthesis problem*[RW87] for discrete systems. Pnueli and Rosner [PR89] addressed this problem as a two-player game, where one of the players is the *controller* which plays against the other player, that is the *environment*. This situation is also known as "game against nature" [Mil51].

Typically, a game is expressed by a direct graph, also called *arena*. Based on the characteristics of the considered system, one may choose the most appropriate arena to model the whole system. An example of arena [Maz02] is the triple consisting of an edge relation (the set of players moves) and two disjoint set of vertices, one for the controller and the other one for the environment. In such a game, a player may move only if the current vertex belongs to its associated set of vertices.

This thesis is focused on an another kind of game, defined by an arena consisting in a single set of vertices and two disjoint sets of moves, one for the controller and the other one for the environment. In such a game, a player may move whenever in the current vertex there exists an edge belonging to its set of moves. This kind of arena is introduced in this chapter, by means of *Game Graphs (GGs)*[TA99], that is a finite graph whose discrete transitions are labeled as *controllable* or *uncontrollable*, to model the actions of the controller and the environment, respectively. A game graph allows controller and environment to make choices that determine the next state of the system, by means of locations.

In Section 1.3 it is formally defined the controller synthesis problem for game graphs, with respect to safety and reachability goals. This problem is equivalent to find a *memoryless* winning strategy, i.e. a function that gives the

right next move of the controller, based only on the current state of the system. Algorithms to synthesize such a strategy have been given based on a backward fix-point calculation [Maz02] of the so called *controllable predecessor operator CPre* [MPS95, AMP95]. Given a set of states X (or, equivalently, a set of locations, since in the discrete case the two sets coincide), $CPre(X)$ computes the set the states from which the controller can force the game into X in a single *step*.

The last section of the chapter formally defines the *CPre* operator and then shows these fix-point algorithms, that work on the exploration of the full game graph in input, for safety and reachability objectives.

There exists also newer algorithms that are on-the-fly, in the sense that they return a memoryless winning strategy as soon as one is found, which avoids the fully exploration of the input graph, which can result in significant saving in performance [TA99]. This kind of approach is not shown, because it is not very relevant for the topic of this thesis (it is not possible to reduce the whole set of states of a hybrid systems into a finite graph, and then an on-the-fly approach it is not practicable).

1.1 Game Graph

This thesis is focused on a model of game whose players are free to choose a move, regardless of the current vertex, taken from two disjoint set of moves. The arena of such a game is a finite automaton that allows to explicitly distinguish the actions of the controller from the actions of the environment. This finite automaton is called *Game Graph (GG)*. A game graph is a tuple $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$, consists of the following:

- A finite set *Loc* of *locations*. A *state* is the current location l.
- A finite set Edg_{c} of *controllable transitions*, that describe changes of locations, governed by the *Controller*.
- A finite set Edg_{u} of *uncontrollable transitions*, that describe changes of locations, governed by the *Environment*.
- An initial location $l_0 \in Loc$.

The abbreviations $S = Loc$ is used to indicate the set of all states and $Edg = Edg_{\mathrm{c}} \cup Edg_{\mathrm{u}}$ is the whole set of moves. Notice that, being a game graph a discrete system, it is possible to refer indiscriminately to a location or to a state.

Each transition consists in a pair $\langle l, l' \rangle \in Edg$, where $l \in Loc$ is called the *source location*, and $l' \in Loc$ is called the *target location*.

A game graph $\overline{G} = (Loc, Edg'_{\mathrm{c}}, Edg'_{\mathrm{u}}, l_0)$ is called to be dual to $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$ if $Edg'_{\mathrm{c}} = Edg_{\mathrm{u}}$ and $Edg'_{\mathrm{u}} = Edg_{\mathrm{c}}$. In other words, $\overline{G}$ is a game graph in which the controller and the environment will exchange the roles they had in G.

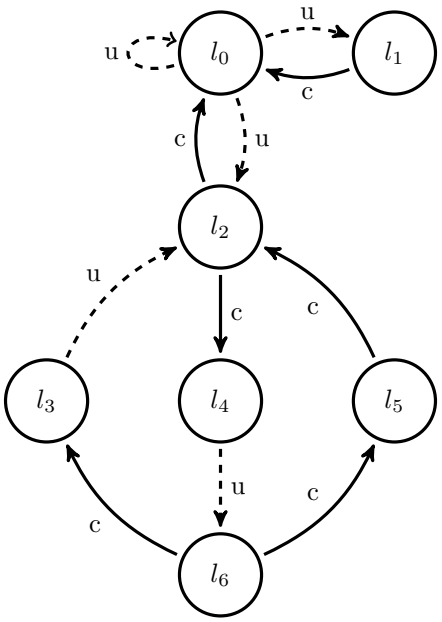


Figure 1.1: A Game Graph as Arena.

Example 1. *Let $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$ be a game graph, whose components are defined as follows:*

- $Loc = \{l_0, l_1, l_2, l_3, l_4, l_5, l_6\}$*, is the set of locations.*
- $Edg_{\mathrm{c}} = \{\langle l_1, l_0 \rangle, \langle l_2, l_0 \rangle, \langle l_2, l_4 \rangle, \langle l_5, l_2 \rangle, \langle l_6, l_3 \rangle, \langle l_6, l_5 \rangle\}$*, is the set of controllable transitions.*
- $Edg_{\mathrm{u}} = \{\langle l_0, l_0 \rangle, \langle l_0, l_1 \rangle, \langle l_0, l_2 \rangle, \langle l_3, l_2 \rangle, \langle l_4, l_6 \rangle\}$*, is the set of uncontrollable transitions.*

Figure 1.1 showes the graphical representation of the game graph G, where controllable transitions are represented by solid arrows labeled by c, and uncontrollable transitions are represented by dashed arrows labeled by u.

1.1.1 Semantics

The behavior of a game graph is based on *discrete* transitions corresponding to the *Edg* component, and produce an instantaneous change in the location.

Runs. Given two states $l, l' \in S$, and a transition $e \in Edg$, we have a *discrete step* $l \xrightarrow{e} l'$ with *source state* l and *target state* l' iff $e = \langle l, l' \rangle$.

Depending on the type of e, we can distinguish the following steps:

- If $e \in Edg_{\mathrm{c}}$, we have a *controllable step* $l \xrightarrow{c} l'$, and l' is called a *controllable successor*.
- If $e \in Edg_{\mathrm{u}}$, we have an *uncontrollable step* $l \xrightarrow{u} l'$, and l' is called an *uncontrollable successor*.

Given the set $X \subseteq S$, $Succ_c(X)$ denotes the set of all controllable successors of states $l \in X$, namely $Succ_c(X) = \{l' \in S \mid l \xrightarrow{c} l', \text{ and } l \in X\}$, and $Succ_u(X)$ denotes the set of all uncontrollable successors of states $l \in X$, namely $Succ_u = \{l' \in S \mid l \xrightarrow{u} l', \text{ and } l \in X\}$. In addition, the set $Succ(X) = Succ_c(X) \cup Succ_u(X)$ is the set of all successors of locations in X.

A *run* is a sequence

$$r = l_0 \xrightarrow{e_0} l_1 \xrightarrow{e_1} l_2 \cdots l_n \cdots \tag{1.1}$$

of discrete steps, such that either the sequence is infinite, or it ends with a discrete step of the type $l_{n-1} \xrightarrow{e_{n-1}} l_n$, with $Succ(\{l_n\}) = \emptyset$.

The *length* of the run r is denoted by $len(r)$, and it is defined as follows:

$$len(r) = \begin{cases} n & \text{if the run r is finite} \\ \infty & \text{otherwise} \end{cases}$$

The set $R(G)$ denotes all possible runs of G, and $States(r)$ is the set of all states visited by r. Formally, $States(r)$ is the set of all states l_i, for all $0 \leq i \leq len(r)$. The set $Inf(r)$ denotes all the locations that occur infinitely often in r. Formally, if r is the run of Equation 1.1 then $Inf(r) = \{l \in States(r) \mid \forall i \geq 0 \; \exists j \geq i \; such \; that \; l_j = l\}$.

1.1.2 Acceptance Conditions

Let $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$ be a game graph. An acceptance condition for G is a set of runs $\Omega \subseteq R(G)$.

Let $T \subseteq Loc$ be a set of states of G and $\mathcal{T} \subseteq 2^{Loc}$ be a power-set of states of G, Table 1.1 lists three set of runs, namely $Reach(G,T)$, $Safety(G,T)$ and

$Muller(G, \mathcal{T})$ that are the set of the accepted runs of G w.r.t. the *reachability*, the *safety* and the *Muller* acceptance conditions, respectively.

$Reach(G,T) = \{r \in R(G) \mid States(r) \cap T \neq \emptyset\}$	r eventually visits T
$Safety(G,T) = \{r \in R(G) \mid States(r) \subseteq T\}$	r always remains in T
$Muller(G,\mathcal{T}) = \{r \in R(G) \mid Inf(r) \in \mathcal{T}\}$	r eventually visits (for ever) all elements of a $T \in \mathcal{T}$

Table 1.1: Acceptance conditions over runs

Example 2 shows when a run is called winner for a player according to a Muller acceptance condition.

Example 2. *Let $G = (Loc, Edg_c, Edg_u, l_0)$ be the arena presented in Figure 1.1, $\mathcal{T} = \{\{l_2, l_4\}, \{l_2, l_3, l_4, l_5, l_6\}\}$ and let $Muller(G, \mathcal{T})$ the acceptance condition.*

A possible infinite run in this game is $r = l_3 l_2 l_4 (l_6 l_5 l_2 l_4 l_6 l_3 l_2 l_4)^\omega$. This run is winning for the controller because $Inf(r) = \{l_2, l_3, l_4, l_5, l_6\} \in \mathcal{T}$. While, the run $r' = (l_2 l_4 l_6 l_3)^\omega$ yields $Inf(r') = \{l_2, l_3, l_4, l_6\} \notin \mathcal{T}$. Hence r' is not winning for the controller.

Strategies. Let G be a game graph as usual. A *strategy* is a partial function $\sigma : Loc^* \to 2^{Edg_P} \setminus \emptyset$, where $P \in \{c, u\}$, such that for all $l_0 \dots l_i l \in Loc^*$ such that $Succ_P(l) \neq \emptyset$, the following conditions hold:

- σ is defined at $l_1 \dots l_n l$, and
- if $e \in \sigma(l_0 \dots l_n l)$, then there exists $l' \in Loc$ such that $l \xrightarrow{e} l'$.

These condition ensures that a strategy can only choose transitions allowed by the game graph.

If $Edg_P = Edg_c$ (resp. $Edg_P = Edg_u$), σ is called a *strategy for the controller* (resp. a *strategy for the environment*).

A prefix of a run $r = l_0 \xrightarrow{e_0} l_1 \xrightarrow{e_1} l_2 \cdots l_n$ is said to be *consistent* with a strategy σ if for every i with $0 \leq i < n$, if $e_i \in Edg_P$ then $e_i \in \sigma(l_0 \dots l_i)$.

Let G be an arbitrary game graph as usual, Ω be an acceptance condition, σ be a strategy for the controller (resp. the environment), and $U \subseteq Loc$ be a set of states. The strategy σ is said to be a *winning strategy* for the controller (resp. environment) on U if all runs which are consistent with σ and start in a location from U are wins for the related player, according to the acceptance condition Ω.

The controller (or the environment) is said to be the *winner* of the game (or *win* the game) on $U \subseteq Loc$ if it has a winning strategy σ on U.

Every game defined as above has at most one winner.

Remarke 1. *[Maz02] For any game graph G, if the controller wins on U_0 and the environment wins on U_1, then $U_0 \cap U_1 = \emptyset$.*

Example 3. *Considering the game of Example 2. When the environment moves from l_0 to l_0 every time the token is located on l_0, then the environment will win every run that visits l_0. This means, in particular, that a strategy for the environment defined by $\sigma_u(yl_0) = \langle l_0, l_0 \rangle$, where $y \in Loc^*$, is a winning strategy for the environment.*

Each run that does not begin in l_0 or l_1, visit the location l_2 at some point. The controller should under no circumstances move the token from l_2 to l_0 because the environment could win as described above. Hence, his only chance is to move the token from l_2 to l_4.

The situation for the controller in location l_6 is a bit more complicated. If it always decides for moving the token to l_3, then the resulting run has the form $r = \ldots (l_2 l_4 l_6 l_3)^\omega$ and is a loss for it. Similarly, it will loose if always moves the token to l_5. But he is able to win if it alternates between l_3 and l_5. To sum this up, consider the function σ_c defined by

$$\sigma_c(r) = \begin{cases} \langle l_2, l_4 \rangle & \text{if } r \in Loc^* l_2 \\ \langle l_6, l_3 \rangle & \text{if } r \in Loc^* l_5 l_2 l_4 l_6 \\ \langle l_6, l_5 \rangle & \text{if } r \in Loc^* l_3 l_2 l_4 l_6 \end{cases}$$

This is a winning strategy for the controller.

1.1.3 Forgetful and Memoryless Strategies

The objective of this section is to introduce some notions that help to explain how complex a winning strategy can be.

As a motivation, consider the game from Example 1 again, where in order to win it is necessary for the controller to alternate between moving the token to l_3 and l_5 when it is on l_6. More precisely, it is necessary not to stick to one of the two locations from some point onwards. This means that the controller has to remember at least one bit, namely whether it moved to l_3 or l_5 when the token was on l_6 the last time. It is clear that it is not necessary to remember more than that. In other words, the controller needs a finite memory to carry out its strategy, that in this case it is called a *forgetful strategy*. The situation is much easier for the environment, that does not need to remember anything; it simply moves to l_0 every time the token is on l_0: the environment has a winning strategy called *memoryless* or *positional*.

In order to formally define the meaning of forgetful and memoryless strategy, consider a game graph G as usual. A strategy σ_p is said to be *finite memory* or *forgetful* if there exists a finite set M, an element $m_I \in M$, and functions

$\alpha : Loc \times M \to M$ and $g : Loc \times M \to 2^{Edg_P}$ such that the following is true. When $r = l_0 l_1 \dots l_{n-1}$ is a prefix of a run in the domain of σ_p and the sequence $m_0 m_1 \dots m_n$ is determined by $m_0 = m_I$ and $m_{i+1} = \alpha(l_i, m_i)$, then $\sigma_p(r) = g(l_n, m_n)$.

Forgetful strategies that do not need memory at all, that is, where one can choose M to be a singleton, are called *memoryless* or *positional*.

Example 4. *In Example 3, the strategy σ_u for the environment is memoryless. To see this, observe that it is possible to choose M to be a singleton, say $M = \{m\}$, and set $g(l_0, m) = \langle l_0, l_0 \rangle$. So, the environment has a memoryless winning strategy. Using the simplified notation, it is possible to write $\sigma_u(l_0) = \langle l_0, l_0 \rangle$.*

The controller needs to store which location between l_3 and l_5 it visited last. This can be done with a memory $M = \{3, 5\}$. choose $m_I = 3$,

$$\alpha(l, m) = \begin{cases} 3 & \text{if } l = l_3 \\ 5 & \text{if } l = l_5 \\ m & \text{otherwise} \end{cases}$$

and

$$g(l, m) = \begin{cases} \langle l_2, l_4 \rangle & \text{if } l = l_2 \\ \langle l_6, l_3 \rangle & \text{if } l = l_6 \text{ and } m = 5 \\ \langle l_6, l_5 \rangle & \text{if } l = l_6 \text{ and } m = 3 \end{cases}$$

Thus, the controller has a forgetful winning strategy.

Example 3 also stated that the controller must not move from l_6 to the same successor every time he visit l_6. So, the controller can't have memoryless winning strategies.

1.2 Safety and Reachability Control Problems

Given a game graph $G = (Loc, Edg_c, Edg_u, l_0)$ and a set os states $T \subseteq Loc$, this thesis is focused in the computation of a winning strategy for the controller with respect to the acceptance conditions $Safety(G, T)$ and $Reach(G, T)$. The former defines the *safety control problem*, and the set T is called the set of "safe states". The latter defines the *reachability control problem*, and the set T is called the set of "target states".

Now, the safety and the reachability control problems for game graph can be formally defined.

Safety control problem. Given a game graph $G = (Loc, Edg_c, Edg_u, l_0)$ and a set of the so-called "safe" states $T \subseteq Loc$, the *safety control problem* consists in checking whether exists a winning strategy σ (for the controller) on l_0 w.r.t. $Safety(G, T)$, i.e. σ is such that, for all runs $r \in Runs(l_0, \sigma)$ it holds $States(r) \subseteq T$.

Reachability control problem. Given a game graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$ and a set of the so-called "target" states $T \subseteq Loc$, the *reachability control problem* consists in checking whether exists a winning strategy σ (for the controller) on l_0 w.r.t. $Reach(G, T)$, i.e. σ is such that, for all runs $r \in Runs(l_0, \sigma)$ it holds $States(r) \cap T \neq \emptyset$.

These problems are stated to be dual, in the sense that given a game graph G the safety control problem w.r.t. the set of safe states T can be solved by considering the reachability control problem for the game graph $\overline{G}$ dual to G w.r.t. the set of target states $\overline{T}$ (see [CES86]). Formally

$$Safety(G, T) = \overline{Reach(G, \overline{T})}.$$

An important result about the reachability control problem, that leads to the computation of a solution, is stated by the following proposition.

Proposition 1. *[Maz02] Given a game graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$ and a set of states $T \subseteq loc$, if there exists a winning strategy σ w.r.t. $Reach(G, T)$ then there exists also a memoryless winning strategy σ' w.r.t. $Reach(G, T)$.*

Hence, by Proposition 1 and by the dual nature of the problems, the safety (resp., reachability) control problems w.r.t. the set of safe (resp., target) states T is equivalent to checking whether exists a memoryless winning strategy w.r.t. $Safety(G, T)$ (resp., $Reach(G, T)$). Also, now the notion of strategy can be refined by restricting the domain of the function from the set loc^* to the set of locations Loc (for the memoryless property), and by restricting the possible moves considering only the set Edg_{c} (because the only interesting strategies are those of the controller). Formally, a *memoryless strategy* is a partial function $\sigma : Loc \to 2^{Edg_{\mathrm{c}}} \setminus \emptyset$, such that for all $l \in Loc$ such that $Succ_c(l) \neq \emptyset$, the following conditions hold:

- σ is defined at l, and
- if $e \in \sigma(l)$, then there exists $l' \in Loc$ such that $l \xrightarrow{e} l'$. This condition ensures that a strategy can only choose controllable transitions allowed by the game graph.

Clearly, the notion of run consistency with a strategy, is slightly different. A run like 1.1 is *consistent* with a memoryless strategy σ, if for all $i \geq 0$ the following condition holds:

- if $e_i \in Edg_{\mathrm{c}}$ then $e_i \in \sigma(l_i)$.

The set $Runs(l, \sigma)$ denote the set of runs starting from the location l and consistent with the strategy σ.

1.3 Solving Safety and Reachability Control Problems

This section shows sound and complete algorithms that compute the winning states and a memoryless winning strategy to solve the safety and the reachability control problem.

In [Maz02] it is showed a constructive proof of the Proposition 1. The proof is constructive in the sense that on a game graph it can be immediately turned into an algorithm which computes a memoryless winning strategy and the set of winning states. In the proof, a memoryless winning strategy and the winning states are defined inductively on the so-called controllable predecessor operator *CPre*. For a set of states A, the controllable predecessor operator $CPre(A)$ returns the set of states from which the controller can ensure that the system remains in A during the next transition. The controllable predecessor operator is the function $CPre : 2^{Loc} \to 2^{Loc}$, defined as follows

$$CPre(A) = \big\{ l \in Loc \,\big|\, (Succ_{\mathrm{c}}(\{l\}) \cap A \neq \emptyset) \wedge (Succ_{\mathrm{u}}(\{l\}) \subseteq A) \big\}.$$

Then, the following proposition holds

Proposition 2. *The answer to the reachability control problem for target set $T \subseteq Loc$ is positive if and only if*

$$l_0 \in \mu W \,.\, T \cup CPre(W). \tag{1.2}$$

In a dual manner, the following proposition also holds

Proposition 3. *The answer to the safety control problem for safe set $T \subseteq Loc$ is positive if and only if*

$$l_0 \in \nu W \,.\, T \cap CPre(W). \tag{1.3}$$

In the following, the algorithm to compute the *CPre* operator and algorithms that implement the above fix-point procedures are showed.

Given a game graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$, a set of states $X \subseteq Loc$, and a set of safe (resp. target) states $T \subseteq Loc$, the Algorithm 1 on parameters G and X computes the set of the controllable predecessors of X, $CPre(X)$. The Algorithm 2 (ref. Algorithm 3) on parameter G and T computes the winning states of the safety (resp. reachability) control problem for G w.r.t. T.

Algorithm 1: $CPre(X)$

Input: Game Graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$, Set of States X
Output: Set of States $CPre$
foreach $l \in Loc$ **do**
  $Succ_c := \emptyset$;
  foreach $\langle l, l' \rangle \in Edg_{\mathrm{c}}$ **do**
    $Succ_c := Succ_c \cup \{l'\}$
  $Succ_u := \emptyset$;
  foreach $\langle l, l' \rangle \in Edg_{\mathrm{u}}$ **do**
    $Succ_u := Succ_u \cup \{l'\}$
  if $\Big(\big(Succ_c \cap W \neq \emptyset\big)$ *and* $\big(Succ_u \subseteq W\big) \Big)$ **then**
    $CPre := CPre \cup \{l\}$;
return $CPre$;

Algorithm 2: $safety(G, T)$

Input: Game Graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$, Set of safe states $T \subseteq Loc$.
Output: Set of States W
$W_0 := T$;
repeat
  $W_{i+1} := T \cap CPre(W_i)$;
  $i := i + 1$;
until $W_{i+1} = W_i$;
return $W := W_i$;

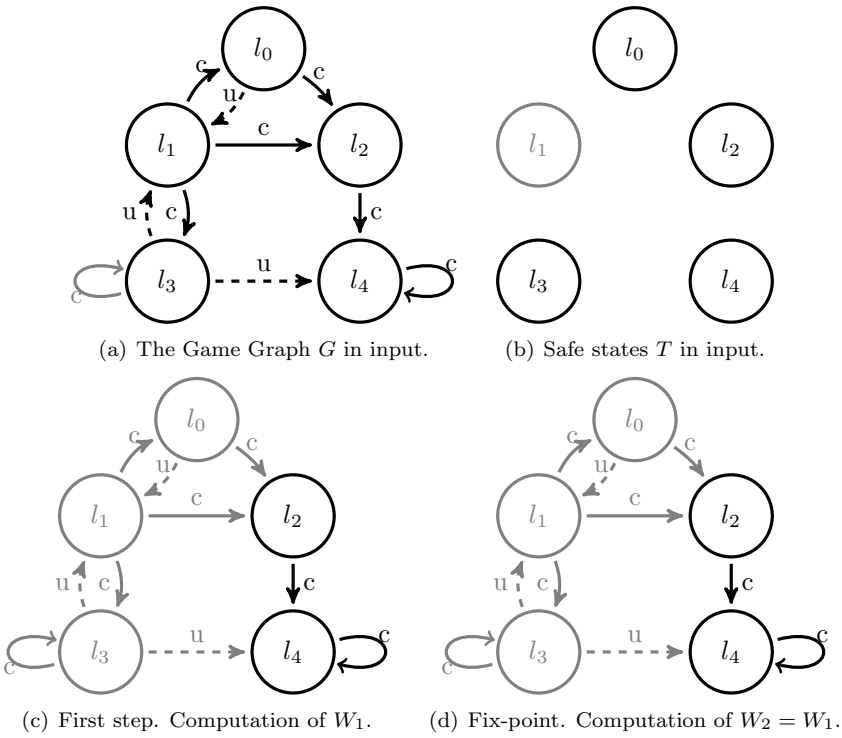
Algorithm 3: $reachability(G, T)$

Input: Game Graph $G = (Loc, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, l_0)$, Set of target states $T \subseteq Loc$.
Output: Set of States W
$W_0 := T$;
repeat
  $W_{i+1} := T \cup CPre(W_i)$;
  $i := i + 1$;
until $W_{i+1} = W_i$;
return $W := W_i$;

Given a game graph G and a set of states T, Example 5 show how Algorithm 2 works in order to solve the safety control problem for G w.r.t. the set of safe states T, while Example 6 show how Algorithm 3 works in order to solve the reachability control problem for G w.r.t. the set of target states T.

Example 5. *Consider the safety control problem for the game graph showed in the Figure 1.2(a) w.r.t. the set of safe states* $T = \{l_0, l_2, l_3, l_4\}$*, showed in Figure 1.2(b)*

Figure 1.2 shows how Algorithm 2 works. Each subgraph of the figure represents a single step of the fix-point procedure.



(a) The Game Graph G in input.

(b) Safe states T in input.

(c) First step. Computation of W_1.

(d) Fix-point. Computation of $W_2 = W_1$.

Figure 1.2: Algorithm for the safety.

The algorithm set $W_0 = T = \{l_0, l_2, l_3, l_4\}$. *In the first step, the fix-point algorithm computes the set* $W_1 = T \cap CPre(\{l_0, l_2, l_3, l_4\})$, *hence the controllable predecessors of* W_0 *must be computed. The location* l_0 *does not belong to CPre, due the uncontrollable transition* $\langle l_0, l_1 \rangle$, *that would lead the system in a location not in* W_0. *From* l_1 *there are not uncontrollable transitions and there exists several controllable transitions, whose target is a location in* W_0. *Hence,* l_1 *belongs to* $CPre(W_0)$. *Also the location* l_2 *belongs to* $CPre(W_0)$*: the only transition whose source location is* l_2, *has* l_4 *as target location and* $l_4 \in W_0$. *From the location* l_3 *the environment may choose the transition* $\langle l_3, l_1 \rangle$, *that would lead the system into the location* $l_1 \notin W_0$. *Hence,* $l_3 \notin CPre(W_0)$. *From the location* l_4, *there is only a self-loop. Since* l_4 *belongs to* W_0, *then* l_4 *belongs also to* $CPre(W_0)$. *Now, the algorithm computes* $W_1 = T \cap CPre(W_0) = \{l_0, l_2, l_3, l_4\} \cap \{l_1, l_2, l_4\} = \{l_2, l_4\}$. *Figure 1.2(c) depicts the situation at the end of the first step.*

In the second step (see Figure 1.2(d)), the algorithm computes $W_2 = T \cap CPre(W_1 = \{l_2, l_4\})$. *The set* $CPre(W_1)$, *besides the location belonging to* W_1, *contains also the location* l_1, *due the controllable transition* $\langle l_1, l_2 \rangle$ *that would lead the system into a location in* W_1. *Notice that there are not uncontrol-*

lable transitions whose source location is l_1. Hence, $W_2 = T \cap CPre(W_1) = \{l_0, l_2, l_3, l_4\} \cap \{l_1, l_2, l_4\} = \{l_2, l_4\}$. Since the set of states W_2 is the same of W_1 ($W_2 = W_1$), the fix-point is reached and the locations in W_2 are winning.

Example 6. *Consider the reachability control problem for the game graph of Example 5, showed again in Figure 1.3(a) w.r.t. the set of target states $T = \{l_4\}$.*

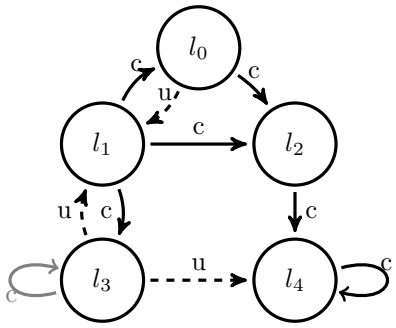
Figure 1.3 shows how Algorithm 3 works, in order to compute the winning states. Each subgraph of the figure represents a single step of the fix-point procedure.

In the first step the algorithm computes the set $W_1 = T \cup CPre(W_0 = T = \{l_4\}$. The only locations from where the controller can enforce the system to reach the location l_4 in one step are l_2 (by the controllable transition $\langle l_2, l_4 \rangle$) and l_4 by the (controllable) self-loop $\langle l_2, l_4 \rangle$. Hence, $W_1 = \{l_4\} \cup \{l_2, l_4\} = \{l_2, l_4\}$. Figure 1.3(c) shows the result of the first step.

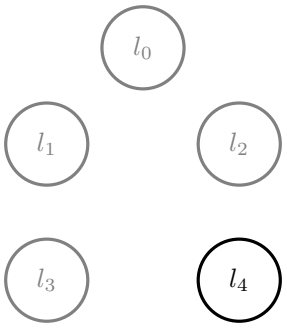
In the second step the algorithm computes the set $W_2 = T \cup CPre(W_1 = \{l_2, l_4\}$. Now, l_1 belongs to $CPre(W_1)$, because the controller may take the transition $\langle l_1, l_2 \rangle$ to enforce the system to lead the location $l_2 \in W_1$, while l_0 (resp. l_3) does not belongs to $CPreW_1$, due the uncontrollable transition $\langle l_0, l_1 \rangle$ (resp. $\langle l_3, l_1 \rangle$) that would lead the system in the location $l_1 \notin W_1$. Hence, at the end of the second step, the algorithm computes $W_2 = T \cup CPre(\{l_2, l_4\}) = \{l_4\} \cup \{l_1, l_2, l_4\} = \{l_1, l_2, l_4\}$. Figure 1.3(d) shows the result of the second step.

In the next step, also the location l_0 belongs to $CPre(W_2)$, because the controller may take the transition $\langle l_0, l_2 \rangle$ in order to reach the location $l_2 \in W_2$, while the only move ($\langle l_0, l_1 \rangle$) for the environment also lead the system in a location belonging to W_2 ($l_1 \in W_2$). Hence, $W_3 = T \cup CPre(\{l_1, l_2, l_4\}) = \{l_4\} \cup \{l_0, l_1, l_2, l_4\} = \{l_0, l_1, l_2, l_4\}$. Figure 1.3(e) shows the result of this step.

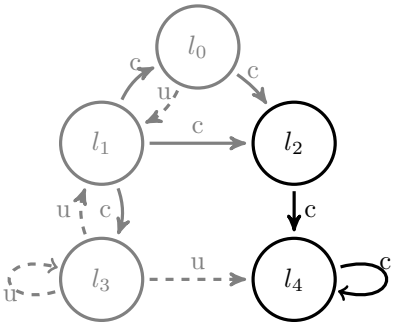
Now, the only location that could be added to W_4, is l_3. But there are not controllable transitions whose source location is l_3 and whose target location is one belonging to W_3. Hence, $W_4 = W_3$ and the fix point is reached (see Figure 1.3(f)).



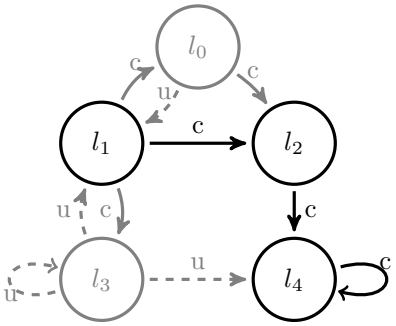
(a) The Game Graph G in input.



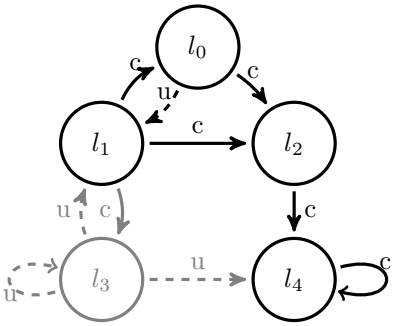
(b) Sef of target states T in input.



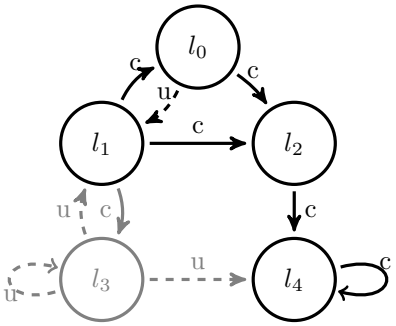
(c) First Step (computation of W_1).



(d) Second Step (computation of W_2).



(e) Third Step (computation of W_3).



(f) Fix-point. Computation of $W_4 = W_3$.

Figure 1.3: How fix-point algorithm for reachability works.

Chapter 2

Control Problems for Timed Games

While in Chapter 1 the considered dynamical systems are discrete, in the rest of the thesis the considered systems are only continuous. In this chapter the results showed in Chapter 1 are extended to one of the first kind of continuous systems studied in computer science, i.e. real-time systems. Such a system is typically modeled by the formalism of *Timed Automata (TAs)*[AD94]. A timed automaton is a finite automaton augmented with a finite set of variables, or *clocks*, over a continuous domain. The behavior of a TA is described by the time elapse in a location (*timed step*), and by instantaneous discrete transitions (*discrete step*). A clock can be reset to zero simultaneously with any transition. At any instant, the reading of a clock equals the time elapsed since the last time it was reset. Each transition is associated with a *clock constraint* (or *guard*), and it is required that the transition can be taken only if the current values of the clocks satisfy this constraint.

This chapter introduces a particular kind of timed automaton, in order to model explicitly the disjoint actions of the controller and of the environment. This is done by the formalism of *Timed Games (TGs)* [MPS95, ACD93, HNSY92], that is a timed automaton whose set of transitions is partitioned in two disjoint sets: the set of the *controllable transitions*, governed by the controller, and the set of *uncontrollable transitions*, governed by the environment.

Once introduced the formalism of timed games, the related notion of strategy is redefined and the controller synthesis problem for timed games is formally introduced. The main idea behind the resolution of this problem is to find a way that allows to view the continuous behavior of a TG as discrete, in order to use the same algorithms seen in Chapter 1. For example, Maler and others [MPS95] proposed an appropriate version of the (timed) controllable predecessor

operator *CPre*, that takes into account the real-time properties of the systems, based on a special kind of step, called "joint step" (a timed step followed by a discrete step). The use of joints steps as basic game steps allows to abstract away from the infinitely many time progress occurring between two consecutive discrete steps, and hence a discretization is obtained.

In the literature, the most used discretization is given by another technique, based on appropriate equivalence relations that allow to reduce the whole (infinite) set of states of a timed games into a finite number of equivalence classes. Examples of such relations are the *region-graph* equivalence [AD94] and the *tabisimulation* [TA99]. These relations abstract away from the exact amount of time elapsed and they are therefore referred to as *time-abstracting* equivalences. By using one of the above relations, a finite automaton (e.g. a game graph) equivalent to the considered timed game, can be obtained. There exist several algorithms [TA99, BFH$^+$94, LY92] to this aim. Tools like UPPAAL [BLP$^+$96] and KRONOS [DOTY96] (for verification), or like UPPAAL-TIGA [BCD$^+$05, JRLD07] (for synthesis), using these algorithms and then, once obtained the corresponding game graph, the algorithms showed in Chapter 1 are used.

This thesis is focused on the first kind of approach (the timed version of *CPre*). In fact algorithms based on time-abstracting equivalence have no practical relevance for the purpose of this work: they cannot be extended in order to correctly handle control problems for hybrid games, because there does not exists a finite abstraction for the latter. Hence, the chapter ends by showing only the controllable predecessor operator for the real-time games (see [AD94, TA99]) for a complete picture of time-abstracting equivalence approachs).

2.1 Timed Games (TGs)

A *Timed Game (TG)* [MPS95, ACD93, HNSY92] is a game graph augmented with a finite set $X = \{x_1, \ldots, x_n\}$ of (real-valued) *clocks*. Let $\mathbb{R}^{\geq 0}$ denote the set of nonnegative real numbers, and let $\mathbb{Q}^{\geq 0}$ denote the set of nonnegative rational numbers.

For a set X of clocks, the set $\Phi(X)$ of *clock constraints* g is defined by the grammar

$$g := x \leq c \mid x \geq c \mid x < c \mid x > c \mid g \wedge g$$

where $x \in X$ and $c \in \mathbb{Q}^{\geq 0}$.

A *valuation* is a function $v : X \to \mathbb{R}^{\geq 0}$. For every $\delta \in \mathbb{R}^{\geq 0}$, $v' = v + \delta$ is a valuation such that $v'(x) = v(x) + \delta$, for all $x \in X$. Given $\mu \subseteq X$, $v[\mu := 0]$ is the valuation v', such that

- $v'(x) = 0$, for each $x \in \mu$, and
- $v'(x) = v(x)$, otherwise.

Let $Val(X)$ denote the set of valuations over X, and let $\Pi g \in 2^{Val(X)}$ denote the set of valuation that satisfy the guard g, i.e. $\Pi g = \{v \in Val(X) \mid v\ satisfies\ g\}$.

A *Timed Game* $A = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Inv, Init)$ consists of the following:

- A finite set Loc of *locations*.
- A finite set $X = \{x_1, \ldots, x_n\}$ of clocks. A *state* is a pair $\langle l, v\rangle$ of a location l and a valuation $v \in Val(X)$.
- Two sets Edg_{c} and Edg_{u} of *controllable* and *uncontrollable reset transitions*, respectively. They describe instantaneous changes of locations, in the course of which variables may be reset. Each transition $(l, g, \mu, l') \in Edg_{\mathrm{c}} \cup Edg_{\mathrm{u}}$ consists of a *source location* l, a *target location* l', a *guard* $g \in \Phi(X)$ that describes the valuations for which the transition is enabled and a *jump relation* $\mu \subseteq X$, that specifies the clocks to reset when the transition occurs.
- A mapping $Inv : Loc \to \Phi(X)$, called the *invariant*.
- A mapping $Init : Loc \to \Phi(X)$, contained in the invariant, which allows the definition of the *initial states* from which all behaviors of the automaton originate.

The abbreviation $S = Loc \times Val(X)$ is used to denotes the set of all states and $Edg = Edg_{\mathrm{c}} \cup Edg_{\mathrm{u}}$ is used for the set of all transitions. Moreover, $InvS = \bigcup_{l \in Loc}\{l\} \times \Pi Inv(l)$ is the set of all admissible valuations and $InitS = \bigcup_{l \in Loc}\{l\} \times \Pi Init(l)$ is the set of all initial states. Notice that $InvS$ and $InitS$ are sets of states.

2.1.1 Semantics

The behavior of a timed game is based on two types of steps:

1. *discrete* or *reset* steps correspond to the Edg component, and produce an instantaneous change in both the location and the variable valuation;
2. *timed* steps describe the time progress of the clocks in X.

Given a state $s = \langle l, v\rangle$, we set $loc(s) = l$ and $val(s) = v$. Additionally, for a valuation $v \in Val(X)$, the *span* of v in l, denoted by $span(v, l)$, is the set of all values $\delta \geq 0$ such that $\langle l, v + \delta'\rangle \in InvS$, for all $0 \leq \delta' \leq \delta$. Intuitively, δ is the span of v iff v never leaves the invariant in the first δ time units. If all non-negative reals belong to $span(v, l)$, we write $\infty \in span(v, l)$.

Runs. Given two states s, s', and a transition $e \in Edg$, there is a *discrete step* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff *(i)* $s, s' \in Invs$, *(ii)* $e = (loc(s), g, \mu, loc(s'))$, *(iii)* $val(s) \in g$, i.e. $val(s)$ satisfies the guard g, and *(iv)* $val(s') = val(s)[\mu := 0]$).

There is a *timed step* $s \xrightarrow{\delta} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ iff *(i)* $s + \delta' \in InvS$, for each $0 \leq \delta' \leq \delta$, and *(ii)* $s' = \langle loc(s), val(s) + \delta\rangle$.

For technical convenience, timed transitions of duration zero are admitted [1].

A special timed step is denoted by $s \xrightarrow{\infty}$ and represents the case when the system remains in the location $loc(s)$ forever. This is only allowed if $val(s)+\delta \in InvS$, for all $\delta \geq 0$.

Finally, a *joint step* $s \xrightarrow{\delta,e} s'$ represents the timed step $s \xrightarrow{\delta} \langle loc(s), val(s)+\delta\rangle$ followed by the discrete step $\langle loc(s), val(s) + \delta\rangle \xrightarrow{e} s'$.

A *run* is a sequence

$$r = s_0 \xrightarrow{\delta_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1} s'_1 \xrightarrow{e_1} s_2 \dots s_n \dots \tag{2.1}$$

of alternating timed and discrete steps, such that either the sequence is infinite, or it ends with a timed transition of the type $s_n \xrightarrow{\infty}$.

The *length* of the run r is denoted by $len(r)$, and it is defined as follows:

$$len(r) = \begin{cases} n & \text{if the run r is finite} \\ \infty & \text{otherwise} \end{cases}$$

The set $States(r)$ denotes the set of all states visited by r. Formally, $States(r)$ is the smallest set containing all states $\langle loc(s_i), f_i(\delta)\rangle$, for all $0 \leq i \leq len(r)$ and all $0 \leq \delta \leq \delta_i$.

Notice that the states from which discrete steps start (states s'_i in (2.1)) appear in $States(r)$. Moreover, if r contains a sequence of one or more zero-time timed steps, all intervening states appear in $States(r)$.

Example 7. *Let $A = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Inv, Init)$ be a timed game, whose components are defined as follows:*

[1] Timed transitions of duration zero can be disabled by adding a clock variable t to the automaton and requesting that each discrete transition happens when $t > 0$ and resets t to 0 when taken.

- $Loc = \{l_1, l_2, l_3, l_4, l_5, l_6\}$.
- $X = \{x\}$.
- $Edg_{\mathrm{c}} = \big\{(l_1, x \geq 1, \emptyset, l_2), (l_2, x \geq 2, \emptyset, l_6), (l_4, x \leq 1, \emptyset, l_2)\big\}$.
- $Edg_{\mathrm{u}} = \big\{(l_1, x < 1, \{x\}, l_3), (l_1, x > 1, \emptyset, l_5), (l_2, x < 1, \emptyset, l_3)\big\}$.
- $Inv(l) = true$, *for all* $l \in Loc$.
- $Init(l_1) = true$ *and* $Init(l) = false$, *for all* $l \neq l_1$.

Figure 2.1 shows the graphical representation of the timed game A. Notice that, controllable transitions are represented by solid arrows, while uncontrollable transitions are represented by dashed arrows. The arrows are labeled with the guards and the set of the clock to reset of the corresponding transition.

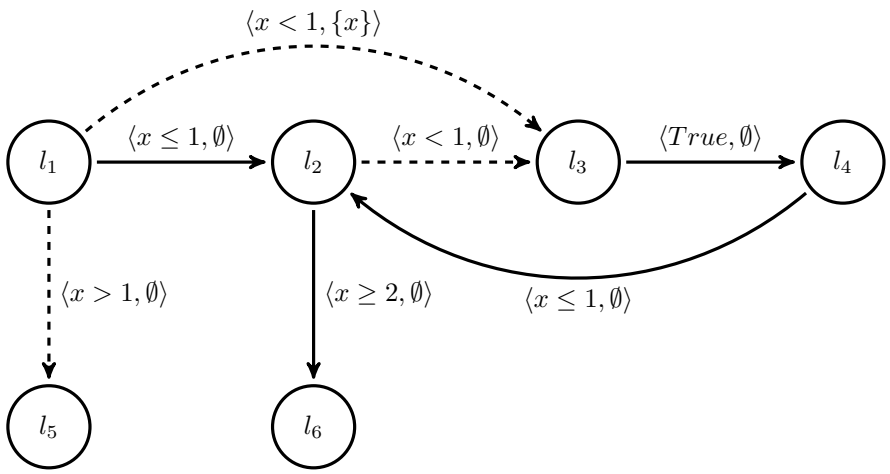


Figure 2.1: A Timed Game (adapted from [CDF+05]).

Zenoness and well-formedness. A well-known problem that affect continuous systems, and hence also real-time, is that definitions like the above admit runs that take infinitely many discrete steps in a finite amount of time, even if such behaviors are physically meaningless. Such runs are called *Zeno* runs (from the "paradoxes of Motion" proposed by the greek philosopher Zeno).

On the other hand, a run of the form (2.1) is called *non-Zeno* if, for all $\delta \geq 0$, there exists $i \geq 0$ such that $\sum_{j=0}^{i} \delta_j > \delta$. In other words, a run is non-Zeno if time diverges along the run.

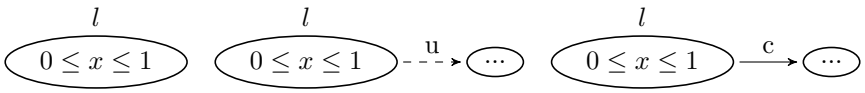
In this thesis it is assumed that the timed game under consideration does not generate Zeno runs. This is easily achieved by using an extra clock t to

ensure that the delay between any two discrete steps is bounded from below by a constant c (all transitions can only be taken when $t \geq c$ and then they reset t to zero).

Moreover, it is assumed that the timed game under consideration is *non-blocking*, i.e., whenever the automaton is about to leave the invariant there must be an uncontrollable transition enabled. Formally, for all states s in the invariant, if the valuation $v = val(s)$ eventually leave the invariant, there exists a time $\delta \in span(v, loc(s))$ such that there is an uncontrollable transition enabled in $\langle loc(s), v+\delta \rangle$, i.e., there exist $s' \in InvS$ and $e \in Edg_{\mathrm{u}}$ such that $\langle loc(s), v+\delta \rangle \xrightarrow{e} s'$.

If a timed game is non-Zeno and non-blocking, is said to be *well-formed*. In the following, all timed games are assumed to be well-formed.

Example 8. *Consider the timed game in Figure 2.2. The fragment in Figure 2.2(a) is blocking, because eventually the clock valuation leaves the invariant and there is not uncontrollable transitions to ensure the progress of the game. The fragment in Figure 2.2(b) is non-blocking, because the system cannot remain in l forever, but an uncontrollable transition leading outside is always enabled. Finally, the fragment in Figure 2.2(c) is blocking, because the system cannot remain in l forever, and no uncontrollable transition is enabled.*



(a) Blocking. (b) Non-blocking. (c) Blocking.

Figure 2.2: Three TG fragments. Locations contain the invariant. Solid (resp., dashed) edges represent controllable (resp., uncontrollable) transitions. Guards are `true`.

Strategies. In order to take into account the density of the time domain, the notion of non-deterministic and memoryless strategy shown in Chapter 1 needs to be redefined.

In the real-time context, the controller may choose also to do nothing. Hence, the strategy can associate the null action to any states, denoted by the symbol $\perp$.

A (timed) *strategy* is a function $\sigma : S \to 2^{Edg_{\mathrm{c}} \cup \{\perp\}} \setminus \emptyset$ such that:

(a) for all $s \in S$, if $e \in \sigma(s) \cap Edg_{\mathrm{c}}$, then there exists $s' \in S$ such that $s \xrightarrow{e} s'$;

(b) if $\perp \in \sigma(s)$, then there exists $\delta > 0$ such that for all $0 < \delta' < \delta$ it holds $val(s) + \delta' \notin Inv(loc(s))$ or $\perp \in \sigma(\langle loc(s), val(s) + \delta' \rangle)$.

In the rest of the chapter, when one says strategy, refers to a timed strategy.

Condition (a) ensures that a strategy can only choose transitions allowed by the automaton. Condition (b) requires that if a strategy chooses the null action, then it must continue to do so for a positive amount of time that remains in the invariant.

A run like (2.1) is *consistent* with a strategy σ if for all $0 \leq i < len(r)$ the following conditions hold:

- for all $\delta \geq 0$ such that $\sum_{j=0}^{i-1} \delta_j \leq \delta < \sum_{j=0}^{i} \delta_j$, we have $\bot \in \sigma(\langle loc(s_i), val(s_i) + \delta - \sum_{j=0}^{i-1} \delta_j \rangle)$;
- if $e_i \in Edg_{\mathrm{c}}$ then $e_i \in \sigma(s'_i)$.

The set $Runs(s, \sigma)$ contains all runs starting from the state s and consistent with the strategy σ.

2.2 Safety and Reachability Control Problems for Timed Games

In the following, the definitions of safety and reachability control problems given in Chapter 1, are extended in order to take into account the real-time properties of the systems considered in this chapter.

Safety control problem. Given a timed game $A = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Inv, Init)$ and a set of safe states $T \subseteq S$, the *safety control problem* for A consists in checking whether exists a winning strategy σ (for the controller) such that, for all initial states $s \in InitS$ and all runs $r \in Runs(l, \sigma)$, it holds $States(r) \subseteq T$.

Reachability control problem. Given a timed game $A = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Inv, Init)$ and a set of target states $T \subseteq S$, the *reachability control problem* for A consists in checking whether exists a winning strategy σ (for the controller) on such that, for all initial states $s \in InitS$ and all runs $r \in Runs(l, \sigma)$ it holds $States(r) \cap T \neq \emptyset$.

Example 9. *Consider the reachability control problem for the timed game of Example 7, w.r.t. the set of target states* $T = \{(l_6, v) \mid v \in InvS\}$.

A winning strategy would consist in taking the transition $(l_1, x \leq 1, \emptyset, l_2)$ *immediately in all states* (l_1, val) *with* $val \leq 1$*; taking* $(l_2, x \geq 2, \emptyset, l_6)$ *immediately in all states* (l_2, val) *with* $val \geq 2$*; taking* $(l_3, True, \emptyset, l_4)$ *immediately in all states* (l_3, val) *and delaying in all states* (l_4, val) *with* $val < 1$ *until the value of* x *is 1 at which point the transition* $(l_4, x \leq 1, \emptyset, l_2)$ *is taken.*

2.3 Solving Safety and Reachability Control Problems

The fix-point algorithm showed in Chapter 1, can be applied also in the real-time context in order to solve control problems, using joint steps as the basic game steps. In particular, Algorithm 2 can be used to compute the winning states for a safety control problem, and Algorithm 3 can be used to compute the winning states for a safety control problem.

Clearly, the difference consists in the computation of the appropriate version of controllable predecessor operator *CPre* for real-time systems. In the following the *CPre* operator is formally defined.

Controllable predecessor operator. For a set of states A, the operator $CPre(A)$ returns the set of states from which the controller can ensure that the system remains in A during the next joint transition. This happens if for all delays δ, one of two situations occurs:

- either the systems stays in A up to time δ, while all uncontrollable transitions enabled up to time δ (included) also lead to A, or
- there exists a time $\delta' < \delta$, such that the system stays in A up to time δ', all uncontrollable transitions enabled up to time δ' (included) also lead to A, and the controller can issue a transition at time δ' leading to A.

To improve readability, before formally defining the controllable predecessor operator, some preliminary operators are defined.

For a set of states A and $x \in \{c, u\}$, the *predecessors* $Pre_x(A)$ is defined as:

$$Pre_x(A) = \{s \in S \mid s \xrightarrow{e} s', \text{ with } s' \in A \text{ and } e \in Edg_x\},$$

and denotes the set of states where some discrete transition belonging to Edg_x is enabled and leads to A.

For a set of states A and a time delay $\delta \geq 0$ (including infinity), the set of states from where waiting δ time units keeps the system in A, and any uncontrollable transition taken meanwhile also leads into A, is denoted by $While(A, \delta)$. Formally,

$$While(A, \delta) = \Big\{ s \in S \Big| \forall 0 \leq \delta' \leq \delta : \langle loc(s), val(s) + \delta' \rangle \in A \setminus Pre_u(\overline{A}) \Big\}.$$

Now, it is possible to formally define the *CPre* operator.

$$CPre(A) = \Big\{ s \in S \;\Big|\; \forall \delta \in span(val(s), loc(s)) : s \in While(A, delta) \\ \text{or } \exists 0 \leq \delta' < \delta : s \in While(A, \delta') \text{ and } \langle loc(s), val(s) + \delta' \rangle \in Pre_c(A) \Big\}.$$

This is the correct version of the operator *CPre* in order to handle the real-time properties of the considered systems.

Algorithms to compute the *CPre* operator are less efficient than the approach based on the time-abstracting equivalence, that usually using on-the-fly technique for the construction of the region graphs. For reasons already explained in the introduction of this chapter, these algorithms are not relevant for the purpose of this thesis and therefore will not be explained. Notice that, being a timed game a proper subclass of hybrid game (see Chapter 3), the related safety and reachability control problem can be solve by using the algorithms for hybrid games that will be introduced in Chapter 4 and in Chapter 5, respectively.

Chapter 3

Hybrid Games (LHGs) and Control Problems

The real-time systems introduced in Chapter 2 are an example of relatively simple continuous systems. Simple in the sense that the trajectory of the systems always faithfully follows the trend of the time, i.e. each continuous variable x has a constant rate of growth, equal to 1. This can be easily expressed by saying that in a timed game the first derivative of each continuous variable is equal to 1.

This chapter is focused on hybrid systems, a particular kind of continuous systems, whose trajectories may be expressed by laws more complicated than the real-time case. Also in hybrid systems, state variables are partitioned into discrete and continuous ones. Given the ability to define a wide variety of evolution of the continuous variables, these can be used to represent physical quantities like temperature, speed, etc. As the case of timed, discrete variables may be used to represent *control modes*, i.e., states of the controller.

The formalism of *Hybrid Automata (HAs)* [Hen96] is the most common syntactic variety of hybrid system: a finite set of locations, similar to the states of a finite automaton, represents the value of the discrete variables. The current location, together with the current value of the (continuous) variables, form the instantaneous description of the system. Change of location happens via discrete transitions, and the evolution of the variables is governed by differential equations attached to each location.

Depending on the class of differential equations attached to the locations, several different classes of hybrid automata can be identified.

The most studied problem for hybrid systems is *reachability*: computing the set of states that are reachable from the initial states, in any amount of time. For a general class of hybrid automata, the reachability problem was proved

undecidable in [HKPV95], indicating that no exact discrete abstraction exists. Considering some restriction on the allowed differential equations (see Section 3), semi-decidable or decidable fragments of hybrid automata [WT97, HKPV95] can be found.

In a similar way as already seen for continuous and discrete systems, this thesis is focused on hybrid automata whose discrete transitions are partitioned into controllable and uncontrollable ones. This is the formalism of *Hybrid Games (HGs)*, in which the controller governs the controllable transitions, while the environment governs not only the uncontrollable transitions, but also may choose a trajectory, according to the differential equations attached to the current location. The resulting control problem is to compute a strategy for the controller to satisfy a given goal, regardless of the evolution of the continuous variables and of the uncontrollable transitions.

In this chapter the formalism of Hybrid Games, its related subclasses, and the control problem (w.r.t. safety and reachability goals) are formally defined.

3.1 Hybrid Games (HGs)

Given an ordered set $X = \{x_1, \ldots, x_n\}$ of variables, a *valuation* is a function $v : X \to \mathbb{R}$. Let $Val(X)$ denote the set of valuations over X, $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ denote the set of dotted variables, used to represent the first derivatives, and X' to denote the set $\{x'_1, \ldots, x'_n\}$ of primed variables, used to represent the new values of variables after a transition. Let $Val(X)$ denote the set of valuations over X.

The set of dotted variables $\dot{X} = \{\dot{x}_1, \ldots, \dot{x}_n\}$ is used to represent the first derivatives, and the set of the primed variables $X' = \{x'_1, \ldots, x'_n\}$ of primed variables, is used to represent the new values of variables after a transition. Arithmetic operations on valuations are defined in the straightforward way. An *activity* over X is a differentiable function $f : \mathbb{R}^{\geq 0} \to Val(X)$.

Let $Acts(X)$ denote the set of activities over X. The *derivative* $\dot{f}$ of an activity f is defined in the standard way and it is an activity over $\dot{X}$.

An *Hybrid Games* $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$ consists of the following:

- A finite set *Loc* of *locations*.
- A finite set $X = \{x_1, \ldots, x_n\}$ of continuous, real-valued *variables*. A *state* is a pair $\langle l, v \rangle$ of a location l and a valuation $v \in Val(X)$.
- Two sets Edg_{c} and Edg_{u} of *controllable* and *uncontrollable transitions*, re-

spectively. They describe instantaneous changes of locations, in the course of which variables may change their value. Each transition $(l, \mu, l') \in Edg_{\mathrm{c}} \cup Edg_{\mathrm{u}}$ consists of a *source location* l, a *target location* l', and a *jump relation* $\mu \in 2^{Val(X \cup \dot{X})}$, that specifies how the variables may change their value during the transition. The projection of μ on X describes the valuations for which the transition is enabled; this is often referred to as a *guard*.

- A mapping $Flow : Loc \to 2^{Val(X \cup \dot{X})}$ attributes to each location a set of valuations over the variables and their derivatives, which determines how variables can change over time.
- A mapping $Inv : Loc \to 2^{Val(X)}$ called the *invariant*. All behaviors is constrained to the invariant at all times.
- A mapping $Init : Loc \to 2^{Val(X)}$, contained in the invariant, defining the *initial states* of the hybrid game.

In the rest of the thesis the abbreviation $S = Loc \times Val(X)$ is used for the set of states and $Edg = Edg_{\mathrm{c}} \cup Edg_{\mathrm{u}}$ for the set of all transitions. Moreover, $InvS = \bigcup_{l \in Loc} \{l\} \times Inv(l)$ is the set of all invariants, and $InitS = \bigcup_{l \in Loc} \{l\} \times Init(l)$ is the set of all the initial states. Notice that $InvS$ and $InitS$ are sets of states. Given a set of states A and a location l, the set $A\lfloor_l$ denotes the projection of A on l, i.e. $\{v \in Val(X) \mid \langle l, v \rangle \in A\}$.

This is a very general definition of hybrid games, where valuations associated to a single location by *Flow*, *Inv* and *Init* mappings are not specified. Notice that the mapping *Flow* is given by a set of valuations over the variables and their derivatives, i.e. is given by a means of a general differential equation. When these mappings are carefully specified, i.e. by constraining the associated valuations into a specified subset of $Val(X)$, a number of proper subclasses of hybrid games can be identified. In order to formally define these subclasses, additional notations are given.

A *convex polyhedron* is a subset of $\mathbb{R}^n$ that is the intersection of a finite number of strict and non-strict affine half-spaces. A *polyhedron* is a subset of $\mathbb{R}^n$ that is the union of a finite number of convex polyhedra. For a general (i.e., not necessarily convex) polyhedron $G \subseteq \mathbb{R}^n$, the polyhedron $cl(G)$ denotes its topological closure, and $[\![G]\!] \subseteq 2^{\mathbb{R}^n}$ denotes its representation as a finite set of convex polyhedra. Notice that there is an obvious bijection between $Val(X)$ and $\mathbb{R}^n$, allowing to extend the notion of (convex) polyhedron to sets of valuations. In addition, $CPoly(X)$ (resp., $Poly(X)$) denotes the set of convex polyhedra (resp., polyhedra) on X.

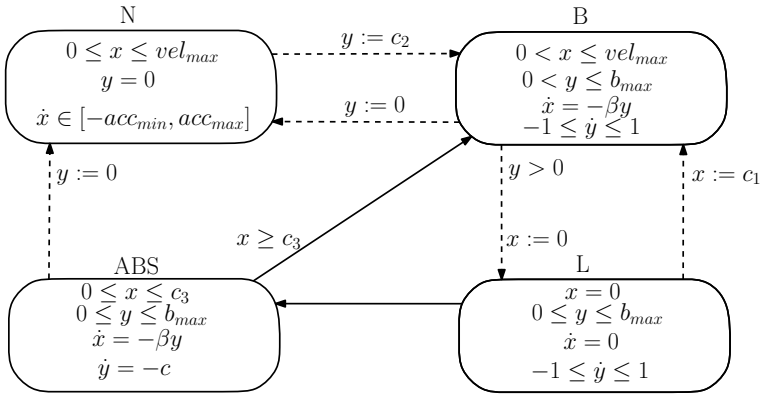


Figure 3.1: ABS modeled as AHG.

Now several subclasses of HGs can be identified. The subclass of *Affine Hybrid Games (AHGs)* is a HG, whose mapping are defined as following:

- The jump relation of each transition is defined on polyhedra on $X \cup X'$, i.e. $\mu \in Poly(X \cup X')$. .
- $Flow : Loc \to CPoly(X \cup \dot{X})$, that allows to model dynamics of the form $\dot{x} = Ax + b$, with the elements of A and b given by intervals[1].
- $Inv : Loc \to Poly(X)$.
- $Init : Loc \to Poly(X)$.

Example 10. *Consider a simplified version of the ABS device of Figure 2 shown in the introduction. The driver (viewed as environment) may choose to brake at any time, with magnitude (modeled by the continuous variable y) between 0 and the constant value $b_{max} > 0$. The angular speed of the wheels is considered to be the same for all wheels, and can be modeled by the continuous variable x. Clearly, the speed is affected by the braking power. The normal running of the vehicle, i.e. when no brake is performed by the driver, is modeled by the location N, while the braking state is modeled by the location B. The dynamics associated to B is expressed by $\dot{x} = -\beta y$ (with $\beta > 0$) to model that the decrease of wheels speed is proportionally to the intensity of the braking. In addition, the differential equation $\dot{y} \in [-1, 1]$ represents the ability to change the braking power. When the driver performs a braking (location B), a wheel lock could occur. This situation is modeled by the uncontrollable transition that leads the system into the location L (representing the locking state), whose update function*

[1] This is the semantics adopted by Frehse ([Fre05]).

μ is such that it assigns the zero value to the speed of the wheels ($x := 0$). In this state, the controller may act on the brake in order to restore the normal running of the vehicle: this is modeled by the controllable transition that leads the system into the location ABS, that represents the state in which the ABS device begins its work, i.e. the reduction of the braking power. This reduction is modeled by the differential equation $\dot{y} = -c$, with $c > 0$. Falling off the braking force, the vehicle eventually returns in the normal running state, i.e. eventually $x > 0$. The ABS device ends its work if (i) the speed reaches a given threshold ($x \geq c_3$, with $c_3 < 0$) and the driver is still using the brake or (ii) the driver ends the use of the brake. In the former, the system returns into the braking state (location B) while in the latter, the system returns into the normal running state. The whole system described in this example can be modeled as the affine hybrid game depicted in Figure 3.1.

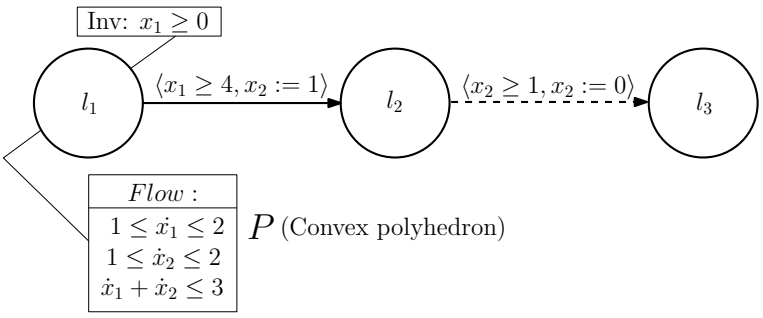


Figure 3.2: Example of LHG.

The formalism of *Linear Hybrid Games (LHGs)* is a subclass of AHGs, where the only allowed differential equations are in fact differential inclusions of the type $\dot{\mathbf{x}} \in P$, where $\dot{\mathbf{x}}$ is the vector of the first derivatives of all variables and P is a convex polyhedron. Formally

$$Flow : Loc \rightarrow CPoly(X).$$

The hybrid game depicted in Figure 3.2 belongs to the subclass of LHGs.

A *Rectangular Hybrid Game (RHG)* is a subclass of a LHG, where the first derivative of each continuous variable x is bounded by constants from below and above, that is $\dot{x} \in [a, b]$. Notice that a timed game is a special subclass of a RHG where the first derivative of each continuous variable x are equal to 1, i.e. $\dot{x} \in [1, 1]$.

Notice that affine, linear and rectangular hybrid games are non-deterministic, allowing for infinitely many solutions. Hence, the environment may choose non-deterministic trajectories. In the opposite, trajectories of a timed games can be

Class	Differential equations	Non-determinism
Affine HGs	$\dot{\mathbf{x}} = Ax + b$	Yes
$\bigcup\!\!\!\!\bigcup$	$\bigcup\!\!\!\!\bigcup$	
Linear HGs	$\dot{\mathbf{x}} \in P \subseteq \mathbb{R}^n$	Yes
$\bigcup\!\!\!\!\bigcup$	$\bigcup\!\!\!\!\bigcup$	
Rectangular HGs	$\dot{x} \in [a, b],\ \forall \dot{x} \in \dot{X}$	Yes
$\bigcup\!\!\!\!\bigcup$	$\bigcup\!\!\!\!\bigcup$	
Timed Games	$\dot{x} = 1,\ \forall \dot{x} \in \dot{X}$	No

Table 3.1: Hybrid Games hierarchy

only deterministic. See Table 3.1 to have the general picture of classes of hybrid games and their relationship.

Figure 3.3 shows the dynamics of the linear hybrid game depicted in Figure 3.2. The differential equation attached to the location l_1 is of the form $\dot{x} \in P$, where P is the convex polyhedron shown in Figure 3.3(a), whose extremal directions are shown in Figure 3.3(b). Consider the valuation represented by the point q in Figure 3.3(c), the gray area in Figure 3.3(d) denotes all the possible evolutions of the system, and the dashed arrow is one of the possible admissible trajectory.

3.1.1 Semantics

The behavior of a HG is based on two types of steps:

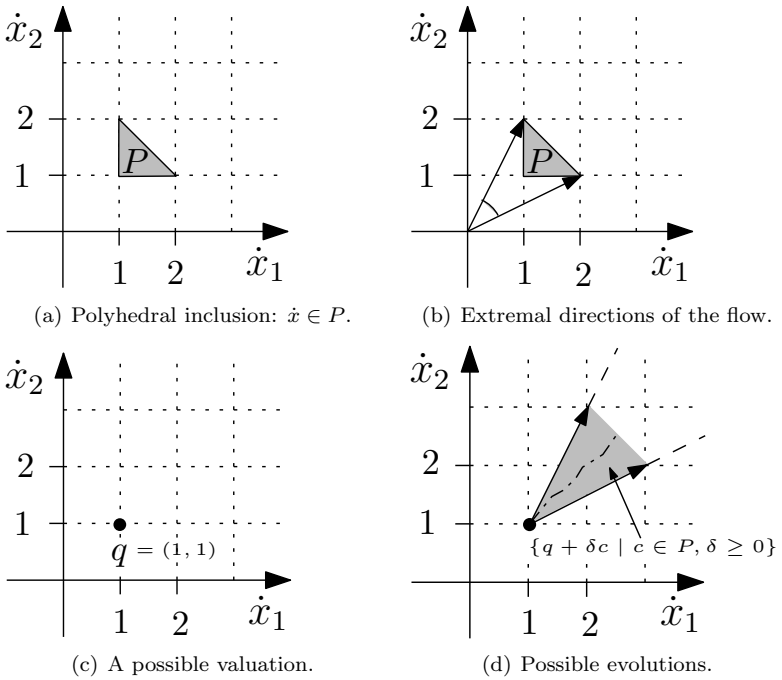
- *discrete* steps correspond to the *Edg* component, and produce an instantaneous change in both the location and the variable valuation;
- *timed* steps describe the change of the variables over time in accordance with the *Flow* component.

Given a state $s = \langle l, v \rangle$, $loc(s)$ denotes the location l and $val(s)$ denotes the valuation v.

An activity $f \in Acts(X)$ is called *admissible from* s if *(i)* $f(0) = v$ and *(ii)* for all $\delta \geq 0$ it holds $\dot{f}(\delta) \in Flow(l)$.

The set $Adm(s)$ contains all the activities that are admissible from s. Additionally, for $f \in Adm(s)$, the *span* of f in l, denoted by $span(f, l)$ is the set of all values $\delta \geq 0$ such that $\langle l, f(\delta') \rangle \in InvS$ for all $0 \leq \delta' \leq \delta$. Intuitively, δ is in the span of f iff f never leaves the invariant in the first δ time units. If all non-negative reals belong to $span(f, l)$, one can write $\infty \in span(f, l)$.

Runs. Given two states s, s', and a transition $e \in Edg$, there is a *discrete step* $s \xrightarrow{e} s'$ with *source* s and *target* s' iff *(i)* $s, s' \in InvS$, *(ii)* $e = (loc(s), \mu, loc(s'))$,



(a) Polyhedral inclusion: $\dot{x} \in P$.

(b) Extremal directions of the flow.

(c) A possible valuation.

(d) Possible evolutions.

Figure 3.3: Dynamics for the LHG of Figure 3.2.

and *(iii)* $(val(s), val(s')[X'/X]) \in \mu$, where $val(s')[X'/X]$ is the valuation in $Val(X')$ obtained from s' by renaming each variable in X with the corresponding primed variable in X'.

There is a *timed step* $s \xrightarrow{\delta,f} s'$ with *duration* $\delta \in \mathbb{R}^{\geq 0}$ and activity $f \in Adm(s)$ iff *(i)* $s \in InvS$, *(ii)* $\delta \in span(f, loc(s))$, and *(iii)* $s' = \langle loc(s), f(\delta) \rangle$.

For technical convenience, timed steps of duration zero are admitted.

Comparison with other models I. Some authors prohibit such time steps [ABD+00, BBV+03]. The formalism used in this thesis is more general, because timed steps of duration zero can be disabled by adding a clock variable t to the hybrid game and requesting that each discrete transition is enabled when $t > 0$ and resets t to 0 when taken. ∎

A special timed step is denoted $s \xrightarrow{\infty,f}$ and represents the case when the system follows an activity forever. This is only allowed if $\infty \in span(f, loc(s))$.

Finally, a *joint step* $s \xrightarrow{\delta,f,e} s'$ represents the timed step $s \xrightarrow{\delta,f} \langle loc(s), f(\delta) \rangle$ followed by the discrete step $\langle loc(s), f(\delta) \rangle \xrightarrow{e} s'$.

A *run* is a sequence

$$r = s_0 \xrightarrow{\delta_0,f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1,f_1} s'_1 \xrightarrow{e_1} s_2 \dots s_n \dots \tag{3.1}$$

of alternating timed and discrete steps, such that either the sequence is infinite, or it ends with a timed step of the type $s_n \xrightarrow{\infty,f}$.

The *length* of the run r is denoted by $len(r)$, and it is defined as follows:

$$len(r) = \begin{cases} n & \text{if the run r is finite} \\ \infty & \text{otherwise} \end{cases}$$

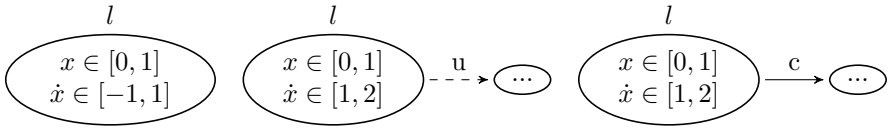
The set $States(r)$ denotes the set of all states visited by r. Formally, $States(r)$ is the smallest set containing all states $\langle loc(s_i), f_i(\delta)\rangle$, for all $0 \leq i \leq len(r)$ and all $0 \leq \delta \leq \delta_i$.

Notice that the states from which discrete steps start (states s'_i in (3.1)) appear in $States(r)$. Moreover, if r contains a sequence of one or more zero-time timed steps, all intervening states appear in $States(r)$.

Zenoness and well-formedness. As seen in Chapter 2, the above definitions admit Zeno runs. Also for the hybrid case, this thesis assumes that the hybrid game under consideration does not generate Zeno runs (see Chapter 2 for the definition of non-Zeno run). This is easily achieved by using an extra variable t, representing a clock ($\dot{t} = 1$), to ensure that the delay between any two discrete steps is bounded from below by a constant c (all transitions can only be taken when $t \geq c$ and then they reset t to zero).

The non-blocking definition seen in Chapter 2, must be replaced in order to take into account all admitted activities in a state of an hybrid game, instead of the only, deterministic, possible flow in thereal-time case. Formally, for all states s in the invariant, if all activities $f \in Adm(s)$ eventually leave the invariant, there exists one such activity f and a time $\delta \in span(f, loc(s))$ such that there is an uncontrollable transition enabled in $\langle loc(s), f(\delta)\rangle$, i.e., there exist $s' \in InvS$ and $e \in Edg_{\mathrm{u}}$ such that $\langle loc(s), f(\delta)\rangle \xrightarrow{e} s'$. If a hybrid game is non-Zeno and non-blocking, is said to be *well-formed*. In the following, all hybrid games are assumed to be well-formed.

Example 11. *Consider the LHGs in Figure 3.4. The fragment in Figure 3.4(a) is non-blocking, because the system may choose derivative $\dot{x} = 0$ and remain indefinitely in location l. The fragment in Figure 3.4(b) is also non-blocking, because the system cannot remain in l forever, but an uncontrollable transition leading outside is always enabled. Finally, the fragment in Figure 3.4(c) is blocking, because the system cannot remain in l forever, and no uncontrollable transition is enabled.*



(a) Non-blocking. (b) Non-blocking. (c) Blocking.

Figure 3.4: Three HG fragments. Locations contain the invariant (first line) and the flow constraint (second line). Solid (resp., dashed) edges represent controllable (resp., uncontrollable) transitions. Guards are `true`.

Comparison with other models II. In the work of Wong-Toi [WT97], the property called "control δ-positivity" plays the role of our non-blocking condition. Such property states that there is a (global) $\delta > 0$ such that if a controllable transition is enabled in state s, it is also possible to let δ time unit pass from s, without leaving the invariant. Essentially, this property constrains the guards of the controllable transitions to be detached from the boundary of the invariant by the equivalent of at least δ time units. The objective is to enable the controller to choose the null action even when the system is on the invariant boundary. The approach used in this thesis achieves the same effect by restricting the guards of the *un*controllable transitions. This is in line with the standard interpretation of the invariant: since it is an internal system constraint, system transitions alone should be able to enforce it. ∎

Strategies. In the hybrid context, the notion of strategy can be defined has the same form of the strategy defined in 2. Hence, a non-deterministic and memoryless (hybrid) strategy is a function $\sigma : S \to 2^{Edg_c \cup \{\bot\}} \setminus \emptyset$, where $\bot$ denotes the null action. A strategy can only choose a transition which is allowed by the hybrid game. Formally, for all $s \in S$, if $e \in \sigma(s) \cap Edg_c$, then there exists $s' \in S$ such that $s \xrightarrow{e} s'$. Moreover, when the strategy chooses the null action, it should continue to do so for a positive amount of time, along each activity that remains in the invariant. If all activities immediately exit the invariant, the above condition is vacuously satisfied. Formally,

- if $\bot \in \sigma(s)$, for all $f \in Adm(s)$ there exists $\delta > 0$ such that for all $0 < \delta' < \delta$ it holds $\delta' \notin span(f, loc(s))$ or $\bot \in \sigma(\langle loc(s), f(\delta')\rangle)$.

If all activities starting from s immediately leave the invariant, the above condition is vacuously satisfied.

Comparison with other models III. The strategies considered by Wong-Toi [WT97] are *deterministic*, i.e., of the type $\sigma : S \to Edg_c \cup \{\bot\}$, and subject to

the following condition: the set of states from which any given discrete transition is chosen is a closed polyhedron. As a consequence, these strategies satisfy the condition 3.1.1 on the null action. However, this condition is less restrictive, putting no restriction on the shape of the set of states where a given discrete transition is chosen. ∎

Notice that a strategy can always choose the null action. The well-formedness condition ensures that the system can always evolve in some way, by a timed step or an uncontrollable transition. In particular, even on the boundary of the invariant, the strategy allows the controller to choose the null action, because by the interpretation used in this thesis (the invariant is part of the system specification), the controller is not the responsible for ensuring that the invariant is not violated. The non-blocking assumption ensures that when *all* activities immediately leave the invariant, an uncontrollable transition is enabled.

A run like (3.1) is said to be *consistent* with a strategy σ if for all $0 \leq i < len(r)$ the following conditions hold:

- for all $\delta \geq 0$ such that $\sum_{j=0}^{i-1} \delta_j \leq \delta < \sum_{j=0}^{i} \delta_j$, we have $\perp \in \sigma(r(\delta))$;
- if $e_i \in Edg_{\mathrm{c}}$ then $e_i \in \sigma(s'_i)$.

The set $Runs(s, \sigma)$ denotes the set of runs starting from the state s and consistent with the strategy σ.

The following result ensures that each strategy has at least one run that is consistent with it, otherwise the controller may surreptitiously satisfy the safety objective by blocking the system.

The result can be proved by induction by considering that: as long as the strategy chooses the null action, the system may continue along one of the activities that remain within the invariant; if a state is reached from which all activities immediately leave the invariant, the non-blocking assumption ensures that there exists an uncontrollable transition that is enabled; finally, if the strategy chooses a discrete transition, that transition is enabled.

Theorem 1. *Given a non-blocking hybrid game, for all strategies σ and states $s \in InvS$, there exists a run that starts from s and is consistent with σ.*

Proof. Let

$$r = s_0 \xrightarrow{\delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1, f_1} s'_1 \xrightarrow{e_1} s_2 \ldots s_n$$

be a finite prefix of a run that starts in $s_0 = s$ and is consistent with σ (as base case, consider $r = s$). One can show that r can be extended with either an infinite timed step, or a timed step followed by a discrete step. If there

exists an activity $f \in Adm(s_n)$ that never leaves the invariant (i.e., such that $\infty \in span(f, loc(s_n))$), then r can be completed by the infinite timed transition $s_n \xrightarrow{\infty, f}$. Otherwise, since the hybrid game is non-blocking, there exists an activity $f \in Adm(s_n)$, a time $\delta \in span(f, loc(s_n))$, and an uncontrollable transition $e_{\mathrm{u}} \in Edg_{\mathrm{u}}$ such that e_{u} is enabled in the state $\langle loc(s_n), f(\delta)\rangle$. If, for all $0 \leq \delta' < \delta$, we have $\bot \in \sigma(\langle loc(s_n), f(\delta')\rangle)$, then r can be extended with the steps $s_n \xrightarrow{\delta, f} s' \xrightarrow{e_{\mathrm{u}}} s''$.

Otherwise, let $\hat{\delta}$ be the infimum of the delays $0 \leq \delta' < \delta$ such that $\bot \notin \sigma(\langle loc(s_n), f(\delta')\rangle)$. Notice that $\hat{\delta} \in span(f, loc(s_n))$. Now will be proved that $\bot \notin \sigma(\langle loc(s_n), f(\hat{\delta})\rangle)$, so that $\hat{\delta}$ is in fact the minimum of the above set. By definition of strategy, if by contradiction $\bot \in \sigma(\langle loc(s_n), f(\hat{\delta})\rangle)$, there exists $\delta^* > \hat{\delta}$ such that for all $\hat{\delta} < \delta' < \delta^*$ it holds $\bot \in \sigma(\langle loc(s_n), f(\delta')\rangle)$, against the definition of $\hat{\delta}$. Hence, $\bot \notin \sigma(\langle loc(s_n), f(\hat{\delta})\rangle)$. Since $\sigma(\langle loc(s_n), f(\hat{\delta})\rangle) \neq \emptyset$, let $e_{\mathrm{c}} \in \sigma(\langle loc(s_n), f(\hat{\delta})\rangle)$. It is easy to verify that r can be extended with the steps $s_n \xrightarrow{\hat{\delta}, f} s' \xrightarrow{e_{\mathrm{c}}} s''$. ∎

3.1.2 Control Problems for LHGs

Now safety and reachability control problems for linear hybrid games can be formally defined.

Safety control problem. Given a hybrid game $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$ and a set of safe states $T \subseteq InvS$, the *safety control problem* asks whether exists a winning strategy σ (for the controller) such that, for all initial states $s \in InitS$ and all runs $r \in Runs(s, \sigma)$ it holds $States(r) \subseteq T$.

Reachability control problem. Given a hybrid game $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$ and a set of target states $T \subseteq InvS$, the *reachability control problem* asks whether exists a winning strategy σ (for the controller) such that, for all initial states $s \in Init$ and all runs $r \in Runs(s, \sigma)$, it holds $States(r) \cap T \neq \emptyset$.

The solutions for safety and reachability control problems are fully explained in Chapter 4 and Chapter 5, respectively.

Part II

Solving the Control Problems for LHGs

Chapter 4

Solving the Safety Control Problem for LHGs

This chapter shows the proposed fix-point procedure in order to solve the safety control problem for linear hybrid games, i.e. the objective of keeping the system within a given region of safe states. This problem is known to be undecidable, for the general case of hybrid games. The complexity standing of the problem for the subclass of linear hybrid games was further refined to *semi-decidable* in [WT97], whose results allows the exact computation of the set of states that are reachable within a bounded number of discrete transitions (*bounded-horizon reachability*). Asarin et al. investigate the synthesis problem for hybrid systems where all discrete transitions are controllable and the trajectories satisfy given linear differential equations of the type $\dot{x} = Ax$ [ABD$^+$00]. The expressive power of these constraints is incomparable with the one offered by the differential inclusions occurring in LHGs. In particular, linear differential equations give rise to deterministic trajectories, while differential inclusions are non-deterministic. In control theory terms, differential inclusions can represent the presence of environmental disturbances.

The most powerful class of hybrid games whose control problem is decidable are initialized rectangular hybrid games[1]. Table 4.1 shows characterizations of control problems for several fragments of hybrid games (see [HKPV95] for more details).

For undecidable classes, there is an extensive literature describing approximate solutions [WT97, ABD$^+$00]. For example, Wong Toi proposed a fix-point procedure based on an appropriate version of the *controllable predecessor operator CPre* for hybrid games. Given a set of states X, the set $CPre^{\mathrm{S}}(X)$ (in the

[1] A RHG is said *initialized* if, whenever a continuous variables changes its dynamics, its value is nondeterministically reinitialized according to the invariant of the target location.

Class	Safety and Reachability Control Problems
Affine HGs	Undecidable [HKPV95]
Linear HGs	Semi-decidable [WT97]
Rectangular HGs	Semi-decidable [WT97]
Initialized RHGs	Decidable [HHM99]
Timed Games	Decidable [AMP95]

Table 4.1: HG fragments and Control Problems

hybrid case) contains all and only the states from which the controller can force the game into X in a single *step*[2]. The heart of the procedure lies in the operator $flow\_avoid(U, V)$, which computes the set of system configurations from which a continuous trajectory may reach the set U while avoiding the set V.

In this chapter it is proved that the $flow\_avoid$ operator provided by Wong-Toi does not work for non-convex V, a case which is very likely to occur in practice, even if the original safety goal is convex. The correct version of this operator, here called *may reach while avoiding operator* RWA^{m}, is then showed. The operator RWA^{m} takes as input two sets of states U and V, and computes the set of points from which there exists a trajectories that leads to the region U, while avoiding the region V.

One of the main contribution of this work is the sound and complete semi-algorithm for the safety control problem, based on the (sound and complete) implementation of the operator RWA^{m}, showed in this chapter.

4.1 Safety Control: the Abstract Algorithm

In this section, a fixed linear hybrid game are considered in order to present a sound and complete procedure to solve the safety control problem.

In order to formally define the correct version of the controllable predecessor for safety $CPre^{\mathrm{S}}$ (for hybrid games), some preliminary operators are now defined. For a set of states A and $x \in \{u, c\}$, the *predecessor* $Pre_x(A)$ is the set:

$$Pre_x(A) = \{s \in S \mid s \xrightarrow{e} s', \text{ with } s' \in A \text{ and } e \in Edg_x\},$$

and contains the states where some discrete transition belonging to Edg_x is enabled and leads to A. Let $\overline{A}$ be the set complement of A.

Controllable predecessor operator. Now the $CPre^{\mathrm{S}}$ operator can be formally defined. For a set of states A, the operator $CPre^{\mathrm{S}}(A)$ returns the set of

[2] A single "step" consists of a timed evolution of the system followed by one discrete transition.

states from which the controller can ensure that the system remains in A during the next pair of timed and discrete steps. This happens if for all activities chosen by the environment and all delays δ, one of the following two situations occurs:

- either the systems stays in A up to time δ, while all uncontrollable transitions enabled up to time δ (included) also lead to A, or
- there exists a time $\delta' < \delta$, such that the system stays in A up to time δ', all uncontrollable transitions enabled up to time δ' (included) also lead to A, and the controller can issue a transition at time δ' leading to A.

To improve readability, for a set of states A, an activity f, and a time delay $\delta \geq 0$ (including infinity), the operator *While* showed in Chapter 2 is now redefined. In particular, $\mathit{While}(A, f, \delta)$ is the set of states from where following the activity f for δ time units keeps the system in A all the time, and any uncontrollable transition taken meanwhile also leads into A. Formally,

$$\mathit{While}(A, f, \delta) = \Big\{ s \in S \,\Big|\, \forall 0 \leq \delta' \leq \delta : \langle \mathit{loc}(s), f(\delta') \rangle \in A \setminus \mathit{Pre}_u(\overline{A}) \Big\}.$$

The $\mathit{CPre}^{\mathrm{S}}$ operator is formally defined by:

$$\begin{aligned} \mathit{CPre}^{\mathrm{S}}(A) = \Big\{ s \in S \,\Big|\, & \forall f \in \mathit{Adm}(s), \delta \in \mathit{span}(f, \mathit{loc}(s)) : s \in \mathit{While}(A, f, \delta) \\ & \text{or } \exists 0 \leq \delta' < \delta : s \in \mathit{While}(A, f, \delta') \text{ and } \langle \mathit{loc}(s), f(\delta') \rangle \in \mathit{Pre}_c(A) \Big\}. \end{aligned}$$

The following theorem shows that the controllable predecessor operator can be used to characterize the safety control problem. This result is classical for all game-theoretic approaches to safety control [MPS95, ABD+00, Mal02]. Being classical, the details of its proof are often taken for granted in the literature, although they strongly depend on the precise definitions of the game model, its semantics and the notion of strategy. Hence, here a detailed proof is provided.

Theorem 2. *The answer to the safety control problem for safe set $T \subseteq \mathit{InvS}$ is positive if and only if*

$$\mathit{InitS} \subseteq \nu W \,.\, T \cap \mathit{CPre}^{\mathrm{S}}(W).$$

Proof. [***if***] First a winning strategy is built in two steps. Let $W^* = \nu W \,.\, T \cap \mathit{CPre}^{\mathrm{S}}(W)$ and let σ be a strategy defined as follows, for all states s:

- $\bot \in \sigma(s)$ and

- if $s \xrightarrow{e} s'$, $s, s' \in W^*$ and $e \in Edg_{\mathrm{c}}$, then $e \in \sigma(s)$.

While σ is clearly a strategy, it is not necessarily a winning strategy, as it may admit runs which delay controllable actions either beyond the safety set W^* or beyond their availability. However, a winning strategy can be recovered by restricting σ in appropriate ways. For all states $s \in S$ and activities $f \in Adm(s)$, let

$$D_{f,s} = \big\{\delta > 0 \mid \forall 0 \leq \delta' \leq \delta : \langle loc(s), f(\delta')\rangle \in W^* \text{ and } \sigma(\langle loc(s), f(\delta')\rangle) \cap Edg_{\mathrm{c}} \neq \emptyset\big\}.$$

denote the set of positive time units for which the system can follow activity f, starting from s, always remaining in W^* with some controllable transition enabled and available to the controller.

Starting from σ, one can define a new strategy σ' which coincides with σ on all the states, except for the states $s \in W^*$ with $Edg_{\mathrm{c}} \cap \sigma(s) \neq \emptyset$, where it satisfies $\sigma'(s) \subseteq \sigma(s)$ and the following two conditions:

a) If there is $f \in Adm(s)$ such that $D_{f,s} = \emptyset$, then $\perp \notin \sigma'(s)$;

b) for all $f \in Adm(s)$, if $D_{f,s} \neq \emptyset$ then there exists a $\delta \in D_{f,s}$ with $\perp \notin \sigma'(\langle loc(s), f(\delta)\rangle)$ and $\forall 0 \leq \delta'' < \delta$, $\perp \in \sigma'(\langle loc(s), f(\delta'')\rangle)$.

Intuitively, the new strategy σ' ensures that following any activity from a state $s \in W^*$ in which some controllable action is enabled, a controllable action will always be taken before none of them is available and before leaving W^*.

One can prove that σ' is winning, by showing that for every $s \in InitS$ and every $r \in Runs(\sigma', s)$, $States(r) \subseteq T$. Let

$$r = s_0 \xrightarrow{\delta_0, f_0} s'_0 \xrightarrow{e_0} s_1 \xrightarrow{\delta_1, f_1} s'_1 \xrightarrow{e_1} s_2 \ldots s_n \ldots$$

be a run consistent with σ'. The following properties can be proved:

1. if $s_i \xrightarrow{\delta_i, f_i} s'_i$ occurs in r, with $\delta_i > 0$ and $s_i \in W^*$, then for all $0 \leq \delta' \leq \delta_i$, it holds $\langle loc(s_i), f_i(\delta')\rangle \in W^*$;
2. if $s_i \xrightarrow{\infty, f_i}$ occurs in r and $s_i \in W^*$, then for all $\delta' \geq 0$, it holds $\langle loc(s_i), f_i(\delta')\rangle \in W^*$;
3. if $s_i \xrightarrow{e} s'_i$ occurs in r and $s_i \in W^*$, then $s'_i \in W^*$.

Now, the property (1) will be proved, as (2) can be proved similarly. Since $\delta_i > 0$, by the consistency of r with σ', then $\perp \in \sigma'(s_i)$. Assume, by contradiction, that $\langle loc(s_i), f_i(\delta')\rangle \notin W^*$ for some $0 < \delta' < \delta_i$. Since $s_i \in W^* =$

$CPre^{\mathrm{S}}(W^*)$, then $s_i \in While(W^*, f_i, \delta)$ for some $\delta \in \mathbb{R}^{\geq 0} \cup \{\infty\}$, and either $\delta = \infty$ or $s_i \xrightarrow{\delta, f_i} s$ and $s \in Pre_c(W^*)$.

If $\delta \geq \delta'$, an immediate contradiction is obtained, since it would imply $s_i \in While(W^*, f_i, \delta')$ and, therefore, $\langle loc(s_i), f_i(\delta')\rangle \in W^*$.

Assume, then, $\delta < \delta'$. Then $\langle loc(s_i), f_i(\delta)\rangle \in Pre_c(W^*)$, i.e., $\langle loc(s_i), f_i(\delta)\rangle \xrightarrow{e} s'$ for some $e \in Edg_{\mathrm{c}}$ and $s' \in W^*$. Therefore, both $e \in \sigma'(\langle loc(s_i), f_i(\delta)\rangle)$ and, by the consistency of r with σ', $\bot \in \sigma'(\langle loc(s_i), f_i(\delta)\rangle)$. Since $\bot \in \sigma'(\langle loc(s_i), f_i(\delta)\rangle)$, by definition of σ' the premise of property *a*) cannot hold. Therefore, by property *b*), there must be a $\delta \leq \delta^* < \delta'$ with $\bot \notin \sigma'(\langle loc(s_i), f_i(\delta^*)\rangle)$. On the other hand, the consistency of r requires that $\bot \in \sigma'(\langle loc(s_i), f_i(\hat{\delta})\rangle)$ for all $0 \leq \hat{\delta} < \delta_i$, which is a contradiction. Therefore, for all $0 \leq \delta' < \delta_i$, $\langle loc(s_i), f_i(\delta')\rangle \in W^*$.

Finally, proceed again by contradiction, is proved that $s'_i \in W^*$. Assume $s'_i \notin W^*$ and let $0 < \delta' < \delta_i$, then $\langle loc(s_i), f_i(\delta')\rangle \in W^*$. Therefore, $\langle loc(s_i), f_i(\delta')\rangle \in CPre^{\mathrm{S}}(W^*)$ and there exists $\delta' \leq \delta^* < \delta_i$ with $\langle loc(s_i), f_i(\delta')\rangle \in While(W^*, f_i, \delta^*)$ and $\langle loc(s_i), f_i(\delta^*)\rangle \in Pre_c(W^*)$. Hence, there is a controllable transition $e \in Edg_{\mathrm{c}}$ enabled in $\langle loc(s_i), f_i(\delta^*)\rangle$ and leading to W^*. As a consequence, $\{e, \bot\} \subseteq \sigma(\langle loc(s_i), f_i(\delta^*)\rangle)$ and, by condition *b*), $\bot \notin \sigma'(\langle loc(s_i), f_i(\overline{\delta})\rangle)$, for some $\delta^* < \overline{\delta} < \delta_i$, which contradicts consistency of r with σ', hence $s'_i \in W^*$.

Let us consider property (3). There are two cases. If $e \in Edg_{\mathrm{c}}$, then the consistency of r ensures that $e \in \sigma'(s_i)$ which, by definition of σ', requires that $s_{i+1} \in W^*$. Assume then that $e \in Edg_{\mathrm{u}}$. Then $\bot \in \sigma'(s_i)$. Since $s_i \in W^* = CPre^{\mathrm{S}}(W^*)$, it must hold $s_i \in While(W^*, f, 0)$, for every $f \in Adm(s_i)$. This, in turn, ensures that $s_i \in W^* \setminus Pre_u(\overline{W^*})$, therefore, all the uncontrollable transitions enabled in s_i lead to W^*. Hence the thesis.

To complete the proof, notice that $W^* \subseteq T$ and $s_0 \in Inits \subseteq W^*$. An easy induction on the length of r, using properties (1), (2) and (3), gives the result.

[***only if***] Let $s \notin W^*$, the fact that for all strategies there is a run that starts in s, that is consistent with the strategy and leaves T, is now proved. Let

- $W_0 = T$,
- $W_\alpha = T \cap CPre^{\mathrm{S}}(W_{\alpha-1})$, for a successor ordinal α, and
- $W_\alpha = \bigcap_{\beta<\alpha} W_\beta$ for a limit ordinal α.

The proof proceeds by induction on the smallest ordinal λ such that $s \notin W_\lambda$. If $\lambda = 0$, it holds $s \notin T$ and the thesis is immediate.

The fact that if $\lambda > 0$ then λ cannot be a limit ordinal, will be now shown. Assume by contradiction that λ is a limit ordinal. Since λ is the smallest

ordinal such that $s \notin W_\lambda$, we have $s \in W_\alpha$, for all $\alpha < \lambda$: this means that $s \in \bigcap_{\alpha<\lambda} W_\alpha$. But, since λ is a limit ordinal, $W_\lambda = \bigcap_{\alpha<\lambda} W_\alpha$ and we have that $s \in W_\lambda$, obtaining a contradiction.

Otherwise, if $\lambda > 0$ is a successor ordinal, then $s \in W_{\lambda-1} \setminus W_\lambda$ and $s \notin \mathit{CPre}^{\mathrm{S}}(W_{\lambda-1})$. According to the definition of $\mathit{CPre}^{\mathrm{S}}$, there exists an activity $f \in \mathit{Adm}(s)$ and $\delta \in \mathit{span}(s, f)$ such that $s \notin \mathit{While}(W_{\lambda-1}, f, \delta)$ and for all $0 \leq \delta' < \delta$ either $s \notin \mathit{While}(W_{\lambda-1}, f, \delta')$ or $\langle \mathit{loc}(s), f(\delta')\rangle \notin \mathit{Pre}_c(W_{\lambda-1})$.

Let δ^* be the infimum of those δ' such that $s \notin \mathit{While}(W_{\lambda-1}, f, \delta')$, i.e.,

$$\delta^* = \inf\{\delta \mid s \notin \mathit{While}(W_{\lambda-1}, f, \delta)\}. \tag{4.1}$$

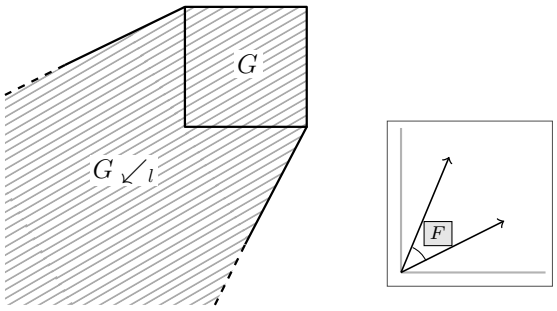
Clearly $0 \leq \delta^* \leq \delta$ and, for all $0 \leq \overline{\delta} < \delta^*$, $\langle \mathit{loc}(s), f(\overline{\delta})\rangle \notin \mathit{Pre}_c(W_{\lambda-1})$. Hence, any controllable transition enabled in $\langle \mathit{loc}(s), f(\overline{\delta})\rangle$, for any such $\overline{\delta}$, leads outside $W_{\lambda-1}$. Therefore, any strategy choosing a controllable transition in some of the states $\langle \mathit{loc}(s), f(\overline{\delta})\rangle$ has a consistent run leading outside $W_{\lambda-1}$. By inductive hypothesis, the thesis is obtained.

If, on the other hand, the strategy allows the controller to stay inactive in all those states, there is a consistent run that reaches δ^*. Then there are two cases. If δ^* is in fact the minimum of the above set, according to the definition of *While*, there exists $\delta_1 < \delta^*$ such that $\langle \mathit{loc}(s), f(\delta_1)\rangle \in \overline{W_{\lambda-1}} \cup \mathit{Pre}_u(\overline{W_{\lambda-1}})$. Therefore, since the controller may not act before δ^* along this strategy, there is a consistent run that reaches $\langle \mathit{loc}(s), f(\delta_1)\rangle$, which either is in $\overline{W_{\lambda-1}}$ or reaches it after an uncontrollable transition. In both cases, the thesis follows from the inductive hypothesis.

Finally, there is the case in which δ^* is the infimum but not the minimum of the above set. In this case $0 \leq \delta^* < \delta$ and $\langle \mathit{loc}(s), f(\overline{\delta})\rangle \notin \mathit{Pre}_c(W_{\lambda-1})$, for all $0 \leq \overline{\delta} \leq \delta^*$. Consider the choice of σ in state $\langle \mathit{loc}(s), f(\delta^*)\rangle$. If $\bot \notin \sigma(\langle \mathit{loc}(s), f(\delta^*)\rangle)$, the controller issues a discrete move which leads into $\overline{W_{\lambda-1}}$. If, instead, $\bot \in \sigma(\langle \mathit{loc}(s), f(\delta^*)\rangle)$, since $\delta^* < \delta \in \mathit{span}(s, f)$, by the definition of strategy σ will keep choosing $\bot$ for a non-zero amount of time γ. By (4.1), there exists $\delta^* < \hat{\delta} < \delta^* + \gamma$ such that $s \notin \mathit{While}(W_{\lambda-1}, f, \hat{\delta})$. As a consequence, there is a consistent run that reaches a state which either is in $\overline{W_{\lambda-1}}$ or reaches it after an uncontrollable transition. Once again, the thesis is obtained by inductive hypothesis. ∎

4.2 Computing the Predecessor Operator on LHGs

This section shows how the value of the predecessor operator on a given set of states A is computed, assuming that the hybrid game is a LHG and that the following operations on arbitrary polyhedra G and G' can be computed:



(a) Preflow of G in l ($G\swarrow_l$). (b) The Flow of l, $Flow(l)$.

Figure 4.1: The pre-flow operator.

1. the Boolean operations $G \cup G$, $G \cap G$, and $\overline{G}$;
2. the topological closure $cl(G)$ of G;
3. for a given location $l \in Loc$, the *pre-flow* of G in l:

$$G\swarrow_l = \{u \in Val(X) \mid \exists \delta \geq 0, c \in Flow(l) : u + \delta \cdot c \in G\}$$

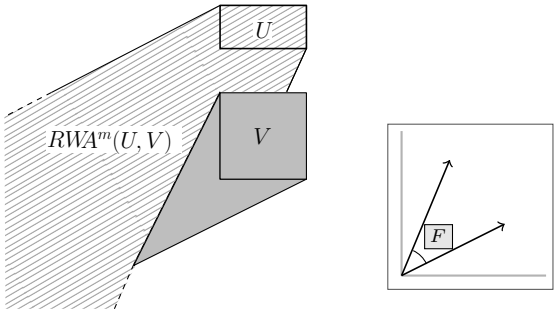
i.e. the set of all valuations from which, for an amount of time δ and a flow c on the location l, eventually leads the region G. For example, the gray pattern area in Figure 4.1(a) represents the pre-flow operator of the polyhedron G w.r.t. the flow shown in Figure 4.1(b).

Notice that, for two convex polyhedra P and P', if $P \subseteq P'$ then $P\swarrow_l \subseteq P'\swarrow_l$ (monotonicity), and $(P\swarrow_l)\swarrow_l = P\swarrow_l$ (idempotence). In the following, the basic components of $CPre^{\mathrm{S}}$ are first considered, and then the full operator will be treated. For all $A \subseteq InvS$ and $x \in \{c, u\}$, it holds:

$$Pre_x(A) = InvS \cap \bigcup_{(l,\mu,l') \in Edg_x} \mu^{-1}(A\downarrow_{l'}),$$

where $\mu^{-1}(Z) = \{v \in Val(X) \mid (v, v'[X'/X]) \in \mu, \text{ with } v' \in Z\}$ denotes the pre-image of Z w.r.t. μ. Also the auxiliary operator *may reach while avoiding* RWA^{m} is introduced. The implementation of RWA^{m} is one of the main topic of the thesis, being the core of the fix-point procedure to solve the safety control problem. Given a location l and two sets of variable valuations U and V, $RWA^{\mathrm{m}}_l(U, V)$ contains the set of valuations from which there exists a system trajectory that reaches U while avoiding $V \cap \overline{U}$ [3]. Notice that on a dense time domain this is not equivalent to reaching U while avoiding V: If an activity

[3] In ATL notation, $RWA^{\mathrm{m}}_l(U, V) \equiv \langle\langle env \rangle\rangle (\overline{V} \cup U) \mathcal{U} U$, where env is the player representing the environment.



(a) The area computed by $RWA^{\mathrm{m}}(U, V)$.

(b) The flow in l.

Figure 4.2: The may reach while avoid operator.

avoids V in a right-closed interval, and then enters $U \cap V$, the first property holds, while the latter does not. Formally:

$$RWA^{\mathrm{m}}_l(U,V) = \Big\{ u \in Val(X) \,\Big|\, \exists f \in Adm(\langle l, u\rangle),\, \delta \geq 0 : \\ f(\delta) \in U \text{ and } \forall\, 0 \leq \delta' < \delta : f(\delta') \in \overline{V} \cup U \Big\}.$$

Figure 4.2 shows the may reach while avoiding RWA^{m} operator on the polyhedra U and V. The filled gray area in Figure 4.2(a) contains all the points from which it is not possible to avoid the region V. Hence, these points do not belong to the output of the operator. On the other hand, the gray-pattern area in Figure 4.2(a) contains those points that may reach the region U meanwhile avoiding the region V, following any admitted activities according to the flow depicted in Figure 4.2(b).

An algorithm for effectively computing RWA^{m} on polyhedral arguments is presented in the next section, while the following lemma states the relationship between $CPre^{\mathrm{S}}$ and RWA^{m}.

Intuitively, consider the set B_l of valuations u such that from state $\langle l, u\rangle$ the environment can take a discrete transition leading outside A, and the set C_l of valuations u such that from $\langle l, u\rangle$ the controller can take a discrete transition into A. The RWA^{m} operator can be used to compute the set of valuations from which there exists an activity that either leaves A or enters B_l, while staying in the invariant and avoiding C_l. These valuations do not belong to $CPre^{\mathrm{S}}(A)$, as the environment can violate the safety goal within (at most) one discrete transition.

Before to formally define the relation between $CPre^{\mathrm{S}}$ and RWA^{m} an auxiliary definition is needed: a set of states $A \subseteq S$ is said to be *polyhedral* if for all $l \in Loc$, the projection $A|_l$ is a polyhedron.

Lemma 1. *For all polyhedral sets of states $A \subseteq InvS$, the following holds*

$$CPre^{\mathrm{S}}(A) = \bigcup_{l \in Loc} \{l\} \times \Big(A\lfloor_l \setminus RWA_l^{\mathrm{m}}\big(InvS\lfloor_l \cap (\overline{A\lfloor_l} \cup B_l), C_l \cup \overline{InvS\lfloor_l}\big)\Big), \quad (4.2)$$

where $B_l = Pre_u\big(\overline{A}\big)\lfloor_l$ and $C_l = Pre_c(A)\lfloor_l$.

Proof. In the following, let $I_l = InvS\lfloor_l$.

[$\subseteq$] Let $s = \langle l, u\rangle \in CPre^{\mathrm{S}}(A)$ and let $f \in Adm(s)$. By definition, $0 \in span(f, l)$ and hence $s \in While(A, f, 0)$. In particular, this implies that $s \in A$ and $u \in A\lfloor_l$.

Assume by contradiction that s does not belong to the r.h.s. of (4.2). Since $u \in A\lfloor_l$, it must be

$$u \in RWA_l^{\mathrm{m}}\big(I_l \cap (\overline{A\lfloor_l} \cup B_l), C_l \cup \overline{I_l}\big).$$

Then, by definition there exists $f^* \in Adm(s)$ and $\delta^* \geq 0$ such that: *(i)* $f^*(\delta^*) \in I_l \cap (\overline{A\lfloor_l} \cup B_l)$, and *(ii)* for all $0 \leq \delta < \delta^*$ it holds $f^*(\delta) \in I_l \cap \big(\overline{C_l} \cup \overline{A\lfloor_l} \cup B_l\big)$. In particular, this implies that δ^* belongs to $span(f^*, l)$. On the other hand, by applying the definition of $CPre^{\mathrm{S}}(A)$ to the activity f^*, we obtain that for all $\delta \in span(f^*, l)$ either $s \in While(A, f^*, \delta)$ or there exists $\delta' < \delta$ such that $s \in While(A, f^*, \delta')$ and $\langle l, f^*(\delta')\rangle \in Pre_c(A)$. This implies that either $f^*(\delta^*) \in A\lfloor_l \cap \overline{B_l}$ or there exists $\delta' < \delta^*$ such that $f^*(\delta') \in A\lfloor_l \cap \overline{B_l} \cap C_l$, which is a contradiction.

[$\supseteq$] Let $l \in Loc$ and $u \in A\lfloor_l \setminus RWA_l^{\mathrm{m}}\big(I_l \cap (\overline{A\lfloor_l} \cup B_l), C_l \cup \overline{I_l}\big)$. By complementing the definition of RWA^{m}, we obtain that for all activities f that start from $s = \langle l, u\rangle$ and for all times $\delta \geq 0$, either $f(\delta) \in \overline{I_l} \cup (A\lfloor_l \cap \overline{B_l})$ or there exists $\delta' < \delta$ such that

$$f(\delta') \in \Big(\overline{I_l} \cup \big(A\lfloor_l \cap \overline{B_l}\big)\Big) \cap (C_l \cup \overline{I_l}) = \overline{I_l} \cup \big(A\lfloor_l \cap \overline{B_l} \cap C_l\big) \triangleq E_l.$$

First, assume that for all $\delta \geq 0$ it holds $f(\delta) \in \overline{I_l} \cup (A\lfloor_l \cap \overline{B_l})$. In this case, for all $\delta \in span(f, l)$, the point $f(\delta)$ belongs to $A\lfloor_l \cap \overline{B_l}$. In other words, $s \in While(A, f, \delta)$ and hence $s \in CPre^{\mathrm{S}}(A)$.

Otherwise, there exists δ' such that $f(\delta') \in E_l$. Let δ^* be the infimum of the δ' with the above property, i.e., $\delta^* = \inf\{\delta' \mid f(\delta') \in E_l\}$. Notice that it holds $f(\delta) \in \overline{I_l} \cup \big(A\lfloor_l \cap \overline{B_l}\big)$ for all $\delta \leq \delta^*$, which implies $s \in While(A, f, \delta^*)$. If there exists $\delta \leq \delta^*$ such that $f(\delta) \in \overline{I_l}$, again we conclude that for all $\delta \in span(f, l)$ it holds $f(\delta) \in A\lfloor_l \cap \overline{B_l}$ and hence $s \in CPre^{\mathrm{S}}(A)$. In the rest of the proof, the property $f(\delta) \in I_l$ for all $\delta \leq \delta^*$, and therefore $\delta^* \in span(f, l)$, is assumed to be true.

If δ^* is in fact the minimum of the above set, i.e., $f(\delta^*) \in E_l$, then according to the current assumptions we have in particular $f(\delta^*) \in C_l = Pre_c(A)\lfloor_l$.

Accordingly, $s \in CPre^{\mathrm{S}}(A)$. Finally, it remains the case in which $f(\delta^*) \notin E_l$. By definition, in any neighborhood of δ^* there is a time δ such that $f(\delta) \in E_l$. Due to the fact that E_l is a polyhedron and that f is differentiable, there exists $\delta' > \delta^*$ such that $f(\delta) \in E_l$ for all $\delta^* < \delta \leq \delta'$. Therefore, $s \in While(A, f, \delta')$, and $\langle l, f(\delta') \rangle \in C_l = Pre_c(A)$. Again, we obtain that $s \in CPre^{\mathrm{S}}(A)$. ∎

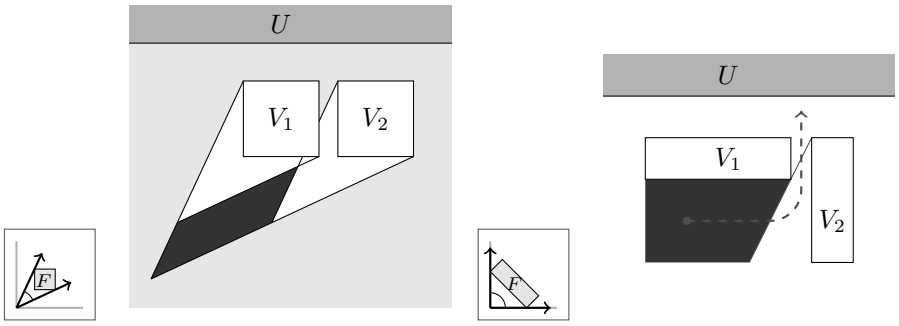
4.3 Computing the RWA^{m} Operator on Polyhedra

It's clear, from Lemma 1, that the computation of the RWA^{m} operator is the heart of the synthesis procedure proposed in this thesis. This section shows the algorithm to correctly compute the RWA^{m} operator, fixing some inaccuracies (4.4) of the similar $flow\_avoid$ operator proposed by Wong-Toi and its implementation in HoneyTech.

The first step of this section is check whether RWA^{m} for non-convex arguments can be expressed in terms of RWA^{m} for convex arguments. It is easy to verify that the first argument of RWA^{m} distributes over union, i.e., for all polyhedra U_1, U_2 and V it holds

$$RWA^{\mathrm{m}}_l(U_1 \cup U_2, V) = RWA^{\mathrm{m}}_l(U_1, V) \cup RWA^{\mathrm{m}}_l(U_2, V).$$

Hence, in the following one can assume that the first argument of RWA^{m} is a convex polyhedron.



(a) Avoiding the non-convex region $V_1 \cup V_2$ cannot be reduced to separately avoiding V_1 and V_2.

(b) Straight-line activities are not sufficient to avoid $V_1 \cup V_2$.

Figure 4.3: Basic properties of RWA^{m}. The boxes on the left represent the convex polyhedron $F = Flow(l)$ in the $(\dot{x}, \dot{y})$ plane. Thick arrows represent the *extremal directions* of flow.

Regarding the second argument (the set to be avoided), a run avoids $V_1 \cup V_2$ if and only if it avoids both V_1 and V_2. On the other hand, if there exists a run

that avoids V_1 (i.e., $s \in RWA_l^{\mathrm{m}}(\cdot, V_1)$) and a (possibly different) run avoiding V_2 (i.e., $s \in RWA_l^{\mathrm{m}}(\cdot, V_2)$), it does not mean that there exists a run that avoids both (i.e., $s \in RWA_l^{\mathrm{m}}(\cdot, V_1 \cup V_2)$). For instance, in the case pictured in Figure 4.3(a), the runs starting from the dotted area can avoid either V_1 or V_2 and reach U, but they cannot avoid both. Hence, the dotted area does not belong to $RWA_l^{\mathrm{m}}(U, V_1 \cup V_2)$, while it belongs to $RWA_l^{\mathrm{m}}(U, V_1) \cap RWA_l^{\mathrm{m}}(U, V_2)$.

Additionally, it is not possible to restrict the analysis from arbitrary activities (i.e., any differentiable function which stays in the invariant and whose slope belongs to $Flow(l)$) to straight-line activities. In Figure 4.3(b), the dotted area contains the set of points that cannot avoid $V_1 \cup V_2$ following straight-line activities. On the other hand, $RWA_l^{\mathrm{m}}(U, V_1 \cup V_2) = \overline{V_1} \cap \overline{V_2}$, because all other points (including those in the dotted area) can avoid $V_1 \cup V_2$ by passing through the gap between V_1 and V_2. In fact, those points can avoid $V_1 \cup V_2$ via a sequence of *two* straight-line activities. The following result shows that this observation can be generalized: if the system can move from point u to point v while avoiding a given polyhedral region, it can also do so via a finite sequence of straight-line activities.

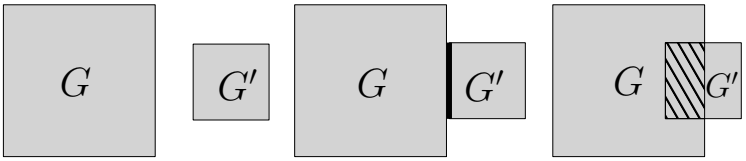
Lemma 2. [WT97] *Let l be a location, u and v be two valuations, and V a polyhedron. If there is an activity $f \in Adm(\langle l, u\rangle)$ and a time $\delta \geq 0$ such that $f(\delta) = v$ and $f(\delta') \notin V$ for all $0 \leq \delta' \leq \delta$, then there is a finite sequence $f_0, \ldots, f_k$ of straight-line activities and a finite sequence of delays $\delta_0, \ldots, \delta_k$ such that $f_0 \in Adm(\langle l, u\rangle)$, $f_k(\delta_k) = v$ for all $i = 0, \ldots, k-1$ it holds $f_{i+1} \in Adm(\langle l, f_i(\delta_i)\rangle)$, and for all $i = 0, \ldots, k$ and $0 \leq \delta' \leq \delta_i$ it holds $f_i(\delta') \notin V$.*

Now the algorithm for computing RWA^{m} can be presented. Given two polyhedra G and G', their *boundary* is defined to be

$$bndry(G, G') = (cl(G) \cap G') \cup (G \cap cl(G')).$$

which identifies a boundary between two (not necessarily closed) polyhedra. Clearly, $bndry(G, G')$ is not empty only if G and G' are adjacent to one another or if they overlap; it is empty, otherwise. Moreover, for any two convex polyhedra P and P', if $bndry(P, P')$ is not empty, then it is a convex polyhedron.

Figure 4.4 shows three possible cases: in Figure 4.4(a) polyhedra G and G' have no boundary, i.e. $bndry(G, G') = \emptyset$, because they are disjoint. Figure 4.4(b) shows the case in which G and G' are adjacent, and then their non-empty boundary is identified by the thick black line in the figure. The last case shown by Figure 4.4(c), is when G and G' are overlapped, and then the resulting boundary is the black pattern area in the figure.



(a) No boundary. (b) Lined-shaped boundary. (c) Area-boundary.

Figure 4.4: Boundary between Polyhedra G and G'.

Lemma 3. *For all convex polyhedra P and P', $bndry(P, P')$ is a convex polyhedron.*

Proof. Let x and y be two points in $bndry(P, P')$, and z_a be a convex combination of x and y, i.e., $z_a = ax + (1 - a)y$, with $0 < a < 1$. We prove that z_a belongs to $bndry(P, P')$.

Let $L = (P \cap cl(P'))$ and $R = (cl(P) \cap P')$, so that $bndry(P, P') = L \cup R$. Notice that L and R are convex polyhedra. Moreover, each point in $P \cap P'$ belongs to both L and R, and therefore to $bndry(P, P')$. Now, two cases can be identified:

1. If both x and y belong to L (resp., R), the thesis is a consequence of the convexity of L (resp., R).

2. Otherwise, assume w.l.o.g. that $x \in L$ and $y \in R$. By definition, we have that: *(i)* $x \in P$ and $y \in cl(P)$, hence $z_a \in P$, and *(ii)* $x \in cl(P')$ and $y \in P'$, hence $z_a \in P'$. Therefore, $z_a \in (P \cap P') \subseteq bndry(P, P')$, which concludes the proof.

∎

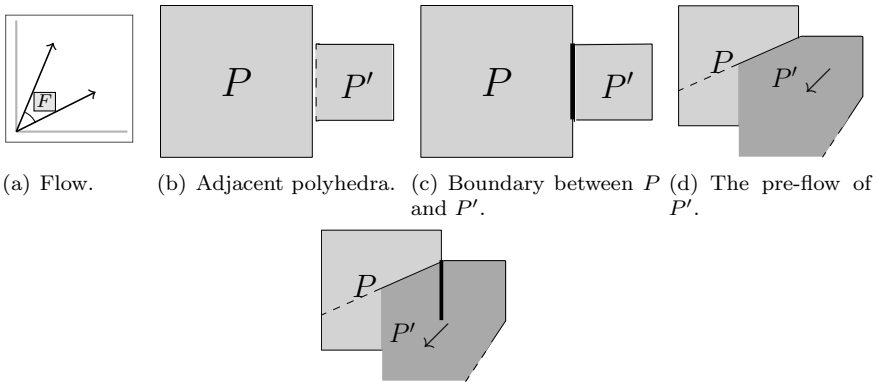
Given a location l and two polyhedra G and G', let $entry(G, G')$, the *entry region* from G to G', denote the set of points of the boundary between G and G', which can reach G' by following some straight-line activity in location l, while always remaining in $G \cup G'$. Formally:

$$entry(G, G') = \big\{p \in bndry(G, G') \mid p + \delta \cdot c \in G', \text{ for some } c \in Flow(l) \text{ and } \delta \geq 0, \text{ and for all } 0 \leq \delta' < \delta,\ p + \delta' \cdot c \in G \cup G'\big\}. \tag{4.3}$$

For two convex polyhedra P and P', $entry(P, P')$ can easily be computed as follows:

$$entry(P, P') = bndry(P, P') \cap P'^{\swarrow_l} . \tag{4.4}$$

Figure 4.5 shows how the entry region between the topological closed polyhedron P and the polyhedron P' (the open side is represented by the dashed line in Figure 4.5(b)) is computed. Figure 4.5(c) depicts the boundary between P and P', while Figure 4.5(d) shows the pre-flow of P', represented by the dark gray area. Finally, the entry region between P and P', i.e. $entry(P, P')$, is the intersection between $P'\swarrow_l$ and the boundary. The results of this intersection is still the boundary shown in Figure 4.5(c).



(a) Flow. (b) Adjacent polyhedra. (c) Boundary between P and P'. (d) The pre-flow of P'.

(e) The boundary and the pre-flow.

Figure 4.5: The entry region from P to P'.

Indeed $bndry(P, P') \subseteq P \cup P'$ is a convex polyhedron and, by definition of $P'\swarrow_l$, $bndry(P, P') \cap P'\swarrow_l$ is the set of points which can reach P' along a straight-line activity, while always remaining in $bndry(P, P') \subseteq P \cup P'$, as required by equation (4.3).

Notice, however, that equation (4.4) does not lift to general polyhedra. Indeed, while equation (4.4) holds even when P is not convex, it may not hold if the second argument is a non-convex polyhedron as demonstrated by the following example.

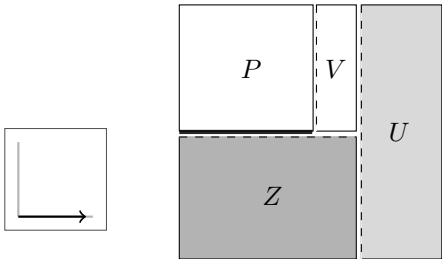


Figure 4.6: Example showing that $entry(P, G) \neq bndry(P, G) \cap G\swarrow_l$, for non-convex G. Flow is deterministic and horizontal.

Example 12. *Consider Figure 4.6 where U and Z are two convex polyhedra represented by gray boxes. Taking $G = U \cup Z$ and applying equation* (4.4) *to compute $entry(P, G)$ would result in the thick solid line between P and Z. However, this line does not belong to $entry(P, G)$ (in fact, $entry(P, G) = \emptyset$ in this case), since all of its points cannot avoid exiting from both P and G before eventually reaching $U \subseteq G$. Therefore, $entry(P, G) \neq bndry(P, G) \cap G_{\swarrow l}$.*

On the other hand, the second argument of *entry*() distributes over union.

Lemma 4. *For all polyhedra G, G_1 and G_2, it holds:*

$$entry(G, G_1 \cup G_2) = entry(G, G_1) \cup entry(G, G_2).$$

Proof. By monotonicity of *entry*() w.r.t. the second argument, it follows immediately that $entry(G, G_1) \cup entry(G, G_2) \subseteq entry(G, G_1 \cup G_2)$.

As to the other direction, let $p \in bndry(G, G_1 \cup G_2)$ be such that for some $c \in Flow(l)$ and $\delta \geq 0$, it holds $p + \delta \cdot c \in G_1 \cup G_2$, and for all $0 \leq \delta' < \delta$, it holds $p + \delta' \cdot c \in G \cup G_1 \cup G_2$. Clearly, $p \in bndry(G, G_1)$ or $p \in bndry(G, G_2)$. Now, for any polyhedron G', let us define:

$$\Delta_p^c(G') = \{\delta \geq 0 \mid p + \delta \cdot c \in G'\}.$$

Intuitively, $\Delta_p^c(G')$ is the set of delays during which the straight-line activity with slope c that starts in p stays in G'. Consider $\delta^* = \inf \Delta_p^c(G_1 \cup G_2)$. Clearly, $\delta^* \leq \delta$. Moreover, by polyhedricity of G_1 and G_2, either $\delta^* = \inf \Delta_p^c(G_1)$ or $\delta^* = \inf \Delta_p^c(G_2)$. Two cases can be identified:

i. if δ^* is the minimum of $\Delta_p^c(G_1 \cup G_2)$, then either we have $p + \delta^* \cdot c \in G_1$ or $p + \delta^* \cdot c \in G_2$, and for all $0 \leq \delta' < \delta^*$, $p + \delta' \cdot c \in G$. Moreover, if $p + \delta^* \cdot c \in G_1$ (resp., $p + \delta^* \cdot c \in G_2$) then $p \in bndry(G, G_1)$ (resp., $p \in bndry(G, G_2)$). In either case, $p \in entry(G, G_1) \cup entry(G, G_2)$;

ii. if $\delta^* \notin \Delta_p^c(G_1 \cup G_2)$, then for all $0 \leq \delta' \leq \delta^*$, $p + \delta' \cdot c \in G$. Assume $\delta^* = \inf \Delta_p^c(G_1)$ (the case where $\delta^* = \inf \Delta_p^c(G_2)$ is similar), then $\delta^* \notin \Delta_p^c(G_1)$ either. Since, however, δ^* is the infimum of the set, in any neighbourhood of δ^* there must be a $\hat{\delta} > \delta^*$ such that for all $\delta^* < \delta'' \leq \hat{\delta}$ we have $p + \delta'' \cdot c \in G_1$. From the above reasoning, $p + \hat{\delta} \cdot c \in G_1$ and, for all $0 \leq \delta' < \hat{\delta}$, $p + \delta' \cdot c \in G \cup G_1$, can be concluded. Moreover, $p + \delta^* \cdot c \in bndry(G, G_1)$ and, since for all $0 \leq \delta'' \leq \delta^*$ it holds $p + \delta'' \cdot c \notin G_2$, it follows that also $p \in bndry(G, G_1)$. Hence, $p \in entry(G, G_1) \subseteq entry(G, G_1) \cup entry(G, G_2)$.

∎

As a consequence of Lemma 4, for any convex P and any general polyhedron G, $entry(P, G)$ can be computed by simply collecting the entry regions from P to each convex polyhedron in $[\![G]\!]$, as follows:

$$entry(P, G) = \bigcup_{P' \in [\![G]\!]} entry(P, P'). \tag{4.5}$$

where $entry(P, P')$ is computed according to equation (4.4).

Now the RWA^{m} operator can be computed by the following fixpoint characterization.

Theorem 3. *For all locations l and sets of valuations U, V, and W, let*

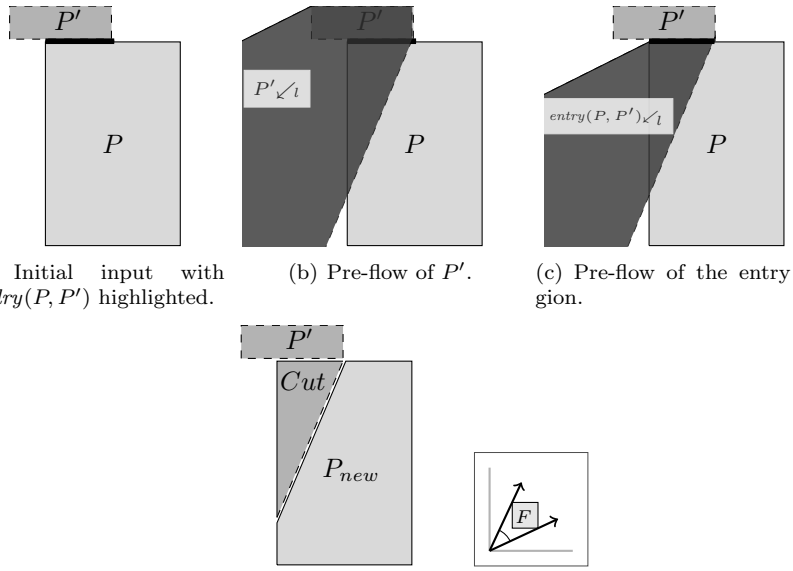
$$\tau(U, V, W) = U \cup \bigcup_{P \in [\![\overline{V}]\!]} \Big(P \cap entry(P, W)\swarrow_l \Big). \tag{4.6}$$

Hence, $RWA^{\mathrm{m}}_l(U, V) = \mu W \,.\, \tau(U, V, W)$.

Roughly speaking, $\tau(U, V, W)$ represents the set of points which either belong to U or do not belong to V and can reach W along a straight line which does not cross V. The fixpoint expression $\mu W \,.\, \tau(U, V, W)$ can be interpreted as an incremental refinement of an under-approximation to the desired result. The process starts with the initial approximation $W_0 = U$. One can easily verify that $U \subseteq RWA^{\mathrm{m}}_l(U, V)$. Additionally, notice that $RWA^{\mathrm{m}}_l(U, V) \subseteq U \cup \overline{V}$. The equation refines the under-approximation by identifying its *entry regions*, i.e., the boundaries between the area which *may* belong to the result (i.e., $\overline{V}$), and the area which already belongs to it (i.e., W).

Example 13. *Figure 4.7 shows a single step in the computation of equation (4.6), for a fixed pair of convex polyhedra P in $\overline{V}$ and P' in W, assumung that $Flow(l)$ is the plolyhedron F displayed in Figure 4.7(e). In all the figures, dashed lines represent topologically open sides. In Figure 4.7(a), the thick segment between P and P' represents $bndry(P, P')$, which, in the example, is contained in P. Since P' is topologically open (denoted by the dashed contour), the rightmost point of $bndry(P, P')$ cannot reach P' along any straight-line activity. Being P' open, so is $P'\swarrow_l$, and its intersection with P, namely $entry(P, P')$, does not contain the rightmost point of the boundary (see Figure 4.7(b)).*

Now, any point of P that can reach $entry(P, P')$ (displayed in Figure 4.7(c)) following some activity can also reach P'. The set $Cut = P \cap entry(P, P')\swarrow_l$ contains precisely those points (see Figure 4.7(d)). All these points must then be added to W, as they all belong to $RWA^{\mathrm{m}}_l(U, V)$.



(a) Initial input with $bndry(P, P')$ highlighted.

(b) Pre-flow of P'.

(c) Pre-flow of the entry region.

(d) The region cut from P and the resulting polyhedron.

(e) $Flow(l)$.

Figure 4.7: One step of RWA^{m} computation.

4.4 Previous Algorithms

As said many times, previous algorithms to solve the safety control problem for linear hybrid games, are wrong. This section shows these algorithms, and compares them with procedure proposed in this thesis.

In the literature, the standard reference for safety control of linear hybrid games is [WT97], where the proposed model and the abstract algorithm are essentially similar to that shown in this thesis.

As to the computation of $CPre^{\mathrm{S}}$, they introduce an operator *flow_avoid*, which corresponds to our RWA^{m} operator. They propose to compute $RWA^{\mathrm{m}}_l(U, V)$ using the following fixpoint formula:

$$\mathit{flow\_avoid}(U, V) = \bigcup_{U' \in [\![U]\!]} \bigcap_{V' \in [\![V]\!]} \Big(\mu W \,.\, U' \cup \bigcup_{P \in [\![\overline{V'}]\!]} \big(cl(P) \cap \overline{V'} \cap (W \cap P)_{\swarrow l} \big) \Big) \tag{4.7}$$

The example in Figure 4.3(a), already discussed in Section 4.3, shows that (4.7) is different from (in particular, larger than) $RWA^{\mathrm{m}}_l(U, V)$ when V is non-convex. The problem lies in the fact that Formula (4.7) treats each convex part of V separately, and then takes the intersection of the resulting sets. Considering the situation in Figure 4.3(a) and taking $V = V_1 \cup V_2$, Formula (4.7) would reduce to $\mathit{flow\_avoid}(U, V_1) \cap \mathit{flow\_avoid}(U, V_2)$, and would include the dotted

area depicted in that figure. As explained in Section 4.3, this is an incorrect result.

In [DMT+01], Deshpande et al. report about an implementation of Wong-Toi's algorithm in the tool HoneyTech, obtained as an extension of HyTech [DMT+01]. The fixpoint formula that is meant to capture $RWA_l^{\mathrm{m}}(U,V)$ is the following:

$$\mu W \,.\, U \cup \bigcup_{P \in [\![\overline{V}]\!]} \Big(P \cap \big(cl(W) \cap cl(P) \cap \overline{V} \cap W{\swarrow_l}\big){\swarrow_l} \Big) \tag{4.8}$$

Differently from (4.7), Formula (4.8) correctly treats the case of non-convex V. However, it suffers from another issue, related to the computation of the entry regions between polyhedra. First, notice that Formula (4.8) can be compared to (4.6), once observed that $(cl(W) \cap cl(P) \cap \overline{V} \cap W{\swarrow_l})$ is meant to correspond to the definition given here of entry regions. In particular, $(cl(W) \cap cl(P) \cap \overline{V})$ essentially corresponds to $bndry(P,W)$, except for subtle, though non crucial, differences. Once computed the boundaries between P and W, Formula (4.8) computes the entry regions from P to the polyhedron W, by intersecting them with $W{\swarrow_l}$, the pre-flow of W. This corresponds to using equation (4.4) with polyhedron W as the second argument. However, Example 12 shows that this may lead to errors when W is not convex. Indeed, consider again Figure 4.6, and assume that W is the union of U and the Z. The result of applying $(cl(W) \cap cl(P) \cap \overline{V} \cap W{\swarrow_l})$ is, in this case, the thick solid line between P and Z. This is precisely the wrong entry region computed by $bndry(P,W) \cap W{\swarrow_l}$ in Example 12. Since this thick line is contained in P, Formula (4.8) ends up adding it to $RWA_l^{\mathrm{m}}(U,V)$. However, this line does not belong to $RWA_l^{\mathrm{m}}(U,V)$, since all its points cannot avoid hitting V before eventually reaching U.

4.5 Soundness and Completeness of the Fixpoint Procedure of Theorem 3.

The first step of this section is to show that the τ operator is monotonic w.r.t. its third argument, so that the least fixpoint

$\mu W \,.\, \tau(U,V,W)$ is well defined.

Lemma 5. *For all polyhedra U, V, and $W \subseteq W'$, it holds $\tau(U,V,W) \subseteq \tau(U,V,W')$.*

Proof. It is sufficient to observe that, for all $P \in [\![\overline{V}]\!]$, the expression $P \cap entry(P,W){\swarrow_l}$ is monotonic w.r.t. W, since it is composed by monotonic operators. ∎

Theorem 3 is an immediate consequence of the following two lemmas.

Lemma 6. *For all locations l and polyhedra U and V, it holds $RWA_l^{\mathrm{m}}(U,V) \subseteq \mu W\,.\,\tau(U,V,W)$.*

Proof. Let $u \in RWA_l^{\mathrm{m}}(U,V)$ and $W^* = \mu W\,.\,\tau(U,V,W)$. By definition, $u \in \overline{V} \cup U$. If u belongs to U, then it belongs to W^* by definition. If u belongs to $\overline{V} \setminus U$, there must be an activity that starts in u and reaches a point $u' \in U$ without visiting $V \setminus U$. By Lemma 2, there is a finite sequence of straightline segments leading from u to u' and avoiding $V \setminus U$. Let $u_0, u_1, \ldots, u_k$ be the corresponding sequence of intermediate corner points, where $u_0 = u$ and $u_k = u'$. The proof proceeds by induction on k. If $k = 0$, it holds $u = u' \in U$, and the thesis is trivially true. If $k > 0$, the inductive hypothesis is applied to u_1, and then $u_1 \in W^*$. Consider the straight path from $u_0 \in \overline{V} \setminus U$ to $u_1 \in W^*$. This path crosses into W^* in a given point v. Formally, v is the first point along the path which belongs to $cl(W^*)$. Hence, there is at least one convex polyhedron $P' \in [\![W^*]\!]$ such that $v \in cl(P')$. If there is more than one such polyhedron, pick the one that contains at least one point of the straight path from v to u_1. In this way, we have $v \in P'{\swarrow_l} \subseteq W^*{\swarrow_l}$.

Let n be the number of convex polyhedra in $[\![\overline{V} \cup U]\!]$ that are crossed by the straight path from u_0 to v. The proof proceeds by a new induction on n. If $n = 1$, the whole line segment from u_0 to v is contained in a given $P \in [\![\overline{V} \cup U]\!]$. Hence, $v \in bndry(P, P')$, where P' is a suitable element of $[\![W^*]\!]$, which ensures that $v \in bndry(P, W^*)$ as well. Summarizing, we have $v \in bndry(P,P') \cap P'{\swarrow_l} = entry(P,P') \subseteq entry(P,W^*)$, and $u_0 \in \{v\}{\swarrow_l} \subseteq entry(P,W^*){\swarrow_l}$. Hence one can conclude that $u_0 \in W^*$. If $n > 1$, the straight path from u_0 to v is divided into n segments, defined by the intermediate points $v_1, \ldots, v_{n-1}$, and the inductive hypothesis is applied to v_1, obtaining that $v_1 \in W^*$. Finally, an argument analogous to the one for $n = 1$ is used to conclude that $u_0 \in W^*$. ■■

Lemma 7. *For all locations l and polyhedra U and V, it holds $RWA_l^{\mathrm{m}}(U,V) \supseteq \mu W\,.\,\tau(U,V,W)$.*

Proof. It suffices to show that $RWA_l^{\mathrm{m}}(U,V)$ is a fixpoint of r, in other words that $RWA_l^{\mathrm{m}}(U,V) = \tau(U,V,RWA_l^{\mathrm{m}}(U,V))$. Let $u \in \tau(U,V,RWA_l^{\mathrm{m}}(U,V))$, the fact that $u \in RWA_l^{\mathrm{m}}(U,V)$ must be proved. If $u \in U$, the thesis is obvious. Otherwise, there exist $P \in [\![\overline{V}]\!]$ such that $u \in P \cap entry(P, RWA_l^{\mathrm{m}}(U,V)){\swarrow_l}$. Hence, there is a straightline activity $f \in Adm(\langle l, u\rangle)$ that reaches a point $v \in RWA_l^{\mathrm{m}}(U,V)$, while remaining in $P \cup RWA_l^{\mathrm{m}}(U,V) \subseteq \overline{V}$. Therefore, we have found an activity from u to $RWA_l^{\mathrm{m}}(U,V)$ which avoids $V \setminus U$ and the thesis follows. Finally, let $u \in RWA_l^{\mathrm{m}}(U,V)$, the property that $u \in \tau(U,V,RWA_l^{\mathrm{m}}(U,V))$

is now proved. First, notice that $u \in U \cup \overline{V}$. If $u \in U$, the thesis is obvious. Otherwise, $u \in P$ for some $P \in [\![RWA_l^{\mathrm{m}}(U,V)]\!]$. Then, $P \subseteq \overline{V}$. Since $P \subseteq bndry(P, RWA_l^{\mathrm{m}}(U,V))$ and $P \subseteq P{\swarrow_l}$, we have that $P \subseteq bndry(P, RWA_l^{\mathrm{m}}(U,V)) \cap P{\swarrow_l}$. Therefore, there is a straight-line activity $f \in Adm(\langle l, u\rangle)$ that reaches $RWA_l^{\mathrm{m}}(U,V)$, starting from u, while remaining in $P \cup RWA_l^{\mathrm{m}}(U,V) = RWA_l^{\mathrm{m}}(U,V)$. Hence, $u \in P \cap entry(P, RWA_l^{\mathrm{m}}(U,V)){\swarrow_l}$ and, by (4.6), the thesis holds. ∎

4.6 Termination of the Fixpoint Procedure in Theorem 2.

In order to prove termination of the fixpoint procedure defined in Theorem 3, an equivalent but finer grained formulation of the τ operator is now provided.

Observe that distribution over union of ${\swarrow_l}$ ensures that, for any convex polyhedron P and any polyhedron G, the following holds:

$$\begin{aligned} P \cap entry(P,G){\swarrow_l} &= P \cap \Big(\bigcup_{P' \in [\![G]\!]} entry(P,P') \Big){\swarrow_l} = \\ &= P \cap \bigcup_{P' \in [\![G]\!]} \Big(entry(P,P'){\swarrow_l} \Big) = \\ &= \bigcup_{P' \in [\![G]\!]} \Big(P \cap entry(P,P'){\swarrow_l} \Big). \end{aligned} \tag{4.9}$$

As a consequence, equation (4.6) can be equivalently reformulate as follows:

$$\tau(U,V,W) = U \cup \bigcup_{P \in [\![\overline{V}]\!]} \bigcup_{P' \in [\![W]\!]} \Big(P \cap entry(P,P'){\swarrow_l} \Big). \tag{4.10}$$

The following theorem ensures that the fixpoint defined in Theorem 3 always terminates.

Theorem 4. *The fixpoint procedure for RWA^{m} defined as $\mu W \,.\, \tau(U,V,W)$ terminates in a finite number of steps.*

In order to prove Theorem 4, some additional definitions and notation are required.

Given two polyhedra E and G and two convex polyhedra $P \in [\![E]\!]$ and $P' \in [\![G]\!]$, if the entry region R from P to P' is not empty, the notation $G \xrightarrow{P,R}_E G'$, where $[\![G']\!] = [\![G]\!] \cup \{P \cap R{\swarrow_l}\}$, is used to denote a *refinement step*.

Intuitively, according to equation (4.10), the fixpoint procedure to compute RWA^{m} applies, at each iteration $k \geq 1$, all the refinement steps of the form $G \xrightarrow{P,R}_E G'$, with $E = \overline{V}$ and $G = \tau^{k-1}(U,V,U)$ (where $\tau^0(U,V,U) = U$ and $\tau^{i+1}(U,V,U) = \tau(U,V,\tau^i(U,V,U))$) for every entry region R of the current

under-approximation G, following a breadth-first policy. For example, Figure 4.7 shows a single refinement step of the form $W \xrightarrow{P,entry(P,P')}_{\overline{V}} W \cup \{Cut\}$, where $P \in [\![\overline{V}]\!]$ and $P' \in [\![W]\!]$.

The following lemma can easily be proved exploiting idempotence and monotonicity of $\swarrow_l$.

Lemma 8. *Assume $G \xrightarrow{P,R}_E G'$. For all entry regions R' of G' that are not entry regions of G it holds $R' \subseteq R\swarrow_l$.*

Proof. By definition of entry region, $R' = bndry(P', P \cap R\swarrow_l) \cap (P \cap R\swarrow_l)\swarrow_l$, with $P' \in [\![E]\!]$ and $P \cap R\swarrow_l \in [\![G']\!]$. Hence, we can write $R' \subseteq (P \cap R\swarrow_l)\swarrow_l$. Moreover, from $(P \cap R\swarrow_l) \subseteq R\swarrow_l$ and by monotonicity and idempotence properties of $\swarrow_l$ it follows that $(P \cap R\swarrow_l)\swarrow_l \subseteq R\swarrow_l$. Hence the thesis $R' \subseteq (P \cap R\swarrow_l)\swarrow_l \subseteq R\swarrow_l$. ∎ ∎

Now the relationship between sequences of refinement steps and it iteretions of the operator $\tau(\cdot)$ can be formally defined with the following lemma.

Lemma 9. *If $\pi = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_m,R_m}_E G_m$ is a sequence of refinement steps with R entry region of G_m, then R is an entry region of $\tau^m(G_0, \overline{E}, G_0)$.*

Proof. Let $\overline{\pi} = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_k,R_k}_E G_k$ be the shortest prefix of π such that R is entry region of G_k. Clearly, $k \leq m$. The proof now proceeds by induction on k. If $k = 0$, then R is entry region of $G_0 = \tau^0(G_0, \overline{E}, G_0)$ and, by monotonicity of the operator τ, the thesis holds.
Assume $k > 0$. Since R is entry region of G_k but not in G_{k-1} and $[\![G_k]\!] = [\![G_{k-1}]\!] \cup \{P_k \cap R_k\swarrow_l\}$, it must be $R = bndry(P, (P_k \cap R_k\swarrow_l)) \cap (P_k \cap R_k\swarrow_l)\swarrow_l$, with $P \in [\![E]\!]$ and R_k entry region in G_{k-1}. By induction hypothesis, R_k is entry region of $\tau^{k-1}(G_0, \overline{E}, G_0)$. Since $P_k \in [\![E]\!]$, by definition of τ we have $(P_k \cap R_k\swarrow_l) \in \tau(G_0, \overline{E}, \tau^{k-1}(G_0, \overline{E}, G_0)) = \tau^k(G_0, \overline{E}, G_0)$. Therefore, R is an entry region in $\tau^k(G_0, \overline{E}, G_0)$. Again, by monotonicity of τ, the thesis follows. ∎

In the following the fact that the number of different entry regions employed by the fixpoint procedure for RWA^{m} is finite and the fact that the number of its iterations is bounded is now proved, thus establishing termination of the procedure itself.

Before proceeding, some properties about sequences of refinement steps are required. Given a sequence $\pi = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_k,R_k}_E G_k$, let $last(\pi)$ denotes G_k. Moreover, for a convex polyhedron R, let $prune(\pi, R)$

be the sequence obtained from π by removing all those edges $\xrightarrow{P_i,R_i}_E$ which depend on R, i.e. such that $R_i \subseteq R\swarrow_l$. Formally, $prune(\pi, R) = G_0 \xrightarrow{P'_1,R'_1}_E G'_1 \xrightarrow{P'_2,R'_2}_E \ldots \xrightarrow{P'_m,R'_m}_E G'_m$ is the largest subsequence of π which is a sequence of refinement steps and such that $R'_i \neq R$, for all $1 \leq i \leq m$. Clearly, we have $m \leq k$.

The following lemma states that $prune(\pi, R)$ preserves all the entry regions of $last(\pi)$ which do not depend on R.

Lemma 10. *Let $\pi = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_k,R_k}_E G_k$ be a sequence of refinement steps and let R be an entry region of G_k, such that $R \not\subseteq R_1\swarrow_l$. Then, there exists a subsequence π' of $prune(\pi, R_1)$ such that R is an entry region of $last(\pi')$.*

Proof. The proof proceeds by induction on k. If $k = 1$, then R is an entry region of G_1, with $R \subseteq R_1\swarrow_l$. By Lemma 8 R must be entry region of G_0.

If $k > 1$, let j be the smallest index such that R is an entry region in G_j. If $j = 0$, the thesis holds. Otherwise, by Lemma 8 we have $R \subseteq R_j\swarrow_l$. Consequently, $R_j \not\subseteq R_1\swarrow_l$ (otherwise, by monotonicity it would hold $R \subseteq R_1\swarrow_l$). Applying the inductive hypothesis to the prefix $G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_{j-1},R_{j-1}}_E G_{j-1}$ and to R_j. We obtain that there exists a sequence π' that starts from G_0, does not use R_1, and ends in a polyhedron G' such that R_j is an entry region of G'. Hence, for the sequence $\pi' \xrightarrow{P_j,R_j}_E G''$, R is an entry region of G'' and the thesis holds. ∎

Now the main property relating entry regions and sequences of refinement steps can be stated.

Lemma 11. *Let $\pi = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_n,R_n}_E G_n$ be a sequence of refinement steps and let R be an entry region of G_n. Then, there exists a subsequence $\pi' = G_0 \xrightarrow{P'_1,R'_1}_E G'_1 \xrightarrow{P'_2,R'_2}_E \ldots \xrightarrow{P'_m,R'_m}_E G'_m$, such that: R is an entry region of G'_m and $P'_i \neq P'_j$ for all $1 \leq i < j \leq m$.*

Proof. Let $\overline{\pi} = G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_k,R_k}_E G_k$ be the shortest prefix of π such that R is an entry region of G_k. We proceed by induction on k. If $k = 0$ or $k = 1$, the thesis immediately follows. If $k > 1$, then R_k is an entry region of G_{k-1} and R is an entry region in G_k. Since k is the first index for which R is an entry region in G_k, then also holds that $R \subseteq P_k \cap R_k\swarrow_l$. The inductive hypothesis is now applied on $G_0 \xrightarrow{P_1,R_1}_E G_1 \xrightarrow{P_2,R_2}_E \ldots \xrightarrow{P_{k-1},R_{k-1}}_E G_{k-1}$ to obtain the subsequence $\pi' = G_0 \xrightarrow{P'_1,R'_1}_E G'_1 \xrightarrow{P'_2,R'_2}_E \ldots \xrightarrow{P'_h,R'_h}_E G'_h$, where $P'_i \neq P'_j$, for all $1 \leq i < j \leq h$, and R_k is still an entry region of G'_h.
Hence, $\pi^* = \pi' \xrightarrow{P_k,R_k}_E G'_{h+1}$ is a sequence of refinement steps, and $R \subseteq$

$P_k \cap R_k \swarrow_l$ implies that R is an entry region of G'_{h+1}. Assume $P'_j = P_k$ for some $1 \leq j \leq h$. Considering the subsequence $\hat{\pi} = G'_{j-1} \xrightarrow{P'_j, R'_j}_E G'_j \xrightarrow{P'_{j+1}, R'_{j+1}}_E \ldots \xrightarrow{P_h, R_h}_E G'_h$, two cases may occur:

1. if $R_k \not\subseteq R'_j \swarrow_l$, then substituting $prune(\hat{\pi}, R'_j)$ for $\hat{\pi}$ in π^*, by Lemma 10, the desired sequence of refinement steps is obtained;

2. if $R_k \subseteq R'_j \swarrow_l$, then the subsequence $G_0 \xrightarrow{*}_E G'_j$ of π' is the desired sequence. Indeed, by idempotence of $\swarrow_l$, $R_k \subseteq R'_j \swarrow_l$ implies $R_k \swarrow_l \subseteq R'_j \swarrow_l$. Since $P'_j = P_k$, then also $P_k \cap R_k \swarrow_l \subseteq P'_j \cap R'_j \swarrow_l$. Therefore, $R \subseteq P_k \cap R_k \swarrow_l \subseteq P'_j \cap R'_j \swarrow_l$. Hence, R is an entry region of G'_j.

∎

An immediate consequence of the previous lemma is that for any entry region R there is a sequence π of refinement steps discovering R (i.e. with R entry region of $last(\pi)$) whose length is bounded by $|[\![E]\!]|$.

Now, the termination of the fixpoint procedure to compute RWA^{m} can be established.

Proof of Theorem 4 Notice that $[\![\overline{V}]\!]$ and $[\![U]\!]$ are finite sets of convex polyhedra, therefore so is the number of initial entry regions from the convex polyhedra of $[\![\overline{V}]\!]$ to $[\![U]\!]$. At each iteration, the fixpoint procedure of Theorem 3 applies the refinement steps in a breadth-first manner, starting from these initial entry regions. Therefore, in every iteration each entry region discovered so far is employed in a refinement step. As a consequence of Lemma 11, taking $E = \overline{V}$ and $G_0 = U$, for every entry region there is a sequence of refinement steps which discoverd it and whose length is bounded by $|[\![\overline{V}]\!]|$. Therefore, by Lemma 9, after at most $|[\![\overline{V}]\!]|$ iterations of the procedure all the entry regions have been discovered, and the fixpoint is reached at the next iteration. ∎

4.7 Exact Computation of Pre-Flow

As seen in the previous section, one of the basic operations on polyhedra that are needed to compute RWA^{m} is the pre-flow operator $\swarrow$. It is sufficient to compute $P \swarrow F$ for convex P and F, for two reasons:

1. For a given location l and a convex polyhedron $Flow(l)$, the polyhedron F is always defined as $F = Flow(l)$.

2. The pre-flow of a general polyhedron is the union of the pre-flows of its convex polyhedra, namely, $(P_1 \cup P_2)\swarrow F = P_1\swarrow F \cup P_2\swarrow F$.

The pre-flow of P w.r.t. F is equivalent to the *post-flow* of P w.r.t. $-F$, defined as:

$$P\nearrow -F = \{x + \delta \cdot y \mid x \in P, y \in -F, \delta \geq 0\}.$$

The post-flow operation coincides with the *time-elapse* operation introduced in [HPR97] for topologically closed convex polyhedra. Notice that for convex polyhedra P and F, the post-flow of P w.r.t. F may not be a convex polyhedron: following [ABD+00], let $P \subseteq \mathbb{R}^2$ be the polyhedron containing only the origin $(0, 0)$ and let F be defined by the constraint $y > 0$, then the post-flow $P\nearrow F = \{(0, 0)\} \cup \{(x, y) \in \mathbb{R}^2 \mid y > 0\}$ is not a convex polyhedron (although it is a convex subset of $\mathbb{R}^2$). The Parma Polyhedral Library (PPL, see [BHZ08]), for instance, only provides an over-approximation of the post-flow operator, that we denote by $\nearrow_{\text{PPL}}$. Precisely, $P \nearrow_{\text{PPL}} F$ is the smallest convex polyhedron containing $P\nearrow F$.

On the other hand, the post-flow of a convex polyhedron is always the union of two convex polyhedra, according to the equation

$$P\nearrow F = P \cup \big(P \nearrow_{>0} F\big),$$

where $P \nearrow_{>0} F$ is the *positive post-flow* of P, i.e., the set of valuations that can be reached from P via a straight line of non-zero length whose slope belongs to F. Formally,

$$P \nearrow_{>0} F = \{x + \delta \cdot y \mid x \in P, y \in F, \delta > 0\}.$$

Hence, in order to exactly compute the post-flow of a convex polyhedron, the exact computation of the positive post-flow is needed.

Convex polyhedra admit two finite representations, in terms of *constraints* or *generators*. Libraries like PPL maintain both representations for each convex polyhedron and efficient algorithms exist for keeping them synchronized [Che68, Ver92]. The constraint representation refers to the set of linear inequalities whose solutions are the points of the polyhedron. The generator representation consists in three finite sets of *points*, *closure points*, and *rays*, that generate all points in the polyhedron by linear combination. More precisely, for each convex polyhedron $P \subseteq \mathbb{R}^n$ there exists a triple (V, C, R) such that V, C, and R are finite sets of points in $\mathbb{R}^n$, and $x \in P$ if and only if it can be written as

$$\sum_{v \in V} \alpha_v \cdot v + \sum_{c \in C} \beta_c \cdot c + \sum_{r \in R} \gamma_r \cdot r, \tag{4.11}$$

where all coefficients α_v, β_c and γ_r are non-negative reals, $\sum_{v\in V}\alpha_v+\sum_{c\in C}\beta_c = 1$, and there exists $v \in V$ such that $\alpha_v > 0$. The triple (V, C, R) is called a *generator* for P.
Intuitively, the elements of V are the proper vertices of the polyhedron P, the elements of C are vertices of the topological closure of P that do not belong to P, and each element of R represents a direction of unboundedness of P.

The following result shows how to efficiently compute the positive post-flow operator, using the generator representation.

Theorem 5. *Given two convex polyhedra P and F, let (V_P, C_P, R_P) be a generator for P and (V_F, C_F, R_F) a generator for F. The triple $(V_P \oplus V_F, C_P \cup V_P, R_P \cup V_F \cup C_F \cup R_F)$ is a generator for $P \nearrow_{>0} F$, where $\oplus$ denotes Minkowski sum.*

Proof. The first step of the proof is to show that, let $z \in P \nearrow_{>0} F$, there are coefficients α_v, β_c and γ_r such that z can be written as (4.11), for $V = V_P \oplus V_F$, $C = C_P \cup V_P$, and $R = R_P \cup V_F \cup C_F \cup R_F$.

By definition, there exist $x \in P$, $y \in F$, and $\delta > 0$ such that $z = x + \delta y$. Hence, there are coefficients α_v^x, β_c^x, and γ_r^x witnessing the fact that $x \in P$, and coefficients α_v^y, β_c^y, and γ_r^y witnessing the fact that $y \in F$. Moreover, there is $i \in V_P$ and $j \in V_F$ such that $\alpha_i^x > 0$ and $\alpha_j^y > 0$. Let $\varepsilon = \min\{\alpha_i^x, \delta\alpha_j^y\}$ and notice that $\varepsilon > 0$. It holds

$$\begin{aligned}\alpha_i^x \cdot i + \delta \cdot \alpha_j^y \cdot j &= (\alpha_i^x - \varepsilon)i + \varepsilon i + (\delta \cdot \alpha_j^y - \varepsilon)j + \varepsilon j = \\ &= \varepsilon(i + j) + (\alpha_i^x - \varepsilon)i + (\delta \cdot \alpha_j^y - \varepsilon)j.\end{aligned}$$

Hence,

$$\begin{aligned}z = &\sum_{v\in V_P} \alpha_v^x \cdot v + \sum_{c\in C_P} \beta_c^x \cdot c + \sum_{r\in R_P} \gamma_r^x \cdot r + \\ &+ \delta\left(\sum_{v\in V_F} \alpha_v^y \cdot v + \sum_{c\in C_F} \beta_c^y \cdot c + \sum_{r\in R_F} \gamma_r^y \cdot r\right) \\ = \varepsilon(i + j) + &\left((\alpha_i^x - \varepsilon)i + \sum_{v\in V_P\setminus\{i\}} \alpha_v^x \cdot v + \sum_{c\in C_P} \beta_c^x \cdot c\right) + \\ &\left((\delta \cdot \alpha_j^y - \varepsilon)j + \sum_{r\in R_P} \gamma_r^x \cdot r + \sum_{v\in V_F\setminus\{j\}} \alpha_v^y \cdot v + \sum_{c\in C_F} \beta_c^y \cdot c + \sum_{r\in R_F} \gamma_r^y \cdot r\right).\end{aligned}$$

One can easily verify that: *(i)* all coefficients are non-negative; *(ii)* the sum of the coefficients of the points in V and C is 1; *(iii)* there exists a point in V, namely $i + j$, such that its coefficient is strictly positive.

Conversely, let z be a point that can be expressed as (4.11), for $V = V_P \oplus V_F$, $C = C_P \cup V_P$, and $R = R_P \cup V_F \cup C_F \cup R_F$. We prove that $z \in P \nearrow_{>0} F$ by identifying $x \in P$, $y \in F$ and $\delta > 0$ such that $z = x + \delta y$.
Notice that *(a)* $\sum_{v \in V_P \oplus V_F} \alpha_v + \sum_{c \in C_P \cup V_P} \beta_c = 1$, and *(b)* there exists $v^* \in V_P \oplus V_F$ such that $\alpha_{v^*} > 0$. Let

$$x = \sum_{\substack{v_1 \in V_P \\ v_2 \in V_F}} \alpha_{v_1+v_2} \cdot v_1 + \sum_{c \in C_P \cup V_P} \beta_c \cdot c + \sum_{r \in R_P} \gamma_r \cdot r.$$

The point x is claimed to belong to P ($x \in P$): first, x is expressed as a linear combination of points in (V_P, C_P, R_P); second, all coefficients are non-negative; third, the sum of the coefficients of the points in V_P and in C_P is 1, due to *(a)* above; finally, since $\alpha_{v^*} > 0$, there is a point in V_P whose coefficient is positive. Then, let

$$\delta = \sum_{v \in V_P \oplus V_F} \alpha_v + \sum_{r \in V_F \cup C_F} \gamma_r, \qquad \text{and}$$

$$y = \frac{1}{\delta} \cdot \left(\sum_{\substack{v_1 \in V_P \\ v_2 \in V_F}} \alpha_{v_1+v_2} \cdot v_2 + \sum_{r \in V_F \cup C_F \cup R_F} \gamma_r \cdot r \right).$$

Since $\alpha_{v^*} > 0$, we have $\delta > 0$. The point y is claimed to belong to F ($y \in F$): first, y is a linear combination of points in (V_F, C_F, R_F); second, all coefficients are non-negative; third, the sum of the coefficients of the points in V_F and in C_F is 1, due to our choice of δ; finally, since $\alpha_{v^*} > 0$, there is a point in V_F whose coefficient is positive. ∎

Now the discussion about the safety control problem for linear hybrid games is complete. The implementation of the operators introduced in this chapter is the focus of Chapter 6.

Chapter 5

Solving the Reachability Control Problem for LHGs

This chapter is focused on the solution of the reachability control problem for linear hybrid games. The reachability goal is the the objective to lead the system in a given set of the so-called "target" states T, regardless of the evolution of the continuous variables and the uncontrollable transitions, i.e. regardless the behavior of the environment. The problem is known to be undecidable, being almost harder than the standard reachability verification (i.e., 1-player reachability) for triangular hybrid automata [HKPV95], that is a special case of LHGs.

Here, a sound and complete semi-algorithm [1] for the problem is proposed. This procedure is a fixpoint similar to that seen for the safety goal but, unlike discrete and real-time cases, the controllable predecessor for reachability is now different from $CPre^{\mathrm{S}}$: the core of the former is based on a novel algorithm for computing, within a given location, the set of states that must reach a given polyhedral region while avoiding another one, the RWA^{M} operator, while the core of the latter is the RWA^{m} operator. Along the way the relationship between the two operators will be clear. The operator RWA^{M} takes as input two sets of states U and V, and computes the set of points from which all the trajectories leads to the region U, while avoiding the region V. Hence, the reachability control problem has a proper version of the controllable predecessor, called $CPre^{\mathrm{R}}$, which is different from the one proposed for the safety control problem. The reason for this lies in the fact that the hybrid game model is asymmetric: the environment may govern also the evolution of the continuous variables, besides the fact that it can choose proper transitions. As the conseguence of

[1] In other words, a procedure that may or may not terminate, and that provides the correct answer whenever it terminates.

this game asymmetry, although the reachability goal (as a language of infinite traces) is the dual of safety, the corresponding synthesis problems are not dual. Hence, it is not possible to solve the control problem with reachability goal T by exchanging the roles of the two players and then solving the safety control problem with goal $\overline{T}$ (i.e., the complement of T).

In work to date, the reachability control problem was never considered for the class of linear hybrid games. Hence, the procedure shown here seems to be the first known solution for the safety control problem for LHGs.

The computation of the RWA^{M} operator is not trivial, and requires the introduction of some additional operator on the polyhedra. This chapter shows these operator and their properties, in order to implement the RWA^{M} operator.

5.1 The Global Semi-Algorithm

In the introduction of the chapter, it was said that the controllable predecessor operator defined for the reachability goal, is different from the one defined for the safety goal. The *controllable predecessor operator for reachability* $CPre^{\mathrm{R}}$ is formally defined in the following section. The following theorem states the general procedure for solving the reachability control problem, based on the controllable predecessor operator for reachability $CPre^{\mathrm{R}}(\cdot)$.

Theorem 6. *The answer to the reachability control problem for target set* $T \subseteq InvS$ *is positive if and only if*

$$InitS \subseteq \mu W \,.\, T \cup CPre^{\mathrm{R}}(W). \tag{5.1}$$

Controllable predecessor operator for reachabilty. Now the controllable predecessor operator for reachability can be formally defined. For a set of states A, the operator $CPre^{\mathrm{R}}(A)$ returns the set of states from which the controller can ensure that the system reaches A within the next joint step. Based on the activity chosen by the environment, this may happen for three reasons:

1. at some point during the activity a controllable transition is enabled that leads into A, and all uncontrollable transitions enabled in the meanwhile also lead to A;

2. the activity naturally enters A, and all uncontrollable transitions enabled in the meanwhile also lead to A;

3. the activity eventually leaves the invariant, and all uncontrollable transitions that are ever enabled along the activity lead to A.

Notice that in case *(3)* the system is forced to reach A because, by well-formedness, an uncontrollable transition must be enabled before the activity leaves the invariant.

The three cases can be formalized as the following predicates Φ_i, on an activity f, location l, and target set A. For a set of states A and $x \in \{\mathrm{u}, \mathrm{c}\}$, let $Pre_x(A)$ be the set of states in $InvS$ where some discrete transition belonging to Edg_x is enabled, which leads to A.

$$\Phi_1(f,l,A) = \exists \delta \in span(f,l) : \langle l, f(\delta)\rangle \in Pre_c(A) \text{ and } \forall 0 \le \delta' \le \delta : \langle l, f(\delta')\rangle \notin Pre_u(\overline{A})$$

$$\Phi_2(f,l,A) = \exists \delta \in span(f,l) : \langle l, f(\delta)\rangle \in A \text{ and } \forall 0 \le \delta' < \delta : \langle l, f(\delta')\rangle \notin Pre_u(\overline{A})$$

$$\Phi_3(f,l,A) = \infty \notin span(f,l) \text{ and } \forall \delta \in span(f,l) : \langle l, f(\delta)\rangle \notin Pre_u(\overline{A})$$

Now the controllable predecessor for reachability is formally defined, as the following:

$$CPre^{\mathrm{R}}(A) = \Big\{ \langle l,u\rangle \in InvS \;\Big|\; \forall f \in Adm(\langle l,u\rangle) : \Phi_1(f,l,A) \text{ or } \Phi_2(f,l,A) \text{ or } \Phi_3(f,l,A) \Big\}.$$

In discrete games (see Chapter 1), the *CPre* operator used for solving reachability games is the same as the one used for the safety goal [Mal02]. In both cases, when the operator is applied to a set of states T, it returns the set of states from which the controller can force the game into T in one step. In hybrid games, the situation is different: a joint step represents a possibly complex behavior, extending over a (possibly) non-zero time interval. While the $CPre^{\mathrm{R}}$ operator for reachability only requires T to be visited once during such interval, *CPre* for safety requires that the entire behavior constantly remains in T. Hence, a novel algorithm for computing $CPre^{\mathrm{R}}$ is presented in Section 5.2.

Clearly, as first step, the Theorem 6 need to be proved, as follows.

Proof. [***if***] Assume equation 1.2 holds, a winning strategy is built in two steps. Let

- $W_0 = T$,
- $W_\alpha = T \cup CPre^{\mathrm{R}}(W_{\alpha-1})$, for a successor ordinal α, and

- $W_\alpha = \bigcup_{\beta<\alpha} W_\beta$ for a limit ordinal α.

Moreover, let $W^* = \mu W . T \cup \mathit{CPre}^{\mathrm{R}}(W)$. By Knaster-Tarski theorem, if $s \in W^*$ then there exists a least ordinal α such that $s \in W_\alpha$. If α is a limit ordinal, by definition of W_α there exists $\beta < \alpha$ such that $s \in W_\beta$, which contradicts minimality of α. Hence α is either 0 or a successor ordinal.

Let σ be a strategy defined as follows, for all states s:

- $\bot \in \sigma(s)$ and
- for all $s \in W^*$, let $\alpha = \beta + 1$ be the smallest ordinal such that $s \in W_\alpha$; for all $e \in \mathit{Edg}_{\mathrm{c}}$, we have $e \in \sigma(s)$ if and only if $s \xrightarrow{e} s'$ and $s' \in W_\beta$.

While σ is clearly a strategy, it is not necessarily a winning strategy, as it may admit consistent runs which delay a controllable action either beyond the winning set W^* or beyond its availability. However, a winning strategy can be recovered by removing the null action $\bot$ from certain states. Let σ' be any strategy which coincides with σ on all the states, except for the states $s \in W^*$ with $\sigma(s) \cap \mathit{Edg}_{\mathrm{c}} \neq \emptyset$, where it satisfies $\sigma'(s) \cap \mathit{Edg}_{\mathrm{c}} = \sigma(s) \cap \mathit{Edg}_{\mathrm{c}}$ and the following two conditions (a) and (b). For all $f \in \mathit{Adm}(s)$, let $D_{f,s} = \{\delta > 0 \mid \forall 0 \leq \delta' \leq \delta : \langle \mathit{loc}(s), f(\delta') \rangle \in W_\alpha \text{ and } \sigma(\langle \mathit{loc}(s), f(\delta') \rangle) \cap \mathit{Edg}_{\mathrm{c}} \neq \emptyset\}$:

(a) If there is $f \in \mathit{Adm}(s)$ such that $D_{f,s} = \emptyset$ then $\bot \notin \sigma'(s)$;

(b) For all $f \in \mathit{Adm}(s)$, if $D_{f,s} \neq \emptyset$ then there exists $\delta \in D_{f,s}$ such that $\bot \notin \sigma'(\langle \mathit{loc}(s), f(\delta) \rangle)$ and $\bot \in \sigma'(\langle \mathit{loc}(s), f(\delta') \rangle)$ for all $0 \leq \delta' < \delta$.

Intuitively, the new strategy σ' ensures that following any activity from a state $s \in W^*$ in which some controllable action is enabled, such an action will always be taken before none of them is available and before leaving W^*.

Showing that for every $s \in \mathit{InitS}$ and every $r \in \mathit{Runs}(\sigma', s)$, it holds that $\mathit{States}(r) \cap T \neq \emptyset$, allowing us to prove that σ' is winning.

In particular, the proof proceeds by transfinite induction on the least ordinal α such that $s \in W_\alpha$ and show that by following the strategy σ' the set T is reached from s within a finite number of joint steps. The statement is trivially true for $\alpha = 0$. Since α cannot be a limit ordinal, the only remaining case is if α is a successor ordinal. Let $s \xrightarrow{\delta, f} s' \xrightarrow{e} s''$ be a joint step starting from s and consistent with σ' (steps of infinite length are discussed later). If $e \in \mathit{Edg}_{\mathrm{c}}$, by construction we have $s'' \in W_{\alpha-1}$, and the thesis follows from the induction hypothesis. Otherwise, $e \in \mathit{Edg}_{\mathrm{u}}$ and $\bot \in \sigma'(\langle l, f(\delta') \rangle)$ for all $0 \leq \delta' < \delta$. Now, it is necessary to prove that either $s'' \in W_{\alpha-1}$ or there is $\delta' \leq \delta$ such that $\langle l, f(\delta') \rangle \in W_{\alpha-1}$. Assume that $s'' \notin W_{\alpha-1}$, i.e., $s' \in \mathit{Pre}_u(\overline{W_{\alpha-1}})$, and hence $s' \notin W_\alpha$. Since $s \in \mathit{CPre}^{\mathrm{R}}(W_{\alpha-1})$, one of the following holds:

1. If $\Phi_1(f, l, W_{\alpha-1})$ holds, the following contradiction are obtained: Considering the current assumption, we have that there is $\delta^* < \delta$, with $s^* = \langle l, f(\delta^*)\rangle \in Pre_c(W_{\alpha-1})$, and, for all $0 \leq \delta' \leq \delta^*$, $\langle l, f(\delta')\rangle \notin Pre_u(\overline{W_{\alpha-1}})$. As a consequence, $s^* \in CPre^{\mathrm{R}}(W_{\alpha-1})$ and, therefore, $s^* \in W_\alpha$. Notice that s^* is an intermediate point along the activity f between s and s', and it holds $\bot \in \sigma'(s^*)$. Let f^* be the suffix of f starting from s^* (i.e., $f^*(\gamma) = f(\gamma + \delta^*)$), and consider the set D_{f^*,s^*} defined in rule (b) in the above construction of σ'. If $D_{f^*,s^*} = \emptyset$, rule (a) in the construction of σ' applies, and we obtain the contradiction that $\bot \notin \sigma'(s^*)$. Otherwise, rule (b) applies, and there exists $\delta' \in D_{f^*,s^*}$ such that $\bot \notin \sigma'(\langle l, f^*(\delta')\rangle) = \sigma'(\langle l, f(\delta' + \delta^*)\rangle)$. By definition of D_{f^*,s^*}, it holds $\delta' + \delta^* < \delta$, because $\langle l, f(\delta)\rangle \notin W_\alpha$. Hence, it also holds $\bot \in \sigma'(\langle l, f(\delta'+\delta^*)\rangle)$, a contradiction.

2. If $\Phi_2(f, l, W_{\alpha-1})$ holds, by definition there is $\delta^* < \delta$ such that $\langle l, f(\delta^*)\rangle \in W_{\alpha-1}$ and for all $0 \leq \delta' < \delta^*$, $\langle l, f(\delta')\rangle \notin Pre_u(\overline{W_{\alpha-1}})$. Since $s' = \langle l, f(\delta)\rangle \in Pre_u(\overline{W_{\alpha-1}})$, we have $\delta^* \leq \delta$, which proves the claim.

3. If $\Phi_3(f, l, W_{\alpha-1})$ holds, an immediate contradiction is obtained: by definition, $\langle l, f(\delta')\rangle \notin Pre_u(\overline{W_{\alpha-1}})$ for all $\delta' \in span(f, l)$, contradicting the fact that $s' = \langle l, f(\delta)\rangle \in Pre_u(\overline{W_{\alpha-1}})$.

Finally, consider the case of an infinite step $s \xrightarrow{\infty,f}$. Clearly, since $\infty \in span(f, l)$ it cannot be $\Phi_3(f, l, W_{\alpha-1})$. If it is assumed that $\Phi_1(f, l, W_{\alpha-1})$, a contradiction is obtained by following a similar argument to case 1 above. Assuming $\Phi_2(f, l, W_{\alpha-1})$, it follows immediately that $W_{\alpha-1}$ is eventually reached along f.

In any of the above cases, the induction hypothesis ensures that after a finite number of joint transitions T is reached from $s \in W_\alpha$.

[***only if***] Let $s \notin W^*$, It will be proved that prove that for all strategies there is a run that starts in s, is consistent with the strategy and remains in $\overline{W^*}$ indefinitely. By induction, it is sufficient to show that there is a joint step starting from s and entirely contained in $\overline{W^*}$. Since $W^* = CPre^{\mathrm{R}}(W^*)$, according to the definition of $CPre^{\mathrm{R}}$, there exists an activity $f \in Adm(s)$ such that $\Phi_i(f, l, W^*)$ is violated for all $i \in \{1, 2, 3\}$, where $l = loc(s)$. By complementing Φ_2 we have that for all $\delta \geq 0$ either $\langle l, f(\delta)\rangle \in \overline{W^*}$ or there exists $\delta' < \delta$ such that $\langle l, f(\delta')\rangle \in Pre_u(\overline{W^*})$. Hence, if f eventually reaches W^*, earlier than that an uncontrollable transition is enabled which leads to $\overline{W^*}$.

Consider the behavior of an arbitrary strategy σ along f. First, assume that $\bot \in \sigma(f(\delta))$, for all $\delta \in span(f, l)$, i.e., the strategy may not take any controllable transition throughout f. If $\infty \in span(f, l)$ and f never reaches W^*, the infinite step $s \xrightarrow{\infty,f}$ witnesses the claim. If $\infty \in span(f, l)$ and f eventually

reaches W^* at time δ, the environment takes an uncontrollable transition leading to $\overline{W^*}$ before δ. If $\infty \notin span(f,l)$, by complementing Φ_3 we have that even if f does not reach W^* there exists $\delta \in span(f,l)$ such that $\langle l, f(\delta)\rangle \in Pre_u(\overline{W^*})$. The thesis follows as before.

Finally, it remains the case that there exists a time $\delta \in span(f,l)$ where $\bot \notin \sigma(f(\delta))$. Hence, σ prescribes at least one controllable transition in δ. If $\langle l, f(\delta)\rangle \notin Pre_c(W^*)$, the environment allows the controller to take the desired transition, and the claim is proved. Otherwise, by complementing Φ_1 we obtain that an uncontrollable transition leading to $\overline{W^*}$ is enabled at a time $\delta' \leq \delta$. By taking this transition, the witnessing joint step is obtained. ∎

5.2 Computing the Predecessor Operator for Reachability

Similarly to the safety problem, where the *CPre* operator is based on the computation of the RWA^{m} operator, the computation of the controllable predecessor for reachability, lie in the *Must Reach While Avoiding* operator, denoted by RWA^{M}. Given a location l and two sets of variable valuations U and V, $RWA^{\mathrm{M}}_l(U,V)$ contains the set of valuations from which all continuous trajectories of the system reach U while avoiding V [2]. The RWA^{M} operator is formally, as follows:

$$RWA^{\mathrm{M}}_l(U,V) = \Big\{ u \in Val(X) \;\Big|\; \forall f \in Adm(\langle l,u\rangle)\, \exists \delta \geq 0 : \\ f(\delta) \in U \text{ and } \forall\, 0 \leq \delta' \leq \delta : f(\delta') \notin V \Big\}. \quad (5.2)$$

By rephrasing the definition of $CPre^{\mathrm{R}}$, one can observe that $s = \langle l,u\rangle \in CPre^{\mathrm{R}}(A)$ iff all activities f starting from u reach a set of "good" points while avoiding a set of "bad" points. Good points include $C_l = Pre_c(A)\downarrow_l$ according to $\Phi_1(f,l,A)$, $A\downarrow_l$ according to $\Phi_2(f,l,A)$, and $\overline{Inv(l)}$ according to $\Phi_3(f,l,A)$. As to the bad points, all predicates Φ_i require that the activity avoids $B_l = Pre_u(\overline{A})\downarrow_l$, with subtle distinctions at the instant when a good point is reached. According to Φ_1, B_l must be avoided also in that instant (when C_l is reached), while Φ_2 permits the activity f to reach B_l at the same time as $A\downarrow_l$. Since satisfaction of one Φ_i is enough for an activity to comply with the requirements of $CPre^{\mathrm{R}}$, the least restrictive avoidance condition prevails, namely, $B_l \setminus A\downarrow_l$. The following lemma formalizes the above argument, recalling the *polyhedral* definition introduced in the last chapter.

[2] In ATL notation [AHK97], we have $RWA^{\mathrm{M}}(U,V) \equiv \langle\!\langle ctr\rangle\!\rangle \overline{V}\, \mathcal{U}\, (U \wedge \overline{V})$, where *ctr* is the player representing the controller.

Lemma 12. *For all polyhedral sets of states $A \subseteq InvS$, the following holds:*

$$CPre^{\mathrm{R}}(A) = InvS \cap \bigcup_{l \in Loc} \{l\} \times RWA_l^{\mathrm{M}}\big(A\lfloor_l \cup C_l \cup \overline{Inv(l)}, B_l \setminus A\lfloor_l\big),$$

where $B_l = Pre_u(\overline{A})\lfloor_l$ and $C_l = Pre_c(A)\lfloor_l$.

Proof. [$\subseteq$] Let $s = \langle l, u\rangle \in CPre^{\mathrm{R}}(A)$ and let $f \in Adm(s)$. If $\Phi_1(f, l, A)$ holds, there is $\delta \in span(f, l)$ such that $f(\delta) \in C_l$ and for all $0 \leq \delta' \leq \delta$ it holds $f(\delta') \notin B_l$ and hence $f(\delta') \notin B_l \setminus A\lfloor_l$, satisfying the requirements of (5.2).
If $\Phi_2(f, l, A)$ holds, there is $\delta \in span(f, l)$ such that $f(\delta) \in A\lfloor_l$ and for all $0 \leq \delta' < \delta$ it holds $f(\delta') \notin B_l$. Since $f(\delta) \notin B_l \setminus A\lfloor_l$, the requirements of (5.2) are satisfied again.
Finally, if $\Phi_3(f, l, A)$ holds, we have $\infty \notin span(f, l)$ and $\langle l, f(\delta)\rangle \notin B_l$ for all $\delta \in span(f, l)$. Pick a time δ^* when f has left $Inv(l)$ and it has never re-entered it. Formally, we have $\delta^* \notin span(f, l)$, $f(\delta^*) \notin Inv(l)$, and $f(\delta) \in span(f, l) \cup \overline{Inv(l)}$ for all $\delta \leq \delta^*$. We obtain $f(\delta^*) \in \overline{Inv(l)}$ and $f(\delta) \notin B_l$ for all $0 \leq \delta \leq \delta^*$, satisfying (5.2) once again.

[$\supseteq$] Let $l \in Loc$ and $u \in RWA_l^{\mathrm{M}}(A\lfloor_l \cup C_l \cup \overline{Inv(l)}, B_l \setminus A\lfloor_l)$. For all $f \in Adm(\langle l, u\rangle)$, let D_f be the set of all $\delta \geq 0$ such that $f(\delta) \in A\lfloor_l \cup C_l \cup \overline{Inv(l)}$ and for all $0 \leq \delta' \leq \delta$ it holds $f(\delta') \notin B_l \setminus A\lfloor_l$. By definition of RWA_l^{M}, we have $D_f \neq \emptyset$. Let $\delta^* = \inf D_f$ and assume for simplicity that $\delta^* \in D_f$, as the other case can be treated similarly.

For all $0 \leq \delta' < \delta^*$ we have both $f(\delta') \in (\overline{A\lfloor_l} \cap \overline{C_l} \cap Inv(l))$ since $\delta' \notin D_f$, and $f(\delta') \in (\overline{B_l} \cup A\lfloor_l)$ since $\delta' < \delta^*$ and $\delta^* \in D_f$. Moreover, $(\overline{A\lfloor_l} \cap \overline{C_l} \cap Inv(l)) \cap (\overline{B_l} \cup A\lfloor_l) = \overline{B_l} \cap \overline{A\lfloor_l} \cap \overline{C_l} \cap Inv(l)$, and we can conclude that $f(\delta') \in \overline{B_l} \cap \overline{A\lfloor_l} \cap \overline{C_l} \cap Inv(l)$. If $f(\delta^*) \in A\lfloor_l$, we have $\delta^* \in span(f, l)$ and $\Phi_2(f, l, A)$. If $f(\delta^*) \in C_l$, we have $\delta^* \in span(f, l)$ again and $\Phi_1(f, l, A)$. Finally, if $f(\delta^*) \in \overline{Inv(l)}$ we have $\Phi_3(f, l, A)$. Therefore, it holds $\langle l, u\rangle \in CPre^{\mathrm{R}}(A)$. ∎

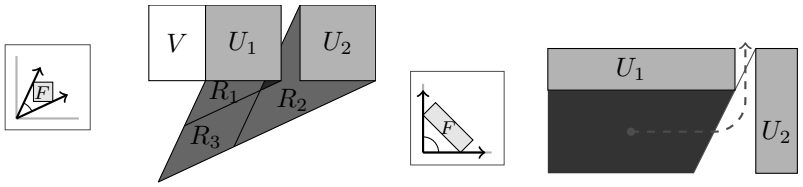
5.3 The Local Algorithm

Lemma 12 reduces the solution of the reachability control problem to the computation of the operator RWA^{M}. In order to show the correct implementation of RWA^{M}, let l be a fixed location, Example 14 shows some basic properties of RWA^{M}.

Example 14. *As witnessed by Figure 5.1(a), the first argument of RWA^{M} does not distribute over union, in other words $RWA_l^{\mathrm{M}}(U_1 \cup U_2, V) \neq RWA_l^{\mathrm{M}}(U_1, V) \cup RWA_l^{\mathrm{M}}(U_2, V)$. In particular, in Figure 5.1(a) we have $RWA_l^{\mathrm{M}}(U_1, V) = U_1 \cup R_1$, $RWA_l^{\mathrm{M}}(U_2, V) = U_2 \cup R_2$, and $RWA_l^{\mathrm{M}}(U_1 \cup U_2, V) = U_1 \cup U_2 \cup R_1 \cup R_2 \cup R_3$.*

Hence, computing $RWA_l^{\mathrm{M}}(U, V)$ *for convex* U *(a relatively simple task) does not extend to general polyhedra.*

Additionally, it is not possible to restrict the analysis from arbitrary activities (i.e., any differentiable function which stays in the invariant and whose slope belongs to $Flow(l)$*) to straight-line activities. In Figure 5.1(b), the dotted area contains the set of points that must reach* $U_1 \cup U_2$ *following straight-line activities. On the other hand,* $RWA_l^{\mathrm{M}}(U_1 \cup U_2, \emptyset) = U_1 \cup U_2$*, because all other points (including those in the dotted area) can avoid* $U_1 \cup U_2$ *by passing through the gap between* U_1 *and* U_2*.*



(a) $RWA_l^{\mathrm{M}}(U_1 \cup U_2, V) \neq RWA_l^{\mathrm{M}}(U_1, V) \cup RWA_l^{\mathrm{M}}(U_2, V)$.

(b) Straight-line activities are not sufficient to avoid $U_1 \cup U_2$.

Figure 5.1: Basic properties of RWA^{M}. The boxes on the left represent the convex polyhedron $F = Flow(l)$ in the $(\dot{x}, \dot{y})$ plane. Thick arrows represent the *extremal directions* of flow.

The RWA^{M} operator is related to the *May Reach While Avoid* operator RWA^{m} used to solve the safety control problem, shown in the last chapter. In particular, it will be shown (see Theorem 7, that it is possible to compute RWA^{M} using the RWA^{m} operator. Recall briefly that, given the polyhedra U and V, $RWA^{\mathrm{m}}(U, V)$ returns the set of states from which *there exists* a trajectory that reaches U while avoiding V, i.e.

$$RWA_l^{\mathrm{m}}(U, V) = \Big\{ u \in Val(X) \,\Big|\, \exists f \in Adm(\langle l, u \rangle),\, \delta \geq 0 : f(\delta) \in U \text{ and } \forall\, 0 \leq \delta' < \delta : f(\delta') \in \overline{V} \cup U \Big\}.$$

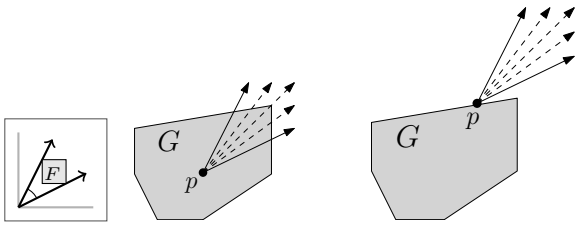
In safety control problems, RWA^{m} is used to compute the states from which the environment may reach an unsafe state (in U) while avoiding the states from which the controller can take a transition to a safe state (in V) (see Chapter 4). This is a classical operator in the literature, known under different names such as *Reach* [TLSS00], *Unavoid_Pre* [BBV+03], and *flow_avoid* [WT97].

Notice that RWA^{M} differs from RWA^{m} only on the quantification of the activity f and on the inequality $\delta' \leq \delta$, which is strict in RWA^{m}. The latter difference is connected to the fact that the controller must prevent "bad" uncon-

trollable transitions even if they occur at the same time as a "good" controllable transitions.

To show the relation between RWA^{m} and RWA^{M}, some additional notations will be introduced. Let $l \in Loc$ a fixed location. For a polyhedron G and $p \in G$, p is said *l-bounded in G* (resp., *l-thin in G*) if all admissible activities starting from p eventually (resp., immediately) exit from G. Formally, p is l-bounded if for all $f \in Adm(\langle l, p\rangle)$ there exists $\delta \geq 0$ such that $f(\delta) \notin G$; p is l-thin if for all $f \in Adm(\langle l, p\rangle)$ and all $\delta > 0$, it holds $f(\delta) \notin G$.

For example, considering the flow depicted in Figure 5.2(a), the point p shown in Figure 5.2(b) is l-bounded in G, while the point p shown in Figure 5.2(c) is l-thin in G. Notice that, the flow does not contain the origin: otherwise, by choosing the origin as activity, it would be possible to remain forever in G and then the point p in Figure 5.2(b) (resp., 5.2(c)), would be not l-bounded (resp., l-thin).



(a) Flow (b) p is l-bounded in G (c) p is l-thin in G

Figure 5.2: Definition of l-bounded and l-thin

The set of points of G that are l-bounded in it, is denoted by $bounded_l(G)$, and if all points $p \in G$ are l-bounded (resp., l-thin) in G, G is called to be *l-bounded* (resp., *l-thin*) in G.

Now, the following result that connects RWA^{M} to RWA^{m}, can be shown by exploiting the following idea. All points in $U \setminus V$ belong to $RWA^{\mathrm{M}}_l(U, V)$ by definition. Accordingly, let the set $Under = U \setminus V$ be an *under-approximation.*

The content of $RWA^{\mathrm{M}}_l(U, V)$ can be partitioned into two regions: the first region is *Under*; the second region must be l-bounded, because each point in the second region must eventually reach *Under*. If one can find a polyhedron *Over* that over-approximates $RWA^{\mathrm{M}}_l(U, V)$ and such that $Over \setminus Under$ is l-bounded, one can use RWA^{m} to refine it. Precisely, the operator RWA^{m} is used to identify and remove the points of *Over* that may leave *Over* without hitting U first.

If $Over \setminus Under$ was not l-bounded, the above technique would not work, because RWA^{m} cannot identify (and remove) the points that may remain forever in *Over* without ever reaching *Under*.

The following Theorem states the relation between RWA^{m} and RWA^{M}.

Theorem 7. *For all polyhedra U and V, let $Under = U \setminus V$ and let $Over$ be a polyhedron such that:* (i) $RWA_l^{\mathrm{M}}(U,V) \subseteq Over \subseteq \overline{V}$ *and* (ii) $Over \setminus Under$ *is l-bounded. Then,*

$$RWA_l^{\mathrm{M}}(U,V) = Over \setminus RWA_l^{\mathrm{m}}(\overline{Over}, U). \tag{5.3}$$

$\subseteq$. Let $u \in RWA_l^{\mathrm{M}}(U,V)$. By assumption *(i)*, it holds $u \in Over$. Now, the fact that $u \notin RWA_l^{\mathrm{m}}(\overline{Over}, U)$ will be proved. Assume the contrary; according to the definition of RWA_l^{m}, there exist an activity $f \in Adm(\langle l, u\rangle)$ and a delay $\delta \geq 0$ such that $f(\delta) \in \overline{Over}$ and $f(\delta') \in \overline{U} \cup \overline{Over}$ for all $0 \leq \delta' < \delta$. Since $\overline{Over} \subseteq \overline{RWA_l^{\mathrm{M}}(U,V)}$, the activity f leads from u to a point in $\overline{RWA_l^{\mathrm{M}}(U,V)}$, without passing through $Under$.

Let f' be an activity witnessing the fact that $f(\delta) \notin RWA_l^{\mathrm{M}}(U,V)$. If U is never reached by f before time δ, the activity obtained by starting with f and then switching to f' from time δ is a witness for $u \notin RWA_l^{\mathrm{M}}(U,V)$ (contradiction). If instead f reaches U at time $\delta' < \delta$, it also holds $f(\delta') \in V$. Then, let $D = \{\delta' \mid f(\delta') \in V\} \neq \emptyset$ and let $\delta^* = \inf D$. For all $\delta' < \delta^*$, it holds $f(\delta') \in \overline{U}$. If $\delta^* \in D$ then $f(\delta^*) \in V$, and f is a witness to the fact that $u \notin RWA_l^{\mathrm{M}}(U,V)$ (contradiction).

Finally, if $\delta^* \notin D$, let $\bar{\delta}$ be any time when f visits U. This time must be strictly greater than δ^*. By definition of δ^*, there exists another time between δ^* and $\bar{\delta}$ where f visits V, proving once again that $u \notin RWA_l^{\mathrm{M}}(U,V)$ (contradiction). We conclude that $u \notin RWA_l^{\mathrm{m}}(\overline{Over}, U)$, and the thesis.

$[\supseteq]$ Let $u \notin RWA_l^{\mathrm{M}}(U,V)$. It is immediate that $u \notin Under$. Now it will be proved that $u \notin Over \setminus RWA_l^{\mathrm{m}}(\overline{Over}, U)$. If $u \notin Over$, we are done. Hence, assume that $u \in Over$. Since $u \notin RWA_l^{\mathrm{M}}(U,V)$, there is an activity $f \in Adm(\langle l, u\rangle)$ such that for all $\delta \geq 0$ either *(a)* $f(\delta) \notin U$, or *(b)* there exists $\delta' \leq \delta$ such that $f(\delta') \in V$. Two cases can be identified:

- First, assume that the activity f never reaches U (and hence, $Under$). By assumption *(ii)*, there exists $\delta' \geq 0$ such that $f(\delta') \notin Over \setminus Under$. Since $f(\delta') \notin Under$, then $f(\delta') \notin Over$. As a consequence, it holds $u \in RWA_l^{\mathrm{m}}(\overline{Over}, U)$, and we are done.

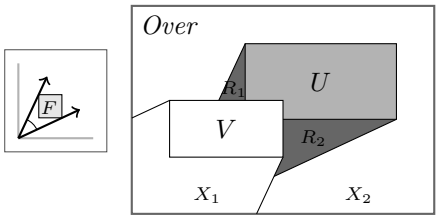
- Otherwise, let $D_U = \{\delta \geq 0 \mid f(\delta) \in U\} \neq \emptyset$ and $\delta_U = \inf D_U$. There can be two cases: first assume $\delta_U \in D_U$; by *(b)* there exists $\delta' \leq \delta_U$ with $f(\delta') \in V$. This implies that f reaches V (and hence $\overline{Over}$) at time δ' while remaining in $\overline{U}$ up until δ' (included). As a consequence, $u \in RWA_l^{\mathrm{m}}(\overline{Over}, U)$ and we are done.

Next, assume $\delta_U \notin D_U$. Let $D_V = \{\delta \mid f(\delta) \in V\}$. We have $D_V \neq \emptyset$ due to $D_U \neq \emptyset$ and property *(b)* above. Let $\delta_V = \inf D_V$. If $\delta_V < \delta_U$, there exists a time between δ_V and δ_U when f reaches V (and hence $\overline{Over}$). Since f remains in $\overline{U}$ until δ_U, then $u \in RWA_l^{\mathrm{m}}(\overline{Over}, U)$.

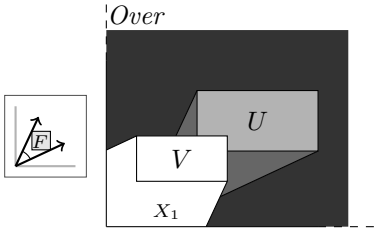
Otherwise, $\delta_V \geq \delta_U$. For all δ' such that $f(\delta') \in U$, $\delta_V \leq \delta'$ by *(b)* and the fact that $\delta_V = \inf D_V$. As a consequence, since all possible intermediate points between δ_U and δ_V cannot belong to U, and $\delta_U = \inf D_U$, no such point exists, i.e., $\delta_V = \delta_U$.

Now, if $\delta_V \in D_V$, then it immediately follows that $u \in RWA_l^{\mathrm{m}}(\overline{Over}, U)$. Otherwise, there are elements of D_V arbitrarily close to δ_V. Since V is a polyhedron and f is differentiable, there exists $\delta' > \delta_V$ such that $f(\delta) \in V \subseteq \overline{Over}$ for all $\delta_V < \delta \leq \delta'$. Therefore, at all times up to δ' (included), f remains in $\overline{U} \cup \overline{Over}$, once again it holds that $u \in RWA_l^{\mathrm{m}}(\overline{Over}, U)$.

∎



(a) Illustrating Theorem 7.



(b) When $Over \setminus Under$ is not l-bounded, Equation (5.3) fails.

Figure 5.3: Relationship between RWA^{M} and RWA^{m}.

Example 15. *An example of the application of Theorem 7 is depicted in Figure 5.3(a), where U and V are the gray boxes and Over is the outer box, excluding V. The set $RWA_l^{\mathrm{m}}(\overline{Over}, U)$ can be divided in two areas: area X_1 contains*

the points that may reach V (which is a part of $\overline{Over}$) while avoiding U, and area X_2 contains the points that may exit Over through its top and right sides. Following Equation 5.3, X_1 and X_2 are removed from Over. The remaining regions are $U \setminus V$ and the two regions R_1 and R_2, whose points are forced to enter U while avoiding V, as requested by $RWA_l^{\mathrm{M}}(U,V)$. On the other hand, Figure 5.3(b) presents a case in which Over is not l-bounded, thus violating one of the conditions of Theorem 7. The set $RWA_l^{\mathrm{m}}(\overline{Over}, U)$ comprises only the region X_1. Hence, the dotted area, which does not belong to $RWA_l^{\mathrm{M}}(U,V)$, is not removed from Over, because those points cannot exit from Over.

5.4 Computing a Suitable Over-Approximation

According to Theorem 7, in order to compute RWA^{M} operator by using RWA^{m}, it is necessary to compute a polyhedron *Over* satisfying certain assumptions. First to show how such a polyhedron can be computed, some preliminary notions will be introduced.

Given a polyhedron G and a convex polyhedron F, the *positive pre-flow* operator $G_{\swarrow_{>0}} F$ is defined are

$$G_{\swarrow_{>0}} F = \{u - \delta c \mid u \in G, c \in F, \delta > 0\}.$$

Intuitively, $G_{\swarrow_{>0}} F$ contains the points that may reach G via a straight trajectory of non-zero length whose slope is in F. The abbreviation $G_{\swarrow_{>0}}$ is used to denote $G_{\swarrow_{>0}\, Flow(l)}$.

For a (not necessarily convex) polyhedron G and a convex polyhedron F, we say that G is *bounded w.r.t.* F if for all $p \in G$ and all $c \in F$ there exists a constant $\delta \geq 0$ such that $p + \delta c \notin G$. Intuitively, G is bounded w.r.t. F if all straight lines starting from G and whose slope belongs to F eventually exit from G. The relationship between this definition of boundedness and the notion of l-boundedness is explored in Section 5.5.

The first step to compute the over approximation that satisfies conditions of Theorem 7, is the introduction of the operator RU (for *Remove Unbounded*) that, given a polyhedron G, removes some convex regions of G that are not l-bounded, in such a way that the resulting set is l-bounded, and every point that was l-bounded in G belongs to the resulting set. Let B be the subset of $[\![G]\!]$ containing the convex polyhedra that are bounded w.r.t. $cl(Flow(l))$, the RU operator is formally defined as

$$\mathrm{RU}(G) = \bigcup_{P \in B} P \cup \bigcup_{P \in [\![G]\!] \setminus B} \left(P \setminus P_{\swarrow_{>0}}\right). \tag{5.4}$$

The following result summarizes the main properties of the RU operator and it is proved in Section 5.5.

Theorem 8. *For all polyhedra G, the following hold:* (i) $\mathrm{RU}(G)$ *is l-bounded, and* (ii) $\mathit{bounded}_l(G) \subseteq \mathrm{RU}(G)$.

Let's start by proving how it is possible to choose over and under-approximations that satisfying conditions of Theorem 7. Given two polyhedra U and V, define $\mathit{Under} = U \setminus V$ and

$$\mathit{Over} = \mathit{Under} \cup \mathrm{RU}(\overline{U} \cap \overline{V}).$$

The polyhedra *Under* and *Over* satisfy the two assumptions of Theorem 7 namely: *(i)* $\mathit{RWA}_l^{\mathrm{M}}(U,V) \subseteq \mathit{Over} \subseteq \overline{V}$ and *(ii)* $\mathit{Over} \setminus \mathit{Under}$ is l-bounded, in fact Theorem 8 ensures that $\mathit{Over} \setminus \mathit{Under}$ is l-bounded. The following lemma proves the other assumption.

Lemma 13. *It holds* $\mathit{RWA}_l^{\mathrm{M}}(U,V) \subseteq \mathit{Over} \subseteq \overline{V}$.

Proof. For the first inclusion, let $u \in \mathit{RWA}_l^{\mathrm{M}}(U,V)$. If $u \in \mathit{Under} = U \setminus V$, we are done. Otherwise, $u \in \overline{U} \cup V$. Moreover, by definition of $\mathit{RWA}^{\mathrm{M}}$, $u \in \overline{V}$ (and hence $u \in \overline{U} \cap \overline{V}$) and for all activities $f \in \mathit{Adm}(\langle l, u\rangle)$ there exists $\delta \geq 0$ such that $f(\delta) \in U$. Hence, u is l-bounded in $\overline{U} \cap \overline{V}$. By property *(ii)* of Theorem 8, $u \in \mathrm{RU}(\overline{U} \cap \overline{V}) \subseteq \mathit{Over}$.

For the second inclusion, let $u \in \mathit{Over}$. If $u \in \mathit{Under} = U \setminus V$, clearly $u \notin V$. Otherwise, $u \in \mathrm{RU}(\overline{U} \cap \overline{V}) \subseteq \overline{U} \cap \overline{V} \subseteq \overline{V}$, and we are done. ∎

Section 5.6 shows how to effectively compute $\mathrm{RU}(\cdot)$, and hence *Over*, using basic operations on polyhedra. Moreover, $\mathit{RWA}_l^{\mathrm{m}}(\cdot,\cdot)$ is shown to be computable in Section 4.3 of Chapter 4. Therefore, $\mathit{RWA}_l^{\mathrm{M}}(U,V)$ can be computed using equation (5.3). In turn, this allows the computation of $\mathit{CPre}^{\mathrm{R}}(\cdot)$ using Lemma 12.

Theorem 9. *For all polyhedral sets of states A, $\mathit{CPre}^{\mathrm{R}}(A)$ is computable.*

Notice that the above result provides no guarantee of termination for the global fixpoint (5.1). In particular, it does not imply semi-decidability of the reachability control problem, as fixpoint (5.1) may not be reached within ω iterations of $\mathit{CPre}^{\mathrm{R}}$.

5.5 On Bounded and Thin Polyhedra

The objective of this section is to prove the properties of the $\mathrm{RU}(\cdot)$ operator pertaining l-boundedness, which are stated by Theorem 8. Since l-boundedness

is hard to directly reason about, we relate it to *geometric* boundedness, i.e., boundedness w.r.t. straight-line activities.

Let us first recall the following lemma, which is an adaptation of Lemma 4.1 in [AHH96], and states that any point reached by an admissible trajectory can be reached with a straight-line admissible trajectory as well.

Lemma 14 ([AHH96])**.** *For all points $p \in Inv(l)$, activities $f \in Adm(\langle l, p\rangle)$ and $\delta > 0$, there exists $c \in Flow(l)$ such that $f(\delta) = p + \delta c$.*

The following is a trivial observation.

Proposition 4. *If F is a convex polyhedron containing the origin, then no polyhedron is bounded w.r.t. F.*

A polyhedron G is said *thin w.r.t.* F if for all $p \in G$, $c \in F$, and $\delta > 0$, it holds $p + \delta c \notin G$. Intuitively, G is bounded (resp., thin) w.r.t. F if all straight lines starting from G and whose slope belongs to F eventually (resp., immediately) exit from G. The relationships between the geometric concepts defined in this section and the notions of l-thin and l-bounded are summarized in Figure 5.4.

Obviously, being thin w.r.t. F implies being bounded w.r.t. F. Moreover, being l-thin implies being thin w.r.t. $Flow(l)$, since straight-line activities are a special case of general activities. The following lemma shows that the converse also holds.

Lemma 15. *For all convex polyhedra P, if P is thin w.r.t. $Flow(l)$ then P is l-thin.*

Proof. Assume that P is not l-thin. Then, there exists a point $p \in P$, an activity $f \in Adm(\langle l, p\rangle)$ and a time $\delta > 0$ such that $f(\delta) \in P$. By Lemma 14, there exists $c \in Flow(l)$ such that $f(\delta) = p + \delta c \in P$. Hence, P is not thin w.r.t. $Flow(l)$. ∎

The following lemma shows that all points of G that are removed by $\mathrm{RU}(G)$ are not l-bounded in G (i.e., $\mathrm{RU}(G)$ does not "remove too much").

Lemma 16. *If a convex polyhedron P is not bounded w.r.t. $cl(Flow(l))$ then each point in $P \cap P_{\swarrow_{>0}}$ is not l-bounded in P.*

Proof. Since P is not bounded w.r.t. $cl(Flow(l))$, there are $p \in P$ and $c \in cl(Flow(l))$ such that for all $\delta \geq 0$ it holds $p + \delta c \in P$. Let p' be a point in $P \cap P_{\swarrow_{>0}}$. If $c \in Flow(l)$, let f be the activity defined by $f(\delta) = p' + \delta c$. Clearly, $f \in Adm(\langle l, p'\rangle)$. Since P is convex, $f(\delta) \in P$ for all $\delta \geq 0$, and the thesis is obtained.

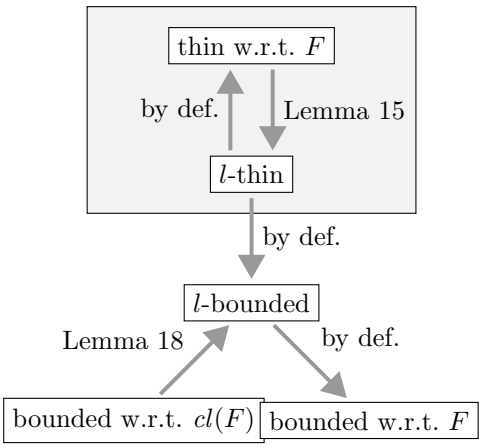


Figure 5.4: Relationships between properties of convex polyhedra. Arrows represent implications and $F = \mathit{Flow}(l)$.

Consider now the case $c \in cl(\mathit{Flow}(l)) \setminus \mathit{Flow}(l)$, and define an activity f that starts in p', remains in P forever, and whose slope tends asymptotically to c, without ever reaching it. Since $p' \in P \cap P_{\swarrow_{>0}}$, there is $c' \in \mathit{Flow}(l)$ and $\delta' > 0$ such that $p' + \delta' c' \in P$. By convexity, it also holds $p' + \delta'' c' \in P$ for all $0 \leq \delta'' \leq \delta'$. For all $\delta \geq 0$, the activity f is defined as follows.

$$f(\delta) = p' + \delta c + \delta' \left(1 - e^{-\frac{\delta}{\delta'}}\right)(c' - c).$$

Notice that $f(0) = p'$ and, for all $\delta \geq 0$, $f(\delta)$ can be expressed as $p' + \alpha c + \beta c'$, where $\alpha \geq 0$ and $\beta \in [0, \delta')$. The point $p' + \beta c'$ belongs to P. Since P is convex and not bounded w.r.t. $\{c\}$, then $f(\delta) = p' + \alpha c + \beta c' \in P$. Next, it must be verified that the slope of f is always contained in $\mathit{Flow}(l)$. We have:

$$\dot{f}(\delta) = c - e^{-\frac{\delta}{\delta'}} \cdot c + e^{-\frac{\delta}{\delta'}} \cdot c' = \left(1 - e^{-\frac{\delta}{\delta'}}\right) c + e^{-\frac{\delta}{\delta'}} \cdot c'.$$

Since, for all $\delta \geq 0$, it holds $e^{-\frac{\delta}{\delta'}} \in (0, 1]$, we have that $\dot{f}$ is a convex combination of c and c', different from c. By the convexity of $\mathit{Flow}(l)$ it is possible to conclude that $\dot{f}(\delta) \in \mathit{Flow}(l)$. ∎

In addition, the fact that the result of $\mathrm{RU}(G)$ is l-bounded (i.e., $\mathrm{RU}(G)$ does not "remove too little"), will be now proved. In order to obtain this result (stated as Lemma 20), a few preliminary lemmata are necessary.

The following result show that if the origin does not belong to the topological closure of the flow, then there is a flow direction u such that all possible flows advance in the direction u by at least $|u|$ for each time unit. Hereinafter, the origin, i.e., the point whose coordinates are 0, are denoted by $\mathbf{0}$.

Lemma 17. *Assume* $\mathbf{0} \notin cl(Flow(l))$. *Then there exists* $u \in cl(Flow(l))$ *such that for all* $v \in Flow(l)$ *the scalar projection of* v *onto* u *is at least* $|u|$ *(i.e.,* $\frac{u \cdot v}{|u|} \geq |u|$*, where* $\cdot$ *denotes the inner product).*

Proof. Let $F = cl(Flow(l))$. Let u be a point in F with minimal distance from the origin (equivalently, minimal length $|u|$). The topological closeness of F ensures that such a point exists. The convexity of F ensures that such a point is unique (i.e., u has *minimum* length). Let v be an arbitrary element of $Flow(l)$. Let v' be the vector projection of v on the direction u, i.e., $v' = \frac{u \cdot v}{|u|^2} u$. The fact that $|v'| \geq |u|$ will be shown. Assume by contradiction that $|v'| < |u|$. By the convexity of F, any convex combination of u and v belongs to F. Considering the triangle with vertices u, v and the origin, the angle of vertex u is less than $90°$. Hence, there is a convex combination of u and v which is closer to the origin than u, which is a contradiction. ∎

The following fact is obvious, since straight lines are a special case of activities.

Proposition 5. *If a polyhedron is* l*-bounded, then it is bounded w.r.t.* $Flow(l)$.

Being bounded w.r.t. $Flow(l)$ is necessary but not sufficient for a polyhedron to be l-bounded, as shown by the following example.

Example 16. *Consider the unbounded polyhedron* P *shown on the r.h.s. of Figure 5.5. The dashed contour of* F *(on the l.h.s. of the figure) indicates that* F *(i.e.,* $Flow(l)$*) is topologically open, so that its extremal directions* $(1, 0)$ *and* $(0, 1)$ *are not proper (i.e., they do not belong to* F*). It turns out that* P *is bounded w.r.t.* $Flow(l)$*, because all straight lines whose slope belongs to* F *eventually exit from it, but it is not* l*-bounded. The figure shows an activity that remains forever in* P*. Its slope approaches asymptotically the extremal direction* $(1, 0)$ *(similarly to the proof of Lemma 16).*

Lemma 18 presents a sufficient condition for being l-bounded.

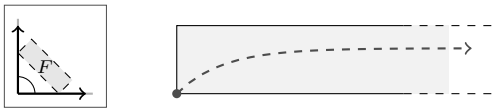


Figure 5.5: On the right, a polyhedron which is bounded w.r.t. $Flow(l)$ but not l-bounded, and an activity that remains forever in it.

Lemma 18. *If a polyhedron is bounded w.r.t.* $cl(Flow(l))$ *then it is* l*-bounded.*

Proof. Let $F = cl(Flow(l))$. By Proposition 4, F does not contain the origin. By Lemma 17, there exists $u \in F$ such that for all $v \in Flow(l)$ it holds $u \cdot v \geq |u|^2$.

Let G be a polyhedron which is bounded w.r.t. F, and let $p \in G$ and $f \in Adm(\langle l, p \rangle)$. For all $\delta \geq 0$, it holds $\dot{f}(\delta) \in Flow(l)$. From the above argument, the vector projection of $\dot{f}(\delta)$ on the direction u has length at least $|u|$. Hence, for each time unit, the activity f advances in the direction u by at least $|u|$. Since G is bounded w.r.t. $\{u\}$, the thesis is obteined. ∎

The following results shows that two l-thin polyhedra cannot be adjacent in a direction of the flow.

Lemma 19. *Let L_1 and L_2 be two l-thin polyhedra. For all $p \in L_1$, $c \in Flow(l)$ and $\delta > 0$ there exists $0 < \delta' \leq \delta$ such that $p + \delta c \notin L_2$.*

Proof. By contradiction, assume that there is $p \in L_1$, $c \in Flow(l)$ and $\delta > 0$ such that for all $0 < \delta' \leq \delta$ it holds $p + \delta c \in L_2$. Let $p_1 = p + \frac{\delta}{2} c$ and $p_2 = p + \delta c = p_1 + \frac{\delta}{2} c$. We have that $p_1, p_2 \in L_2$, which contradicts the fact that L_2 is thin w.r.t. $Flow(l)$. ∎

The following lemma lifts l-boundedness from convex polyhedra to general polyhedra.

Lemma 20. *Let G be a polyhedron such that each $P \in [\![G]\!]$ is l-bounded. Then, G is l-bounded.*

Proof. If G is empty, the result is trivially true. If $\mathbf{0} \in cl(Flow(l))$, by Proposition 4 no polyhedron is bounded w.r.t. $cl(Flow(l))$. As a consequence, by Lemma 16, for every convex polyhedron in $P \in [\![G]\!]$, we have that each point in $P \cap P{\swarrow_{>0}}$ is not l-bounded. Since, however, P is l-bounded by assumption, it must be $P \cap P{\swarrow_{>0}} = \emptyset$, and this can only hold if P is thin w.r.t. $Flow(l)$. Since, by Lemma 15, every convex polyhedron thin w.r.t. $Flow(l)$ is also l-thin, it is possible to conclude that each $P \in [\![G]\!]$ is l-thin. By Lemma 19, two l-thin polyhedra cannot be adjacent in a direction of flow. Therefore, any activity that starts in G immediately exits from it, and the thesis is obtained.

Assume now that $\mathbf{0} \notin cl(Flow(l))$. The proof proceeds by induction on the cardinality of $[\![G]\!]$. If the cardinality is 1, the thesis immediately follows.

Let $|[\![G]\!]| > 1$, and pick an arbitrary $P \in [\![G]\!]$. By inductive hypothesis, $G \setminus P$ is l-bounded. By contradiction, assume that G is not l-bounded, and let $p \in G$ and $f \in Adm(\langle l, p \rangle)$ be such that $f(\delta) \in G$ for all $\delta \geq 0$. If f eventually remains forever in $G \setminus P$ (i.e., there is $\delta \geq 0$ such that for all $\delta' \geq \delta$ it holds $f(\delta') \in G \setminus P$), then $G \setminus P$ is not l-bounded, contradicting the inductive

hypothesis. If f eventually remains forever in P, the contradiction follows form the assumption that P is l-bounded. Therefore, f enters and exits from P infinitely often. Formally, for all $\delta \geq 0$ there exist $\delta', \delta'' \geq \delta$ such that $f(\delta') \in P$ and $f(\delta'') \in G \setminus P$. Since $[\![G \setminus P]\!]$ is a finite set of convex polyhedra, there must be a convex polyhedron $P' \in [\![G \setminus P]\!]$ which is adjacent to P and such that f crosses the boundary between P and P' infinitely often.

Let, now, $b = \mathit{bndry}(P, P')$ be the boundary between P and P', such that f crosses b infinitely often, i.e., for all $\delta \geq 0$ there is $\delta' > \delta$ such that $f(\delta') \in b$. Since both P and P' are convex and l-bounded by assumption, then b is both convex and l-bounded. Let $\{\delta_i\}_{i \in \mathbb{N}}$ be a sequence of time instants such that *(i)* $f(\delta_i) \in b$ and *(ii)* $\delta_{i+1} \geq \delta_i + 1$.

The fact that b must be bounded w.r.t. $cl(\mathit{Flow}(l))$ will be proved. Assume, by contradiction, that it is not, then b must be l-thin. Consider now any $i \in \mathbb{N}$. By Lemma 14, there is a $c \in \mathit{Flow}(l)$, with $f(\delta_{i+1}) = f(\delta_i) + \delta \cdot c$ and $\delta = (\delta_{i+1} - \delta_i) \geq 1$, by the choice of $\{\delta_i\}_{i \in \mathbb{N}}$. For all $0 < \delta' < \delta$, the point $f(\delta_i) + \delta' \cdot c$ is different from $f(\delta_i)$ and $f(\delta_{i+1})$ since $c \neq \mathbf{0}$, and, by convexity of b, belongs to b, contradicting the fact that b is l-thin. Therefore, b is bounded w.r.t. $cl(\mathit{Flow}(l))$.

Now, by Lemma 17, let $u \in cl(\mathit{Flow}(l))$ be such that, for all for all $v \in \mathit{Flow}(l)$, the scalar projection of v on u is at least $|u|$. Hence, for each $i \in \mathbb{N}$, when going from $f(\delta_i)$ to $f(\delta_{i+1})$ the activity f progresses by at least $|u|$ in the direction of u. This contrasts with the fact that b is bounded w.r.t. $\{u\}$, and the thesis is obtained. ∎

Now, there are all the results and the notions in order to prove Theorem 8.

Proof of Theorem 8. *(i)* $\mathrm{RU}(G)$ *is l-bounded.* For a convex polyhedron P, the set $P \setminus (P_{\swarrow_{>0}})$ is l-thin, as may be easily verified from the definitions. Hence, each convex polyhedron in $[\![\mathrm{RU}(G)]\!]$ is either bounded w.r.t. $cl(\mathit{Flow}(l))$ or l-thin. Since each l-thin polyhedron is l-bounded by definition, and by Lemma 18, we obtain that each convex polyhedron in $[\![\mathrm{RU}(G)]\!]$ is l-bounded. By Lemma 20, $\mathrm{RU}(G)$ is l-bounded.

(ii) $\mathit{bounded}_l(G) \subseteq \mathrm{RU}(G)$. Let $p \in \mathit{bounded}_l(G)$. Then there must be at least one convex polyhedron $P \in [\![G]\!]$ with $p \in P$. If P is bounded w.r.t. $cl(\mathit{Flow}(l))$ then, by equation (5.4), $p \in \mathrm{RU}(G)$. If, on the other hand, P is not bounded w.r.t. $cl(\mathit{Flow}(l))$, by Lemma 16 we have that $P \cap P_{\swarrow_{>0}}$ is not l-bounded in P and, *a fortiori*, not l-bounded in G . Therefore, $p \in P \setminus P_{\swarrow_{>0}}$ and, by equation (5.4), $p \in \mathrm{RU}(G)$. Hence the conclusion. ∎

5.6 Computing the RU Operator

As shown in the previous section, let G be a polyhedra, the operator RU (remove unbounded) removes some convex regions of G that are not l-bounded, in such a way that the remaining set is l-bounded. In order to compute $RWA_l^{\mathrm{M}}(U,V)$, it is necessary to be able to (i) compute the positive-preflow $P \swarrow_{>0} F$ of a convex polyhedron P w.r.t. another convex polyhedron F, and (ii) collect, for any polyhedron G, the convex polyhedra $P \in [\![G]\!]$ which are bounded w.r.t. the convex polyhedron F. The first operation is explained in Section 4.7 of Chapter 4, while next section shows how the second operation can be efficiently implemented employing a canonical representation of convex polyhedra.

5.6.1 Testing for Boundedness w.r.t. the Flow

For a convex polyhedron P, let $O_P = (\{\mathbf{0}\}, \emptyset, R_P)$ denote its *characteristic cone*, i.e., the closed polyhedron generated by the origin $\mathbf{0}$ and all the rays of P. The following theorem (see Figure 5.6) shows how one can effectively and efficiently test whether P is bounded w.r.t. F.

Theorem 10. *For all convex polyhedra P and F, P is bounded w.r.t. F iff $O_P \cap F = \emptyset$.*

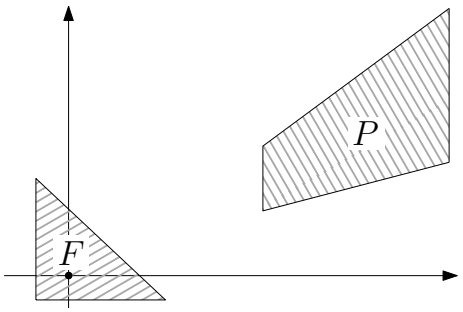
Proof. [$\Rightarrow$] By hypothesis, for all $p \in P$ and for all $c \in F$ there exists $\delta \geq 0$ such that $p + \delta \cdot c \notin P$. By Proposition 4 we have that $\mathbf{0} \notin F$. Let $c \in F$, the fact that $c \notin O_P$ will be now shown. Assume by contradiction that $c \in O_P$, we can write $c = 1 \cdot \mathbf{0} + \sum_{r \in R_p} \beta_r r = \sum_{r \in R_p} \beta_r r$. Now, let $x \in V_p$ be a vertex of P, the point $x' = x + \gamma c$ belongs to P, for all $\gamma \geq 0$. Indeed,

$$x' = x + \gamma c = 1 \cdot x + \gamma \sum_{r \in R_p} \beta_r r = 1 \cdot x + \sum_{r \in R_p} \gamma \beta_r r.$$

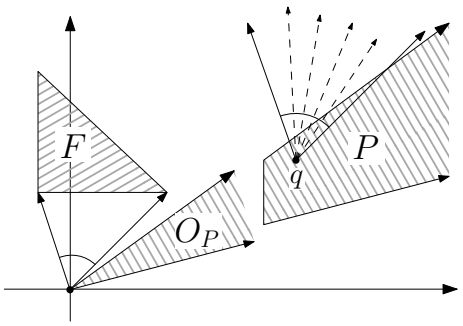
Therefore, $x' \in P$, i.e. P is not bounded w.r.t. F, contradicting the hypothesis.

[$\Leftarrow$] Assume by contradiction that $c \in F \cap O_P$. By the decomposition theorem for convex polyhedra [Sch86], since O_P is the characteristic cone of P, there exists a non-empty convex polyhedron P' such that $P = P' \oplus O_P$. Moreover, since $c \in O_P$, also $\delta c \in O_P$ for all $\delta \geq 0$. We can then conclude that for all $p' \in P'$, it holds $p' + \delta c \in P$ for all $\delta \geq 0$. Therefore, P is not bounded w.r.t. $\{c\}$ and *a fortiori* w.r.t. F. ∎

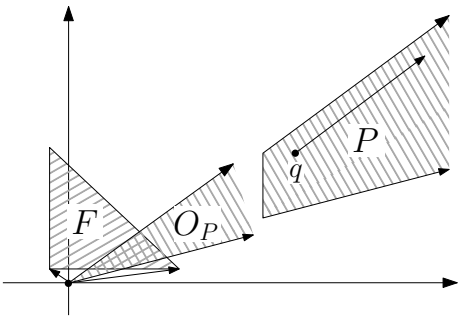
Theorem 10 and Proposition 4 identify three different cases, depicted in Figure 5.6, as follows:



(a) $\mathbf{0} \in F$. P is not bounded w.r.t. F.



(b) $F \cap O_P = \emptyset$. P is bounded w.r.t. F.



(c) $F \cap O_P \neq \emptyset$. P is not bounded w.r.t. F.

Figure 5.6: Test for boundness w.r.t. the flow.

1. Figure 5.6(a) shows the first case, where the origin belongs to the flow F. Hence P is not bounded w.r.t. F because one may choose the activity corresponding to the origin, and then it is no possible to leave P.

2. Figure 5.6(b) shows the second case, where the origin does not belong to F and the characteristic cone of P, namely O_P, is disjoint from F. When such conditions are verified, it is never possible to remain forever in P (e.g., starting from the point q in the figure each activity eventually leads out of P). Then P is bounded w.r.t. F.

3. Figure 5.6(c) shows the last case, when the flow F and O_P have common points: under this condition, one may choose an activity $f \in F$ such that all points in P always remain in P following f (e.g., from the point q, following the activity shown in the figure by the internal arrow, it is never possible to leave P). In particular, such f belongs to the intersection between O_P and F. Then P is not bounded w.r.t. F.

Now the discussion about the reachability control problem for linear hybrid games is complete. The implementation of the operators introduced in this chapter is the focus of Chapter 6.

Part III

Implementation and Experiments

Chapter 6

Implementations of Algorithms for the Safety

In this Chapter, techniques require to efficiently implement the algorithms shown in Chapter 4 and Chapter 5, are discussed. First sections are dedicated to the implementation of the general fixpoint algorithm to solve the safety control problem that, as in the case for both discrete (see Chapter 1) and timed (see Chapter 2), is based on the computation of the controllable predecessor operator (for safety) $CPre^{\mathrm{S}}$. This operator requires, in turn, the implementation of the may reach while avoiding RWA^{m} operator. Actually, instead of RWA^{m}, this thesis shows the implementation of the dual operator *must stay or reach* SOR^{M} that, given two polyhedra Z and V, contains the points which either remain in Z forever or reach V along a system trajectory that does not leave Z. Section 6.2 shows several implementations of SOR^{M}, starting from that resulting directly from the fixpoint characterization of the algorithm (derived from the dual operator RWA^{m}) and introducing along the way a number of efficiency improvements.

The second part of this chapter is dedicated to the implementation of the general fixpoint algorithm to solve the reachability control problem. Also this algorithm is based on the controllable predecessor operator but, unlike discrete and continuous cases, in the hybrid context the controllable operator for reachability (here called $CPre^{\mathrm{R}}$) is different from that for safety ($CPre^{\mathrm{S}}$) (see Chapters 4 and 5). In order to compute the $CPre^{\mathrm{R}}$ operator, the implementation of the must reach while avoiding operator RWA^{M} is required. In Chapter 5 is shown a connection between RWA^{M} and RWA^{m}. Hence, the RWA^{M} operator can be implemented by using the SOR^{M} implementation done for the safety and by using some additional operators as RU and the positive preflow, in order to compute a suitable over-approximation that allows the computation of RWA^{M}

based on SOR^{M}.

Operators and algorithms presented here are implemented on the top of the tool PHAVer. PHAVer is an open tool for the verification of hybrid automata that, being based on polyhedra abstraction, makes an extensive use of operators made available by the Parma Polyhedra Library (PPL) [BHZ08]. Extending PHAVer with algorithms seen in previous chapters a new tool, called PHAVer+, comes out with additional synthesis features.

Notice that, contrary to most recent literature on the subject, this work is focused on exact algorithms. Although it is established that exact analysis and synthesis of realistic hybrid systems is computationally demanding, it can be used as solid ground for the ongoing research effort on approximate techniques. For instance, a tool implementing an exact algorithm, like PHAVer+, may serve as a benchmark to evaluate the performance and the precision of an approximate tool.

6.1 Implementation of the Global Fixpoint for Safety

Theorem 2 gives a fix-point characterization of the safety control problem for linear hybrid games, that can be easily turned into the Algorithm 4.

Algorithm 4: $Safety(H, T)$

Input: LHG $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$, Set of States T.
Output: Set of winning states W.
Data: Set of states W_{new}, Boolean *fix_point*.
fix_point := $false$;
$W := T$;
while *(fix_point* = $false$*)* **do**
  $W_{new} := T \cap get\_CPre^{\mathrm{S}}(H, W)$;
  fix_point := $get\_fixpoint(W, W_{new}, Loc)$;
return W;

The auxiliary function $get\_fixpoint$ check if the fixpoint is reached. In this version it is trivially fixed that

$$get\_fixpoint(W, W_{new}, Loc) = \begin{cases} true & \text{if } W = W_{new} \\ false & \text{otherwise} \end{cases}$$

Given the linear hybrid game $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$ and the set of safe states T, the safety control problem for H w.r.t. T can be solved by calling Algorithm 4 on parameters G and T. The heart of this algorithm involves into the computation of the operator $CPre^{\mathrm{S}}$. Since, by Equation 4.2, $CPre^{\mathrm{S}}$ can be expressed as

$$CPre^{\mathrm{S}}(A) = \bigcup_{l \in Loc} \{l\} \times \Big(A\lfloor_l \setminus RWA_l^{\mathrm{m}}\big(InvS\lfloor_l \cap (\overline{A\lfloor_l} \cup B_l), C_l \cup \overline{InvS\lfloor_l}\big)\Big), \quad (6.1)$$

the implementation of $CPre^{\mathrm{S}}$ requires the computation of the operator RWA^{m}. Actually, instead of computing RWA^{m}, the implementation proposed in this thesis is based on the computation of the dual operator *must stay or reach* $SOR_l^{\mathrm{M}}(Z, V)$. This operator contains the points which either remain in Z forever or reach V along a system trajectory that does not leave Z, and can be formally defined as follows:

$$SOR_l^{\mathrm{M}}(Z, V) = \overline{RWA_l^{\mathrm{m}}(\overline{Z}, V)}. \quad (6.2)$$

As a consequence, $CPre^{\mathrm{S}}(A)$ can be expressed as

$$\bigcup_{l \in Loc} \{l\} \times \Big(A\lfloor_l \cap SOR_l^{\mathrm{M}}\big(\overline{Inv}\lfloor_l \cup (A\lfloor_l \setminus B_l), C_l \cup \overline{Inv\lfloor_l}\big)\Big).$$

The implementation of $CPre^{\mathrm{S}}$ is done by Algorithm 5 that takes in input the hybrid game H and the set of safe states T.

Algorithm 5: $get\_CPre^{\mathrm{S}}(H, W)$

Input: LHG $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$, Set of States W.
Output: Set of States $CPre^{\mathrm{S}}(W)$.
Data: Set of States $CPre^{safe}$, *Poly* A, B, C.

```
CPre^safe := ∅;
foreach (l ∈ Loc) do
    A := W⌊_l;
    B := ∅;
    C := ∅;
    foreach (t ∈ Edg_c | t = ⟨l, μ, l'⟩) do
        C := C ∪ get_pre(W⌊_l', Inv(l'), μ);
    foreach (t ∈ Edg_u | t = ⟨l, μ, l'⟩) do
        B := B ∪ get_pre(¬W⌊_l', Inv(l'), μ);
    CPre^safe :=
    CPre^safe ∪ {l, A ∩ get_SOR^M(¬Inv(l) ∪ (A \ B), C ∪ ¬Inv(l), Flow(l))};
return CPre^safe;
```

The core of this algorithm is a loop over the set of locations *Loc*: for each location $l \in Loc$, Algorithm 7 (the implementation of SOR^{M}) is called on parameters (i) $\overline{Inv(l)} \cup (A \setminus B)$, (ii) $C \cup \overline{Inv(l)}$ and (iii) $Flow(l)$, where:

- The polyhedron $A = A\lfloor_l$ is the projection of A on l and represents an over-approximation of the "good" region for the location l (the points in A potentially belong to SOR^{M}).

- The polyhedron B (for "bad" region) contains the set of all valuations v such that there exists an uncontrollable transition $t = \langle l, \mu, l' \rangle \in Edg_{\mathrm{u}}$, whose guard is satisfied by v, that leads the system immediately outside of the good region A, i.e. by taking t the system immediately reaches $\overline{A}$. In other words, B is the projection of $Pre_u(\overline{A})$ on l, obtained by calling Algorithm 6 on parameter $\overline{W}$ and t, for each $t \in Edg_{\mathrm{u}}$ whose source location is l.

- The polyhedron C (for "controllable" region) contains the set of all valuations v such that there exists a controllable transition $t = \langle l, \mu, l' \rangle \in Edg_{\mathrm{c}}$, whose guard is satisfied by v, that leaves the system into the safe region A, i.e. by taking t the system remains in $\overline{A}$. In other words, C is the projection of $Pre_c(A)$ on l, obtained by calling Algorithm 6 on parameter W and t, for each $t \in Edg_{\mathrm{c}}$ whose source location is l.

Notice that Algorithm 6 is called in order to compute polyhedra B and C. Before of proceeding to the detailed explanation of this algorithm, some additional notions are introduced.

Given a polyhedron α, a set of continuous variables $X = \{x_1, \ldots, x_n\}$ and a set of continuous (primed) variables $X' = \{x'_1, \ldots, x'_n\}$, the notation $\alpha[X]$ (resp., $\alpha[X']$) explicates that the polyhedron α is defined over the set X (resp., X'). Moreover, given a polyhedron $\alpha[X]$, the notation $\alpha[X/X']$ identifies the operation of renaming the variable $x_i \in X$ with the variable $x'_i \in X'$, for all $1 \leq i \leq n$. Given a polyhedron $Z \subseteq X$, the notation $exists(\alpha[X], Z)$ denotes the projection of α over Z. Formally, for all polyhedra $\alpha[X]$ and $Z \subseteq X$

$$exists(\alpha[X], Z)) = \exists Z.\alpha[X].$$

Using such notations, Algorithm 6 on parameters $\alpha[X]$, I and $\mu[X \cup X']$, computes the polyhedron Pre defined as follows:

$$Pre = I \cap \{\exists x'.(\alpha[X/X'] \cap \mu)\}.$$

Notice that, the existential quantifier on the expression above can be interpreted geometrically as the projection of $\alpha[X/X'] \cap \mu$ over the variable in X'. The Parma Polyhedra Library provides primitives for both operations involved in the computation of Pre, i.e. (i) renaming the variables and (ii) the existential quantifier. Algorithm 6 refers to these functions by *reneame* and *exists*, respectively. The former takes as input the polyhedron $\alpha[X]$, the set $X = \{x_1, \ldots, x_n\}$ and the set $X' = \{x_1, \ldots, x_n\}$ of the new variables, and gives

in output the polyhedron $\alpha[X/X']$. The latter takes as input a polyhedron $\alpha[X]$ and a set of variables $Z \subseteq X$ and gives in output the projection of α on Z.

Algorithm 6: $get\_pre(\alpha, I, t)$

Input: *Poly* $\alpha[X]$, I, $\mu[X \cup X']$.
Output: *Poly* $Pre_x(\alpha)$.
$Pre := I \cap exists(reneame(\alpha, X, X') \cap \mu)$;
return Pre;

This ends the discussion about the global algorithms, and now the focus becomes the local fixpoint algorithm for SOR^{M}. The next two sections show several implementation of this operator.

6.2 Efficient Computation of SOR^{M}

This section shows how the must stay or reach operator $SOR_l^{\mathrm{M}}(Z, V)$ is computed, given two polyhedra Z and V.

From (6.2), the following fixpoint characterizes the operator SOR_l^{M}:

$$SOR_l^{\mathrm{M}}(Z,V) = \overline{RWA_l^{\mathrm{m}}(\overline{Z},V)} = \overline{\mu W \,.\, \overline{Z} \cup \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![W]\!]} \big(P \cap entry(P,P')_{\swarrow_l}\big)} =$$

$$= \nu W \,.\, Z \cap \overline{\bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W}]\!]} \big(P \cap entry(P,P')_{\swarrow_l}\big)} =$$

$$= \nu W \,.\, Z \setminus \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W}]\!]} \big(P \cap entry(P,P')_{\swarrow_l}\big). \tag{6.3}$$

The fixpoint equation (6.3) can easily be converted into an iterative algorithm, consisting in generating a (potentially infinite) sequence of polyhedra $(W_n)_{n\in\mathbb{N}}$, where $W_0 = Z$ and

$$W_{i+1} = W_i \setminus \bigcup_{P\in[\![\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W_i}]\!]} \Big(P \cap entry(P,P')_{\swarrow_l}\Big). \tag{6.4}$$

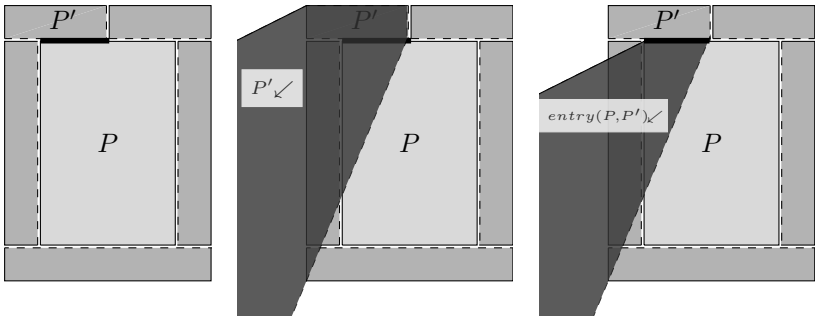
Theorem 4 proves that such sequence converges to a fixpoint within a finite number of steps.

The naive implementation of the algorithm is done by an outer loop over the polyhedra $P \in [\![\overline{V}]\!]$ and an inner loop over $P' \in [\![\overline{W_i}]\!]$. As a first improvement, one can observe that each iteration of the outer loop removes from W_i a portion of $P \in [\![\overline{V}]\!]$. Hence, the portion of P that is not contained in W_i is irrelevant, and Equation (6.4) may be replaced with:

$$W_{i+1} = W_i \setminus \bigcup_{P\in[\![W_i\cap\overline{V}]\!]} \bigcup_{P'\in[\![\overline{W_i}]\!]} \Big(P \cap entry(P,P')_{\swarrow_l}\Big). \tag{6.5}$$

Moreover, it is possible to avoid the need to intersect W_i with $\overline{V}$ at each iteration, by starting with $W_0 = Z \cap \overline{V}$ and setting:

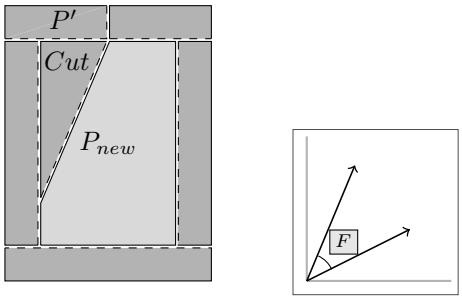
$$W_{i+1} = W_i \setminus \bigcup_{P \in [\![W_i]\!]} \bigcup_{P' \in [\![\overline{W_i}]\!]} \Big(P \cap entry(P, P')_{\swarrow l} \Big). \tag{6.6}$$



(a) Initial input with $bndry(P, P')$ highlighted.

(b) Pre-flow of P'.

(c) Entry region.



(d) The region cut from P and the resulting polyhedron.

(e) $Flow(l)$.

Figure 6.1: One step of SOR^{M} computation.

Consider the Figure 6.1. At the beginning, all points in $W_0 = Z$ (light gray in Figure 6.1(a)) are considered to be safe. Then, for each $P \in [\![W_0]\!]$, the set of points $p \in P$ that might reach some polyhedron $P' \in \overline{W}_0$, is identified. In order to compute this area the algorithm check, for each $P' \in \overline{W}_0$, if P and P' are adjacent: let $b = bndry(P, P')$ be the polyhedron $\big(cl(P) \cap P'\big) \cup \big(P \cap cl(P')\big)$, P and P' are called to be *adjacent* iff $b \neq \emptyset$ (b is represented by the thick line in Figure 6.1(a)). If P and P' are adjacent then the algorithm computes the related entry region from P to P', that is $entry(P, P') = b \cap P'_{\swarrow l}$. This entry region contains all the points of b that may reach the polyhedron P' by following some straight-line activity in the considered location l (see Figure 6.1(c)). Now, the algorithm can computes the effective set of points belong to P that may reach P' by following a straight-line activity. This set of points is defined by

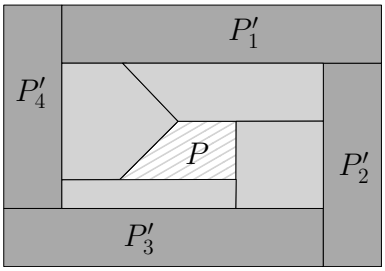


Figure 6.2: Unnecessary boundary checks.

$Cut = P \cap entry(P, P')_{\swarrow l}$. Clearly, the polyhedron Cut must be removed from P, and the resulting safe polyhedra is $P_{new} = P \setminus Cut$ (Figure 6.1(d)). In the next step, W_1 will contain the polyhedra P_{new} (notice that, if $entry(P, P') = \emptyset$, W_1 will still contain the polyhedron P). Notice that *(i)* P_{new}, being the result of a set-difference operation, may be non-convex, and *(ii)* Cut becomes an unsafe area (will belong to $\overline{W}_1$).

The implementation described so far is called the *basic approach* in the following.

6.2.1 Introducing Adjacency Relations

In the basic approach, the inner loop is repeated for each $P' \in [\![\overline{W_i}]\!]$, even if the convex polyhedra P' is not adjacent to P (i.e., $entry(P, P') = \emptyset$). For example, the polyhedra $P'_1, P'_2, P'_3, P'_4 \in [\![\overline{W_i}]\!]$ shown in Figure 6.2 are not adjacent to the considered $P \in [\![W_i]\!]$. In such situation the basic approach performs a number of irrelevant boundary checks.

In order to avoid these unnecessary checks, the binary relation of *external adjacency* Ext_i, which associates a polyhedron $P \in [\![W_i]\!]$ with its entry regions $entry(P, P') \neq \emptyset$, for all $P' \in [\![\overline{W}_i]\!]$, is introduced. Formally,

$$Ext_i = \big\{\langle P, entry(P, P')\rangle \mid P \in [\![W_i]\!], P' \in [\![\overline{W}_i]\!], \text{ and } entry(P, P') \neq \emptyset\big\}. \tag{6.7}$$

Once Ext_i is introduced and properly maintained, it allows to optimize the outer loop: this is achieved by considering only those polyhedra that are associated with at least one entry region R in Ext_i, instead of all polyhedra in $[\![W]\!]$. Then, by using Ext_i the Equation (6.6) can be replaced with

$$W_{i+1} = W_i \setminus \bigcup_{\langle P,R\rangle \in Ext_i} \Big(P \cap R_{\swarrow l}\Big). \tag{6.8}$$

Comparing Equation 6.6 and Equation 6.8, one can conclude that by using the external adjacency, the number of steps performed by Algorithm 7 can be

reduced, because Ext_i contains only polyhedra that have a non-empty entry region. Formally, fixing $Expose = \bigcup_{\langle P,R\rangle \in Ext_i} \left(P \cap R_{\swarrow l}\right)$, the following holds

$$\left|Expose\right| \leq \left|W_i\right| * \left|\overline{W}_i\right|.$$

More, experimental evidence shows that the cardinality of the l.h.s. is much less than the cardinality of the r.l.h. This allows a large performance gain by using the external adjacency relation.

Clearly, some extra effort is required to initialize and maintain Ext_i. Initialization is performed by simply applying (6.7). The maintenance requires the efficient computation of Ext_{i+1}, that is now briefly discussed.

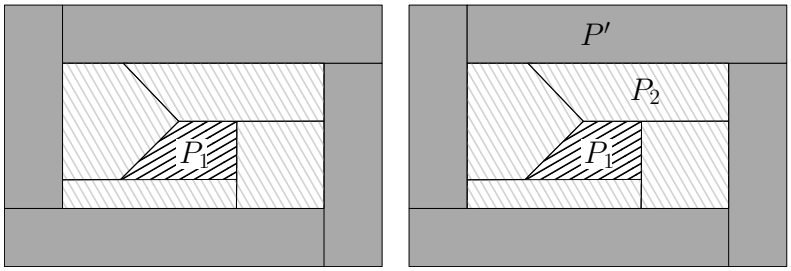
During the i-th iteration, certain convex polyhedra $P \in [\![W_i]\!]$ are cut by removing the points that may directly reach a convex polyhedron $P' \in [\![\overline{W}_i]\!]$. These cuts may *expose* other convex polyhedra in $[\![W_i]\!]$, that were previously covered by P. These exposed polyhedra will be the only ones to have associated entry regions in Ext_{i+1}. In order to be exposed by a cut made to P, a convex polyhedron must be adjacent to P.

For example, the non-exposed polyhedron P_1 shown in Figure 6.3(a), could become exposed only if some polyhedron adjacent to it (one of the polyhedron belongs to the lighter gray area in the figure), are cut during an iteration. This situation is depicted in Figure 6.3(b) where the polyhedron P_2, adjacent to P_1, has a non-empty entry region to P'. Then, the area Cut shown in Figure 6.3(c) is removed from P_2. This cutting generates the non-empty entry region from P_1 to Cut, (i.e. P_1 is now exposed) represented by the thick line in Figure 6.3(d).

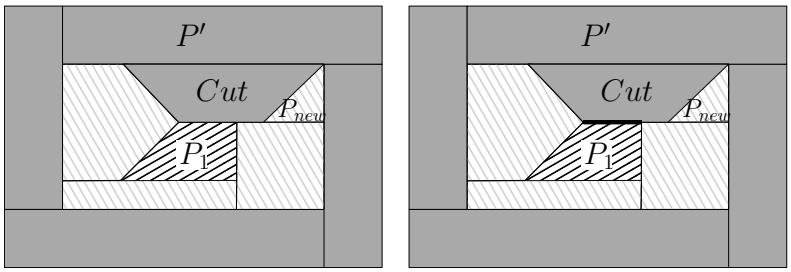
Hence, in order to compute Ext_{i+1} it is useful to have information about the adjacency among the polyhedra in $[\![W_i]\!]$. To this aim, the binary relation of *internal adjacency* Int_i between polyhedra in $[\![W_i]\!]$ is introduced:

$$Int_i = \left\{\langle P_1, P_2\rangle \mid P_1, P_2 \in [\![W_i]\!], P_1 \neq P_2 \text{ and } bndry(P_1, P_2) \neq \emptyset\right\}. \qquad (6.9)$$

The computation of Int_0 requires the complete scan of all $P_1, P_2 \in [\![W_0]\!]$, while Int_{i+1} is obtained incrementally from Int_i and Ext_i. Given $\langle P, R\rangle \in Ext_i$, let $Cut = P \cap \left(R_{\swarrow l}\right)$ and $P_{new} = P \setminus Cut$. Notice that P_{new} may be non-convex, being the result of a set-theoretical difference between two convex polyhedra. The relation Int_{i+1} is obtained by adding to Int_i the pairs of adjacent convex polyhedra (P_1, P_2) such that either *(i)* both P_1 and P_2 belong to $[\![P_{new}]\!]$, or *(ii)* one of them belongs to $[\![P_{new}]\!]$ and the other is adjacent to P according to Int_i. Moreover, once P_{new} replaces P in W_{i+1}, it is necessary to remove all the pairs $\langle P, P'\rangle$ from Ext_i and Int_i.



(a) P_1 is not exposed. (b) P_2 (adjacent to P_1) is exposed.



(c) P_2 is replaced by P_{new}. (d) P_1 is now exposed by Cut.

Figure 6.3: New entry regions.

Algorithms 7,8 and 9 represent a concrete implementation of the technique described so far. In Algorithm 7, Ext_{old} and Int_{old} represent the old adjacency relations, while Ext_{new} and Int_{new} the new ones. The first "for each" loop initializes both relations, followed by a "while" loop that iterates until the external adjacency relation is empty. Maintenance of the adjacency relations is delegated to Algorithms 8 and 9, that receive as input the relation they have to update, the convex polyhedron P whose adjacencies need to be examined, and a general polyhedron *Candidates* containing the convex polyhedra that may be adjacent to P. Additionally, Algorithm 9 also needs to know the input set V (region to be avoided) and the location flow $F = Flow(l)$.

The auxiliary function *PotentialEntry* returns the potential entry region for P, namely the set of convex polyhedra contained in $\bar{Z}$ that can have non-empty entry region from P. In this version, it is simply fixed that

$$PotentialEntry(P, Int_0, F) = [\![\bar{Z}]\!].$$

In the following, the implementation described so far is called the *global approach*, and it will be improved in Section 6.2.2.

Algorithm 7: $get\_SOR^{\mathrm{M}}(Z, V, F)$

Input: *Poly* Z, V, *CPoly* F
Output: *Poly* $SOR^{\mathrm{M}}(Z, V, F)$
foreach *CPoly* $P \in [\![Z]\!]$ **do**
  $Int_{new} \leftarrow UpdInt(Int_{new}, P, Z)$;
  $E \leftarrow PotentialEntry(P, Int_{new}, F)$;
  $Ext_{new} \leftarrow UpdExt(Ext_{new}, P, E, F, V)$;
while $Ext_{new} \neq \emptyset$ **do**
  $Ext_{old} \leftarrow Ext_{new}$;
  $Int_{old} \leftarrow Int_{new}$;
  $Ext_{new} \leftarrow \emptyset$;
  foreach P *s.t.* $\langle P, R\rangle \in Ext_{old}$ **do**
    $B \leftarrow \bigcup \{R \mid \langle P, R\rangle \in Ext_i\}$;
    $Cut \leftarrow P \cap (B_{\swarrow l})$;
    if $Cut \neq \emptyset$ **then**
      $P_{new} \leftarrow P \setminus Cut$;
      foreach $P' \in [\![P_{new}]\!]$ **do**
        $Int_{new} \leftarrow UpdInt(Int_{new}, P', P_{new})$;
      foreach P' *s.t.* $\langle P, P'\rangle \in Int_{old}$ **do**
        $Int_{new} \leftarrow UpdInt(Int_{new}, P', P_{new})$;
        $Ext_{new} \leftarrow UpdExt(Ext_{new}, P', Cut, F, V)$;
      $Int_{new} \leftarrow Int_{new} \setminus \{\langle P, Q\rangle \in Int_{old}\}$;
return $\{P \mid \langle P, P'\rangle \in Int_{new}\}$;

Algorithm 8: $UpdInt(Int, P, Candidates)$

Input: Set of *CPoly* pairs Int; *CPoly* P;
  Poly $Candidates$;
Output: Set of *CPoly* pairs Int;
$Int \leftarrow Int \cup \{\langle P, \emptyset\rangle\}$;
foreach *CPoly* $P' \in [\![Candidates]\!]$, *with* $P' \neq P$ **do**
  if $bndry(P, P') \neq \emptyset$ **then**
    $Int \leftarrow Int \cup \{\langle P, P'\rangle\}$;
return Int;

6.2.2 Further Improving the Performance (1)

Recall that $PotentialEntry(P, Int_0, F)$ returns $\bar{Z}$, regardless of its inputs. Experimental evidence (see Chapter 7) shows that $\bar{Z}$ is often a very large set of convex polyhedra. On the other hand, it is often the case that the portion of $\bar{Z}$ which is relevant to computing the entry regions of a given a convex polyhedron P is much smaller than the whole set $\bar{Z}$. This often leads to a large number of attempts to compute entry regions which end up empty. To

Algorithm 9: *UpdExt*$(Ext, P, Candidates, F, V)$

Input: Set of *CPoly* pairs *Ext*; *CPoly* P, F;
Poly $Candidates, V$;
Output: Set of *CPoly* pairs *Ext*;

```
if P ⊈ V then
    foreach CPoly P' ∈ [[Candidates]] do
        R ← entry(P, P');
        if R ≠ ∅ then
            Ext ← Ext ∪ {⟨P, R⟩};
return Ext;
```

avoid this, for each P in $[\![Z]\!]$ (see Figure 6.4(a)), another approach are now explained. The first step consists in computing the polyhedra P_{adj} (see Figure 6.4(b)) that contains P and all convex polyhedra in $[\![Z]\!]$ that are adjacent to it: $P_{adj} = \{P\} \cup \{P' \mid \langle P, P' \rangle \in Int_0\}$.

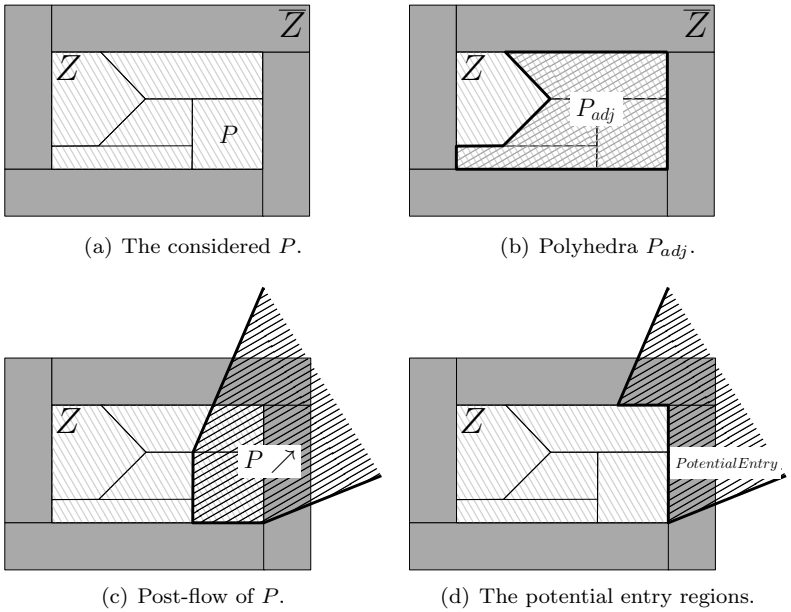
Then the polyhedra that contains all and only the convex polyhedra of $\overline{Z}$ which, if adjacent to P, contain a non empty entry region, is computed as

$$PotentialEntry(P, Int_0, F) = (P \nearrow F) \setminus P_{adj}.$$

Notice that, all the convex polyhedra in $[\![\bar{Z}]\!] \setminus PotentialEntry(P, Int_0, F)$ must have an empty entry region from P, since they cannot be reached from P following a straight-line activity in F. Moreover, all the convex polyhedra in $PotentialEntry(P, Int_0, F) \cap [\![Z]\!]$ cannot be adjacent to P, since, otherwise, they would belong to P_{adj} as well. Hence, all their entry regions from P are empty. Therefore, the resulting polyhedron $PotentialEntry(P, Int_0, F)$ contains all and only the convex polyhedra which, if adjacent to P, belong to $\bar{Z}$ and have a non-empty entry region from P.

Figure 6.4(d) shows the resulting polyhedra, computed by removing from $P \nearrow F$ (see Figure 6.4(c)) the polyhedra P_{adj}. The implementation based on the computation of $PotentialEntry(P, Int_0, F) = (P \nearrow F) \setminus P_{adj}$ is called the *local approach* in the following.

The comparison between *PotentialEntry* computed in global and in local approaches is shown in Figure 6.5, where the former is represented by the falling-pattern area while the latter is represented by the raining-pattern area: one can argue that (generally) the local approach works on a smaller set of *PotentialEntry* with respect to the global approach, and this improves the performance (see Chapter 7).



(a) The considered P. (b) Polyhedra P_{adj}.

(c) Post-flow of P. (d) The potential entry regions.

Figure 6.4: The local approach.

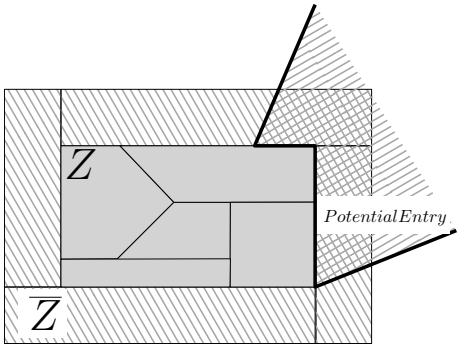


Figure 6.5: Comparision between global and local approach.

6.2.3 Further Improving the Performance (2)

Notice that each call to Algorithm 7 involves into the construction of completely news adjacency relations (performed by the first three instructions of the algorithm). Actually, after the first call to Algorithm 5, this can be avoided by using, for each location $l \in Loc$, the last relations carried out by Algorithm 7 that computes SOR_l^{M}. The idea behind this approach is now explained, but first some additional notions are required. First of all, each location $l \in Loc$ is associated to the last Int and Ext computed by SOR_l^{M}. The mapping Int_{rel} returns the associated internal relation to the location l, and similarly the mapping Ext_{rel} returns the associated external relation to the location l. Each call to Algorithm

7 after the first $CPre^{\mathrm{S}}$ (implemented by the Algorithm 5), instead of rebuild the relations, uses the relations $Int = Int_{rel}(l)$ and $Ext = Ext_{rel}(l)$. Notice that, once performed the first $CPre^{\mathrm{S}}$, Int contains adjacency between polyhedra belonging to A (by construction of algorithms, $A = W\lfloor_l = \{P \mid \langle P, P' \rangle \in Int_{old}\}$), while the new internal relation that Algorithm 7 would build, would contains the adjacency between polyhedra belonging to $\overline{Inv(l)} \cup (A \setminus B)$. The relation Int contains only the points of $\overline{Inv(l)}$ that were not removed by the last call to SOR_l^{M}. Hence, take into account the whole $\overline{Inv(l)}$, in the next step, is not necessary. Moreover, Int may also contain some points of B, because B can increase, by definition, and then B may overlap some polyhedron belongs to Int. This means that Int may contain not desired points (i.e. points from which it is possible to reach a bad area). But such points can be removed if one builds the Ext relation by calling, for each $P_1 \in \{P \mid \langle P, P' \rangle \in Int\}$, the Algorithm 9 on parameters $Ext_{rel}(l)$, P_1, B, F and V. By following this way, Algorithm 7 removes from Int all the "bad" points, obtaining the same results of the three previous approaches described above. In the remainder of the thesis, this new approach is called *local+* and requires a little bit different version of Algorithm 7 used after the first call to $CPre^{\mathrm{S}}$. In particular, this different version is carried out by removing the first two instructions, and by replacing one parameter of the function called in the third instruction (Algorithm 9). In particular, instead of $E = PotentialEntry(P, Int_{new}, F)$, one calls the algorithm on the parameter $E = B$.

An Alternative Implementation of the Function get_{fix_point}. Notice that, by associating to each location the last two computed polyhedra B (called B_{old} and B_{new}, resp.) and C (called C_{old} and C_{new}, resp.) by Algorithm 5, one can change the technique in order to check whether the fixpoint of the global algorithm is reached. Recall that the function $get_fixpoint$, called by Algorithm 5, simply verifies whether $W_i = W_{i+1}$. The last is a highly computational demanding operation, due the fact that it is performed on set of states. Actually, the fixpoint is reached when, for each $l \in Loc$, $B_{old} = B_{new}$ and $C_{old} = C_{new}$ (indicating that there is nothing more to do, i.e. all the bad points was removed). The check whether $B_{old} = B_{new}$ and $C_{old} = C_{new}$ are less expensive w.r.t. the check if $W_i = W_{i+q}$ because it is performed on polyhedra, instaed of set of states. This is achieved by using Algorithm 10, that is a different implementation of the function $get\_fixpoint$ (called by Algorithm 5).

Algorithm 10: $get\_fixpoint(W, W_{new}, Loc)$

Input: Set of states W, W_{new}, Set of locations Loc.
Output: Boolean *fix_point*.
forall the $(l \in Loc)$ **do**
  if $(B^l_{old} \neq B^l_{new})$ And $(C^l_{old} \neq C^l_{new})$ **then**
    return $false$;
return $true$;

6.3 Implementation of the Global Fixpoint for Reachability

Theorem 6 gives a fix-point characterization of the reachability control problem for linear hybrid games, that can be easily turned into the Algorithm 11.

Algorithm 11: $Reach(H, T)$

Input: Hybrid Game $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$, Set of States T.
Output: Set of winning states W.
Data: Set of states W_{new}, Boolean *fix_point*.
fix_point $:= false$;
$W := T$;
while $(fix\_point = false)$ **do**
  $W_{new} := T \cup CPre(W)$;
  if $(W_{new} = W)$ **then**
    fix_point $:= true$;
return W;

Given the linear hybrid game $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$ and the set of safe states T, the reachability control problem for H w.r.t. T can be solved by calling Algorithm 11 on parameters G and T. The heart of this algorithm involves into the computation of the controllable predecessor operator for reachability $CPre^{\mathrm{R}}$. Since, by Equation 4.2, $CPre^{\mathrm{R}}$ can be expressed as

$$CPre^{\mathrm{R}}(A) = InvS \cap \bigcup_{l \in Loc} \{l\} \times RWA^{\mathrm{M}}_l\big(A\downarrow_l \cup Pre_c(A)\downarrow_l \cup \overline{Inv(l)}, Pre_u(\overline{A})\downarrow_l \setminus A\downarrow_l\big),$$

then the implementation of $CPre^{\mathrm{R}}$, done by Algorithm 12, requires the computation of the operator RWA^{M}. By Theorem 7, the computation of RWA^{M} requires (i) the implementation of the RWA^{m} operator and (ii) the implementation of the operator RU (see Chapter 5) in order to correctly compute the over-approximation $Over$. For the first point, similarly to the case of the safety,

instead of RWA^{m}, the implementation is based on the computation of the dual operator SOR^{M}, that has been widely discussed in Section 6.2. Hence, Algorithm 12 calls Algorithm 7.

Algorithm 12: $get\_CPre^{\mathrm{R}}(H, W)$

Input: LHG $H = (Loc, X, Edg_{\mathrm{c}}, Edg_{\mathrm{u}}, Flow, Inv, Init)$, Set of States W.
Output: Set of States $CPre^{\mathrm{R}}(W)$.
Data: *Poly* A, B, C, $Over$, $Under$.
$CPre^{reach} := \emptyset$;
foreach $(l \in Loc)$ **do**
  $A := W\lfloor_l$;
  $B := \emptyset$;
  $C := \emptyset$;
  foreach $(t \in Edg_{\mathrm{c}} \mid t = \langle l, \mu, l' \rangle)$ **do**
    $C := C \cup get\_pre(W\lfloor_{l'}, Inv(l'), \mu)$;
  foreach $(t \in Edg_{\mathrm{u}} \mid t = \langle l, \mu, l' \rangle)$ **do**
    $C := C \cup get\_pre(\overline{W\lfloor_{l'}}, Inv(l), \mu)$;
  $Under := A \setminus B$;
  $Over := Under \cup get\_\mathrm{RU}(\overline{A} \cap \overline{B})$;
  $CPre^{reach} := CPre^{reach} \cup \{l, Over \cap get\_SOR^{\mathrm{M}}(Over, C)\}$;
return $CPre^{reach}$;

Notice that, for the sake of efficiency, the called Algorithm 7 is whose that use the local approach. The local+ approach can not be used, because too specific for the safety objectives (see Section 6.2.3)

About the operator RU, implemented by Algorithm 13, Equation 5.4 shows that it is based on the positive pre-flow operator (see Chapter 4) and on the additional operator *is_bounded*. Algorithm 14 shows the effective implementation of the operator *is_bounded* that, given in input the polyhedron G and the convex polyhedron F, test whether P is bounded w.r.t. F (see Theorem 10).

Algorithm 13: $get\_\mathrm{RU}(G, F)$

Input: *Poly* G, *CPoly* F.
Output: *Poly* $\mathrm{RU}(G, F)$.
$G_{new} := \emptyset$;
foreach *CPoly* $P \in [\![G]\!]$ **do**
  if $is\_bounded(P, F) = true$ **then**
    $G_{new} := G_{new} \cup P$;
return G_{new};

The first step performed by the algorithm is checking whether the origin **0** belongs to F: if the answer is positive, then it returns false (that means that P

Algorithm 14: $is\_bounded(P, F)$

Input: *CPoly* P, F.
Output:
if $\mathbf{0} \in F$ **then**
  return $false$;
$P_{rays} = get\_rays(P)$;
$O_P = \mathbf{0}$;
foreach *CPoly* $r \in P_{rays}$ **do**
  $O_P = O_P.add\_ray(r)$;
if $(O_P \cap F) = \emptyset$ **then**
  return $true$;
return $false$;

is not bounded). Otherwise, it checks whether the characteristic cone O_P and F are disjoint: in this case the Algorithm returns true (that means that P is bounded), else returns again false. Notice that the called function $get\_rays(P)$ is a common operation over convex polyhedron that return the extremal rays of the convex specified as parameter. In a similar manner, the called method $.add\_rays(r)$ is a common operation over convex polyhedron in order to add the ray specified as parameter to the convex polyhedron. Both function and method are provided by the PPL library.

All the operators seen in this chapter are implemented on the top of the tool PHAVer. Now, by the extensions described in the thesis, the obtained tool PHAVer+ can be used also for the task of the synthesis, and the main contribution of this thesis is achieved.

Next Chapter is dedicated to several experiments on the use of PHAVer+, tested on some examples, both for the safety control problem and the reachability control problem.

Chapter 7

Experiments with PHAVer+

This chapter shows some experiments on safety and reachability control problem, performed with several implementations of the procedures described in Chapter chap:impl. The synthesis procedures are been implemented on the top of the open-source tool PHAVer [Fre05]. For the purpose of evaluating the present thesis, a binary pre-release of this implementation, called PHAVer+, can be downloaded at `http://people.na.infn.it/minopoli/phaver`. The experiments were performed on an Intel Xeon (2.80GHz) PC.

7.1 Macro Analysis

Truck Navigation Control (adapted from [DMT$^+$01]). Consider an autonomous toy truck, which is responsible for avoiding some 2 by 1 rectangular pits. The truck can take 90-degrees left or right turns: the possible directions are North-East (NE), North-West (NW), South-East (SE) and South-West (SW). One time unit must pass between two changes of direction. The control goal consists in avoiding the pits. Notice that the *TNC* proposed in [DMT$^+$01] is limited to one turn only, while our analysis is extended to the complete case (an unlimited number of turns is allowed). Figure 7.1 shows the linear hybrid game that models the system: there is one location for each direction, where the derivative of the position variables (x and y) are set according to the corresponding direction. The variable t represents a clock ($\dot{t} = 1$) that enforces a one-time-unit wait.

Figure 7.2 shows the three iterations needed to compute the fixpoint in Theorem 2, in the case of two pits. The safe set is the white area, while the gray region contains the points wherefrom all admitted activities leads into the

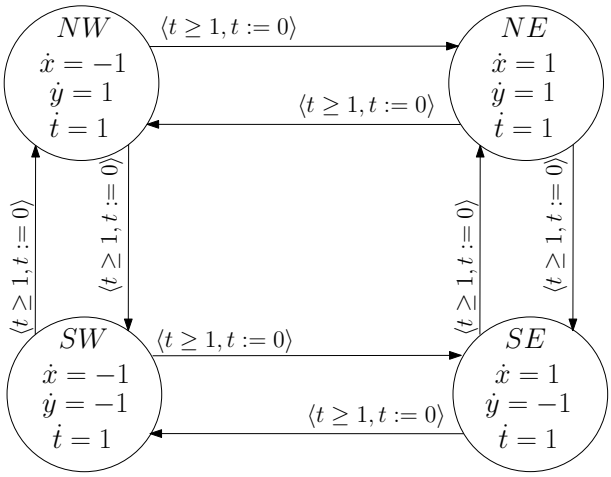


Figure 7.1: TNC modeled as LHGs.

obstacle (pit) and then are not winning for the controller.

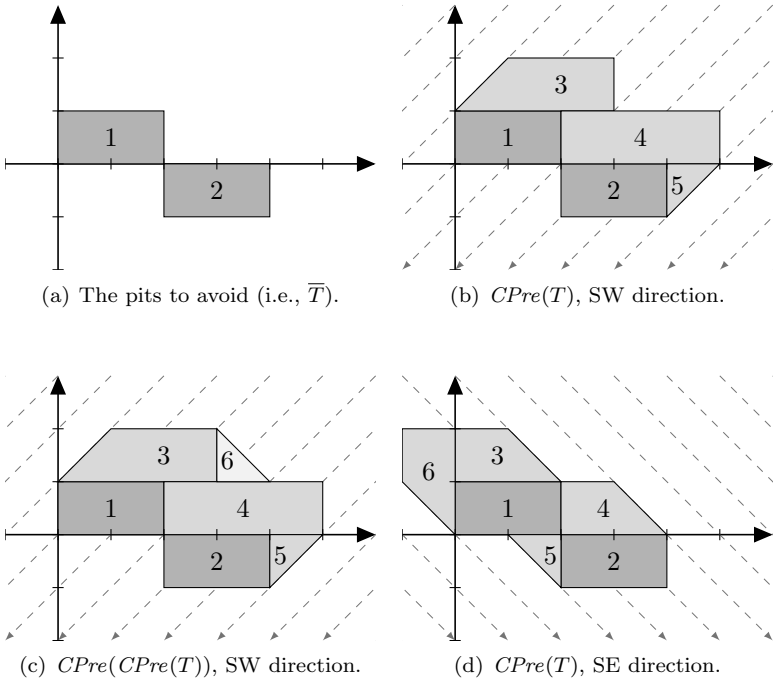
The input safe region T is the area outside the gray boxes 1 and 2 in Figure 7.2(a). The first iteration (Figure 7.2(b)) computes $CPre(T)$ and extends the unsafe set to those points (areas 3, 4, and 5) that will inevitably flow into the pits, before the system reaches $t = 1$ and the truck can turn. The second iteration (Figure 7.2(c)) computes $CPre(CPre(T))$ and extends the unsafe set by adding the area 6: those points may turn before reaching the pits, but after the turn they end up in $CPre(T)$ anyway (for instance, if turning left, they end up in area 4 of Figure 7.2(d)). The third iteration reaches the fixpoint.

The implementations was tested on progressively more complex control goals, by increasing the number of obstacles. Table 7.1 shows the run time required by the different approaches, i.e. basic, global and local, in order to solve the safety control problem for TNC, up to 10 obstacles.

Table 7.1: Performance with respect to number of pits

Pits num.:	2	3	4	5	6	7	8	9	10
Basic	5,2	26,9	121,7	206,6	607,9	1177,3	2328,4	4390,7	6047,5
Global	4,5	14,5	44,6	71,9	127,8	217,8	386,0	423,4	647,5
Local	3,1	9,6	23,0	38,8	57,0	86,5	124,7	132,8	177,3
Local+	1,0	2,3	4,4	9,1	14,8	24,8	34,8	58,8	78,1

The graph shown in Figure 7.3(a) is a rapid way to compare the run time required by the different implementations of the algorithm. In particular, the thin solid line represents the run times for the basic approach, the thin dotted line represents the run times for the global approach, the harder solid line is for



(a) The pits to avoid (i.e., $\overline{T}$). (b) $CPre(T)$, SW direction.

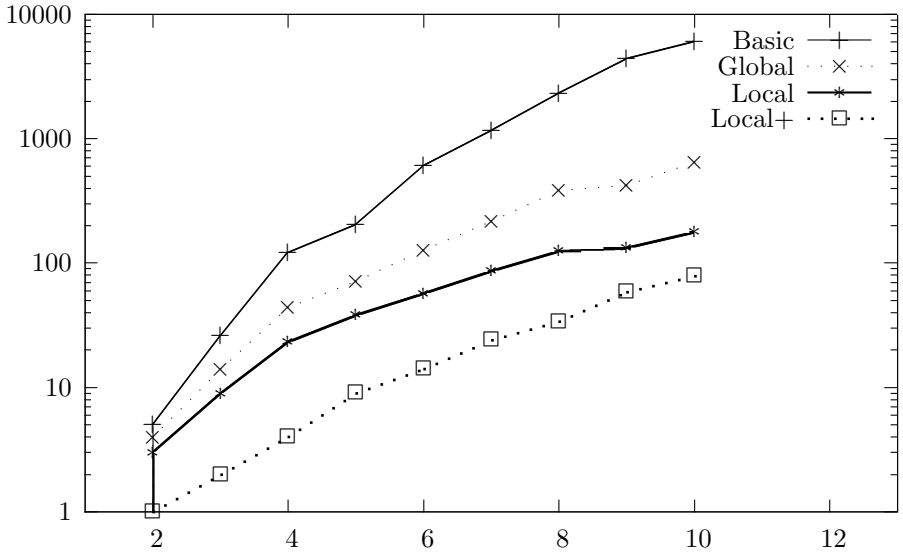
(c) $CPre(CPre(T))$, SW direction. (d) $CPre(T)$, SE direction.

Figure 7.2: Evolution of the fixpoint in the case of two pits. All figures are cross-sections for $t = 0$. Dashed arrows represent flow direction.

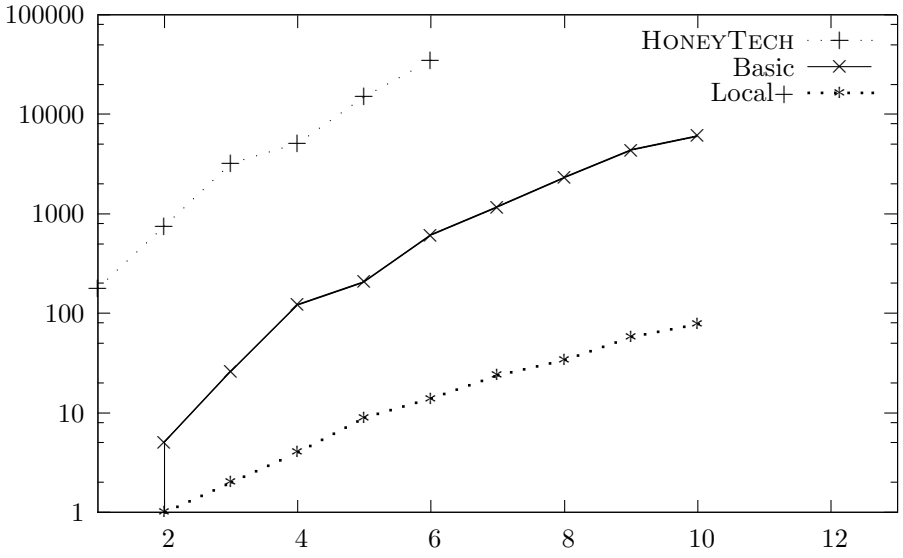
the local approach and the hard dotted line is for the local+ approach.

Moreover, the performance of the basic and local+ implementations have been compared with the implementation given in [DMT+01]. Figure 7.3(a) depicts these performances comparison, where the thin dotted line represents the performance reported in [DMT+01]), the solid line stay for the basic implementation and the dotted line stay for the local+ approach. Since HoneyTech is not publicly available, it was no possible to replicate the experiments in [DMT+01]. Notice that the time axis is logarithmic. Because of the different hardware used, only a qualitative comparison can be made: going from 1 to 6 pits (as the case study in [DMT+01]), the run time of HoneyTech shows an exponential behavior, while PHAVer+ (in particular the local+ approach) exhibits an approximately linear growth, as shown in Figure 7.3(b), where the performance of PHAVer+ is plotted up to 10 pits.

Truck Navigation Control with Non-Deterministic Flow. The TNC version described above is defined only by a deterministic flow, according to the work in [DMT+01] (see Figure 7.4(a)). In this thesis a version of TNC with



(a) Run times for TNC required by the different implementations.



(b) Comparision with HoneyTech performance on TNC.

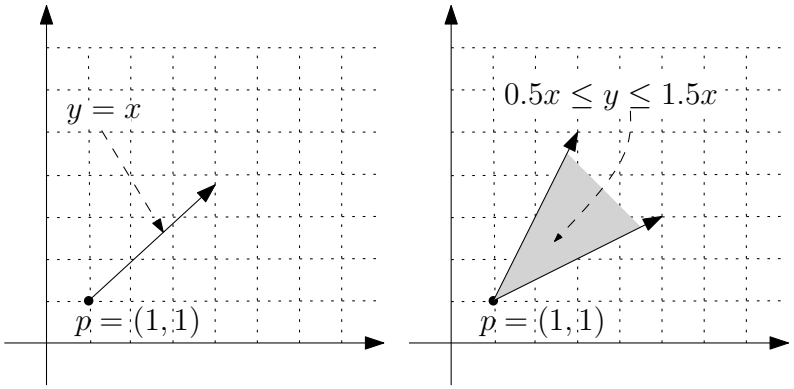
Figure 7.3: Performance for different implementations.

non-deterministic continuous flow, was also considered. In such version, the possible directions of the truck, in a given location, are expressed by differential equations as following:

- SE location: $\dot{x} = 1$, $-1.5 \leq \dot{y} \leq -0.5$.
- SW location: $\dot{x} = -1$, $-1.5 \leq \dot{y} \leq -0.5$.

- NE location: $\dot{x} = 1$, $0.5 \leq \dot{y} \leq 1.5$.
- NW location: $\dot{x} = -1$, $0.5 \leq \dot{y} \leq 1.5$.

Notice that, such flow allows some uncertainty on the exact direction taken by the vehicle, as shown in Figure 7.4(b) for the NE direction.



(a) The only allowed trajectory in the deterministic case for NE direction.

(b) The several allowed trajectories in the non-deterministic case for NE direction.

Figure 7.4: Deterministic and not-deterministic flow allowed by the two different versions of TNC.

Table 7.2: Performance for non-deterministic TNC with respect to number of pits

Pits num.:	1	2	3	4	5
Basic	0.5	9.3	51,8	241.1	580.0
Global	0,6	8,1	35,0	107,6	268,6
Local	0,6	5,0	15,5	46,4	69,2
Local+	0,2	1,4	3,5	8,9	17,7

Table 7.2 shows the run time required by the different approaches in order to solve the safety control problem for the non-deterministic TNC, up to 5 obstacles.

Three Dimensional Truck Navigation Control. This thesis also introduces a three dimensional extension of the TNC proposed in [DMT+01]. The considered obstacles are some rectangular box whose x, y and z dimensions are respectively 2, 2 and 1. The 90-degrees left or right turns are extended by also taking into account the vertical direction (up and down). Hence, the possible directions are Up-North-East (UNE), Up-North-West (UNW), Up-South-East (USE), Up-South-West (USW), Down-North-East (DNE), Down-North-West

(DNW), Down-South-East (DSE), Down-South-West (DSW). Also in this version, one time unit must pass between two changes of direction. The LHG that model the system has one location for each direction, where the derivative of the position variables (x, y and z) are set according to the corresponding direction. The variable t represents a clock ($\dot{t} = 1$) that enforces a one-time-unit wait. In such version, the possible directions of the truck, in a given location, are expressed by differential equations as following:

- DSE location: $\dot{x} = 1$, $\dot{y} = -1$, $\dot{z} = -1$.
- DSW location: $\dot{x} = -1$, $\dot{y} = -1$, $\dot{z} = -1$.
- DNE location: $\dot{x} = 1$, $\dot{y} = 1$, $\dot{z} = -1$.
- DNW location: $\dot{x} = -1$, $\dot{y} = 1$, $\dot{z} = -1$.
- USE location: $\dot{x} = 1$, $\dot{y} = -1$, $\dot{z} = 1$.
- USW location: $\dot{x} = -1$, $\dot{y} = -1$, $\dot{z} = 1$.
- UNE location: $\dot{x} = 1$, $\dot{y} = 1$, $\dot{z} = 1$.
- UNW location: $\dot{x} = -1$, $\dot{y} = 1$, $\dot{z} = 1$.

Table 7.3 shows how many time (in seconds) the local and the local+ approaches require to solve the three-dimensional TNC problem.

Table 7.3: Performance for three-dimensional TNC with respect to number of pits

Pits num.:	2	3	4	5	6	7	8
Local	126,9	289,2	529,8	1240,5	1853,2	2552,5	3692,5
Local+	33,8	80,8	142,7	246,9	359,3	539,6	650,4

Water Tank Control (adapted from [LLL09]). Consider a system where two tanks — A and B — are linked by a one-directional valve *mid* (from A to B). There are two additional valves: the valve *in* to fill A and the valve *out* to drain B. The two tanks are open-air: the level of the water inside also depends on the potential rain and evaporation. It is possible to change the state of one valve only after one second since the last valve operation. Figure 7.5 is a schematic view of the system.

The *in* and *mid* flow rate, p_{in} and p_{mid} resp., are set to 1, the *out* flow rate p_{out} to 3, the maximum evaporation rate b to 0.5 and maximum rain rate a to 1. The goal consists in to solve the synthesis problem for the safety specification

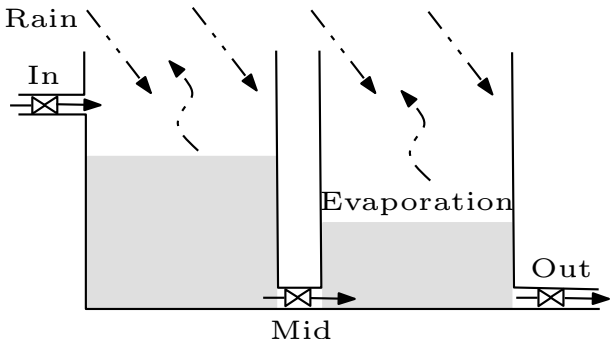


Figure 7.5: Schema of the system.

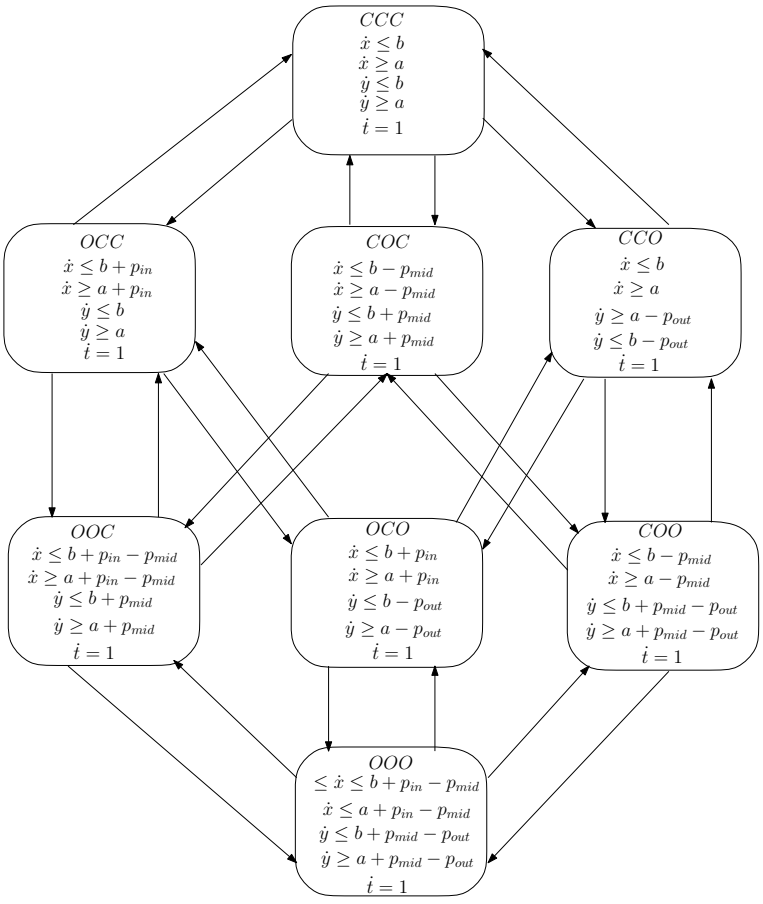
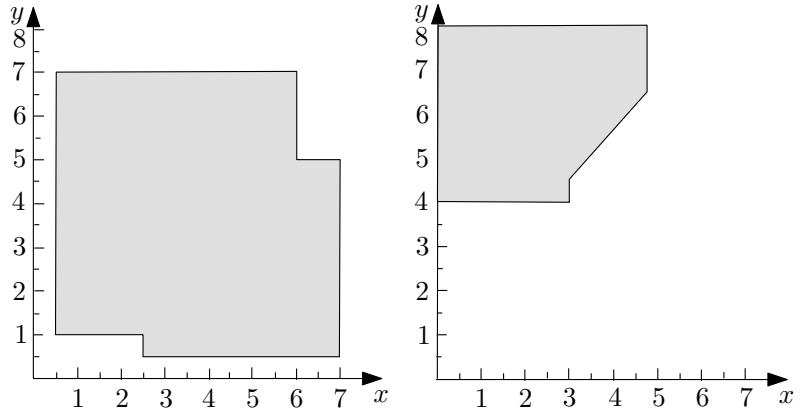


Figure 7.6: Two tanks modeled as LHG.

requiring the water levels to be between 0 and 8. The corresponding hybrid game, depicted in Figure 7.6 has eight locations, one for each combination of the state (open/closed) of the three valves, and three variables: x and y for the water level in the tanks, and t as the clock that enforces a one-time-unit

wait between consecutive discrete transitions. Since the tanks are in the same geographic location, rain and evaporation are assumed to have the same rate in both tanks, thus leading to a proper LHG, that is not rectangular (see Chapter 3 and [HHM99]).

Figure 7.7(a) (resp., 7.7(b)) shows the fixpoint result in the case of all valves closed (resp., *in* and *out* open and *mid* closed). Due to the necessity of one second wait before taking a discrete action, in the case of Figure 7.7(a), x and y must be between 0.5 and 7: otherwise, for example with a level greater than 7 and maximum rain, after one second the level will exceed the limit. In a similar way, with a level less than 0.5 and maximum evaporation, after one second the level would go below the lower bound. The result is computed after 5 iterations in 11 seconds, using the local+ approach.



(a) Result for all valves closed and $t = 0$. (b) Result with only *mid* valve closed and $t = 0$.

Figure 7.7: Output for Water Tank Control example.

The maze example. A vehicle navigates in a spiral-shaped labyrinth, by taking 90-degree left or right turns: the possible directions are North (N), South (S), West (W) and East (E). One time unit (say, second) must pass between two changes of direction, while the vehicle speed is 2 unit of length per second. The corridors of the maze are 1 unit wide, so that the vehicle can never u-turn without hitting a wall. The goal consists in reaching a target area positioned along the corridors. The implementation based on the local approach, was tested on progressively more complex mazes, by increasing the number of corridors (the angle between consecutive corridors is 90-degrees). The r.h.s. of the Figure 7.1 shows the shape of the maze: in the case of the first two corridors, the target is T_1. In the case of three, the target is T_2, and so on. The LHG modeling

the system contains three continuous variables: x and y for the position of the vehicle, and a clock t, to enforce the wait between consecutive turns. The walls are modeled by uncontrollable transitions leading to a sink location.

The l.h.s. of the Figure 7.1 shows the section of the solution for $t = 0$, in the case of a maze with two corridors and the vehicle initially going along the *North* direction: the light-gray areas (A and B) contain the points that can reach the target T_1. If the vehicle is located in A, it can reach the target by turning *East* and then *North* again. Notice that the area A covers only half the width of the vertical corridor. In fact, if the vehicle is located in the other half of the corridor, when turning *East* it will be too close to the target and will not be able to take the second turn towards the target in time. The area A ends 2 units of length before the north wall, as beyond that the vehicle cannot avoid hitting the wall before being able to turn *East*. Finally, the points in the area B are trivially winning, as they can reach the target by proceeding *North*. The example of maze described above was also extended to a three-dimensional version. In addition to the horizontal directions of the 2D case, in the 3D version the vehicle can perform 90-degree turns upwards (UP) and downwards (DOWN). The resulting LHG model includes two additional locations to move up and down, and one additional continuous variable z for the position of the vehicle along the third dimension, for a total of four variables. Table 7.4 shows the run time in seconds for the two different versions of maze of increasing size, in terms of number of corridors and targets along them. Although still limited in scope, the results show that the proposed approach is practical, at least for relatively small problems.

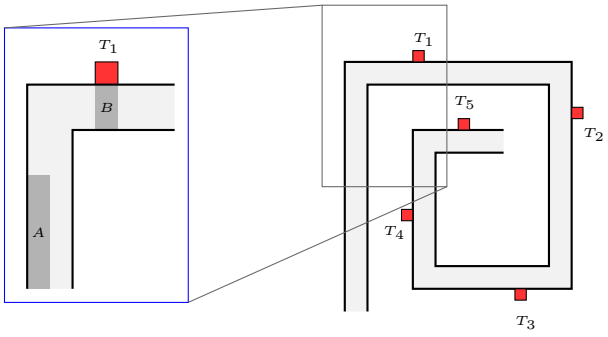


Figure 7.8: Structure of the maze.

# of corridors	Time (sec.) 2D	Time (sec.) 3D
3	0.9	2.3
5	5.9	11.8
7	24.7	42.1
9	85.6	137.7

Table 7.4: Performance.

7.2 Micro Analysis

This section shows the behavior of individual calls to $SOR^{\mathrm{M}}(Z, V)$, implemented by using basic, global and local approach described in Section 6.2 of this chapter. The local+ approach was not take into account, because the corresponding implementation involves in a different implementation of the global fixpoint algorithm (in particular, in a different check on the fixpoint). The evaluation of the efficiency of the three versions is carried out based on the number of comparisons that the three algorithms perform in order to identify the boundaries between polyhedra in Z and polyhedra in *PotentialEntry*, with respect to the size of the input, i.e. $|[\![Z]\!]| + |[\![V]\!]|$. The choice to to highlight the number of computed boundaries derives from the idea that led to the realization of the local version of the algorithm is precisely to avoid unnecessary adjacency checks (hence the idea of introducing the adjacency relations).

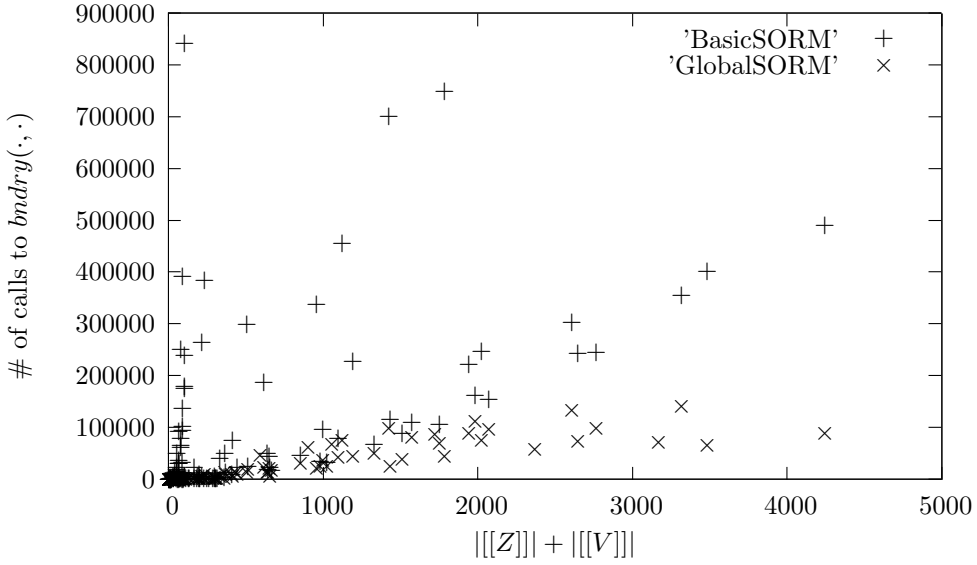


Figure 7.9: Number of boundary checks of basic and global algorithms for SOR^{M} w.r.t. the size of the input.

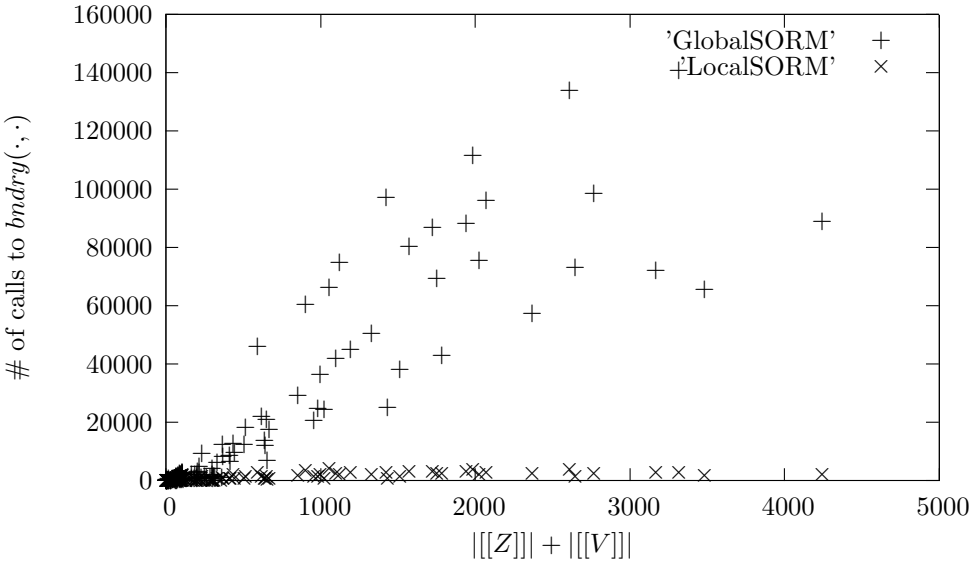


Figure 7.10: Number of boundary checks of global and local algorithms for SOR^{M} w.r.t. the size of the input.

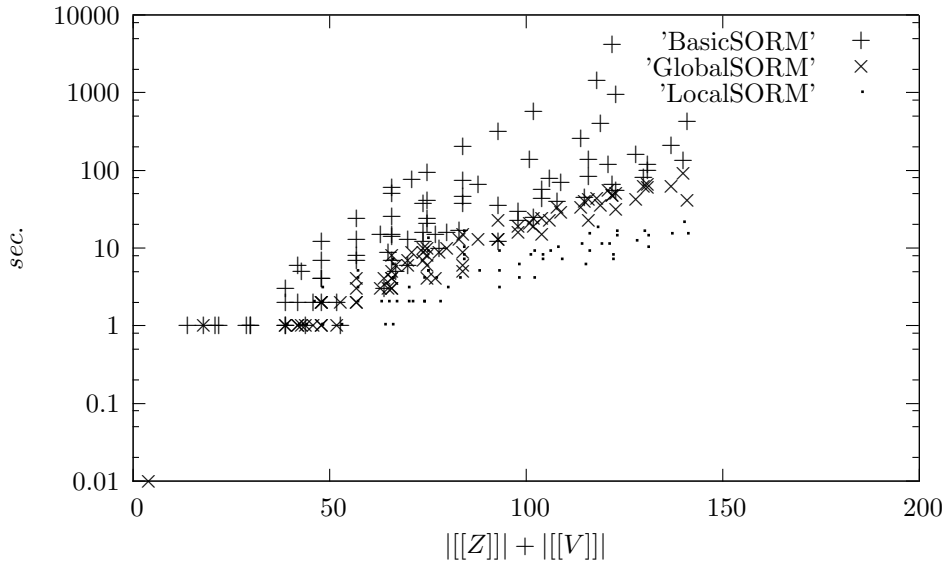


Figure 7.11: Run time (in sec.) of algorithms basic, global and local for SOR^{M} w.r.t. the size of the input.

Figure 7.9 and Figure 7.10 shows the number of boundary computations made by the three approaches. As expected, the number of calls made by the basic algorithm is higher than those made by the global approach (see Figure7.9), which in turn is higher then those made by the algorithm that implements the

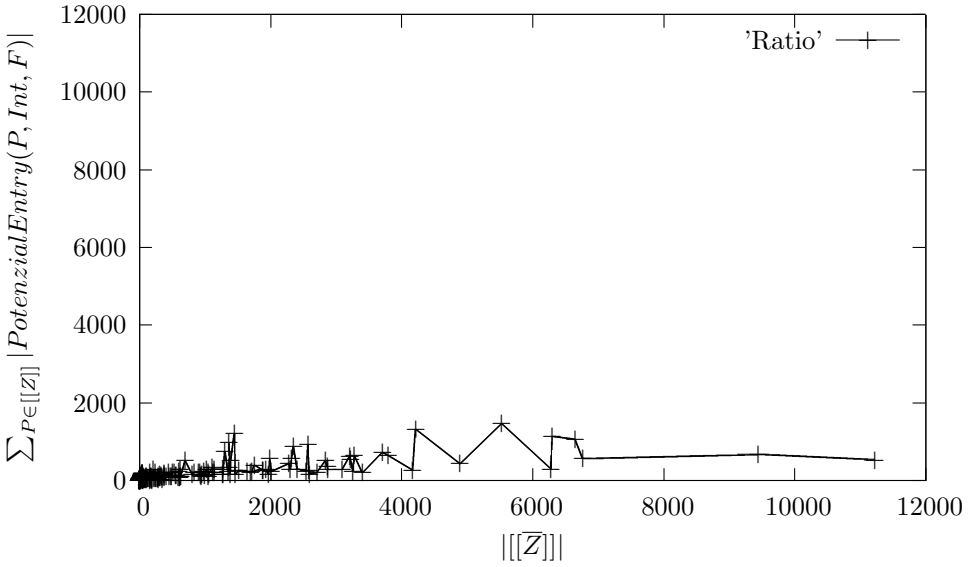


Figure 7.12: Size of *PotentialEntry* in the global and the Local algorithms.

local approach (see Figure7.10). This is reflected in the execution times of the three procedures, as shown in Figure 7.11.

One also notices a certain instability in the case of the basic algorithm, due to the fact that in some instances of the problem, even with small inputs, the algorithm can cut an individual polyhedron in many parts: this dramatically increases the size of the sets Z and $\bar{Z}$ in the next steps and consequently the number of comparisons required. This instability is held much more under control with the introduction of the adjacency relations, that potentially preventing unnecessary checks. Note that in the local version the number of comparisons required is much lower: this fact can easily explain, recalling that *PotentialEntry* in the global version returns the whole $\bar{Z}$, forcing Algorithm 9 to perform $|\bar{Z}|$ iterations of its "foreach" loop.

Figure 7.12 shows, for the same inputs, the relationship between the size of *PotentialEntry* in the basic and in the global versions (i.e., $\bar{Z}$) and in the local version: the ratio is 1 to 10, which reduces drastically the number of checks, and consequently the overall run time.

Conlusion and Future Works

In this thesis, the problem of automatically synthesizing a switching controller for an LHG w.r.t. safety objectives is revisited and the same problem w.r.t. reachability objectives is introduced.

For the safety goal, the synthesis procedure is based on the RWA^{m} operator, for which a novel fixpoint characterization is proposed and whose termination is formally proved.

For the reachability goal, the synthesis procedure is based on the RWA^{M} operator, for which a novel fixpoint characterization is proposed and whose termination is formally proved.

To the best of our knowledge, these represents the first sound and complete procedures for the tasks in the literature.

The open source tool PHAVer was extended with the synthesis procedures showed in the thesis, and a series of promising experiments are performed and shown.

One of the possibile future work is to work on the automatic construction of a concrete control strategy, which, coupled with the hybrid system, would result in a closed system, amenable to automatic verification by state-of-the-art analysis tools. Another interested future scenario could be to focus on a more general technique involves modifying the goal in order to take the Zenoness phenomenon into account.

We leave it to future work to combine our results with more sophisticated approaches to Zenoness known in the literature [BBV$^+$03, dAFH$^+$03].

Bibliography

[ABD+00] E. Asarin, O. Bournez, T. Dang, O. Maler, and A. Pnueli. Effective synthesis of switching controllers for linear systems. *Proceedings of the IEEE*, 88(7):1011–1025, 2000.

[ABM04] R. Alur, M. Bernadsky, and P. Madhusudan. Optimal reachability for weighted times games. In *31th International Colloquium on Automata, Languages and Programming*, volume 3142 of *Lecture Notes in Computer Science*. Springer, 2004.

[ACD93] R. Alur, C. Courcoubetis, and D. Dill. Model-checking in dense real-time. *Information and Computation*, 104:2–34, 1993.

[AD94] L. Alur and D. L. Dill. A theory of timed automata. *Theoretical Computer Science*, 126:183–235, 1994.

[ADM02] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Proceedings of the 14th International Conference on Computer Aided Verification*, CAV '02, pages 365–370, London, UK, UK, 2002. Springer-Verlag.

[AHH96] R. Alur, T.A. Henzinger, and P.-H. Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22, 1996.

[AHK97] R. Alur, T.A. Henzinger, and O. Kupferman. Alternating-time temporal logic. In *Symposium on Foundations of Computer Science*, pages 100–109. IEEE Computer Society Press, 1997.

[ALW89] M. Abadi, L. Lamport, and P. Wolper. Realizable and unrealizable concurrent program specifications. In *Proceeding of the 16th Int. Colloq. Aut. Lang. Prog*, volume 372 of *Lecture Notes in Computer Science*, pages 1–17. Springer, 1989.

[AM99] E. Asarin and O. Maler. As soon as possible: Time optimal control for timed automata. In *Proceeding of the 2nd International Workshop on Hybrid Systems: Computation and Control*, volume 1569 of *Lecture Notes in Computer Science*, pages 19–31. Springer, 1999.

[AMP95] E. Asarin, O. Maler, and A. Pnueli. Symbolic controller synthesis for discrete and timed systems. In *Hybrid Systems II*, volume 999 of *Lecture Notes in Computer Science*, pages 1–20. Springer, 1995.

[BB95] T. Basar and P. Bernhard. *H-Infinity Optimal Control and Related Minimax Design Problems*. Springer-Verlag, 1995.

[BBM98] M.S. Branicky, V.S. Borkar, and S.K. Mitter. A unified framework for hybrid control: Model and optimal control theory. *IEEE transaction on automatic control*, 43:31–45, 1998.

[BBV+03] A. Balluchi, L. Benvenuti, T. Villa, H. Wong-Toi, and A. Sangiovanni-Vincentelli. Controller synthesis for hybrid systems with a lower bound on event separation. *International Journal of Control*, 76(12):1171–1200, 2003.

[BCC+06] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *Proceeding of the international symposium on mathematical theory of networks and systems*, 2006.

[BCD+05] G. Behrmann, A. Cougnard, R. David, E. Fleury, K.G. Larsen, and D. Lime. UPPAAL-TIGA: Timed games for everyone. In *In Proceeding of the 17th Nordic Workshop on Programming Theory*, 2005.

[BDM+98] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. KRONOS: A model-checking tool for real-time systems. In *Computer Aided Verification*, pages 546–550, 1998.

[BFH+94] A. Bouajjani, J-C. Fernandez, N. Halbwachs, P. Raymond, and C. Ratel. Minimal state graph generation. In *Science of Computer Programming*, 1994.

[BFM11a] M. Benerecetti, M. Faella, and S. Minopoli. Revisiting synthesis of switching controllers for linear hybrid systems. In *Proceeding of the 50th IEEE Conference on Decision and Control (CDC 2011)*, 2011. To appear.

[BFM11b] M. Benerecetti, M. Faella, and S. Minopoli. Towards efficient exact synthesis for linear hybrid systems. In *Proceeding of the 2th International Symposium on Games, Automata, Logics and Formal Verification*, pages 263–277, 2011.

[BFM12] M. Benerecetti, M. Faella, and S. Minopoli. Reachability games for linear hybrid systems. In *Proceeding of The 15th International Conference on Hybrid Systems: Computation and Control (HSCC 2012)*, 2012. Submitted to.

[BHZ08] R. Bagnara, P. M. Hill, and E. Zaffanella. The Parma Polyhedra Library: Toward a complete set of numerical abstractions for the analysis and verification of hardware and software systems. *Science of Computer Programming*, 72(1–2):3–21, 2008.

[BL69] J.R. Buchi and L.H. Landweber. Solving sequential conditions by finite state strategies. In *Proceeding of American Mathematial Society*, pages 295–311, 1969.

[BLMR06] P. Bouyer, K.G. Larsen, N. Markey, and J.I. Rasmussen. Almost optimal strategies in one clock priced timed games. In *In Proceeding of the 26th International Conference on Foundations of Software Technology and Theoretical Computer Science*, pages 345–356, 2006.

[BLP+96] J. Bengtsson, F. Larsson, P. Pettersson, W. Yi, P. Christensen, J. Jensen, P. Jensen, K. Larsen, and T. Sorensen. UPPAAL: a tool suite for validation and verification of real-time systems. In *Proceedings of the DIMACS/SYCON workshop on Hybrid systems III : verification and control*, pages 232–243, Secaucus, NJ, USA, 1996. Springer-Verlag New York, Inc.

[BO97] T. Basar and G.J. Olsder. Dynamic noncooperative game theory, 2nd ed. *Journal of Economic Dynamics and Control*, 21(6):1113–1116, 1997.

[Bou06] P. Bouyer. Weighted timed automata: model-checking and games. *Electronic Notes in Theoretical Computer Science*, 158:2006, 2006.

[BS89] R. Back and K. Sere. Stepwise refinement of action systems. In *Mathematics of Program Construction*, volume 375 of *Lecture Notes in Computer Science*, pages 115–138. Springer Berlin / Heidelberg, 1989.

[BT00] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Proceedings of the 3th International Workshop on Hybrid Systems: Computation and Control*, HSCC '00, pages 73–88, London, UK, 2000. Springer-Verlag.

[CDF+05] F. Cassez, A. David, E. Fleury, K.G. Larsen, and D. Lime. Efficient on-the-fly algorithms for the analysis of timed games. In *Conference on concurrency theory*, volume 3653 of *Lecture Notes in Computer Science*, pages 66–80. Springer, 2005.

[CES86] E.M. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8:244–263, 1986.

[Che68] N. V. Chernikova. Algorithm for discovering the set of all the solutions of a linear programming problem. *USSR Computational Mathematics and Mathematical Physics*, 8(6):282–293, 1968.

[Chu62] A. Church. Logic, arithmetic and automata. In *Proceeding International Congress of Mathematicians*, pages 23–35, Djursholm, Sweden, 1962. Inst. Mittag-Leffler.

[CPPSV06] L.P. Carloni, R. Passerone, A. Pinto, and A.L. Sangiovanni-Vincentelli. Languages and tools for hybrid systems design. *Found. Trends Electron. Des. Autom.*, 1:1–193, 2006.

[CW98] P.E. Caines and Y.J. Wei. Hierarchical hybrid control systems: a lattice theoretic formulation. *IEEE Transaction on automatic control*, 43:501–508, 1998.

[dAFH+03] L. de Alfaro, M. Faella, T.A. Henzinger, R. Majumdar, and M. Stoelinga. The element of surprise in timed games. In *Proceeding of the 14th International Conference con Concurrency Theory*, volume 2761 of *Lecture Notes on Computer Science*, pages 144–158. Springer, 2003.

[DJW97] S. Dziembowski, M. Jurdzinski, and I. Walukiewicz. How much memory is needed to win infinite games? In *Proceedings of the 12th Annual IEEE Symposium on Logic in Computer Science*, LICS '97, pages 99–110, 1997.

[DMT+01] R.G. Deshpande, D.J. Musliner, J.E. Tierno, S.G. Pratt, and R.P. Goldman. Modifying HYTECH to automatically synthesize hybrid controllers. In *40th IEEE Conference on Decision and Control*, pages 1223–1228. IEEE Computer Society Press, 2001.

[DOTY96] C. Daws, A. Olivero, S. Tripakis, and S. Yovine. The tool KRONOS. In *Proceeding of Hybrid Systems III*, volume 1066 of *Lecture Notes in Computer Science*, pages 208–219. Springer Verlag, 1996.

[EJ91] E. A. Emerson and C. S. Jutla. Tree automata, mu-calculus and determinacy. In *Proceedings of the 32nd annual symposium on Foundations of computer science*, SFCS '91, pages 368–377, Washington, DC, USA, 1991. IEEE Computer Society.

[Far02] B. Farwer. ω automata. In *Automata, logics and infinite games*, pages 3–22. Springer-Verlag, London, UK, 2002.

[Fre05] G. Frehse. PHAVER: Algorithmic verification of hybrid systems past HYTECH. In *Proceeding of the 8th International Workshop on Hybrid Systems: Computation and Control (HSCC)*, volume 3414 of *Lecture Notes on Computer Science*, pages 258–273. Springer, 2005.

[GH82] Y. Gurevich and L. Harrington. Trees, automata, and games. In *Symposium on Theory of Computing (STOC)*, pages 60–65, 1982.

[Gra99] G. Grammel. Maximum principle for a hybrid system via singular perturbations. *SIAM J. on Control and Optimization*, 37:1162–1175, 1999.

[GZ05] H. Gimbert and W. Zielonka. *Games where you can play optimally without any memory*, pages 428–442. Springer-Verlag, London, UK, 2005.

[Hen96] T.A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science*, pages 278–292, New Brunswick, New Jersey, 1996. IEEE Computer Society Press.

[HHM99] T.A. Henzinger, B. Horowitz, and R. Majumdar. Rectangular hybrid games. In *Proceeding on the 10th international conference on concurrency theory*, volume 1664 of *Lecture Notes on Computer Science*, pages 320–335. Springer, 1999.

[HHWT95] T.A. Henzinger, P.H. Ho, and H. Wong-Toi. HyTech: The next generation. In *In Proceedings of the 16th IEEE Real-Time Systems Symposium*, pages 56–65. IEEE Computer Society press, 1995.

[HKPV95] T.A. Henzinger, P.W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Symposium on Theory of Computing*, STOC '95, pages 373–382. ACM, 1995.

[HM97] F.L. Heymann and G. Mayer. Controller synteshis for a class of hybrid systems subject to configuration-based safety constraints. *Hybrid and RealTime Systems*, 1201:376–391, 1997.

[HNSY92] T.A. Henzinger, X. Nicollin, J. Sifakis, and S. Yovine. Symbolic model checking for real-time systems. *Information and Computation*, 111:394–406, 1992.

[HPR97] N. Halbwachs, Y.-E. Proy, and P. Roumanoff. Verification of real-time systems using linear relation analysis. *Formal Methods in System Design*, 11:157–185, 1997.

[HWT92a] G. Hoffman and H. Wong-Toi. Symbolic synthesis of supervisory controllers. In *Proceeding of the American Control Conference*, pages 2789 – 2793, Chicago, IL, 1992. IEEE Computer Society Press.

[HWT92b] G. Hoffmann and H. Wong-Toi. The input-output control of real-time discrete event systems. In *Proceeding of the 13th Real-Time Systems Symposium*, pages 256 – 265, Phoenix, AZ, 1992. IEEE Computer Society Press.

[JRLD07] J.J. Jessen, J.I. Rasmussen, K.G. Larsen, and A. David. Guided controller synthesis for climate controller using uppaal-tiga. In *Proceedings of the 5th international conference on Formal modeling and analysis of timed systems*, pages 227–240, Berlin, Heidelberg, 2007. Springer-Verlag.

[KN93] W. Kohn and A. Nerode. Multiple-agent hybrid control architecture. In *Hybrid Systems*, volume 763 of *Lecture Notes in Computer Science*, pages 297–316. Springer, 1993.

[KO94] E. Knuth and M.J. Osborne. *A course in game Theory.* MIT Press, 1994.

[Kop07] E. Kopczynski. Omega-regular half-positional winning conditions. In *Proceeding of 16th Computer Science Logic (CSL)*, pages 41–53, 2007.

[LB93] M. Le Borgne. *Dynamical Systems over finite fields*. PhD thesis, Université de Rennes, 1993.

[Lew94] J. Lewin. *Differential Games*. Springer-Verlag, 1994.

[LLL09] J. Lunze and F. Lamnabhi-Lagarrigue. *Handbook of Hybrid Systems Control - Theory, Tools, Applications*. Cambridge University Press, Cambridge, United Kingdom, 2009.

[LY92] D. Lee and M. Yannakakis. Online minimization of transition systems. In *Proceedings of the 24th annual symposium on Theory of computing*, pages 264–274, New York, NY, USA, 1992. ACM.

[Mal02] O. Maler. Control from computer science. *Annual Reviews in Control*, 26(2):175–187, 2002.

[Maz02] R. Mazala. Infinite games. In *Automata, logics and infinite games*, pages 23–42. Springer-Verlag, London, UK, 2002.

[Mil51] J. Milnor. *Games against Nature*. John Wiley, 1951.

[MPS95] O. Maler, A. Pnueli, and J. Sifakis. On the synthesis of discrete controllers for timed systems. In *Proceeding of the 12th Annual Symposium on Theoretical Aspects of Computer Science*, volume 900 of *Lecture Notes on Computer Science*. Springer, 1995.

[MW99] C. Meder and W. Wonham. The TTCT tool. Personal communication., 1999.

[NYY92] A. Nerode, A. Yakhnis, and V. Yakhnis. Concurrent programs as strategies in games. *Logic from computer science*, pages 405–480, 1992.

[PLS00] G.J. Pappas, G. Lafferriere, and S. Sastry. Hierarchically consistent control systems. *IEEE Transactions on Automatic Control*, 45:1144–1160, 2000.

[PR89] A. Pnueli and R. Rosner. On the synthesis of a reactive module. In *Proceedings of the 16th ACM SIGPLAN-SIGACT symposium on Principles of programming languages*, POPL '89, pages 179–190, New York, NY, USA, 1989. ACM.

[Rab72] M.O. Rabin. *Automata on Infinite Objects and Church's Problem.* American Mathematical Society, Boston, MA, USA, 1972.

[RS07] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation-based abstraction refinement. *ACM Transactions in Embedded Computing Systems*, 6, February 2007.

[RW87] P.J. Ramadge and W.M. Wonham. Supervisory control of a class of discrete-event processes. *SIAM Journal of Control and Optimization*, 25:206–230, 1987.

[Sch86] A. Schrijver. *Theory of linear and integer programming.* John Wiley and Sons, 1986.

[SL98] R. Sengupta and S. Lafortune. An optimal control theory for discrete event systems. *SIAM Journal on Control and Optimization*, 36:488–541, 1998.

[TA99] S. Tripakis and K. Altisen. On-the-fly controller synthesis for discrete and dense-time systems. In *Proceeding of Formal Methods*, volume 1708 of *Lecture Notes in Computer Sciences*, pages 233–252. Springer Verlag, 1999.

[Tho95] Wolfgang Thomas. On the synthesis of strategies in infinite games. In *In Proceeding of the 12th Annual Symposium on Theoretical Aspects of Computer Science (STACS)*, pages 1–13, 1995.

[TLSS00] C.J. Tomlin, J. Lygeros, and S. Shankar Sastry. A game theoretic approach to controller design for hybrid systems. *Proceeding of the IEEE*, 88(7):949–970, 2000.

[Tri98] S. Tripakis. *The formal analysis of timed systems in practice.* PhD thesis, Université Joseph Fourier de Grenoble, 1998.

[TW94] J.G. Thistle and W.M. Wonham. Control of infinite behavior of finite automata. *SIAM Journal of Control and Optimization*, 32:1075–1097, 1994.

[TY96] S. Tripakis and S. Yovine. Analysis of timed systems based on time-abstracting bisimulations. In *Formal Methods in System Design*, pages 232–243. Springer-Verlag, 1996.

[Ver92] H. Le Verge. A note on Chernikova's algorithm. Technical Report 635, IRISA, Rennes, 1992.

[vNM47] J. von Neumann and O. Morgenstern. *Theory of Games and Econimic Behaviour*. Princeton University Press, Princeton, NJ, USA, 1947.

[WT97] H. Wong-Toi. The synthesis of controllers for linear hybrid automata. In *Proceeding of the 36th IEEE Conference on Decision and Control*, pages 4607 – 4612, San Diego, CA, 1997. IEEE Computer Society Press.