Accelerated Simulation of Hybrid Systems : Method combining static analysis and run-time execution analysis.

Ayman Aljarbouh

ANNÉE 2017

DOCTOR
EUROPEUS

THÈSE / UNIVERSITÉ DE RENNES 1
sous le sceau de l'Université Bretagne Loire

pour le grade de

DOCTEUR DE L'UNIVERSITÉ DE RENNES 1

Label : Doctorat Européen

Mention : Informatique

Ecole doctorale MathSTIC

présentée par

Ayman ALJARBOUH

préparée à l'unité de recherche HYCOMES
Centre INRIA Rennes - Bretagne Atlantique
Université de Rennes 1

Simulation accélérée des systèmes hybrides : méthode combinant analyse statique et analyse à l'exécution

Accelerated Simulation of Hybrid Systems : Method Combining Static Analysis and Run-time Execution Analysis

Thèse soutenue à Rennes le 13 Septembre 2017
devant le jury composé de :

Erika ÁBRAHÁM
Professeur, RWTH Aachen University / Rapporteur

Eugenio MOGGI
Professeur, Università di Genova / Rapporteur

Luc JAULIN
Professeur, ENSTA-Bretagne / Président du jury

Sophie PINCHINAT
Professeur, Université de Rennes 1 / Examinatrice

Nathalie BERTRAND
Chargée de recherche, INRIA / Examinatrice

Benoît CAILLAUD
Directeur de recherche, INRIA / Directeur de thèse

He who loves practice without theory is like the sailor who boards ship without a rudder and compass and never knows where he may cast.

(Leonardo da Vinci)

This thesis is dedicated to my parents for their love, endless support and encouragement.

Acknowledgments

First and foremost, I would like to express my deepest gratitude to my supervisor, Benoît Caillaud, for supervising this research work in a very professional manner, and for being patient, caring, and supportive. He continuously motivated me to aim for high-quality research, and was always available whenever I had a question about my research or writing. His insightful criticism and comments were an enormous help to me, and have led to highly improve the quality of my work.

I would like to thank all the members of my doctoral committee for their constructive evaluation of my thesis, and for their valuable comments which helped me to improve this document. In particular, I would like to thank Prof. Erika Ábrahám and Prof. Eugenio Moggi for accepting the role of reviewers. Also I would like to thank Prof. Luc Jaulin, Prof. Sophie Pinchinat, and Dr. Nathalie Bertrand who accepted to be part of the committee as examiners.

I greatly appreciate and wish to thank Prof. Albert Benveniste, Dr. Khalil Ghorbal, Aurélien Lamercerie, and Angélique Jarnoux for providing me a great and friendly work atmosphere duing my research work in the team Hycomes. I also would like to thank Prof. Marc Pouzet and Dr. Timothy Bourke for the constructive discussions and meetings during my visits with my supervisor to the team Parkas at École Normale Supérieure.

During my Ph.D. project I had the opportunity to spend three months as a visiting researcher at Halmstad University, where I worked in the team EMG (Effective Modeling Group) under the guidance of Prof. Walid Taha. For this amazing experience, I would like to thank Prof. Walid Taha for accepting me in his team and for providing me an amazing constructive working environment. I always enjoyed the discussions and attending the team meetings. I would also like to express my thanks to all the colleagues in EMG team, especially Adam Duracz, Yingfu Zeng, and Fei Xu who made my working days in Halmstad University instructive and at the same time really enjoyable.

This work is supported by the European project ITEA2-MODRIO (Contract N^{o} 6892), and the ARED grant of Brittany Regional Council. For part of this work, I was supported by the Collège Doctoral International (CDI) of the European University of Brittany. Their support is gratefully acknowledged.

Finally, I would like to express my gratitude to my family and friends for their constant spiritual and emotional support and encouragement throughout my Ph.D. project.

Résumé en Français

Cette thèse de doctorat porte sur la modélisation et la simulation de systèmes hybrides comportant des phénomènes Zénon.

Les systèmes hybrides peuvent être définis comme des systèmes dynamiques dans lesquels les dynamiques en temps continu et les dynamiques en temps discret interagissent les unes avec les autres. De tels systèmes existent dans un grand nombre d'applications technologiques où l'évolution de la partie physique du système, le plus souvent modélisée par un système dynamique en temps continu, est combinée avec des actions de contrôle intégrées, modélisée par un système dynamique en temps discret. Les modèles mathématiques des systèmes hybrides consistent généralement en dynamiques de temps continu habituellement décrites par des équations différentielles qui décrivent le comportement continu du système, et des dynamiques d'événements discrets telles que les machines à états finis, qui décrivent le comportement discret du système.

Cependant, en raison de l'interaction complexe entre les dynamiques continues et les dynamiques discrètes des systèmes hybrides, les concepteurs des systèmes complexes technologiques devraient accorder une attention particulière lors de la modélisation des systèmes hybrides. En fait, les modèles réalistes de systèmes hybrides nécessitent presque toujours l'abstraction d'une partie du comportement physique du système modélisé. Cette abstraction se fait au moyen d'équations idéales telles que la ré-initialisation et les contraintes conditionnelles qui conduisent généralement à des discontinuités dans les signaux physiques du modèle. Le terme *abstraction de modélisation* est donc désigné pour tout mécanisme qui permet de "cacher" un comportement physique concret en considérant des modèles idéalisés. Les modèles ainsi produits peuvent présenter des comportements Zénon, c'est à dire une séquence infinie d'événements discrets se produisant dans un intervalle de temps fini. Fondamentalement, la présence du comportement Zénon indique que le modèle décrive de manière incomplète le comportement physique réel du système hybride. Ce comportement peut être considéré donc comme un problème de modélisation.

Nous distinguons deux types de comportement Zénon dans les systèmes hybrides: 1) chattering-Zénon, et 2) genuinely-Zénon. Dans les modèles qui présentent de comportement chattering-Zénon, le système évolue de façon infinie entre ses états discrets, selon un alternance de transitions de modes et de dynamiques continues différentes à une fréquence infinie. Tout comportement Zénon qui n'est pas chattering-Zénon peut être classifié comme un comportement genuinely-Zénon. Dans cette thèse nous étudions,

d'une manière systématique et analytique, le comportement chattering-Zénon et aussi un type particulier de comportement genuinely-Zénon appelé genuinely-Zénon géométrique. Dans les modèles qui présentent de comportement genuinely-Zénon géométrique, une accumulation d'un nombre infini de commutations entre les états discrets — du système hybride — se produit en temps fini. Le comportement genuinely-Zénon géométrique amène la solution du système à converger vers un point limite Zénon selon une série géométrique, c'est à dire, dans les modèles qui présentent de comportement genuinely-Zénon géométrique, les événements discrets se produisent à une période de plus en plus faible, convergeant vers une date limite.

La simulation des comportements Zénon — des systèmes hybrides — pose une difficulté majeure: la simulation devient numériquement incorrect; le simulateur devient incapable d'avancer la simulation au delà du point limite Zénon, à cause de l'infinité des commutations discrètes en temps fini.

Dans cette thèse, nous considérons les modèles de systèmes hybrides comme des programmes exécutables écrits par des langages de programmation ayant des sémantiques hybrides. Fondamentalement, la définition d'un modèle sémantique hybride approprié est la première étape vers un développement d'un environnement propre de simulation pour les systèmes hybrides. Le développement d'un environnement de simulation hybride consiste généralement à suivre les étapes suivantes:

1. Définir correctement un modèle sémantique hybride, qui peut considérer les propriétés de comportement continue/discret des systèmes hybrides.

2. Concevoir et développer un simulateur capable à calculer les dynamiques et les solutions des modèles des systèmes hybrides conformément au modèle sémantique hybride défini.

3. Concevoir et développer un langage de programmation capable d'exprimer tous les éléments et composants des modèles hybrides conformément au modèle sémantique hybride défini. La vérification de code doit être incluse dans cette étape pour prouver de manière statique la validité sémantique des modèles simulés.

4. Concevoir et développer un compilateur pour le langage de programmation développé. Le compilateur devrait être capable d'effectuer une vérification statique efficace des modèles, et rejeter les modèles invalides.

De nombreux outils de modélisation et de simulation pour les systèmes hybrides ont été développés ces dernières années. Ils peuvent être classés en deux catégories: ceux qui accordent une attention particulière à une définition rigoureuse des modèles, comme par exemple SpaceEx [44], Ptolemy [27], et Zélus [57], et ceux qui utilisent une approche informelle pour la définition des modèles, comme Simulink[1], le langage Modelica [55] et ses outils associés. Tous ces outils partagent la même approche de l'exécution du modèle hybride alternant entre l'évolution continue et les séquences de commutations discrètes similaire à l'approche définie par la notion d'automate hybride [30]. Pour tous

[1]https://fr.mathworks.com/products/simulink.html

ces outils, la sémantique opérationnelle de la dynamique continue (équations différentielles) n'est pas incluse dans le modèle sémantique: les solveurs numériques exécutent le comportement continu en faisant progresser le temps et en calculant les valeurs des variables continues physiques. Aucun de ces outils n'a un modèle sémantique qui permet de détecter le comportement Zénon et de l'éliminer. Ils comptent tous sur l'analyse de la sortie du solveur à chaque pas d'intégration numérique, sans aucun moyen de spécifier le comportement du solveur au point limite Zénon.

Dans cette thèse, nous proposons une solution à ce problème. En particulier, nous proposons une méthode combinant un analyse statique et un analyse à l'execution pour détecter et éliminer le comportement Zénon lors de la simulation des modèles des systèmes hybrides. Nous montrons aussi comment notre méthode peut être utilisée dans le développement d'outils de modélisation et de simulation qui produisent une simulation correcte éliminatoire de comportement Zénon.

La première partie de notre contribution est destinée à proposer de sémantique non-standard pour les exécutions Zénon des automates hybrides. Ceci est basé sur l'interprétation des exécutions Zénon dans un domaine de temps hybride non-standard. L' avantage de l'utilisation de la sémantique non-standard dans l'analyse du comportement Zénon c'est que l'analyse non-standard permet aux solutions des modèles ayant un comportement Zénon d'être bien définies au-delà des points limites Zénon, ainsi que de représenter l'interaction complexe entre les dynamiques continues et les dynamiques discrètes de manière concrète:

1. Les dynamiques continues du système hybride sont réduites à des équations récurrentes qui représentent rigoureusement l'itération infinie des commutations discrètes en durée infinitésimale. Par conséquent, nous pouvons gérer les dynamiques hybrides en appuyant seulement sur un paradigme complètement discret.

2. La représentation non-standard des dynamiques hybrides est complète, qui permet de traiter les points limites Zénon.

La deuxième partie de notre contribution est attribuée à proposer une technique de calcul éliminatoire de comportement Zénon. L'idée clé dans notre technique est d'effectuer la détection et l'élimination de comportement Zénon en utilisant l'analyse comportementale du système, au lieu de seulement considérer le nombre des passages à zéro, comme ce qui est le cas désormais dans tous les outils de modélisation et de simulation développés pour les systèmes hybrides. La technique d'analyse comportementale que nous proposons pour le traitement de comportement Zénon est basée sur un analyse systématique des deux types de Zénon. Nous proposons des conditions formelles pour bien distinguer si les modèles hybrides simulés présentent ou non de comportement Zénon. Nous proposons également des méthodes pour une définition correcte des solutions au delà des points limites Zénon. La question de l'existence et l'unicité de la solution au delà du point limite Zénon est également bien étudiée dans cette thèse. Nos méthodes proposées dans cette thèse permettent de sacrifier la notion de Zénon-free: 1) la décision algorithmique est basé sur des conditions formelles explicitement définies et

fournies au simulateur hybride, et 2) la notion correcte de solution au delà du point limite Zénon est bien définie et établie dans notre technique proposée.

Des exemples de systèmes hybrides, illustrant l'utilisation des méthodes proposées dans cette thèse, sont également présentés.

Abstract

The theme of this dissertation is to deal with Zeno behavior of hybrid systems from a simulation perspective.

Hybrid systems can be defined as dynamical systems in which continuous and discrete dynamics interact with each other. Such systems exist in a large number of technological systems where the physical continuous evolution of the system is combined with embedded control actions. The mathematical models of hybrid systems consist typically of continuous time dynamics usually described by differential equations describing the continuous behavior of the system, and discrete event dynamics such as finite state machines (synchronous or data-flow programs) that describe the discrete behavior of the system.

However, due to the complex interaction between the continuous and discrete dynamics, designers should pay special attention when modeling hybrid systems. In fact, realistic models of hybrid systems almost always necessitate part of the hybrid system's physical behavior to be abstracted by means of "ideal equations" such as reset and conditional constraints that typically lead to discontinuities in physical signals. The term *modeling abstraction* is thus designated to any mechanism that enables concrete physical behavior to be "hidden" by considering idealized models. Intuitively, because of such abstraction, the model jumps over instants corresponding to the violation of abstraction mechanisms. Such modeling abstraction mechanism may result in hybrid models that exhibit Zeno behavior. Formally, we define Zeno behavior as an infinite sequence of discrete events occurring in a finite amount of time. Basically, the presence of Zeno behavior indicates that the hybrid system's model incompletely describes the real physical behavior of the system being modeled. If we consider the standard semantics[2] of executions of hybrid systems models, the problem can best be described as follows: the physical system keeps evolving continuously beyond a certain point, but the model's continuous evolution is undefined beyond that point, because of the infinity of the discrete transitions or mode switchings. Such inherent limitation of the hybrid system model makes the solution of the system reaches a (Zeno limit) point in time at

[2]Apart from Zélus [57] which uses non-standard semantics, all the other modeling and simulation tools of hybrid systems use standard semantics of executions of hybrid models.

which the model is no longer valid. This is due the fact that the modeling abstraction mechanism incompletely describes the complex interaction between the continuous and discrete dynamics of the hybrid system being modeled. That is, Zeno behavior can be seen as a modeling artifact that can never occur in reality.

Analytically, we distinguish between two different types of Zeno behavior in hybrid systems: i) chattering-Zeno, and ii) genuinely-Zeno. In models that exhibit chattering-Zeno, the system infinitely moves back and forth between modes in a discrete fashion with infinitesimal time spent between the successive mode switchings. Any Zeno behavior that is not chattering-Zeno can be classified as genuinely-Zeno. In this dissertation we focus on both chattering-Zeno and a particular type of genuinely-Zeno which we call geometric-Zeno. In models that exhibit geometric-Zeno, an accumulation of an infinite number of mode switchings occurs in finite time. Geometric-Zeno behavior leads the solution of the system to converge to a Zeno limit point according to a geometric series. Roughly speaking, in geometric-Zeno models discrete events occur at an increasingly smaller distance in time, converging against a limit point.

Zeno behavior is highly challenging from a simulation perspective. In fact, although chattering-Zeno and geometric-Zeno are analytically different, the effect of these two types of Zeno during the numerical simulation is similar: the simulation process effectively stalls, halts and terminates with an error message, or becomes numerically incorrect and produces faulty results, as infinitely many discrete transitions would need to be simulated.

This dissertation takes the perspective that models of hybrid systems are executable programs written in programming languages having a hybrid system semantics. Basically, defining a proper hybrid semantic model is the first step of developing a simulation framework for hybrid systems. This step is mandatory even before designing the language or the simulator. The development of a hybrid simulation framework typically include the following steps:

1. Properly define a hybrid semantic model that can account for the expected properties of the hybrid systems under simulation.

2. Design and develop a simulator that could approximate the model dynamics conforming to the defined hybrid semantic model.

3. Design a language capable of expressing all models elements and components conforming to the hybrid semantic model. Type-checking must be included in this step to prove statically the semantic validity of the simulated models.

4. Design a compiler for the language. The compiler should be capable of performing static checks of models and also rejecting models that are invalid.

Many modeling and simulation tools for hybrid systems have been developed in the past years. They can be classified into two categories: those who put special attention on defining models rigorously, such as for instance SpaceEx [44], Ptolemy [27] (based on the super-dense time semantics in [47]), and Zélus [57] (based on the non-standard semantics in [54]); and those who use informal approach for model definition such as Simulink[3], Modelica language [55] and its associated tools. All these modeling and simulation tools share the same approach of hybrid model execution alternating between continuous evolution and sequences of discrete switchings [57] as defined by the notion of hybrid automata [30]. For all of these tools, the operational semantics of continuous dynamics (differential equations) is not included in the core semantic model: numerical solvers execute the continuous behavior by advancing time and computing the values of physical continuous variables. None of the above tools have a Zeno-free semantic model. They all rely on analyzing the solver output at each integration time step, with the solver behavior at the Zeno-limit point being usually unspecified.

In this dissertation we focus on the first two steps above. In particular, we focus on proposing a rigorous Zeno-free computational framework for hybrid semantic model design, and how this Zeno-free computational framework can be implemented in the design of hybrid systems simulators.

The first part of our contribution is to propose non-standard semantics for Zeno executions of hybrid systems models. This is based on interpreting Zeno executions in a non-standard *densely ordered* hybrid time domain. The advantages of using non-standard semantics in the analysis of Zeno behavior is that it allows for solutions of Zeno hybrid models to be well-defined beyond the Zeno limit points, as well as representing the complex interaction between continuous and discrete dynamics in a concrete way:

1. The continuous dynamics of the hybrid system is reduced to the recurrence equation that represents the infinite iteration of infinitesimal discrete changes with infinitesimal duration. Therefore, we can handle the hybrid dynamics based only on fully discrete paradigm.
2. The representation of dynamics based on non-standard analysis is complete and the exact limit point of discrete change, like chattering-Zeno and geometric-Zeno limit points, can be handled.

The second part of our contribution is to propose a rigorous Zeno-free computational framework for hybrid semantic model design and implementation. The key idea in our proposed computational framework is to perform Zeno detection and avoidance by using behavioral analysis of the system, instead of only considering zero-crossings in a hybrid time domain. The behavioral analysis technique we propose for treating Zeno is based on analyzing both types of Zeno systematically. We provide formal conditions on when the simulated models of hybrid systems display chattering-Zeno and geometric-Zeno executions. We also provide methods for carrying solutions beyond Zeno. The issue of existence and uniqueness of solution beyond Zeno is also studied in this dissertation. Our Zeno-free computational framework allows sacrificing the notion of *Zeno-freeness* as: i) the decision on whether or not the Zeno limiting state is chattering-Zeno or geometric-Zeno is based on formal conditions explicitly defined and provided to the hybrid simulator's solver, and ii) the correct notion of solution beyond Zeno is well defined and established in our framework.

[3]https://fr.mathworks.com/products/simulink.html

Our approach also supports mixing compile-time transformations of hybrid programs (i.e. generating what is necessary for Zeno detection and avoidance), the decision, in run-time, for the urgent transition from pre-Zeno to post-Zeno state (based on formal conditions for the existence of Zeno), and the computation, in run-time, of the system dynamics beyond Zeno.

Examples of hybrid systems with domains, guard sets, vector fields, and reset maps, illustrating the use of the methods proposed in this dissertation, are also provided.

Contents

List of Figures

List of Tables

Publications

1. Ayman Aljarbouh, Yingfu Zeng, Adam Duracz, Benoît Caillaud, Walid Taha. Chattering-Free Simulation for Hybrid Dynamical Systems [Semantics and Prototype Implementation]. In *Proceedings of the 19th IEEE International Conference on Computational Science and Engineering (CSE 2016)*, August 24-26, 2016 - Paris, France.

2. Ayman Aljarbouh, Benoît Caillaud. Chattering-Free Simulation of Hybrid Dynamical Systems with the Functional Mock-Up Interface 2.0. In *Proceedings of the First Japanese Modelica Conference*, May 23-24, 2016, Tokyo, Japan. Linköping Electronic Conference Proceedings, 124(013), pp. 95-105, 2016.

3. Ayman Aljarbouh, Benoît Caillaud. Robust Simulation for Hybrid Systems: Chattering Path Avoidance. In *Proceedings of the 56th Conference on Simulation and Modelling (SIMS 56)*, October 7-9, 2015, Linköping, Sweden. Linköping Electronic Conference Proceedings, 119(018), pp. 175-185, 2015.

4. Ayman Aljarbouh, Benoît Caillaud. On the Regularization of Chattering Executions in Run-Time Simulation of Hybrid Systems. In *Proceedings of the 11th Baltic Young Scientists Conference*, July 2015, Tallinn, Estonia.

5. Adam Duracz, Ferenc A. Bartha, Ayman Aljarbouh, Jawad Masood, Roland Philippsen, Henrik Eriksson, Jan Duracz, Fei Xu, Yingfu Zeng, Walid Taha, Christian Grante. Using Rigorous Simulation to Support Hazard Analysis and Risk Assessment (HARA) in the ISO 26262 Functional Safety Standard. *Paper submitted to ACM Transactions on Embedded Computing Systems (TECS), for a special Issue paper as an extension to the paper of A. Duracz et al* [4] *presented at ICESS'15, New York, USA.*

[4] http://www.duracz.net/adam/publications/icess15.pdf

Chapter 1

Introduction

This dissertation explores a new Zeno-free simulation technique for hybrid systems models exhibiting Zeno behavior. The motivation is to provide a rigorous general Zeno-free computational framework so that hybrid systems modeling and simulation tools can provide an efficient Zeno detection and avoidance support. This introductory chapter motivates the work, and describes its contributions and origins.

1.1 Scientific Context

1.1.1 Hybrid Systems

Because of their heterogeneous structure, the term *hybrid* is attributed to dynamical systems having state variables that can evolve continuously and/or discontinuously. That is, the presence of two different behaviors, continuous and discrete, is the cause of heterogeneity [2, 4]. In such systems, continuous and discrete dynamics interact with each other. Systems of this type exist in a large number of technological systems where discrete control switches combine the continuous evolution of the physical processes, such as process control systems, embedded computation, avionic systems, mechatronic systems, robotic systems, and many other technological systems [8].

Hybrid systems theory is multidisciplinary and can be considered as a young research area in contrast to the mono-disciplinary research fields such as electrical, mechanical, or software engineering. The urgent demand for interdisciplinary design and incorporative development methods for complex technological systems has accelerated the growth of hybrid systems theory in recent years. Combining time-driven and event-driven dynamics, hybrid systems framework has proved to be an important and effective tool for a large class of engineering systems that combine analog and digital devices, interact through networks, conduct tasks collaboratively, and operate in environments filled with uncertainties [6]. The mathematical models of hybrid systems consist typically of continuous time-driven dynamics, usually represented by differential equations, to describe the continuous behavior of the system, and discrete event-driven dynamics such as finite state machines to describe the discrete behavior of the system.

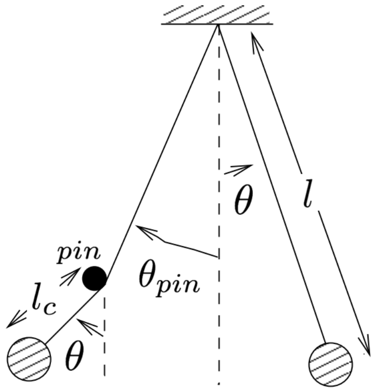


Figure 1.1: Constrained pendulum.

The execution of a hybrid system is typically characterized by smooth continuous evolutions during which the discrete mode remains constant, separated by discrete events (i.e. control actions). The discrete events are either time events or state events often given by zero-crossing functions, guards being enabled, or invariants about to be violated. The discrete actions could include instantaneous jumps in a continuous state variable and/or switches to completely different dynamical modes [1,3,6,9].

As a simple example of a two-modes hybrid system, consider a constrained pendulum of length l as sketched in Figure 1.1, where a pin is placed to constrain the string's movement such that the length of the pendulum becomes l_c after hitting the pin. The string's trajectory changes at $\theta = \theta_{pin}$, and another equation set is used for the trajectory of the string. The continuous state $x = (\theta, v)$ is given by the angle θ and the angular velocity v. This example is a hybrid system having two modes: i) unconstrained pendulum when $\theta > \theta_{pin}$, and ii) constrained pendulum when $\theta \leq \theta_{pin}$. Thus, it is no longer possible to define the whole trajectory of the system with a single set of differential equations or dynamics $\dot{x} = f(x)$. In this case, it should be defined by two different dynamics $f_1(x)$ (the dynamics of the unconstrained pendulum) and $f_2(x)$ (the dynamics of the constrained pendulum):

$$f(x) = \begin{cases} f_1(x) & \text{if } \theta > \theta_{pin}, \\ f_2(x) & \text{if } \theta \leq \theta_{pin}, \end{cases} \tag{1.1}$$

where

$$f_1(x) = \begin{cases} \dot{\theta} = & \frac{1}{l} \cdot v, \\ \dot{v} = & -(g \cdot sin(\theta)) - (\alpha \cdot v), \end{cases} \tag{1.2}$$

$$f_2(x) = \begin{cases} \dot{\theta} = & \frac{1}{l_c} \cdot v, \\ \dot{v} = & -(g \cdot sin(\theta)) - (\alpha \cdot v), \end{cases} \tag{1.3}$$

with g being the gravity constant, and α a friction coefficient. Obviously, the discrete change between the modes is described in terms of the changing the pendulum length from l_c to l and vice versa. Indeed, in this hybrid system no state jumps occur during the discrete switching between the two modes.

1.1.2 Modeling Abstraction

With powerful modeling tools and languages it is possible to model a large variety of physical hybrid phenomena, but even so it is possible to generate models that are unreasonable, mathematically or physically because of a bad use of the modeling tool or language during modeling abstraction.

In fact, realistic models of hybrid systems almost always necessitate part of the hybrid system's physical behavior to be abstracted by means of "ideal equations" such as reset and conditional constraints that typically lead to discontinuities in physical signals.

The term *modeling abstraction* is designated to any mechanism that enables concrete physical behavior to be "hidden" by considering idealized models. Intuitively, because of such abstraction, the model jumps over instants corresponding to the violation of abstraction mechanisms. Thus, idealization yields explicitly in providing constraints to be satisfied at these instants.

Practically, the price to pay to profit from high-fidelity modeling is most often quite expensive in terms of simulation performance and model complexity.

Simulation performance may dramatically run into trouble if the simulated model has excess modeling details. A solver that performs very well when simulating an idealized (equation-based) model would dramatically struggles (and often breaks down) when it is simulates a highly detailed model of the same system. Intuitively, the solver struggles because most of the CPU time is attributed in solving the additional details of the high-fidelity model, corresponding exactly to the physical details which we want to abstract away.

This brings us to a similar conclusion: modeling abstraction helps concentrating on the important parts of physical phenomena, making the model often simpler and easy to control. However, even though modeling abstraction is very useful in practice, modelers should pay a special attention when abstracting or over-approximating a system, because a bad use of a modeling formalism during abstraction can result in models that are invalid at certain points, such as Zeno models, and also can lead to singularities in the simulation code. A simple typical example is the presence of ideal (or idealized) diodes in electrical circuits, where unilateral constraints on currents and voltages (as ideal equations from abstraction) require a special attention in the simulation code to avoid singularity.

1.2 Scientific Problem and Challenges

The interaction between the continuous and discrete dynamics of hybrid systems is complex in its nature, which necessitates a special attention from designers when using modeling abstraction in building idealized models of hybrid systems. In fact, modeling abstraction or over-approximation mechanisms can lead to deviate models whose evolution is undefined at a certain point. In the following, we discuss this issue from both modeling and simulation perspectives.

1.2.1 Modeling Issue

Even with powerful modeling languages and tools, once can easily generate models that admit no solutions at a certain point, for some given initial states. This property is undesirable in physical systems modeling because the mathematical model, in this case, incompletely describes the real physical behavior of the system being modeled: the physical system keeps evolving continuously beyond a certain point, but the model's continuous evolution is undefined beyond that point.

For instance, the execution of a hybrid system's model may take an infinite sequence of discrete mode switchings occurring in finite execution time. Such property is referred to Zeno executions of idealized models of hybrid systems. Zeno behavior can be seen as a modeling artifact that can never occur in reality. Basically, the presence of Zeno behavior is an indication that the abstraction mechanism used in modeling imperfectly describes the complex interaction between the discrete and continuous dynamics of the modeled hybrid system. In the standard semantics of executions of hybrid systems, Zeno behavior makes the evolution of the Zeno model reach a (Zeno limit) point in time beyond which the model is no more valid. Roughly speaking, the model has no solutions defined past the Zeno limit point. Since it is crucial to use abstraction in complex systems modeling, it is then quite important to understand when it leads to Zeno.

Formally, we define Zeno behavior as an infinite sequence of discrete mode switchings (i.e. transitions) that occur in a finite period of time. Analytically, we can distinguish between two types of Zeno behavior: i) chattering-Zeno, and ii) genuinely-Zeno. In models that exhibit chattering-Zeno, the system infinitely moves back and forth between modes in a discrete fashion with infinitesimal time spent between the successive mode switchings. Any Zeno behavior that is not chattering-Zeno can be classified as genuinely-Zeno. In this thesis we focus on both chattering-Zeno and a particular type of genuinely-Zeno behavior which we call geometric-Zeno. In models that exhibit geometric-Zeno, an accumulation of an infinite number of mode switchings occurs in finite time. The convergence of the solution to a geometric-Zeno limit point occurs according to a geometric series.

- **Chattering-Zeno:**
  Chattering-Zeno can be defined as a fast infinite repeated switching between different control actions or modes of operation. Physically, chattering-Zeno occurs when nearly infinitesimally equal thresholds for transition conditions of different modes are given and the system starts to oscillate around them. Numerical errors may also lead to chattering-Zeno because infinitesimally equal thresholds for transition conditions may be satisfied due to local errors.

  As a simple example of a hybrid system model exhibiting chattering-Zeno, consider for instance the following discontinuous differential equation

$$\dot{x}(t) = \begin{cases} 1 & \text{for } x(t) < 0, \\ -1 & \text{for } x(t) \geq 0. \end{cases} \tag{1.4}$$

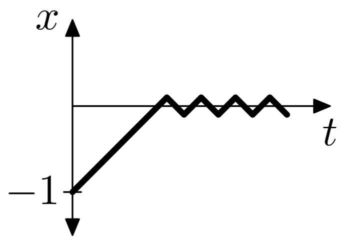


Figure 1.2: Illustration of a chattering-Zeno trajectory.

For an initial condition $x(0)$, we obtain a solution of the initial value problem

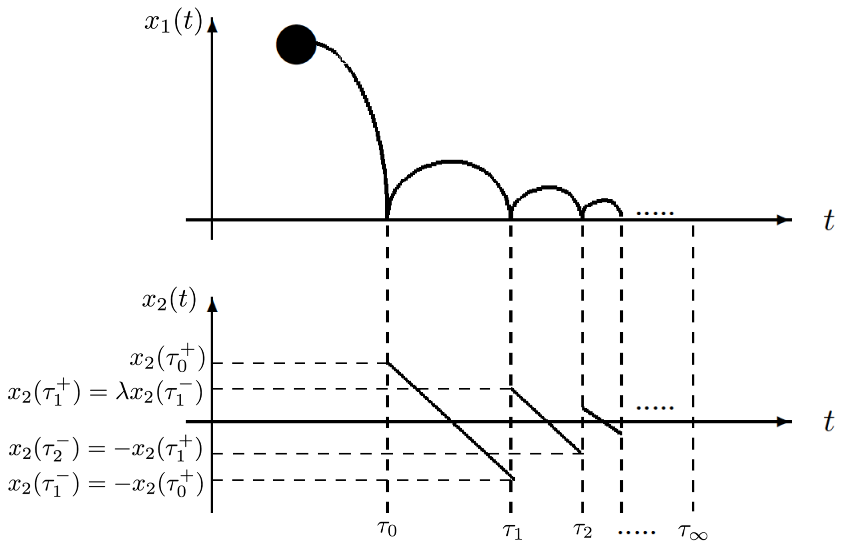
$$x(t) = \begin{cases} t + a & \text{for } x(t) < 0, \\ -t + b & \text{for } x(t) \geq 0, \end{cases} \tag{1.5}$$

with constants a and b being determined by the initial condition. Clearly, with any time discretization scheme, a solution initialized outside $x(t) = 0$ can reach the hyper discontinuous surface $x(t) = 0$ in finite time.

Consider for instance forward Euler discretization method $x(t_{i+1}) - x(t_i) = h \cdot f(x(t_i))$, where $h > 0$ is the integration step size. At a given time instant $t \in (t_i, t_{i+1}]$, if the solution arrives at $x(t) = 0$ it can not leave it because $x(t_{i+1}) - x(t_i) = h > 0$ for $x(t_i) < 0$ and $x(t_{i+1}) - x(t_i) = -h < 0$ for $x(t_i) \geq 0$. The solution starts then to exhibit on the hyper surface $x(t) = 0$ a chattering-Zeno back and forth between the two disjoint continuous domains $x(t) < 0$ and $x(t) > 0$ (see Figure 1.2).

- **Geometric-Zeno:**
  Geometric-Zeno behavior is characterized by an accumulation of an infinity of discrete mode switchings (i.e. events) that occur in a finite execution time. Geometric-Zeno behavior leads the solution of the system to converge to a Zeno limit point according to a geometric series. Roughly speaking, in models that exhibit geometric-Zeno behavior, discrete events occur at an increasingly smaller distance in time, converging against a limit point. For example, if a new discrete event occurs after a time duration equal to the half of the time between the two previous discrete events, a series of events features that, after n events, has proceeded in time according to $\sum_{k=1}^{n} \frac{1}{2^k}$. This series asymptotically converges to 1 in the limit of $n \to \infty$.

  Consider for example the model of a bouncing ball whose collisions are inelastic; see Figure 1.3. During each bouncing collision, the loss of ball's kinetic energy occurs with a restitution coefficient $\lambda \in (0, 1)$. We denote the height of the ball by x_1, with $x_1 \geq 0$ being the invariant constraint, and $\ddot{x}_1 = -g$ being the dynamics of the ball during the falling phase, where g is the gravitational constant. We denote the velocity $\dot{x}_1$ of the ball by x_2. We include Newton's restitution rule $x_2(t) := -\lambda x_2(t)$ when $x_1(t) \leq 0$ and $x_2(t) < 0$. We can easily compute the time



$x_1(t)$
t
$x_2(t)$
$x_2(\tau_0^+)$
$x_2(\tau_1^+) = \lambda x_2(\tau_1^-)$
t
$x_2(\tau_2^-) = -x_2(\tau_1^+)$
$x_2(\tau_1^-) = -x_2(\tau_0^+)$
τ_0
τ_1
τ_2
τ_∞

Figure 1.3: Illustration of the geometric-Zeno behavior.

instants of discrete events (or resets) $\{\tau_i\}_{i \in \mathbb{N}}$ by

$$\tau_{i+1} = \tau_i + \frac{2 \cdot \lambda^i \cdot x_2(\tau_0)}{g}; \quad i \in \mathbb{N}, \tag{1.6}$$

assuming that $x_1(0) = 0$ and $x_2(0) > 0$. Hence, the series $\{\tau_i\}$ has a finite limit $\tau_\infty = \frac{2x_2(\tau_0)}{g \cdot (1-\lambda)} < \infty$, so the continuous state $(x_1,\ x_2)$ converges to $(0,\ 0)$ when $t \to \tau_\infty$.

The physical interpretation is that the ball is at rest within a finite time span, but after infinitely many bounces. This example is a typical example of a hybrid system model having geometric-Zeno behavior, where in this example we have an infinite number of state re-initializations and the set of event times for the bouncing ball contains a geometric-Zeno limit point.

1.2.2 Simulation Issues

Efficient simulation has to be explicitly concerned with the nature of the separate parts that comprise hybrid dynamic system behavior.

Typically, the simulation of hybrid models uses numerical solvers for handling the continuous evolution of the hybrid model. Whenever discrete events occur, the continuous-time state vector of the model may change (in value and in dimension), and new initial values may have to be computed. In the event-driven simulation technique (see Section 2.3), the continuous-time evolution of the system is interrupted by discrete steps that are triggered by the activation of discrete events.

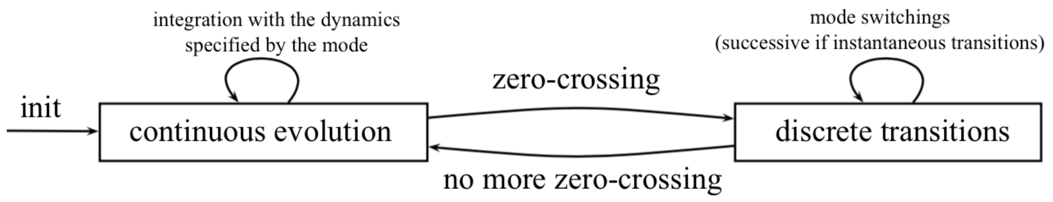

integration with the dynamics specified by the mode
mode switchings (successive if instantaneous transitions)
init
continuous evolution
zero-crossing
discrete transitions
no more zero-crossing

Figure 1.4: The simulation loop of a typical event-driven simulator.

Discrete events are events that occur during the numerical integration of the differential equations that describe the continuous dynamics of the system.

A discrete event may be activated by a discrete time step. In this case, events are called *time events*. Such time events are easier to handle as they may be specified directly before the simulation begins.

A discrete event may also be triggered when some *zero-crossing* occurs. We call this type of events *state events*. A zero-crossing is an arithmetic expression of the form `up(`z`)` that uses a real-valued function z to identify the boundary at which the discrete change -according to state events- occurs. A zero-crossing event function z may change its sign or its domain, and is usually made of constants, computed variables, state variables, as well as arithmetic operations of constants and/or variables. Numerical solvers have to be very cautious not to miss zero-crossings because the latter are regions of maximal stiffness. It is highly recommended then to use variable step size integration techniques when simulating a model with event driven mode.

A typical event-driven simulator works as follows (Figure 1.4): The main simulation loop starts by initializing the solver with the system's state $x(0)$ at initial time $t = 0$, a system of differential equations, and a set of zero-crossing functions. Then the solver integrates using the specified integration method until a time event occurs or one of the zero-crossings is detected and located. When this occurs, a discrete control action is sent back to the simulation loop, which triggers one or many discrete execution steps before reinitializing the solver (reset) and continuing the integration.

The nature of zero-crossings detection is to ask the numerical solver to watch the sign or domain of the values of the event functions at the beginning and at the end of each time integration step $[t_{i-1}, t_i]$, and if it changes, declare a state event. For a given event function z, a zero-crossing could be triggered either when $z(t_{i-1}) \leq 0$ and then $z(t_i) \geq 0$ (the so-called at-zero semantics), or when $z(t_{i-1}) < 0$ and then $z(t_i) \geq 0$ (the so-called contact semantics), or when $z(t_{i-1}) \leq 0$ and then $z(t_i) > 0$ (the so-called crossing semantics) [18]. Simulation tools for hybrid systems use only the last two semantics. In Simulink[1], the semantics of zero-crossings detection is a disjunction of "contact" and "crossing" semantics. In Modelica[2], the programmer has the choice to use either "contact" or "crossing" semantics.

For localizing the detected zero-crossing state events, solvers usually have to back-

[1]https://fr.mathworks.com/products/simulink.html
[2]https://www.modelica.org/

track and decrease the integration step in order to approximate the point of zero-crossing up to a certain precision (usually initialized in advance). This requires, however, a robust approach to detect when exactly a zero-crossing occurs.

Zeno hybrid models are particularly highly challenging from a simulation perspective. In particular, simulation algorithms may collapse if the solution of the simulated model is unspecified beyong a given point. The most challenging case is the simulation of hybrid models exhibiting Zeno behavior.

Traditionally, the time-line of simulation relies either on a fixed time integration step, where the size of the step is usually provided as a simulation parameter, or a variable time integration step whose size is adjusted automatically during the simulation based on the detected discrete events.

The advantage of the former is that it is a Zeno-free simulation technique by construction, as at each integration step the time proceeds by a constant value. However, fixed time stepping generates faulty simulation results whenever the solution activity is higher than the frequency of the fixed time stepping.

On the other hand, variable step size simulation improves the accuracy of simulation. It is based on bracketing the zero-crossing events (bi-sectional search or Secant method) and using increasingly smaller and smaller step sizes to identify (up to a certain precision) when the zero crossing event has occurred. Basically, the dynamic adjustment of the variable time step size is done automatically by the solver, where the solver increases the integration step size when a variable is changing slowly, and decreases the integration step size when the variable is changing rapidly. Such mechanism makes the solver to take very small integration steps in the neighborhood of a discontinuity region, because usually the system variables are changing rapidly in these regions. Clearly, this improves the accuracy and numerical precision of the simulation.

However, because of the nature of finite precision arithmetic on digital computers, the time that the event occurred can only be located within an interval $[t_{Left}, t_{Right}]$ that corresponds to the machine precision.

During each iteration of the zero-crossing localization, the zero-crossing function z is evaluated twice: at the left and the right side of the reducing interval. Once the discrete event is bracketed by t_{Left} and t_{Right}, the solver firstly proceeds integration time from the previous time instant t_{i-1} to t_{Left}. The solver is then reset before advancing to t_{Right} followed by switching the mode. In doing so, the assumption of continuity holds throughout the numerical integration.

Although such mechanism is very efficient for many types of simulations, it can result however in a simulation halt whenever an infinity of zero-crossing events is present. In particular, Zenoness causes the previous t_{Right} becomes t_{Left} of the next reducing interval leading the continuous integration to be broken down.

We conclude that, although analytically distinctly different, the effect of chattering-Zeno and geometric-Zeno behaviors during the numerical simulation is similar: both chattering-Zeno and geometric-Zeno behaviors are highly problematic and challenging in simulation (and thus in analysis) of hybrid systems, as the solver is asked to take infinitely many discrete execution steps corresponding to the triggered discrete transitions.

1.3 Literature Survey

Early works have considered only the executions of hybrid systems under the assumption that they are not Zeno. Attention was later attributed to Zeno phenomena when the research area of hybrid systems was extended to hybrid systems having rich continuous physical dynamics. Both chattering-Zeno and geometric-Zeno problems have been investigated by means of different methods and from many different perspectives.

A method that was proposed to avoid both types of Zeno is to add a small hysteresis of size ϵ to the zero-crossing event functions z. This approach was adopted in the specification of the standard FMI (Functional Mock-up Interface)[3] [10]. The disadvantages of this approach are the followings:

1. The size of the small value ϵ of the hysteresis should be directly related to the value of the event function z. That is, in order to determine the correct size of ϵ in the simulation environment which simulates the model, the nominal value of the event function z has to be reported by the modeler. This requires a manual manipulation by the modeler to provide an appropriate set of ϵ for all the event function z used in the simulation program. In the context of the FMI standard, the situation becomes more complicated, because the nominal value of the event function z has to be reported by the FMU (Functional Mock-up Unit) representing the model, which requires more information from the simulation tool that has generated and exported the FMU, but cannot be handled efficiently in the simulation environment that imports and simulates this FMU. If this would be handled in the simulation environment which simulates the FMU, there is always the danger that the simulation environment does not handle it properly, but the FMU would be blamed for a failure.
2. Adding hysteresis to the event functions does not guarantee an efficient treatment of Zeno, because the notion of Zeno-freeness here is done by construction, i.e. equivalent to the above mentioned notion of Zeno-freeness using a fixed time step. For example, in case of chattering-Zeno, the physics in the model cause the solution $x_\epsilon(\cdot)$ be a saw-toothed, or zigzag function, i.e. a function that oscillates around the switching surface with peaks at $-\epsilon < 0$ and $+\epsilon > 0$, with $t_{i+1} - t_i = 2\epsilon$. Clearly, for both types of Zeno this results in incorrect simulation results as it disregard and ignores the zero crossings `up(`z`)` when their magnitude remains below the level ϵ. The incorrectness here means that there may be zero crossing events `up(`z`)` with magnitude below ϵ and these events are associated with jumps to completely different dynamics. In this case, ignoring such events produces a behavior deviant from the expected behavior.
3. Many simulation tools add a hysteresis when handling zero crossings detection, to ensure that the zero crossings `up(`z`)` happen with non-zero values of the input arguments of z at the integration restart. Therefore, adding a hysteresis to avoid Zeno will introduce the hysteresis twice to the event functions z, and as a result, the resulting triggered state events are inaccurate.

[3]https://www.fmi-standard.org/start.

This method was proposed as a unified approach to deal with both chattering-Zeno and geometric-Zeno from a practical simulation perspective. Below we list the literature survey on both types of Zeno as they were investigated from different point of views.

1.3.1 Other Literature Survey on Chattering-Zeno

Chattering-Zeno behavior has also been investigated using sliding mode control approach. The sliding mode approach is based on detecting regions on the switching manifold on which chattering-Zeno occurs, and then forcing the solution trajectory of the system to slide on the manifold in these regions [11, 13, 14, 20]. While the infinitely fast switching between modes occurs, a smooth sliding motion takes place on the switching surface to eliminate the fast chattering between modes. An additional mode, the so-called sliding mode, can be inserted into the hybrid model to represent the dynamics during sliding, and thus, replaces chattering-Zeno. Filippov's differential inclusions [17, 20] (the so-called equivalent dynamics) is a method that was developed by Filippov to define the system dynamics on the switching surface in such a way that the state trajectory slides along the switching surface on which chattering-Zeno occurs. In this method, projections of the solution trajectories on both sides in a small neighborhood around the surface are used to determine the average velocity on the surface. However, the computation of the equivalent dynamics turns out to be difficult and highly challenging whenever the system chatters between more than two dynamics. One of the properties of chattering-Zeno behavior is that the solution trajectory may chatter between modes on a large number of discontinuous switching surfaces having different dimensions. This special case of chattering-Zeno on discontinuous surfaces intersections arises naturally when chattering-Zeno occurs in hybrid models having multiple discontinuous control variables. Such dimensionally different chattering-Zeno behavior may lead to non-uniqueness of sliding solution even when applying Filippov's approach, as the computation of sliding equivalent dynamics requires, in this case, solving stiff nonlinear equations for which a unique solution is not guaranteed.

1.3.2 Other Literature Survey on Geometric-Zeno

Due to its complex nature, geometric-Zeno behavior has been studied in many forms and from many different perspectives, especially from asymptotic stability perspective.

A technique that has been proposed in the hybrid systems literature to deal with geometric-Zeno is that of regularizing the original system, which was illustrated for particular examples in [97, 103, 107]. This technique is based on perturbing the dynamical system in order to obtain non-Zeno solution, and then taking the limit as the perturbation goes to zero. However, such perturbation may invalidate the notion of instantaneous discrete transitions (i.e. mode switchings). Consequently, this can result in models that are stiff, and as a result the simulation performance might run into trouble. Furthermore, depending on the methods of regularizations, different behavioral scenarios after the Zeno limit point may be generated.

Necessary and sufficient conditions for geometric-Zeno behavior in linear complementarity systems were provided in [95, 99], and more recently in [98].

In the context of Lagrangian hybrid systems, Ames et al. have shown that geometric-Zeno limit points belong to the zero set of the unilateral switching function, with velocity vector being tangential to this switching surface [106]. Therefore, they postulated that after the Zeno time, the system switches to holonomically constrained dynamics, where the holonomic constraints are based on the unilateral constraints on the configuration space that originally defined the hybrid system.

In [102], Lamperski and Ames provided Lyapunov-like conditions for geometric-Zeno behavior in Lagrangian hybrid systems near isolated Zeno equilibrium points. The results in [102] were later extended in [105], where Zeno stability approach was described as a special form of asymptotic stability. Moreover, [105] provided Lyapunov conditions for Zeno stability of compact sets. More recently, the results of [102] were extended in [96] to Zeno equilibria that are non-isolated. Sufficient conditions for geometric-Zeno behavior in a special class of hybrid systems, called first quadrant hybrid systems were derived in [100]. This work was then generalized in [104].

We observed that most of the proposed conditions for geometric-Zeno behavior tend to be conservative, and apply to particular classes of hybrid systems, i.e. non-smooth mechanics, Lagrangian hybrid systems, first quadrant hybrid systems, linear complementarity systems, etc. We believe that this is because geometric-Zeno behavior is a global problem in its nature, which prevents the use of local methods in its regularization. Furthermore, while understanding geometric-Zeno in these domains is quite sophisticated, there is no proposition on how such methods can serve in a hybrid semantic model design for modeling and simulation tools.

1.4 Motivations

From a practical simulation perspective, in order to guarantee a maximal robustness of any simulation framework, it is important to define, at a first stage, a proper semantic model for simulation. In fact, this is a mandatory step even before designing the simulator or the language. The development of a simulation framework typically include the following steps:

1. Defining properly a semantic model that can account for the expected properties of the systems under simulation.

2. Designing and developing a simulator that could approximate the model dynamics conforming to the defined semantic model. This step can also be used to validate the semantic model (i.e. a semantic model is valid if all conforming models can be simulated).

3. Designing a language capable of expressing all models elements and components conforming to the semantic model. Type-checking must be included in this step to prove statically the semantic validity of the simulated models.

4. Designing a compiler for the language. The compiler should be capable of performing static checks of models and also rejecting models that are invalid.

This dissertation focuses on the first two steps above. It takes the perspective that models of hybrid systems can be written as executable codes or programs in programming languages that support and provide hybrid systems semantics.

Many modeling and simulation tools for hybrid systems have been developed in the past years. They can be classified into two categories: those who put special attention on defining models rigorously, such as for instance SpaceEx [44], Ptolemy [27] (based on the super-dense time semantics in [47]), and Zélus [57] (based on the non-standard semantics in [54]); and those who use informal approach for model definition such as Simulink[4], Modelica language [55] and its associated tools. All these modeling and simulation tools share the same approach of hybrid model execution alternating between continuous evolution and sequences of discrete switchings [57] as defined by the notion of hybrid automata [30]. For all of these tools, the operational semantics of continuous dynamics (differential equations) is not included in the core semantic model: numerical solvers execute the continuous behavior by advancing time and computing the values of physical continuous variables. None of the above tools have a Zeno-free semantic model. They all rely on analyzing the solver output at each integration time step, with the solver behavior at the Zeno-limit point being usually unspecified.

Our Motivation: Because in the physical hybrid system a solution exists past the Zeno limit point, simulation tools for hybrid systems should be able to predict correctly the solution after Zeno, and to provide an efficient Zeno detection and avoidance support. This motivates the need of Zeno-free semantics for simulation synthesis to allow for the Zeno execution (or trajectory) to be carried correctly past the Zeno limit point. This is, in fact, the main motivation of our work.

1.5 Contributions

Our contribution is proposing a new Zeno-free computational framework for semantic model design, and how this Zeno-free computational framework can be implemented in the design of hybrid systems simulators.

The first part of our contribution is proposing non-standard semantics for Zeno executions of hybrid systems models. This is based on interpreting Zeno executions in a non-standard *densely ordered* hybrid time domain.

1.5.1 Non-Standard Semantics for Zeno Executions

The advantages of using non-standard semantics in the analysis of Zeno behavior is that the completeness in the space of the continuous dynamics and discrete dynamics is naturally introduced so that it allows for solutions of Zeno hybrid models to be well-defined beyond the Zeno limit points:

[4]https://fr.mathworks.com/products/simulink.html

1. The continuous dynamics of the hybrid system is reduced to the recurrence equation that represents the infinite iteration of infinitesimal discrete changes with infinitesimal duration. Therefore, we can handle the hybrid dynamics based only on fully discrete paradigm.
2. The representation of dynamics based on non-standard analysis is complete and the exact limit point of discrete change, like chattering-Zeno and geometric-Zeno limit points, can be handled.

The second part of our contribution is to propose a rigorous Zeno-free computational framework for hybrid semantic model design and implementation. The key idea in our proposed computational framework is to perform Zeno detection and avoidance by using behavioral analysis of the system, instead of only considering zero-crossings in a hybrid time domain. The behavioral analysis technique we propose for treating Zeno is based on analyzing both types of Zeno systematically. We provide formal conditions on when the simulated models of hybrid systems display chattering-Zeno and geometric-Zeno executions. We also provide methods for carrying solutions beyond Zeno. The issue of existence and uniqueness of a solution beyond Zeno is also studied in this dissertation. Our Zeno-free computational framework allows sacrificing the notion of *Zeno-freeness* as: i) the decision on whether or not the Zeno limiting state is chattering-Zeno or geometric-Zeno is based on formal conditions explicitly defined and provided to the hybrid simulator's solver, and ii) the correct notion of solution beyond Zeno is well defined and established in our framework.

Our approach also supports mixing compile-time transformations of hybrid programs (i.e. generating what is necessary for Zeno detection and avoidance), the decision, in run-time, for the urgent transition from pre-Zeno to post-Zeno state (based on formal conditions for the existence of Zeno), and the computation, in run-time, of the system dynamics beyond Zeno.

1.5.2 Behavioral-based Zeno Detection and Avoidance

Our behavioral analysis technique of treating Zeno is based on analyzing both types of Zeno systematically. We provide formal conditions on when the simulated models of hybrid systems display chattering-Zeno and geometric-Zeno executions. We also provide methods for carrying solutions beyond Zeno. The issue of existence and uniqueness of solution beyond Zeno is also studied and well-established in this dissertation.

- **Chattering-Zeno Detection and Avoidance:** We propose a novel sliding mode computational framework for simulation semantic model design. It establishes solvability requirements for simulating hybrid models while effectively performing chattering-Zeno detection and avoidance. The main benefit of our proposed framework is that chattering-Zeno detection and elimination is done "on the fly" without any need neither to add small hysteresis to the zero-crossing event functions, nor to solve stiff nonlinear equations for the computation of the equivalent sliding dynamics in case of chattering-Zeno on discontinuous surfaces intersections of high dimensions. Our computational framework establishes an automatic switching between transversality mode and sliding mode simulation, as well as integrating each particular state appropriately and localizing the structural changes in the system in a highly accurate way.

Furthermore, we show by a special hierarchical application of convex combinations that unique solutions can be found in general cases when the discontinuous switching manifold (on which chattering-Zeno occurs) takes the form of finitely many intersecting discontinuous surfaces, so that an efficient numerical treatment of the sliding motion constrained on the entire discontinuity region, including discontinuous surfaces intersections, is guaranteed.

- **Geometric-Zeno Detection and Avoidance:** We propose a novel technique for detecting and eliminating "orbitally" geometric-Zeno behavior. In particular, we derive sufficient conditions for the existence of geometric-Zeno behavior, and also we provide methods on how to allow for solutions to be carried beyond the geometric-Zeno point. A cyclic path is a prerequisite and a necessary condition for a hybrid system's state machine to accept geometric-Zeno executions. In our proposed technique, we consider that every pair of two consecutive triggers of the same guard as the input argument for detecting cycles. We show that the execution of a hybrid system's model converges to a geometric-Zeno limit point if all the cycles — detected during the execution — converge. We consider that the evolution of the hybrid solution trajectory through cycles is described by a transition system with a sequence of hybrid states. In order to derive sufficient conditions for the hybrid system's execution to be convergent to the Zeno limit point, we study based on non-standard analysis the existence of a non-standard contraction map in a complete metric space, and the convergence of the solution to a geometric-Zeno limit point, through such map, according to a Cauchy sequence. Such map indicates when exactly a decision should be taken to transition the solution from pre-Zeno to post-Zeno, and thus eliminating Zeno behavior.

1.5.3 Prototype Implementations

A part of this dissertation was attributed to design, test, and validate Zeno-free simulator prototypical implementations. The motivations behind such implementations are the followings:

1. The first motivation was to validate our proposed Zeno-free computational framework in standard semantics model of time. The emphasis of the validation is one of correctness and preciseness.

2. The second motivation is to provide -via these implementations- a guidance for the development of a Zeno-free version for hybrid systems simulation tools which use standard model of time, giving a Zeno-free computational framework for an ideal manipulation of Zeno.

We have developed, tested, and validated two Zeno-free simulator implementations for chattering-Zeno detection and elimination. The first Zeno-free simulator implementation was developed in the language Acumen, while the second one was developed conforming to the Functional Mock-up Interface (FMI) standard for Model Exchange.

Also we have developed, tested, and validated two Matlab/Simulink Zeno-free simulator implementations for geometric-Zeno detection and elimination. The Matlab implementation includes a stand-alone simulator written in Matlab code, while the Simulink implementation includes the basic ready-to-use Zeno-free Simulink library blocks that allow for geometric-Zeno detection and elimination of any Zeno hybrid model exhibiting geometric-Zeno.

The motivation behind choosing different simulation tools for the prototype implementations is to show that the methods proposed in this thesis for Zeno-free simulation can be implemented in any hybrid systems simulation tool. Also another motivation is to give via these implementations — in different simulation tools — a guidance for the development of a Zeno-free version of hybrid systems simulation tools.

1.6 Outline

The structure of this dissertation is organized as follows:

Chapter 2 introduces the concepts and definitions relevant for understanding and developing the theoretical results of this thesis. In this chapter we also present our contribution of proposing non-standard semantics for Zeno executions of hybrid systems models. We start this chapter by giving a thorough introduction to dynamical systems in Section 2.1. Afterwards, we introduce hybrid systems in Section 2.2, including their model formalization and executions. We start Section 2.2 by giving a brief introduction to discrete event systems (Section 2.2.1), and then we give an overview to hybrid systems modeling (Section 2.2.2), and the mathematical representation of hybrid systems (Section 2.2.3). We introduce also hybrid automata (Section 2.2.4) as a modeling framework for hybrid systems. A number of examples are presented to illustrate, in a hybrid automata formalism, the interaction between the time-driven continuous variable dynamics and the event-driven discrete logic dynamics. In Section 2.2.5 we discuss the limitation of the standard semantics of hybrid automata in the full treatment of hybrid dynamics. To overcome this problem, we present in Section 2.2.6 a non-standard semantics for hybrid automata as well as a unified rigorous definition of Zeno behavior based on a non-standard analysis of the hybrid time domain. Finally, we give an introduction to hybrid simulation in Section 2.3, and we describe shortly some languages and tools for hybrid simulation in Section 2.4.

In Chapter 3 we investigate chattering-Zeno behavior in details. We start this chapter by giving an brief introduction to chattering-Zeno executions. To illustrate the problem of simulating chattering-Zeno executions, we provide next a set of representative examples and realistic case studies of chattering-Zeno models (Section 3.1.1), and then we discuss the challenges when simulating their executions (Section 3.1.2). Afterwards, we present in details Fillipov's equivalent dynamics approach as a method proposed in the literature to deal with chattering-Zeno, and we discuss the limitation of this approach in treating chattering-Zeno (Section 3.2). Next we provide our chattering-free simulation technique for the purpose of detecting and eliminating chattering-Zeno behavior “on the fly” during the simulation, including treating chattering-Zeno on the intersection of finitely many intersected discontinuous surfaces (Section 3.3). Finally, we sketch two prototype implementations for applying the chattering-free computational framework to hybrid systems simulation tools (Section 3.4). Simulation results as well as a performance analysis of our proposed chattering-free technique and implementations are also presented.

In Chapter 4, we investigate geometric-Zeno behavior of hybrid models in details. We start this chapter by giving a brief introduction to geometric-Zeno behavior (Section 4.1.1), and then we show by a set of illustrative examples the problem of simulating hybrid systems models exhibiting geometric-Zeno behavior (Section 4.1.2 and Section 4.1.3). Afterwards, we present a method for geometric-Zeno detection and elimination (Section 4.2). We derive sufficient conditions for the existence of geometric-Zeno behavior based on the existence of a non-standard contraction map in a complete metric space, and the convergence of the solution to a geometric-Zeno limit point, through such map, according to a Cauchy sequence. Such map indicates when exactly a decision should be taken to transition the solution from pre-Zeno to post-Zeno, and thus eliminating Zeno behavior. Simulation results —using a prototype implementation of the proposed technique— will be presented at the end of this chapter (Section 4.3).

Chapter 5 is dedicated to the conclusion and future work.

Chapter 2

Modeling and Simulation of Hybrid Systems

We start this chapter by giving a thorough introduction to dynamical systems in Section 2.1, including the basic notations and formal definitions behind the theory of continuous-time and discrete-time dynamical systems, as well as the solution concepts. Afterwards, in Section 2.2, we introduce hybrid systems, including their model formalization and executions. We start Section 2.2 by giving a brief introduction to discrete event systems (Section 2.2.1), and then we give an overview to hybrid systems modeling (Section 2.2.2), and the mathematical representation of hybrid systems (Section 2.2.3). We introduce also hybrid automata (Section 2.2.4) as a modeling framework for hybrid systems. A number of examples are presented to illustrate, in a hybrid automata formalism, the interaction between the time-driven continuous variable dynamics and the event-driven discrete logic dynamics. In Section 2.2.5 we discuss the limitation of the standard semantics of hybrid automata in the full treatment of hybrid dynamics. To overcome this problem, we present in Section 2.2.6 a non-standard semantics for hybrid automata as well as a unified rigorous definition of Zeno behavior based on a non-standard analysis of the hybrid time domain. Finally, we give an introduction to hybrid simulation in Section 2.3, and we describe shortly some languages and tools for hybrid simulation in Section 2.4.

2.1 Dynamical Systems

Concerning the qualitative definitions of dynamical systems, we can find two main features in the literature. First, a system consists of interacting "components" and, secondly, a system is associated with a "function" it is intended to perform.

In order to analyze and to develop design and control techniques for systems, quantitative definitions are needed. Hence, a model or an abstraction of a system is sought. A model is a mathematical device (or tool) which mimics the behavior of a system. A mathematical model describes the relations between different variables and quantities in a system by means of mathematics and physical laws.

Dynamical systems can be defined as systems whose behavior depends on the present and the past values of their variables.

For a dynamical system, three important concepts need to be distinguished, i) the state variables x, ii) the inputs u, and iii) the outputs y. The state variables correspond to the minimal information that is necessary to describe the system behavior uniquely for a given time instant. In other words, the state variables represent the "memory" that the system has of its past. The state of a system refers to a specific value of state variables and is generally denoted by x. The inputs u constitute the external variables that influence the system. The outputs y are typically the variables that can be measured, and they can be also variables acting as inputs to open connected systems.

Before introducing in more details the mathematical framework and solution concepts of dynamical systems we start by giving a brief introduction to ODEs and flows.

2.1.1 Ordinary Differential Equations (ODEs) and Flows

We start by giving a short review on the classical theory of ordinary differential equations (ODEs).

Notation 2.1. *Let $A \subseteq \mathbb{R}^n$, $B \subseteq \mathbb{R}^m$, and $k \in \mathbb{N}$. We denote by $C^k(A,B)$ the set of functions $A \to B$ that have continuous derivatives up to the order k. We abbreviate $C^0(A,B) = C(A,B)$, $C^\infty(A,B) = \cap_{k\in\mathbb{N}} C^k(A,B)$, and $C^k(A) = C^k(A,\mathbb{R})$.*

An implicit ordinary differential equation (ODE) is typically of the form

$$F(t, x, x^{(1)}, x^{(2)}, \cdots, x^{(k)}) = 0 \tag{2.1}$$

for unknown $x \in C^k(J)$, where $J \subseteq \mathbb{R}$ being an interval, and $x^{(j)}$ being the j-th derivative of x [109]

$$x^{(j)}(t) = \frac{d^j x(t)}{dt^j}, \quad j \in \mathbb{N}. \tag{2.2}$$

Here the relation F satisfies $F \in C(A)$ where A is a subset of $\mathbb{R}^{k+2}$ that is open. Frequently, we call the variable t the **independent** variable, and the variable x the **dependent variable**.

The **order** of the implicit ODE (2.1) (sometimes it is called the **index of smoothness** of the differential equation) is the highest derivative that appears in the relation F.

A **solution** for the implicit ODE (2.1) is a function $\varphi(t) \in C^k(I)$, satisfying

$$\text{For all } t \in I: \quad F(t, \varphi(t), \varphi^{(1)}(t), \varphi^{(2)}(t), \cdots, \varphi^{(k)}(t)) = 0 \tag{2.3}$$

with $I \subseteq J$ being an interval. This implicitly implies $(t, \varphi(t), \varphi^{(1)}(t), \varphi^{(2)}(t), \cdots, \varphi^{(k)}(t)) \in A$ for all $t \in I$.

As it is dependent on the initial state $x_0 = x(t_0)$, the solution of (2.1) can be written as $\varphi(t, x_0)$. This solution depending on initial conditions is called the **flow** of the differential equation.

Definition 2.2 (Flow) *The flow $\varphi(t, x_0)$ is the solution of the ODE (2.1) with $x(t_0) = x_0$ being the initial state.*

We will assume that the general differential equation in (2.1) can be solved for the highest derivative of x. This results in an explicit ODE given by

$$x^{(k)} = f(t, x, x^{(1)}, x^{(2)}, \cdots, x^{(k-1)}), \tag{2.4}$$

where in this new form we have $f \in C(A)$, with the subset $A \subseteq \mathbb{R}^{1+k}$ being open. Implicitly, we can do this locally in the neighborhood of some point $(t, y) \in A$ if, at the point (t, y), the partial derivative which is taken with respect to the highest derivative does not vanish. That is, $\frac{\partial F}{\partial y_k}(t, y) \neq 0$. The case in which the dependent variable x being $x : J \subseteq \mathbb{R} \to \mathbb{R}^n$ gives us a **system of ODEs** of the form

$$\begin{aligned} x_1^{(k)} &= f_1(t, x, x^{(1)}, x^{(2)}, \cdots, x^{(k-1)}), \\ &\cdots, \\ x_n^{(k)} &= f_n(t, x, x^{(1)}, x^{(2)}, \cdots, x^{(k-1)}). \end{aligned} \tag{2.5}$$

This system is linear **homogeneous** if we can write it in the form

$$x_i^{(k)} = \sum_{l=1}^{n} \sum_{j=0}^{k-1} f_{i,j,l}(t, x_l^{(j)}). \tag{2.6}$$

In addition, we can always reduce any system to a first-order system by substituting a new set of variables $y = (x, x^{(1)}, x^{(2)}, \cdots, x^{(k-1)})$. This yields the **first-order system**

$$\begin{aligned} \dot{y}_1 &= y_2, \\ &\cdots, \\ \dot{y}_{k-1} &= y_k, \\ \dot{y}_k &= f(t, y). \end{aligned} \tag{2.7}$$

Also, in order to make the right-hand side to be independent of the variable t, we can add the independent variable t to the dependent variables $(t, y) = z$

$$\begin{aligned} \dot{z}_1 &= 1, \\ \dot{z}_2 &= z_3, \\ &\cdots, \\ \dot{z}_k &= z_{k+1}, \\ \dot{z}_{k+1} &= f(z). \end{aligned} \tag{2.8}$$

This system is called **autonomous** since the function f does not depend on the time variable t.

From a physical modeling point of view, it is sufficient to consider only first-order autonomous systems in the modeling of dynamical systems, since — as mentioned above — any higher order ODE, hence any higher order system, can be reduced to a (larger) first order system.

2.1.2 Continuous-Time/Discrete-Time Dynamical Systems

Let's consider a (nontrivial) first-order autonomous system of ODEs

$$\dot{x} = f(x), \quad x(0) = x_0, \tag{2.9}$$

where the variable $x : \mathbb{R} \to \mathbb{R}^n$ represents the system's state, and $f : \mathbb{R}^n \to \mathbb{R}^n$ is a right hand side (r.h.s) function that represent the continuous dynamics of the system. Usually, we can write most of the ODEs in this form. This kind of system of ordinary differential equations typically appears in a very wide variety of physical systems applications such as biological, physical, electrical, mechanical, chemical, economical applications, and a lot of other engineering and technological applications. It is even impossible to list all the applications where this type of ODEs is used to define the system dynamics [109].

For the differential equation having initial condition $p = x(0)$, we denote its solution by $\varphi(t, p)$. It is easy to prove that the function φ is continuously differentiable and satisfies the assumptions in the following definition of a continuous-time dynamical system.

Definition 2.3 (Continuous-Time Dynamical System) *A function $\varphi : \mathbb{R}\times\mathbb{R}^n \to \mathbb{R}^n$ is a continuous-time dynamical system if this function satisfies the following conditions:*

- *For all $p \in \mathbb{R}^n$, it holds that $\varphi(0, p) = p$,*
- *For all $p \in \mathbb{R}^n$ and $s, t \in \mathbb{R}$, it holds that $\varphi(t, \varphi(s, p)) = \varphi(t + s, p)$,*

where $\mathbb{R}^n$ represents the state space of the system, $p \in \mathbb{R}^n$ represents the system's state, and $\varphi(t, p)$ is the system's state after time t starting from p.

As we have stated above, a continuous-time dynamical system can be determined by the solution of an autonomous first order system of ODEs (if necessary after rescaling of time t). Furthermore, we can define an autonomous system of ODEs from a given continuous-time dynamical system. Therefore, a continuous-time dynamical can be seen to be an equivalent notion to an autonomous system of ODEs. Definition 2.3 can be extended in many directions. One main alternative is when the time scale is discrete. In such a case, the set of real numbers $\mathbb{R}$ is replaced by the set of real integers $\mathbb{Z}$. Such extension leads us to the notion of discrete-time dynamical systems.

Definition 2.4 (Discrete-Time Dynamical System) *A function $\varphi : \mathbb{Z} \times \mathbb{R}^n \to \mathbb{R}^n$ is called a discrete-time dynamical system if it satisfies the following conditions:*

- *For all $p \in \mathbb{R}^n$, it holds that $\varphi(0, p) = p$,*
- *For all $p \in \mathbb{R}^n$ and $m, k \in \mathbb{Z}$, it holds that $\varphi(k, \varphi(m, p)) = \varphi(m + k, p)$.*

Similarly to the case of continuous-time dynamical systems where the system was obtained from first-order autonomous ordinary differential equations, discrete-time dynamical systems can be obtained from difference equations. As in the case of continuous-time dynamical systems the notion "flow" is used, in the case of discrete-time dynamical systems the notion "map" is used. Namely, consider the function $g : \mathbb{R}^n \to \mathbb{R}^n$ that is a map of a state space into itself, and the difference equation (or recursion) $x_{k+1} = g(x_k)$ that have an initial condition $p = x_0$, where x_k is the state at the discrete time $k \in \mathbb{Z}$. If the solution function $\varphi(k, p) = x_k$ of this difference equation satisfies the conditions in Definition 2.4, then we can consider that the difference equation $x_{k+1} = g(x_k)$ represents a discrete-time dynamical system.

Both discrete-time and continuous-time dynamical systems can be dealt with concurrently. Therefore, as a common notation for both $\mathbb{Z}$ and $\mathbb{R}$, we denote the set of points in time by $\mathbb{T}$.

In general, the main objective of studying dynamical systems is to characterize geometrically the system's orbits. Orbits in continuous-time/discrete-time dynamical systems can be defined as follows.

Definition 2.5 (Orbit) *We define the orbit of a dynamical system as the set $\{\varphi(t, p) : t \in \mathbb{T}\}$ starting from an initial point p. This set is a curve in continuous-time dynamical systems. In discrete-time dynamical systems, it is represented by a sequence of points.*

2.2 Hybrid Systems

A particular feature of many complex dynamical systems is that they are hybrid systems, i.e. the mathematical model of the system changes its operational evolution in time depending on some indicators. As we have stated before, this a modeling issue because of the approximation of the fast nonlinear phenomena of dynamical physical systems, which typically results in non-smooth dynamical systems with discrete mode switchings (transitions). The term *non-smooth dynamics* is related to dynamical systems for which the state is not required to be a smooth (differentiable) function of time. Such systems are usually represented by different mathematical formalisms, such as switching systems, piecewise systems, projected dynamical systems, complementarity dynamical systems (there exists many different form of these systems), differential inclusions, variational evolution inequalities, impulsive ODEs, and so on.

Combining time-driven and event-driven dynamics, the hybrid systems framework can be thought as an important and effective tool for modeling non-smooth dynamical systems, as well as engineering systems that combine analog and digital devices, interact through networks, conduct tasks collaboratively, and operate in environments filled with uncertainties. In general, any dynamical system with discrete logical modes that interact with continuous variable states can be modeled in this framework.

Typical examples for hybrid systems are electrical circuits with ideal (or idealized) components, where for example the system contains different devices for different ranges of frequency, or switching components like electric switches or diodes are included in the system. For example, the presence of diodes in circuits may imply unilateral constraints on currents and voltages, which in turn results in a switched system. Consider for example a Buck converter whose schematic is sketched in Figure 2.1, where L is an inductor, Q is a MOSFET transistor, D is a diode, R is a resistance, C is a capacitor, and E is the stored energy. By applying Kirchoff's laws, the dynamics of the converter

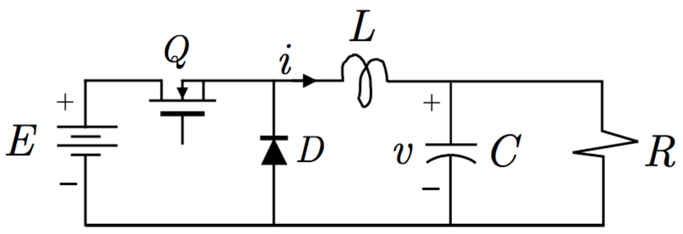


Figure 2.1: Schematic of the Buck converter.

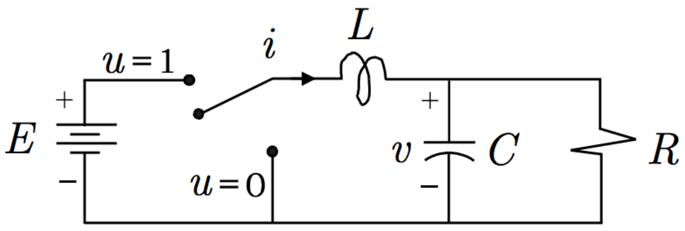


Figure 2.2: Ideal representation of the converter's switching.

model is given by

$$L\frac{di}{dt} = -v + uE, \tag{2.10}$$

$$C\frac{dv}{dt} = i - \frac{v}{R}, \tag{2.11}$$

where u is a control input that is used in the converter's model to represent the switching between the diode's non-conducting mode (when the control u have the value $u = 1$) and the diode's conducting mode (when $u = 0$); see Figure 2.2. Another important class of applications that display hybrid behavior are non-smooth mechanical systems with unilateral constraints, such as unilateral contact, dry friction, and impact phenomena. Consider for example the famous model of a bouncing ball, whose trajectory can be

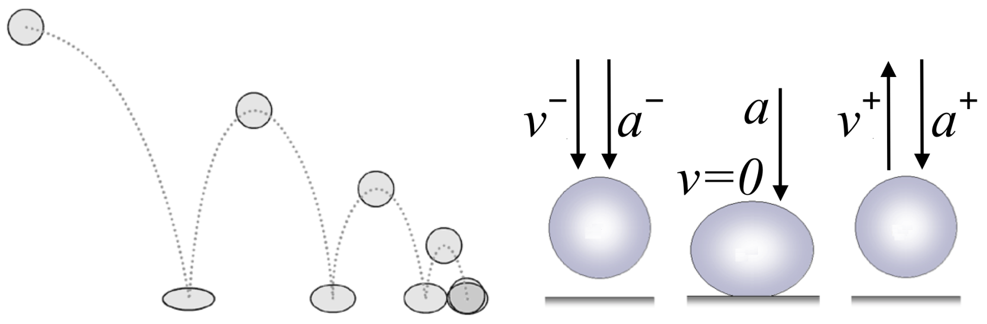


Figure 2.3: Trajectory of the bouncing ball system.

best represented as a trajectory of an impact system. In such model example, the ball has a non-vanishing pre-contact velocity v^- just before the contact with the ground. Once the ball is in contact with the ground, the velocity has to jump to a post-contact velocity v^+, otherwise it penetrates the ground. Figure 2.3 illustrates the trajectory of the bouncing ball model. In general, non-smooth models of mechanical systems are often used in plasticity and contact dynamics theory. Another class of applications that can also be modeled with the hybrid systems framework are structure varying systems with changing number of degrees of freedom, such like robot manipulators or automatic gear-boxes.

Hybrid systems can be defined as *dynamical systems having components with interacting continuous-time dynamics and discrete-event dynamics, where discrete-event actions are combined with continuous-time physical processes.* The main characteristic of any hybrid system is that it consists of two different types of states variables; continuous states taking values in a non-denumerable set, usually real numbers $\mathbb{R}^n$, and discrete states taking values in a countable set. A hybrid model is, then, one that specifies evolution of the discrete states and continuous states, and also specifies the interaction between the continuous-time and discrete-event dynamics. We have defined continuous-time dynamical systems in Section 2.1.2. Before we give a mathematical representation of hybrid dynamical systems, we first introduce, in the following, discrete event systems.

2.2.1 Discrete Event Systems

From a control engineering point of view, a discrete event system can be seen as an event-driven discrete-state dynamical system whose evolution depends completely on the discrete occurrence of events. A discrete event system consists essentially of event-driven discrete transition mechanisms of states and a discrete phase (or state) spaces. The discrete transitions between the states are combined with events which asynchronously occur at points in time that are discrete. Thus, the system's behavior is typically observed in terms of sequences of discrete events. Discrete event systems are in general represented by finite automata. An automaton is a device which generates a sequence of state transitions in accordance with a set of well-defined rules. The term finite automata reveals that the state space of the discrete event system is considered to be finite.

Definition 2.6 (Finite Automata) *A finite automaton is a tuple $(Q, E, \pi, Init)$, where Q is a finite set of discrete states or locations whose elements are often denoted by $q \in Q$, E is a finite set of input symbols or events denoted by $\xi \in E$, $\pi \subseteq Q \times E \times Q$ is a transition relation, and $Init \subseteq Q$ is the set of initial states. The transition relation defines the next discrete states q^+, such that $(q, \xi, q^+) \in \pi$. Next states q^+ refer to a discrete state that can be reached after a transition caused by the occurrence of the event ξ. A finite automaton is called a deterministic finite automaton (DFA) if the relation π is a function $\pi : Q \times E \to Q$.*

The set E is also known as an alphabet. Finite automata can also be represented by transition diagrams or graphs with vertices given by the elements of Q, and the edges by the transition rules or events.

2.2.2 Modeling of Hybrid Systems

In terms of control theory, a continuous-time system (i.e. plant) and a discrete-event system (i.e. controller) form together a hybrid system. The continuous-time system is modeled using ODEs, whereas the discrete-event system is a computer program represented by a discrete transition system. In a hybrid system composed of continuous-time and discrete-event sub-systems, each sub-system can interact with the other; see Figure 2.4. The continuous-time behavior of the hybrid system is usually influenced by discrete-event components when the value of the continuous-time variables is changed by discrete-event actions. A continuous-time subsystem can also influence the discrete-event behavior, as a boolean condition which depends on the value of the continuous-time variables becomes true and triggers a discrete-event action. The discrete-event and continuous-time sub-systems — composing a hybrid system — have distinct types of signals; see Figure 2.5. For continuous-time systems, all output signals are continuous waveforms, while for discrete-event systems the activation signals are discrete and the output signals are piecewise constant.

When putting together these two domains, the signals at the boundaries must be treated carefully. A Zero-Order Hold (ZOH) converts a discrete-event into a continuous waveform to have a value at the time points were no events occur. The output of the resulting hybrid dynamical system is a piecewise continuous function.

Basically, the discrete-event part is activated by two types of events: 1) *time events* that occur at a given time, and 2) *state events* that appear when a variable reaches a certain threshold value. The continuous-time part can make a change in the discrete-event part only if it generates a state event; see Section 1.2.2.

- **Time events:** Time events occur at a specified future time when the event is scheduled, such as the sampling-time of a discrete-time controller. Time events are easier to handle as they may be specified directly before the simulation begins. The integration thereby may be stopped and restarted exactly at the possible discontinuity point. An example of time events is process where an operator resets a valve opening at a specified time.

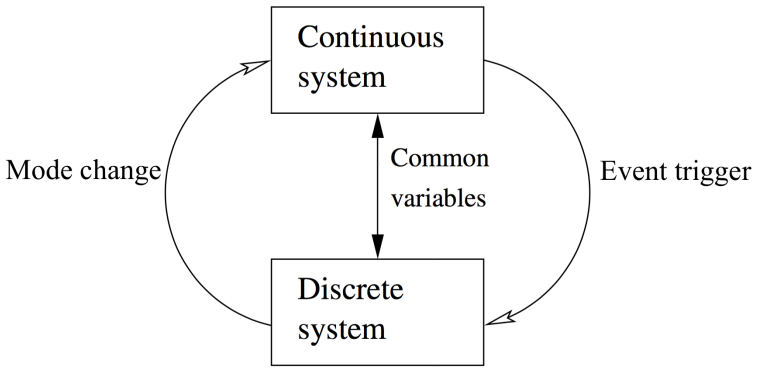


Figure 2.4: Continuous-time and discrete-event parts interaction in a hybrid system.

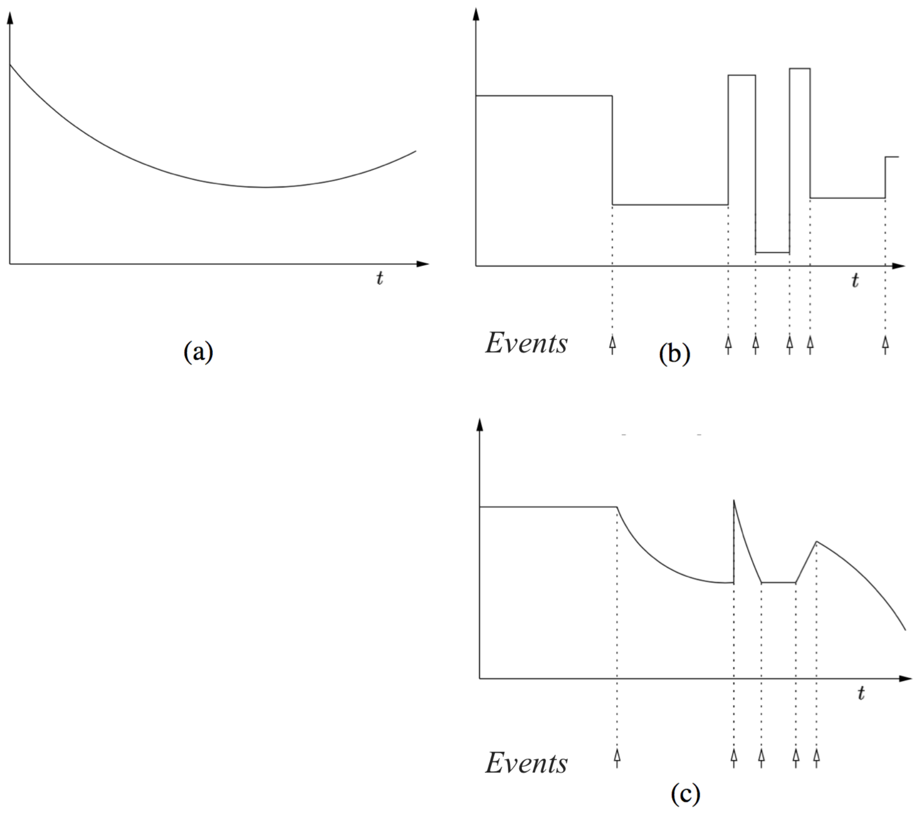


Figure 2.5: Output signals: (a) continuous-time dynamical system, (b) discrete-event dynamical system, and (c) hybrid dynamical system.

- **State events:** It is possible for a hybrid system to have several possible configurations whereby in each configuration the behavior of the system is described by a system of differential equations. Such a system switches from one configuration to another based on a condition. This condition may depend on the input control of the system (such as human operators intervention), or may depend on system states. In general, when modeling with state events, a switching from a given operational mode to another one takes place when some constraints on the continuous-time state variables are fulfilled. Therefore, the timing of discrete switching can be seen as a function of the solution to the ordinary differential equations that govern the system. The transition time is given by a transition condition which depends on a zero-crossing event function $z(t)$; see Section 1.2.2. Whenever $z(t)$ crosses zero, the transition condition switches its logical value. In this case, an event is defined as the earliest time at which one of the currently pending transition conditions becomes true. *State events are then the mechanism whereby the state of the continuous-time subsystem influences the discrete-event* *subsystem.* State events which are dependent on state variables of the system are more difficult to handle accurately in time than the time events. One possible way to detect state events is by calculating the value of the zero-crossing function at the borders of each time step interval. To locate precisely the instant in time where the state event appears, a bracketing may be employed in the time step interval at which the state event is detected.

Numerical simulation tools and languages such as Modelica, HyVisual, Charon, Acumen, Zélus, Simulink/Stateflow, Scicos, Shift, and many other languages and tools are usually used in modeling of hybrid systems. Such tools also provide an integrated environment for simulation, as well as test automation and code generation.

2.2.3 Mathematical Representation of Hybrid Systems

Definition 2.7 (Hybrid System) *Following [70], a mathematical representation of a hybrid system $\mathcal{H}$ can be given either in a set-valued mapping of the form:*

$$\mathcal{H}: \begin{cases} \dot{x} \in F(x), & \text{if } x \in C, \\ x^{+} \in G(x), & \text{if } x \in D, \end{cases} \tag{2.12}$$

or in a less general representation involving equations:

$$\mathcal{H}: \begin{cases} \dot{x} = f(x), & \text{if } x \in C, \\ x^{+} = g(x), & \text{if } x \in D, \end{cases} \tag{2.13}$$

where $C \subset \mathbb{R}^n$ is the ***flow set****, $D \subset \mathbb{R}^n$ is the* ***jump set****. The state of the hybrid system, denoted by x, can change according to a* ***flow map*** *given by a differential inclusion $\dot{x} \in F(x)$ or differential equation $\dot{x} = f(x)$ while evolving in the flow set C, and it can change according to a* ***jump map*** *defined by a difference inclusion $x^+ \in G(x)$ or difference equation $x^+ = g(x)$ while in the jump set D. $\dot{x}$ represents the velocity of x, while x^+ represents the value of the state after a discrete instantaneous change. In (2.13), f, respectively g, is assumed to be defined on at least the set C, respectively D. In (2.12), F, respectively G, is assumed to have nonempty values on C, respectively D.*

Remark 2.8 A special class of hybrid systems are non-smooth discontinuous dynamical systems (DDS), e.g. ODEs with discontinuous r.h.s (right hand side). A DDS may have one or multiple hyper discontinuous surfaces, where each hyper discontinuous surface $\Sigma = \{x \in \mathbb{R}^n : \ \gamma(x) = 0\}$ splits the phase space into two disjoint regions $\mathcal{S}_i = \{x \in \mathbb{R}^n : \ \gamma(x) > 0\}$ and $\mathcal{S}_j = \{x \in \mathbb{R}^n : \ \gamma(x) < 0\}$ (Figure 2.6), defined as the set of states where a smooth event function $\gamma(x)$ is positive and negative, respectively. In this case, each hyper surface of discontinuity Σ is a switching manifold of dimension $\mathbb{R}^{(n-1)}$. This manifold is included in $\partial\mathcal{S}_i$ and $\partial\mathcal{S}_j$ which are the boundaries of the disjoint regions $\mathcal{S}_i$ and $\mathcal{S}_j$, respectively. That is, $\Sigma = \overline{\mathcal{S}}_i \cap \overline{\mathcal{S}}_j$. The flows $\varphi_i(t, x(0))$ and $\varphi_j(t, x(0))$ in the regions $\mathcal{S}_i$ and $\mathcal{S}_j$ on both sides of Σ are both well defined.

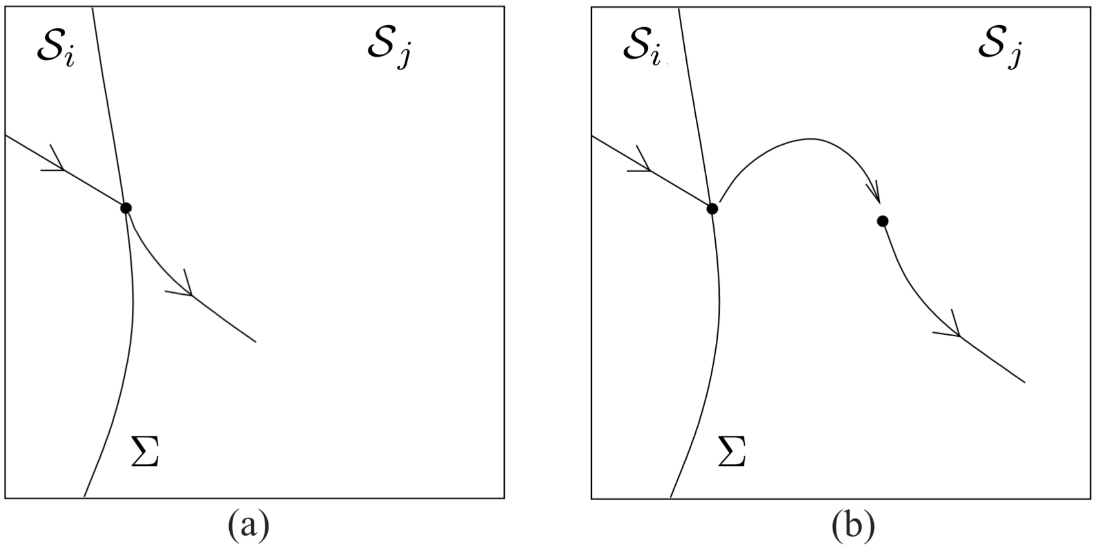


Figure 2.6: Hybrid trajectories for DDS: (a) A hybrid solution trajectory with identity reset jump map, (b) a hybrid solution trajectory with non-identity reset jump map.

2.2.4 Hybrid Automata Modeling Formalism

The hybrid automata formalism is one of the main modeling formalisms in hybrid systems theory. It has been developed as a formalism to model systems in which discrete control logic interacts with a real-valued reality, and to facilitate mathematical proofs about their behavioral properties. The focus lies on dependability of such systems and hence on formal verification of properties. Besides their formal semantics, hybrid automata offer a pleasing visual notation accessible with only a minimum of formal training.

2.2.4.1 Syntax

Hybrid automata results from an extension of finite-state machines by associating with each discrete state a continuous state model. A hybrid automaton typically consists of locations, transitions, invariants, guards, $\mathbb{R}^n$-dimensional continuous functions, jump functions, and synchronization labels [1, 2].

In particular, hybrid automata integrate diverse models such as differential equations and state machines in a single formalism with a uniform mathematical semantics for multi-modal control synthesis and for safety and real-time performance analysis.

By combining the definition of a continuous-time dynamical system (Definition 2.3), and finite automata (Definition 2.6), we can define hybrid automata as follows:

Definition 2.9 (Hybrid Automata) *A hybrid automaton $\mathcal{H}$ is a tuple*

$$\mathcal{H} = (Q, X, Init, f, D, E, G, R),$$

where

- *Q is a finite set of locations or discrete states (also called modes of operation), often denoted by $q_i \in Q$;*
- *$X \subseteq \mathbb{R}^n$ is the state space in which the continuous-time variables evolve. A state is often denoted by $x \in X$;*
- *$Init \subseteq Q \times X$ is a finite set of initial states;*
- *$f : Q \times X \to X$ is the vector field representing the system's dynamics, often defined by a differential equation $\dot{x} = f(q_i, x)$;*
- *$D : Q \to X$ describes the set of continuous invariants (or domains) $Inv(q_i) \subseteq X$ of the discrete states $q_i \in Q$;*
- *$E \subseteq Q \times Q$ is a finite set of discrete transitions;*
- *$G : E \to X$ is a finite set of guard conditions;*
- *$R : E \to X \times X$ is a finite set of reset maps.*

In the context of hybrid automata, the hybrid state (q_i, x) of the system is a pair of a discrete state $q_i \in Q$ and a valuation $x \in X$ of the continuous state variables. The hybrid state space of the system is the set of all possible values for the discrete location and the continuous variables, i.e. $Q \times X$.

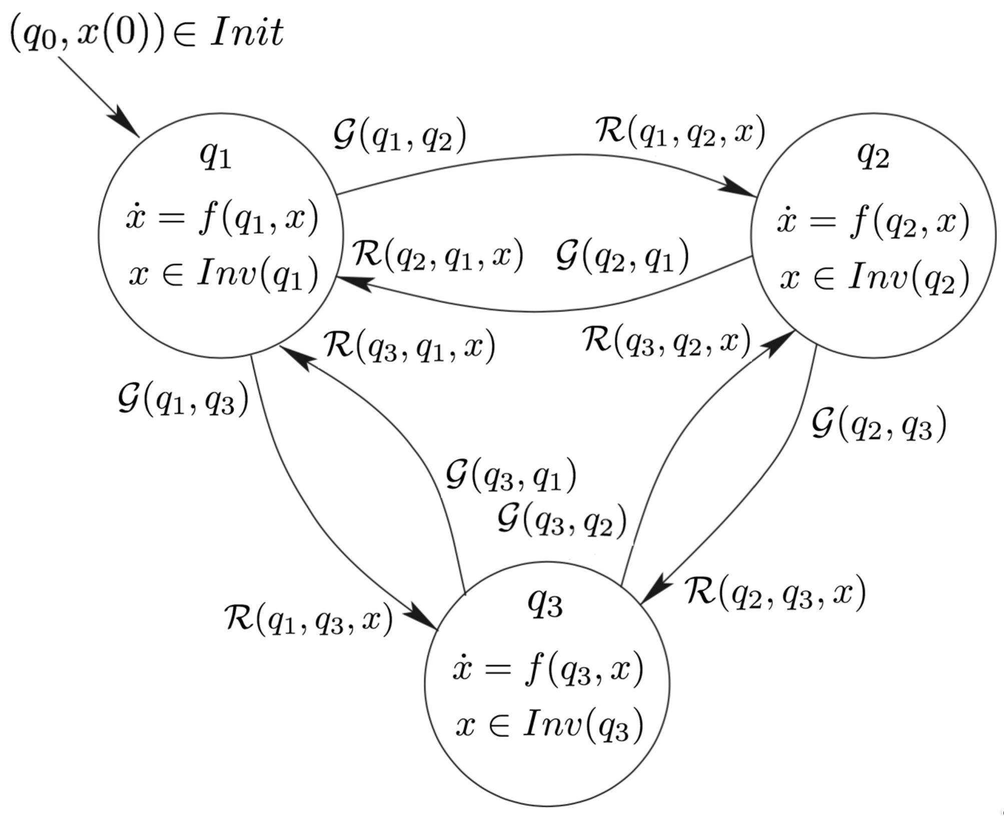


Figure 2.7: Schematic representation of a hybrid automaton with three discrete states.

The hybrid automaton model elements lead to represent the hybrid system graphically as sketched in Figure 2.7. The discrete evolution of the hybrid system's dynamics is modeled in terms of a graph in which vertices represent the discrete states, and edges represent state transitions.

Every vertex is associated with vector field $f : Q \times \mathbb{R}^n \to \mathbb{R}^n$ which determines the continuous evolution of the system's state $\dot{x} = f(q_i, x)$ if the location (or discrete state) is $q_i \in Q$.

For every discrete state $q_i \in Q$ there is an invariant $Inv(q_i) \subseteq \mathbb{R}^n$ associated to it. This invariant is typically combined with constraints that should be satisfied by the continuous state x to stay at this invariant.

Mode transitions can be triggered either by time events, or by state events due to the invariants and guards. The change of the discrete state q_i is described by the state transition function $(q_i, q_j) \in E$, which determines the discrete successor state q_j if the system is in a given discrete state q_i. This function is graphically represented by the arrows among the discrete states in Figure 2.7.

A discrete transition $(q_i, q_j) \in E$ is enabled if its guard condition $\mathcal{G}(q_i, q_j) \in G$ is true, i.e. the valuation of the continuous variables must fulfill the guard. The guard conditions $\mathcal{G}(q_i, q_j) \in G$ are basically represented by expressions of the form $\mathcal{G}(q_i, q_j) := a \bowtie b$, where $\bowtie$ is a boolean relation (i.e. $\bowtie \in \{<, \leq, >, \geq, \cdots\}$), and a and b are usually made of constants, computed variables, state variables x, as well as arithmetic operations $\diamond \in \{+, -, \times, \div\}$ of constants and/or variables. From a practical simulation perspective, it is possible to convert such guard expressions $\mathcal{G}$ into zero-crossing event functions z by converting each guard expression $\mathcal{G}(q_i, q_j) := a \bowtie b$ into a real-valued function $z(t, x) := a - b$.

During the transition, the continuous variables are reinitialized according to the reset mapping $\mathcal{R}(q_i, q_j, x) \in R$. The reset map is, in general, a set-valued function that specifies the new value of the continuous states (and its relation to the previous continuous states) for a particular transition.

Remark 2.10 Invariants and guards play complementary roles: while invariants can be seen as constraints that determine when a discrete transition must take place (namely when the evolution of the continuous state variables would violate the invariant), the guards can be seen as "triggering conditions" that determine when a particular discrete transition may be taken. Whenever both guard and invariant condition are true simultaneously, a choice between discrete and continuous evolution must be taken. A transition is therefore said to be deterministic if: i) it has a unique destination, and ii) whenever this transition is possible, continuous evolution is impossible. Determinism or non-determinism of transitions is hence a modeling issue.

2.2.4.2 Executions

In order to define formally the execution of a hybrid automaton model (i.e. the types of solutions that it accepts) we first need to consider the sets of times over which these solutions will be defined.

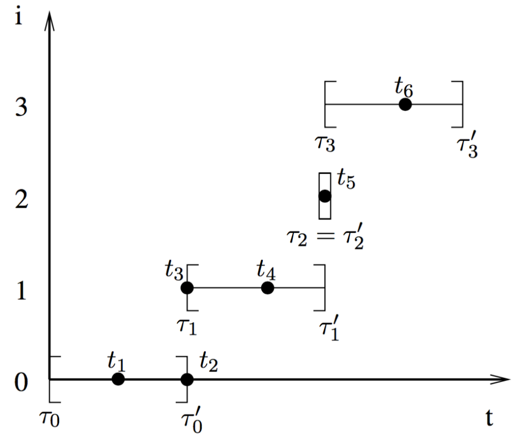


Figure 2.8: Hybrid time domain $\tau = \{[\tau_i, \tau'_i]\}_{i=0}^{3}$.

Definition 2.11 (Hybrid Time Set) *We define a hybrid time set τ as a sequence of time intervals $\tau = \{I_i\}_{i=0}^{N}$ that is finite ($N < \infty$) or infinite ($N = \infty$) such that:*

- *$I_i = [\tau_i, \tau'_i]$ for $0 \le i < N$;*
- *if $N < \infty$, then either $I_N = [\tau_N, \tau'_N]$ or $I_N = [\tau_N, \tau'_N)$ with $\tau'_N = \infty$; and*
- *$\tau_i \le \tau'_i = \tau_{i+1}$ for all i.*

The instants τ_i are times of discrete transitions, i.e. discrete events time instants. It is assumed that discrete transitions are instantaneous, therefore $\tau'_i = \tau_{i+1}$, where τ'_i corresponds to the time instant just before taking a discrete transition (i.e. right endpoint of one interval I_i), whereas τ_{i+1} corresponds to the time instant just after taking a discrete transition (i.e. left endpoint of the following interval I_{i+1}); see Figure 2.8. The benefit of adopting this convention is that it allow us to model scenarios where several discrete transitions emerges consecutively at the same time instant, in which case we have $\tau'_{i-1} = \tau_i = \tau'_i = \tau_{i+1}$ (c.f., the interval $I_2 = [\tau_2, \tau'_2]$ in Figure 2.8).

However, this convention pose a fundamental problem in case of Zeno executions because no definition of solution is possible in this case past the Zeno limit point (because of the infinity of discrete transitions). From a practical perspective this leads simulation tools developers to adopt the notion of adding hysteresis to event functions, which in turns leads to ignore events.

Definition 2.12 (Hybrid Solution Trajectory) *Consider $x \in \mathbb{R}^n$ as the state vector (Definition 2.9). We define a hybrid solution trajectory as a triple $\chi =(\tau, \{q_i\}_{i=0}^{N}, \{x_i = x(I_i)\}_{i=0}^{N})$ consisting of a time set $\tau = \{I_i\}_{i=0}^{N}$ that is hybrid, and two sets of sequences represented by $q_i \in Q$ and $x_i : I_i \to \mathbb{R}^n$.*

A typical hybrid solution trajectory is illustrated in Figure 2.9. If the hybrid time τ is an infinite sequence or if τ is a finite sequence which ends with an interval like $[\tau_N, \infty)$, then the hybrid trajectory is said to be an infinite trajectory.

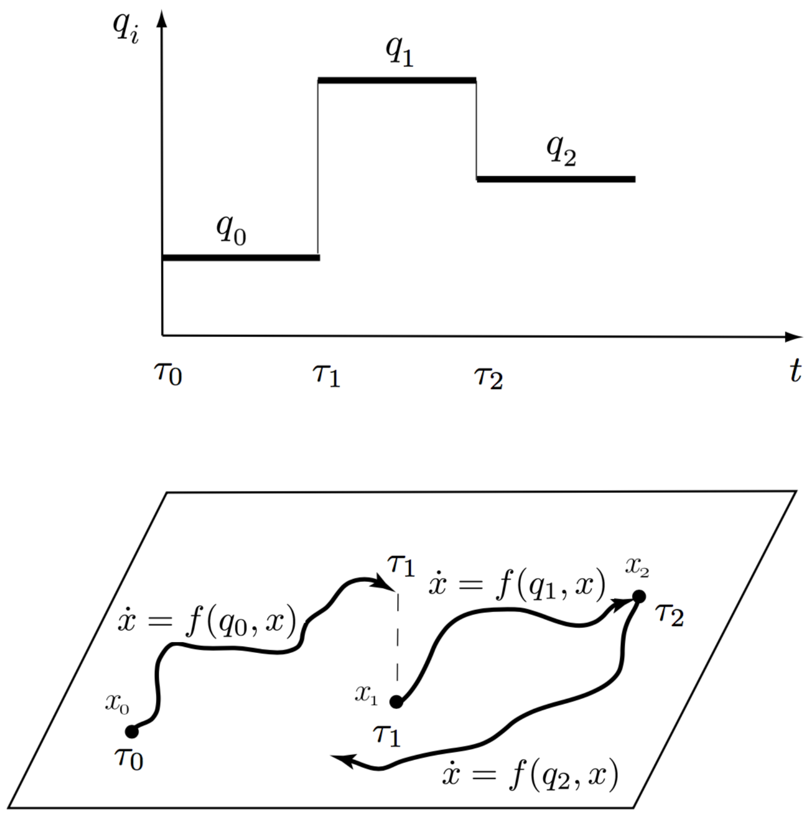


Figure 2.9: Typical hybrid solution trajectory.

Definition 2.13 (Hybrid Execution) *A hybrid solution trajectory* $\chi = (\tau, \{q_i\}_{i=0}^{N}, \{x_i\}_{i=0}^{N})$ *is said to be an execution to a hybrid automaton model* $\mathcal{H}$ *if it satisfies the following properties:*

- $(q_0, x_0) = (q_0, x_0(\tau_0)) \in Init$*, with* $\tau_0 = 0$*;*
- *for all* $i < N$*:* $e = (q_i, q_{i+1}) \in E$*,* $x_i(\tau_i') \in \mathcal{G}(e)$*,* $x_{i+1}(\tau_{i+1}) \in \mathcal{R}(e, x_i(\tau_i'))$*;*
- *for all* i *and all* $t \in [\tau_i, \tau_i')$*:* $x_i(t) \in Inv(q_i)$*,* $\dot{x}_i(t) = f(q_i, x_i(t))$*.*

Definition 2.13 determines which one of the hybrid solution trajectories is an execution to the hybrid automaton $\mathcal{H}$, and which one is not, by setting some restrictions.

The first restriction says that any execution of a hybrid automaton should start with an initial state that is included in *Init*.

The second restriction specifies when discrete transitions have to be triggered and what is the new value of the state after taking a discrete transition. The requirements relate the state $(q_i,\ x_i(\tau_i'))$ before the discrete transition to the state $(q_{i+1}, x_{i+1}(\tau_{i+1}))$ after the discrete transition: they should be such that $(e = q_i, q_{i+1})$ is a discrete transition in E, $x_i(\tau_i')$ belongs to the guard of this transition, and $x_{i+1}(\tau_{i+1})$ belongs to its reset map. It is convenient to think, in this context, of the guard conditions $\mathcal{G}(e) \in G$ to be as *enabling* conditions for the discrete transitions $e = (q_i, q_{i+1}) \in E$: a discrete transition $e \in E$ is taken at time t during the execution when $x_i(t) \in \mathcal{G}(e)$.

The third restriction specifies what happens during the continuous evolution of the system, and how the continuous evolution can result in a discrete transition. The first part says that, the discrete state keeps constant along continuous evolution. The second part constrains that the continuous state of the system evolves according to the vector field $\dot{x}_i(t) = f(q_i, x_i(t))$. The third part constrains that the state must remain in the invariant $Inv(q_i)$ of the discrete state q_i as long as the continuous evolution takes place. It is convenient to think, in this context, of invariants $Inv(q_i)$ as *forcing* discrete transitions: a transition must be taken when the continuous state is about to leave the invariant.

2.2.4.3 Examples of Hybrid Automata Models

Example 2.1 (Thermostat):
Consider for example a thermostat, which measures the temperature of a room and controls the temperature x to be always between 18°C and 22°C by turning *off* and *on* a heater. In its initial state, the heater is assumed to be *on* and the initial temperature of the room is 20°C. When the heater is in the mode *on*, the ambient temperature of the room increases according to the dynamics $\dot{x} = K(h-x)$, where K is a constant parameter representing a room constant, and h is a heater constant. During the heating phase, if the temperature is equal to or is greater than 21°C, but at latest when it is equal to 22°C, then the control is given to the heater to switch it *off*. When the heater is switched *off*, the temperature decreases according to the dynamics $\dot{x} = -Kx$. During the cooling phase, if the temperature is equal to or is lower than 19°C, but at latest if it is equal to 18°C, then the control is given back to the heater to switch it *on*. Figure 2.10 shows the hybrid automaton model of the system, and Figure 2.11 visualizes a possible execution of the system. This system is fully hybrid. The continuous behavior is the temperature's evolution in time. The discrete behavior is the heater's control switching. This hybrid system is non-deterministic because the control is given within temperature intervals (i.e. switching from *on* to *off* in the temperature interval [21°C· · 22°C], and switching from *off* to *on* in the temperature interval [18°C· · 19°C]). To get a deterministic model one may replace these temperature intervals by single values.

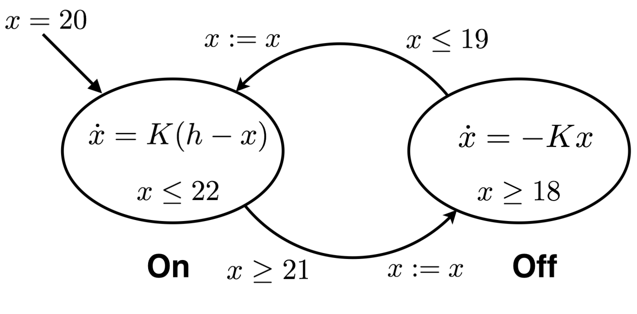


Figure 2.10: Thermostat system: the model represented by a hybrid automaton.

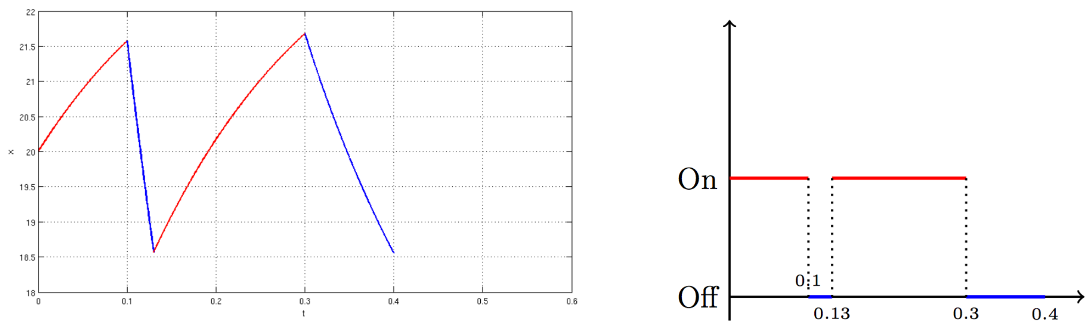


Figure 2.11: A possible execution of the hybrid automaton in Figure 2.10.

Example 2.2 (Collision of Three Masses):
Consider a mechanical system of three collision balls of masses m_1, m_2 and m_3 disposed on a surface (a table) having a height h and a length L and as sketched in Figure 2.12. It is assumed that friction between the balls and the table be neglected. Masses m_2 and m_3 are at rest, while mass m_1 is considered to be moving according to an initial velocity $v_{1,0}$. Eventually, mass m_1 collides with mass m_2 which, as results moves towards m_3 and collides with it. As a result, mass m_3 falls and starts to bounce on the ground.

The law of the conservation of momentum and Newton's collision rule govern each collision between two masses. Let consider for example the collision between m_1 and m_2. Denote v_i and v_i^+ to the velocity just before and just after the impact between the masses, respectively. Therefore, Newton's collision rule implies that $v_1^+ - v_2^+ = -\epsilon(v_1 - v_2)$, where ϵ is the coefficient of restitution.

The rule of conservation of momentum determines the velocities just after the collision: $m_1(v_1^+ - v_1) = m_2(v_2 - v_2^+)$. A collision between m_1 and m_2 happens when $x_1 \geq x_2$ and $v_1 > v_2$. In this case, the post-collisions velocities are

$$v_1^+ = v_1 \frac{(m_1 - \epsilon m_2)}{m_1 + m_2} + v_2 \frac{m_2(1+\epsilon)}{m_1 + m_2}, \tag{2.14}$$

$$v_2^+ = v_1 \frac{m_1(1+\epsilon)}{m_1 + m_2} + v_2 \frac{(m_2 - \epsilon m_1)}{m_1 + m_2}. \tag{2.15}$$

The same applies for the collision between m_2 and m_3, where we assume that $x_{2,0} < x_{3,0}$. Figure 2.13 shows the hybrid automaton model of this system with all possible scenarios of collisions between the three masses and/or their bouncing on the ground. The system's state variables are represented by the position and velocity of each mass. The labels Fi represent the guard conditions and the reset maps when mass m_i falls from the surface towards the ground. The labels Cij specify the guard conditions and also the reset maps in case in which a collision between a mass i and a mass j occurs. The labels Bi represent the guard conditions and the reset maps when a falling mass m_i bounces.

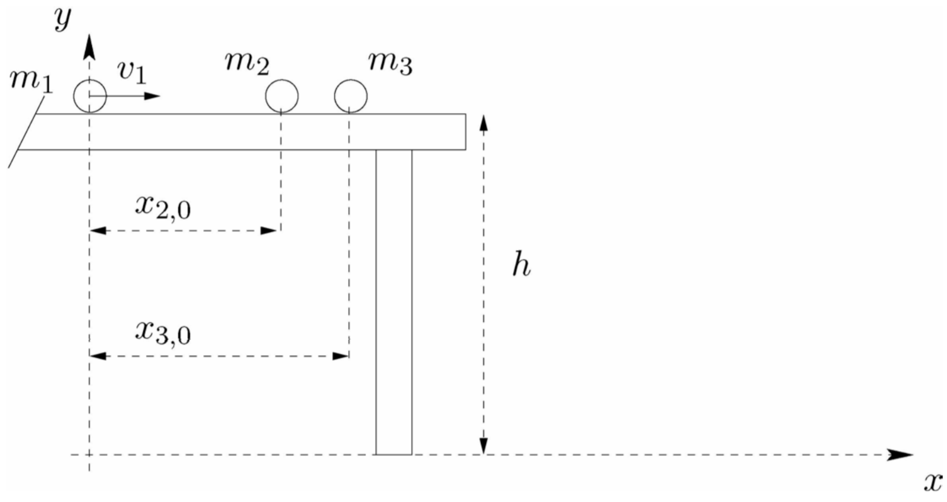


Figure 2.12: The system with three point masses.

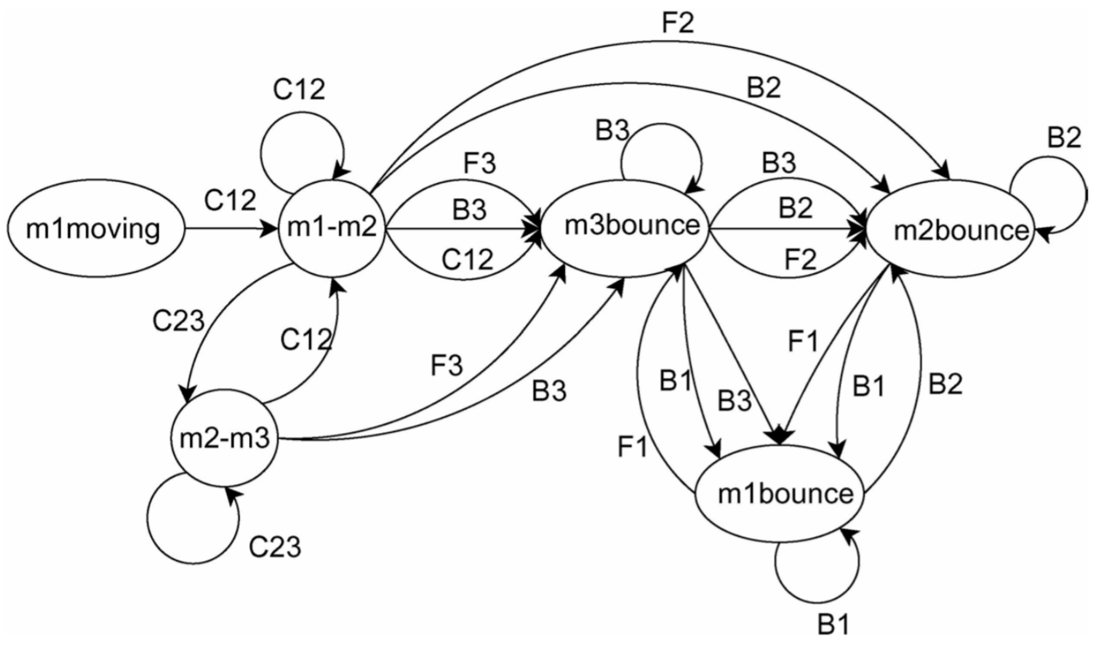


Figure 2.13: The hybrid automaton model of the three point masses system.

During bounces, there is a loss of energy on the x and y directions. This is modeled by the coefficients γ_x and γ_y, respectively. For each discrete state (or location), it is assumed that its invariant is the conjunction of the complement of guards conditions associated to the existing transitions form that discrete state.

Label	**Guard**	**Reset**
C12	$x_1 \geq x_2 \ \wedge \ vx_1 > vx_2$	$vx_1 = vx_1^+ \ \wedge \ vx_2 = vx_2^+$
C23	$x_2 \geq x_3 \ \wedge \ vx_2 > vx_3$	$vx_2 = vx_2^+ \ \wedge \ vx_3 = vx_3^+$
F1	$x_1 \geq L \ \wedge \ y_1 > 0 \ \wedge \ vx_1 > 0$	$ay_1 = -g$
F2	$x_2 \geq L \ \wedge \ y_2 > 0 \ \wedge \ vx_2 > 0$	$ay_2 = -g$
F3	$x_3 \geq L \ \wedge \ y_3 > 0 \ \wedge \ vx_3 > 0$	$ay_3 = -g$
B1	$y_1 \leq 0 \ \wedge \ vy_1 < 0$	$vx_1 = \gamma_x vx_1 \ \wedge vy_1 = -\gamma_y vy_1$
B2	$y_2 \leq 0 \ \wedge \ vy_2 < 0$	$vx_2 = \gamma_x vx_2 \ \wedge vy_2 = -\gamma_y vy_2$
B3	$y_3 \leq 0 \ \wedge \ vy_3 < 0$	$vx_3 = \gamma_x vx_3 \ \wedge vy_3 = -\gamma_y vy_3$

Table 2.1: Guard conditions and reset maps for the hybrid automaton in Figure 2.13.

Table 2.1 lists the guard conditions and the reset maps for each discrete transition in the system. For the scenario sketched in Figure 2.12, the system behavior can be described as follows: At start, all the masses are at rest on the table. We assume $y_i = h,\ a_i = 0,\ i = 1, 2, 3$ (i.e. all the masses are at rest on the table, and all accelerations are initialized to zero). Also it is assumed that that $x_{1,0} = 0$, and $x_i = x_{i,0}$ for $i = 2, 3$. Mass m_1 moves to the right initially with a given velocity $v_{1,0} > 0$ and collides with m_2 (mode *m1-m2*). After the impact, mass m_2 moves to the right and, in turn, collides with mass m_3 (mode *m2-m3*). Eventually, mass m_3 drops from the table (transition *F3*) and starts to bounce on the ground (state *m3bounce* and transitions *B3*).

To consider that the ground manifests some friction when bouncing, we consider both horizontal and vertical loss of energy during bouncing. During the bouncing of m_3 on the ground, the two masses m_1, m_2 (depending on the values of the three masses) may either fall also and start bouncing or eventually stop on the table. In all the discrete states, the system's dynamics is represented by linear differential equations. Let's denote ay, vy and ax, vx to the vertical and horizontal components of the acceleration and the velocity, respectively. Then, we can write the system dynamics as: $\dot{x}_i = vx_i$, $\dot{vx}_i = ax_i$, $\dot{y}_i = vy_i$, $\dot{vy}_i = ay_i$.

From a practical simulation point of view, this example shows interesting phenomena. For example, three discrete events can occur simultaneously (i.e. event iteration) if we position mass m_3 at the extreme edge of the table (i.e. setting $x_{3,0} = L$): the mass m_2 collides with the mass m_3 and both of them then fall and start bouncing. These events are ordered sequentially even though they are occurring simultaneously. In fact, this is the reason for having many discrete states with the same system dynamics. With only one state, the model would be a non-deterministic model, therefore not capable of properly order the events. When both masses m_2 and m_3 fall and start bouncing simultaneously, this also results in a non-determinism because the events of bounces can be ordered arbitrarily. Indeed, this system can be seen as a system with geometric-Zeno behavior because at least the mass m_3 will fall from the table and starts bouncing, and its sub-model becomes the one of a bouncing ball model which has a geometric-Zeno behavior.

2.2.5 Hybrid Automata: Limitations of Standard Semantics

Although hybrid automata formalism is a well-established formalism for specifying hybrid systems in the context of modeling and verification, it has several limitations due to its simplicity and it is sometimes insufficient for the full treatment of hybrid dynamics. Among these limitations, we focus on the problem of Zeno executions.

Although analytically distinctly different, both chattering-Zeno and geometric-Zeno behaviors share the same property: an infinite sequence of discrete transitions occurring in a finite execution time. For an execution $\chi = (\tau, q(\cdot), x(\cdot))$ satisfying the conditions in Definition 2.13, the *execution time* $\mathcal{T}(\chi)$ is the sum of the time intervals in τ, that is $\mathcal{T}(\chi) = \sum_{i=0}^{N}(\tau_i' - \tau_i)$. According to Lygeros et al [2], the execution χ is called:

- **Infinite execution** if either τ is an infinite sequence (i.e. $i \to \infty$) or $\mathcal{T}(\chi) = \infty$;
- **Finite execution** if the sequence τ is finite (i.e., $N < \infty$) and the last interval in this sequence is closed (i.e. $I_N = [\tau_N, \tau_N']$);
- **Maximal execution** if the execution is not a prefix of any other execution;
- **Zeno execution** if τ is an infinite sequence and $\mathcal{T}(\chi)$ is a finite time:
  1. It is **chattering-Zeno** if it is Zeno and there exists a finite C such that $\forall i \geq C : \tau_i' - \tau_i = 0$.
  2. It is **geometric-Zeno** if it is Zeno and $\forall i \in \mathbb{N} : \tau_i' - \tau_i > 0$.

Figure 2.14 (taken from [2]) shows the hybrid time sets for finite, infinite and Zeno executions. The limitation of the standard semantics of executions of hybrid automata is that, it is incomplete in the meaning that the limit point of discrete change, such as Zeno limiting state cannot be handled properly. From a practical simulation perspective, this poses a fundamental problem when simulating Zeno executions: In fact, many modeling and simulation tools for hybrid systems have been developed in the past years. They can be classified into two categories: those who put special attention on defining models rigorously, such as for instance SpaceEx [44], Ptolemy [27] (based on the super-dense time semantics in [47]), and Zélus [57] (whose semantics is based on non-standard analysis [54]); and those who use an informal approach for model definition such as Simulink[1], Modelica language [55] and its associated tools. The problem is that all these modeling and simulation tools share the same approach of hybrid model execution alternating between continuous evolution and sequences of discrete switchings [57] as formalized by the standard executions semantics of hybrid automata, which in turn cannot handle properly Zeno limit points. To deal with this problem we propose to replace the standard notion of hybrid time domain (Definition 2.11) by the notion of *non-standard densely ordered* hybrid time domain. The advantages of using a non-standard time domain in the executions semantics of hybrid automata are the following:

[1]https://fr.mathworks.com/products/simulink.html

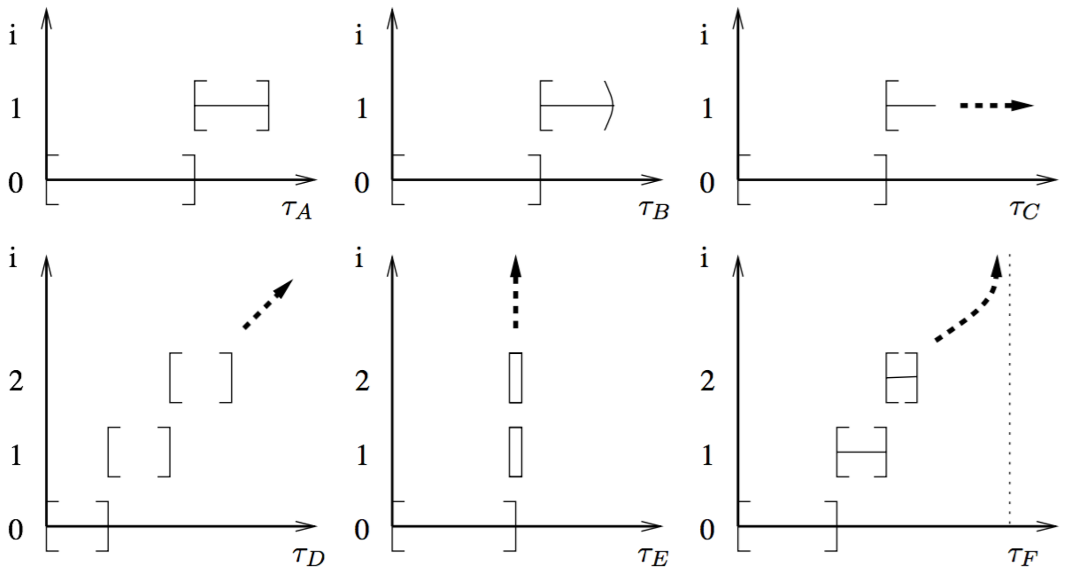


Figure 2.14: Hybrid time sets of different executions: τ_A is finite, τ_C and τ_D are infinite, τ_E and τ_F are Zeno.

1. The continuous dynamics of the hybrid system is reduced to the recurrence equation that represents the infinite iteration of infinitesimal discrete changes with infinitesimal duration. Therefore, we can handle the hybrid dynamics based only on fully discrete paradigm.
2. The representation of dynamics based on non-standard analysis is complete and the exact limit point of discrete change, like chattering-Zeno and geometric-Zeno limit points, can be handled.

2.2.6 Non-Standard Semantics for Hybrid Automata

Before introducing the non-standard model of time, and the proposed non-standard semantics of hybrid automata, we start by giving a brief introduction to the theory of non-standard reals $^*\mathbb{R}$.

2.2.6.1 The Theory of Non-Standard Reals $^*\mathbb{R}$

The field $^*\mathbb{R}$ of non-standard reals has been used by several authors to define operational semantics of continuous and hybrid systems [54, 57]. The idea is to enlarge the time domain by enlarging the standard reals numbers $\mathbb{R}$ and integers $\mathbb{N}$ into non-standard real numbers $^*\mathbb{R}$ and non-standard $^*\mathbb{N}$. The set of non-standard real number $^*\mathbb{R}$ is constructed from the set of real number $\mathbb{R}$ via the ultra product construction [108].

Notation 2.14 We denote a non-standard object with the prefix *. For example, the variable symbol $^*x$ denotes an element of $^*\mathbb{R}$. Whenever we can obviously know that the object is non-standard, we omit the prefix *.

Definition 2.15 (Free Ultra filter) *A filter $\mathcal{U}$ on a set J is a subset of $\mathcal{P}(J)$, the power set of J, satisfying the following properties:*

1. *Proper filter: $\emptyset \notin \mathcal{U}$.*

2. *Finite intersection property: If $A, B \in \mathcal{U}$, then $A \cap B \in \mathcal{U}$.*

3. *Superset property: If $A \in \mathcal{U}$ and $A \subseteq B$, then $B \in \mathcal{U}$.*

A filter $\mathcal{U}$ is said to be a free ultra filter if it also satisfies:

1. *Maximality: For all $A \subseteq J$, either $A \in \mathcal{U}$ or $J \setminus A \in \mathcal{U}$.*

2. *Freeness: $\mathcal{U}$ contains no finite subsets of J.*

Definition 2.16 (Non-Standard Real Number) *We fix a free ultra filter $\mathcal{U}$ of $\mathbb{N}$. Let $W = \mathbb{R}^{\mathbb{N}}$ denote a set of sequences of real numbers $\langle a_1, a_2, \cdots \rangle$. A non-standard real number is defined by a set of sequences $\langle a_i \rangle_{i \in \mathbb{N}}$ closed under the following equivalence relation on W:*

$$\langle a_1, a_2, \cdots \rangle \sim \langle b_1, b_2, \cdots \rangle \tag{2.16}$$

if and only if $\{k|\ a_k = b_k\} \in \mathcal{U}$. We denote the equivalence class of $\langle a_1, a_2, \cdots \rangle$ by $[\langle a_1, a_2, \cdots \rangle]$. Namely, the set of all non-standard real numbers is defined as the quotient $^*\mathbb{R} = \mathbb{R}^{\mathbb{N}} / \sim$. Following the same principle, we denote the set of non-standard numbers generated by integral sequences $\langle a_i \rangle \in \mathbb{N}$, $i \in \mathbb{N}$, by $^*\mathbb{N}$.

Every element of $\mathbb{R}$ is also an element of $^*\mathbb{R}$ because for $x \in \mathbb{R}$, we have $[\langle x, x, \cdots \rangle] \in {}^*\mathbb{R}$. Namely, $\mathbb{R} \subseteq {}^*\mathbb{R}$. By virtue of the ultra product construction of $^*\mathbb{R}$, all properties of $\mathbb{R}$ also hold in $^*\mathbb{R}$ (transfer principle [54]). From the construction of $^*\mathbb{R}$, the arithmetic operations and the relations on $\mathbb{R}$ are extended to $^*\mathbb{R}$ as follows: Addition for $^*\mathbb{R}$ is defined as $[\langle a_1, a_2, \cdots \rangle] + [\langle b_1, b_2, \cdots \rangle] = [\langle a_1 + b_1, a_2 + b_2, \cdots \rangle]$. The order on $^*\mathbb{R}$ is defined such that $[\langle a_1, a_2, \cdots \rangle] < [\langle b_1, b_2, \cdots \rangle]$ if and only if $\{n|\ a_n < b_n\} \in \mathcal{U}$. Other operations and relations are also defined by the same way. For example, consider two non-standard numbers $^*j = [\langle \frac{1}{n} \rangle] \in {}^*\mathbb{R}$ and $^*w = [\langle n \rangle] \in {}^*\mathbb{R}$. Then it holds that $^*j \times^* w = [\langle \frac{1}{n} \times n \rangle] = [\langle 1 \rangle] = 1$, $\sqrt{^*j} = [\langle \frac{1}{\sqrt{n}} \rangle]$, $^*w^2 = [\langle n^2 \rangle]$.

Definition 2.17 (Infinitesimal, Infinite) *An infinitesimal $\partial \in {}^*\mathbb{R}$ is a number whose absolute value is smaller than any non-negative standard real number. Namely, $\exists \partial \in {}^*\mathbb{R} :\ \forall a \neq 0 \in \mathbb{R} : |\partial| < |a|$. An infinite $\omega \in {}^*\mathbb{R}$ is a number which absolute value is larger than any standard real number. Namely, $\exists \omega \in {}^*\mathbb{R} :\ \forall a \in \mathbb{R} : |\omega| > |a|$.*

Definition 2.18 (Closeness) *We introduce a binary relation $a \approx b$ between two non-standard elements $a \in {}^*\mathbb{R}$ and $b \in {}^*\mathbb{R}$, meaning that a is infinitesimally close to b, and such that $a \approx b$ if and only if $|a - b|$ is infinitesimal.*

For every finite number $a \in {}^*\mathbb{R}$ (namely a is not an infinitely large number), there is a unique standard number $b \in \mathbb{R}$ such that $a \approx b$. b is called a standard part of a and denoted by $b = {}^\circ a$. For any $x = [\langle x_1, x_2, \cdots \rangle] \in {}^*\mathbb{R}$, we define a non-standard ${}^*f(x)$ for a standard function $f(x)$ by ${}^*f(x) = [\langle f(x_1), f(x_2), \cdots \rangle]$. The continuity of a non-standard function ${}^*f(x)$ is described as follows: for all $x \in {}^*\mathbb{R}$ there exists $y \in {}^*\mathbb{R}$ such that $y \approx x$ and ${}^*f(y) \approx^* f(x)$.

2.2.6.2 Non-Standard Time Domain

The proposal in [54] consists in defining the enlarged time set as

$$\mathbb{T}_\partial = \{t_n = n \times \partial |\ n \in {}^*\mathbb{N}\} \tag{2.17}$$

for a time base (which is infinitesimal) $\partial \in {}^*\mathbb{R}, \partial > 0, \partial \approx 0$. That is, the following property is satisfied:

$$\forall t \in \mathbb{R}^+ : \exists t_n \in \mathbb{T}_\partial \text{ such that } t \approx t_n \tag{2.18}$$

In the following, we give the hybrid automata's non-standard semantics, as well as a unified rigorous definition to Zeno behavior in the non-standard hybrid time domain.

2.2.6.3 Hybrid Automata: Non-Standard Semantics

The non-standard semantics of hybrid automata is based on using $\mathbb{T}_\partial$ as its time set. It suffices in this context to keep the symbolic structure of the standard hybrid automata $\mathcal{H}$ (Definition 2.9) and only change the domain of interpretation of its executions from $\mathbb{R}$ to ${}^*\mathbb{R}$. To ease future discussions, we define a function $q(t)$ as the mode selector function that returns a discrete state in Q for any time instant $t \in \mathbb{T}_\partial$.

Definition 2.19 (Non-Standard Execution of $\mathcal{H}$) *Given a time base $\partial \in {}^*\mathbb{R}$, $\partial > 0$, $\partial \approx 0$, time index set $\mathbb{T}_\partial = \{t_n = n \times \partial |\ n \in {}^*\mathbb{N}\}$. Fix $\mathbb{B} = \{True, False\}$. A non-standard execution ${}^*\chi$ of a hybrid automaton $\mathcal{H}$ is a tuple of functions $z : \mathbb{T}_\partial \to \mathbb{B}$, $q : \mathbb{T}_\partial \to Q$, and ${}^*x : \mathbb{T}_\partial \to {}^*\mathbb{R}^n$ satisfying the following properties:*

1. *$(q(t_0), {}^*x(t_0)) \in {}^*Init$ and $z(0) = False$ when $t_0 = 0$;*
2. *ODE micro-step: For all $t \in \mathbb{T}_\partial$: If $z(t+\partial) = False$ then $q(t+\partial) = q(t)$, ${}^*x(t+\partial) \in {}^*Inv(q(t))$, ${}^*x(t+\partial) = {}^*x(t) + \partial \times {}^*f(q(t), {}^*x(t))$;*
3. *Location change micro-step: For all $t \in \mathbb{T}_\partial$: If $z(t+\partial) = True$ then ${}^*x(t) \in {}^*\mathcal{G}(e), e = (q, q') \in E$, $q(t) = q$, $q(t+\partial) = q'$, and ${}^*x(t+\partial) \in {}^*\mathcal{R}(e, {}^*x(t))$.*

Remark 2.20 As mentioned in Remark 2.10 (page 29), determinism or non-determinism is a modeling issue.Whenever the invariant of the current location holds and some outgoing transitions are enabled, then it is up to the modeler to give a priority either to stay in the invariant or to exit from the invariant according to a transition.

2.3 Simulation of Hybrid Systems

Simulation methods for hybrid systems fall broadly into two categories: *time-stepping*, and *event-driven*. Each of them may be worked out in many different ways.

2.3.1 The Event-Driven Method

The method of event-driven simulation is based on generating hybrid solution trajectories according to the following steps:

1. Simulation of the smooth dynamics within a given mode (discrete state).
2. Event detection.
3. Determination of a new discrete state (new mode).
4. Determination of a new continuous state (re-initialization).

The idea is to simulate the continuous motion in some given mode with numerical solvers using time-stepping method until a discrete event is detected.

Events within hybrid dynamical systems simulation can be distinguished in what we called *externally induced events* such as time events (e.g. when switches are tuned in an electrical network according to a predetermined schedule), and *internally induced events* (or state events) which occur when an inequality constraint of a given event function becomes active (zero-crossing activation). To catch the internally induced events, a hybrid system simulator needs to be equipped with an event detection module. Such a module will monitor the sign of certain event functions of the state to see if the required inequality constraints are still satisfied.

In the combination with a time-stepping algorithm for the simulation of continuous dynamics, one issue to be taken into account is that the time at which an event takes place will in general not coincide with one of the grid points that the continuous simulator has placed on the time axis. Both the event time itself and the value of the continuous state at the time of the event will have to be found by some interpolation method. A search is then a necessary requirement to find accurately the time of the event and the corresponding state values. At this stage, ODE solvers usually have to backtrack in order to accurately approximate the date and state at which the event occurs. Since this is quite expensive and slows down simulation in some cases, some simulation tools, e.g. Simulink/Stateflow, offer the option to execute discrete events without localizing their time points precisely. However, this can easily result in unpredictable behavior.

The problem of finding the next discrete state is called the *mode selection problem*. This problem may be easy in some cases (i.e. deterministic models). However, there are other cases though in which the problem can be quite complicated so that decisions will be very sensitive (i.e. the case of non-deterministic models). In such situations the simulation software should provide a warning to the user, and if it is difficult to make a definitive choice between several possibilities perhaps the solver should even work out all reasonable options in parallel.

The next step is the restarting the integration from the new initial time and initial condition in the new selected mode, and here the problem of re-initialization comes down to determine the value of the continuous state at the event time so that the simulation of the smooth dynamics in the new mode can start from an initial state that is correct up to the specified tolerance. In some cases however, such as mechanical systems subject to unilateral constraints, jumps need to be calculated.

Theoretically, the state after jump should satisfy certain constraints exactly; finite word length effects however will cause small deviations due to the machine precision. Such deviation may cause an interaction with the mode selection module; in particular it may appear that a certain constraint is violated so that a new event is detected. In this way it may happen that cycling between different modes occurs (i.e. chattering back and forth between modes in a discrete fashion), and the simulator does not return to a situation in which the continuous evolution according to some continuous dynamics is generated, so that effectively the simulation stops.

2.3.2 The Time-Stepping Method

In a number of papers (see for instance [26,34–36,38,40,42,45,50,51,61,82] it has been suggested that in fact it is not necessary to track events in order to obtain approximate trajectories of hybrid dynamical systems. The term "time-stepping methods" has been used to refer to methods that do not aim to determine event times. The basic idea of time-stepping is to only check constraints and corresponding slack variables (Lagrange multipliers) at fixed times at intervals Δt. There are adaptations to standard methods for integrating differential equations that are specifically designed for complementarity systems, some of which are based on linear complementarity problem solvers that have been developed in optimization theory [36,42,45].

Rather than giving a formal discussion of time-stepping methods let's illustrate the idea on an example. Consider for instance a relay system given by

$$\dot{x} = \begin{cases} \dot{x}_1 = & -\mathrm{sgn}(x_1) + 2\mathrm{sgn}(x_2), \\ \dot{x}_2 = & -2\mathrm{sgn}(x_1) - \mathrm{sgn}(x_2), \end{cases} \tag{2.19}$$

where the signum function (or relay element) sgn is actually not a function but a relation (or multi-valued function) specified by

$$\mathrm{sgn}(x) = \begin{cases} -1, & \text{when } x < 0, \\ +1, & \text{when } x > 0, \\ [-1, +1], & \text{when } x = 0. \end{cases} \tag{2.20}$$

This system may be described as a piecewise constant system; in each quadrant of the (x_1, x_2)-plane the right hand side is a constant vector. As demonstrated in Figure 2.15, the solutions of this system are spiraling towards the origin, which is an equilibrium point. It can easily be proved that solutions that start with initial conditions $(x_1(0), x_2(0))$ cannot stay away from the origin (0,0) for a time longer than $\frac{1}{2}(|x_1(0)| + |x_2(0)|)$. However,

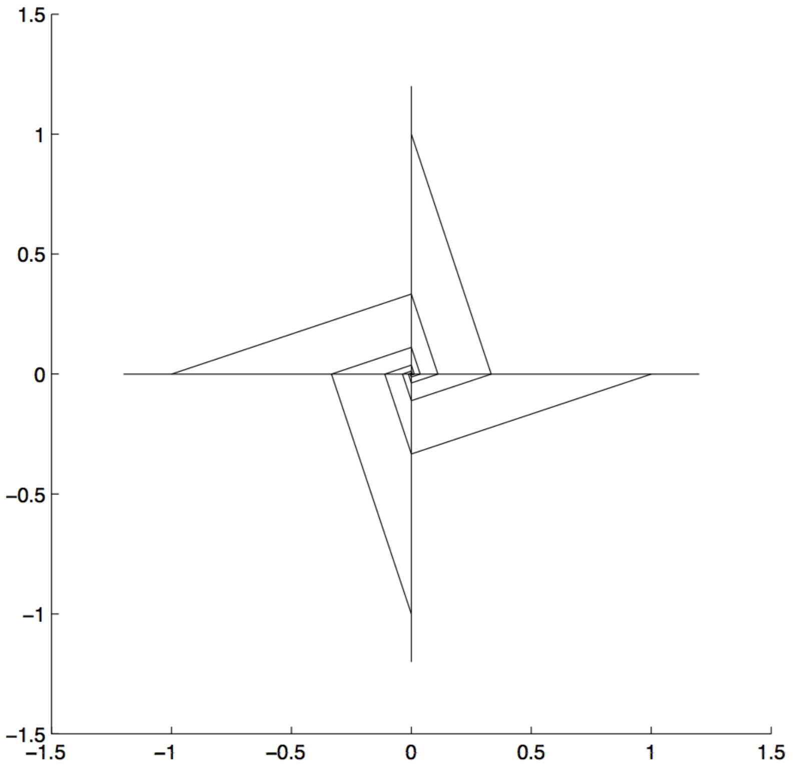


Figure 2.15: Solutions of the relay system in (2.19).

the solutions cannot reach the origin (0,0) without passing by an infinity of discrete modes switchings; and as these discrete mode switchings would have to happen in finite time, there must be an accumulation of events. This is a typical example of a hybrid dynamical system with Zeno behavior.

Clearly, an event-driven method is in principle not able to carry out simulation across the Zeno accumulation point. The simplest fixed-step discretization scheme for (2.19) is the forward Euler scheme

$$\frac{x_{1,k+1} - x_{1,k}}{h} = -\mathrm{sgn}(x_{1,k}) + 2\mathrm{sgn}(x_{2,k}), \tag{2.21}$$

$$\frac{x_{2,k+1} - x_{2,k}}{h} = -2\mathrm{sgn}(x_{1,k}) - \mathrm{sgn}(x_{2,k}), \tag{2.22}$$

where h denotes the size of the time step and the variable $x_{i,k}|_{i=1,2}$ is intended to an approximation of $x_i(t)$ for $t = kh$. With the interpretation (2.20) of the signum function, the system (2.19) is non-deterministic. An alternative is to use an implicit scheme. Even though it is essential to use implicit scheme in such case, it should be noted however that implicit method requires an extra computation and is harder to implement in general. In addition, choosing large step sizes of time can result in a solution that is inaccurate.

Therefore, it is strongly required to verify the results by reducing the time step size until the solution of the system does not change anymore. Back to our example, the simplest choice of such a scheme is the following:

$$\frac{x_{1,k+1} - x_{1,k}}{h} = -\mathrm{sgn}(x_{1,k+1}) + 2\mathrm{sgn}(x_{2,k+1}), \tag{2.23}$$

$$\frac{x_{2,k+1} - x_{2,k}}{h} = -2\mathrm{sgn}(x_{1,k+1}) - \mathrm{sgn}(x_{2,k+1}). \tag{2.24}$$

At each step, $x_{1,k}$ and $x_{2,k}$ are given and the equations (2.23) and (2.24) are to be solved for $x_{1,k+1}$ and $x_{2,k+1}$. Equations (2.23) and (2.24) may be written as a system of equalities and inequalities by introducing some extra variables. Simplifying notation a bit by writing simply x_i instead of $x_{i,k+1}$ and $x_i^\natural$ instead of $x_{i,k}$ we obtain the following set of equations and inequalities:

$$x_1 = x_1^\natural - hu_1 + 2hu_2, \tag{2.25}$$

$$x_2 = x_2^\natural - 2hu_1 - hu_2, \tag{2.26}$$

$$\text{where for } i = 1,2: \ u_i = \begin{cases} -1, & \text{when } x_i < 0, \\ +1, & \text{when } x_i > 0, \\ [-1, +1], & \text{when } x_i = 0. \end{cases} \tag{2.27}$$

This system is to be solved in the unknowns x_1, x_2, u_1, and u_2 for arbitrary given $x_1^\natural$ and $x_2^\natural$; h is a parameter. It can be verified directly that for each positive value of h and for each given $(x_1^\natural, x_2^\natural)$ the above system has a unique solution; alternatively, one may recognize the system in (2.25), (2.26), and (2.27) as an instance of the Linear Complementarity Problem and infer the same result of general facts about the LCP. Figure 2.16 shows the partitioning of the $(x_1^\natural, x_2^\natural)$ plane that corresponds to the nine possible ways in which the disjunctions in (2.27) can be satisfied. For instance, the solution $x_1 = 0$ and $x_2 = 0$ is obtained from the values of $(x_1^\natural, x_2^\natural)$ such that the solution of the equation

$$\begin{bmatrix} x_1^\natural \\ x_2^\natural \end{bmatrix} + h \begin{bmatrix} -1 & 2 \\ -2 & -1 \end{bmatrix} \begin{bmatrix} u_1 \\ u_2 \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \end{bmatrix} \tag{2.28}$$

satisfies $|u_1| < 1$ and $|u_2| < 1$. A simple matrix inversion shows this happens when

$$-5h < x_1^\natural + 2x_2^\natural < 5h, \qquad -5h < -2x_1^\natural + x_2^\natural < 5h, \tag{2.29}$$

which corresponds to the central area in Figure 2.16. For this discretized system, the solution behaves like that of the original system except in the narrow strips which do not influence the solution very much, and except in the central area where the solution jumps to zero whereas the continuous system goes through mode changes at a higher and higher pace. When the step size h shrinks to zero, then the solution of the discretized system converges to the solution of the original system, including the continuation of

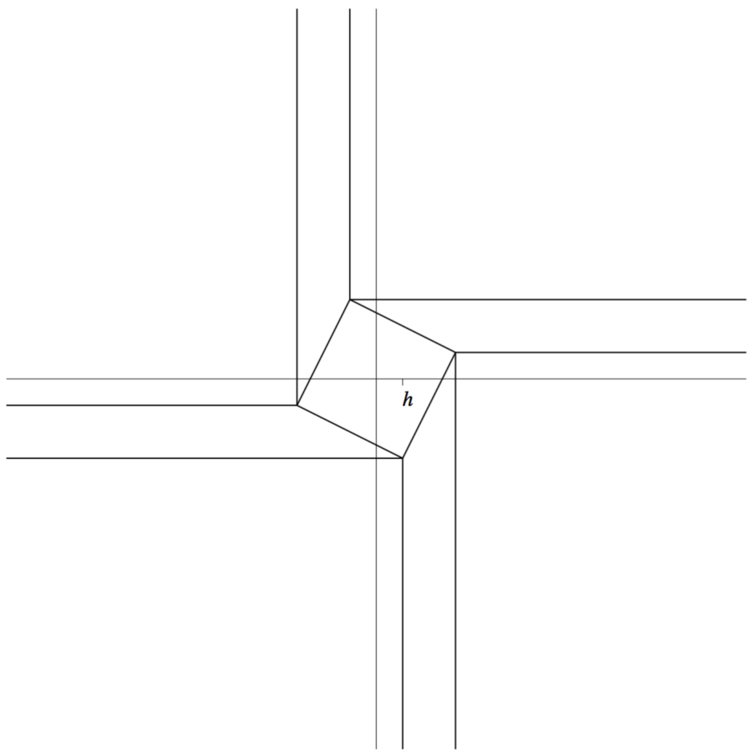


Figure 2.16: Partitioning of the plane induced by (2.23) and (2.24).

this solution by $x(t) = 0$ beyond the accumulation of event times. Note that the explicit scheme (2.21) and (2.22) shows a rather different and less satisfactory behavior.

We can argue from the discussion of this example that at least in some cases and by using suitably selected discretization schemes it is possible to get an accurate approximation of the trajectories of hybrid systems without capturing discrete events. However, errors are introduced by not accurately detecting the transition times, and therefore time-stepping schemes are often of low-order accuracy (i.e. with error estimates that $\sim O(\Delta t)^q$ for a low q) and can completely miss events associated with low-velocity collisions. Several commercially available implementations of time-stepping algorithms are available, especially for the specific case of rigid body mechanics; see the review by Brogliato and co-workers [58] and the Chapter 2 by Abadie in [56] for more details. In addition, there are many questions to be asked when using a time-stepping method such as: i) under what conditions it is possible to use a time-stepping method, ii) which discretization methods are most suitable in general cases of hybrid systems models with non-trivial dynamics, and iii) what the consequences of using a fixed time step rather than a variable time step, keeping in mind that the latter provides a much more accurate results of the numerical simulation and that the fixed step can easily lead to unpredictable behavior.

2.4 Survey of Hybrid Simulation Tools

The most widely used modeling languages for hybrid dynamical systems are those provided by numerical simulation tools. They offer features like modularity, hierarchy and a data-flow or equational syntax. In the following we review briefly some languages and tools for hybrid simulators.

2.4.1 Simulink/Stateflow

Simulink[2] is an integrated tool platform for modeling, simulation, code-generation and test automation of hybrid dynamical systems. It has a graphical data-flow-based input language for specifying the continuous and discrete behavior. Consider for example the hybrid automaton model of a bouncing ball system as shown in Figure 2.17, where x_1 denotes the height and x_2 denotes the velocity of the ball, respectively. In Figure 2.18, a Simulink block diagram model of this automaton is shown.

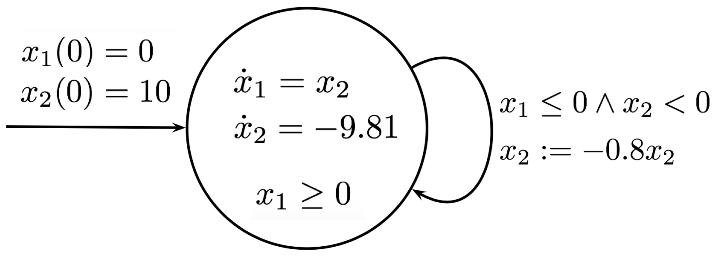


Figure 2.17: Bouncing ball: The hybrid automaton model.

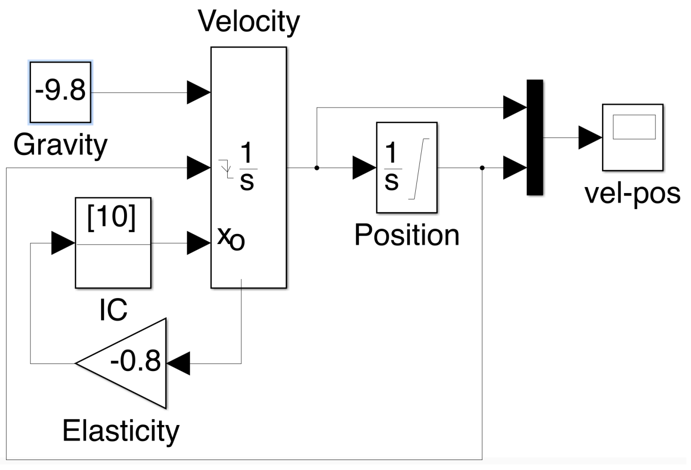


Figure 2.18: Bouncing ball model: Simulink diagram.

[2]https://fr.mathworks.com/products/simulink.html

The simulation engine in Simulink handles the model design's components by using a semantic domain that is continuous-time domain as a unique domain whenever both discrete-time and continuous-time components are included in the model. The simulation engine has many integration algorithms, which are called solvers. They are based on the ODE (ordinary differential equation) suite of Matlab. The ODE solver sophisticatedly uses an adaptive algorithm of variable time-step size. The time-step is usually selected and tuned adaptively during the simulation. In this algorithm, truncation errors are allowed when estimating the correct size of time-step, and a backtracking is done whenever the error exceeds a given threshold usually defined by the user as a parameter at the beginning of the simulation. The simulation algorithm in Simulink is conservative in terms of allowing all signals of the system to be evaluated at the time-step specified by the algorithm of integration if there is no event occurred at these time-steps. To improve the efficiency of the simulation, there are many multi-rate integration algorithms that have been developed for the ODEs. However, these proposed algorithms have an overhead that can make them slower than the conservative original algorithm.

Figure 2.19 shows the simulation of the bouncing ball model as parametrized in Figure 2.18, i.e. with $x_1(0) = 0$, $x_2(0) = 10$, and the elasticity coefficient being 0.8. With non-adaptive algorithm of zero-crossing, the simulation gets stuck and terminates with a simulation error; see Figure 2.20. This is because the simulation exceeds the default limit of 1000 for the number of consecutive zero crossings allowed in Simulink. With adaptive algorithm of zero-crossing, the simulation does not get stuck, but a significant oscillation around the switching surface results after the Zeno point.

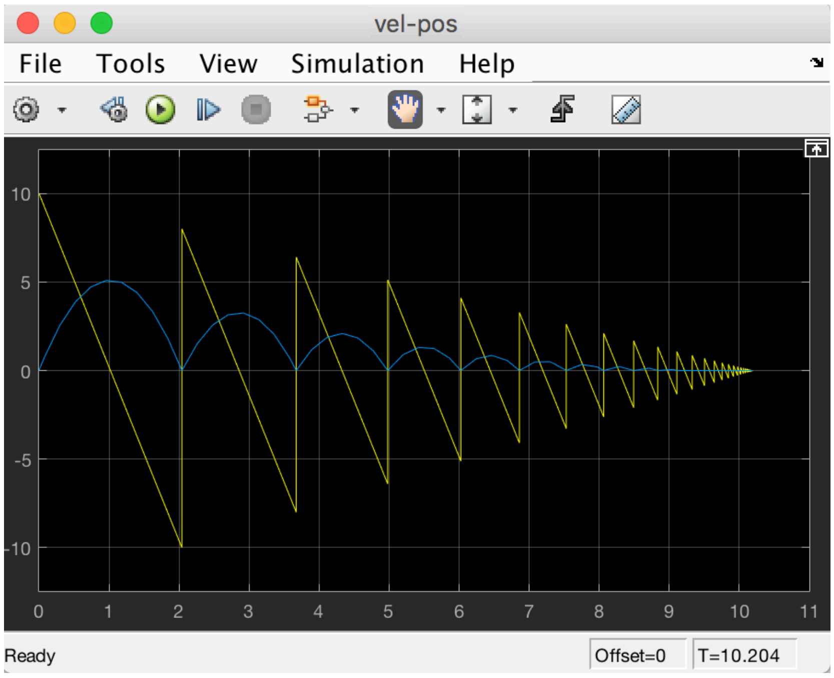
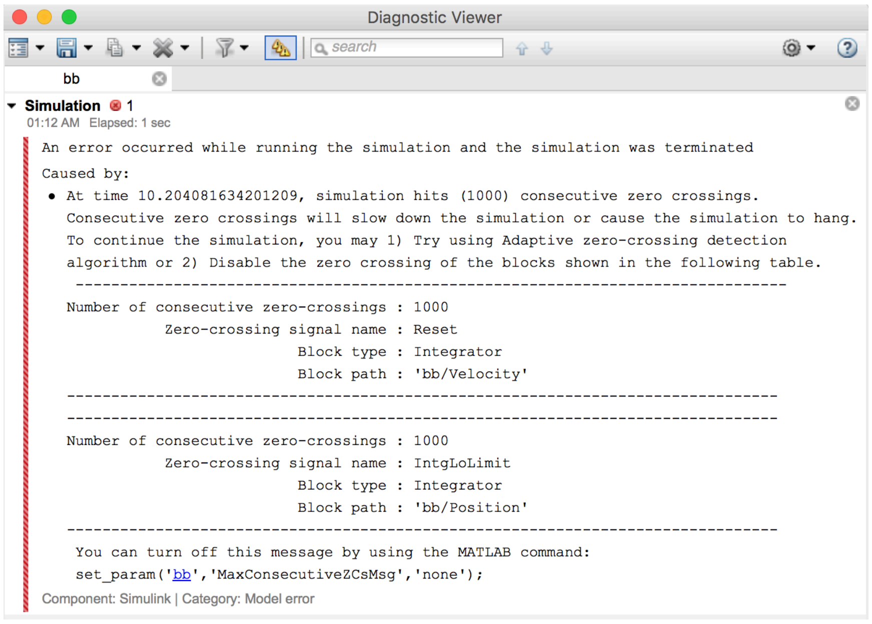


Figure 2.19: Bouncing ball simulation in Simulink: Simulation halt.



```
An error occurred while running the simulation and the simulation was terminated
Caused by:
 • At time 10.204081634201209, simulation hits (1000) consecutive zero crossings.
   Consecutive zero crossings will slow down the simulation or cause the simulation to hang.
   To continue the simulation, you may 1) Try using Adaptive zero-crossing detection
   algorithm or 2) Disable the zero crossing of the blocks shown in the following table.
    --------------------------------------------------------------------------------
   Number of consecutive zero-crossings : 1000
              Zero-crossing signal name : Reset
                             Block type : Integrator
                             Block path : 'bb/Velocity'
   --------------------------------------------------------------------------------
   --------------------------------------------------------------------------------
   Number of consecutive zero-crossings : 1000
              Zero-crossing signal name : IntgLoLimit
                             Block type : Integrator
                             Block path : 'bb/Position'
   --------------------------------------------------------------------------------
    You can turn off this message by using the MATLAB command:
    set_param('bb','MaxConsecutiveZCsMsg','none');
Component: Simulink | Category: Model error
```

Figure 2.20: Bouncing ball simulation in Simulink: Error report.

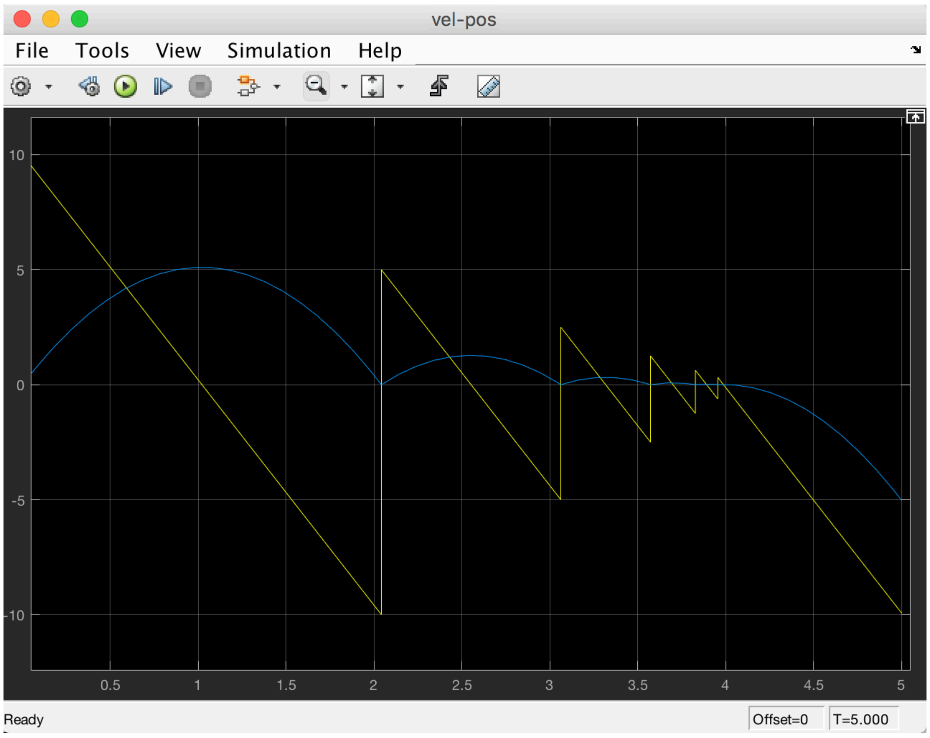


Figure 2.21: Bouncing ball simulation in Simulink: Faulty simulation results.

Now if we set the elasticity coefficient to be 0.5 instead of 0.8, the simulation does not terminate with a halt, but generates faulty results: The ball falls through the surface on which it is bouncing and then it goes into a free-fall in the space below; see Figure 2.21. This artifact is due to the fact that discrete events are missed during simulation because of the accumulation of an infinite number of discrete transitions. In this case, events which should be detected by the level crossing detector are missed. The level crossing detector works with the solver in attempt to identify the precise point in time when event occurs. However, when numbers become sufficiently small — just before the Zeno limit point — they are dominated by numerical errors, and these errors cause discrete events to be missed. Because the sign of the vertical position and of the vertical velocity remain negative between two integration steps, a new bouncing event is not generated, and the ball falls below the level of the bouncing surface.

Simulink also provides the option of using integration methods with fixed time-step. Even though this can considerably simplify the simulator's control part, it may result however in many problems. For example, if the model is stiff, i.e. a model with substantially different time constants, in this case the integration algorithm, for stability reasons, must use a time step specified by the fastest mode [21]. This can obviously yield an inefficient simulation performance when the fast modes are not allowed and the system's behavior is only determined by the slower modes. Furthermore, the time constants need to be known in order to select appropriately the time step. Finally, the inability to control the time step size may result in inaccurate simulation due to inaccurate estimation of the time at which the discrete events occur, or even worth due to missing the discrete events altogether.

Stateflow[3] is an interactive tool for development and design for complex dynamical systems, supervisory logic and control systems. Stateflow provides a visual modeling and simulation environment for complex systems by using simultaneously the concept of finite state machine, flow diagram, and Statecharts [22]. It is possible to integrate a Stateflow model as a sub-model in a Simulink model. Simulink and Stateflow interact with each other at the data and events boundaries. The simulation of an entire system's model consisting of both Stateflow and Simulink sub-models is realized by using the method of co-simulation, that is, the control of the execution to the two simulation engines (of the two tools) is released alternatively. As there is a change of control alternatively between the two simulation engines of Stateflow and Simulink, there may be an overhead that may even be significant when events are exchanged in a high frequency rate. One alternative simulation technique would be using a unified simulation engine. However, this may require an overhaul of the two tools and their semantic models.

Overall, by its symbolic simulation tools, Simulink/Stateflow provides a toolset powerful enough for complex systems modeling and design. However, there is often a need to subject Simulink models to a more rigorous and complex domain-specific analysis. In addition, one of the main drawbacks in Simulink/Stateflow is the lack of their formal semantics. Recently, there are many research attempts carried out in the the purpose of formalizing the semantics and providing a support for translation from and to

[3]https://fr.mathworks.com/products/stateflow.html

Simulink/Stateflow, specially the design of automatic semantic translators that can be interfaced to Simulink/Stateflow and translate their models into models of other modeling and simulation tools. In [23], Caspi et al. discuss an approach for translating the discrete-time part of models in Simulink into Lustre programs. The method proposed in [23] consists of hierarchical bottom-up translation, clock inference, and type inference. The implementation of this method has resulted in a prototype tool called S2L.

2.4.2 Modelica

Modelica[4] is an object-oriented modeling and simulation language for physical hierarchical modeling [24,25,29,32,55] targeting efficient simulation. The advantages of using object orientation in modeling and simulation is that it helps writing reusable models. With this method it is possible to define a set of equations common in different dynamical systems and then, depending on the real application, specialize a model. A great feature in Modelica is the non-causality of modeling. In such modeling's paradigm, it is required from the modelers to specify directly the relationship between inputs and outputs of the system in terms of a function, but rather they can define this relationship in terms of variables and the equations they should satisfy. Modelica provides a formal type system, where many primitive types are included like `Real`, `Boolean`, `Integer`, and `String`. Similarly as Java and C++, there is also a possibility for building more complicated types by defining classes. Some of the supported types of classes: blocks, packages, connectors, records, types, and models. It is also possible in Modelica to specify models that are causal by defining functions which are in Modelica considered as a special class and can have algorithm section as well as inputs and outputs. Loops and control statements are also provided in Modelica. There exist two loop statements, `for` and `while`, and two control statements, `if` and `when`.

However, hybrid systems modeling in Modelica is not that trivial. For example, both reset maps and guard constraints can be defined in algorithm sections or equation sections and they have very different meanings in these sections. When they are written in algorithm sections, simultaneous events may be missed. A non-expert user of Modelica may attempt to employ an `if` statement instead of a `when` statement. In this case the meaning of the model may be completely different. The `if` statement has a "stateless" expression meaning that it is tested without taking into account its previous value. For the `when` statement, its expression evaluates to true when its value switches from false to true. Indeed, by using the keyword `edge`, it is possible to transform a `when` into an `if` statement, where one simply can transform a statement `when expression then` into `if edge(expression) then`. An `edge(expression)` is true if the expression is true in the current time step and was false in the previous time step.

Furthermore, with the non-causal modeling in Modelica, a discrete state at a given time instant is typically not explicit but it is determined by a sequence of events that occurred until that time instant. Also, discrete events and continuous states are defined by a set of non-causal equations. Such two peculiarities make debugging in Modelica less

[4]https://www.modelica.org/

intuitive than other modeling and simulation tools like for instance HyVisual in which models are causal.

There exist many industrial commercial environments for Modelica, such as Dymola[5] (Dynamic Modeling Laboratory), and MathModelica[6], which is a Modelica -based simulation environment embedded into Mathematica. Threre exist also OpenModelica[7], which is an open-source Modelica-based modeling and simulation environment developed by the Open Source Modelica Consortium (OSMC) and intended for industrial and academic usage.

Figure 2.22 shows the simulation of the bouncing ball example in OpenModelica. With any data set and simulation scenario the result is the same: The ball falls through the surface on which it is bouncing and then it goes into a free-fall in the space below. This artifact of generating faulty simulation results instead of terminating the simulation with a halt is that Modelica tools always introduce a delay when executing a discrete transition, so for model executions having an accumulation of an infinite number of discrete transitions — as it is the case of geometric-Zeno models —, the simulation continues because in this case it is Zeno-free by construction, but error-prone. The faulty simulation results are due to numerical errors, making the error control phase — performed by Modelica simulation tools for every discrete state transition — to be no more valid because of the infinity of the discrete transitions at the Zeno limit point. For the vertical position of the ball being negative, the model has more than one solution: one that reverses the ball speed and another that continues decreasing x_1.

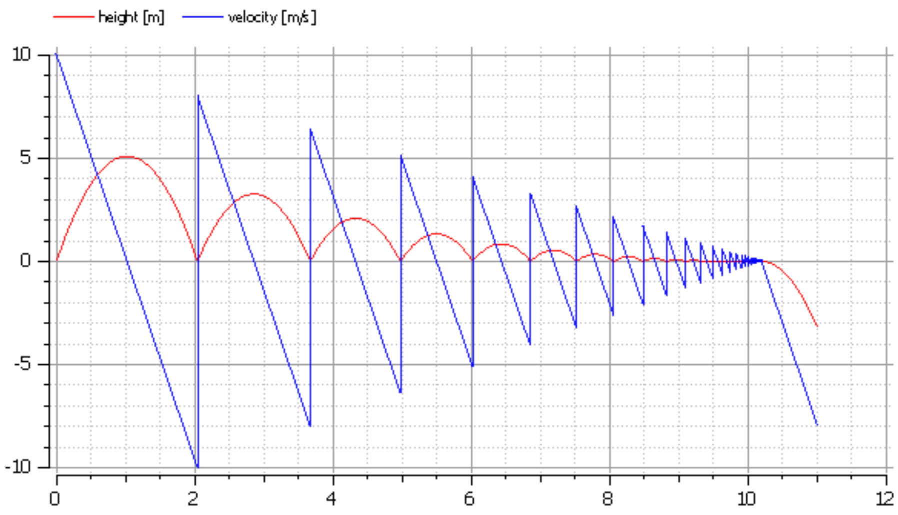


Figure 2.22: Simulation of the bouncing ball model in OpenModelica: The time evolution of the height x_1 and velocity x_2 of the bouncing ball.

[5]http://www.modelon.com/products/dymola/
[6]http://www.mathcore.com/products/mathmodelica/
[7]https://www.openmodelica.org/

OpenModelica Code of the Bouncing Ball Model:

```
model bouncing-ball
type Height = Real(unit = "m");
type Velocity = Real(unit = "m/s");
parameter Real lambda = 0.8;
parameter Height x10 = 0.0;
parameter Velocity x20 = 10.0;
Height x1; Velocity x2;
initial equation
x1 = x10; x2 = x20;
equation
der(x1) = x2;
der(x2) = -9.81;
when x1 <= 0 then reinit(x2, -lambda*pre(x2));
end when;
end bouncing-ball;
```

2.4.3 HyVisual

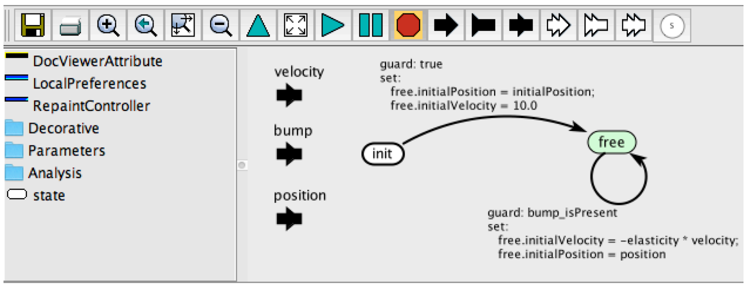
HyVisual is a simulator and block-diagram editor for hybrid systems and continuous-time dynamical systems [33]. It is developed by the Ptolemy project [39, 41], which is a framework supporting the design of domain-specific tools. The continuous time behavior of the systems, which is defined by ordinary differential equations (ODEs), is usually represented in HyVisual as block-diagrams, and the discrete behavior of the system, which is defined by finite state machines, is represented by bubble-and-arc diagrams.

HyVisual has a solid operational semantics and, in opposite to Simulink/Stateflow and Modelica that completes the definition of their operational semantics by relying in a particular simulator, HyVisual formally dos not assume any particular solver when defining the traces that result from the model's execution.

However, compared to Modelica, we can only define causal models in HyVisual, models that are based on a graphical syntax that is sometimes difficult to manipulate. That is, when a model is highly complex, this can result in an increasing of the connections between the model blocks quadratically with the blocks, a scenario which makes sometimes the model diagram quite difficult to edit and to manipulate.

Figure 2.23 shows the bouncing ball model in HyVisual. It is a modal model with two states: *init* and *free*. During the time a modal model in a state, its behavior is specified by the mode refinement. In this case, only the *free* state has a refinement; see Figure 2.24. The *init* state is the initial state, which is used only for its outgoing transition, and has set actions to initialize the ball model.

Figure 2.25 shows the simulation of the HyVisual bouncing ball model. Similarly to the case shown in Figure 2.21 and Figure 2.22, the ball falls through the surface on which it is bouncing and then it goes into a free-fall in the space below. This is because the Level Crossing actor block has missed some events just before reaching the Zeno limit point, because of numerical errors.



guard: true
set:
  free.initialPosition = initialPosition;
  free.initialVelocity = 10.0

guard: bump_isPresent
set:
  free.initialVelocity = -elasticity * velocity;
  free.initialPosition = position

Figure 2.23: HyVisual bouncing ball model

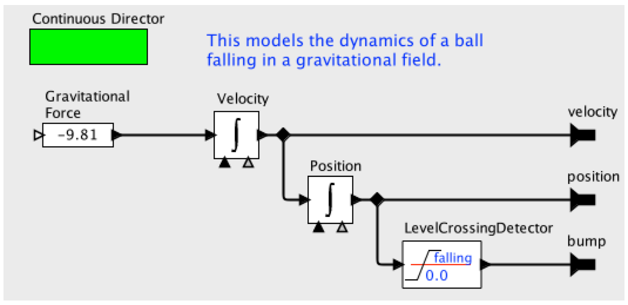


Figure 2.24: HyVisual bouncing ball model: model of the ball's dynamics during falling.

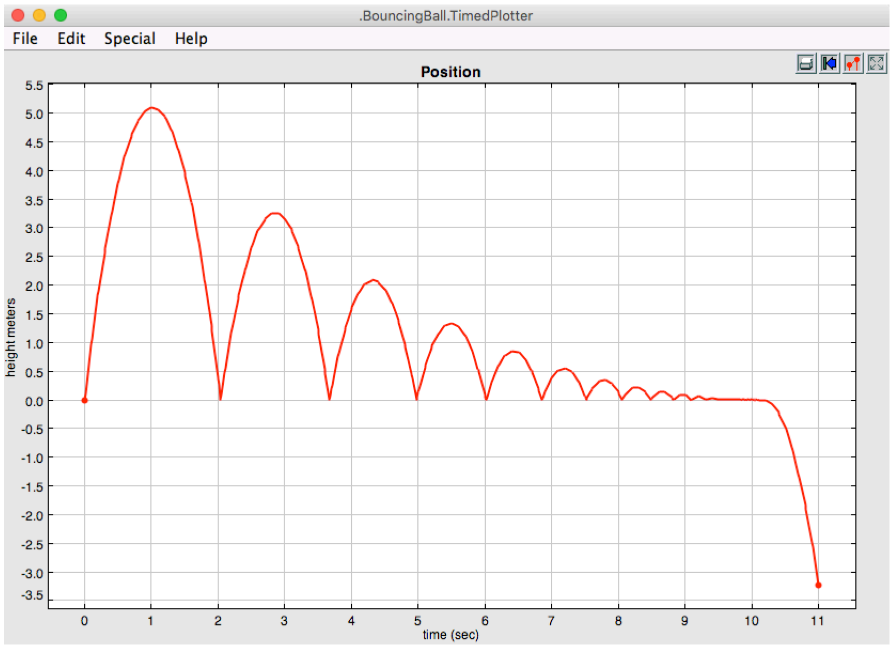


Figure 2.25: Simulation of the bouncing ball model in HyVisual.

2.4.4 Scicos

Scicos[8] is a Scilab package with a graphical environment for modeling and simulation of hybrid systems [46, 48]. Scilab[9] (Scientific Laboratory) is a numerical computations software package with a powerful computing environment for scientific and engineering applications [52]. In Scicos, models can be constructed by composing functional blocks that are predefined in the tool library. Scicos share with HyVisual the same property of dealing only with causal models. The main application of using Scicos in modeling and simulation is in embedded control: continuous-time blocks to model the physical continuous behavior of the system, and other discrete blocks to model the controller's functionality. In addition. users in Scicos can generate executable C codes. Typically the user generates an executable C code on the target hardware after simulating and refining the model and the controller's design if necessary. However, one of the drawbacks of Scicos is that modifying an invariant condition or removing or adding a state to the model could require a major change in the net-list of the model.

Consider again the bouncing ball model in Figure 2.17. The computational function of a Scicos block realizing this model is the following:

```c
#include <scicos/scicos_block4.h>
void Bounceball(scicos_block *block,int flag)
{
double *y = GetOutPortPtrs(block,1);
double *x = GetState(block);
double *h = &x[0];
double *v = &x[1];
double *xd = GetDerState(block);
double *hd = &xd[0];
double *vd = &xd[1];
double *alpha = GetRparPtrs(block);
if (flag==1){
*y = *h;
}else if (flag==0){
*hd = *v;
*vd = -9.8-*alpha*(*v)*(*v)*(*v);
}else if ((flag==2) & (GetNevIn(block)==-1)){
int *zcd=GetJrootPtrs(block);
if (*zcd<0){
*v = -(*v);
}
}else if (flag==9){
double *g=GetGPtrs(block);
*g = *h;
}
}
```

The function `jroot` is used in order to switch the sign of the ball's velocity only when the ball crosses the zero level going downward. To use this function, first it should be

[8]http://www.scicos.org/
[9]http://www.scilab.org/fr

linked with Scilab. This can be done with the followingc Scilab commands (assuming that the corresponding file is in the current directory):

```
ilib_for_link('Bounceball','bounceball.o',[],'c')
exec loader.sce
```

The first command compiles the program and creates a shared library. The second command links the shared library with Scilab. Figure 2.26 shows the diagram of the bouncing ball model in Scicos including the block resulted from compilation. The name of the simulation function is set to Bounceball, the function type to 4, the initial continuous-time state to [0;10]. Figure 2.27 shows the simulation of the bouncing ball model in Scicos. Scicos also gives faulty simulation results because of numerical errors. In such a case, numerical errors make the test on the velocity's sign by the function `jroot` to fail, and therefore the ball bounces off the zero line and goes downward thereafter.

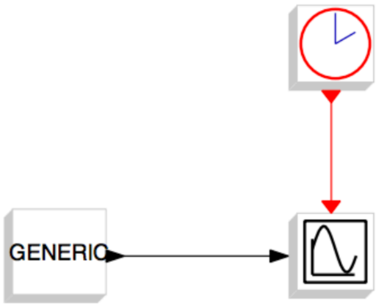


Figure 2.26: Bouncing ball model: Scicos diagram.

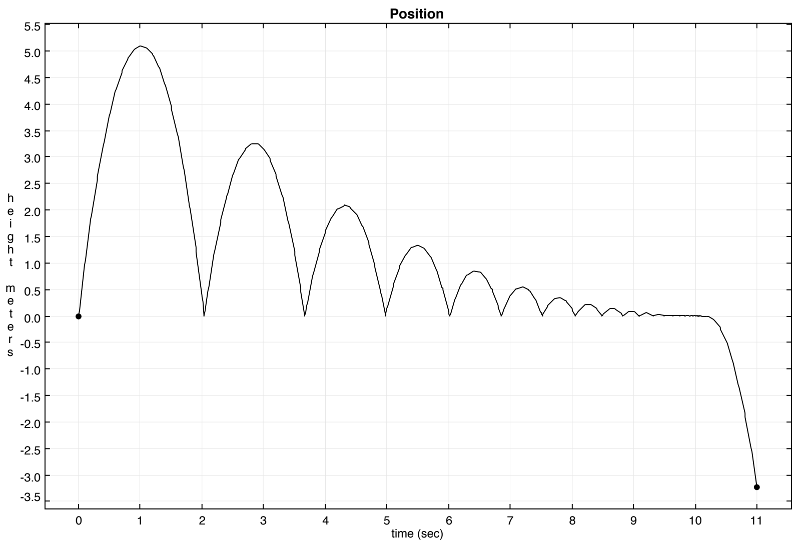


Figure 2.27: Simulation of the bouncing ball model in Scicos.

2.4.5 Acumen

Acumen[10] is an experimental modeling and simulation environment for cyber-physical systems and hybrid systems. It is built around a small, textual modeling language, and was developed as an event-driven paradigm extension that adopts a similar flavor to synchronous languages. That is, Acumen adds systems of functions or equations on dense time for describing the purely cyber controllers and the physical environment associated to them in synchronous formalisms. The standard mode for using the Acumen environment is through the GUI, which makes it possible to: i) browse files in a given directory, ii) load, edit, and save the text of a model, iii) run models, iv) view a plot, a table, or a 3D visualization of variables over time, and v) read error messages reported to the system. There are five types of statements in Acumen, namely: continuous assignments, conditional (or guarded) statements, discrete assignments, iteration, and sequences of statements. The semantics in Acumen is specified in terms of a series of translations from a large source language to smaller subsets of the language. The core language of Acumen is considered as the minimal subset that is needed to express all the features of the full source language. For more details about the formal and operational semantics of the Acumen language, the reader is referred to [67, 77, 83, 87, 101].

Let’s consider again the famous bouncing ball model. Such a model can be simulated in Acumen with the following code:

```
model Main(simulator) =
initially
x = 0 , x’ = 10, x” = -9.81 , mode = "free-fall",
always
match mode with [
"free-fall" -> x”=-9.8,
if x <= 0 && x’ < 0 then x’+ = -0.8*x’,
mode + = "free-fall" noelse ],
simulator.endTime+ = 15,
simulator.minTimeStep+ = 0.5*30
```

Figure 2.28 and Figure 2.29 show the simulation of the bouncing ball model in Acumen. As we can see, events are lost just before reaching the Zeno point, when a large number of discrete transitions start to take place. This is because Acumen uses a fixed-step integration scheme during simulation, and therefore it is Zeno-free by construction (simulation does not get stuck), but it gives faulty results in this case, as events are naturally missed in the neighborhood of the Zeno limit point. Adaptive time stepping algorithm with event localization has been implemented in Acumen very recently[11], where the function simulator.minTimeStep+ can be used in this case for the adaptive time stepping. However, adaptive time stepping algorithm also gives faulty results similar to the results generated with fixed time stepping.

[10]http://www.acumen-language.org/

[11]https://bitbucket.org/effective/acumen-dev/overview

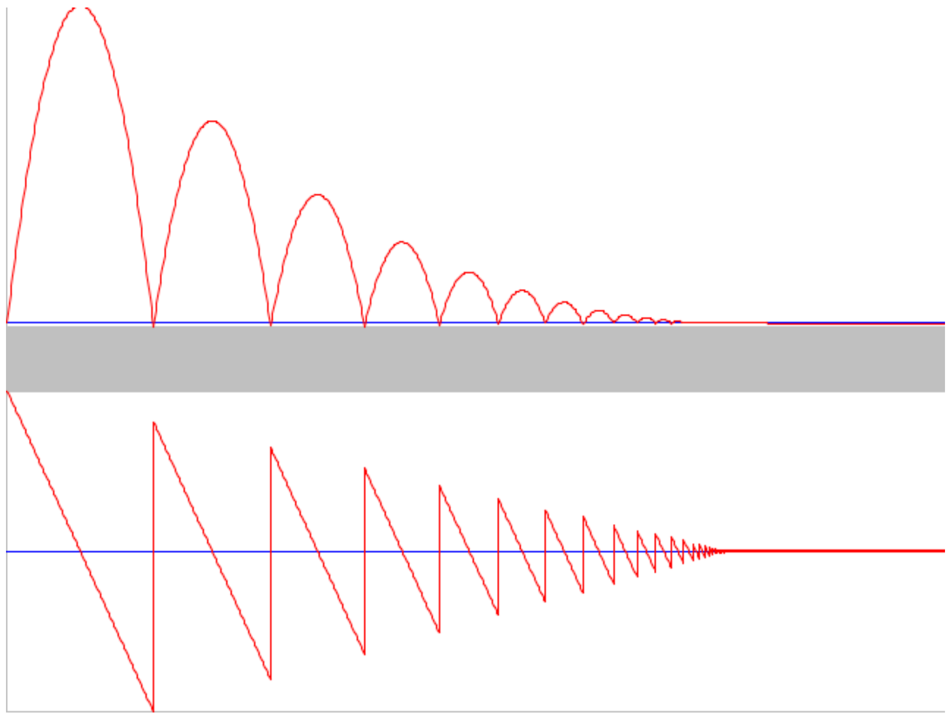


Figure 2.28: Simulation of the bouncing ball model in Acumen.

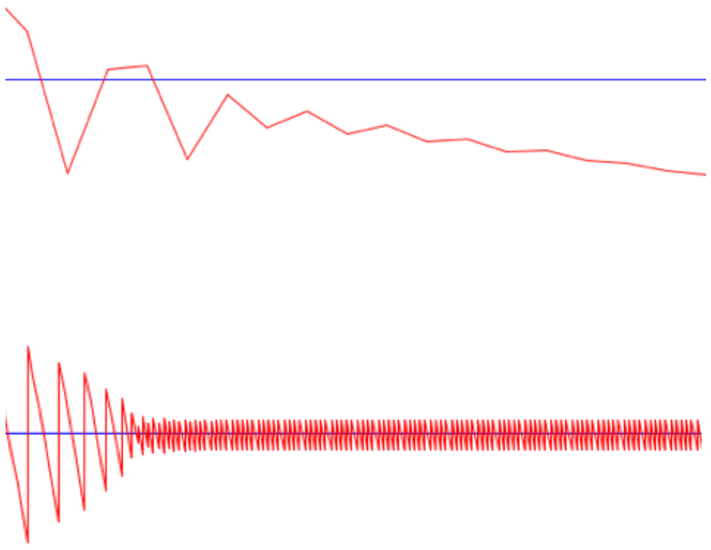


Figure 2.29: Simulation of the bouncing ball model in Acumen: Zoom on the neighborhood of the Zeno limit point.

2.4.6 Zélus

Zélus[12] is a data-flow synchronous programming language that is extended with hierarchical hybrid automata resettable first-order ordinary differential equations. It was mainly developed in the purpose of modeling and simulation of hybrid systems. The originality of Zélus comes from extending a Lustre-like synchronous language[13] with ordinary differential equations [84–86]. This extension is conservative, meaning that a synchronous program coded as hierarchical automata and data-flow equations can be arbitrarily composed with ordinary differential equations in the same source code. Zélus provides a combinations of continuous, discrete, and combinational elements. Concerning discontinuities, they must occur on discrete clocks; any program that does not respect this role is assumed to be rejected by the compiler. There is also static causality analysis and type checking system in Zélus to ensure that there are no time leaks, that is, no discontinuities occur during the continuous integration of the system since all discrete transitions are assumed to be aligned with zero-crossing events. More details about the formal rules and principles that underly the type system in Zélus can be found in [88,89].

Programs in Zélus are scheduled and translated statically into a sequential code which runs in bounded time. Compilation of programs is done by source-to-source translation to a small synchronous subset processed by a standard synchronous compiler architecture. An off-the-shelf numerical solver is combined with the resulted code. Once the source program is compiled and transformed into an executable program, a choice can be taken to choose the zero-crossing detection algorithm as well as the numerical solver, and also to set — from the command-line — their parameters. Zélus also provides a modular framework which is based on OCaml first-class modules and functors to integrate solvers. There are many features that are also available in Zélus such as the possibility to use several numerical solvers, interfacing to the Sundials cvode solver [90], as well as the possibility of using the "Illinois" false position technique for detecting zero-crossing by using standard techniques [91] (Hermite interpolation, error estimation, and Butcher tables).

Comparing with Simulink, the Zélus language is most distinguished by its type system that regulates the compositions of the continuous discrete elements of the model, and also the compilation technique based on source-to-source transformation. Furthermore, in Simulink modelers who aim to compile their Simulink models into controllers are advised to avoid some features; as for example to use function call triggers in order to determine explicitly the order of execution of blocks. In contrast, Zélus has the benefit of having a simple and consistent semantics, and the code generated from source-to-source translations can be served for both embedded targets and simulation. In case of using the code for embedded targets, there should be however a customization for specific targets. The method of executing models in Zélus is based on alternating between sequences of "run-to-completion" discrete actions and continuous phases, similar to SpaceEx[14], and Charon. The difference in Zélus is that it uses a synchronous paradigm which enforces a

[12] http://zelus.di.ens.fr/
[13] http://www-verimag.imag.fr/The-Lustre-Programming-Language.html?lang=fr
[14] http://spaceex.imag.fr/

strong discipline on causality — since the language ensures a single value per variable per instant —, and also on communication through shared variables, which are considered in Zélus as clocked streams.

The bouncing ball model in Figure 2.17 can be written in Zélus by the following code:

```
(** Bouncing ball. *)

(* [ground x] returns the position in [y] *)
let ground x = Flatworld.ground(x)
let ground_abs x = Flatworld.ground_abs(x)
let x_0 = 0.0
let y_0 = 10.0
let g = 9.81
let loose = 0.8
(* The bouncing ball *)
let hybrid ball(x, y_0) = (y, y_v, z) where
rec
der y = y_v init y_0
and
der y_v = -. g init 0.0 reset z -> (-. loose *. last y_v)
and z = up(ground(x) -. y)
(* Main entry point *)
let hybrid main () =
let (y, _, z) = ball(x_0, y_0) in
present (period (0.04)) | z -> Showball.show (x_0, y fby y, x_0, y);
()
```

Figure 2.30 shows the simulation of the bouncing ball model in Zélus. The ball falls through the bouncing surface due to limited floating-point precision and numerical errors occurring when the ball's trajectory is too close to the Zeno limit point.

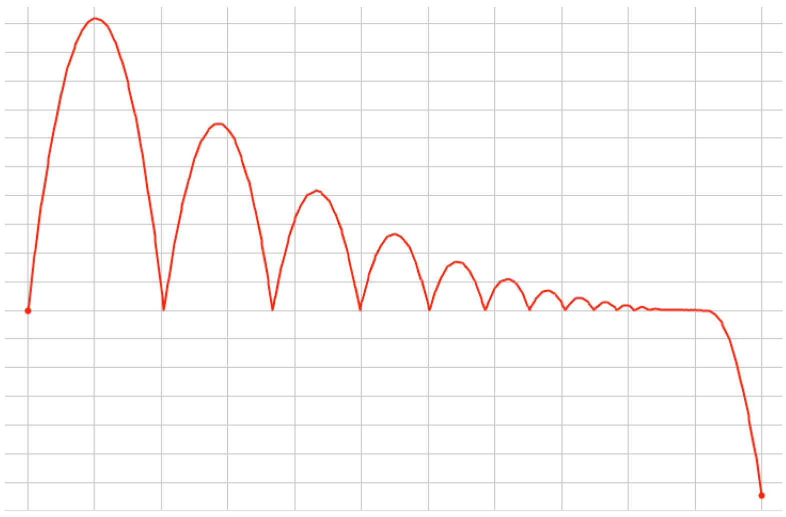


Figure 2.30: Simulation of the bouncing ball model in Zélus.

Chapter 3

Chattering-Zeno Detection and Avoidance

In this chapter, we investigate chattering-Zeno behavior of hybrid models in detail. We start this chapter by giving a brief introduction to chattering-Zeno executions. To illustrate the problem of simulating chattering-Zeno executions, we provide next a set of examples and realistic case studies of chattering-Zeno models (Section 3.1.1), and then we discuss the challenges when simulating their executions (Section 3.1.2). Afterwards, we present in details Fillipov's equivalent dynamics approach as a method proposed in the literature to deal with chattering-Zeno, and we discuss the limitation of this approach in treating chattering-Zeno (Section 3.2). Next, we introduce our computational framework of chattering-Zeno freeness (Section 3.3), including treating chattering-Zeno when it occurs on discontinuous surfaces intersections. Finally, we sketch two prototype implementations for applying the chattering-free computational framework to hybrid systems simulation tools (Section 3.4). Simulation results as well as a performance analysis of our proposed chattering-free technique and implementations are also presented.

3.1 Chattering-Zeno in Hybrid Systems

An execution of a hybrid system is called chattering-Zeno if it undergoes, in a finite amount of time, an infinite repeated switching between different control actions or modes of operation with opposed zero-crossings.

The main characteristic of chattering-Zeno behavior is that it causes the execution to infinitely move back and forth between modes in a discrete fashion with infinitesimal time spent between the repeated mode switchings.

Physically, chattering-Zeno occurs when nearly infinitesimally equal thresholds for transition conditions of different modes are given and the system starts to oscillate around them. Numerical errors may also lead to chattering-Zeno because infinitesimally equal thresholds for transition conditions may be satisfied due to local errors.

In the following, we give examples of hybrid models exhibiting chattering-Zeno.

3.1.1 Examples of Chattering-Zeno Models

Example 3.1 (Discontinuous ODE)
Consider the following differential equation with discontinuous right hand side

$$\dot{x} = \begin{cases} u+1 & \text{for } x \leq 0, \\ u-1 & \text{for } x \geq 0, \quad \text{with } \ |u| < 1. \end{cases} \tag{3.1}$$

For an initial condition $x(0) \neq 0$ we can obtain a solution of the initial value problem

$$x = \begin{cases} (u+1)t + a & \text{for } x \leq 0, \\ (u-1)t + b & \text{for } x \geq 0, \end{cases} \tag{3.2}$$

with constants a and b being determined by the initial condition. Adopting the "crossing" semantics of zero-crossings, i.e. $\mathtt{up(}z\mathtt{):=}\ z(t_{i-1}) \leq 0$ and $z(t_i) > 0$, with the zero-crossing function z being the state variable x, leads to a hybrid automaton model of the discontinuous ODE (3.1) as sketched Figure 3.1.

Notation 3.1 Denote $\overline{A}$ as the closure of an open subset A, and $\mathcal{S}_i$ as the invariant $Inv(q_i)$ of the discrete state $q_i \in Q$.

In Figure 3.1 we have $Inv(q_1) = \{x \in \mathbb{R} : x \leq 0\}$, $Inv(q_2) = \{x \in \mathbb{R} : x \geq 0\}$, $\mathcal{G}(q_1, q_2) = \{x \in \mathbb{R} : x > 0\}$, and $\mathcal{G}(q_2, q_1) = \{x \in \mathbb{R} : x < 0\}$. Clearly, the ODE (3.1) is discontinuous on a hyper surface $\Sigma = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x \in \mathbb{R} : x = 0\}$, and the vector fields in the invariants $\mathcal{S}_1 = \{x \in \mathbb{R} : x \leq 0\}$ and $\mathcal{S}_2 = \{x \in \mathbb{R} : x(t) \geq 0\}$ — on the both sides of the surface Σ — "oppose" each other. Each solution initialized outside the hyper discontinuous surface Σ can reach it in finite time. If the solution arrives at Σ it can not leave it, that is, the gradient of continuous-time behavior in each one of two disjoint invariants is directed towards their common hyper switching surface.

However, the chattering-execution of this hybrid automaton model has no standard semantics. By applying the non-standard semantics to this automaton, its non-standard behavior can be described as follows: Consider a non-standard discretization

$$x_{n+1} = x_n + \partial(u_n + y_n) \tag{3.3}$$

of the system in a non-standard time domain $\mathbb{T}_\partial$ (as defined in Section 2.2.6.2). In this case, the evolution of the control variable y_n is given by

$$y_{n+1} = \begin{cases} +1 & \text{if } y_n = +1 \text{ and } x_n \leq 0, \\ -1 & \text{if } y_n = -1 \text{ and } x_n \geq 0, \\ +1 & \text{if } y_n = -1 \text{ and } x_n < 0, \\ -1 & \text{if } y_n = +1 \text{ and } x_n > 0. \end{cases} \tag{3.4}$$

For all non-standards n, m such that $n\partial$ and $m\partial$ are not infinite it holds that $(n - m)\partial \approx 0$ implies $u_n - u_m \approx 0$.

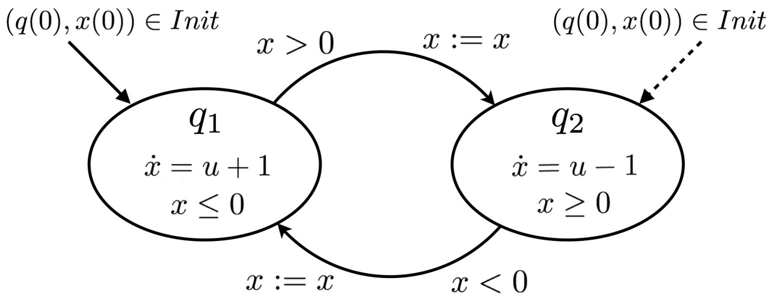


Figure 3.1: The hybrid automaton representing the discontinuous ODE (3.1).

It follows from (3.3) and (3.4) that: $x_{n+2} < x_n$ for all $x_n > 0$; $x_{n+2} > x_n$ for all $x_n < 0$; $x_{n+1} \approx 0$ for all $x_n \approx 0$; and $x_m \approx 0$ for all $m \geq n$ and $x_n \approx 0$.

We interpret a non-standard execution of hybrid automaton in Figure 3.1 as following: When in either of the two adjacent invariants $\mathcal{S}_i = Inv(q_i)|_{i=1,2}$ on their common hyper switching surface Σ an infinitesimal step ∂ causes a mode change. In any of the two adjacent invariants, the gradient directs behavior to the other invariant, and after an infinitesimal step in the current invariant a change to the other invariant occurs. The solution starts then to exhibit on Σ a chattering-Zeno back and forth between the two disjoint invariants $\mathcal{S}_1$ and $\mathcal{S}_2$.

Note that the hybrid automaton in Figure 3.1 behaves differently in the standard semantics, for the simple reason that the guard of the exiting transition does not intersect the invariant of the mode, and as a result, the time steps stop as soon as x becomes equal to 0. This also applies to several examples mentioned in this thesis. All examples presented in this thesis must be understood in the framework of the non-standard semantics.

Example 3.2 (Stick-Slip System of Two Blocks Connected Vertically)
Consider a mechanical system consisting of two blocks of masses m and M as sketched in Figure 3.2, where only the block of mass m is connected to a fixed support by a linear spring of stiffness k, and is under the action of a sinusoidal external force u generated by an actuator P. We denote x_m and x_M to the position of the small mass m and the inertial mass M, respectively, and $\mathcal{F}$ to the tangential contact force on the frictional interface between them. The friction between the inertial mass M and the ground is neglected. The origin of the displacements x_m and x_M is taken where the spring is unstretched. The external force u is modeled as a sine wave. The system's state space representation is given by

$$\dot{x} = f(x) = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m - \mathcal{F}), \\ \dot{x}_M = & v_M, \\ \dot{v}_M = & \frac{1}{M}\mathcal{F}, \end{cases} \tag{3.5}$$

with $x = [x_m \; v_m \; x_M \; v_M]^T$, where v_m and v_M are the velocities of the mass m and the mass M, respectively.

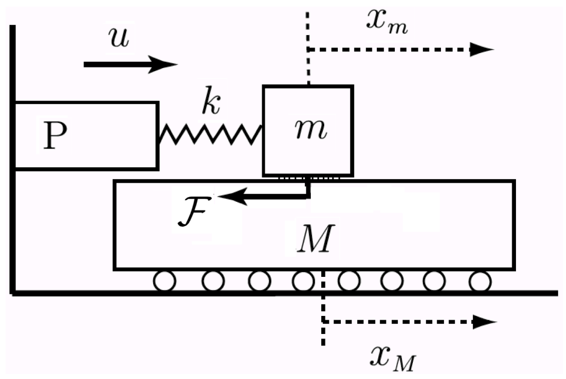


Figure 3.2: Schematic of the Stick-Slip system in Example 3.2.

The friction force $\mathcal{F}$ is modeled phenomenologically, as a function of the relative velocity $v_r = v_m - v_M$ between the two blocks

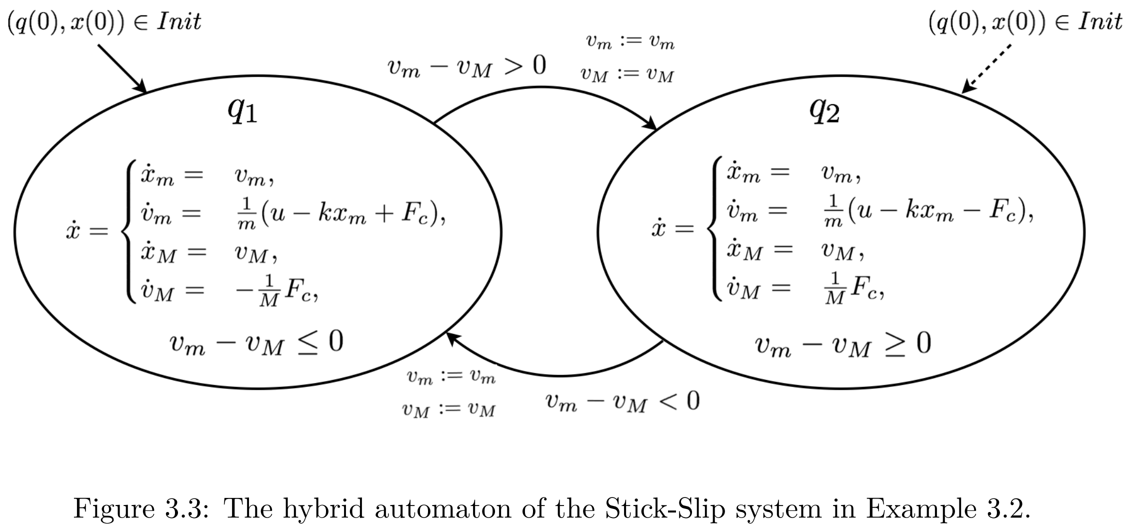
$$\mathcal{F} = \begin{cases} -F_c, & \text{for } v_r \leq 0, \\ +F_c, & \text{for } v_r \geq 0, \end{cases} \tag{3.6}$$

where F_c is the level of the Coulomb friction.

Figure 3.3 shows the hybrid automaton of the system. Similarly to Example 3.1, the system dynamics $f(x)$ is discontinuous on a hyper surface

$$\Sigma = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x \in \mathbb{R}^4 : v_m - v_M = 0\}$$

which separates the phase space into two disjoint invariants $\mathcal{S}_1 = Inv(q_1) = \{x \in \mathbb{R}^4 : v_m - v_M \leq 0\}$ and $\mathcal{S}_2 = Inv(q_2) = \{x \in \mathbb{R}^4 : \ v_m - v_M \geq 0\}$.



$(q(0), x(0)) \in Init$
$(q(0), x(0)) \in Init$
$v_m - v_M > 0$
$v_m := v_m$
$v_M := v_M$
q_1
q_2
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m + F_c), \\ \dot{x}_M = & v_M, \\ \dot{v}_M = & -\frac{1}{M}F_c, \end{cases}$
$v_m - v_M \leq 0$
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m - F_c), \\ \dot{x}_M = & v_M, \\ \dot{v}_M = & \frac{1}{M}F_c, \end{cases}$
$v_m - v_M \geq 0$
$v_m := v_m$
$v_M := v_M$
$v_m - v_M < 0$

Figure 3.3: The hybrid automaton of the Stick-Slip system in Example 3.2.

In the physical system, the two masses stick and move together with $v_r = v_m - v_M = 0$ when the force acting on the frictional interface between the two masses does not exceed the level of Coulomb friction F_c.

However, in the hybrid model of the system, mimicking such scenario, the situation becomes much more complicated because when $\frac{M}{m+M}(u - kx_m) < F_c$ we get $\dot{v}_r > 0$ when $v_r < 0$ and $\dot{v}_r < 0$ when $v_r > 0$. Roughly speaking, we have $\dot{v}_m - \dot{v}_M = \frac{1}{m}(u - kx_m) + \frac{m+M}{m \cdot M} F_c > 0$ when $v_m - v_M < 0$ and $\dot{v}_m - \dot{v}_M = \frac{1}{m}(u - kx_m) - \frac{m+M}{m \cdot M} F_c < 0$ when $v_m - v_M > 0$, so that the dynamics in the invariants $\mathcal{S}_1$ and $\mathcal{S}_2$ on both sides of Σ are pointing towards Σ.

As a result, the solution trajectory starts to perform — on the hyper switching surface Σ — an infinite number of repeated switchings back and forth between the two invariants $\mathcal{S}_1$ and $\mathcal{S}_2$, such that in either of the two disjoint invariants $\mathcal{S}_i = Inv(q_i)|_{i=1,2}$ an infinitesimal step ∂ causes a mode change.

Both Example 3.1 and Example 3.2 represent the simplest case of chattering-Zeno, where the system chatters between two different dynamics onto a single hyper switching manifold.

In general, a system's vector field could have several discontinuous switching surfaces, and chattering-Zeno may occur on the intersection of finitely many switching manifolds. Naturally, this may arise in systems having multiple discontinuous control variables.

The following examples demonstrate this special case of chattering-Zeno behavior on discontinuous surfaces intersections.

Example 3.3 (ODE with Double Discontinuities)
Consider a discontinuous (x_1, x_2)-plane system with the dynamics

$$\dot{x} = \begin{cases} \dot{x}_1 = \begin{cases} 1 & \text{for } x_1 \leq 0, \\ -1 & \text{for } x_1 \geq 0, \end{cases} \\ \dot{x}_2 = \begin{cases} 1 & \text{for } x_2 \leq 0, \\ -1 & \text{for } x_2 \geq 0. \end{cases} \end{cases} \tag{3.7}$$

This system is a piecewise constant system, where in each quadrant of the (x_1, x_2)-plane the right hand side is a constant vector.

This example represents the simplest case of chattering-Zeno on a switching intersection. The phase space is split into 2^2 disjoint invariants by two intersected discontinuous surfaces $\Sigma_1 = \{x \in \mathbb{R}^2 : \; x_1 = 0\}$ and $\Sigma_2 = \{x \in \mathbb{R}^2 : \; x_2 = 0\}$.

The finite time convergence to the origin (0,0) is easy to establish since for $i = 1, 2$ we have $\dot{x}_i > 0$ when $x_i < 0$ and $\dot{x}_i < 0$ when $x_i > 0$.

This system also has an infinity of spontaneous switches from the origin, that is, there is an infinity of trajectories which start with the initial data (0,0) and, except for the trivial solution that stays at the origin, they all cross infinitely the switching surfaces Σ_1 and Σ_2 in a finite amount of time.

Example 3.4 (Stick-Slip System of Three Blocks Connected Vertically)
Consider a mechanical Stick-Slip system of three blocks and two frictional interfaces as sketched in Figure 3.4. Similarly to Example 3.1, we denote x_m, x_{M_1}, and x_{M_2} to the position of the small mass m and the two inertial masses M_1 and M_2 respectively.

We denote $\mathcal{F}_1$ to the tangential contact force on the frictional interface between the small mass m and the inertial mass M_1, and $\mathcal{F}_2$ to the tangential contact force on the frictional interface between the small mass m and the inertial mass M_2. The friction between the inertial mass M_2 and the ground is neglected. The origin of the displacements x_m, x_{M_1}, and x_{M_2} is taken where the spring is unstretched. For $x = [x_m\ v_m\ x_{M_1}\ v_{M_1}\ x_{M_2}\ v_{M_2}]^T$, the system's dynamics is given by

$$\dot{x} = f(x) = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m - \mathcal{F}_1 - \mathcal{F}_2), \\ \dot{x}_{M_1} = & v_{M_1}, \\ \dot{v}_{M_1} = & \frac{1}{M_1}\mathcal{F}_1, \\ \dot{x}_{M_2} = & v_{M_2}, \\ \dot{v}_{M_2} = & \frac{1}{M_2}\mathcal{F}_2, \end{cases} \tag{3.8}$$

where v_m, v_{M_1}, and v_{M_2} are the velocities of the blocks. The friction forces are given by

$$\mathcal{F}_1 = \begin{cases} -F_{c_1}, & \text{for } v_{r_1} \leq 0, \\ +F_{c_1}, & \text{for } v_{r_1} \geq 0, \end{cases} \qquad \mathcal{F}_2 = \begin{cases} -F_{c_2}, & \text{for } v_{r_2} \leq 0, \\ +F_{c_2}, & \text{for } v_{r_2} \geq 0, \end{cases} \tag{3.9}$$

where F_{c_1} and F_{c_2} are the levels of the Coulomb friction, $v_{r_1} = v_m - v_{M_1}$, and $v_{r_2} = v_m - v_{M_2}$. Figure 3.5 shows the hybrid automaton model of the system. This system is discontinuous on two intersected hyper switching manifolds $\Sigma_1, \Sigma_2 \subseteq \mathbb{R}^6$ defined as

$$\Sigma_1 = \Sigma_{11} \cup \Sigma_{12} = \{x \in \mathbb{R}^6 : v_{r_1} = v_m - v_{M_1} = 0\},$$

$$\Sigma_2 = \Sigma_{21} \cup \Sigma_{22} = \{x \in \mathbb{R}^6 : v_{r_2} = v_m - v_{M_2} = 0\},$$

where

$$\Sigma_{11} = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x \in \mathbb{R}^6 : v_{r_1} = 0 \wedge v_{r_2} \geq 0\},$$

$$\Sigma_{12} = \overline{\mathcal{G}(q_3, q_4)} \cap Inv(q_3) = \overline{\mathcal{G}(q_4, q_3)} \cap Inv(q_4) = \{x \in \mathbb{R}^6 : v_{r_1} = 0 \wedge v_{r_2} \leq 0\},$$

$$\Sigma_{21} = \overline{\mathcal{G}(q_1, q_4)} \cap Inv(q_1) = \overline{\mathcal{G}(q_4, q_1)} \cap Inv(q_4) = \{x \in \mathbb{R}^6 : v_{r_1} \geq 0 \wedge v_{r_2} = 0\},$$

$$\Sigma_{21} = \overline{\mathcal{G}(q_2, q_3)} \cap Inv(q_2) = \overline{\mathcal{G}(q_3, q_2)} \cap Inv(q_3) = \{x \in \mathbb{R}^6 : v_{r_1} \leq 0 \wedge v_{r_2} = 0\}.$$

As demonstrated in Figure 3.6, the intersection $\Delta = \Sigma_1 \cap \Sigma_2 = \{x \in \mathbb{R}^6 :\ v_{r_1} = 0\ \wedge\ v_{r_2} = 0\}$ splits the phase space into 2^2 disjoint invariants $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ where

$$\mathcal{S}_1 = \{x \in \mathbb{R}^6 :\ v_{r_1} \geq 0\ \wedge\ v_{r_2} \geq 0\}, \quad \mathcal{S}_2 = \{x \in \mathbb{R}^6 :\ v_{r_1} \leq 0\ \wedge\ v_{r_2} \geq 0\},$$

$$\mathcal{S}_3 = \{x \in \mathbb{R}^6 :\ v_{r_1} \leq 0\ \wedge\ v_{r_2} \leq 0\}, \quad \mathcal{S}_4 = \{x \in \mathbb{R}^6 :\ v_{r_1} \geq 0\ \wedge\ v_{r_2} \leq 0\}.$$

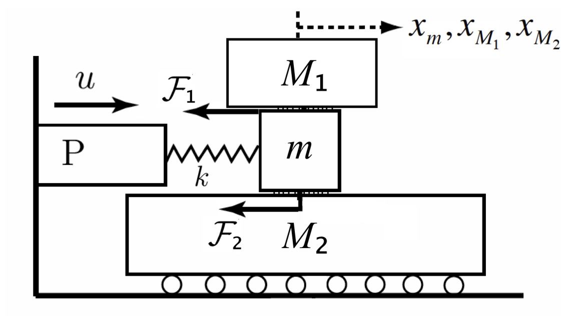


Figure 3.4: Schematic of the Stick-Slip system in Example 3.4.

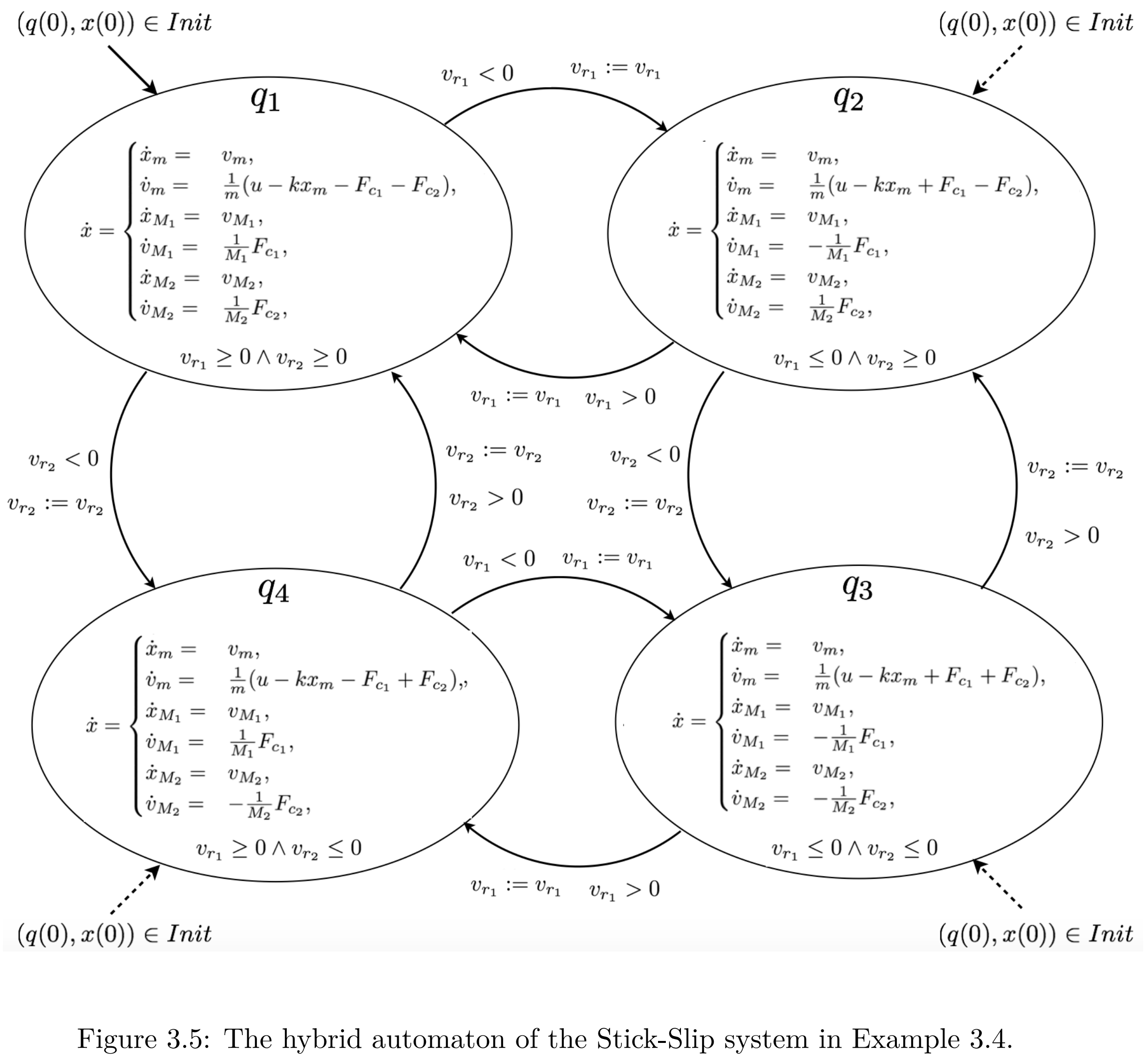

$(q(0), x(0)) \in Init$
$(q(0), x(0)) \in Init$
$v_{r_1} < 0$
$v_{r_1} := v_{r_1}$
q_1
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m - F_{c_1} - F_{c_2}), \\ \dot{x}_{M_1} = & v_{M_1}, \\ \dot{v}_{M_1} = & \frac{1}{M_1}F_{c_1}, \\ \dot{x}_{M_2} = & v_{M_2}, \\ \dot{v}_{M_2} = & \frac{1}{M_2}F_{c_2}, \end{cases}$
$v_{r_1} \geq 0 \wedge v_{r_2} \geq 0$
q_2
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m + F_{c_1} - F_{c_2}), \\ \dot{x}_{M_1} = & v_{M_1}, \\ \dot{v}_{M_1} = & -\frac{1}{M_1}F_{c_1}, \\ \dot{x}_{M_2} = & v_{M_2}, \\ \dot{v}_{M_2} = & \frac{1}{M_2}F_{c_2}, \end{cases}$
$v_{r_1} \leq 0 \wedge v_{r_2} \geq 0$
$v_{r_1} := v_{r_1}$
$v_{r_1} > 0$
$v_{r_2} < 0$
$v_{r_2} := v_{r_2}$
$v_{r_2} := v_{r_2}$
$v_{r_2} > 0$
$v_{r_2} < 0$
$v_{r_2} := v_{r_2}$
$v_{r_2} := v_{r_2}$
$v_{r_2} > 0$
$v_{r_1} < 0$
$v_{r_1} := v_{r_1}$
q_4
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m - F_{c_1} + F_{c_2}),, \\ \dot{x}_{M_1} = & v_{M_1}, \\ \dot{v}_{M_1} = & \frac{1}{M_1}F_{c_1}, \\ \dot{x}_{M_2} = & v_{M_2}, \\ \dot{v}_{M_2} = & -\frac{1}{M_2}F_{c_2}, \end{cases}$
$v_{r_1} \geq 0 \wedge v_{r_2} \leq 0$
q_3
$\dot{x} = \begin{cases} \dot{x}_m = & v_m, \\ \dot{v}_m = & \frac{1}{m}(u - kx_m + F_{c_1} + F_{c_2}), \\ \dot{x}_{M_1} = & v_{M_1}, \\ \dot{v}_{M_1} = & -\frac{1}{M_1}F_{c_1}, \\ \dot{x}_{M_2} = & v_{M_2}, \\ \dot{v}_{M_2} = & -\frac{1}{M_2}F_{c_2}, \end{cases}$
$v_{r_1} \leq 0 \wedge v_{r_2} \leq 0$
$v_{r_1} := v_{r_1}$
$v_{r_1} > 0$
$(q(0), x(0)) \in Init$
$(q(0), x(0)) \in Init$

Figure 3.5: The hybrid automaton of the Stick-Slip system in Example 3.4.

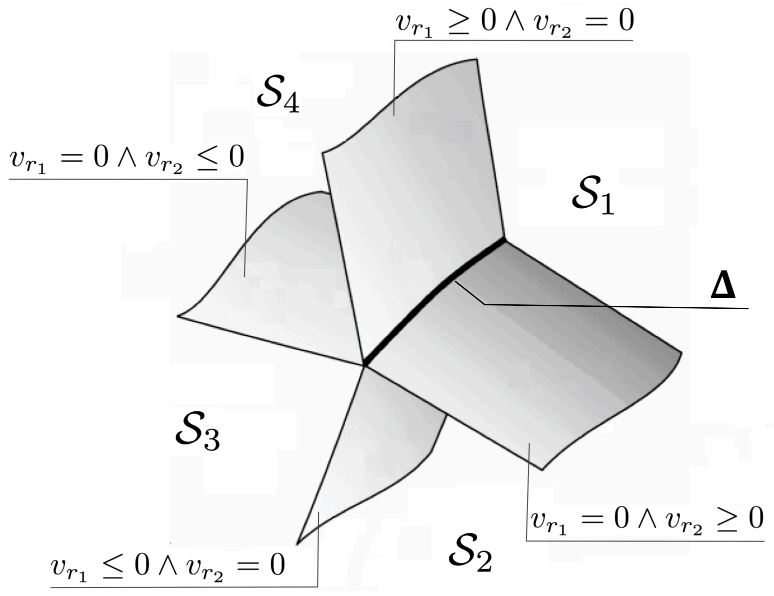

$v_{r_1} \geq 0 \wedge v_{r_2} = 0$
$\mathcal{S}_4$
$v_{r_1} = 0 \wedge v_{r_2} \leq 0$
$\mathcal{S}_1$
$\mathbf{\Delta}$
$\mathcal{S}_3$
$v_{r_1} = 0 \wedge v_{r_2} \geq 0$
$\mathcal{S}_2$
$v_{r_1} \leq 0 \wedge v_{r_2} = 0$

Figure 3.6: The state space of the Stick-Slip system in Example 3.4.

Let $\Sigma = \Sigma_1 \cup \Sigma_2$ be the entire discontinuity region in the model's phase space. A switching between the four different vector fields in $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ takes place in the neighborhood of the intersection Δ. A trajectory that crosses Σ transversally will switch instantaneously between these vector fields without any specific form of flow vector field on Σ. The only alternative is that the model exhibits a chattering-Zeno on Σ when either on one of the frictional interfaces or on both of them the applied tangential friction force acting on the interface is lower than the level of the corresponding coulomb friction. This yields: i) a chattering-Zeno on Σ_1 either between $\mathcal{S}_1$ and $\mathcal{S}_2$ or between $\mathcal{S}_3$ and $\mathcal{S}_4$, ii) a chattering-Zeno on Σ_2 either between $\mathcal{S}_1$ and $\mathcal{S}_4$ or between $\mathcal{S}_2$ and $\mathcal{S}_3$, or iii) a chattering-Zeno on the intersection Δ between the invariants $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$.

Example 3.5 (Spring-Block Chain on a Conveyor Belt)
Consider a mechanical system consisting of three blocks of masses m_1, m_2, and m_3 on a moving belt with velocity v_d, as sketched in Figure 3.7. The three blocks are connected along a line by two linear springs of stiffness k_{12} and k_{23}, and connected to a fix support by to linear springs of stiffness k_1, k_2, and k_3. The system's dynamics is given by

$$\dot{x} = f(x) = \begin{cases} \dot{x}_{m_1} = & v_{m_1}, \\ \dot{v}_{m_1} = & \frac{1}{m_1}(u_1 - k_1 x_{m_1} - \mathcal{F}_1), \\ \dot{x}_{m_2} = & v_{m_2}, \\ \dot{v}_{m_2} = & \frac{1}{m_2}(u_2 - k_2 x_{m_2} - \mathcal{F}_2), \\ \dot{x}_{m_3} = & v_{m_3}, \\ \dot{v}_{m_3} = & \frac{1}{m_3}(u_3 - k_3 x_{m_3} - \mathcal{F}_3), \end{cases} \tag{3.10}$$

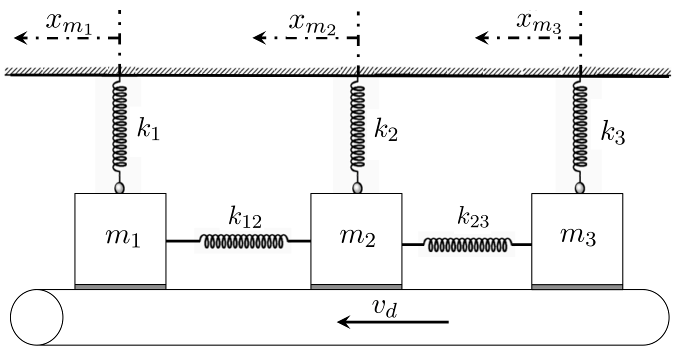


Figure 3.7: Schematic of the Spring-Block chain system in Example 3.5.

with

$$u_1 = k_{12}(x_{m_2} - x_{m_1}) + k_{13}(x_{m_3} - x_{m_1}),$$
$$u_2 = k_{12}(x_{m_1} - x_{m_2}) + k_{23}(x_{m_3} - x_{m_2}),$$
$$u_3 = k_{13}(x_{m_1} - x_{m_3}) + k_{23}(x_{m_2} - x_{m_3}),$$

where $v_{r_i} = v_{m_i} - v_d$. We denote x_{m_1}, x_{m_2}, and x_{m_3} to the position of the masses m_1, m_2, and m_3; $\mathcal{F}_1$, $\mathcal{F}_2$, and $\mathcal{F}_3$ to the tangential contact force on the frictional interfaces between the moving belt and the three masses m_1, m_2, and m_3 respectively, where for all $i \in \{1, 2, 3\}$ we have $\mathcal{F}_i = -F_{c_i}$ if $v_{r_i} \leq 0$ and $\mathcal{F}_i = +F_{c_i}$ if $v_{r_i} \geq 0$. This system is discontinuous on three hyper switching manifolds $\Sigma_1, \Sigma_2, \Sigma_3 \in \mathbb{R}^5$ given by

$$\Sigma_1 = \Sigma_{11} \cup \Sigma_{12} \cup \Sigma_{13} \cup \Sigma_{14} = \{x \in \mathbb{R}^6 : v_{r_1} = v_{m_1} - v_d = 0\},$$
$$\Sigma_2 = \Sigma_{21} \cup \Sigma_{22} \cup \Sigma_{23} \cup \Sigma_{24} = \{x \in \mathbb{R}^6 : v_{r_1} = v_{m_2} - v_d = 0\},$$
$$\Sigma_3 = \Sigma_{31} \cup \Sigma_{32} \cup \Sigma_{33} \cup \Sigma_{34} = \{x \in \mathbb{R}^6 : v_{r_1} = v_{m_3} - v_d = 0\},$$

where

$$\Sigma_{11} = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x : v_{r_1} = 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\Sigma_{12} = \overline{\mathcal{G}(q_3, q_4)} \cap Inv(q_3) = \overline{\mathcal{G}(q_4, q_3)} \cap Inv(q_4) = \{x : v_{r_1} = 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\Sigma_{13} = \overline{\mathcal{G}(q_5, q_6)} \cap Inv(q_5) = \overline{\mathcal{G}(q_6, q_5)} \cap Inv(q_6) = \{x : v_{r_1} = 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\Sigma_{14} = \overline{\mathcal{G}(q_7, q_8)} \cap Inv(q_7) = \overline{\mathcal{G}(q_8, q_7)} \cap Inv(q_8) = \{x : v_{r_1} = 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\Sigma_{21} = \overline{\mathcal{G}(q_1, q_4)} \cap Inv(q_1) = \overline{\mathcal{G}(q_4, q_1)} \cap Inv(q_4) = \{x : v_{r_2} = 0 \ \wedge \ v_{r_1} \geq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\Sigma_{22} = \overline{\mathcal{G}(q_2, q_3)} \cap Inv(q_2) = \overline{\mathcal{G}(q_3, q_2)} \cap Inv(q_3) = \{x : v_{r_2} = 0 \ \wedge \ v_{r_1} \leq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\Sigma_{23} = \overline{\mathcal{G}(q_5, q_8)} \cap Inv(q_5) = \overline{\mathcal{G}(q_8, q_5)} \cap Inv(q_8) = \{x : v_{r_2} = 0 \ \wedge \ v_{r_1} \geq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\Sigma_{24} = \overline{\mathcal{G}(q_6, q_7)} \cap Inv(q_6) = \overline{\mathcal{G}(q_7, q_6)} \cap Inv(q_7) = \{x : v_{r_2} = 0 \ \wedge \ v_{r_1} \leq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\Sigma_{31} = \overline{\mathcal{G}(q_1, q_5)} \cap Inv(q_1) = \overline{\mathcal{G}(q_5, q_1)} \cap Inv(q_5) = \{x : v_{r_3} = 0 \ \wedge \ v_{r_1} \geq 0 \ \wedge \ v_{r_2} \geq 0\},$$
$$\Sigma_{32} = \overline{\mathcal{G}(q_2, q_6)} \cap Inv(q_2) = \overline{\mathcal{G}(q_6, q_2)} \cap Inv(q_6) = \{x : v_{r_3} = 0 \ \wedge \ v_{r_1} \leq 0 \ \wedge \ v_{r_2} \geq 0\},$$

$$\Sigma_{33} = \overline{\mathcal{G}(q_4, q_8)} \cap Inv(q_4) = \overline{\mathcal{G}(q_8, q_4)} \cap Inv(q_8) = \{x : v_{r_3} = 0 \ \wedge \ v_{r_1} \ \geq 0 \ \wedge \ v_{r_2} \leq 0\},$$

$$\Sigma_{34} = \overline{\mathcal{G}(q_3, q_7)} \cap Inv(q_3) = \overline{\mathcal{G}(q_7, q_3)} \cap Inv(q_7) = \{x : v_{r_3} = 0 \ \wedge \ v_{r_1} \ \leq 0 \ \wedge \ v_{r_2} \leq 0\}.$$

Figure 3.8 shows the hybrid automaton model of the system.

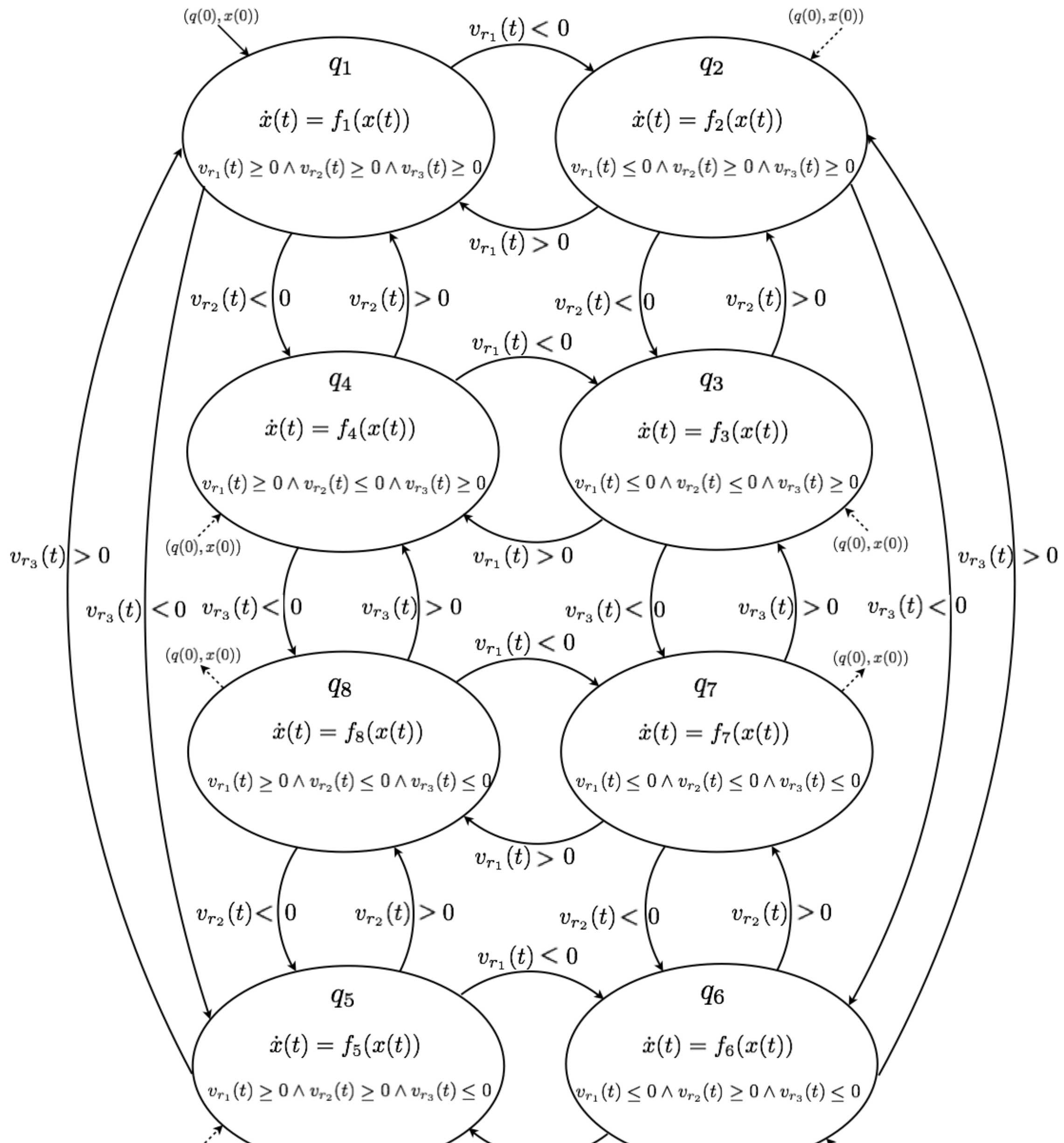


Figure 3.8: The hybrid automaton of the Spring-Block chain system in Example 3.5.

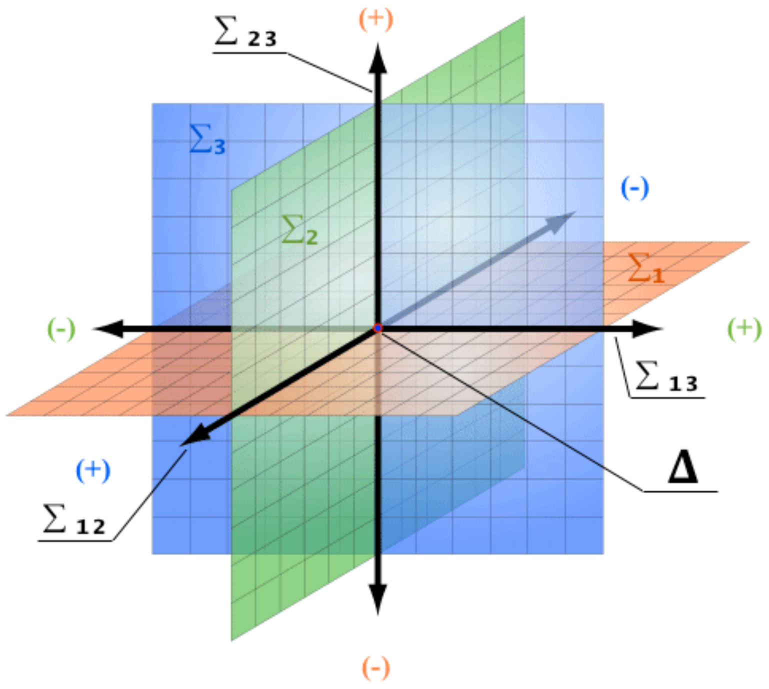


Figure 3.9: The state space of the Spring-Block chain system in Example 3.5.

In addition to the $\mathbb{R}^5$-dimensional switching manifolds Σ_1, Σ_2, and Σ_3, the discontinuity region includes three $\mathbb{R}^4$-dimensional switching intersections $\Sigma_i \cap \Sigma_j|_{i,j\in\{1,2,3\},i\neq j}$, and a single $\mathbb{R}^3$-dimensional switching intersection (point) $\Delta = \bigcap_{k=1}^{3} \Sigma_k$.

As demonstrated in Figure 3.9, the intersection $\Delta = \Sigma_1 \cap \Sigma_2 \cap \Sigma_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} = 0 \ \wedge \ v_{r_2} = 0 \ \wedge \ v_{r_3} = 0\}$ splits the phase space into 2^3 disjoint invariants

$$\mathcal{S}_1 = \{x \in \mathbb{R}^6 : \ v_{r_1} \geq 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\mathcal{S}_2 = \{x \in \mathbb{R}^6 : \ v_{r_1} \leq 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\mathcal{S}_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} \leq 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\mathcal{S}_4 = \{x \in \mathbb{R}^6 : \ v_{r_1} \geq 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \geq 0\},$$
$$\mathcal{S}_5 = \{x \in \mathbb{R}^6 : \ v_{r_1} \geq 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\mathcal{S}_6 = \{x \in \mathbb{R}^6 : \ v_{r_1} \leq 0 \ \wedge \ v_{r_2} \geq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\mathcal{S}_7 = \{x \in \mathbb{R}^6 : \ v_{r_1} \leq 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \leq 0\},$$
$$\mathcal{S}_8 = \{x \in \mathbb{R}^6 : \ v_{r_1} \geq 0 \ \wedge \ v_{r_2} \leq 0 \ \wedge \ v_{r_3} \leq 0\}.$$

A chattering-Zeno occurs on the discontinuous surface Σ_i if the applied friction force on the frictional interface between the block of mass m_i and the moving belt is lower than the corresponding level of Coulomb friction F_{c_i}. When a chattering-Zeno occurs on a switching intersection, the solution trajectory moves infinitely back and force between all the invariants in the neighborhood of the intersection.

3.1.2 Challenges of Simulating Chattering-Zeno Models

As we mentioned in Section 1.2.2, hybrid systems simulation tools struggle when simulating chattering-Zeno models. We distinguish two different situations:

1. **Variable step simulation:** When simulating a chattering-Zeno model with variable step simulation, the root finding to locate the exact time of occurrence of the chattering-Zeno event breaks down the numerical integration. The simulation inevitably halts at the chattering-Zeno limit point, as infinitely many discrete transitions would need to be simulated.
   For example, consider the numerical simulation of Example 3.2 (the case of chattering on a single switching manifold) and Example 3.3 (the case of chattering on switching intersection) with a variable step simulation in OpenModelica simulation tool. When simulating the model of the system in Example 3.2 in OpenModelica with a variable step simulation for a data set $m = M = 1[kg]$, $F_c = 0.4[N]$, $\omega = 0.055[rad/sec]$, $k = 1[N \cdot m^{-1}]$, $x_0 = [1\ 1\ 1\ 0]^T$, the simulation effectively terminates with a halt when the solution reaches the chattering-Zeno point at time 2.895. OpenModelica reports the following message: *{Chattering detected around time 2.89522335659..2.89522338163 (100 state events in a row with a total time delta less than the step size 0.0012)}*.

   OpenModelica Code of Example 3.2:
   ```
   model example-3.2
   parameter Real Fc=0.4, w=0.055,k=1,m=1,M=1;
   parameter Real xm0=1, vm0=1, xM0=1, vM0=0;
   Real xm,vm,xM,vM,F;
   initial equation
   xm = xm0; vm = vm0; xM = xM0; vM = vM0; F = Fc * sign(vm - vM);
   equation
   when vm - vM < 0 then
         F = -Fc;
   elsewhen vm - vM > 0 then
         F = Fc;
   end when;
   der(xm) = vm;
   der(vm) = sin(w) - k * xm - F;
   der(xM) = vM;
   der(vM) = F;
   end example-3.2;
   ```

   Similarly, when simulating the model in Example 3.3 with variable step simulation, the simulation gets stuck at time 1.0. In OpenModelica, the simulation halts with the following error message: *Chattering detected around time 1.0000000001..1.0000000199 (100 state events in a row with a total time delta less than the step size 0.008).*

OpenModelica Code of Example 3.3:

```
model example-3.3
parameter Real x10 = 1, x20 = 2;
Real x1,x2, u1, u2;
initial equation
x1 = x10; x2 = x20; u1 = -sign(x1); u2 = -sign(x2);
equation
when x1 < 0 then
      u1 = 1;
elsewhen x1 > 0 then
      u1 = -1;
end when;
when x2 < 0 then
      u2 = 1;
elsewhen x2 > 0 then
      u2 = -1;
end when;
der(x1) = u1; der(x2) = u2;
end example-3.3;
```

2. **Fixed step simulation:** When simulating a chattering-Zeno model with fixed step simulation, the simulation does not terminate with a halt, as fixed step simulation has the advantage of being Zeno-free by construction, i.e. the time advances by a constant value ignoring the localization of the events. However this also produces wrong simulation results whenever the solution signal activity is higher than the fixed sampling frequency. Events associated with jumps to completely different dynamics could be missed, which results a behavior deviant from the one expected. Furthermore, the smaller the fixed step size the higher is the frequency of the fast oscillation around the switching surface, and the greater is then the time consumption of the simulation process, i.e. simulation becomes excessively slow. Consider for example the fixed step simulation of Example 3.2 and Example 3.3 in Acumen. Figure 3.10 shows the simulation of Example 3.2 in Acumen with a fixed time step of 0.0012, and the same data above.

Acumen Code of Example 3.2:

```
model Main(simulator) =
initially
xm = 1, vm = 1, xM = 1, vM = 0, xm'= 0, xM'= 0, vm'= 0, vM'= 0,
Fc = 0.4, w = 0.055, vr = 0, k = 1, m = 1, M = 1, F = 0
always xm' = vm, vm' = (1/m) * (sin(w) - (k * xm) - F),
xM' = vM, vM' = (1/M) * F,
if vm - vM > 0 then F = Fc
elseif vm - vM < 0 then F = -Fc noelse,
simulator.timeStep += 0.0012, simulator.endTime += 10
```

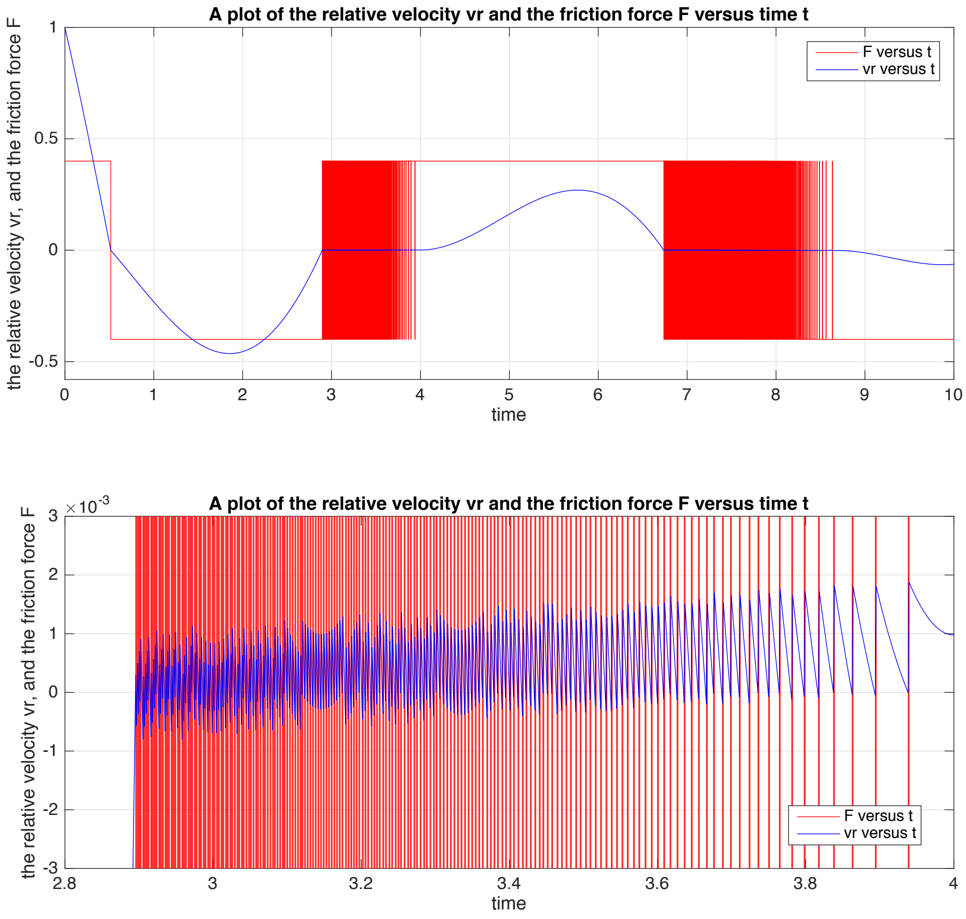


Figure 3.10: Fixed time step simulation of Example 3.2 in Acumen. Up: time evolution of the event function and the control input. Down: zoom on the first chattering-Zeno window around the switching surface $\Sigma = \{x \in \mathbb{R}^4 : v_r = v_m - v_M = 0\}$.

Figure 3.11 shows the fixed time step simulation of Example 3.3 in Acumen for a step size of 0.0012, and initial conditions $x_1(0) = 1, x_2(0) = 2$.

Acumen Code of Example 3.3:

```
model Main(simulator) =
initially x1 = 1, x1' = 0, x2 = 2, x2' = 0, u1 = 0, u2 = 0
always x1' = u1 - (2*u1*u2*u2), x2' = u2 - (2*u2*u1*u1),
if x1 > 0 then u1 = 1 elseif x1 < 0 then u1 = -1 noelse,
if x2 > 0 then u2 = 1
elseif x2 < 0 then u2 = -1 noelse,
simulator.timeStep += 0.0012,
simulator.endTime += 4
```

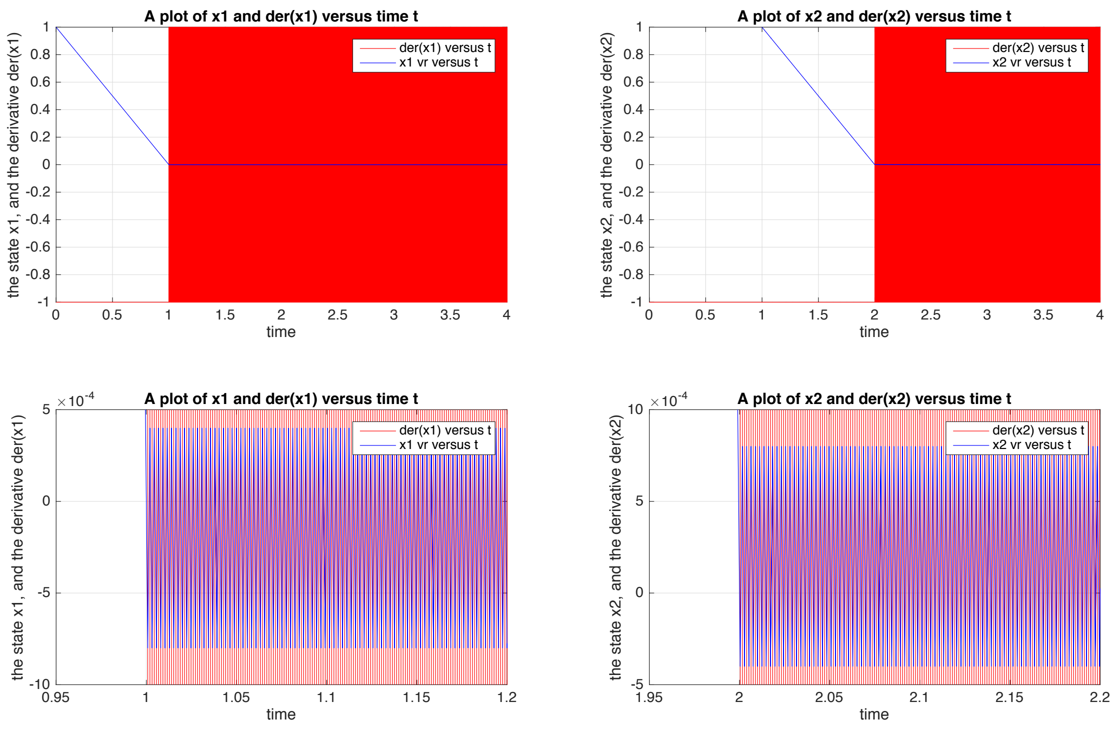


Figure 3.11: Fixed time step simulation of Example 3.3 in Acumen. Up: time evolution of the states and their derivatives. Down: zoom on the chattering-Zeno windows around $\Sigma_1 = \{x \in \mathbb{R}^2 : x_1 = 0\}$ and the origin $\Delta = \{x \in \mathbb{R}^2 : x_1 = 0 \wedge x_2 = 0\}$.

Many methods have been proposed in the literature to deal with models exhibiting chattering-Zeno behavior.

One of the proposed methods to add a small hysteresis ϵ to the event functions which represent the switching surfaces. In Section 1.3 we discussed the disadvantages of this approach.

Another proposed method to deal with chattering-Zeno is based on sliding mode approach, that is, detecting regions on the switching manifold in which chattering-Zeno occurs and then forcing the solution trajectory to slide on the manifold in these regions. While the fast infinite switching between modes occurs, a smooth sliding motion along the switching surface may emerge while eliminating the fast chattering. Differential Inclusions (DIs) of the Filippov type (the so-called equivalent dynamics) is a method that was developed by Filippov for formulating the equations for flows that slide on the chattering-Zeno portions of switching surfaces.

In the following section, we give an introduction to the sliding mode approach, and then we discuss the limitations of the Filippov method.

3.2 Sliding Mode Approach

We restrict ourselves to abstracted models of hybrid systems where the system's dynamics is discontinuous on a finite number of hyper switching surfaces. Firstly, we start with the simplest case of chattering-Zeno where the model solution trajectory chatters between two different dynamics on a single switching surface, and then we discuss the general case of chattering-Zeno where chattering-Zeno occurs on switching surfaces intersections between more than two dynamics.

3.2.1 The Case of Chattering-Zeno Between Two Dynamics

This is the case of a hybrid automaton model of two locations, where the state space $\mathbb{R}^n$ of the system is split into two disjoint invariants $\mathcal{S}_1$ and $\mathcal{S}_2$ by a hyper switching surface Σ such that $\mathbb{R}^n = \mathcal{S}_1 \cup \Sigma \cup \mathcal{S}_2$; see Figure 3.12. The hyper surface Σ is defined by a scalar continuous function $\gamma(x)$ where the state x is in Σ when $\gamma(x) = 0$. The invariants $\mathcal{S}_1$ and $\mathcal{S}_2$ and the hyper surface Σ can be formulated as

$$\Sigma = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x \in \mathbb{R}^n : \gamma(x) = 0\}, \tag{3.11}$$

$$\mathcal{S}_1 = Inv(q_1) = \{x \in \mathbb{R}^n : \ \gamma(x) \leq 0\}, \tag{3.12}$$

$$\mathcal{S}_2 = Inv(q_2) = \{x \in \mathbb{R}^n : \ \gamma(x) \geq 0\}. \tag{3.13}$$

The dynamics of the system in the state space is given by

$$\dot{x} = f(x) = \begin{cases} f_1(x), & \text{for } x \in \mathcal{S}_1, \\ f_2(x), & \text{for } x \in \mathcal{S}_2, \end{cases} \tag{3.14}$$

where the vector field f_1, respectively f_2, is assumed to be Lipschitz continuous on the invariant $\mathcal{S}_1$, respectively $\mathcal{S}_2$, to ensure that the solution within $\mathcal{S}_1$ and $\mathcal{S}_2$ exists and is unique.

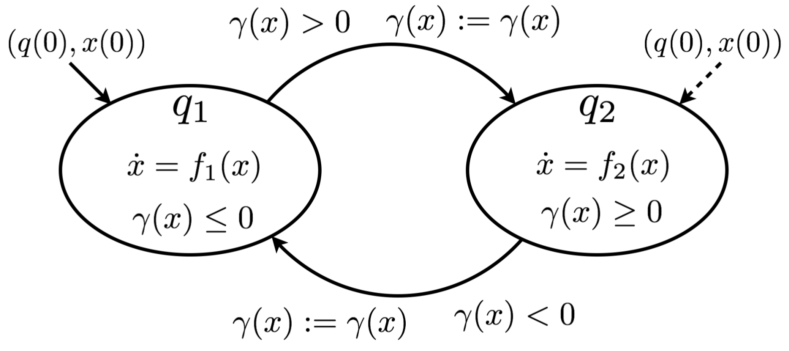


Figure 3.12: A hybrid automaton with two modes and a hyper switching surface.

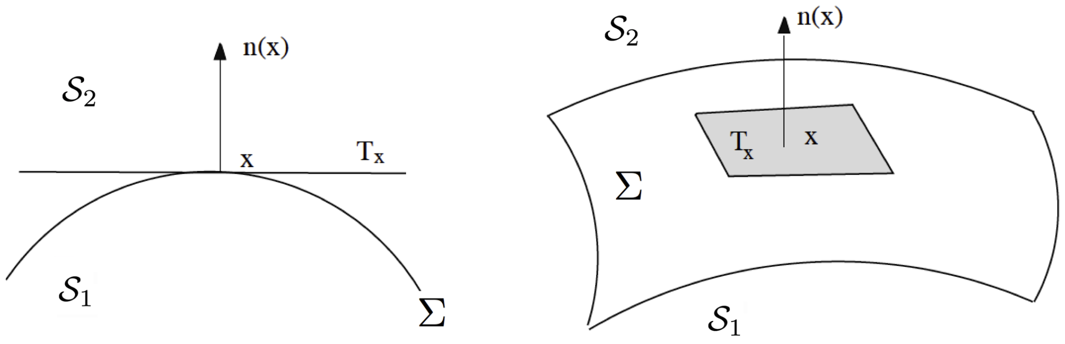


Figure 3.13: The normal n orthogonal to the tangent plane $T_x(\Sigma)$ and normal to Σ.

Notation 3.2 Denote $x^\star$ to a discontinuity point in Σ, and $x^\bullet$ to the limit point approaching $x^\star$ from a small neighborhood in the invariant $\mathcal{S}_i$.

The normal n perpendicular to Σ (Figure 3.13) is given then for all $x^\star$ by

$$n(x^\star) = \frac{\nabla\gamma(x^\star)}{||\nabla\gamma(x^\star)||}, \quad \nabla\gamma(x^\star) = \frac{\partial\gamma(x^\star)}{\partial x}, \tag{3.15}$$

where we assume that γ is in $\mathcal{C}^1(\mathbb{R}^n)$ and that $\frac{\partial\gamma(x^\star)}{\partial x} \neq 0$ for all $x^\star$.

Let $f_1^N(x^\star)$ and $f_2^N(x^\star)$ be the normal projections of dynamics f_1 and f_2 at the discontinuity point $x^\star \in \Sigma$. The normal projections$f_1^N(x^\star)$ and $f_2^N(x^\star)$ are given by

$$f_1^N(x^\star) = n(x^\star)^T \cdot f_1(x^\bullet), \quad x^\bullet \in \mathcal{S}_1, \tag{3.16}$$

$$f_2^N(x^\star) = n(x^\star)^T \cdot f_2(x^\bullet), \quad x^\bullet \in \mathcal{S}_2. \tag{3.17}$$

From a geometric point of view, the normal projections $f_1^N(x^\star)$ and $f_2^N(x^\star)$ represent the limits $f_1^+(x^\star)$ and $f_2^+(x^\star)$ as the point $x^\star \in \Sigma$ is approached from opposite sides of the tangent plane to Σ, that is

$$f_1^N(x^\star) = f_1^+(x^\star) = \lim_{x^\bullet \in \mathcal{S}_1 \to x^\star \in \Sigma} x^\bullet, \tag{3.18}$$

$$f_2^N(x^\star) = f_2^+(x^\star) = \lim_{x^\bullet \in \mathcal{S}_2 \to x^\star \in \Sigma} x^\bullet. \tag{3.19}$$

Upon hitting the hyper discontinuous surface Σ, the behavior of the solution trajectory is characterized by the directions of the vector fields on both sides of the Σ. We distinguish between the following three different cases:

1. If the vector fields — in the invariants — on both sides of the hyper switching surface Σ have the same direction, that is

$$f_1^N(x^\star) \cdot f_2^N(x^\star) > 0, \tag{3.20}$$

then the solution hits Σ and passes through the discontinuity; see Figure 3.14. Any solution $x(t)$ initialized outside Σ and hits Σ transversally exists and is unique.

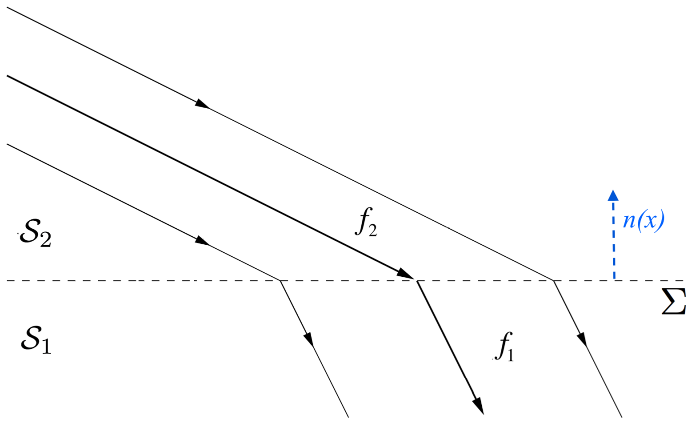


Figure 3.14: The case in which the solution trajectory crosses Σ.

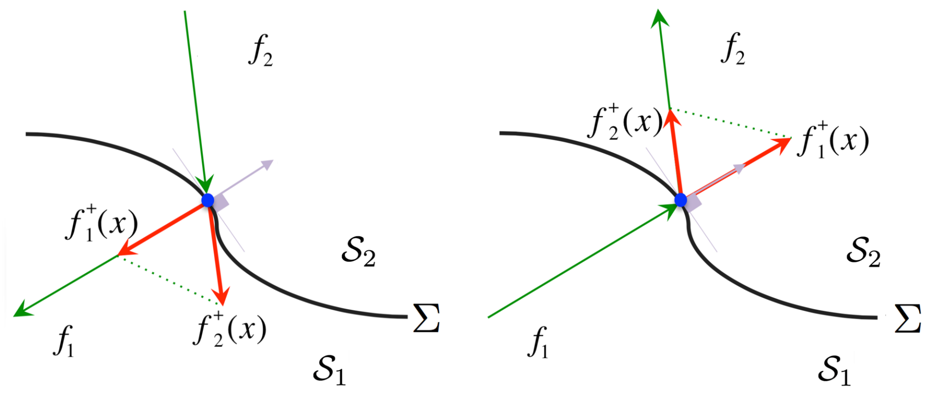


Figure 3.15: The case in which the solution trajectory crosses Σ.

Geometrically, if the line segment joining the endpoints of $f_1^N(x^\star)$ and $f_2^N(x^\star)$ is on one side of the tangential plane $T_x(\Sigma)$, then the solution crosses the hyper switching surface Σ to the region to which belong the line; see Figure 3.15.

2. If the vector fields — in the invariants — on both sides of the hyper switching surface Σ have opposite directions, then two scenarios are possible:

   (a) **Chattering-Zeno:** If the normal projections $f_1^N(x^\star)$ and $f_2^N(x^\star)$ have opposite signs but pointing towards the surface, that is

$$f_1^N(x^\star) \cdot f_2^N(x^\star) < 0, \tag{3.21}$$

$$f_i^N(x^\star) \cdot \gamma(x^\bullet) < 0|_{i\in\{1,2\}}, \quad x^\bullet \in \mathcal{S}_i, \tag{3.22}$$

   then the solution after hitting Σ has a tendency to remain on Σ and move along it. This is the case in which chattering-Zeno occurs, as the solution is pushed from both sides of the switching surface Σ; see Figure 3.16.

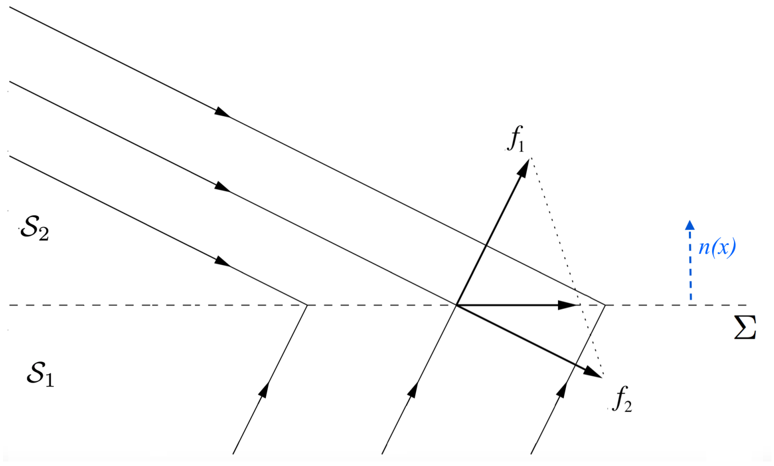


Figure 3.16: The case in which a chattering-Zeno occurs on Σ.

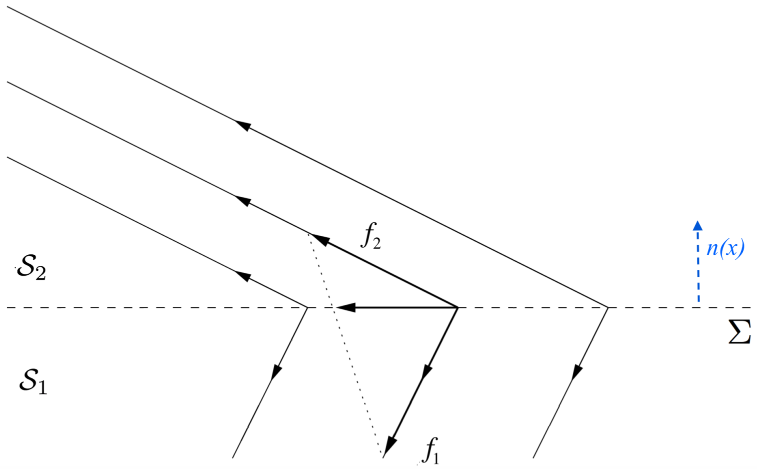


Figure 3.17: The case of non-uniqueness of solution on Σ.

(b) **Non-determinism:** A special case is when the normal projections $f_1^N(x(t))$ and $f_2^N(x(t))$ have opposite signs but pointing out of Σ, that is

$$f_1^N(x^\star) \cdot f_2^N(x^\star) < 0, \tag{3.23}$$

$$f_i^N(x^\star) \cdot \gamma(x^\bullet) > 0|_{i\in\{1,2\}}, \quad x^\bullet \in \mathcal{S}_i. \tag{3.24}$$

In this case, a solution which starts close to Σ will move away from it. But a solution emanating from Σ may either stay on Σ or go off the surface Σ into the invariants $\mathcal{S}_1$ or $\mathcal{S}_2$; see Figure 3.17. Such situations lead to non-unique solutions, and hence a non-deterministic hybrid model. That is, the solution still exists but is not unique in forward time.

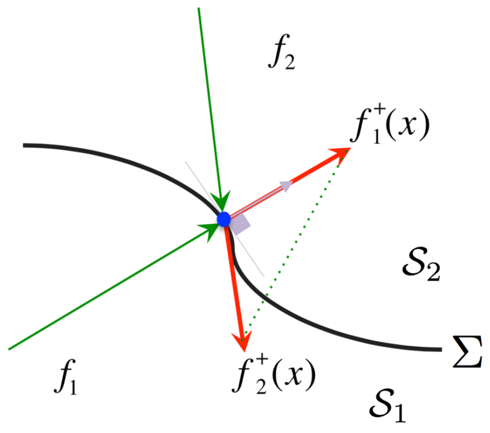


Figure 3.18: The case in which the solution trajectory chatters on Σ.

Clearly, when the normal projections $f_1^N(x^\star)$ and $f_2^N(x^\star)$ have opposite signs, the line segment joining the endpoints of $f_1^N(x^\star)$ and $f_2^N(x^\star)$ intersects the tangential plane $T_x(\Sigma)$; see Figure 3.18 for the case of chattering-Zeno.

3. The case in which $f_1^N(x^\star) \cdot f_2^N(x^\star) = 0$ indicates that one of the vector fields is tangential to the switching surface Σ, that is, one of the normal projections starts to change its sign at $x^\star \in \Sigma$. In this case $x^\star \in \Sigma$ is called tangential discontinuity point. A solution which is initialized at a tangential discontinuity point or hits it coming from an invariant $\mathcal{S}_i$ will cross Σ and evolve either in the invariant $\mathcal{S}_j$ or in the invariant $\mathcal{S}_i$. A solution that hits a tangential discontinuity point during sliding on Σ will exit from sliding tangentially and evolve in one of the disjoint invariants $\mathcal{S}_i$ or $\mathcal{S}_j$; see Figure 3.19. Roughly speaking, a smooth exit from sliding always takes place on tangential exit points, and therefore such points form the borders of the sliding region, and can be considered as points that belong to the crossing region on Σ. In Figure 3.19, the sets of tangential discontinuity points ζ_1 and ζ_2 are given by
$$\zeta_1 = \{x \in \Sigma : f_1^N(x) = 0\}, \tag{3.25}$$
$$\zeta_2 = \{x \in \Sigma : f_2^N(x) = 0\}. \tag{3.26}$$

Definition 3.3 (Crossing Region) *The crossing region on the hyper switching surface* $\Sigma = \overline{\mathcal{G}(q_1, q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2, q_1)} \cap Inv(q_2) = \{x^\star \in \mathbb{R}^n : \gamma(x^\star) = 0\}$ *of a hybrid automaton model with dynamics of the form*
$$\dot{x} = f(x) = \begin{cases} f_1(x), & \textit{for } \gamma(x) \le 0, \\ f_2(x), & \textit{for } \gamma(x) \ge 0, \end{cases}$$
is given by that portion $\Sigma^c \subseteq \Sigma$ *for which* $f_1^N(x^\star) \cdot f_2^N(x^\star) \ge 0$, $x^\star \in \Sigma$, *where*
$$f_i^N(x^\star) = \left(\frac{\nabla\gamma(x^\star)}{||\nabla\gamma(x^\star)||}\right)^T \cdot f_i(x^\bullet), \quad x^\bullet \notin \Sigma \to x^\star \in \Sigma.$$

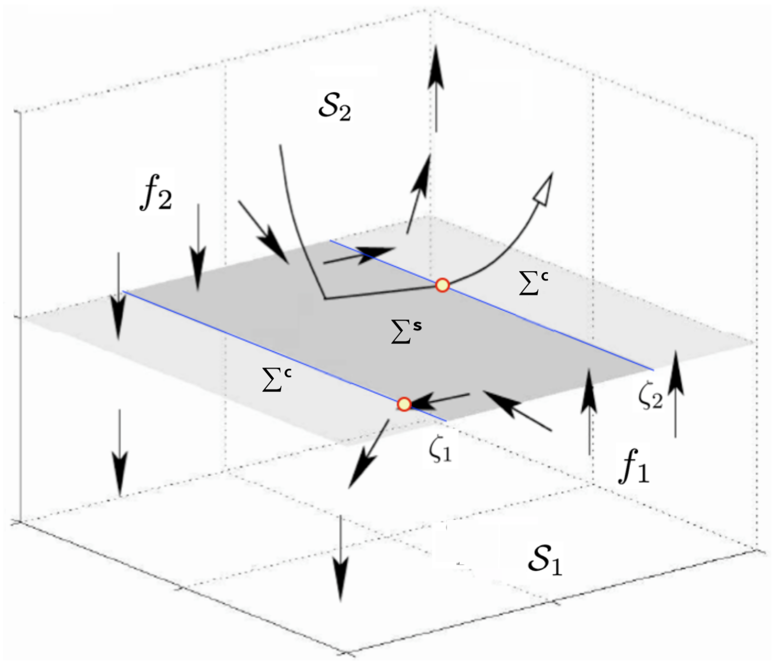


Figure 3.19: Tangential discontinuity points of crossing or exit from sliding.

Definition 3.4 (Sliding Region) *The sliding region on the hyper switching surface* $\Sigma = \overline{\mathcal{G}(q_1,q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2,q_1)} \cap Inv(q_2) = \{x^\star \in \mathbb{R}^n : \gamma(x^\star) = 0\}$ *of a hybrid automaton model with dynamics of the form*

$$\dot{x} = f(x) = \begin{cases} f_1(x), & \textit{for } \gamma(x) \le 0, \\ f_2(x), & \textit{for } \gamma(x) \ge 0, \end{cases}$$

is given by that portion $\Sigma^s \subseteq \Sigma$ *for which: i)* $f_1^N(x^\star) \cdot f_2^N(x^\star) < 0,\ x^\star \in \Sigma$, *and ii)* $f_i^N(x^\star) \cdot \gamma(x^\bullet) < 0$, *where*

$$f_i^N(x^\star) = \left(\frac{\nabla\gamma(x^\star)}{||\nabla\gamma(x^\star)||}\right)^T \cdot f_i(x^\bullet), \quad x^\bullet \notin \Sigma \rightarrow x^\star \in \Sigma.$$

Yet, it is not clear how the dynamics is defined on the hyper switching surface Σ. That is, the system's dynamics in (3.14) does not define a vector field that can be used to represent sliding on the surface Σ when chattering-Zeno occurs on it. This problem can be overcome by replacing (3.14) by a set-valued extension $F(x)$ which includes the convex set with two right-hand sides $f_1(x)$ and $f_2(x)$

$$\dot{x} \in F(x) = \begin{cases} f_1(x), & \text{for } x \in \mathcal{S}_1, \\ \overline{co}\{f_1(x), f_2(x)\} = \{(1-\alpha)f_1(x) + \alpha f_2(x)\}, & \text{for } x \in \Sigma, \\ f_2(x), & \text{for } x \in \mathcal{S}_2, \end{cases} \tag{3.27}$$

where the parameter α is a coefficient which defines the convex combination and has no physical meaning. This parameter α should satisfy $\alpha = [0, 1]$ to ensure sliding on Σ.

The extension of (3.14) into a convex differential inclusion (3.27) is known as *Filippov's convex method*. According to Fillipov, when chattering-Zeno occurs on the hyper switching surface Σ, the dynamics of the system should be such that the evolution of the system takes place on that surface. Thus, in the sliding region Σ^s (Definition 3.4) the sliding dynamics $f_s(x)$ should satisfy $\dot{x} = f_s(x) \in \overline{co}\{f_1(x), f_2(x)\}$ for every $x \in \Sigma^s$. To force the solution to be in a sliding motion we impose $\dot{\gamma}(x) = 0$ for $x \in \Sigma^s$. That is, the normal projection of the sliding dynamics onto Σ is equal to 0

$$f_s^N(x) = \left(\frac{\nabla\gamma(x)}{||\nabla\gamma(x)||}\right)^T \cdot \overline{co}\{f_1(x), f_2(x)\} = 0, \tag{3.28}$$

which yields

$$\alpha = \frac{f_1^N(x)}{f_1^N(x) - f_2^N(x)} = \frac{\nabla\gamma(x)^T \cdot f_1(x)}{\nabla\gamma(x)^T \cdot (f_1(x) - f_2(x))}, \tag{3.29}$$

and therefore the sliding vector field $f_s(x)$ is given by

$$\dot{x} = f_s(x) = \frac{f_1^N(x) f_2(x) - f_2^N(x) f_1(x)}{f_1^N(x) - f_2^N(x)}. \tag{3.30}$$

It is obvious from the signs of normal projections $f_i^N(x)$ that the value of the parameter α in (3.29) always satisfies $\alpha \in (0, 1)$ in the sliding region Σ^s, so that the sliding vector field $f_s(x)$ is always tangential to the surface Σ in this region. The solution on the switching surface in this case is known as a "sliding motion" or "sliding mode" solution. Clearly, the values $\alpha = 0$ and $\alpha = 1$ are the borders of the sliding region, i.e. the points of tangential exit from sliding.

From a geometric point of view, the end point of the vector field $f_s(x)$ in (3.30) lies on the intersection of the tangent plane $T_x(\Sigma)$ with a linear segment joining the endpoints of the the normal projections $f_1^N(x)$ and $f_2^N(x)$; see Figure 3.20. Existence can be guaranteed with the following notion of upper semi-continuity of set-valued functions.

Definition 3.5 (Upper Semi-Continuity) *A set valued function $F(x)$ is upper semi-continuous in x if for $y \to x$*

$$sup_{a \in F(y)} inf_{b \in F(x)} |a - b| \to 0 \tag{3.31}$$

The following theorem is proven in Aubin and Cellina [19] (theorem 3, page 98):

Theorem 3.6 (Existence of Solution of a Differential Inclusion) *Let F be a set-valued function. We assume that F is upper semi-continuous, non-empty, closed, convex and bounded for all $x \in \mathbb{R}^n$. Then for each $x_0 \in \mathbb{R}^n$ there exists a $\tau > 0$ and an absolutely continuous function $x(t)$ defined on $[0, \tau]$, which is a solution of the initial value problem*

$$\dot{x} \in F(x), \quad x(0) = x_0. \tag{3.32}$$

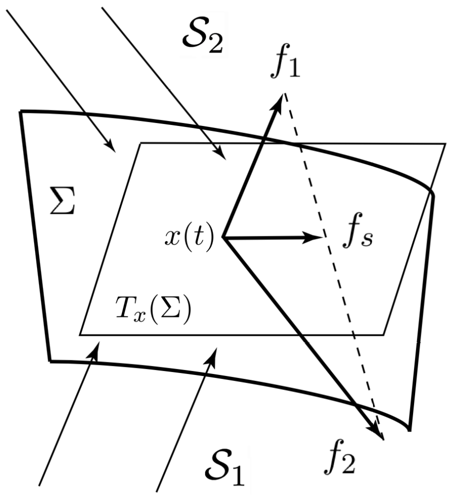


Figure 3.20: A geometric sketch of the sliding vector field.

Filippov's convex method together with the above existence theorem defines the *solution in the sense of Filippov* for the discontinuous dynamics in (3.14).

Definition 3.7 (Solution in The Sense of Filippov) *An absolute continuous function $x(t) : [0, \tau] \to \mathbb{R}^n$ is said to be a solution of $\dot{x} = f(x)$ in (3.14) in the sense of Filippov if for all $t \in [0, \tau]$ it holds that $\dot{x} \in F(x)$, where the differential inclusion $F(x)$ in (3.27) is the closed convex hull of all the limits of $f(x)$.*

Remark 3.8 If the solution $x(t)$ is in a region where the vector field is continuous, i.e. an invariant $\mathcal{S}_i$, then of course it must hold that $F(x(t)) = f_i(x(t))$. If the solution $x(t)$ slides on a hyper surface of discontinuity Σ, then $\dot{x}(t) \in \overline{co}\{f_i(x(t))\}$. The solution proposed by Filippov disregards the values (if any) of the vector fields on surfaces of Zero Lebesgue measure in the state space, and therefore one can assign any value to the vector field on Σ and this does not change the r.h.s of the differential inclusion; see [43] for more details.

3.2.2 The Case of Chattering-Zeno Between More than Two Dynamics

Obviously, when chattering-Zeno occurs on the intersection of p hyper switching surfaces, i.e. on $\bigcap_{i=1}^{p} \Sigma_i$, then one needs to find a sliding vector field f_s which keeps the solution in sliding on all the discontinuity surfaces Σ_i on which chattering occurs. That is, one needs to find p sliding coefficients α_i, $i \in \{1, \cdots, p\}$, satisfying: i) $\alpha_i \in (0, 1)$ for all i, and ii) $\sum_{i=1}^{p} \alpha_i = 1$. In this case, the construction procedure of the sliding dynamics with Filippov convexification becomes much more complicated and cumbersome when the number of the switching surfaces Σ_i increases, because non-uniqueness of solution for α_i arises for $p > 2$. In the following, we discuss this issue in more details.

3.2.2.1 The Case of Chattering-Zeno Between 2^p Dynamics with p = 2

This is the case of a hybrid automaton model of four locations, where the state space $\mathbb{R}^n$ of the system is split into four disjoint invariants $\mathcal{S}_1$, $\mathcal{S}_2$, $\mathcal{S}_3$, and $\mathcal{S}_4$ by the intersection of two hyper switching surfaces Σ_1 and Σ_2; see Figure 3.21 and Figure 3.22. The invariants $\mathcal{S}_i|_{i=1,2,3,4}$ and the hyper surfaces Σ_1 and Σ_2 are given by

$$\Sigma_1 = \Sigma_{11} \cup \Sigma_{12} = \{x \in \mathbb{R}^n : \gamma_1(x) = 0\},$$

$$\Sigma_2 = \Sigma_{21} \cup \Sigma_{22} = \{x \in \mathbb{R}^n : \gamma_2(x) = 0\},$$

$$\Sigma_{11} = \overline{\mathcal{G}(q_1,q_2)} \cap Inv(q_1) = \overline{\mathcal{G}(q_2,q_1)} \cap Inv(q_2) = \{x \in \mathbb{R}^n : \gamma_1(x) = 0 \ \wedge \ \gamma_2(x) \geq 0\},$$

$$\Sigma_{12} = \overline{\mathcal{G}(q_3,q_4)} \cap Inv(q_3) = \overline{\mathcal{G}(q_4,q_3)} \cap Inv(q_4) = \{x \in \mathbb{R}^6 : \gamma_1(x) = 0 \ \wedge \ \gamma_2(x) \leq 0\},$$

$$\Sigma_{21} = \overline{\mathcal{G}(q_1,q_4)} \cap Inv(q_1) = \overline{\mathcal{G}(q_4,q_1)} \cap Inv(q_4) = \{x \in \mathbb{R}^6 : \gamma_1(x) \geq 0 \ \wedge \ \gamma_2(x) = 0\},$$

$$\Sigma_{22} = \overline{\mathcal{G}(q_2,q_3)} \cap Inv(q_2) = \overline{\mathcal{G}(q_3,q_2)} \cap Inv(q_3) = \{x \in \mathbb{R}^6 : \gamma_1(x) \leq 0 \ \wedge \ \gamma_2(x) = 0\},$$

$$\mathcal{S}_1 = Inv(q_1) = \{x \in \mathbb{R}^n : \ \gamma_1(x) \geq 0 \ \wedge \ \gamma_2(x) \geq 0\},$$

$$\mathcal{S}_2 = Inv(q_2) = \{x \in \mathbb{R}^n : \ \gamma_1(x) \leq 0 \ \wedge \ \gamma_2(x) \geq 0\},$$

$$\mathcal{S}_3 = Inv(q_3) = \{x \in \mathbb{R}^n : \ \gamma_1(x) \leq 0 \ \wedge \ \gamma_2(x) \leq 0\},$$

$$\mathcal{S}_4 = Inv(q_4) = \{x \in \mathbb{R}^n : \ \gamma_1(x) \geq 0 \ \wedge \ \gamma_2(x) \leq 0\}.$$

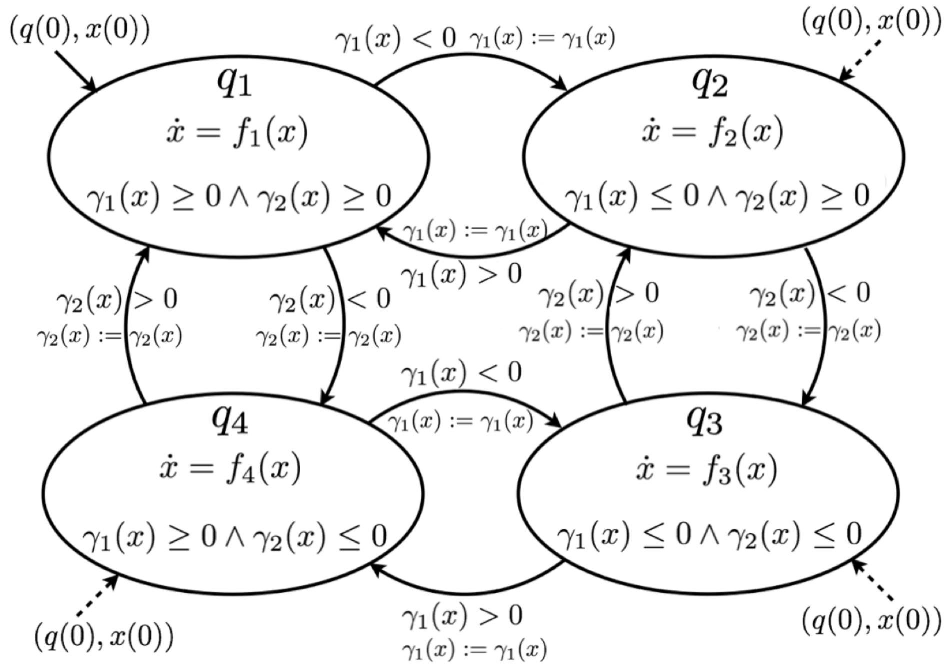
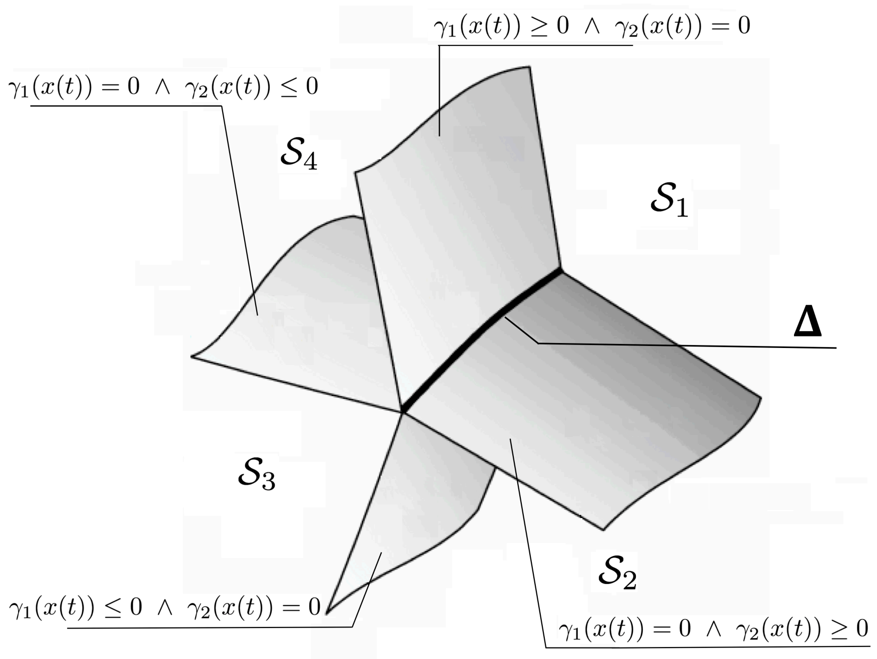

$(q(0), x(0))$
$\gamma_1(x) < 0$ $\gamma_1(x) := \gamma_1(x)$
$(q(0), x(0))$
q_1
$\dot{x} = f_1(x)$
$\gamma_1(x) \geq 0 \wedge \gamma_2(x) \geq 0$
q_2
$\dot{x} = f_2(x)$
$\gamma_1(x) \leq 0 \wedge \gamma_2(x) \geq 0$
$\gamma_1(x) := \gamma_1(x)$
$\gamma_1(x) > 0$
$\gamma_2(x) > 0$ $\gamma_2(x) := \gamma_2(x)$
$\gamma_2(x) < 0$ $\gamma_2(x) := \gamma_2(x)$
$\gamma_2(x) > 0$ $\gamma_2(x) := \gamma_2(x)$
$\gamma_2(x) < 0$ $\gamma_2(x) := \gamma_2(x)$
$\gamma_1(x) < 0$
$\gamma_1(x) := \gamma_1(x)$
q_4
$\dot{x} = f_4(x)$
$\gamma_1(x) \geq 0 \wedge \gamma_2(x) \leq 0$
q_3
$\dot{x} = f_3(x)$
$\gamma_1(x) \leq 0 \wedge \gamma_2(x) \leq 0$
$(q(0), x(0))$
$\gamma_1(x) > 0$
$\gamma_1(x) := \gamma_1(x)$
$(q(0), x(0))$

Figure 3.21: A hybrid automaton with four modes and two hyper switching surfaces.



$\gamma_1(x(t)) \geq 0 \ \wedge \ \gamma_2(x(t)) = 0$
$\gamma_1(x(t)) = 0 \ \wedge \ \gamma_2(x(t)) \leq 0$
$\mathcal{S}_4$
$\mathcal{S}_1$
Δ
$\mathcal{S}_3$
$\mathcal{S}_2$
$\gamma_1(x(t)) \leq 0 \ \wedge \ \gamma_2(x(t)) = 0$
$\gamma_1(x(t)) = 0 \ \wedge \ \gamma_2(x(t)) \geq 0$

Figure 3.22: The state space of the hybrid automaton model in Figure 3.21.

The system's dynamics in this case is given by

$$\dot{x} = f(x) = \begin{cases} f_1(x), & \text{for } x \in \mathcal{S}_1, \\ f_2(x), & \text{for } x \in \mathcal{S}_2, \\ f_3(x), & \text{for } x \in \mathcal{S}_3, \\ f_4(x), & \text{for } x \in \mathcal{S}_4. \end{cases} \tag{3.33}$$

Let's denote Σ to the entire discontinuity region in the state space, that is $\Sigma = \Sigma_1 \cup \Sigma_2$. Then we can replace the discontinuous dynamics in (3.33) by the following set-valued extension

$$\dot{x} \in F(x) = \begin{cases} f_i(x)|_{i=1,2,3,4}, & \text{for } x \in \mathcal{S}_i, \\ \overline{co}\{f_i(x^\star)|_{i=1,2,3,4}\}, & \text{for } x^\star \in \Sigma, \end{cases} \tag{3.34}$$

with

$$\overline{co}\{f_i(x^\star)|_{i=1,2,3,4}\} = \sum_{i=1}^{4} \lambda_i f_i(x^\star), \tag{3.35}$$

where $\lambda_1 = \alpha_1\alpha_2$, $\lambda_2 = (1-\alpha_1)\alpha_2$, $\lambda_3 = (1-\alpha_1)(1-\alpha_2)$, and $\lambda_4 = \alpha_1(1-\alpha_2)$. For $j = 1, 2$ we have $\alpha_j \in (0,1)$ for $x^\star \in \Sigma_j$ in case of chattering-Zeno on Σ_j.

To be consistent with Filippov convexification we restrict λ_i to satisfy $\sum_{i=1}^{4} \lambda_i = 1$, that is, the parameters α_1 and α_2 should satisfy

$$\alpha_1\alpha_2 + (1-\alpha_1)\alpha_2 + (1-\alpha_1)(1-\alpha_2) + \alpha_1(1-\alpha_2) = 1. \tag{3.36}$$

Similarly to what we stated in equations (3.28), (3.29), and (3.30), one way to compute α_1 and α_2 is by using the projections of the convex combination (3.35) on the normals n_1 and n_2 of Σ_1 and Σ_2. Roughly speaking, we suppose a chattering-Zeno scenario on the intersection $\Delta = \Sigma_1 \cap \Sigma_2$, and we try to find α_1 and α_2 that keep the solution trajectory in sliding on Δ as long as chattering-Zeno is occurring on it:

$$(1-\alpha_1)W_1^{N_1} + \alpha_1 W_2^{N_1} = 0, \tag{3.37}$$

$$(1-\alpha_2)W_3^{N_2} + \alpha_2 W_4^{N_2} = 0, \tag{3.38}$$

where

$$W_1^{N_1} = (1-\alpha_2)f_3^{N_1}(x^\star) + \alpha_2 f_2^{N_1}(x^\star), \tag{3.39}$$
$$W_2^{N_1} = (1-\alpha_2)f_4^{N_1}(x^\star) + \alpha_2 f_1^{N_1}(x^\star), \tag{3.40}$$
$$W_3^{N_2} = (1-\alpha_1)f_3^{N_2}(x^\star) + \alpha_1 f_4^{N_2}(x^\star), \tag{3.41}$$
$$W_4^{N_2} = (1-\alpha_1)f_2^{N_2}(x^\star) + \alpha_1 f_1^{N_2}(x^\star). \tag{3.42}$$

Existence of Solution: From (3.37) and (3.38) we have a fixed point problem. The solution of this problem exists. It is given by

$$\alpha_1 = \frac{(1-\alpha_2)f_3^{N_1}(x^\star) + \alpha_2 f_2^{N_1}(x^\star)}{\Big((1-\alpha_2)f_3^{N_1}(x^\star) + \alpha_2 f_2^{N_1}(x^\star)\Big) - \Big((1-\alpha_2)f_4^{N_1}(x^\star) + \alpha_2 f_1^{N_1}(x^\star)\Big)}, \tag{3.43}$$

$$\alpha_2 = \frac{(1-\alpha_1)f_3^{N_2}(x^\star) + \alpha_1 f_4^{N_2}(x^\star)}{\Big((1-\alpha_1)f_3^{N_2}(x^\star) + \alpha_1 f_4^{N_2}(x^\star)\Big) - \Big((1-\alpha_1)f_2^{N_2}(x^\star) + \alpha_1 f_1^{N_2}(x^\star)\Big)}, \tag{3.44}$$

for $W_1^{N_1} \neq W_2^{N_1}$ and $W_3^{N_2} \neq W_4^{N_2}$.

Uniqueness of Solution: By the substitution of (3.43) in (3.44) we get a quadratic equation for α_2

$$\Pi(\alpha_2) = m_2\alpha_2^2 + m_1\alpha_2 + m_0 = 0, \tag{3.45}$$

where

$$m_2 = (f_3^{N_1}f_4^{N_2} - f_4^{N_1}f_3^{N_2}) + (f_1^{N_1}f_3^{N_2} - f_3^{N_1}f_1^{N_2}) + (f_2^{N_1}f_1^{N_2} - f_1^{N_1}f_2^{N_2}) + (f_4^{N_1}f_2^{N_2} - f_2^{N_1}f_4^{N_2}),$$
$$m_1 = (f_2^{N_1}f_4^{N_2} - f_4^{N_1}f_2^{N_2}) + (f_3^{N_1}f_1^{N_2} - f_1^{N_1}f_3^{N_2}) + 2(f_4^{N_1}f_3^{N_2} - f_3^{N_1}f_4^{N_2}),$$
$$m_0 = f_3^{N_1}f_4^{N_2} - f_4^{N_1}f_3^{N_2}.$$

In addition to the constraint $\alpha_2 \in (0,1)$, the parameter α_2 should also satisfy (3.45) during sliding on the intersection $\Delta = \Sigma_1 \cap \Sigma_2$. For $\alpha_2 = 0$ we have $\Pi(0) = m_0 = (f_3^{N_1}(x^\star) \cdot f_4^{N_2}(x^\star)) - (f_4^{N_1}(x^\star) \cdot f_3^{N_2}(x^\star))$. For $\alpha_2 = 1$ we have $\Pi(1) = m_2 + m_1 + m_0 = (f_2^{N_1}(x^\star) \cdot f_1^{N_2}(x^\star)) - (f_1^{N_1}(x^\star) \cdot f_2^{N_2}(x^\star))$. As stated in Definition 3.4 (Sliding Region), when chattering-Zeno occurs on the intersection $\Delta = \Sigma_1 \cap \Sigma_2$, the dynamics $f_i(x^\star)$ in the neighborhood of Δ pushes the solution towards it. That is, the normal projections $f_i^{N_j}(x^\star)$, in our case, satisfy

$$f_1^{N_1}(x^\star) < 0, \quad f_1^{N_2}(x^\star) < 0,$$

$$f_2^{N_1}(x^\star) > 0, \quad f_2^{N_2}(x^\star) < 0,$$

$$f_3^{N_1}(x^\star) > 0, \quad f_3^{N_2}(x^\star) > 0,$$

$$f_4^{N_1}(x^\star) < 0, \quad f_4^{N_2}(x^\star) > 0.$$

It holds then that $\Pi(0) > 0$ and $\Pi(1) < 0$, and therefore there exist a solution to $\Pi(\alpha_2)$ satisfying $\alpha_2 \in (0, 1)$. Furthermore, the solution of $\Pi(\alpha_2)$ is unique since $\Pi(\alpha_2)$ is a parabola function. The same applies for α_1.

3.2.2.2 The Case of Chattering-Zeno Between 2^p Dynamics with p > 2

The situation becomes much more complicated when chattering-Zeno occurs on the intersection of $p > 2$ discontinuous surfaces.

Let's consider the most general case where we have a hybrid automaton model having $p > 2$ hyper switching surfaces $\Sigma_1, \Sigma_2, \cdots, \Sigma_p$, such that $\Sigma_j = \{x \in \mathbb{R}^n : \gamma_j(x) = 0\}$, with the dimension n satisfying $n \geq p+1$, where p is the total number of the intersected surfaces. Denote Δ as a switching intersection, i.e. $\Delta = \cap_{j=1:k}\Sigma_j$ for some index $k \neq 1$ in $\{1, \cdots, p\}$, and denote Σ as the entire discontinuity region, i.e. $\Sigma = \cup_{j=1}^{p}\Sigma_j$. In this case, Filippov differential inclusion $F(x)$ is given in a general form by

$$\dot{x} \in F(x) = \begin{cases} f_i(x)|_{i=1,\cdots,2^p}, & \text{for } x \in \mathcal{S}_i, \\ \overline{co}\{f_i(x^\star)|_{i=1,\cdots,2^p}\}, & \text{for } x^\star \in \Sigma, \end{cases} \tag{3.46}$$

where

$$\overline{co}\{f_i(x^\star)|_{i=1,\cdots,2^p}\} = \sum_{i=1}^{2^p} \lambda_i f_i(x^\star), \qquad \sum_{i=1}^{2^p} \lambda_i = 1, \tag{3.47}$$

$$\lambda_i = \prod_{j=1}^{p} \left(\frac{1 - \Psi_{j,i} + 2\Psi_{j,i}\alpha_j}{2} \right), \tag{3.48}$$

$$\forall j : \alpha_j \in (0, 1) \text{ for } x^\star \in \Sigma_j \text{ when chattering on } \Sigma_j. \tag{3.49}$$

The coefficient $\Psi_{j,i}$ is a sign parameter which gives the sign of $\gamma_j(x^\bullet)$ for all $x^\bullet \in \mathcal{S}_i$ in a small neighborhood to the discontinuity point $x^\star \in \Sigma$, that is

$$\Psi_{j,i} = \text{sgn}(\gamma_j(x^\bullet))|_{x^\bullet \in \mathcal{S}_i \to x^\star \in \Sigma_j} \tag{3.50}$$

Lemma 3.9 *In agreement with Definition 3.4, a chattering-Zeno occurs on a discontinuous surfaces intersection $\Delta = \cap_{j=1:k}\Sigma_j$ for some index $k \neq 1$ in $\{1, \cdots, p\}$ if*

$$\forall i, j : \quad sgn(f_i^{N_j}(x^\star) = -\Psi_{j,i}. \tag{3.51}$$

To keep the solution trajectory in a sliding motion on the intersection $\Delta = \cap_{j=1:k}\Sigma_j$ on which a chattering-Zeno occurs, the sliding vector field should be in the tangent plane $T_x(\Delta)$, i.e. the sliding vector field in the tangent plane $T_x(\Sigma_j)$ for all j. This, however, requires solving the following non-linear problem

$$\forall j = 1, 2, \cdots, k \leq p : \quad \sum_{i=1}^{2^k} \prod_{j=1}^{k} \left(\frac{1 - \Psi_{j,i} + 2\Psi_{j,i}\alpha_j}{2} \right) f_i^{N_j}(x^\star) = 0. \tag{3.52}$$

Similarly to what we did in (3.37) and (3.38), we can write (3.52) as

$$(1 - \alpha_j) W_1^{N_j} + \alpha_j W_2^{N_j} = 0, \tag{3.53}$$

and therefore

$$\alpha_j = \frac{W_1^{N_j}}{W_1^{N_j} - W_2^{N_j}}, \tag{3.54}$$

where

$$W_1^{N_j} = \sum_{i:\Psi_{j,i}=-1} \left[\prod_{m=1, m\neq j}^{k} \left(\frac{1 - \Psi_{m,i} + 2\Psi_{m,i}\alpha_m}{2} \right) \right] f_i^{N_j}(x^\star), \tag{3.55}$$

$$W_2^{N_j} = \sum_{i:\Psi_{j,i}=1} \left[\prod_{m=1, m\neq j}^{k} \left(\frac{1 - \Psi_{m,i} + 2\Psi_{m,i}\alpha_m}{2} \right) \right] f_i^{N_j}(x^\star). \tag{3.56}$$

Note that, the product term $\prod_{m=1,m\neq j}^{k} \left(\frac{1-\Psi_{m,i}+2\Psi_{m,i}\alpha_m}{2} \right)$ in (3.55) and (3.56) is a product of α_m and $(1 - \alpha_m)$, i.e. α_m for $\Psi_{m,i} = 1$ and $(1 - \alpha_m)$ for $\Psi_{m,i} = -1$, and therefore this product takes always a value (0,1) for $\alpha_m \in (0, 1)$. As a result, when chattering-Zeno occurs on an intersection $\Delta = \cap_{j=1:k}\Sigma_j$ it holds $\forall j = 1, 2, \cdots, k \leq p$: $W_1^{N_j} > 0 \ \wedge \ W_2^{N_j} < 0$.

A solution to the non-linear problem (3.52) exists since (3.54) maps a hypercube convex hull $(0, 1)^{k \leq p}$ strictly into itself, and thus it has a fixed point.

However, it is much more elusive to prove that the solution of (3.52) is unique. This is the main limitation of the classical Filippov convex approach for treating chattering-Zeno behavior on discontinuous surfaces intersections.

We conclude that, although Filippov convexification method is efficient to deal with chattering-Zeno, it is however limited to the case of chattering-Zeno between 2^p dynamics with $p \leq 2$, as it requires solving stiff non-linear equations for computing the sliding coefficients α_j when the number of the hyper switching surfaces on which chattering-Zeno occurs increases. Furthermore, high computational load results when solving such non-linear equations in the case in which chattering-Zeno occurs on the intersection of a large number of hyper switching surfaces; a scenario which prevents the simulation to be completed in a reasonable time.

In the following section, we provide a computational framework for an efficient treatment of chattering-Zeno in terms of detection and elimination. Our proposed computational framework is based on extending Filippov convexification to more complete convexification so that there is no need at all to deal with solving stiff non-linear equations for the computation of the sliding coefficients α_j (in case of chattering-Zeno occurring on the intersection of a large number of hyper switching surfaces).

3.3 Chattering-Free Simulation Framework

We consider the most general case where we have a hybrid automaton model $\mathcal{H} = \{Q, X, Init, f, D, E, G, R\}$ whose the state space $\mathbb{R}^n$ is split into 2^p $\mathbb{R}^n$-dimensional disjoint invariants $\mathcal{S}_i = Inv(q_i) \in D$, $q_i \in Q$, $i = 1, \cdots, 2^p$, by $p \geq 1$ hyper switching surfaces $\Sigma_j \in \mathbb{R}^{n-1}$, where in each invariant the system dynamics is defined by a vector field $f_i(x)$ extendable over a small neighborhood of the surfaces Σ_j.

To ease for future discussion, we consider each hyper switching surface Σ_j splits the state space into two disjoint invariants $\mathcal{S}_{j1}$ and $\mathcal{S}_{j2}$ such that $\mathbb{R}^n = \mathcal{S}_{j1} \cup \Sigma_j \cup \mathcal{S}_{j2}$. The subspaces $\mathcal{S}_{j1}$ and $\mathcal{S}_{j2}$ and the hyper surfaces Σ_j can be formulated for all j as

$$\mathcal{S}_{j1} = Inv(q_{j1}) = \{x \in \mathbb{R}^n : \ \gamma_j(x) \leq 0\}, \tag{3.57}$$

$$\mathcal{S}_{j2} = Inv(q_{j2}) = \{x \in \mathbb{R}^n : \ \gamma_j(x) \geq 0\}, \tag{3.58}$$

$$\Sigma_j = \overline{\mathcal{G}(q_{j1}, q_{j2})} \cap Inv(q_{j1}) = \overline{\mathcal{G}(q_{j2}, q_{j1})} \cap Inv(q_{j2}) = \{x \in \mathbb{R}^n : \ \gamma_j(x) = 0\}. \tag{3.59}$$

For each hyper switching surface Σ_j, the normal n_j perpendicular to Σ_j is given then for all $x^\star \in \Sigma_j$ by

$$n_j(x^\star) = \frac{\nabla\gamma_j(x^\star)}{||\nabla\gamma_j(x^\star)||}, \quad \nabla\gamma_j(x^\star) = \frac{\partial\gamma_j(x^\star)}{\partial x}, \tag{3.60}$$

where we assume that $\frac{\partial\gamma(x^\star)}{\partial x} \neq 0$ for all $x^\star$. Moreover, n_j are assumed to be linearly independent on any intersection $\bigcap_{j=1}^{k\leq p} \Sigma_j$.

At a discontinuity point $x^\star \in \Sigma_j$, the normal projection of the dynamics f_i in a small neighborhood to $x^\star$ is given by

$$f_i^{N_j}(x^\star) = n_j(x^\star)^T \cdot f_i(x^\bullet), \tag{3.61}$$

where $x^\bullet \in \mathcal{S}_i \to x^\star \in \Sigma_j$.

Any hybrid simulation computational framework supporting chattering-Zeno detection and avoidance should be able to detect regions on the switching manifold on which chattering-Zeno occurs and force the solution trajectory of the system to slide on the manifold in these regions. It also has to decide when a smooth exit from sliding should take place to evolve continuously in the next $\mathbb{R}^n$-dimensional invariant. Therefore, the following steps should be performed:

1. Continuous integration outside the entire discontinuity region $\Sigma = \cup_{j=1}^{p} \Sigma_j$.

2. Accurate detection and location of the discontinuity points $x^\star \in \Sigma_j$, including the intersections discontinuity points $x^\star \in \bigcap_{j=1}^{k \leq p} \Sigma_j$.

3. Check at the detected discontinuity points $x^\star$ whether $x^\star$ is a transversality point, i.e. $x^\star$ belongs to the crossing region Σ^c (Definition 3.3), or a chattering-Zeno limit point, i.e. $x^\star$ belongs to the sliding region Σ^s (Definition 3.4).

4. In case $x^\star$ is a chattering-Zeno limit point, integrate the model at $x^\star$ with equivalent sliding dynamics $f_s(x^\star)$.

5. Decision of whether the solution should stay in sliding, or exit tangentially from its sliding to evolve in an $\mathbb{R}^n$-dimensional invariant $\mathcal{S}_i$.

1. Continuous Integration Outside the Discontinuity Region:
For discretization the system dynamics $f_i(x)$ in the invariants $\mathcal{S}_i$ outside the discontinuity region Σ, any explicit/implicit integration scheme proposed by the simulator's solver can be used.

2. Handling of Discontinuities:
At the end of each continuous time integration step $[t_i, t_{i+1}]$ we check whether or not the discontinuity region has been crossed. Denote x_i to $x(t_i)$ and x_{i+1} to $x(t_{i+1})$. We distinguish between the following two cases:

1. If $\gamma_j(x_i) \cdot \gamma_j(x_{i+1}) > 0$ for all $j = 1, \cdots, p$, then we continue at x_{i+1} the continuous integration with the same vector field used in the last continuous integration step.

2. If there exist $j \in \{1, \cdots, p\}$ such that $\gamma_j(x_i) \cdot \gamma_j(x_{i+1}) < 0$, this indicates a zero-crossing of a hyper switching surface Σ_j during the current integration step. In this case we have a continuous smooth switching function $\gamma_j(x(t))$ taking opposed signs at t_i and t_{i+1}, and therefore it has a zero at $t \in (t_i, t_{i+1})$ which defines the discontinuity point $x^\star \in \Sigma_j$ at which the zero-crossing occurs, i.e. $\gamma_j(x^\star(t)) = 0$. In this case, the solver have to backtrack and use root finding in order to locate the discontinuity point $x^\star$ up to a certain precision. Note that, if $\gamma_j(x_i) \cdot \gamma_j(x_{i+1}) < 0$ and $\gamma_j(x^\star(t)) = 0$ for all $j = 1, \cdots, k \leq p$, with $p > 1$, then the discontinuity point $x^\star$ belongs to a switching intersection $\Delta = \cap_{j=1:k}\Sigma_j$ for some index $k \neq 1$ in $\{1, \cdots, p\}$.

3. Chattering-Zeno Detection:
The next step is to check whether or not the detected discontinuity point $x^\star \in \Sigma_j$ is a transversality point, i.e. leads to cross Σ_j, or a chattering-Zeno point, i.e. leads to slide on Σ_j. This should be done by analyzing the gradient of the continuous-time behavior of the system in a small neighborhood to detected discontinuity point.

- **Definition 3.10** *A discontinuity point $x^\star \in \Sigma_j$ is a transversality point if*
$$\forall j : \quad f_{j1}^{N_j}(x^\star) \cdot f_{j2}^{N_j}(x^\star) \geq 0, \tag{3.62}$$
where f_{j1} and f_{j2} are the dynamics in the disjoint invariants $\mathcal{S}_{j_1} = \{x \in \mathbb{R}^n : \gamma_j(x) \leq 0\}$ and $\mathcal{S}_{j_2} = \{x \in \mathbb{R}^n : \gamma_j(x) \geq 0\}$ in the neighborhood of $x^\star$.

- **Definition 3.11** *A discontinuity point $x^\star \in \Sigma_j$ is a chattering-Zeno limit point if*
$$\exists j : \quad f_{j1}^{N_j}(x^\star) \cdot f_{j2}^{N_j}(x^\star) < 0 \quad and \quad f_{ji}^{N_j}(x^\star) \cdot \gamma_j(x^\bullet) < 0|_{i=1,2}, \tag{3.63}$$
where $x^\bullet \in \mathcal{S}_{ji} \to x^\star \in \Sigma_j$ and f_{j1} and f_{j2} are the dynamics in the disjoint invariants $\mathcal{S}_{j_1} = \{x \in \mathbb{R}^n : \gamma_j(x) \leq 0\}$ and $\mathcal{S}_{j_2} = \{x \in \mathbb{R}^n : \gamma_j(x) \geq 0\}$ in the neighborhood of $x^\star$.

  We can re-write Definition 3.11 as following:

- **Definition 3.12** *A discontinuity point $x^\star \in \Sigma_j$ is a chattering-Zeno limit point if at $x^\star$ the normal projections of the dynamics in all the invariants $\mathcal{S}_i$ in the neighborhood of $x^\star$ are pointed towards $x^\star$, that is for all i, j it hold that $f_i^{N_j}(x^\star) \cdot \gamma_j(x^\bullet) < 0$, where $x^\bullet \in \mathcal{S}_i \to x^\star \in \Sigma$.*

Note that, Definition 3.10 and Definition 3.11 take the most general case where $x^\star$ may belong to an $\mathbb{R}^{(n-1)}$-dimensional hyper switching surface as well as to hyper switching surfaces intersections $\Delta = \cap_{j=1:k}\Sigma_j$.

4. Chattering-Zeno Elimination:

If a discontinuity point $x^\star$ is a chattering-Zeno limit point, then the model should be integrated with sliding dynamics f_s at $x^\star$. The sliding dynamics f_s is constructed at $x^\star$ as a convex combination $\overline{co}\{f_i(\cdot)\}$ of all the dynamics f_i used in the invariants $\mathcal{S}_i$ in the neighborhood of $x^\star$. During the continuous integration with the sliding dynamics f_s, a monitoring is done at the end of each integration step to decide whether or not we should continue integrating with f_s (i.e. continue sliding), or exit tangentially from sliding to evolve in one of the invariants $\mathcal{S}_i$.

1. **Defining the sliding dynamics f_s:** As the sliding dynamics $f_s(x^\star)$ is constructed as a convex combination of all the dynamics $f_i|_{i=1,\cdots,2^p}$ in the neighborhood of $x^\star$ then $f_s(x^\star)$ is disgustingly different from one discontinuity point to another, and it depends strictly on the number p of the hyper switching surfaces Σ_j to which belong $x^\star$. Formally we write f_s as
$$f_s(x^\star) = \sum_{i=1}^{2^p} \left(\frac{\kappa_i(x^\star)}{\sum_{k=1}^{2^p} \kappa_k(x^\star)} \cdot f_i(x^\star) \right), \tag{3.64}$$
where $\kappa_i(x^\star)$ are convexification coefficients, given as rational functions by
$$\kappa_i(x^\star) = \frac{\left(\prod_{l=1;l\neq i}^{2^p-1}(\Omega_l)\right)^{\frac{1}{2^p-1}}}{\left(\prod_{l=1;l\neq i}^{2^p-1}(\Omega_l)\right)^{\frac{1}{2^p-1}} - \Omega_i}, \quad \Omega_i = \sum_{j=1}^{p} \left(\Psi_{j,i} \cdot f_i^{N_j}(x^\star)\right). \tag{3.65}$$

To allow for sliding when chattering-Zeno occurs, $\Psi_{j,i}$ should satisfy

$$\Psi_{j,i} = \begin{cases} \operatorname{sgn}(f_i^{N_j}(x^\star)), & \text{for } i = 1, \\ -\operatorname{sgn}(f_i^{N_j}(x^\star)), & \text{for } i = 2, \cdots, 2^p. \end{cases} \tag{3.66}$$

The advantage of using the signs constraint in (3.66) is that it gives us always $\Omega_1 > 0$ and $\Omega_i < 0$ for all $i \in \{2, .., 2^p\}$, which yields: i) $\kappa_i(x^\star) \in (0,1)$ as long as chattering-Zeno takes place at $x^\star$, and ii) $\kappa_k(x^\star) = 0$ for all $k \neq i$ when $\kappa_i(x^\star) = 1$, which in turns allows us to detect easily when the solution exits tangentially from sliding to evolve in one of the invariants $\mathcal{S}_i$ in the neighborhood of $x^\star$.

Application to the case of chattering-Zeno between two dynamics:
Let's consider again the case of chattering-Zeno between two dynamics as presented in Section 3.2.1. In this case, the discontinuity dynamics is given by

$$f_s(x^\star) = \frac{\kappa_1(x^\star)}{\kappa_1(x^\star) + \kappa_2(x^\star)} f_1(x^\star) + \frac{\kappa_2(x^\star)}{\kappa_1(x^\star) + \kappa_2(x^\star)} f_2(x^\star), \tag{3.67}$$

$$\kappa_1(x^\star) = \frac{\Omega_2}{\Omega_2 - \Omega_1}, \qquad \kappa_2(x^\star) = \frac{\Omega_1}{\Omega_1 - \Omega_2}, \tag{3.68}$$

$$\Omega_1 = \Psi_{1,1} f_1^N(x^\star), \qquad \Psi_{1,1} = \operatorname{sgn}(f_1^N(x^\star)), \tag{3.69}$$

$$\Omega_2 = \Psi_{1,2} f_2^N(x^\star), \qquad \Psi_{1,2} = -\operatorname{sgn}(f_2^N(x^\star)). \tag{3.70}$$

With respect to what stated in (3.12), (3.13) and (3.14), a chattering-Zeno occurs on the discontinuous surface $\Sigma = \{x^\star \in \mathbb{R}^n : \gamma(x^\star) = 0\}$ at $x^\star \in \Sigma$ if $f_1^N(x^\star) > 0$ and $f_2^N(x^\star) < 0$, for which we have from (3.69) and (3.70) $\Psi_{1,1} = 1$, $\Psi_{1,2} = 1$, $\Omega_1 > 0$, $\Omega_2 < 0$, and therefore $\kappa_1(x^\star), \kappa_2(x^\star) \in (0,1)$ as long as chattering-Zeno occurs on Σ. A smooth exit from sliding on Σ to evolve in the invariant $\mathcal{S}_1$ is expected at the time instant at which the gradient the continuous time behavior according to the vector field f_1 starts to change it sign. That is, when the normal projection $f_1^N(x^\star)$ is tangential to Σ, i.e. when $f_1^N(x^\star) = 0$. In this case, we have $\Omega_1 = 0$, and therefore $\kappa_2 = 0$ and $\kappa_1 = 1$, which yields an immediate selection of the vector field f_1 in (3.67). The same applies for the smooth exit from sliding on Σ to evolve in the invariant $\mathcal{S}_2$, i.e. $f_2^N(x^\star) = 0$, $\Omega_2 = 0$, $\kappa_1(x^\star) = 0$, $\kappa_2(x^\star) = 1$.

Application to the case of chattering-Zeno between 2^p dynamics with $p = 2$:
Now let's consider the case of chattering-Zeno between 2^p dynamics with $p = 2$ presented in Section 3.2.2.1. In this case the discontinuity dynamics is given by

$$f_s(x^\star) = \frac{1}{\kappa_1(x^\star) + \kappa_2(x^\star) + \kappa_3(x^\star) + \kappa_4(x^\star)} \sum_{i=1}^{4} \left(\kappa_i(x^\star) \cdot f_i(x^\star)\right), \tag{3.71}$$

where

$$\kappa_1(x^\star) = \frac{\sqrt[3]{\Omega_2 \cdot \Omega_3 \cdot \Omega_4}}{\sqrt[3]{\Omega_2 \cdot \Omega_3 \cdot \Omega_4} - \Omega_1}, \qquad \kappa_2(x^\star) = \frac{\sqrt[3]{\Omega_1 \cdot \Omega_3 \cdot \Omega_4}}{\sqrt[3]{\Omega_1 \cdot \Omega_3 \cdot \Omega_4} - \Omega_2},$$

$$\kappa_3(x^\star) = \frac{\sqrt[3]{\Omega_1 \cdot \Omega_2 \cdot \Omega_4}}{\sqrt[3]{\Omega_1 \cdot \Omega_2 \cdot \Omega_4} - \Omega_3}, \qquad \kappa_4(x^\star) = \frac{\sqrt[3]{\Omega_1 \cdot \Omega_2 \cdot \Omega_3}}{\sqrt[3]{\Omega_1 \cdot \Omega_2 \cdot \Omega_3} - \Omega_4},$$

$$\begin{aligned}
\Omega_1 &= \Psi_{1,1} f_1^{N_1}(x^\star) + \Psi_{2,1} f_1^{N_2}(x^\star), & \forall j : \Psi_{j,1} &= \operatorname{sgn}(f_1^{N_j}(x^\star)),\\
\Omega_2 &= \Psi_{1,2} f_2^{N_1}(x^\star) + \Psi_{2,2} f_2^{N_2}(x^\star), & \forall j : \Psi_{j,2} &= -\operatorname{sgn}(f_2^{N_j}(x^\star)),\\
\Omega_3 &= \Psi_{1,3} f_3^{N_1}(x^\star) + \Psi_{2,3} f_3^{N_2}(x^\star), & \forall j : \Psi_{j,3} &= -\operatorname{sgn}(f_3^{N_j}(x^\star)),\\
\Omega_4 &= \Psi_{1,4} f_4^{N_1}(x^\star) + \Psi_{2,4} f_4^{N_2}(x^\star), & \forall j : \Psi_{j,4} &= -\operatorname{sgn}(f_4^{N_j}(x^\star)).
\end{aligned}$$

Consider the scenario where chattering-Zeno occurs on the intersection $\Delta = \Sigma_1 \cap \Sigma_2$. We know now that a chattering-Zeno occurs on Δ if Δ is nodal attractive:

$$\begin{aligned}
f_1^{N_1}(x^\star) &< 0, & f_1^{N_2}(x^\star) &< 0,\\
f_2^{N_1}(x^\star) &> 0, & f_2^{N_2}(x^\star) &< 0,\\
f_3^{N_1}(x^\star) &> 0, & f_3^{N_2}(x^\star) &> 0,\\
f_4^{N_1}(x^\star) &< 0, & f_4^{N_2}(x^\star) &> 0,
\end{aligned}$$

so we have $\Omega_1 > 0$, $\Omega_2 < 0$, $\Omega_3 < 0$, and $\Omega_4 < 0$ as long as chattering-Zeno occurs on Δ, and therefore, a smooth sliding on Δ is guaranteed with $\kappa_i(x^\star) \in (0, 1)$, $i = 1, 2, 3, 4$. A smooth exit from sliding on Δ to evolve in the invariant $\mathcal{S}_i|_{i=1,2,3,4}$ is expected at the time instant at which the normal projections of f_i normal onto Δ start to change their sign, i.e. a smooth exit from sliding on Δ to evolve in $\mathcal{S}_1$ is expected when $f_1^{N_j}(x^\star) = 0$ for $j = 1, 2$, so that $\Omega_1 = 0$ and therefore $\kappa_1(x^\star) = 1$ and $\kappa_k(x^\star) = 0$ for all $k \neq 1$.

2. **Sliding/Smooth Exit from Sliding:** For discretization the sliding dynamics (3.64), any explicit/implicit integration scheme proposed by the simulator's solver can be used. However, at the end of each sliding integration step $[t_m, t_{m+1}]$ an immediate decision should be taken (depending on the values of $\kappa_i(x^\star)$) on whether or not we should continue integrating with f_s, i.e. keep sliding, or exit from sliding tangentially. We distinguish between the following two cases:
   (a) If $\kappa_i(x^\star(t_{m+1})) \in (0, 1)$ for all i, then we continue at $x^\star(t_{m+1})$ integrating with the same sliding dynamics used at $x^\star(t_m)$.
   (b) If there exists i such that $\kappa_i(x^\star(t_{m+1})) = 1$, this indicates that the sufficient condition for chattering-Zeno is no more satisfied after the current sliding step. In this case, the solver have to backtrack and use root finding in order to locate, up to a certain precision, the state $x^\star(t_\sigma)|_{t_m < t_\sigma \le t_{m+1}}$ at which the change from $\kappa_i(x^\star(t_m)) \in (0, 1)$ to $\kappa_i(x^\star(t_{m+1})) = 1$ took place, so that a smooth exit from sliding emerges at $x^\star(t_\sigma)$ to evolve in the invariant $\mathcal{S}_i$. Indeed, the exit from sliding to evolve in $\mathcal{S}_i$ is tangential since in this case we have $f_i^{N_j}(x^\star(t_\sigma)) = 0$ for all $j \in \{j : \gamma_j(x^\star(t_\sigma)) = 0\}$.

As we have raised in the previous section, applying Filippov's convexification method in case of chattering-Zeno on switching intersections requires solving stiff nonlinear equations for computing the chattering-free coefficients. This is computationally time-consuming and does not guarantee a unique solution for sliding if the solution chatters on the intersection of more than two hyper switching surfaces surfaces. It is clear to notice that, in contrary with the classical Filippov approach, the advantages of the convexification in (3.64) is that it does not require solving stiff nonlinear equations for computing the sliding coefficients κ_i.

Remark 3.13 *Denoted by x_s, a standard sliding solution in $\mathbb{R}$ is the standardization of a non-standard chattering-Zeno solution ${}^*x_{ch}$ in ${}^*\mathbb{R}$*, namely, $x_s = st({}^*x_{ch})$. In the following, we clarify this issue.

Notation 3.14 Denote x_s to the sliding mode solution of the hybrid system in a standard time domain, and ${}^*x_{ch}$ to the chattering-Zeno solution of the hybrid systems in a non-standard time domain $\mathbb{T}_\partial$.

We consider the most general case where we have a hybrid automaton model whose the state space $\mathbb{R}^n$ is split into 2^p $\mathbb{R}^n$-dimensional disjoint invariants $\mathcal{S}_i = Inv(q_i) \in D$, $q_i \in Q$, $i = 1, \cdots, 2^p$, by $p \geq 1$ hyper switching surfaces $\Sigma_j = \{x \in \mathbb{R}^n : \ \gamma_j(x) = 0\}$, where in each invariant the system dynamics is defined by a vector field $f_i(x)$ Lipschitz continuous on the invariant $\mathcal{S}_i$.

In each invariant $\mathcal{S}_i$, the evolution of the solution in the non-standard time domain $\mathbb{T}_\partial$ is given, for $n \in {}^*\mathbb{N}$, $m \in \mathbb{N}$, and $\partial \approx 0$, $\partial \in {}^*\mathbb{R}$, by

$$ {}^*x_{n+m} = {}^*x_n + m\partial {}^*f_i({}^*x_n)) \tag{3.72} $$

The evolution of a chattering-Zeno execution in a non-standard time domain is typically characterized by a repeated switching back and forth between modes in a discrete fashion with infinitesimal time ∂ spent in between the repeated mode switchings. Roughly speaking, when in either of the adjacent invariants, on their common hyper switching surface, an infinitesimal step ∂ causes a mode change and the solution enters a new invariant. In the new invariant the gradient directs behavior to the previous invariant, and after another infinitesimal step ∂ a change to the previous invariant occurs.

Mathematically, a non-standard chattering-Zeno evolution around a switching surface (or an intersection of many switching surfaces) is described by

$$ {}^*x_{n+m} = {}^*x_n + m\partial \sum_{i=1}^{2^p} ({}^*f_i({}^*x_n) \cdot M_i), \tag{3.73} $$

where 2^p is the total number of the invariants $\mathcal{S}_i$ in the neighborhood of the switching surface/intersection on which chattering-Zeno occurs, and M_i is the probability for ${}^*f_i$ to be used in the current infinitesimal micro step. M_i is given by

$$ M_i = \frac{1}{m} \sum_{l=0}^{k<m} w_{il}, \quad \text{with} \quad w_{il} = \begin{cases} 1, & \text{if } {}^*x_{n+l} \in \mathcal{S}_i, \\ 0, & \text{if } {}^*x_{n+l} \notin \mathcal{S}_i. \end{cases} \tag{3.74} $$

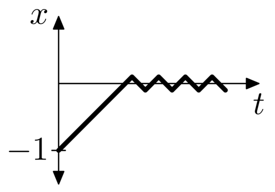


Figure 3.23: Illustration of a chattering-Zeno trajectory.

Consider again the simplest example of chattering-Zeno behavior (Figure 3.23), where the system dynamics is given by

$$\dot{x}(t) = \begin{cases} 1 & \text{for } x(t) < 0, \\ -1 & \text{for } x(t) \geq 0, \quad \text{with } x(0) < 0 \end{cases} \tag{3.75}$$

In $\mathbb{T}_\partial$, the non-standard chattering-Zeno evolution of this example behaves as follows:

- At $t = n\partial$ the solution becomes infinitesimally close to the switching surface $x = 0$, so we have $-\partial \leq {}^*x_n < 0$.
- At $t = (n+1)\partial$ we have $0 \leq {}^*x_{n+1} = {}^*x_n + \partial < \partial$.
- At $t = (n+2)\partial$ we have $-\partial \leq {}^*x_{n+2} = {}^*x_n < 0$.
- At $t = (n+3)\partial$ we have $0 \leq {}^*x_{n+3} = {}^*x_n + \partial < \partial$.
- At $t = (n+4)\partial$ we have $-\partial \leq {}^*x_{n+4} = {}^*x_n < 0$.

So in general, by applying (3.73), we get ${}^*x_{n+4} = {}^*x_n + 4\partial(\frac{1}{2} - \frac{1}{2}) = {}^*x_n$, ${}^*x_{n+5} = {}^*x_n + 5\partial(\frac{3}{5} - \frac{2}{5}) = {}^*x_n + \partial = {}^*x_{n+1}$, ${}^*x_{n+6} = {}^*x_n$, ${}^*x_{n+7} = {}^*x_{n+1}$, and so on.

In this context, it holds that a chattering-Zeno with ∂ micro steps takes place on a switching surface Σ_j as long as M_i in (3.73) is bounded $|M_i| < 1$ for all i, i.e. the dynamics in each invariant $\mathcal{S}_i$, in the neighborhood of the switching surface Σ_j, directs behavior towards Σ_j.

It is assumed that all the switching functions $\gamma_j(x)$ — which define the switching surfaces $\Sigma_j = \{x \in \mathbb{R}^n : \ \gamma_j(x) = 0\}$ — be continuous and differentiable on x. From a non-standard sense, a sliding evolution from ${}^*x_n$ to ${}^*x_{n+m}$ on any switching surface Σ_j implies that the value of ${}^*\gamma_j({}^*x_{n+k})$ should satisfy

$$-\partial \leq {}^*\gamma_j({}^*x_{n+k}) \approx {}^*\gamma_j({}^*x_n) \leq \partial, \tag{3.76}$$

for all k such that $n < k \leq m$. Note that, in the case in which the chattering-Zeno takes place on an intersection of many switching surfaces Σ_j, (3.76) should be satisfied for all ${}^*\gamma_j({}^*x_{n+k})$. Equation (3.76) yields

$$-\partial \leq {}^*\gamma_j({}^*x_n + k\partial \sum_{i=1}^{2^p} {}^*f_i({}^*x_n) \cdot M_i) \leq \partial, \tag{3.77}$$

$$-\partial \leq {}^*\gamma_j({}^*x_n) + k\partial(\sum_{i=1}^{2^p} {}^*f_i^{N_j}({}^*x_n) \cdot M_i) \leq \partial, \tag{3.78}$$

for all k such that $n < k \leq m$, where again ${}^*f_i^{N_j}({}^*x_n)$ is the normal projection at ${}^*x_n$ of ${}^*\gamma_j$ in the direction of ${}^*f_i$, given by

$$ {}^*f_i^{N_j}({}^*x_n) = \frac{\partial^*\gamma_j({}^*x_n)}{\partial x} \cdot {}^*f_i({}^*x_n). \tag{3.79}$$

Therefore, a non-standard sliding motion emerges on Σ_j and satisfies $-\partial \leq {}^*\gamma_j({}^*x_{n+k}) \approx {}^*\gamma_j({}^*x_n) \leq \partial$, if and only if

$$\sum_{i=1}^{2^p} {}^*f_i^{N_j}({}^*x_n) \cdot M_i = 0. \tag{3.80}$$

Equation (3.80) is equivalent, in the standard domain, to the constraint on the sliding vector field f_s to be tangential to the switching surface/intersection on which chattering-Zeno occurs; see (3.64), (3.65), and (3.66).

We conclude that, any sliding solution in the standard domain $\mathbb{R}^n$ is the standardization of a non-standard chattering-Zeno solution in the non-standard domain ${}^*\mathbb{R}^n$.

3.4 Prototype Implementations and Simulation Results

A part of this dissertation was attributed to design, test, and validate hybrid simulator prototypical implementations supporting chattering-Zeno freeness. The motivation behind these implementations are the followings:

1. The first motivation was to validate our proposed computational framework for chattering-Zeno freeness in a standard semantic model of time. The emphasis of validation is one of correctness and preciseness.
2. The second motivation is to provide — via these implementations — a guidance for the development of a chattering-free version for hybrid systems simulation tools which use standard model of time, giving a computational framework for an ideal manipulation of chattering-Zeno.

As simulation environments, we have used the *Functional Mock-up Interface (FMI) v2.0 for Model Exchange*, and *Acumen* Simulation tool. In both implementations, chattering-Zeno is treated internally "on the fly" by the implemented chattering-free simulator, without any need neither to a manual manipulation by the modeler (including adding a small hysteresis around the hyper switching surfaces on which chattering-Zeno occurs), nor to solve stiff nonlinear equations for the computation of the equivalent sliding dynamics in case of chattering-Zeno on switching surfaces intersections. An automatic inspection for chattering-Zeno is performed at each detected discontinuity point (i.e. state event). Whenever chattering-Zeno is detected, the simulator internally generates the sliding dynamics and switches to integrate the simulated model with the sliding dynamics in the time intervals in which the chattering-Zeno is detected. An inspection for a smooth exit from sliding is also performed by the simulator in each sliding time integration step.

3.4.1 Prototype Implementation of Chattering-Zeno Freeness in FMI

The Functional Mock-up Interface (FMI)[1] is an open standard for model exchange and co-simulation between multiple software systems. This new standard, resulting from the ITEA2 project MODELISAR in 2010, is a response to the industrial need to connect different environments for modeling, simulation and control system design.

The goal of the FMI is to describe input/output blocks of dynamic systems defined by differential, algebraic and discrete equations and to provide an interface to evaluate these equations as needed in different simulation environments, as well as in embedded control systems, with explicit or implicit integrators and fixed or variable step-size. Some details of the type of systems that can be handled are shown in Figure 3.24.

Formally, FMI is used to create an instance of a model which can be loaded into any simulator providing an import function for FMI. A software instance compatible to the FMI is called a Functional Mock-up Unit (FMU). An FMU is distributed as a compressed archive with a .fmu file extension. It contains a concrete mathematical model described by differential, algebraic and discrete equations with possible events of a dynamic physical system. An FMU consists basically of two parts:

- an XML format for model interface information,
- C API model interface functions according to the FMI specification, for model execution.

The XML format is specified by an XML schema conforming to the FMI specification. It contains all static information about model variables, including names, units and types, as well as model meta data. The C API, on the other hand, contains C functions for data management, as setting and retrieving parameter values, and evaluation of the model equations. The implementation of the C API may be provided either in C source code format or in binary forms, e.g. in the form of Windows dynamic link library .dll or a Linux shared object library .so files, to protect the model developer's intellectual property. Additional parts can be added and compressed into the FMU such as the documentation and the icon of the model. FMUs can be written manually or can be generated automatically from a modeling and simulation environment. More information about the FMI standard can be found in [10]. In the current specifications version of the FMI standard, chattering-Zeno is treated by adding a small hysteresis of size ϵ to the event functions in the FMU level. That is, an FMU — which represents the model — should add a small hysteresis to the event functions to exclude chattering-Zeno scenarios [10]. In Section 1.3 we have raised the disadvantages of this approach. The main motivation of this implementation is then to provide a correct simulation for any chattering-Zeno model represented as an FMU, which may be either a generic FMU developed manually by a modeler, or an exported FMU generated automatically by a modeling and simulation environment in which chattering-Zeno models can not be simulated correctly, whenever the compliance with FMI specification for model exchange is fulfilled. In the following section, we describe the functionality of the implemented prototypical implementation for chattering-Zeno freeness in FMI.

[1]https://www.fmi-standard.org/

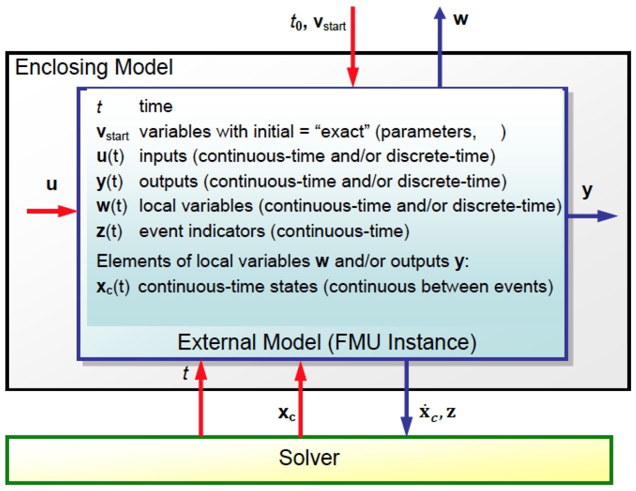

t_0, $\mathbf{v}_{start}$
w
Enclosing Model
t time
$\mathbf{v}_{start}$ variables with initial = "exact" (parameters,)
u(t) inputs (continuous-time and/or discrete-time)
y(t) outputs (continuous-time and/or discrete-time)
w(t) local variables (continuous-time and/or discrete-time)
z(t) event indicators (continuous-time)
Elements of local variables w and/or outputs y:
$\mathbf{x}_c$(t) continuous-time states (continuous between events)
External Model (FMU Instance)
u
y
t
$\mathbf{x}_c$
$\dot{\mathbf{x}}_c$, z
Solver

Figure 3.24: Data flow between the simulation environment and an FMU.

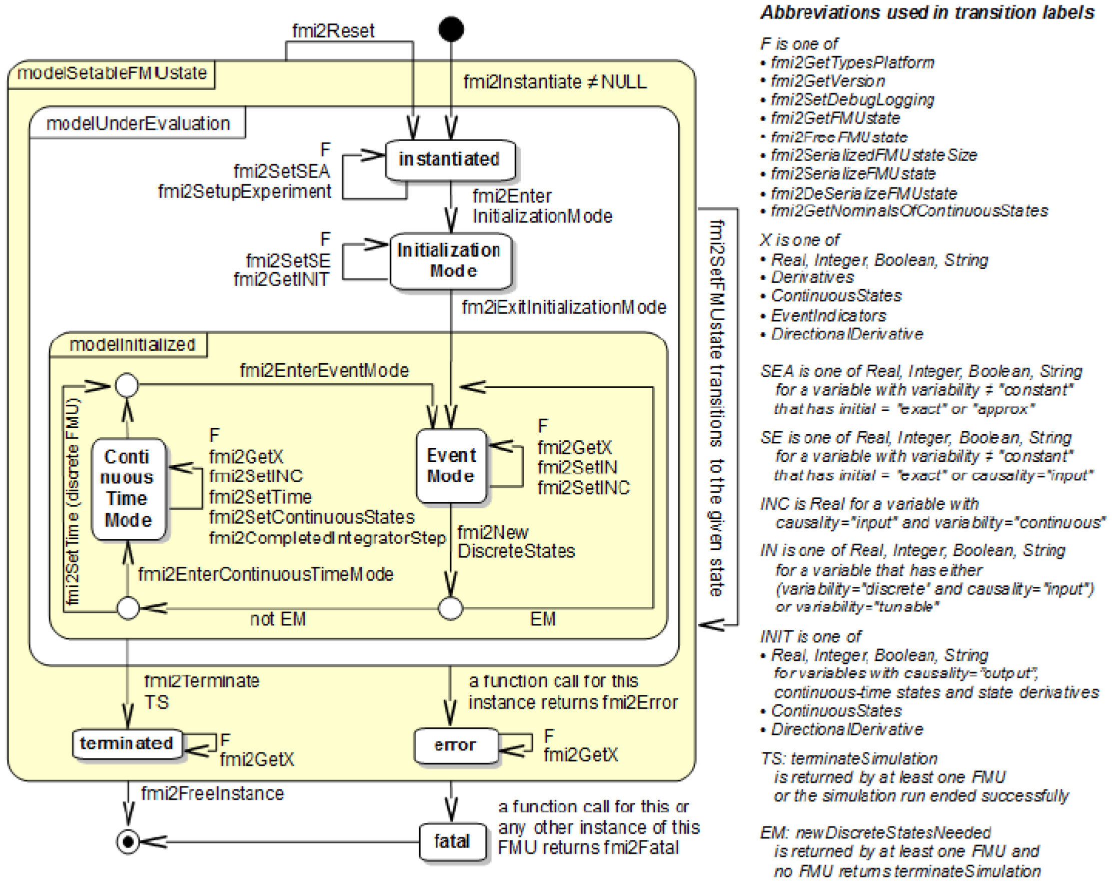
3.4.1.1 Chattering-Zeno Freeness Support for FMI Standard

Prior to a simulation experiment, the model has to be instantiated. This includes extracting the files in the FMU, loading the DLL and XML files and calling the instantiation function available in the DLL. A model can be instantiated multiple times for which the function `fmi2SetupExperiment` is provided.

Simulating an FMI model means to split the solution computation process in three different phases, categorized according to three modes: *Initialization Mode*, *Continuous-Time Mode*, and *Event Mode* (Figure 3.25).

In the *Initialization Mode*, the model is initialized by calling the FMI function `fmi2EnterInitializationMode` in order to compute the continuous-time states and the output variables at the initial time t_0. There are FMI functions used in this Mode such as `fmi2GetContinuousStates`, as well as functions of the form `fmi2`(`Get`/`Set`)(type) for setting and getting values of type `Real`, `Integer`, `String`, and `Boolean` values. The input arguments to the Initialization Mode functions consist of the all variables that are declared with "input" and "independent" causality in the FMU XML files, as well as all variables that have a start value with ($initial = exact$). Once the model is instantiated and initialized it can be simulated.

The main simulation loop starts once the FMI function `fmi2ExitInitializationMode` is called. The simulation is performed by calculating the derivatives and updating time and states in the model via the functions `fmi2SetContinuousStates`, `fmi2SetTime`, `fmi2GetContinuousStates`, `fmi2GetDerivatives`, as well as the four `fmi2`(`Get`/`Set`)(type) functions. To retrieve or set variable data during a simulation, value-references are used as keys. All variables are connected to a unique number defined and provided in the FMU XML-file. This number can then be used to retrieve information about variables via functions in the interface or can be used to set input values during a simulation.



Abbreviations used in transition labels

F is one of
- *fmi2GetTypesPlatform*
- *fmi2GetVersion*
- *fmi2SetDebugLogging*
- *fmi2GetFMUstate*
- *fmi2FreeFMUstate*
- *fmi2SerializedFMUstateSize*
- *fmi2SerializeFMUstate*
- *fmi2DeSerializeFMUstate*
- *fmi2GetNominalsOfContinuousStates*

X is one of
- *Real, Integer, Boolean, String*
- *Derivatives*
- *ContinuousStates*
- *EventIndicators*
- *DirectionalDerivative*

SEA is one of Real, Integer, Boolean, String for a variable with variability ≠ "constant" that has initial = "exact" or "approx"

SE is one of Real, Integer, Boolean, String for a variable with variability ≠ "constant" that has initial = "exact" or causality="input"

INC is Real for a variable with causality="input" and variability="continuous"

IN is one of Real, Integer, Boolean, String for a variable that has either (variability="discrete" and causality="input") or variability="tunable"

INIT is one of
- *Real, Integer, Boolean, String for variables with causality="output", continuous-time states and state derivatives*
- *ContinuousStates*
- *DirectionalDerivative*

TS: terminateSimulation is returned by at least one FMU or the simulation run ended successfully

EM: newDiscreteStatesNeeded is returned by at least one FMU and no FMU returns terminateSimulation

Figure 3.25: Calling sequence of Model Exchange C functions in form of an UML 2.0 state machine.

Discrete events are monitored via the FMI functions `fmi2GetEventIndicators` and `fmi2CompletedIntegratorStep`. Events are always triggered from the environment in which the FMU is called, so they are not triggered inside the FMU. Step events are checked after calling the function `fmi2CompletedIntegratorStep` when an integration step was successfully completed. A step event occurs if indicated by the return argument ($nextMode = EventMode$).

For capturing state events during continuous integration, the master algorithm of the FMI monitors, at every completed integrator step, the set of event indicator functions $z_j(x) = \gamma_j(x)$ provided in the function `fmi2GetEventIndicators`. All event indicators $z_j(x)$ are piecewise continuous and are collected together in one vector of real numbers. A state event occurs when one of the functions changes its sign. In this case, the simulation environment informs the FMU by calling the function `fmi2NewDiscreteStates`. Denote $x_i = x(t_i)$ and $x_{i+1} = x(t_{i+1})$. Therefore, during the continuous integration, we distinguish the following cases:

1. If $z_j(x_i) \cdot z_j(x_{i+1}) > 0$ for all $j = 1, ..., p$, where p is the total number of the event functions, then we continue the integration with the same dynamics used at x_i.
2. If there exist $j \in \{1, \cdots, p\}$ such that $z_j(x_i) \cdot z_j(x_{i+1}) < 0$, this indicates a zero-crossing during the current integration step. In this case we have a continuous smooth switching function $z_j(x_{i+1})$ taking opposed signs at t_i and t_{i+1}, and therefore it has a zero, which defines the discontinuity point (state event) $x^\star \in \Sigma_j = \{x \in \mathbb{R}^n | \ z_j(x) = 0\}$ at which the zero-crossing occurs. In this case, the solver have to backtrack and use root finding in order to locate the discontinuity point $x^\star$ up to a certain precision. If $z_j(x_i) \cdot z_j(x_{i+1}) < 0$ and $z_j(x_{i+1}(\sigma^\star)) = 0$ for all $j = 1, \cdots, k \leq p$, with $p > 1$, then the discontinuity point $x^\star$ belongs to a switching intersection $\Delta = \cap_{j=1:k}\Sigma_j$.

At a discontinuity point $x^\star$, the function `fmi2NewDiscreteStates` has to be called. This function updates and re-initializes the model in order for the simulation to be continued. Information is also given about if the states have changed values, if new state variables have been selected, and also information about upcoming time events. The computation of the chattering-free solution is embedded in the function `fmi2NewDiscreteStates`, and is split in two phases: i) chattering-Zeno detection, and ii) chattering-Zeno elimination.

The chattering-Zeno detection phase starts once a discontinuity point $x^\star$ is detected and located. The algorithm inspects whether the $x^\star$ is a chattering-Zeno point or not. This implies analyzing the gradients of the continuous time behavior in the neighborhood of $x^\star$ (Definition 3.10 and Definition 3.11). For doing so, the the normal projections at $x^\star$ of all dynamics in the neighborhood of $x^\star$, are computed and evaluated.

In the chattering elimination phase, the set-valued convexification in equations (3.64) to (3.66) is employed in order to compute the smooth equivalent sliding dynamics internally, given the dynamics $f_i(x)$ in the neighborhood of $x^\star$, and the set of event functions $z_j(x) = \gamma_j(x)$. A monitoring for a smooth exit from sliding is performed while integrating with the sliding dynamics. This is done by monitoring the value of the sliding coefficient κ_i at the end of each sliding integration step $[t_i, t_{i+1}]$ in which the FMU is integrated with the sliding dynamics.

Finally, once the solution is at the final time of a simulation, the function `fmi2Terminate` is called to terminate the simulation. After a simulation is terminated, the memory has to be deallocated. The function `fmi2FreeInstance` is then called to deallocate all the memory that have been allocated since the initialization.

3.4.1.2 Simulation Results

Figure 3.26 and Figure 3.27 show the chattering-free simulation of the Stick-Slip system in Example 3.4. The model was simulated with a time step of size 0.0012, and with the data set $x_0 = [0.8295 \ 0.8491 \ 0.3725 \ 0.5932 \ 0.8726 \ 0.9335]^T$, $m = M_1 = M_2 = 1[kg]$, $k = 0.88[N \cdot m^{-1}]$, $F_{c_1} = 0.01996[N]$, and $F_{c_2} = 0.062[N]$. The force u was modeled as a sine wave of frequency of $\omega = 0.073[rad/sec]$.

In this simulation, the sliding bifurcations depend on the effect of the external force u and the levels of Coulomb friction F_{c_1} and F_{c_2}.

At the beginning of the simulation, the solution starts in the invariant $\mathcal{S}_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \leq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \leq 0\}$. The two masses M_1 and M_2 slip then on the small mass m as long as the solution evolves continuously in the invariant $\mathcal{S}_3$.

At the time instant $t = 32.69$, the two masses m and M_2 stick together and the solution starts to slide on the hyper switching surface $\Sigma_2 = \{x \in \mathbb{R}^6 : \ v_{r_2} = v_m - v_{M_2} = 0\}$ ensuring a chattering-Zeno path avoidance on Σ_2 between the two invariants $\mathcal{S}_2 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \leq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \geq 0\}$ and $\mathcal{S}_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \leq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \leq 0\}$.

A smooth exit from sliding on Σ_2 to evolve again into the invariant $\mathcal{S}_3$ was detected at the time instant $t = 77.23$. A transversality switching (crossing) from the invariant $\mathcal{S}_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \leq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \leq 0\}$ to the invariant $\mathcal{S}_1 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \geq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \geq 0\}$ passing by the intersection $\Delta = \Sigma_1 \cap \Sigma_2 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_{r_2} = 0\}$ was detected at $t = 92.04$.

At $t = 108$, the two masses m and M_1 stick together and the solution start to slide on the hyper switching surface $\Sigma_1 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} = 0\}$ replacing a chattering on Σ_1 between the two invariants $\mathcal{S}_3 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \leq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \leq 0\}$ and $\mathcal{S}_4 = \{x \in \mathbb{R}^6 : \ v_{r_1} = v_m - v_{M_1} \geq 0 \ \wedge \ v_{r_2} = v_m - v_{M_2} \leq 0\}$. During a simulation time of 120 seconds, 5 mode switches have been detected.

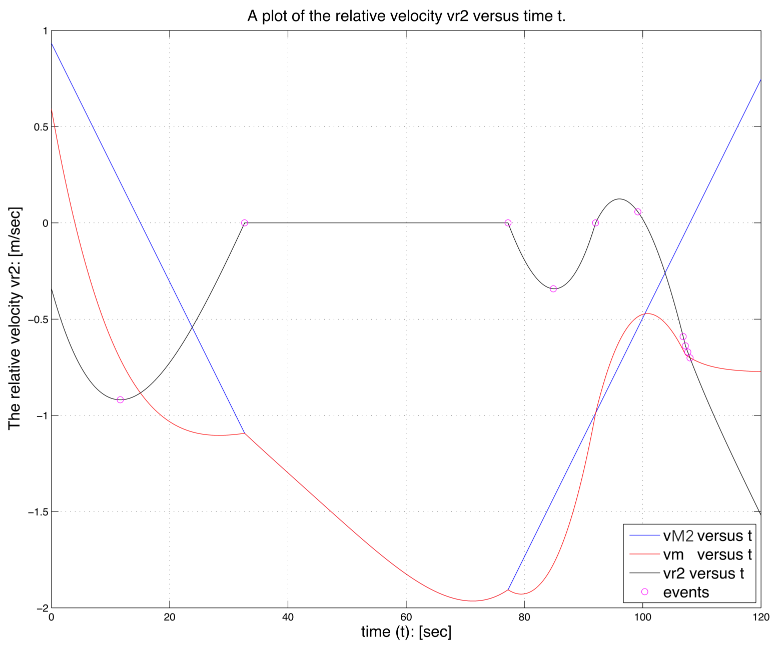


Figure 3.26: Chattering-free simulation of the Stick-Slip system of three blocks in Example 3.4: The time evolution of the relative velocity $v_{r_2} = v_m - v_{M_2}$.

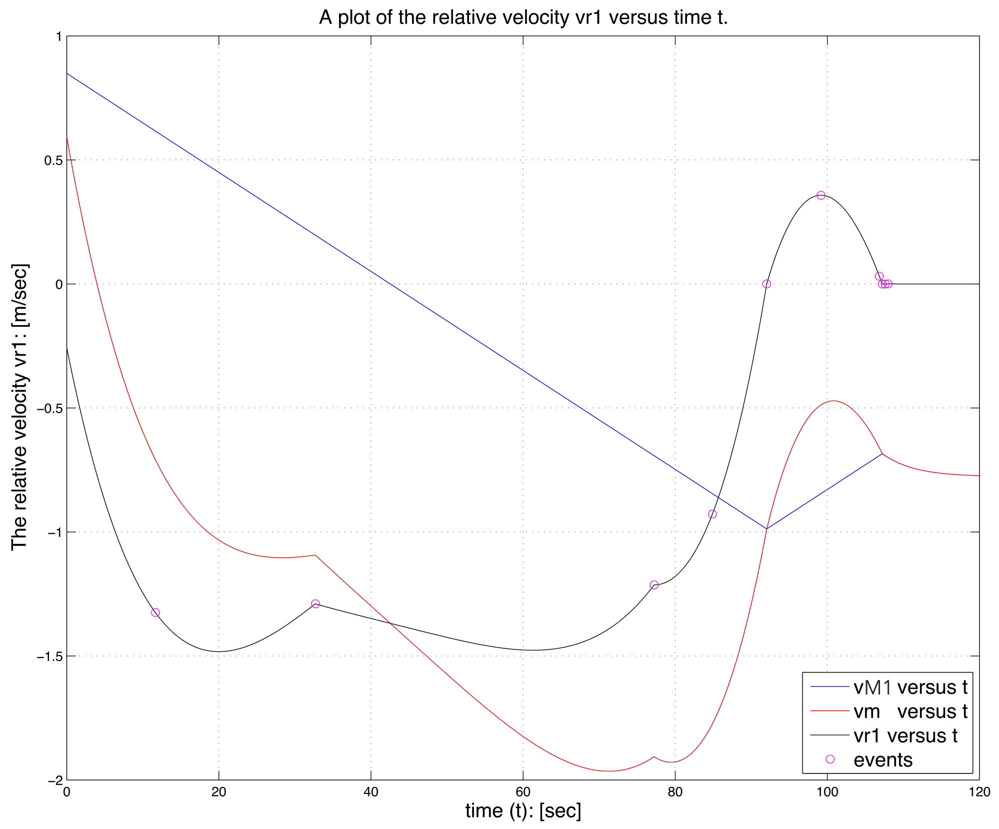


Figure 3.27: Chattering-free simulation of the Stick-Slip system of three blocks in Example 3.4: The time evolution of the relative velocity $v_{r_1} = v_m - v_{M_1}$.

Figure 3.28 and Figure 3.29 show the chattering-free simulation of the Spring-Block chain system in Example 3.5.

The system was simulated with a time step of size 0.0012, and with the data set $x_0 = [4.7799\ 0.2797\ 4.0038\ 1.7144\ 1.2922\ 4.1263]^T$, $m_1 = m_2 = m_3 = 1[kg]$, $F_{c_1} = 0.14[N]$, $F_{c_2} = 0.13[N]$, $F_{c_3} = 0.12[N]$, $k_1 = k_2 = k_3 = k_{12} = k_{13} = k_{23} = 0.01[N \cdot m^{-1}]$, and $v_d = 0.5[m/sec]$.

During a simulation time of 100 seconds, four chattering-Zeno windows were detected in the time intervals $\mathbb{I}_1 = [14.83 \cdot \cdot 15.55]$, $\mathbb{I}_2 = [26.83 \cdot \cdot 31.63]$, $\mathbb{I}_3 = [74.84 \cdot \cdot 89.07]$, and $\mathbb{I}_4 = [37.19 \cdot \cdot 100.00]$.

The sliding motion takes place on the hyper switching surface $\Sigma_1 = \{x \in \mathbb{R}^6 : v_{r_1} = v_{m_1} - v_d = 0\}$ in the intervals $\mathbb{I}_1$ and $\mathbb{I}_3$, and on the hyper switching surface $\Sigma_2 = \{x \in \mathbb{R}^6 :\ v_{r_2} = v_{m_2} - v_d = 0\}$ in the intervals $\mathbb{I}_2$ and $\mathbb{I}_4$.

At $t = 76.69$ the solution passes through the intersection $\Delta = \Sigma_1 \cap \Sigma_2 \cap \Sigma_3 = \{x \in \mathbb{R}^6 : v_{m_1} = v_{m_2} = v_{m_3} = v_d\}$ switching from a sliding on the intersection $\Sigma_1 \cap \Sigma_2$ in the positive direction of the discontinuous surface $\Sigma_3 = \{x \in \mathbb{R}^6 :\ v_{r_3} = v_{m_3} - v_d = 0\}$ to a sliding on $\Sigma_1 \cap \Sigma_2$ in the negative direction of Σ_3.

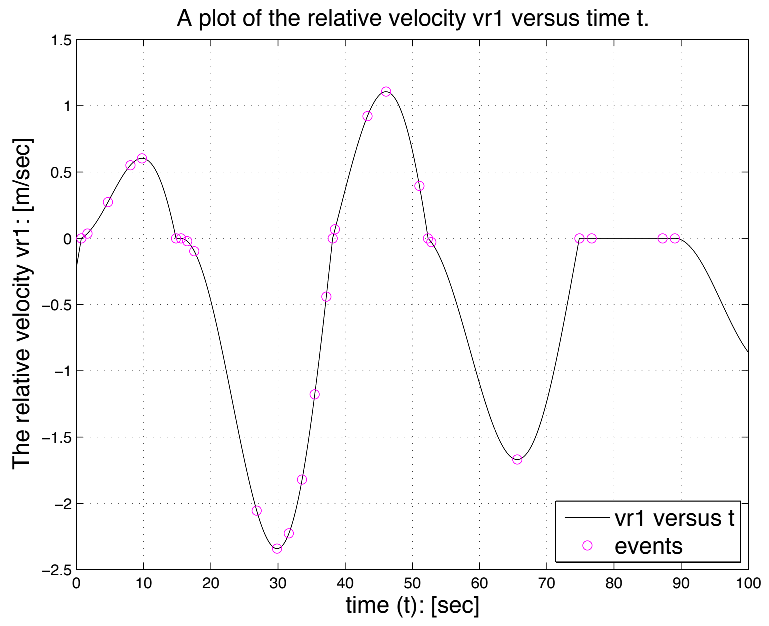


Figure 3.28: Chattering-free simulation of the Spring-Block chain system in Example 3.5: The time evolution of the relative velocity $v_{r_1} = v_{m_1} - v_d$.

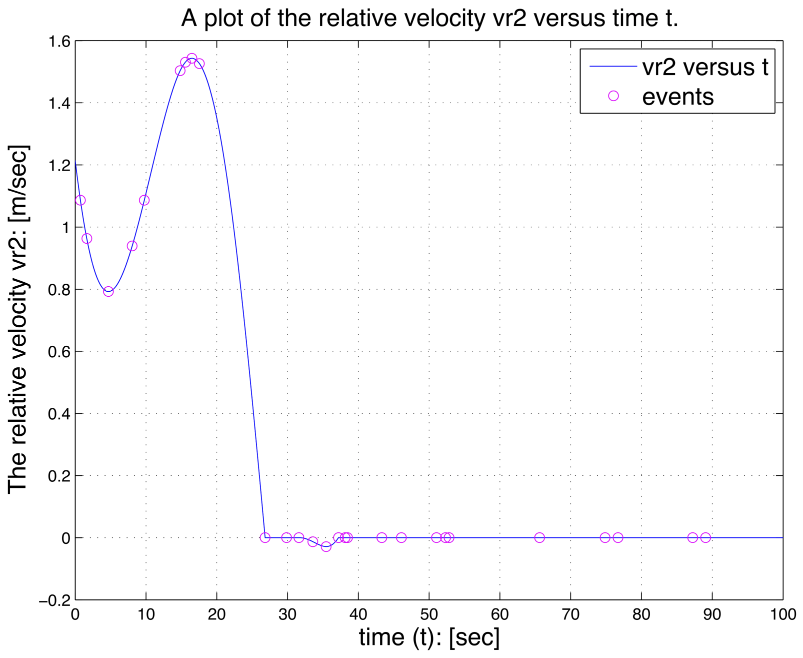


Figure 3.29: Chattering-free simulation of the Spring-Block chain system in Example 3.5: The time evolution of the relative velocity $v_{r_2} = v_{m_2} - v_d$.

3.4.2 Prototype of Chattering-Zeno Freeness in Acumen

Models of hybrid systems are simulated in Acumen [67, 77, 83, 87, 101] by a fine interleaving of sequences that can consist of multiple discrete computations, followed by a single computation updating the values that should evolve continuously. Thus, simulating what is happening at any single instant in time consists of zero or more discrete steps followed by a single continuous step; see Figure 3.30.

The discrete steps capture sudden changes in the system's state, e.g. the impact of two objects, and consist of collecting and evaluating all active discrete actions (discrete assignments or structural actions) in the program until the whole system is stabilized. A system is stabilized when no more discrete steps are required.

The following example illustrates how discrete assignments are handled in Acumen:

Acumen Example:

```
class Main (simulator)
private x = 0; y = 1; z = 1; end
if x<5 x = x+1 end;
end
```

The entire model is repeatedly evaluated until the condition in the **if** statement is false. Simulation time (or logical time) is not advanced during these iterations. Acumen considers such changes to all be happening in the same instant. Using this type of global fixed point semantics allows Acumen to realize, among other things, what is sometimes called the "synchrony hypothesis" whereby the author of the model assume that certain discrete or digital events can happen "fast enough" so that we can view them in the rest of the model as happening instantaneously. In the example above, because the initial value of x is zero, the iteration will end when x has the value 5. Once all discrete actions have taken place, the system moves on to perform all adjustments to the continuous state of the system. The continuous step performs all updates in parallel, meaning that all updates are based on the state that results after the sequence of discrete steps, rather than some later state that resulted from other continuous updates.

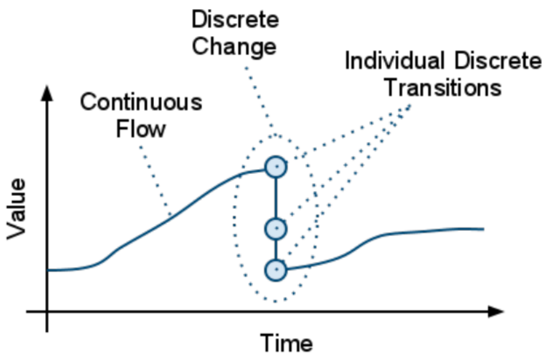


Figure 3.30: Global fixed point semantics in Acumen: Continuous step performs all updates in parallel; all updates are based on the state after discrete steps.

3.4.2.1 Chattering-Zeno Freeness Support for Acumen

In our chattering-Zeno freeness implementation in Acumen, a check for detecting state events $x^\star$ is performed at the end of each time integration step $[t_i, t_{i+1}]$ by using the sign of the event functions $\gamma_j(x)$ represented by the **`if`** statements in Acumen program. The technique of detecting state events is similar to what we used in the FMI chattering-Zeno freeness implementation above. However, in Acumen the technique of computing the event functions $\gamma_j(x)$ is by parsing the **`if`** statement (**`if`** `Expr` **`then`** `Action`). The branches of an **`if`** statement are scanned sequentially until a guard sequence `Expr` which evaluates to true is found. The found predicate `Expr` is then converted into a set of switching functions, by converting a Boolean relation $\bowtie$ of the form {`Expr`$:= a \bowtie b$} into a switching function $\gamma_j = a - b$, where $\bowtie \in \{<, \leq, >, \geq, \ldots\}$. The sub-expressions a and b are usually made of constants, computed variables, and state variables, as well as of arithmetic operations $\diamond \in \{+, -, \times, \div, \ldots\}$ of constants and/or variables.

Similarly to our chattering-Zeno freeness implementation in FMI, chattering-Zeno condition is checked once a state event $x^\star$ is detected. Acumen solver starts the chattering-Zeno detection phase by computing and evaluating, at the detected state event, the normal projections of the dynamics in the neighborhood of the detected state event. The normal projections are then used to check whether the detected state event is a transversality point or a chattering-Zeno point (Definition 3.10 and Definition 3.11).

In the chattering elimination phase, the set-valued convexification in equations (3.64) to (3.66) is employed in order to compute the smooth equivalent sliding dynamics internally, given the dynamics $f_i(x)$ in the neighborhood of $x^\star$, and the set of event functions $\gamma_j(x)$. A monitoring for a smooth exit from sliding is performed while integrating with the sliding dynamics. This is done by monitoring the value of the sliding coefficient κ_i at the end of each sliding integration step $[t_i, t_{i+1}]$. Once the solution is at the final time of a simulation, and the previous step was Fixed Point, then the simulation is terminated.

3.4.2.2 Simulation Results

Figure 3.31 and Figure 3.32 show the chattering-free simulation in Acumen for the Stick-Slip system in Example 3.2, with a fixed time step of size 0.0012. The simulation data set $m = M = 1[kg]$, $F_c = 0.4[N]$, $k = 1[N \cdot m^{-1}]$, and $x_0 = [1\ 1\ 1\ 0]^T$. The external force u was modeled as a sine wave of frequency $\omega = 0.055[rad/sec]$. The first chattering-Zeno event was detected at time $t = 2.89$. During a simulation time of 10 seconds, two chattering-Zeno windows were detected at $\mathbb{I}_1 = [2.89 \ldots 3.99]$ and $\mathbb{I}_2 = [6.73 \ldots 8.69]$, where the system was integrated in these intervals with the sliding dynamics. Figure 3.33 shows the chattering-free simulation of Example 3.3 in Acumen. The system was simulated with the data set $x_1(0) = 1, x_2(0) = 2$ and a step size 0.0012. The first chattering event was detected at $t = 1$, where the solution trajectory started to chatter on the switching manifold $\Sigma_1 = \{x \in \mathbb{R}^2 : \ x_1 = 0 \ \wedge \ x_2 > 0\}$. The solution trajectory converged to the origin (0,0) at time $t = 2$, where the chattering on the intersection $\Delta = \Sigma_1 \cap \Sigma_2 = \{x \in \mathbb{R}^2 : \ x_1 = 0 \ \wedge \ x_2 = 0\}$ was replaced by sliding on it with the chattering-free dynamics (0,0).

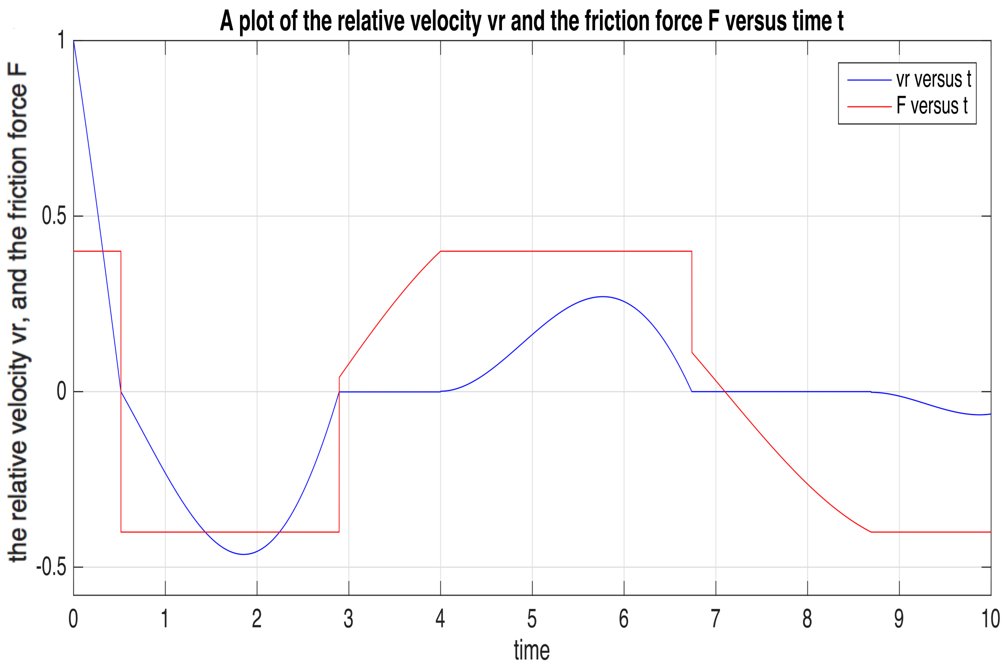


Figure 3.31: Chattering-free simulation in Acumen of Example 3.2: Time evolution of the event function and the control input.

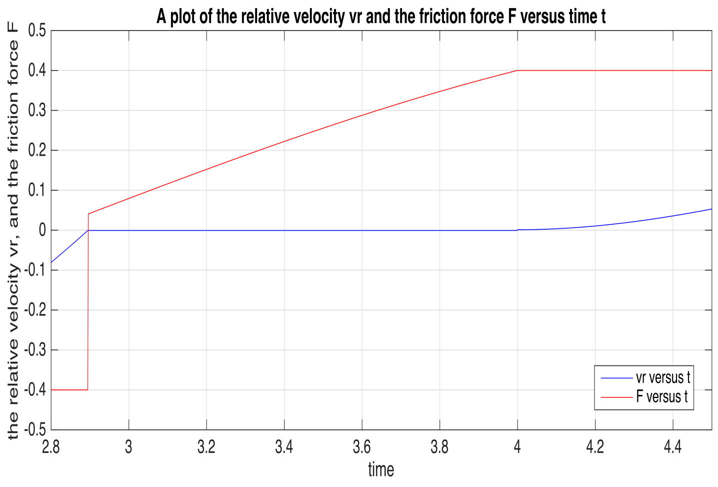


Figure 3.32: Chattering-free simulation in Acumen of Example 3.2: Zoom on the first sliding window.

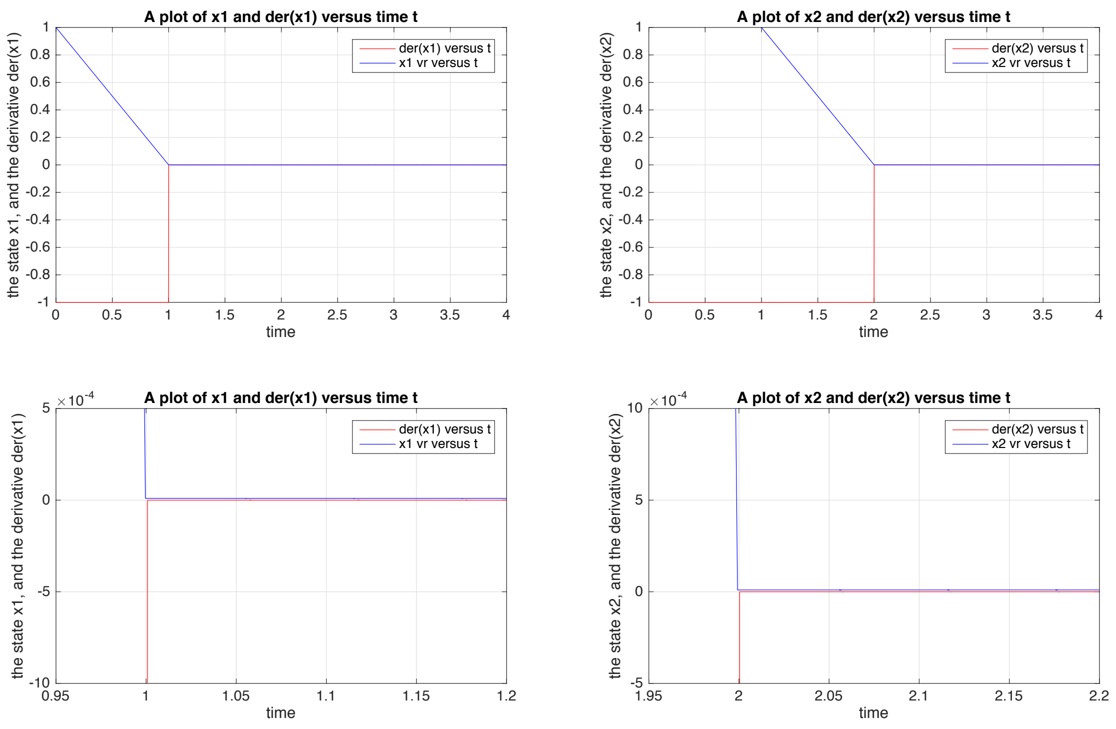


Figure 3.33: Chattering-free simulation in Acumen for the system in Example 3.3. Up: time evolution of the states and their derivatives. Down: zoom on the sliding on $\Sigma_1 = \{x \in \mathbb{R}^2 : x_1 = 0\}$ and the origin $\Delta = \{x \in \mathbb{R}^2 : x_1 = 0 \wedge x_2 = 0\}$.

3.4.3 Performance Analysis and Testing

Other chattering-Zeno case studies were simulated with different simulation scenarios for the purpose of evaluating the efficiency of both FMI and Acumen prototype implementations. In particular, a performance analysis and testing was done, where the performance of the two implementations was evaluated based on the following criteria:

1. The time spent by the simulator to compute the chattering-free solution.
2. The accuracy of the chattering-free simulation. Namely, whether or not:
   (a) all chattering-Zeno events are captured;
   (b) the solution enters the sliding mode instantly when chattering-Zeno occurs;
   (c) the exit from sliding is exactly at the sliding set's boundary;
   (d) the resulted sliding dynamics are correct (based on (b) and (c)).

Table 3.1 shows the performance testing results on six case studies simulated in both FMI and Acumen chattering-free implementations. Table 3.2 compares the time taken to generate the chattering-free solution for the six case studies listed in Table 3.1 when: i) chattering-Zeno is handled manually by the user on the model level, ii) chattering-Zeno is handled internally by the FMI chattering-free simulator, and iii) chattering-Zeno is handled internally by the Acumen chattering-free simulator.

Case St.	**The Model**	**ACED?**[†]	**CTI**[‡][s]	**TISE**[♭][s]
1	$\dot{x}_1 = v_1$, $\dot{x}_2 = v_2$, $\dot{v}_1 = \sin(\omega)$ - x_1 - $(F_c \cdot \text{sign}(v_r))$, $\dot{v}_2 = F_c \cdot \text{sign}(v_r)$, $v_r = v_1$ - v_2, $x_1(0) = 1$, $v_1(0) = 1$, $x_2(0) = 1$, $v_2(0) = 0$, ω=0.055, F_c=0.4, **Simulation Time:** 10.	YES	$\mathbb{I}_1$=[2.89··3.99] $\mathbb{I}_2$=[6.74··8.69]	3.99 8.69
2	$\dot{x}_1$ = - sign (x_1), $\dot{x}_2$ = - sign (x_2), $x_1(0) = 1$, $x_2(0) = 2$, **Simulation Time:** 4.	YES	$\mathbb{I}_1$=[1.00··4.00] $\mathbb{I}_2$=[2.00··4.00]	- -
3	$\dot{x}_1$ = -3x_1 + x_2 - u, $\dot{x}_2$ = -3x_1 + x_3 + u, $\dot{x}_3$ = -x_1 - 0.25u, $u = \text{sign}(x_1)$, $x_1(0) = 0.5$, $x_2(0) = 3$, $x_3(0) = 0.1$, **Simulation Time:** 10.	YES	$\mathbb{I}_1$=[2.65··3.42] $\mathbb{I}_2$=[8.22··9.03]	3.42 9.03
4	$\dot{x}$ = -2 $\cdot$ sign(y), $\dot{\phi} = 0.5$, $y = x$ - $(0.3 \cdot \exp(\phi))$, $x(0) = 1$, $\phi(0) = 0$, **Simulation Time:** 6.	YES	$\mathbb{I}_1$=[0.32··5.18]	5.18
5	$\dot{x}_1 = x_2$, $\dot{x}_2 = \sin(\phi)$ - x_1 - $(F_c \cdot \text{sign}(x_2))$, $\dot{\phi} = \omega$, $\omega = 0.4$, $F_c = 0.4$, $x_1(0) = 1$, $x_2(0) = 0$, $\phi(0) = 1$, **Simulation Time:** 20.	YES	$\mathbb{I}_1$=[0.00··3.74] $\mathbb{I}_2$=[8.92··11.0] $\mathbb{I}_3$=[16.5··19.2]	3.74 11.0 19.2
6	$\dot{x}_1$ = -2x_1 + x_3 - u, $\dot{x}_2$ = -101x_1 + x_3 + 4u, $\dot{x}_3$ = -100x_1 - u, $u = \text{sign}(x_1)$, $x_1(0) = 0.5$, $x_2(0)$ = -1, $x_3(0) = 1$, **Simulation Time:** 1.	YES	$\mathbb{I}_1$=[0.45··1.00]	-

Table 3.1: Summary of the performance analysis and testing for both Acumen and FMI implementations of chattering-Zeno freeness.

† ACED?: All Chattering Events are Detected?
‡ CTI: Chattering Time Intervals.
♭ TISE: Time Instants of the Smooth Exits.

Case St.	**Chattering-Free Manually (User)**	**Chattering-Free (FMI Solver)**	**Chattering-Free (Acumen Solver)**
1	10.288 [s]	1.50 [s]	1.864 [s]
2	5.212 [s]	0.25 [s]	0.480 [s]
3	7.221 [s]	1.20 [s]	1.605 [s]
4	1.728 [s]	0.10 [s]	0.765 [s]
5	15.33 [s]	2.00 [s]	3.196 [s]
6	0.888 [s]	0.10 [s]	0.164 [s]

Table 3.2: Mean time of the chattering-free simulation: Manual manipulation by the user versus automatic detection and elimination by the simulator.

Chapter 4

Geometric-Zeno Detection and Avoidance

In this chapter, we investigate geometric-Zeno behavior of hybrid models in details. We start this chapter by giving a brief introduction to geometric-Zeno behavior in Section 4.1.1. To better understand geometric-Zeno behavior, we give in Section 4.1.2 illustrative examples of hybrid systems models having geometric-Zeno executions, and then we discuss in Section 4.1.3 the problem of simulating their executions. Afterwards, in Section 4.2 we present a method for geometric-Zeno detection and elimination. We derive sufficient conditions for the existence of geometric-Zeno behavior based on the existence of a non-standard contraction map in a complete metric space, and the convergence of the solution to a geometric-Zeno limit point, through such map, according to a Cauchy sequence. Such map indicates when exactly a decision should be taken to transition the solution from pre-Zeno to post-Zeno, and thus eliminating Zeno behavior. In Section 4.3 we present simulation results using a prototype implementation of our proposed technique of geometric-Zeno detection and avoidance.

4.1 Geometric-Zeno in Hybrid Systems

4.1.1 Geometric-Zeno

Geometric-Zeno solutions involve an accumulation of an infinite number of discrete events occurring in finite time, leading to the convergence of solutions to a limit point. In hybrid models that exhibit geometric-Zeno behavior, discrete events occur at an increasingly smaller distance in time, converging against a limit point. The convergence itself to the geometric-Zeno limit point is according to a geometric series. For example, if a new event occurs after half the time between the two previous events, a series of events emerges that, after n events, has moved in time according to $\sum_{k=1}^{n} \frac{1}{2^k}$. This series converges against 1 in the limit of $n \to \infty$. In the following section, we give two typical examples of hybrid systems models having geometric-Zeno behavior, and we discuss the simulation of their executions.

4.1.2 Examples of Geometric-Zeno Models

Example 4.1 (Bouncing Ball)
A typical example of a model that exhibits geometric-Zeno behavior is the model of a bouncing ball whose collisions are inelastic. We consider that the collision of the ball with the ground occurs with a restitution coefficient $\lambda \in (0, 1)$. Figure 4.1 shows the hybrid automaton of the bouncing ball, and Figure 4.2 illustrates the geometric-Zeno behavior of the model. The height of the ball is denoted by x_1, with the invariant constraint $x_1(t) \geq 0$ and dynamics $\ddot{x}_1(t) = -g$, where g is the gravitational constant. The velocity $\dot{x}_1$ of the ball will be denoted by x_2. We include Newton's restitution rule $x_2(t) := -\lambda x_2(t)$ when $x_1(t) \leq 0$ and $x_2(t) < 0$. We can easily compute the time instants of discrete events (or resets) $\{\tau_i\}_{i\in\mathbb{N}}$ by

$$\tau_i = \tau_0 + \frac{2x_2(\tau_0)}{g}\sum_{k=0}^{i-1}\lambda^k; \quad i \in \mathbb{N}. \tag{4.1}$$

Hence, $\{\tau_i\}$ has a limit $\tau_\infty = \tau_0 + \frac{2x_2(\tau_0)}{g\cdot(1-\lambda)} < \infty$, and (x_1, x_2) converges to (0, 0) at τ_∞.

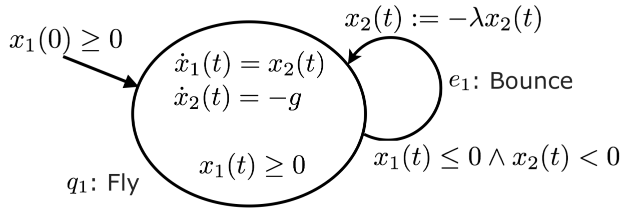


Figure 4.1: Example 4.1 (bouncing ball): The model represented by a hybrid automaton.

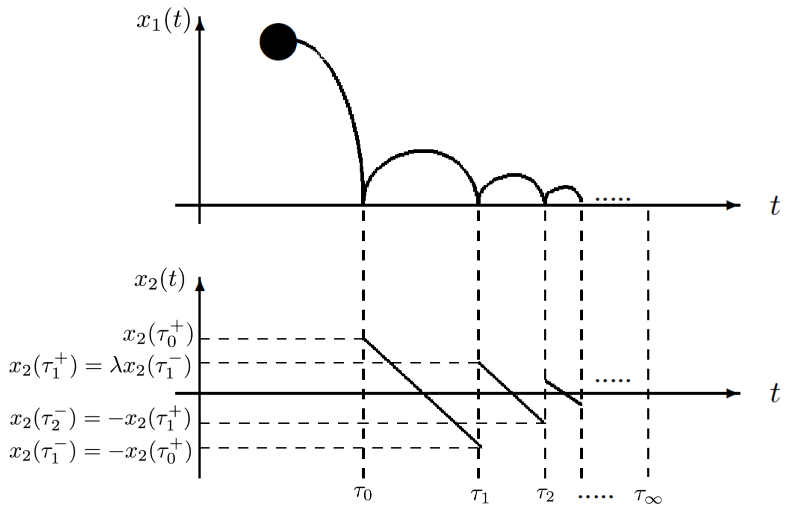


Figure 4.2: Example 4.1 (bouncing ball): Geometric-Zeno behavior.

The physical interpretation is that the ball is at rest within a finite time span, but after infinitely many bounces. A continuation beyond τ_∞ can be defined by $(x_1(t), x_2(t)) = (0, 0)$ for $t > \tau_\infty$. This example is a typical example of a hybrid model having geometric-Zeno behavior, where in this example we have an infinite number of state re-initializations, where the set of event times contains a geometric-Zeno limit point.

Example 4.2 (Two Tanks)
Another canonical example of a geometric-Zeno model is the model of a water tanks system as sketched in Figure 4.3. We denote x_1 and x_2 to the water levels, r_1 and r_2 to the critical thresholds, v_1 and v_2 to the constant water flow going out of the tanks, and w to the constant water flow going into either tank at any given point of time. We assume that $(v_1+v_2) > w$. Thus, the water levels x_1 and x_2 keep dropping. When in either tanks the water level drops below the critical threshold, the pipe switches to deliver the water to that tank. Figure 4.4 shows the hybrid automaton model of the system, where switching the input water pipe between the two tanks occurs with zero time. With initial conditions $x_1 > r_1$ and $x_2 > r_2$, the water levels x_1 and x_2 will drop and as a result the input water pipe gets switched between the two tanks each time the water level hits the critical threshold in either tanks.

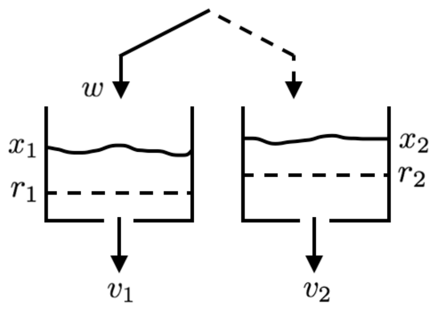


Figure 4.3: Example 4.2: Schematic of the two tanks system.

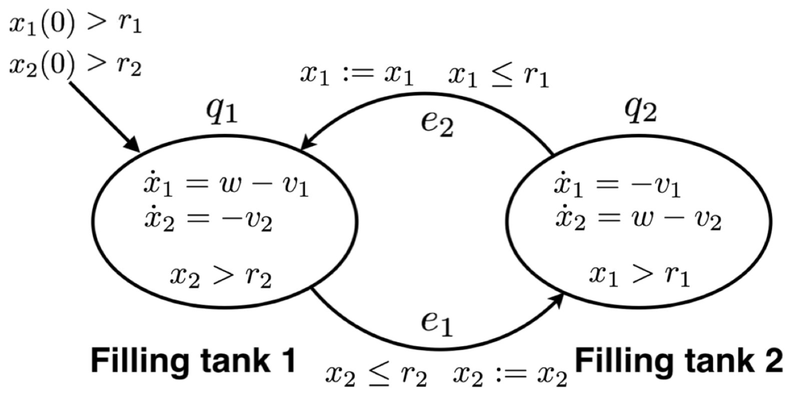


Figure 4.4: Example 4.2: The hybrid automaton model of the system.

As much and much water go out of tanks, we will see that the switching frequency of the input water pipe becomes greater and greater. In the limit point at which $x_1 = r_1$ and $x_2 = r_2$, the switching frequency becomes infinite, and both guards conditions $x_1 \leq r_1$ and $x_2 \leq r_2$ become instantaneously true. As a result, the hybrid automaton would not operate anymore in either of the modes q_1 and q_2. Comparing with the physical system's behavior, we can obviously see how the model can result in a deviant behavior. In the physical system, both tanks become empty at some point in time. In the hybrid automaton model, the water level never gets below the threshold.

Example 4.3 (Spiraling Piecewise Constant System)
Consider the following differential equation with discontinuous right hand side

$$\dot{x} = \begin{cases} \dot{x}_1 = & -\alpha + 2\beta, \\ \dot{x}_2 = & -2\alpha - \beta, \\ \dot{x}_3 = & 1, \end{cases} \quad \alpha = \begin{cases} -1 & \text{if } x_1 < 0, \\ +1 & \text{if } x_1 > 0, \end{cases} \quad \beta = \begin{cases} -1 & \text{if } x_2 < 0, \\ +1 & \text{if } x_2 > 0, \end{cases} \tag{4.2}$$

with initial state $x_1(0) > 0, x_2(0) > 0, x_3(0) \in \mathbb{R}$. In this system, each quadrant of the (x_1, x_2)-plane the right hand side is a constant vector. Figure 4.5 shows the hybrid automaton model of this example. As we can see in Figure 4.6, the solutions of this system are spiraling towards the (x_1, x_2)-plane's origin $(0, 0, x_3)$. Solutions starting at $(x_1(0) > 0, x_2(0) > 0, x_3(0))$ cannot stay away from $(0, 0, x_3)$ for longer than $\frac{1}{2}(|x_1(0)| + |x_2(0)|)$ units of time. However, solutions cannot arrive at $(0, 0, x_3)$ without going through an infinite number of modes switches; since these mode switches would have to occur in a finite time interval, there must be an accumulation of events.

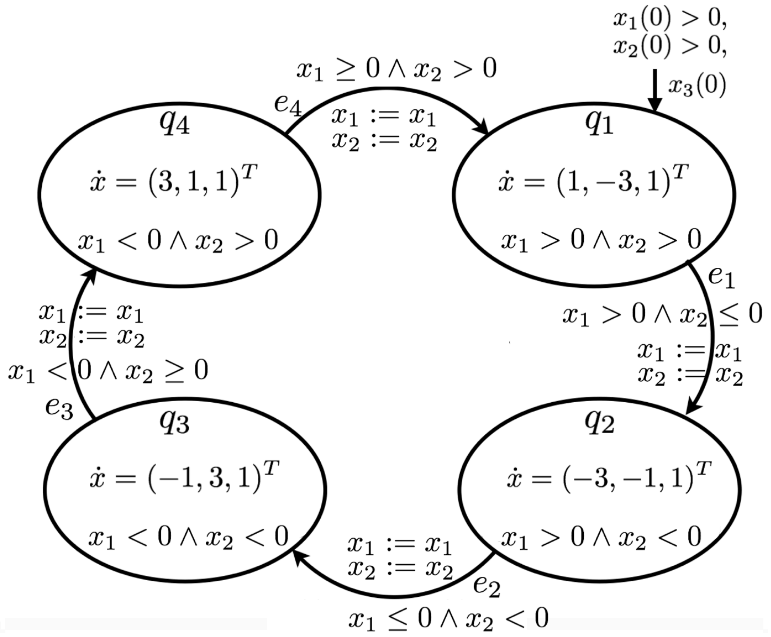


Figure 4.5: Example 4.3: The hybrid automaton model of the system.

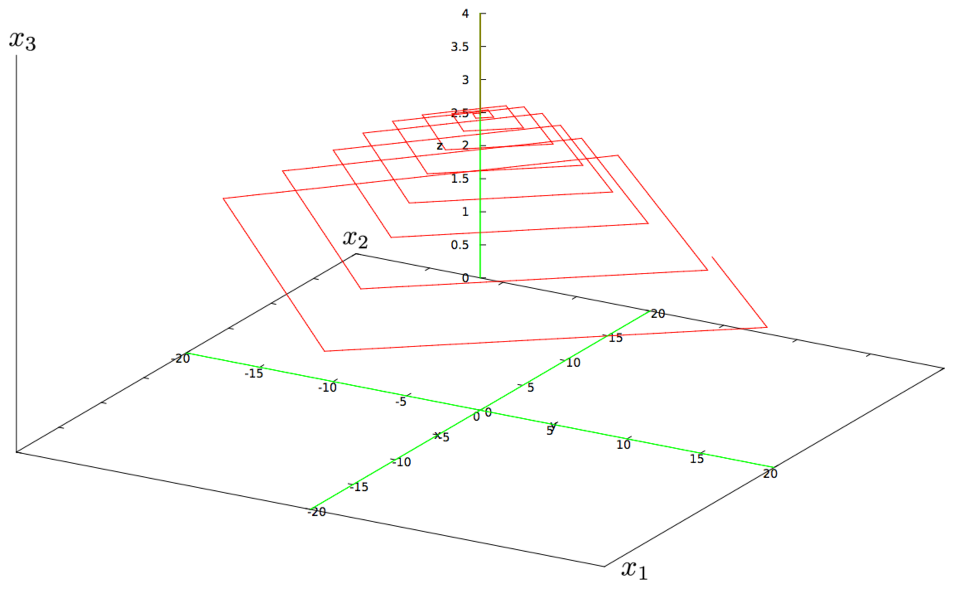


Figure 4.6: Example 4.3: Geometric-Zeno behavior.

4.1.3 Challenges of Simulating Geometric-Zeno Models

Now imagine that one tries to determine, by a numerical simulation, the trajectory of the above three examples, by iteratively integrating over the continuous smooth phases, detecting state events, localizing the time instants of the detected state events, resetting the states, and so on. Such simulation would be broken down and the simulator would get stuck at or before the Zeno limit point. In Simulink, the simulation of the bouncing ball system in Example 4.1 with the data set: $\lambda = 0.5$, $g = 9.81$, and $x(0) = [1\ 0]^T$ terminates with a halt at $t = 1.3546$, which is the Zeno time. The simulation in Simulink of the two tanks system in Example 4.2 with the data set: $w = 1.8$, $v_1 = 1$, $v_2 = 1$, $r_1 = 5$, $r_2 = 5$, and $x(0) = [8\ 6]^T$ terminates also with a halt at $t = 20.0018$. Most of the simulation tools give faulty simulation results when simulating geometric-Zeno models. Consider for example the simulation of Example 4.1, Example 4.2, and Example 4.3 with Modelica simulation tools. In simulation of Example 4.1 with either OpenModelica or Dymola, when the solution converges to the Zeno limit point $x_\infty = \{x(t) \in \mathbb{R}^2 :\ x_1(t) = 0\ \wedge\ x_2(t) = 0\}$ at $t = 1.3546[sec]$, the height x_1 of the ball becomes negative, decreases monotonically, and remains negative forever. Figure 4.7 clearly shows that at some point the value of x_1 will constantly decrease assuming negative value while the equations describing the model are still satisfied. Similarly in Example 4.2, when both $x_1(t)$ and $x_2(t)$ converge to the Zeno limit point $x_\infty = \{x(t) \in \mathbb{R}^2 :\ x_1(t) = r_1\ \wedge\ x_2(t) = r_2\}$ at $t = 20.0018[sec]$, the water level in one tank increases monotonically above the threshold, while the water level in the other tank decreases monotonically below the threshold; see Figure 4.8. Other simulation tools, such as HyVisual, Scicos, Acumen, Zélus also generate faulty results the same as the results generated by Modelica simulation tools.

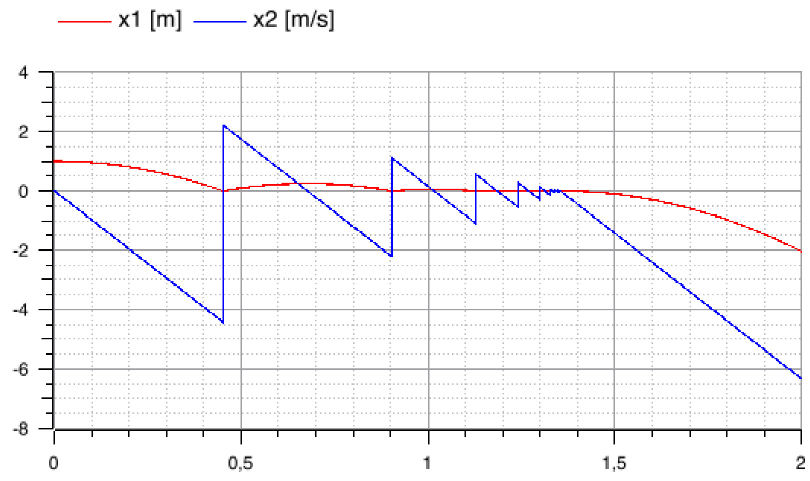


Figure 4.7: Simulation of Example 4.1 in Modelica simulation tools: The time evolution of the height x_1 and velocity x_2 of the bouncing ball.

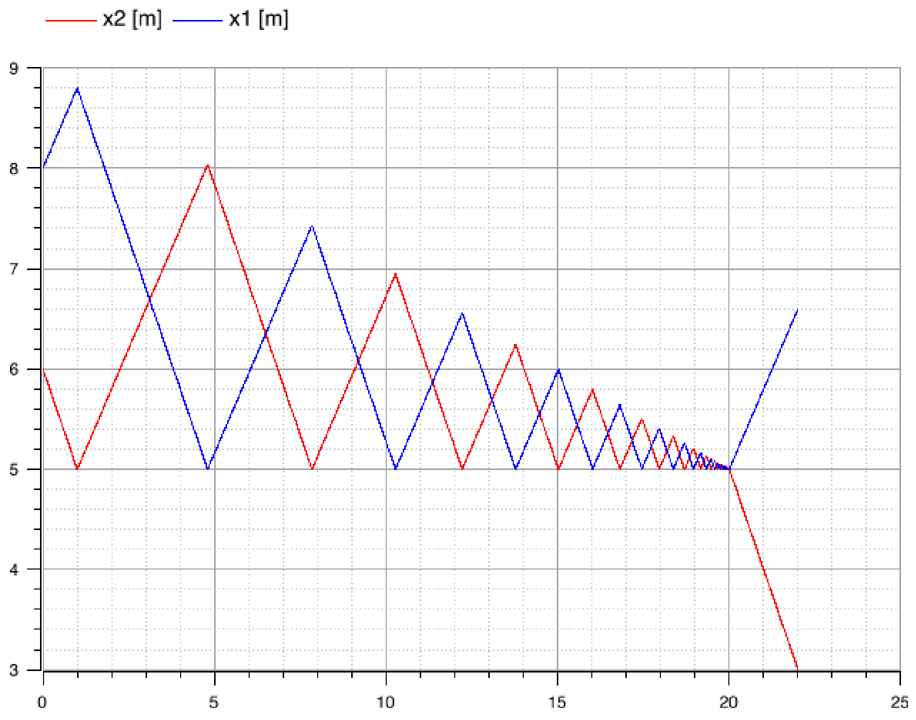


Figure 4.8: Simulation of Example 4.2 in Modelica simulation tools: The time evolution of the water levels x_1 and x_2.

As we have mentioned in Section 2.4; this is due to numerical errors which prevents the simulator from capturing all the state events, when numbers become sufficiently small just before the Zeno limit point.

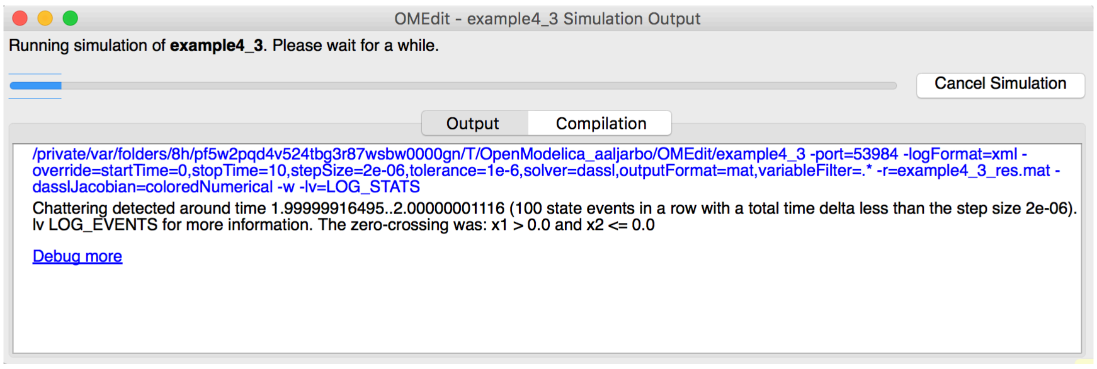
OpenModelica Code of Example 4.1:

```
model Example4_1
type Height = Real(unit = "m");
type Velocity = Real(unit = "m/s");
parameter Real lambda = 0.5;
parameter Height x10 = 1.0;
Height x1; Velocity x2;
initial equation
x1 = x10;
equation
x2 = der(x1); der(x2) = -9.81;
when x1 <= 0 then
reinit(x2, -lambda*pre(x2));
end when;
end Example4_1;
```

OpenModelica Code of Example 4.2:

```
model Example4_2
parameter Real w = 1.8, v1 = 1, v2 = 1, r1 = 5, r2 = 5;
parameter Real x10 = 8.0, x20 = 6.0;
Real x1, x2, u1, u2;
initial equation
x1 = x10; x2 = x20; u1 = w - v1; u2 = 0 - v2;
equation
der(x1) = u1; der(x2) = u2; der(u1) = 0; der(u2) = 0;
when x1 <= r1 then
reinit(u1, w - v1);  reinit(u2, -v2);
end when;
when x2 <= r2 then
reinit(u1, -v1);
reinit(u2, w - v2);
end when;
end Example4_2;
```

Simulating Example 4.3 with both `when` and `if` statements in OpenModelica terminates with a halt. With an initial state ($x(0) = [2, 2, 0]^T$) the simulation halts around time 2.0; see Figure 4.9. OpenModelica reports the following error message: *{Chattering detected around time 1.99999916495..2.00000001116 (100 state events in a row with a total time delta less than the step size 2e-6)}*.



Running simulation of **example4_3**. Please wait for a while.

```
/private/var/folders/8h/pf5w2pqd4v524tbg3r87wsbw0000gn/T/OpenModelica_aaljarbo/OMEdit/example4_3 -port=53984 -logFormat=xml -
override=startTime=0,stopTime=10,stepSize=2e-06,tolerance=1e-6,solver=dassl,outputFormat=mat,variableFilter=.* -r=example4_3_res.mat -
dasslJacobian=coloredNumerical -w -lv=LOG_STATS
Chattering detected around time 1.99999916495..2.00000001116 (100 state events in a row with a total time delta less than the step size 2e-06).
lv LOG_EVENTS for more information. The zero-crossing was: x1 > 0.0 and x2 <= 0.0
```

Figure 4.9: Simulation of Example 4.3 in OpenModelica.

OpenModelica Code of Example 4.3 with when Statement:

```
model Example4_3
parameter Real x10 = 2.0; x20 = 2.0; x30 = 0.0;
Real x1, x2, x3, u1, u2;
initial equation
x1 = x10; x2 = x20; x3 = x30; u1 = 1; u2 = -3;
equation
when x1 > 0 and x2 <= 0 then reinit(u1, -3); reinit(u2, -1); end when;
when x1 <= 0 and x2 < 0 then reinit(u1, -1); reinit(u2, 3); end when;
when x1 < 0 and x2 >= 0 then reinit(u1, 3); reinit(u2, 1); end when;
when x1 >= 0 and x2 > 0 then reinit(u1, 1); reinit(u2, -3); end when;
der(x1) = u1; der(x2) = u2; der(x3) = 1; der(u1) = 0; der(u2) = 0;
end Example4_3;
```

OpenModelica Code of Example 4.3 with if Statement:

```
model Example4_3
parameter Real x10 = 2.0; x20 = 2.0; x30 = 0.0;
Real x1, x2, x3, alpha,beta;
initial equation
x1 = x10; x2 = x20; x3 = x30; alpha = 1; beta = 1;
equation
if x1>0 then alpha=1; elseif x1<0 then alpha=-1; else alpha = 0; end if;
if x2>0 then beta=1; elseif x2<0 then beta=-1; else beta=0; end if;
der(x1) = -alpha + 2*beta; der(x2) = -2*alpha - beta; der(x3) = 1;
end Example4_3;
```

Because in the physical system the hybrid solution can exist beyond the geometric-Zeno limit point, the simulator should be able to predict correctly the solution behavior after Zeno. This motivates the need to a Zeno-Free simulation technique that allows for the geometric-Zeno execution (or trajectory) to be carried correctly beyond the geometric-Zeno limit point.

4.2 Geometric-Zeno Detection and Elimination

A cyclic path is a prerequisite and a necessary condition for a hybrid system's state machine (i.e. hybrid automaton) to accept geometric-Zeno executions. We consider that every pair of two consecutive triggers of the same guard as the input argument for detecting cycles. The convergence of the execution of a hybrid automaton to a geometric-Zeno limit point is completely determined by the convergence of all the cycles detected during the execution through the discrete locations.

4.2.1 Cycles Detection

We firstly introduce the notion of simple cycles and finite cyclic paths in the finite directed graph $A = (Q, E)$.

Definition 4.1 (Finite Cyclic Path) *Given a directed graph $A = (Q, E)$, $E \subseteq Q \times Q$. A path is an alternating sequence of discrete states $q_i \in Q$ and edges $e_i \in E$ of the form*

$$q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \xrightarrow{e_3} \cdots \xrightarrow{e_m} q_m \xrightarrow{e_{m+1}} \cdots$$

such that $e_i = (q_{i-1}, q_i)$ for all i. A path is called:

1. *Simple path: if all the discrete states $q_i \in Q$ appearing in the path are distinct.*
2. *Finite path: if it is a finite sequence $q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \xrightarrow{e_3} \cdots \xrightarrow{e_m} q_m$.*
3. *Finite cyclic path: if it is finite, and the starting state is the same as the ending state. For example the finite path $q_0 \xrightarrow{e_1} q_1 \xrightarrow{e_2} q_2 \xrightarrow{e_3} \cdots \xrightarrow{e_m} q_m$ is called a finite cyclic path if $q_0 = q_m$. Also a portion of a finite path can be a finite cyclic path such as $q_1 \xrightarrow{e_2} q_2$ with $q_1 = q_2$, or $q_2 \xrightarrow{e_3} \cdots \xrightarrow{e_m} q_m$ with $q_2 = q_m$, etc.*

Notation 4.2 To ease future discussion, we denote $C^{q_i} = \langle q_i; e_{i+1}, e_{i+2}, \cdots, e_m; q_i \rangle$ to a finite cyclic path with q_i being its starting and ending state, where $i \geq 0$ and both $m, i \in \mathbb{N}$ being finite. Furthermore, we denote $E(C^{q_i})$ to the set of all edges e_i that appear in the finite cyclic path C^{q_i}. When applied to a finite cyclic path $C^{q_i} = \langle q_i; e_{i+1}, e_{i+2}, \cdots, e_m; q_m = q_i \rangle$ we get $E(C^{q_i}) = \{e_{i+1}, e_{i+2}, \cdots, e_m\}$.

During the execution of a hybrid automaton, a cycle is detected at τ_b on the finite cyclic path C^{q_i} if there exists a transition $e_k \in E(C^{q_i})$ such that

$$x(\tau_a) \in \mathcal{G}(e_k) \ \wedge \ x(\tau_b) \in \mathcal{G}(e_k), \quad \tau_a, \tau_b \in \tau, \quad \tau_b > \tau_a \tag{4.3}$$

where τ is a sequence of intervals (see Definition 2.11 and Figure 2.8), $\mathcal{G}(e_k) \in G$ is the guard of the transition e_k. In other words, a cycle is detected for every pair of two consecutive triggers of the same guard, within a non-zero time duration.

Lemma 4.3 (Necessary Condition for Geometric-Zeno Behavior) *A hybrid automaton $\mathcal{H}$ accepts a geometric-Zeno execution only if there exists a finite cyclic path in the directed graph (Q, E) of $\mathcal{H}$.*

Proof. *If (Q, E) has no cyclic path, then $\mathcal{H}$ accepts executions only with a finite number of discrete mode changes. Such an execution cannot be Zeno.*

4.2.2 The Convergence to a Geometric-Zeno Limit Point

We consider the evolution of the hybrid solution through cycles be represented as a transition system on a metric space. Therefore, in order to derive sufficient conditions for the hybrid automaton's execution to be convergent to a geometric-Zeno limit point, we study, based on non-standard analysis, the existence of a contraction map in a complete metric space, and the convergence of the solution to a geometric-Zeno limit point, through such map, according to a Cauchy sequence.

Definition 4.4 (Non-Standard Euclidean Distance) *In Cartesian coordinates, consider two non-standard points $^*p = (^*p_1, ^*p_2, \cdots, ^*p_n)$ and $^*q = (^*q_1, ^*q_2, \cdots, ^*q_n)$ in a non-standard Euclidean n-space $^*\mathbb{R}^n$, then the Euclidean distance $d \in {}^*\mathbb{R}$ between $^*p$ and $^*q$ is given by the Pythagorean formula:*

$$d(^*p, ^*q) = ||^*p, ^*q||_E = \sqrt{\sum_{i=1}^{n} (^*p_i - ^*q_i)^2}. \tag{4.4}$$

Note that, in $^*\mathbb{R}$, the distance between $^*p_i$ and $^*q_i$ is given by

$$d(^*p_i, ^*q_i) = |^*p_i - ^*q_i|. \tag{4.5}$$

Definition 4.5 (Non-Standard Metric Space) *A non-standard metric on a set $^*X \subseteq {}^*\mathbb{R}^n$ is a function $d : {}^*X \times {}^*X \to {}^*\mathbb{R}$ satisfying for $^*x, ^*y, ^*z \in {}^*X$:*

1. *$d(^*x, ^*y) \geq 0$ for all $^*x, ^*y \in {}^*X$;*
2. *$d(^*x, ^*y) = 0$ if and only if $^*x = {}^*y$, and $d(^*x, ^*y) > 0$ when $^*x \neq {}^*y$;*
3. *$d(^*x, ^*y) = d(^*y, ^*x)$ for all $^*x, ^*y \in {}^*X$ (Symmetry);*
4. *$d(^*x, ^*y) + d(^*y, ^*z) \geq d(^*x, ^*z)$ for all $^*x, ^*y, ^*z \in {}^*X$ (Triangle Inequality).*

A non-standard metric space is a pair $(^*X, d)$ consisting of a set $^*X \subseteq {}^*\mathbb{R}^n$ and a non-standard metric d on $^*X$.

Definition 4.6 (Open and Closed Balls and Sets in a Non-Standard Metric Space) *Given a non-standard metric space $(^*X, d)$, where $^*X \subseteq {}^*\mathbb{R}^n$, the open ball with center $^*x \in {}^*X$ and radius $^*r$ is the set*

$$^*B^o(^*x, ^*r) = \{^*y \in {}^*X : d(^*x, ^*y) < {}^*r\}. \tag{4.6}$$

A closed ball is defined analogously as the set

$$^*B^c(^*x, ^*r) = \{^*y \in {}^*X : d(^*x, ^*y) \leq {}^*r\}. \tag{4.7}$$

A subset $^*U$ of a metric space $(^*X, d)$ is open if for all $^*x \in {}^*U$, there is $^*r > 0$ such that $^*B^o(^*x, ^*r) \subset {}^*U$. A subset $^*U$ of a metric space $(^*X, d)$ is closed if it complement $^*X \setminus {}^*Y = \{^*x \in {}^*X | ^*x \notin {}^*Y\}$ is open. A subset $^*U$ of a metric space $(^*X, d)$ is bounded if there exists a closed ball of finite radius that contains it. In other words, $d(^*x, ^*y) \leq {}^*k$ for all $^*x, ^*y \in {}^*X$ and some constant $^*k < \infty$.

Definition 4.7 (Non-Standard Convergent Sequences) *In any set $^*X$, a sequence $\{^*x_n\}$ in $^*X$ is just a mapping $^*\rho : {}^*X \to {}^*X$, $n \mapsto {}^*x_n$, $n \in {}^*\mathbb{N}$. Let $(^*X, d)$ be a metric space and $\{^*x_n\} \subset {}^*X$, $n \in {}^*\mathbb{N}$, be a sequence in $^*X$, we say that $\{^*x_n\}$ converges to an element $^*x \in {}^*X$ if for all $\varepsilon > 0$, $\varepsilon \in {}^*\mathbb{R}$, there exists an $N = N(\varepsilon) \in {}^*\mathbb{N}$ such that for all $n \geq N$, $d(^*x_n, {}^*x) < \varepsilon$. We denote this by $^*x_n \to {}^*x$, and in this case, $^*x$ is said to be the limit of the sequence $\{^*x_n\}$, namely $st(d(^*x_n, {}^*x)) \to 0$ as $n \to \infty$, where $st(d(\cdot))$ denotes the standardization of $d(\cdot)$. If a sequence $\{^*x_n\}$ has a limit, this limit is unique.*

Definition 4.8 (Non-Standard Cauchy Sequences) *Let $(^*X, d)$ be a metric space and $\{^*x_n\} \subset {}^*X$, $n \in {}^*\mathbb{N}$, be a sequence $^*X$, we say that $\{^*x_n\}$ is a Cauchy sequence if for all $\varepsilon > 0$, $\varepsilon \in {}^*\mathbb{R}$, there exists an $N = N(\varepsilon) \in {}^*\mathbb{N}$ such that for all $m, n \geq N$, $m, n \in {}^*\mathbb{N}$, $d(^*x_n, {}^*x_m) < \varepsilon$. As $n, m \to \infty$ we have $st(d(^*x_n, {}^*x_m)) \to 0$.*

Lemma 4.9 *A non-standard metric space $(^*X, d)$ is complete if every non-standard Cauchy sequence $\{^*x_n\}$ contained in $^*X$ is convergent to some $^*x \in {}^*X$.*

Definition 4.10 (Continuity in a Non-Standard Metric Space) *Let $^*\rho : {}^*X \to {}^*X$ be a non-standard map on the metric space $(^*X, d)$, and let $^*x \in {}^*X$. We say that $^*\rho$ is continuous at $^*x$ if for all $\varepsilon > 0$, $\varepsilon \in {}^*\mathbb{R}$, there is $\delta > 0$, $\delta \in {}^*\mathbb{R}$, such that for all $^*y \in {}^*X$, if $d(^*x, {}^*y) < \delta$ then $d(^*\rho(^*x), {}^*\rho(^*y)) < \varepsilon$. We say $^*\rho$ is continuous if it is continuous at every $^*x \in {}^*X$. Equivalently, $^*x_n \to {}^*x$ implies $^*\rho(^*x_n) \to {}^*\rho(^*x)$.*

Definition 4.11 (Non-Standard Contraction Mapping in Non-Standard Metric Space) *Let $^*\rho : {}^*X \to {}^*X$ be a non-standard map on the non-standard metric space $(^*X, d)$. We say that $^*\rho$ is a contraction of modulus $\beta \in {}^*\mathbb{R}$ if there exists $\beta \in (0, 1)$, $\beta \in {}^*\mathbb{R}$, such that $d(^*\rho(^*x), {}^*\rho(^*y)) \leq \beta d(^*x, {}^*y)$ for all $^*x, {}^*y \in {}^*X$. Informally, a contraction map brings any two points of a set closer to each other.*

Theorem 4.12 *Let $(^*X, d)$ be a non-standard complete metric space and $^*\rho : {}^*X \to {}^*X$ be a non-standard contraction with modulus β, then:*

1. *$^*\rho$ has a unique fixed point $^*x^\star \in {}^*X$ satisfying $^*\rho(^*x) = {}^*x^\star$;*
2. *the sequence $^*x_1 = {}^*\rho(^*x_0)$, $^*x_2 = {}^*\rho(^*x_1)$, ..., $^*x_{n+1} = {}^*\rho(^*x_n)$ is a Cauchy sequence in $^*X$, and converges to $^*x^\star$ for any starting point $^*x_0 \in {}^*X$.*

Proof. *Pick any $^*x_1 \in {}^*X$ and iterate $^*x_{n+1} = {}^*\rho(^*x_n)$, $n = 1, 2, \cdots$. For all n we have*

$$d(^*x_{n+1}, {}^*x_n) \leq \beta^{n-1} d(^*x_2, {}^*x_1). \tag{4.8}$$

(4.8) results from induction, noting that

$$d(^*x_{n+2}, {}^*x_{n+1}) = d(^*\rho(^*x_{n+1}), {}^*\rho(^*x_n)) \leq \beta d(^*x_{n+1}, {}^*x_n). \tag{4.9}$$

If $m > n$ we deduce from (4.8) that

$$d(^*x_m, {}^*x_n) \leq \beta^{n-1}(1 + \cdots + \beta^{m-n-1}) d(^*x_2, {}^*x_1) = \frac{\beta^{n-1}}{1-\beta} d(^*x_2, {}^*x_1). \tag{4.10}$$

Hence $\{^*x_n\}$ is a Cauchy sequence, and since $^*X$ is complete then $^*x_n \to {}^*x^\star$ for some $^*x^\star \in {}^*X$. Passing to the limit in $^*x_{n+1} = {}^*\rho(^*x_n)$ we get $^*x^\star = {}^*\rho(^*x^\star)$.

Theorem 4.13 (Sufficient Condition for Geometric-Zeno Behavior) *An execution of a hybrid automaton $\mathcal{H}$ is geometric-Zeno if the following two conditions are satisfied:*

1. *The directed graph $(\mathcal{Q}, E)$ of $\mathcal{H}$ contains at least one finite cyclic path C^{q_i}, i.e. the necessary condition in Lemma 4.3 for the existence of geometric-Zeno behavior is fulfilled.*

2. *The continuous part of the automaton's hybrid solution trajectory is a non-standard Cauchy sequence $\{^*x_n\}$, $n \in {}^*\mathbb{N}$, in a complete non-standard metric space $^*X$. Any line that starts from the limit $x^\star$ of $\{^*x_n\}$ and intersects all cycles will form a non-standard Cauchy subsequence $\{^*x_j\} \subset \{^*x_n\}$, $j \in {}^*\mathbb{N}$, satisfying $^*x_{j+1} = {}^*\rho(^*x_j)$ for all j, with $^*\rho$ being non-standard contraction map of fixed modulus β. The limit point to which both Cauchy sequences $\{^*x_n\}$ and $\{^*x_j\}$ converge is the geometric-Zeno limit point.*

The second condition in Theorem 4.13 indicates that for any $^*x_n \in \{^*x_n\}$ we can find a subsequence $\{^*x_j\} \subset \{^*x_n\}$ that is a Cauchy sequence, and that converges — according to a geometric series — to the same limit point to which converges $\{^*x_n\}$.

4.2.3 Geometric-Zeno Elimination

A way to eliminate geometric-Zeno behavior is by enabling a transition from the pre-Zeno to a post-Zeno — and thus stopping the Cauchy sequence $\{^*x_n\}$ of the solution — at $^*x_n \in \{^*x_n\}$ once $^*x_n \approx {}^*\rho(^*x_n)$. The step in which this transition is taken would be used as the final step with the original dynamics, and this step would be used to carry the transition from pre-Zeno to post-Zeno state.

The idea of carrying the execution beyond the geometric-Zeno limit point is by forcing the system to slide on the switching surface to which belong the geometric-Zeno limit point. In the interval in which the transition from pre-Zeno to post-Zeno is taken, the system switches its dynamics to the sliding dynamics $^*f_\infty = 0 \cdot {}^*f(q_i, {}^*x)$ where $^*f(q_i, {}^*x)$ is the original dynamics of the system. Note that the transition from pre-Zeno to post-Zeno is urgent.

When the transition from pre-Zeno to post-Zeno is taken, the simulator switches instantly to integrate the system with the new dynamics and the rest of the events before the Zeno time point are discarded.

Example 4.1 (Bouncing Ball) revisited: Consider again the bouncing ball model in Example 4.1. The ball's velocity and height as functions of time from $^*t_s \in \mathbb{T}_\partial$ to $^*t \in \mathbb{T}_\partial$ are given by

$$\begin{aligned} {}^*x_2({}^*t - {}^*t_s) &= {}^*x_2({}^*t_s) - ({}^*t - {}^*t_s) \cdot g, \\ {}^*x_1({}^*t - {}^*t_s) &= {}^*x_1({}^*t_s) + ({}^*t - {}^*t_s) \cdot {}^*x_2({}^*t_s) - \frac{g \cdot ({}^*t - {}^*t_s)^2}{2}. \end{aligned} \tag{4.11}$$

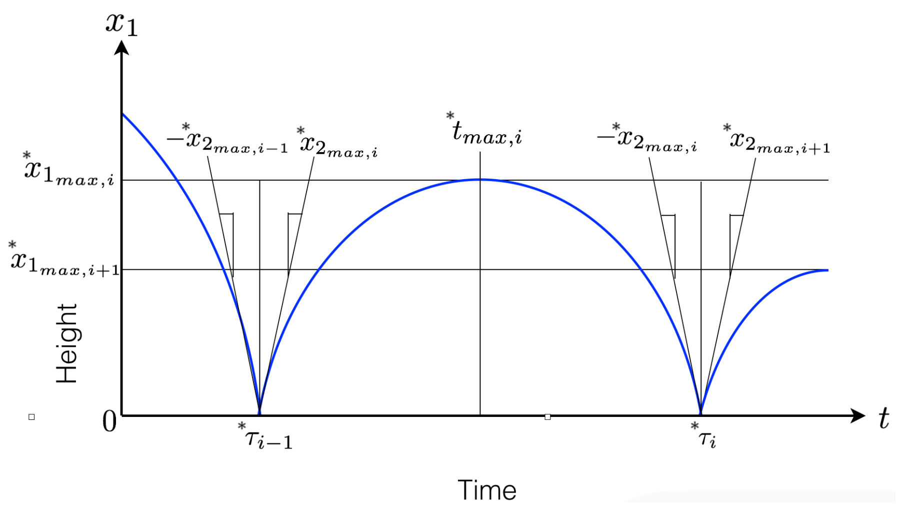


Figure 4.10: Generalized bounce cycle.

Consider a bouncing cycle occurring from $^*\tau_{i-1}$ to $^*\tau_i$ as shown in Figure 4.10. We can find out at what time $^*t_{max,i}$ the ball reaches its maximum height $^*x_{1_{max,i}}$ by solving for the time at which the velocity is zero using (4.11) with $^*t_s = {}^*\tau_{i-1}$ and $^*x_2(^*t_s) = {}^*x_{2_{max,i}}$. This gives us:

$$^*t_{max,i} = \frac{^*x_{2_{max,i}}}{g}. \tag{4.12}$$

Next, we can then find the ball's maximum height $^*x_{1_{max,i}}$ at time $^*t_{max,i}$ using (4.11) and (4.12) with $^*x_1(^*t_s) = 0$, and again $^*x_2(^*t_s) = {}^*x_{2_{max,i}}$. This gives us

$$^*x_{1_{max,i}} = \frac{^*x^2_{2_{max,i}}}{2g}. \tag{4.13}$$

Denote $^*x_{20}$ to the initial rebound velocity. As the ball is bouncing according to a coefficient of restitution λ on velocity, the relation for any rebound velocity $^*x_{2_{max,i}}$ to the initial rebound velocity $^*x_{20}$ is given by

$$^*x_{2_{max,i}} = \lambda^i \cdot {}^*x_{20}. \tag{4.14}$$

Using (4.14) we can re-write (4.12) and (4.13) as

$$^*t_{max,i} = \frac{\lambda^i \cdot {}^*x_{20}}{g}, \qquad ^*x_{1_{max,i}} = \frac{\lambda^{2i} \cdot {}^*x^2_{20}}{2g}. \tag{4.15}$$

Figure 4.11 shows the trajectory of the bouncing ball on (x_1, x_2)-plane. For each bouncing cycle i, the curve is given by

$$^*x_{1_{,i}} = \frac{1}{2g}(^*x^2_{20}\lambda^{2i} - {}^*x^2_{2_{,i}}). \tag{4.16}$$

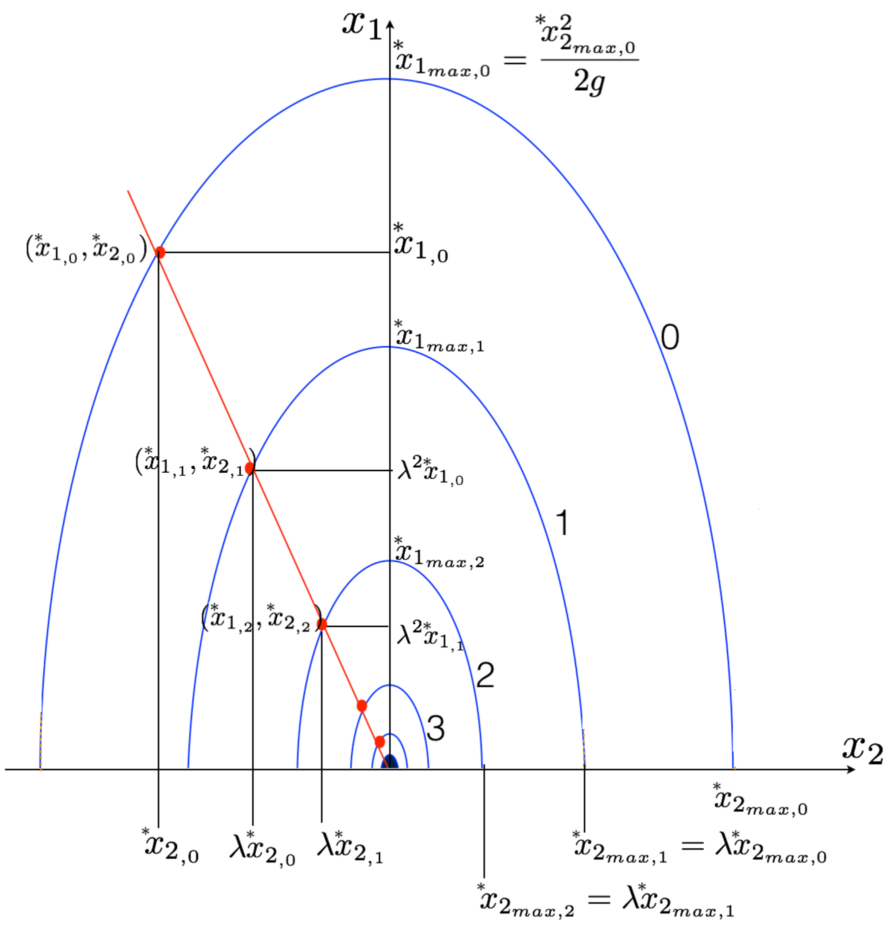

x_1
$\overset{*}{x}_{1_{max,0}} = \frac{\overset{*}{x}^2_{2_{max,0}}}{2g}$
$(\overset{*}{x}_{1,0}, \overset{*}{x}_{2,0})$
$\overset{*}{x}_{1,0}$
$\overset{*}{x}_{1_{max,1}}$
0
$(\overset{*}{x}_{1,1}, \overset{*}{x}_{2,1})$
$\lambda^2 \overset{*}{x}_{1,0}$
1
$\overset{*}{x}_{1_{max,2}}$
$(\overset{*}{x}_{1,2}, \overset{*}{x}_{2,2})$
$\lambda^2 \overset{*}{x}_{1,1}$
2
3
x_2
$\overset{*}{x}_{2_{max,0}}$
$\overset{*}{x}_{2,0}$
$\lambda \overset{*}{x}_{2,0}$
$\lambda \overset{*}{x}_{2,1}$
$\overset{*}{x}_{2_{max,1}} = \lambda \overset{*}{x}_{2_{max,0}}$
$\overset{*}{x}_{2_{max,2}} = \lambda \overset{*}{x}_{2_{max,1}}$

Figure 4.11: The hybrid solution trajectory of the bouncing ball model on (x_1, x_2)-plane.

We can observe that the hybrid solution trajectory in this example is a non-standard Cauchy sequence in a non-standard complete metric space. If we draw a line starting from the Zeno limit point (0,0) and intersecting each cycle i at a state $({}^*x_{1,i}, {}^*x_{2,i})$, we find that all the states $({}^*x_{1,i}, {}^*x_{2,i})$, that lie on this line form also a non-standard Cauchy subsequence $\{{}^*x_j\}$, satisfying ${}^*x_{j+1} = {}^*\rho({}^*x_j)$ for all j, with ${}^*\rho$ being non-standard contraction map of fixed modulus $\beta \in (0,1)$. Consider for example the three states $({}^*x_{1,0}, {}^*x_{2,0})$, $({}^*x_{1,1}, {}^*x_{2,1})$, and $({}^*x_{1,2}, {}^*x_{2,2})$ picked up as sketched in Figure 4.11. We have:

$$\begin{aligned} {}^*x_{1,0} &= \frac{1}{2g}({}^*x_{20}^2 - {}^*x_{2,0}^2), \\ {}^*x_{1,1} &= \frac{1}{2g}({}^*x_{20}^2\lambda^2 - {}^*x_{2,1}^2), \\ {}^*x_{1,2} &= \frac{1}{2g}({}^*x_{20}^2\lambda^4 - {}^*x_{2,2}^2). \end{aligned} \tag{4.17}$$

In (4.17), we can see that:

- For the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$, with ${}^*x_{2_{,1}} = \lambda^* x_{2_{,0}}$ we have ${}^*x_{1_{,1}} = \lambda^{2*} x_{1_{,0}}$.
- Similarly, for the pair of states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{2_{,2}} = \lambda^* x_{2_{,1}}$ we have ${}^*x_{1_{,2}} = \lambda^{2*} x_{1_{,1}}$.
- Similarly, for the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{2_{,2}} = \lambda^{2*} x_{2_{,0}}$ we have ${}^*x_{1_{,2}} = \lambda^{4*} x_{1_{,0}}$.

Denote $\Psi = \{{}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})$ that lie on the same line which starts from the Zeno limit point (0,0). We have ${}^*x_{2_{,i+m}} = \lambda^{m*} x_{2_{,i}}$, ${}^*x_{1_{,i+m}} = \lambda^{2m*} x_{1_{,i}}$ for all ${}^*x_{1_{,i}}, {}^*x_{2_{,i}} \in \Psi$. Also, let ${}^*\rho_1, {}^*\rho_2$ be two maps such that ${}^*x_{1_{,i+1}} = {}^*\rho_1({}^*x_{1_{,i}})$, and ${}^*x_{2_{,i+1}} = {}^*\rho_2({}^*x_{2_{,i}})$, then it holds that:

1. For all ${}^*x_{2_{,i}} \in \Psi$ we have
$$d({}^*\rho_2({}^*x_{2_{,i}}), {}^*\rho_2({}^*x_{2_{,i+1}})) = \lambda d({}^*x_{2_{,i}}, {}^*x_{2_{,i+1}}). \tag{4.18}$$
2. For all ${}^*x_{1_{,i}} \in \Psi$ we have
$$d({}^*\rho_1({}^*x_{1_{,i}}), {}^*\rho_1({}^*x_{1_{,i+1}})) = \lambda^2 d({}^*x_{1_{,i}}, {}^*x_{1_{,i+1}}). \tag{4.19}$$

Both ${}^*\rho_1$ and ${}^*\rho_2$ are contraction maps because $\lambda \in (0,1)$ as it is assumed in the model. Geometric-Zeno elimination involves enabling a transition from pre-Zeno to post-Zeno by integrating the system with dynamics (0,0) once ${}^*x_{1_{,i}} \approx {}^*\rho_1({}^*x_{1_{,i}})$ and ${}^*x_{2_{,i}} \approx {}^*\rho_2({}^*x_{2_{,i}})$, namely once $d({}^*x_{1_{,i}}, {}^*x_{1_{,i+1}}) \approx 0$ and $d({}^*x_{2_{,i}}, {}^*x_{2_{,i+1}}) \approx 0$. Note that, this naturally occurs when $\lambda^m \to 0$ and $\lambda^{2m} \to 0$ as $m \to \infty$.

Example 4.2 (Two Tanks) revisited: Consider again the two tanks model in Example 4.2. The system hybrid automaton (Figure 4.4) contains two cyclic paths, $C^{q_1} = \langle q_1; e_1, e_2; q_1\rangle$ and $C^{q_2} = \langle q_2; e_2, e_1; q_2\rangle$, where e_1 is the transition from q_1 to q_2 guarded by the guard condition $\mathcal{G}(e_1) = \{x(t) \in \mathbb{R}^2 : x_2(t) \le r_2\}$, and e_2 is the transition from q_2 to q_1 guarded by the guard condition $\mathcal{G}(e_2) = \{x(t) \in \mathbb{R}^2 : x_1(t) \le r_1\}$.

In the discrete state q_1 (mode filling tank 1), the water levels in the two tanks as functions of time from ${}^*t_s \in \mathbb{T}_\partial$ to ${}^*t \in \mathbb{T}_\partial$ are given by

$$\begin{aligned} {}^*x_1({}^*t - {}^*t_s) &= {}^*x_1({}^*t_s) + ({}^*t - {}^*t_s) \cdot (w - v_1), \\ {}^*x_2({}^*t - {}^*t_s) &= {}^*x_2({}^*t_s) - ({}^*t - {}^*t_s) \cdot v_2. \end{aligned} \tag{4.20}$$

In the discrete state q_2 (mode filling tank 2), the water levels in the two tanks as functions of time from ${}^*t_s \in \mathbb{T}_\partial$ to ${}^*t \in \mathbb{T}_\partial$ are given by

$$\begin{aligned} {}^*x_1({}^*t - {}^*t_s) &= {}^*x_1({}^*t_s) - ({}^*t - {}^*t_s) \cdot v_1, \\ {}^*x_2({}^*t - {}^*t_s) &= {}^*x_2({}^*t_s) + ({}^*t - {}^*t_s) \cdot (w - v_2). \end{aligned} \tag{4.21}$$

On either of the two cyclic paths $C^{q_1} = \langle q_1; e_1, e_2; q_1\rangle$ and $C^{q_2} = \langle q_2; e_2, e_1; q_2\rangle$, each cycle occurs within two discrete transitions from ${}^*\tau_{i-1}$ to ${}^*\tau_{i+1}$; see Figure 4.12.

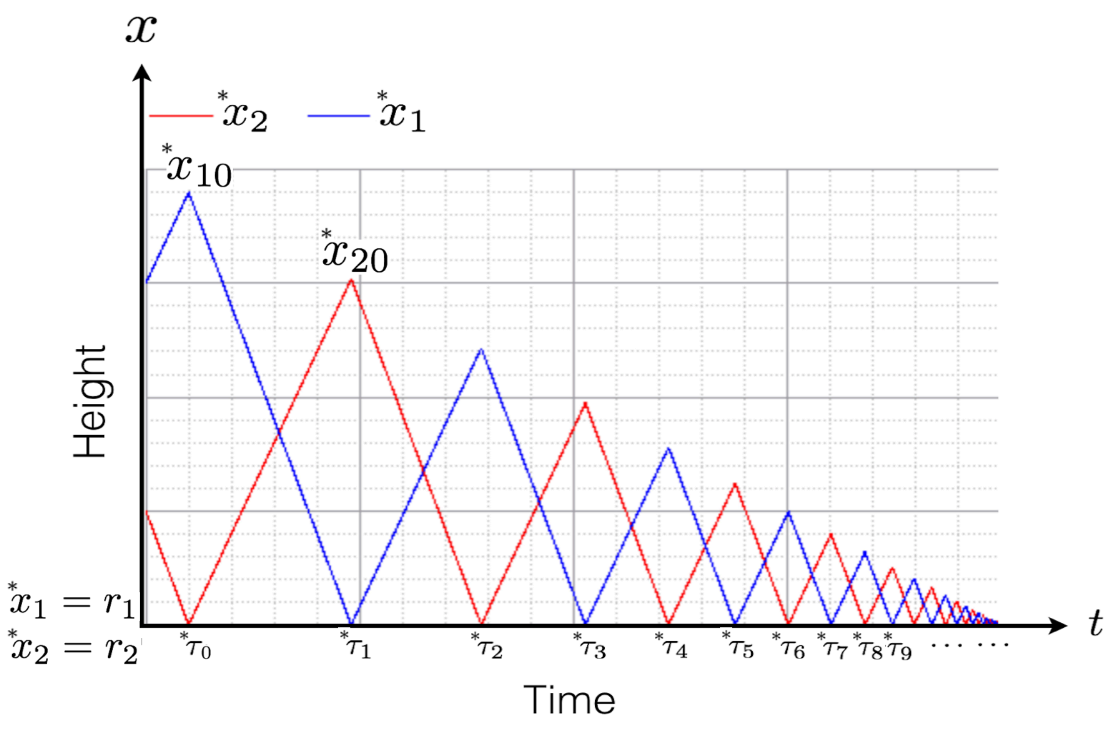


Figure 4.12: Example 4.2: The time evolution of the water levels in the two tanks.

We can find out what time $^*t_{1_{max,i}}$ the water level in tanks 1 takes to reach its maximum height $^*x_{1_{max,i}}$ at $^*\tau_i$ starting from the threshold level r_1, by solving for the time at which the water level in tanks 2 is equal to the threshold level r_2, using (4.20) with $^*t_s = {}^*\tau_{i-1}$, $^*t = {}^*\tau_i$, $^*x_2(^*t_s) = {}^*x_{2_{max,i-1}}$, and $^*x_2(^*t - {}^*t_s) = r_2$. This gives us:

$$^*t_{1_{max,i}} = \frac{^*x_{2_{max,i-1}} - r_2}{v_2}. \tag{4.22}$$

Next, we can then find the maximum height $^*x_{1_{max,i}}$ at $^*\tau_i$ starting from the threshold level r_1, using (4.20) and (4.22) with $^*x_1(^*t_s) = r_1$ and $^*x_2(^*t_s) = {}^*x_{2_{max,i-1}}$.
This gives us:

$$^*x_{1_{max,i}} - r_1 = k_1 \cdot (^*x_{2_{max,i-1}} - r_2), \qquad k_1 = \frac{w - v_1}{v_2}. \tag{4.23}$$

Similarly, we can find out what time $^*t_{2_{max,i}}$ the water level in tanks 2 takes to reach its maximum height $^*x_{2_{max,i}}$ at $^*\tau_i$ starting from the threshold level r_2, by solving for the time at which the water level in tanks 1 is equal to the threshold level r_1, using (4.21) with $^*t_s = {}^*\tau_{i-1}$, $^*t = {}^*\tau_i$, $^*x_2(^*t_s) = {}^*x_{2_{max,i-1}}$, and $^*x_1(^*t - {}^*t_s) = r_1$.
This gives us:

$$^*t_{2_{max,i}} = \frac{^*x_{1_{max,i-1}} - r_1}{v_1}. \tag{4.24}$$

From (4.24) and (4.21) with ${}^*x_2({}^*t_s) = r_2$ and ${}^*x_1({}^*t_s) = {}^*x_{1_{max,i-1}}$, the maximum height $x_{2_{max,i}}$ of the water level in tank 2 at τ_i is given by

$$ {}^*x_{2_{max,i}} - r_2 = k_2 \cdot ({}^*x_{1_{max,i-1}} - r_1), \qquad k_2 = \frac{w - v_2}{v_1}. \tag{4.25}$$

Denote ${}^*x_{10}$ to the initial highest level of water in tank 1, and ${}^*x_{20}$ to the initial highest level of water in tank 2 (given in Figure 4.12 by ${}^*x_{10} = {}^*x_1({}^*\tau_0)$ and ${}^*x_{10} = {}^*x_2({}^*\tau_1)$).

From (4.23) and (4.25), the relation for any maximum water level's height ${}^*x_{k_{max,j}}$ to its previous maximum height ${}^*x_{k_{max,j-1}}$, $k \in \{1, 2\}$ and to the initial highest level ${}^*x_{k0}$ is given for both tanks by

$$\begin{aligned} {}^*x_{1_{max,j}} - r_1 &= \lambda \cdot ({}^*x_{1_{max,j-1}} - r_1) \\ &= \lambda^j ({}^*x_{10} - r_1), \end{aligned} \tag{4.26}$$

$$\begin{aligned} {}^*x_{2_{max,j}} - r_2 &= \lambda \cdot ({}^*x_{2_{max,j-1}} - r_2) \\ &= \lambda^j ({}^*x_{20} - r_2), \end{aligned} \tag{4.27}$$

where

$$\lambda = k_1 \cdot k_2 = \left(\frac{w - v_1}{v_2}\right) \cdot \left(\frac{w - v_2}{v_1}\right). \tag{4.28}$$

From (4.26) and (4.27), we see obviously —with $w < v_1 + v_2$ hence $\lambda < 1$ as it is assumed in the model— that the maximum heights of water levels in both tanks converge asymptotically and geometrically to a Zeno limit point at the limit $j \to \infty$.

In the system's model, when in either tanks the water level drops below the critical threshold, the inflow pipe switches to deliver the water to that tank, so the water level increases in one tank while it is decreasing in the other tank.

Denote ${}^*x_p$ to the initial highest water level at the initial switching time ${}^*\tau_0$, where ${}^*x_p$ is measured starting from a threshold level.

Let's consider the same scenario in the hybrid automaton of the model (Figure 4.4) where the initial mode is q_1 (filling tanks 1). In this case ${}^*x_p$ would be ${}^*x_p = {}^*x_{10} - r_1$, i.e. the initial highest peak will be recorded in tank 1.

In Figure 4.13 and Figure 4.14, we show two regions for the solution trajectory:

1. The violet regions (Figure 4.13) correspond to filling tank 2. In all these violet regions, the evolution of the state (${}^*x_1$,${}^*x_2$) on (x_1,x_2)-plane is given by

$$({}^*x_{1_{,i}} - r_1) + ({}^*x_{2_{,i}} - r_2) = k_1^i \cdot k_2^i \cdot {}^*x_p, \qquad k_1 = \frac{w - v_1}{v_2}, \;\; k_2 = \frac{w - v_2}{v_1}, \tag{4.29}$$

   where i corresponds to the i-th cycle on the cyclic path $C^{q_1} = \langle q_1; e_1, e_2; q_1\rangle$.

2. The green regions (Figure 4.14) correspond to filling tank 1. In all these green regions, the evolution of the state (${}^*x_1$,${}^*x_2$) on (x_1,x_2)-plane is given by

$$({}^*x_{1_{,i}} - r_1) + ({}^*x_{2_{,i}} - r_2) = k_1^i \cdot k_2^{i+1} \cdot {}^*x_p, \qquad k_1 = \frac{w - v_1}{v_2}, \;\; k_2 = \frac{w - v_2}{v_1}, \tag{4.30}$$

   where i corresponds to the i-th cycle on the cyclic path $C^{q_2} = \langle q_2; e_2, e_1; q_2\rangle$.

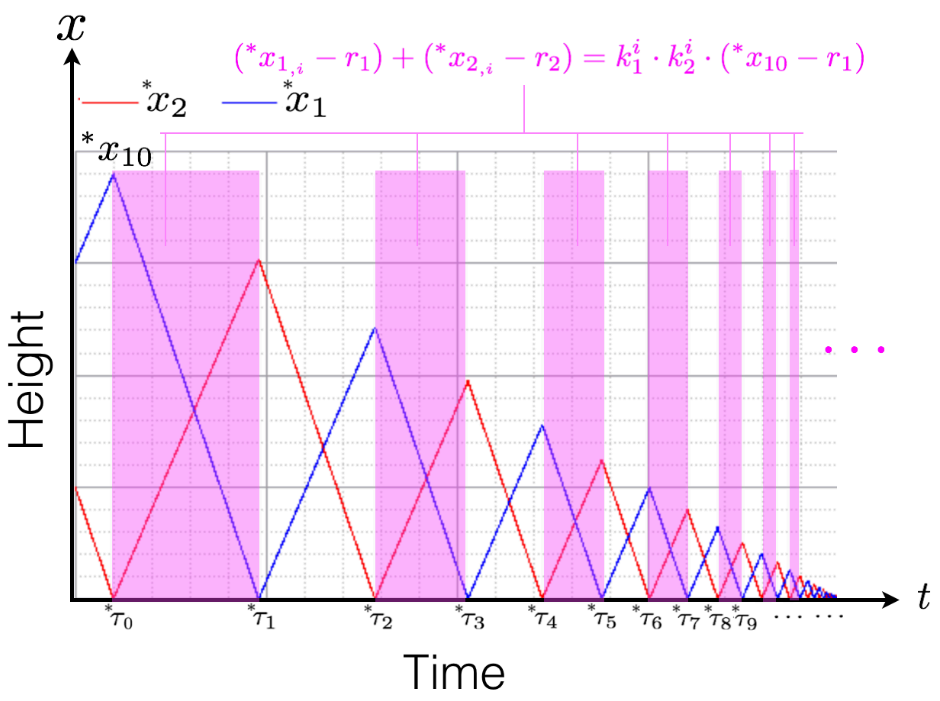


Figure 4.13: The time evolution of the water levels in the two tanks.

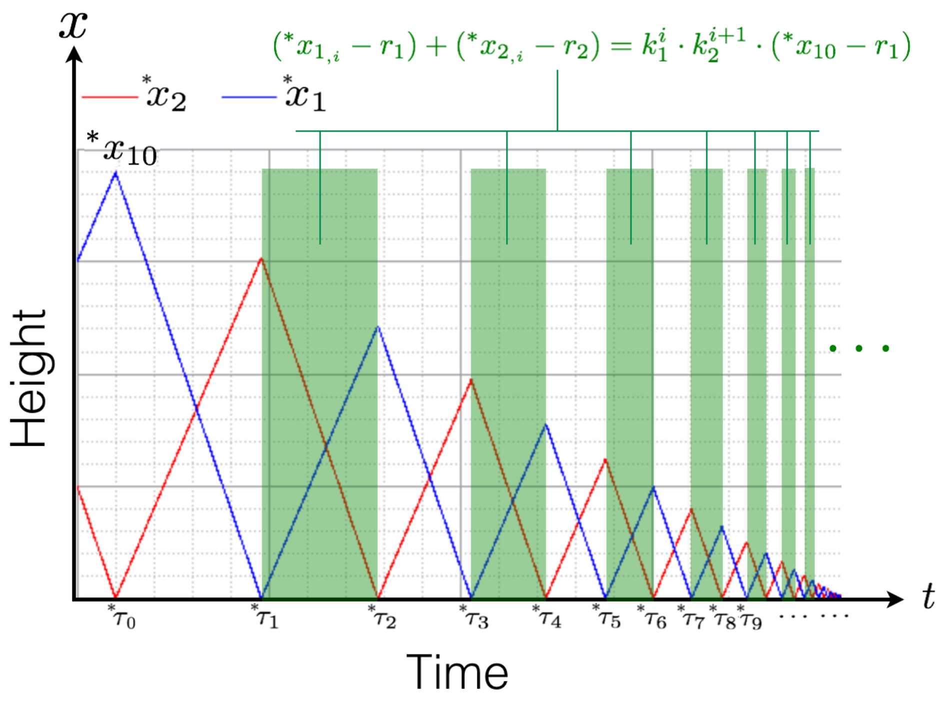


Figure 4.14: The time evolution of the water levels in the two tanks.

If we consider the opposite scenario where the hybrid automaton starts initially in mode q_2 (filling tanks 2), then ${}^*x_p$ would be ${}^*x_p = {}^*x_{20} - r_2$ (the initial highest peak would be recorded in tank 2). In this case, the violet regions would correspond to filling tank 1, thus increasing ${}^*x_1$ with the evolution relation given by

$$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^i \cdot {}^*x_p, \qquad k_1 = \frac{w - v_1}{v_2}, \quad k_2 = \frac{w - v_2}{v_1}, \tag{4.31}$$

where i corresponds to the i-th cycle on the cyclic path $C^{q_2} = \langle q_2; e_2, e_1; q_2 \rangle$, while the green regions would correspond to filling tank 2, thus increasing ${}^*x_2$ with the relation

$$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^{i+1} \cdot k_2^i \cdot {}^*x_p, \qquad k_1 = \frac{w - v_1}{v_2}, \quad k_2 = \frac{w - v_2}{v_1}, \tag{4.32}$$

where i corresponds to the i-th cycle on the cyclic path $C^{q_1} = \langle q_1; e_1, e_2; q_1 \rangle$.

Let's keep going with what is stated in hybrid automaton (Figure 4.4), i.e. the initial mode is q_1, and therefore the same as what is sketched in Figure 4.13 and Figure 4.14, hence (4.29) and (4.30) hold. Figure 4.15 shows the trajectory of the two tanks system on (x_1, x_2)-plane.

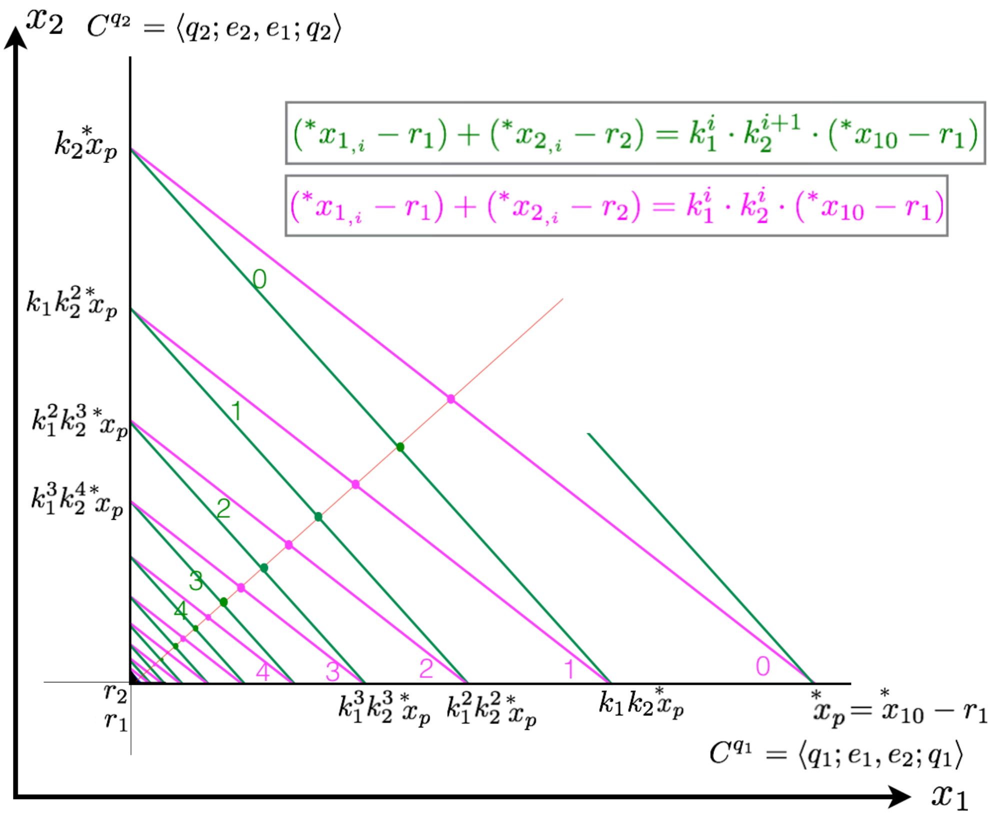

x_2
$C^{q_2} = \langle q_2; e_2, e_1; q_2 \rangle$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^{i+1} \cdot ({}^*x_{10} - r_1)$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^i \cdot ({}^*x_{10} - r_1)$
$k_2 {}^*x_p$
$k_1 k_2^2 {}^*x_p$
$k_1^2 k_2^3 {}^*x_p$
$k_1^3 k_2^4 {}^*x_p$
0
1
2
3
4
4
3
2
1
0
r_2
r_1
$k_1^3 k_2^3 {}^*x_p$
$k_1^2 k_2^2 {}^*x_p$
$k_1 k_2 {}^*x_p$
${}^*x_p = {}^*x_{10} - r_1$
$C^{q_1} = \langle q_1; e_1, e_2; q_1 \rangle$
x_1

Figure 4.15: The evolution of the water levels on (x_1,x_2)-plane.

We can observe that the hybrid solution trajectory in this example is a non-standard Cauchy sequence in a non-standard complete metric space. If we draw a line starting from the Zeno limit point (r_1, r_2) and intersecting each cycle i at a state $({}^*x_{1,i}, {}^*x_{2,i})$, we find that all the states $({}^*x_{1,i}, {}^*x_{2,i})$ on this line form also a non-standard Cauchy subsequence $\{{}^*x_j\}$, satisfying ${}^*x_{j+1} = {}^*\rho({}^*x_j)$ for all j, with ${}^*\rho$ being non-standard contraction map of fixed modulus $\beta \in (0, 1)$.

Consider for example the three states $({}^*x_{1,0}, {}^*x_{2,0})$, $({}^*x_{1,1}, {}^*x_{2,1})$, and $({}^*x_{1,2}, {}^*x_{2,2})$ picked when filling tank 2 (violet regions), as sketched in Figure 4.16. From (4.29) we have:

$$\begin{aligned}({}^*x_{2,0} - r_2) &= {}^*x_p - ({}^*x_{1,0} - r_1),\\ ({}^*x_{2,1} - r_2) &= k_1 k_2 {}^*x_p - ({}^*x_{1,1} - r_1),\\ ({}^*x_{2,2} - r_2) &= k_1^2 k_2^{2*} x_p - ({}^*x_{1,2} - r_1).\end{aligned} \tag{4.33}$$

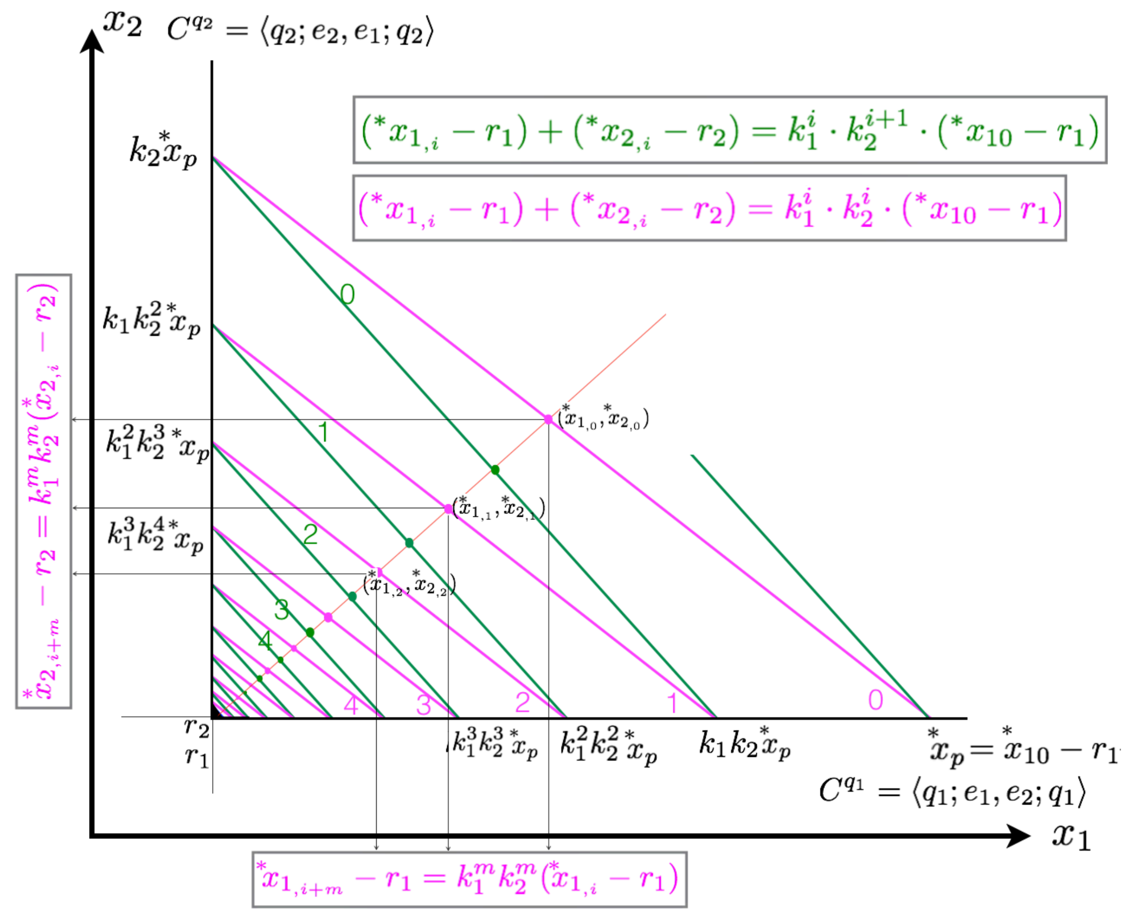

x_2 $C^{q_2} = \langle q_2; e_2, e_1; q_2\rangle$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^{i+1} \cdot ({}^*x_{10} - r_1)$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^i \cdot ({}^*x_{10} - r_1)$
$k_2 {}^*x_p$
$k_1 k_2^{2*} x_p$
$k_1^2 k_2^{3*} x_p$
$k_1^3 k_2^{4*} x_p$
${}^*x_{2,i+m} - r_2 = k_1^m k_2^m ({}^*x_{2,i} - r_2)$
$({}^*x_{1,0}, {}^*x_{2,0})$
$({}^*x_{1,1}, {}^*x_{2,1})$
$({}^*x_{1,2}, {}^*x_{2,2})$
0 1 2 3 4
4 3 2 1 0
r_2 r_1
$k_1^3 k_2^{3*} x_p$
$k_1^2 k_2^{2*} x_p$
$k_1 k_2 {}^*x_p$
${}^*x_p = {}^*x_{10} - r_1$
$C^{q_1} = \langle q_1; e_1, e_2; q_1\rangle$
x_1
${}^*x_{1,i+m} - r_1 = k_1^m k_2^m ({}^*x_{1,i} - r_1)$

Figure 4.16: The evolution of the water levels on (x_1,x_2)-plane.

In (4.33), we can see that:

- For the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$, with ${}^*x_{1_{,1}} - r_1 = k_1k_2({}^*x_{1_{,0}} - r_1)$ we have ${}^*x_{2_{,1}} - r_2 = k_1k_2({}^*x_{2_{,0}} - r_2)$.
- Similarly, for the pair of states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{1_{,2}} - r_1 = k_1k_2({}^*x_{1_{,1}} - r_1)$ we have ${}^*x_{2_{,2}} - r_2 = k_1k_2({}^*x_{2_{,1}} - r_2)$.
- Similarly, for the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{1_{,2}} - r_1 = k_1^2k_2^2({}^*x_{1_{,0}} - r_1)$ we have ${}^*x_{2_{,2}} - r_2 = k_1^2k_2^2({}^*x_{2_{,0}} - r_2)$.

Denote $\Psi = \{{}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})$ that lie on the same line which starts from the Zeno limit point (r_1,r_2) as sketched in Figure 4.16. We have ${}^*x_{1_{,i+m}} - r_1 = k_1^mk_2^m({}^*x_{1_{,i}} - r_1)$ and ${}^*x_{2_{,i+m}} - r_2 = k_1^mk_2^m({}^*x_{2_{,i}} - r_2)$ for all ${}^*x_{1_{,i}}, {}^*x_{2_{,i}} \in \Psi$.

Also, let ${}^*\rho_1, {}^*\rho_2$ be two maps such that ${}^*x_{1_{,i+1}} = {}^*\rho_1({}^*x_{1_{,i}})$, and ${}^*x_{2_{,i+1}} = {}^*\rho_2({}^*x_{2_{,i}})$, then it holds that:

1. For all ${}^*x_{2_{,i}} \in \Psi$ we have

$$d({}^*\rho_2({}^*x_{2_{,i}}), {}^*\rho_2({}^*x_{2_{,i+1}})) = k_1k_2d({}^*x_{2_{,i}}, {}^*x_{2_{,i+1}}). \tag{4.34}$$

2. For all ${}^*x_{1_{,i}} \in \Psi$ we have

$$d({}^*\rho_1({}^*x_{1_{,i}}), {}^*\rho_1({}^*x_{1_{,i+1}})) = k_1k_2d({}^*x_{1_{,i}}, {}^*x_{1_{,i+1}}). \tag{4.35}$$

Now consider the three states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$, $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$, and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$ picked when filling tank 1 (green regions), as sketched in Figure 4.17. From (4.31) we have:

$$\begin{aligned}
({}^*x_{2_{,0}} - r_2) &= k_2{}^*x_p - ({}^*x_{1_{,0}} - r_1),\\
({}^*x_{2_{,1}} - r_2) &= k_1k_2^{2*}x_p - ({}^*x_{1_{,1}} - r_1),\\
({}^*x_{2_{,2}} - r_2) &= k_1^2k_2^{3*}x_p - ({}^*x_{1_{,2}} - r_1).
\end{aligned} \tag{4.36}$$

In (4.36), when filling tank 1 (green regions) we will have the same as when when filling tank 2 (violet regions):

- For the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$, with ${}^*x_{1_{,1}} - r_1 = k_1k_2({}^*x_{1_{,0}} - r_1)$ we have ${}^*x_{2_{,1}} - r_2 = k_1k_2({}^*x_{2_{,0}} - r_2)$.
- Similarly, for the pair of states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{1_{,2}} - r_1 = k_1k_2({}^*x_{1_{,1}} - r_1)$ we have ${}^*x_{2_{,2}} - r_2 = k_1k_2({}^*x_{2_{,1}} - r_2)$.
- Similarly, for the pair of states $({}^*x_{1_{,0}}, {}^*x_{2_{,0}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{1_{,2}} - r_1 = k_1^2k_2^2({}^*x_{1_{,0}} - r_1)$ we have ${}^*x_{2_{,2}} - r_2 = k_1^2k_2^2({}^*x_{2_{,0}} - r_2)$.

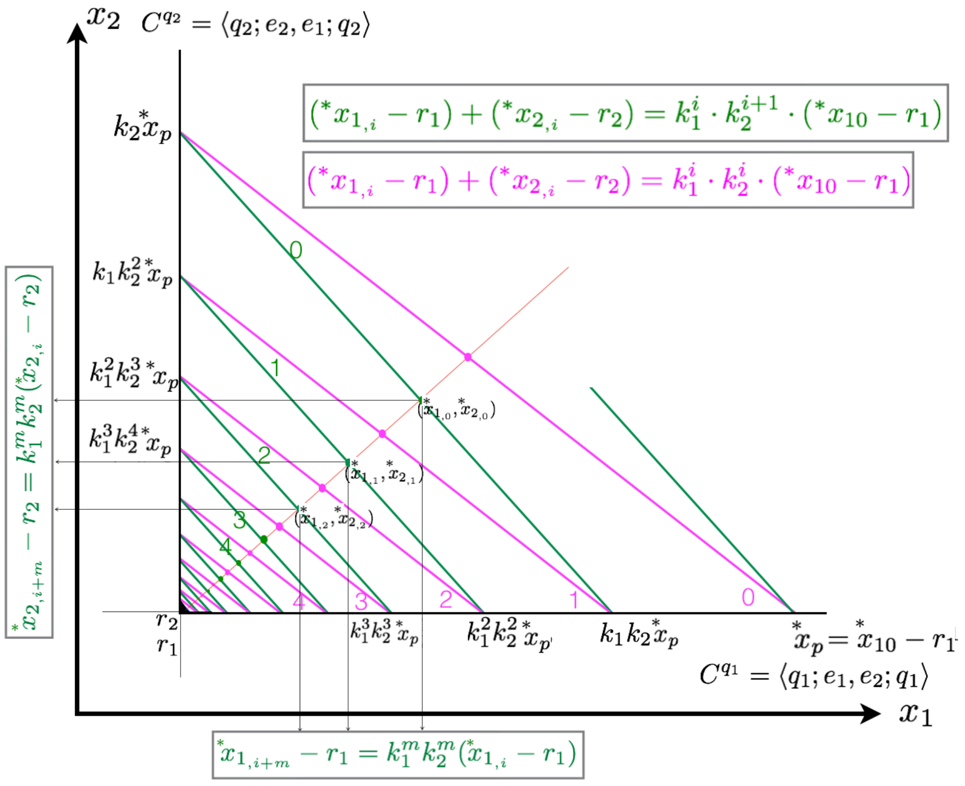

x_2
$C^{q_2} = \langle q_2; e_2, e_1; q_2\rangle$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^{i+1} \cdot ({}^*x_{10} - r_1)$
$({}^*x_{1,i} - r_1) + ({}^*x_{2,i} - r_2) = k_1^i \cdot k_2^i \cdot ({}^*x_{10} - r_1)$
$k_2 {}^*x_p$
$k_1 k_2^2 {}^*x_p$
$k_1^2 k_2^3 {}^*x_p$
$k_1^3 k_2^4 {}^*x_p$
${}^*x_{2,i+m} - r_2 = k_1^m k_2^m ({}^*x_{2,i} - r_2)$
$({}^*x_{1,0}, {}^*x_{2,0})$
$({}^*x_{1,1}, {}^*x_{2,1})$
$({}^*x_{1,2}, {}^*x_{2,2})$
0
1
2
3
4
r_2
r_1
$k_1^3 k_2^3 {}^*x_p$
$k_1^2 k_2^2 {}^*x_p$
$k_1 k_2 {}^*x_p$
${}^*x_p = {}^*x_{10} - r_1$
$C^{q_1} = \langle q_1; e_1, e_2; q_1\rangle$
x_1
${}^*x_{1,i+m} - r_1 = k_1^m k_2^m ({}^*x_{1,i} - r_1)$

Figure 4.17: The evolution of the water levels on (x_1,x_2)-plane.

Denote $\Psi = \{{}^*x_i = ({}^*x_{1,i}, {}^*x_{2,i})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1,i}, {}^*x_{2,i})$ that lie on the same line which starts from the Zeno limit point (r_1,r_2) as sketched in Figure 4.17. We have ${}^*x_{1,i+m} - r_1 = k_1^m k_2^m ({}^*x_{1,i} - r_1)$ and ${}^*x_{2,i+m} - r_2 = k_1^m k_2^m ({}^*x_{2,i} - r_2)$ for all ${}^*x_{1,i}, {}^*x_{2,i} \in \Psi$. Also, let ${}^*\rho_1, {}^*\rho_2$ be two maps such that ${}^*x_{1,i+1} = {}^*\rho_1({}^*x_{1,i})$, and ${}^*x_{2,i+1} = {}^*\rho_2({}^*x_{2,i})$, then it holds that:

1. For all ${}^*x_{2,i} \in \Psi$ we have
$$d({}^*\rho_2({}^*x_{2,i}), {}^*\rho_2({}^*x_{2,i+1})) = k_1 k_2 d({}^*x_{2,i}, {}^*x_{2,i+1}). \tag{4.37}$$
2. For all ${}^*x_{1,i} \in \Psi$ we have
$$d({}^*\rho_1({}^*x_{1,i}), {}^*\rho_1({}^*x_{1,i+1})) = k_1 k_2 d({}^*x_{1,i}, {}^*x_{1,i+1}). \tag{4.38}$$

Both ${}^*\rho_1$ and ${}^*\rho_2$ in (4.34), (4.35), (4.37), and (4.38) are contraction maps because $k_1 = \frac{w - v_1}{v_2} < 1$ and $k_2 = \frac{w - v_2}{v_1} < 1$ as it is assumed in the model with $w < v_1 + v_2$. Note that, we get the same results when the initial model of the system's hybrid automaton is q_2. Geometric-Zeno elimination involves enabling a transition from pre-Zeno to post-Zeno by integrating the system with dynamics (0,0) once ${}^*x_{1,i} \approx {}^*\rho_1({}^*x_{1,i})$ and ${}^*x_{2,i} \approx {}^*\rho_2({}^*x_{2,i})$, namely once $d({}^*x_{1,i}, {}^*x_{1,i+1}) \approx 0$ and $d({}^*x_{2,i}, {}^*x_{2,i+1}) \approx 0$. Note that, this naturally occurs when $(k_1 k_2)^m \to 0$ as $m \to \infty$.

Example 4.3 (Spiraling Piecewise Constant System) revisited: Consider again the Spiraling piecewise constant system model in Example 4.3.

The solution, as a function of time from ${}^*t_s \in \mathbb{T}_\partial$ to ${}^*t \in \mathbb{T}_\partial$ is given by

$$\begin{aligned}
{}^*x_1({}^*t - {}^*t_s) =& {}^*x_1({}^*t_s) + ({}^*t - {}^*t_s) \cdot (2\beta - \alpha), \\
{}^*x_2({}^*t - {}^*t_s) =& {}^*x_2({}^*t_s) - ({}^*t - {}^*t_s) \cdot (2\alpha + \beta), \\
{}^*x_3({}^*t - {}^*t_s) =& {}^*x_3({}^*t_s) + ({}^*t - {}^*t_s).
\end{aligned} \tag{4.39}$$

Figure 4.18 shows the time evolution of ${}^*x_1$, ${}^*x_2$, and ${}^*x_3$. As the switching between the four modes is based on the values of ${}^*x_1$, and ${}^*x_2$, then each time the state ${}^*x_2$ switches its domain at ${}^*\tau_i$ from negative to positive or vice-versa we can find out what time ${}^*t_{1_{\mp max,i}}$ the state ${}^*x_1$ reaches its positive/negative peak ${}^*x_{1_{\mp max,i}}$ by solving for the time at which the state ${}^*x_2$ is zero using (4.39). If ${}^*x_2$ switches its domain at the initial switching time ${}^*\tau_0$, then from (4.39) we have with ${}^*x_2({}^*t - {}^*t_s) = 0$, ${}^*t = {}^*\tau_0$, and ${}^*x_2({}^*t_s) = {}^*x_2(0)$; see Figure 4.18. This gives us:

$$\begin{aligned}
{}^*t_{1_{\mp max,0}} =& \frac{{}^*x_2(0)}{2\alpha + \beta}, \\
{}^*x_{1_{\mp max,0}} =& {}^*x_1({}^*\tau_0 - {}^*t_s) = {}^*x_1(0) + \frac{2\beta - \alpha}{2\alpha + \beta} \cdot {}^*x_2(0).
\end{aligned} \tag{4.40}$$

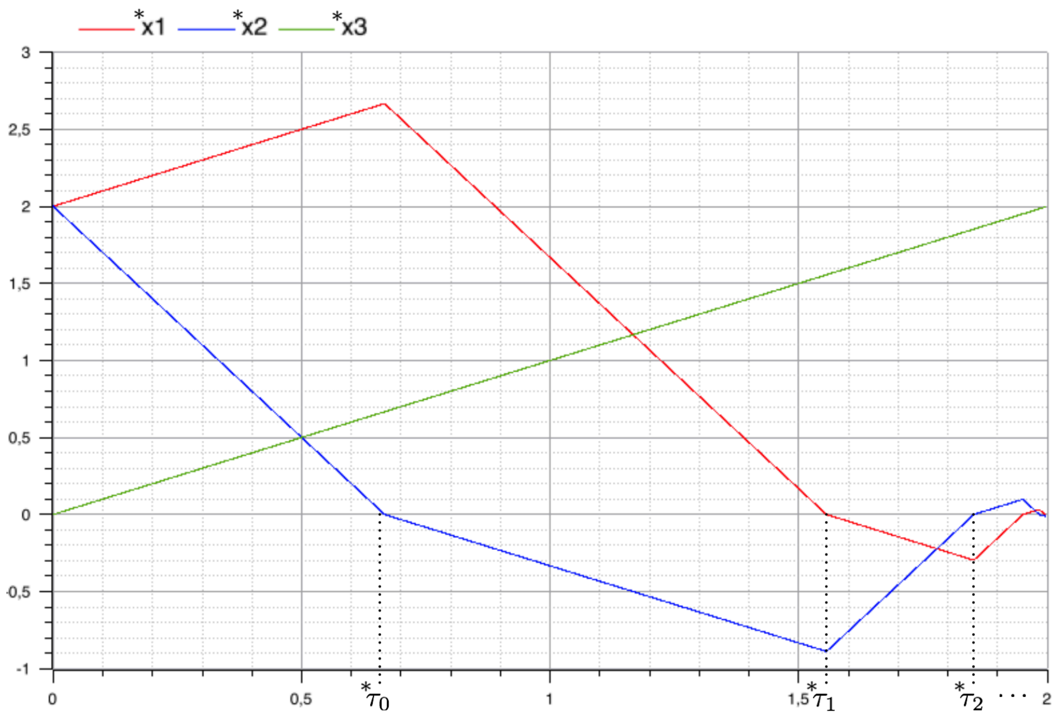


Figure 4.18: Example 4.3: The time evolution of the system.

For all other switching times ${}^*\tau_i$ when ${}^*x_2$ switches its domain, we have ${}^*x_2({}^*t - {}^*t_s) = 0$, ${}^*t_s = {}^*\tau_{i-1}$, ${}^*t = {}^*\tau_i$, ${}^*x_1({}^*t_s) = 0$, and ${}^*x_2({}^*t_s) = {}^*x_{2_{\mp max,i-1}}$. This gives us:

$$\begin{aligned}
{}^*t_{1_{\mp max,i}} &= {}^*\tau_i - {}^*\tau_{i-1} = \frac{{}^*x_{2_{\mp max,i-1}}}{2\alpha+\beta},\\
{}^*x_{1_{\mp max,i}} &= {}^*x_1({}^*\tau_i - {}^*\tau_{i-1}) = \frac{2\beta-\alpha}{2\alpha+\beta} \cdot {}^*x_{2_{\mp max,i-1}}.
\end{aligned} \tag{4.41}$$

Similarly, each time ${}^*x_1$ switches its domain at ${}^*\tau_i$ from negative to positive or vice-versa we can find out what time ${}^*t_{2_{\mp max,i}}$ the state ${}^*x_2$ reaches its positive/negative peak ${}^*x_{2_{\mp max,i}}$ by solving for the time at which the state ${}^*x_1$ in is zero using (4.39). If ${}^*x_1$ switches its domain at the initial switching time ${}^*\tau_0$, then from (4.39) we have with ${}^*x_1({}^*t - {}^*t_s) = 0$, ${}^*t = {}^*\tau_0$, and ${}^*x_1({}^*t_s) = {}^*x_1(0)$. This gives us:

$$\begin{aligned}
{}^*t_{2_{\mp max,0}} &= \frac{{}^*x_1(0)}{\alpha-2\beta},\\
{}^*x_{2_{\mp max,0}} &= {}^*x_2({}^*\tau_0 - {}^*t_s)\\
&= {}^*x_2(0) - \frac{2\alpha+\beta}{\alpha-2\beta} \cdot {}^*x_1(0).
\end{aligned} \tag{4.42}$$

For all other switching times ${}^*\tau_i$ when ${}^*x_1$ switches its domain we have ${}^*x_1({}^*t - {}^*t_s) = 0$, ${}^*t_s = {}^*\tau_{i-1}$, ${}^*t = {}^*\tau_i$, ${}^*x_2({}^*t_s) = 0$, and ${}^*x_1({}^*t_s) = {}^*x_{1_{\mp max,i-1}}$. This gives us

$$\begin{aligned}
{}^*t_{2_{\mp max,i}} &= {}^*\tau_i - {}^*\tau_{i-1}\\
&= \frac{{}^*x_{1_{\mp max,i-1}}}{\alpha-2\beta},\\
{}^*x_{2_{\mp max,i}} &= {}^*x_2({}^*\tau_i - {}^*\tau_{i-1})\\
&= \frac{2\alpha+\beta}{2\beta-\alpha} \cdot {}^*x_{1_{\mp max,i-1}}.
\end{aligned} \tag{4.43}$$

As it was shown in the hybrid automaton of the system (Figure 4.5), we have four locations q_1, q_2, q_3, q_4, where each cycle is completed after four zero-crossings: two zero-crossings by ${}^*x_1$ (changing its domain from positive to negative and from negative to positive), and two zero-crossings by ${}^*x_2$ (also changing its domain from positive to negative and from negative to positive).

Denote ${}^*x_p$ to the initial highest peak of either ${}^*x_1$ or ${}^*x_2$ at the initial switching time ${}^*\tau_0$.

Let's consider the case as sketched in Figure 4.18, that is, at the initial switching time ${}^*\tau_0$, it is ${}^*x_2$ who changes its domain from positive to negative. In this case, from (4.40) we have ${}^*x_p = {}^*x_{1_{+max,0}} = {}^*x_1(0) + \frac{2\beta-\alpha}{2\alpha+\beta} \cdot {}^*x_2(0)$, and (4.41), respectively (4.43), applies for computing —using ${}^*x_p$— the peaks ${}^*x_{1_{\mp max,i}}$, respectively ${}^*x_{2_{\mp max,i}}$, each time ${}^*x_2$, respectively ${}^*x_1$, changes its domain at ${}^*\tau_i$.

Starting from the initial switching time ${}^*\tau_0$, we have four zero crossings at ${}^*\tau_1$, ${}^*\tau_2$, ${}^*\tau_3$, and ${}^*\tau_4$ for completing the first cycle; (see Figure 4.18):

$$\begin{aligned}
&{}^*x_1({}^*\tau_1) = {}^*x_2({}^*\tau_2) = {}^*x_1({}^*\tau_3) = {}^*x_2({}^*\tau_4) = 0,\\
&{}^*x_2({}^*\tau_1) = {}^*x_{2_{-max,1}} = \ {}^*x_p \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=1,\beta=-1} < 0,\\
&{}^*x_1({}^*\tau_2) = {}^*x_{1_{-max,2}} = \ {}^*x_{2_{-max,1}} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=-1,\beta=-1}\\
&\qquad = {}^*x_p \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=1,\beta=-1} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=-1,\beta=-1} < 0,\\
&{}^*x_2({}^*\tau_3) = {}^*x_{2_{+max,3}} = \ {}^*x_{1_{-max,2}} \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=-1,\beta=1}\\
&\qquad = {}^*x_p \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=1,\beta=-1} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=-1,\beta=-1} \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=-1,\beta=1} > 0,\\
&{}^*x_1({}^*\tau_4) = {}^*x_{1_{+max,4}} = \ {}^*x_{2_{+max,4}} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=1,\beta=1}\\
&\qquad = {}^*x_p \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=1,\beta=-1} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=-1,\beta=-1} \cdot \left(\frac{2\alpha+\beta}{2\beta-\alpha}\right)_{\alpha=-1,\beta=1} \cdot \left(\frac{2\beta-\alpha}{2\alpha+\beta}\right)_{\alpha=1,\beta=1} > 0.
\end{aligned} \tag{4.44}$$

So in (4.44) we have

$$\begin{aligned}
{}^*x_1({}^*\tau_1) = 0,\ \ {}^*x_2({}^*\tau_1) &= \frac{-1}{3} \cdot {}^*x_p,\\
{}^*x_2({}^*\tau_2) = 0,\ \ {}^*x_1({}^*\tau_2) &= \frac{-1}{3} \cdot \frac{1}{3} \cdot {}^*x_p,\\
{}^*x_1({}^*\tau_3) = 0,\ \ {}^*x_2({}^*\tau_3) &= \frac{-1}{3} \cdot \frac{1}{3} \cdot \frac{-1}{3} \cdot {}^*x_p,\\
{}^*x_2({}^*\tau_4) = 0,\ \ {}^*x_1({}^*\tau_4) &= \frac{-1}{3} \cdot \frac{1}{3} \cdot \frac{-1}{3} \cdot \frac{1}{3} \cdot {}^*x_p.
\end{aligned} \tag{4.45}$$

Note that the initial cycle that started at ${}^*\tau_0$ is completed at ${}^*\tau_4$ because at ${}^*\tau_0$ we had ${}^*x_2({}^*\tau_0) = 0$, ${}^*x_1({}^*\tau_0) > 0$, and now at ${}^*\tau_4$ we have the same mode change happened at ${}^*\tau_0$, namely ${}^*x_2({}^*\tau_4) = 0$, ${}^*x_1({}^*\tau_4) > 0$. In general, for all the switching times ${}^*\tau_i$, a cycle that starts at ${}^*\tau_i$ terminates at ${}^*\tau_{i+4}$ where, depending on the initial switching at ${}^*\tau_0$, we have either $x_1({}^*\tau_{i+4}) = {}^*x_1({}^*\tau_i) = 0$ and ${}^*x_2({}^*\tau_{i+4}) = (\frac{1}{3})^4 \cdot {}^*x_2({}^*\tau_i)$, or ${}^*x_2({}^*\tau_{i+4}) = {}^*x_2({}^*\tau_i) = 0$ and ${}^*x_1({}^*\tau_{i+4}) = (\frac{1}{3})^4 \cdot {}^*x_1({}^*\tau_i)$. The relation with the initial peak ${}^*x_p$ at ${}^*\tau_0$ is the following: Whenever a cycle is completed at ${}^*\tau_i$ we have, again depending on the initial switching at ${}^*\tau_0$, either ${}^*x_1({}^*\tau_i) = {}^*x_1({}^*\tau_0) = 0$ and ${}^*x_2({}^*\tau_i) = (\frac{1}{3})^{4i} \cdot {}^*x_p$, or ${}^*x_2({}^*\tau_i) = {}^*x_2({}^*\tau_0) = 0$ and ${}^*x_1({}^*\tau_i) = (\frac{1}{3})^{4i} \cdot {}^*x_p$. We see now obviously why the solution forms a Cauchy sequence that converges to the state $(0, 0, {}^*x_3)$; whatever is the case of the initial switching at ${}^*\tau_0$, namely whether it is a zero-crossing on ${}^*x_1$ or ${}^*x_2$, the state $({}^*x_1, {}^*x_2)$ converges to (0,0) at the limit when $i \to \infty$.

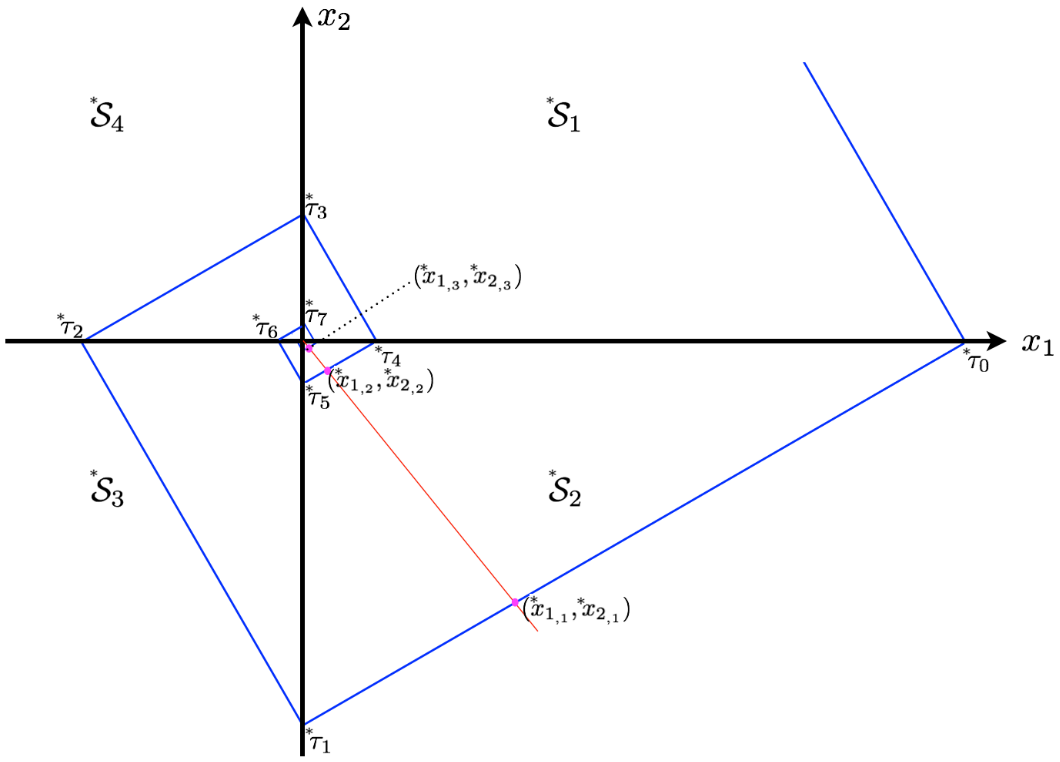


Figure 4.19: The evolution of the states x_1 and x_2 on $(x_1$,$x_2)$-plane.

Denote $^*\mathcal{S}_1$, $^*\mathcal{S}_2$, $^*\mathcal{S}_3$, and $^*\mathcal{S}_4$ to the invariants of the four discrete states q_1, q_2, q_3, and q_4 of the system's hybrid automaton in Figure 4.5. These invariants are given by

$$\begin{aligned} &^*\mathcal{S}_1 = \{^*x \in {}^*\mathbb{R}^3 : \ ^*x_1 > 0 \wedge {}^*x_2 > 0\}, && ^*\mathcal{S}_2 = \{^*x \in {}^*\mathbb{R}^3 : \ ^*x_1 > 0 \wedge {}^*x_2 < 0\}, \\ &^*\mathcal{S}_3 = \{^*x \in {}^*\mathbb{R}^3 : \ ^*x_1 < 0 \wedge {}^*x_2 < 0\}, && ^*\mathcal{S}_4 = \{^*x \in {}^*\mathbb{R}^3 : \ ^*x_1 < 0 \wedge {}^*x_2 > 0\}. \end{aligned} \tag{4.46}$$

Figure 4.19 shows the trajectory of the system's hybrid automaton on (x_1, x_2)-plane, where the evolution of the state variables $^*x_1$ and $^*x_2$ on (x_1, x_2)-plane is given, for each invariant, by

$$\begin{aligned} ^*x_{1,i} + {}^*x_{2,i} &= (\frac{1}{3^4})^i \cdot \text{sign}(^*x_{1,i}) \cdot {}^*x_p \quad \text{for} \quad (^*x_{1,i}, {}^*x_{2,i}) \in {}^*\mathcal{S}_1, \\ ^*x_{1,i} + {}^*x_{2,i} &= (\frac{1}{3})^i \cdot \text{sign}(^*x_{2,i}) \cdot {}^*x_p \quad \text{for} \quad (^*x_{1,i}, {}^*x_{2,i}) \in {}^*\mathcal{S}_2, \\ ^*x_{1,i} + {}^*x_{2,i} &= (\frac{1}{3^2})^i \cdot \text{sign}(^*x_{1,i}) \cdot {}^*x_p \quad \text{for} \quad (^*x_{1,i}, {}^*x_{2,i}) \in {}^*\mathcal{S}_3, \\ ^*x_{1,i} + {}^*x_{2,i} &= (\frac{1}{3^3})^i \cdot \text{sign}(^*x_{2,i}) \cdot {}^*x_p \quad \text{for} \quad (^*x_{1,i}, {}^*x_{2,i}) \in {}^*\mathcal{S}_4, \end{aligned} \tag{4.47}$$

where i corresponds to the i-th cycle on the cyclic path $c^{q_1} = \langle q_1; e_1, e_2, e_3, e_4; q_1\rangle$.

We can observe that the hybrid solution trajectory in this example is a non-standard Cauchy sequence in a non-standard complete metric space. On (x_1, x_2)-plane, if we draw a line starting from the Zeno limit point (0,0) and intersecting each cycle i at a state $({}^*x_{1,i}, {}^*x_{2,i})$, we find that all the states $({}^*x_{1,i}, {}^*x_{2,i})$, that lie on this line, form also a non-standard Cauchy subsequence $\{{}^*x_j\}$, satisfying ${}^*x_{j+1} = {}^*\rho({}^*x_j)$ for all j, with ${}^*\rho$ being non-standard contraction map of fixed modulus $\beta \in (0, 1)$. Consider for example the states $({}^*x_{1,1}, {}^*x_{2,1})$, $({}^*x_{1,2}, {}^*x_{2,2})$, and $({}^*x_{1,3}, {}^*x_{2,3})$ picked in ${}^*\mathcal{S}_2$, as sketched in Figure 4.19. From (4.47) we have:

$$\begin{aligned} {}^*x_{2,1} &= (\frac{1}{3}) \cdot \operatorname{sign}({}^*x_{2,1}) \cdot {}^*x_p - {}^*x_{1,1}, \\ {}^*x_{2,2} &= (\frac{1}{3})^2 \cdot \operatorname{sign}({}^*x_{2,2}) \cdot {}^*x_p - {}^*x_{1,2}, \\ {}^*x_{2,3} &= (\frac{1}{3})^3 \cdot \operatorname{sign}({}^*x_{2,3}) \cdot {}^*x_p - {}^*x_{1,3}. \end{aligned} \tag{4.48}$$

In (4.48), we can see that:

- For the pair of states $({}^*x_{1,1}, {}^*x_{2,1})$ and $({}^*x_{1,2}, {}^*x_{2,2})$, with ${}^*x_{1,2} = \frac{1}{3}{}^*x_{1,1}$ we have ${}^*x_{2,2} = \frac{1}{3}{}^*x_{2,1}$.
- Similarly, for the pair of states $({}^*x_{1,2}, {}^*x_{2,2})$ and $({}^*x_{1,3}, {}^*x_{2,3})$, with ${}^*x_{1,3} = \frac{1}{3}{}^*x_{1,2}$ we have ${}^*x_{2,3} = \frac{1}{3}{}^*x_{2,2}$.
- Similarly, for the pair of states $({}^*x_{1,1}, {}^*x_{2,1})$ and $({}^*x_{1,3}, {}^*x_{2,3})$, with ${}^*x_{1,3} = (\frac{1}{3})^2{}^*x_{1,1}$ we have ${}^*x_{2,3} = (\frac{1}{3})^2{}^*x_{2,1}$.

Denote $\Psi = \{{}^*x_i = ({}^*x_{1,i}, {}^*x_{2,i})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1,i}, {}^*x_{2,i})$ that lie on the same line which starts from the Zeno limit point (0,0) and intersects cycles i as sketched in Figure 4.19. We have ${}^*x_{2,i+m} = (\frac{1}{3})^{m*}x_{2,i}$, ${}^*x_{1,i+m} = (\frac{1}{3})^{m*}x_{1,i}$ for all ${}^*x_{1,i}, {}^*x_{2,i} \in \Psi$. Also, let ${}^*\rho_1, {}^*\rho_2$ be two maps such that ${}^*x_{1,i+1} = {}^*\rho_1({}^*x_{1,i})$, and ${}^*x_{2,i+1} = {}^*\rho_2({}^*x_{2,i})$, then it holds that:

1. For all ${}^*x_{2,i} \in \Psi$ we have

$$d({}^*\rho_2({}^*x_{2,i}), {}^*\rho_2({}^*x_{2,i+1})) = \frac{1}{3}d({}^*x_{2,i}, {}^*x_{2,i+1}). \tag{4.49}$$

2. For all ${}^*x_{1,i} \in \Psi$ we have

$$d({}^*\rho_1({}^*x_{1,i}), {}^*\rho_1({}^*x_{1,i+1})) = \frac{1}{3}d({}^*x_{1,i}, {}^*x_{1,i+1}). \tag{4.50}$$

Both ${}^*\rho_1$ and ${}^*\rho_2$ are contraction maps because $\beta = \frac{1}{3} \in (0, 1)$.

Now consider the states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$, $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, and $({}^*x_{1_{,3}}, {}^*x_{2_{,3}})$ picked in $\mathcal{S}_3$, as sketched in Figure 4.20. From (4.47) we have:

$$\begin{aligned} {}^*x_{2_{,1}} &= (\frac{1}{3^2}) \cdot \text{sign}({}^*x_{2_{,1}}) \cdot {}^*x_p - {}^*x_{1_{,1}}, \\ {}^*x_{2_{,2}} &= (\frac{1}{3^2})^2 \cdot \text{sign}({}^*x_{2_{,2}}) \cdot {}^*x_p - {}^*x_{1_{,2}}, \\ {}^*x_{2_{,3}} &= (\frac{1}{3^2})^3 \cdot \text{sign}({}^*x_{2_{,3}}) \cdot {}^*x_p - {}^*x_{1_{,3}}. \end{aligned} \tag{4.51}$$

In (4.51), we can see that:

- For the pair of states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$ and $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$, with ${}^*x_{1_{,2}} = \frac{1}{3^2}{}^*x_{1_{,1}}$ we have ${}^*x_{2_{,2}} = \frac{1}{3^2}{}^*x_{2_{,1}}$.
- Similarly, for the pair of states $({}^*x_{1_{,2}}, {}^*x_{2_{,2}})$ and $({}^*x_{1_{,3}}, {}^*x_{2_{,3}})$, with ${}^*x_{1_{,3}} = \frac{1}{3^2}{}^*x_{1_{,2}}$ we have ${}^*x_{2_{,3}} = \frac{1}{3^2}{}^*x_{2_{,2}}$.
- Similarly, for the pair of states $({}^*x_{1_{,1}}, {}^*x_{2_{,1}})$ and $({}^*x_{1_{,3}}, {}^*x_{2_{,3}})$, with ${}^*x_{1_{,3}} = (\frac{1}{3^2})^2{}^*x_{1_{,1}}$ we have ${}^*x_{2_{,3}} = (\frac{1}{3^2})^2{}^*x_{2_{,1}}$.

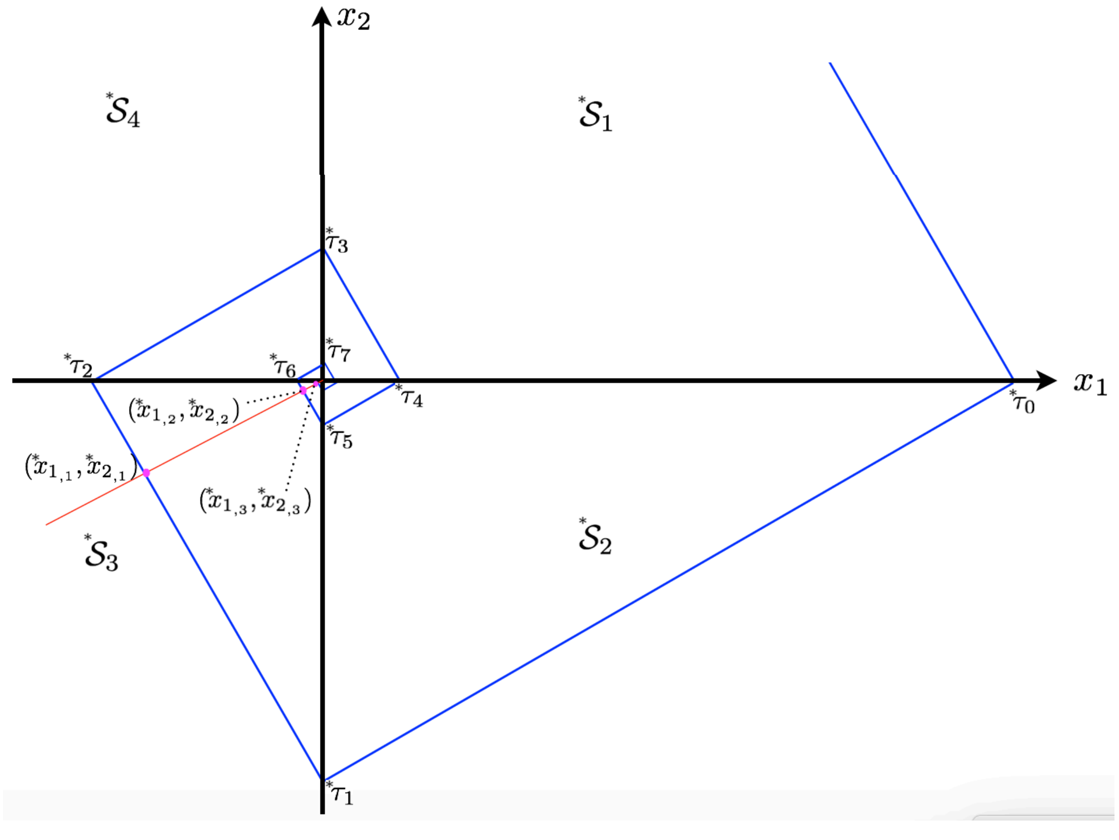


Figure 4.20: The evolution of the states x_1 and x_2 on (x_1,x_2)-plane.

Denote $\Psi = \{{}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}})$ in ${}^*\mathcal{S}_3$ and that lie on the same line which starts from the Zeno limit point (0,0) and intersects cycles i as sketched in Figure 4.20. We have ${}^*x_{2_{,i+m}} = (\frac{1}{3^2})^{m*}x_{2_{,i}}$, ${}^*x_{1_{,i+m}} = (\frac{1}{3^2})^{m*}x_{1_{,i}}$ for all ${}^*x_{1_{,i}}, {}^*x_{2_{,i}} \in \Psi$. Also, let ${}^*\rho_1, {}^*\rho_2$ be two maps such that ${}^*x_{1_{,i+1}} = {}^*\rho_1({}^*x_{1_{,i}})$, and ${}^*x_{2_{,i+1}} = {}^*\rho_2({}^*x_{2_{,i}})$, then it holds that:

1. For all ${}^*x_{2_{,i}} \in \Psi$ we have

$$d({}^*\rho_2({}^*x_{2_{,i}}), {}^*\rho_2({}^*x_{2_{,i+1}})) = \frac{1}{3^2} d({}^*x_{2_{,i}}, {}^*x_{2_{,i+1}}). \tag{4.52}$$

2. For all ${}^*x_{1_{,i}} \in \Psi$ we have

$$d({}^*\rho_1({}^*x_{1_{,i}}), {}^*\rho_1({}^*x_{1_{,i+1}})) = \frac{1}{3^2} d({}^*x_{1_{,i}}, {}^*x_{1_{,i+1}}). \tag{4.53}$$

Both ${}^*\rho_1$ and ${}^*\rho_2$ are contraction maps because $\beta = \frac{1}{3^2} \in (0, 1)$.

The same analysis applies in the invariants ${}^*\mathcal{S}_1$ and ${}^*\mathcal{S}_4$, where in ${}^*\mathcal{S}_4$ the modulus β for both ρ_1 and ρ_2 is $\beta = \frac{1}{3^3}$, and in ${}^*\mathcal{S}_1$ the modulus β for both ρ_1 and ρ_2 is $\beta = \frac{1}{3^4}$.

In the following, we prove that there exists also the solution on the ${}^*x_3$-plan forms also a Cauchy sequence and converges to a Zeno limit point.

Denote ${}^*x_{30}$ to the state ${}^*x_3$ at the initial switching time ${}^*\tau_0$, that is ${}^*x_{30} = {}^*x_3({}^*\tau_0)$, and again ${}^*x_p$ to the initial highest peak of either ${}^*x_1$ or ${}^*x_2$ at the initial switching time ${}^*\tau_0$.

If ${}^*x_2$ switches its domain at the initial switching time ${}^*\tau_0$, then from (4.39) (see also Figure 4.18) we have

$${}^*x_{30} = {}^*x_3({}^*\tau_0) = {}^*x_3(0) + \frac{{}^*x_2(0)}{2\alpha + \beta}, \tag{4.54}$$

otherwise if it is ${}^*x_1$ who switches its domain at ${}^*\tau_0$, then from (4.39) we have

$${}^*x_{30} = {}^*x_3({}^*\tau_0) = {}^*x_3(0) + \frac{{}^*x_1(0)}{\alpha - 2\beta}. \tag{4.55}$$

For all other switching times ${}^*\tau_i$ if ${}^*x_2$ switches its domain at ${}^*\tau_i$, then from(4.39) ${}^*x_3({}^*\tau_i)$ is give by

$${}^*x_3({}^*\tau_i) = {}^*x_3({}^*\tau_{i-1}) + \frac{{}^*x_2({}^*\tau_{i-1})}{2\alpha + \beta}, \tag{4.56}$$

otherwise if it is ${}^*x_1$ who switches its domain at ${}^*\tau_i$, then from(4.31) ${}^*x_3({}^*\tau_i)$ is give by

$${}^*x_3({}^*\tau_i) = {}^*x_3({}^*\tau_{i-1}) + \frac{{}^*x_1({}^*\tau_{i-1})}{\alpha - 2\beta}. \tag{4.57}$$

where in (4.54) and (4.55) the values of α and β are evaluated for ${}^*t \in [0, {}^*\tau_0)$, and in (4.56) and (4.57) the values of α and β are evaluated for ${}^*t \in ({}^*\tau_{i-1}, {}^*\tau_i)$.

Let's consider the case as sketched in Figure 4.18, that is, at the initial switching time ${}^*\tau_0$, it is ${}^*x_2$ who changes its domain from positive to negative. In this case, (4.54) applies for computing ${}^*x_{30} = {}^*x_3({}^*\tau_0)$, and ${}^*x_p$ is given by ${}^*x_p = {}^*x_1({}^*\tau_0) = {}^*x_1(0) + \frac{{}^*x_2(0)}{2\alpha+\beta}$. From (4.54), (4.56), and (4.57) we have

$$\begin{aligned}
{}^*x_3({}^*\tau_0) &= {}^*x_{30} = {}^*x_3(0) + \frac{{}^*x_2(0)}{2\alpha+\beta} = {}^*x_3(0) + \frac{{}^*x_2(0)}{3}, \\
{}^*x_3({}^*\tau_1) &= {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{\alpha-2\beta} = {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{3}, \\
{}^*x_3({}^*\tau_2) &= {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{\alpha-2\beta} + \frac{{}^*x_2({}^*\tau_1)}{2\alpha+\beta} = {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{3} - \frac{{}^*x_2({}^*\tau_1)}{3}, \\
{}^*x_3({}^*\tau_3) &= {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{\alpha-2\beta} + \frac{{}^*x_2({}^*\tau_1)}{2\alpha+\beta} + \frac{{}^*x_1({}^*\tau_2)}{\alpha-2\beta} = {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{3} - \frac{{}^*x_2({}^*\tau_1)}{3} - \frac{{}^*x_1({}^*\tau_2)}{3}, \\
{}^*x_3({}^*\tau_4) &= {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{\alpha-2\beta} + \frac{{}^*x_2({}^*\tau_1)}{2\alpha+\beta} + \frac{{}^*x_1({}^*\tau_2)}{\alpha-2\beta} + \frac{{}^*x_2({}^*\tau_3)}{2\alpha+\beta} \\
&= {}^*x_{30} + \frac{{}^*x_1({}^*\tau_0)}{3} - \frac{{}^*x_2({}^*\tau_1)}{3} - \frac{{}^*x_1({}^*\tau_2)}{3} + \frac{{}^*x_2({}^*\tau_3)}{3}.
\end{aligned} \tag{4.58}$$

By using (4.45), and with ${}^*x_p = {}^*x_1({}^*\tau_0) = {}^*x_1(0) + \frac{{}^*x_2(0)}{3}$ we can re-write (4.58) as following

$$\begin{aligned}
{}^*x_3({}^*\tau_0) &= {}^*x_{30} = {}^*x_3(0) + \frac{{}^*x_2(0)}{3} = {}^*x_3(0) - {}^*x_1(0) + {}^*x_p, \\
{}^*x_3({}^*\tau_1) &= {}^*x_3(0) - {}^*x_1(0) + {}^*x_p + \frac{{}^*x_p}{3}, \\
{}^*x_3({}^*\tau_2) &= {}^*x_3(0) - {}^*x_1(0) + {}^*x_p + \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2}, \\
{}^*x_3({}^*\tau_3) &= {}^*x_3(0) - {}^*x_1(0) + {}^*x_p + \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3}, \\
{}^*x_3({}^*\tau_4) &= {}^*x_3(0) - {}^*x_1(0) + {}^*x_p + \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3} + \frac{{}^*x_p}{3^4},
\end{aligned} \tag{4.59}$$

so in general at the switching time ${}^*\tau_i$ the state ${}^*x_3({}^*\tau_i)$ is given by

$${}^*x_3({}^*\tau_i) = {}^*x_3(0) - {}^*x_1(0) + \sum_{k=0}^{i} \frac{{}^*x_p}{3^k}. \tag{4.60}$$

The geometric series in (4.60) is convergent because $\frac{1}{3} < 1$. At the limit, when $i \to \infty$, the time interval ${}^*\tau_i - {}^*\tau_{i-1} = \frac{{}^*x_p}{3^i}$ —between two successive switching times ${}^*\tau_{i-1}$ and ${}^*\tau_i$— shrinks to 0. The geometric series (4.60) converges then to a geometric-Zeno state given by

$$\begin{aligned}
{}^*x_3({}^*\tau_\infty) = {}^*x_3(0) - {}^*x_1(0) + \lim_{i\to\infty} \sum_{k=0}^{i} \frac{{}^*x_p}{3^i} &= {}^*x_3(0) - {}^*x_1(0) + \frac{{}^*x_p}{1-\frac{1}{3}}, \\
{}^*x_3(\tau_\infty) &= {}^*x_3(0) - {}^*x_1(0) + \frac{3}{2}{}^*x_p.
\end{aligned} \tag{4.61}$$

With (0.0) being the Zeno state on (x_1, x_2)-plane, the solution of Example 4.3 converges asymptotically to a Zeno limit point given by $(0,0,{}^*x_3(0) - {}^*x_1(0) + \frac{3}{2}{}^*x_p)$ with ${}^*x_p = {}^*x_1(0) + \frac{{}^*x_2(0)}{3}$.

As mentioned above, the hybrid automaton of this system contains four locations q_1, q_2, q_3, q_4. Each cycle is completed after four zero-crossings, namely from ${}^*\tau_{i-4}$ to ${}^*\tau_i$.

For example, the initial cycle that started at ${}^*\tau_0$ is completed at ${}^*\tau_4$ because at ${}^*\tau_0$ we have ${}^*x_2({}^*\tau_0) = 0,\ {}^*x_1({}^*\tau_0) > 0$, and at ${}^*\tau_4$ we have the same mode change happened at ${}^*\tau_0$, namely ${}^*x_2({}^*\tau_4) = 0,\ {}^*x_1({}^*\tau_4) > 0$.

From (4.59) we can see that, during the initial first cycle from ${}^*\tau_0$ to ${}^*\tau_4$, the state ${}^*x_3$ has evolved

$$ {}^*x_3({}^*\tau_4) - {}^*x_3({}^*\tau_0) = \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3} + \frac{{}^*x_p}{3^4}. \tag{4.62} $$

Denote $\Delta^*x_{3_{,i}}$ to the evolution of ${}^*x_3$ at each cycle. For all the cycles we have:

$$ \begin{aligned}
\text{Cycle 0:}\quad \Delta^*x_{3_{,0}} &= \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3} + \frac{{}^*x_p}{3^4}, \\
\text{Cycle 1:}\quad \Delta^*x_{3_{,1}} &= \frac{{}^*x_p}{3^5} + \frac{{}^*x_p}{3^6} + \frac{{}^*x_p}{3^7} + \frac{{}^*x_p}{3^8} \\
&= \frac{1}{3^4}\Delta^*x_{3_{,0}}, \\
\text{Cycle 2:}\quad \Delta^*x_{3_{,2}} &= \frac{{}^*x_p}{3^9} + \frac{{}^*x_p}{3^{10}} + \frac{{}^*x_p}{3^{11}} + \frac{{}^*x_p}{3^{12}} \\
&= \frac{1}{3^4}\Delta^*x_{3_{,1}}, \\
\text{Cycle 3:}\quad \Delta^*x_{3_{,3}} &= \frac{{}^*x_p}{3^{13}} + \frac{{}^*x_p}{3^{14}} + \frac{{}^*x_p}{3^{15}} + \frac{{}^*x_p}{3^{16}} \\
&= \frac{1}{3^4}\Delta^*x_{3_{,2}}, \\
&\vdots \\
\text{Cycle i:}\quad \Delta^*x_{3_{,i}} &= \frac{{}^*x_p}{3^{4i+1}} + \frac{{}^*x_p}{3^{4i+2}} + \frac{{}^*x_p}{3^{4i+3}} + \frac{{}^*x_p}{3^{4i+4}} \\
&= \frac{1}{3^4}\Delta^*x_{3_{,i-1}} \\
&= \frac{1}{3^{4i}}\Delta^*x_{3_{,0}}.
\end{aligned} \tag{4.63} $$

Consider the three states ${}^*x_3({}^*\tau_0)$, ${}^*x_3({}^*\tau_4)$, and ${}^*x_3({}^*\tau_8)$ that lie on the same line starting from the Zeno state ${}^*x_3({}^*\tau_\infty)$; see Figure 4.19 with ${}^*x_3$ plane being vertical eye view plane. We have:

$$\begin{aligned}
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_0) &= \frac{{}^*x_p}{3} + \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3} + \frac{{}^*x_p}{3^4} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_4) &= \frac{{}^*x_p}{3^5} + \frac{{}^*x_p}{3^6} + \frac{{}^*x_p}{3^7} + \frac{{}^*x_p}{3^8} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
&= \frac{1}{3^4}({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_0)), \\
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_8) &= \frac{{}^*x_p}{3^9} + \frac{{}^*x_p}{3^{10}} + \frac{{}^*x_p}{3^{11}} + \frac{{}^*x_p}{3^{12}} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
&= \frac{1}{3^4}({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_4)), \\
&= (\frac{1}{3^4})^2({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_0)).
\end{aligned} \tag{4.64}$$

The same, if we consider the three states ${}^*x_3({}^*\tau_1)$, ${}^*x_3({}^*\tau_5)$, and ${}^*x_3({}^*\tau_9)$ that lie on the same line starting from the Zeno state ${}^*x_3({}^*\tau_\infty)$, we have

$$\begin{aligned}
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_1) &= \frac{{}^*x_p}{3^2} + \frac{{}^*x_p}{3^3} + \frac{{}^*x_p}{3^4} + \frac{{}^*x_p}{3^5} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_5) &= \frac{{}^*x_p}{3^6} + \frac{{}^*x_p}{3^7} + \frac{{}^*x_p}{3^8} + \frac{{}^*x_p}{3^9} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
&= \frac{1}{3^4}({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_1)), \\
{}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_9) &= \frac{{}^*x_p}{3^{10}} + \frac{{}^*x_p}{3^{11}} + \frac{{}^*x_p}{3^{12}} + \frac{{}^*x_p}{3^{13}} + \cdots + \frac{{}^*x_p}{3^\infty}, \\
&= \frac{1}{3^4}({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_5)), \\
&= (\frac{1}{3^4})^2({}^*x_3({}^*\tau_\infty) - {}^*x_3({}^*\tau_1)).
\end{aligned} \tag{4.65}$$

The same also results when we consider the three states ${}^*x_3({}^*\tau_2)$, ${}^*x_3({}^*\tau_6)$, and ${}^*x_3({}^*\tau_{10})$ that lie on the same line starting from the Zeno state ${}^*x_3({}^*\tau_\infty)$, and also for the three states ${}^*x_3({}^*\tau_3)$, ${}^*x_3({}^*\tau_7)$, and ${}^*x_3({}^*\tau_{11})$ that lie on the same line starting from the Zeno state ${}^*x_3({}^*\tau_\infty)$.

Denote $\Psi = \{{}^*x_i = ({}^*x_{3_{,i}}, {}^*x_{2_{,i}}, {}^*x_{3_{,i}})\}$ to the sequence of states ${}^*x_i = ({}^*x_{1_{,i}}, {}^*x_{2_{,i}}, {}^*x_{3_{,i}})$ that lie on the same line which starts from the Zeno limit point $(0, 0, {}^*x_{3_{,i}})$ and intersects all the cycles i on (x_1, x_2, x_3)-plane, and let ${}^*\rho_3$ be a map such that ${}^*x_{3_{,i+1}} = {}^*\rho_3({}^*x_{3_{,i}})$, then it holds that for all ${}^*x_{3_{,i}} \in \Psi$ we have

$$d({}^*\rho_3({}^*x_{3_{,i}}), {}^*\rho_3({}^*x_{3_{,i+1}})) = \frac{1}{3^4} d({}^*x_{3_{,i}}, {}^*x_{3_{,i+1}}). \tag{4.66}$$

The map ${}^*\rho_3$ is a contraction map because $\beta = \frac{1}{3^4} \in (0, 1)$.

Geometric-Zeno elimination involves enabling a transition from pre-Zeno to post-Zeno by integrating the system with dynamics (0,0,0) once ${}^*x_{1_{,i}} \approx {}^*\rho_1({}^*x_{1_{,i}})$, ${}^*x_{2_{,i}} \approx {}^*\rho_2({}^*x_{2_{,i}})$, and ${}^*x_{3_{,i}} \approx {}^*\rho_3({}^*x_{3_{,i}})$, namely once $d({}^*x_{1_{,i}}, {}^*x_{1_{,i+1}}) \approx 0$, $d({}^*x_{2_{,i}}, {}^*x_{2_{,i+1}}) \approx 0$, and $d({}^*x_{3_{,i}}, {}^*x_{3_{,i+1}}) \approx 0$.

4.3 Simulation Results

As a simulation environment for the prototype implementation of our Zeno-free simulation technique we have used Matlab/Simulink. We have implemented two Zeno-free simulators in both Matlab and Simulink environments. The motivation is to provide a correct Zeno-free simulation for hybrid systems models exhibiting geometric-Zeno behavior. The Matlab implementation [110] includes a Zeno-free simulator written in Matlab code, while the Simulink implementation [111] includes the basic Simulink library blocks that allow for Zeno-free simulation of Zeno models written in Simulink.

Figure 4.21, Figure 4.22, and Figure 4.23 show the Zeno-free simulation results for the bouncing ball system in Example 4.1 with the data set: $\lambda = 0.5$, $g = 9.81$, and $x(0) = [x_1(0)\ x_2(0)]^T = [1\ 0]^T$. The simulation time is set to 2.0. During the simulation, the ball's velocity x_2 decreases due to the energy loss through impact. As a consequence, more and more collisions are triggered and the time interval between two consecutive collisions keeps shrinking. One cyclic path $C^{q_1} = \langle q_1; e_1; q_1 \rangle$ was detected by the simulator when executing the hybrid automaton of the system in Figure 4.1, where $q_1 = \mathit{Fly}$ and $e_1 = \mathit{Bounce}$. The first cycle was detected at the time instant $t = 0.9031$. The solution converged of to the Zeno limit point at the time instant 1.3549. The dense points near the time 1.3549 indicates that more and more computation steps are taken near the estimated Zeno limit point. The simulation closely approaches the Zeno point before the behavior of the ball automatically switches to what the sliding dynamics ($\dot{x}_1 = 0, \dot{x}_2 = 0$) specifies. Therefore, the simulation does not halt, freely moving beyond the geometric-Zeno point in a manner consistent with physical reality.

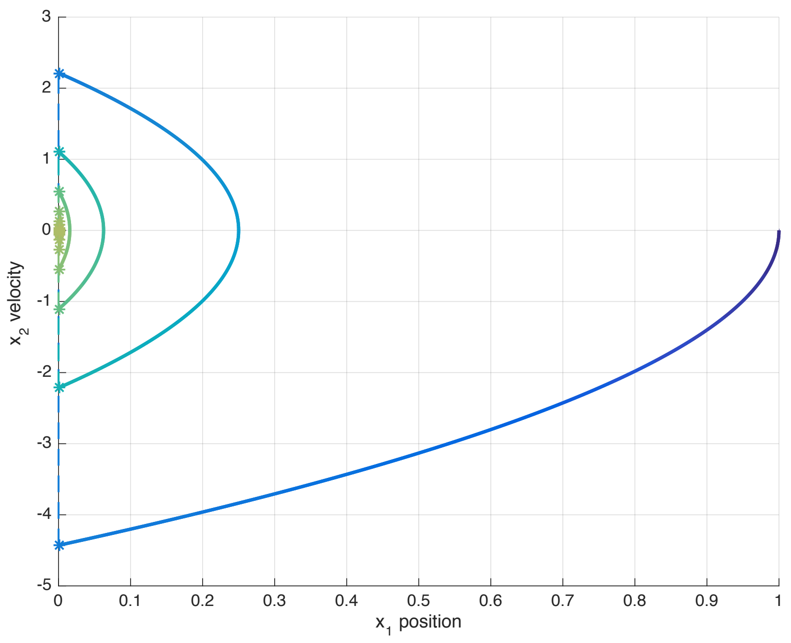


Figure 4.21: Zeno-free simulation of the bouncing ball model: x_2 versus x_1.

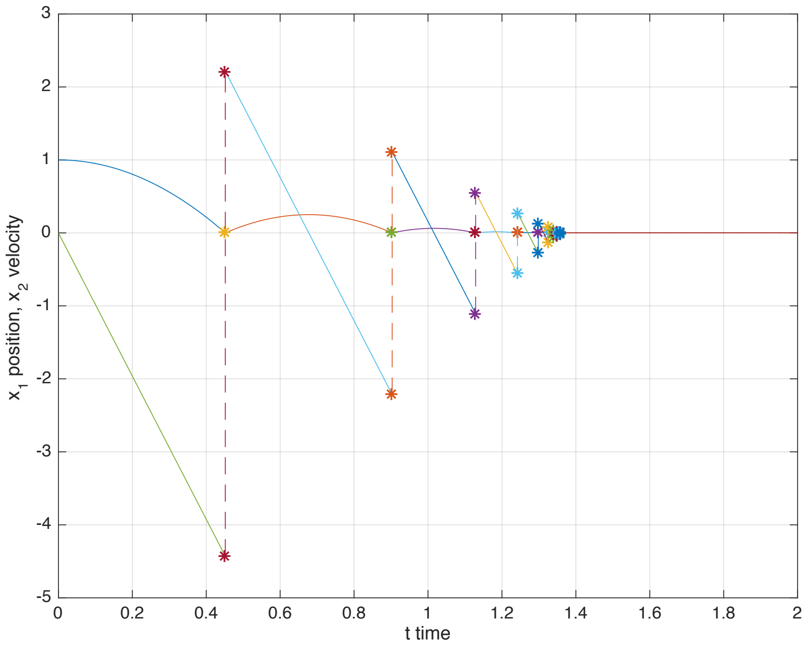


Figure 4.22: Zeno-free simulation of the bouncing ball model: The time evolution of the position x_1 and velocity x_2 of the bouncing ball.

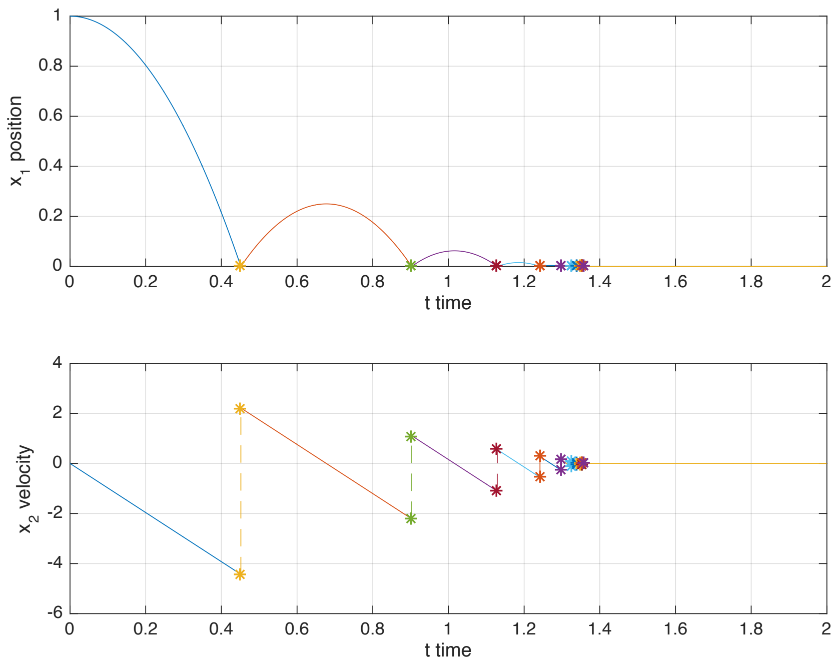


Figure 4.23: Zeno-free simulation of the bouncing ball model. Up: The time evolution of the position x_1. Down: The time evolution of the velocity x_2.

Figure 4.24, Figure 4.25, and Figure 4.26 show the Zeno-free simulation results for the two tanks system in Example 4.2 with the data set: $w = 1.8$, $v_1 = 1$, $v_2 = 1$, $r_1 = 5$, $r_2 = 5$, and $x(0) = [x_1(0)\ x_2(0)]^T = [8\ 6]^T$. The simulation time is set to 25.0.

Two cyclic paths, $C^{q_1} = \langle q_1; e_1, e_2; q_1 \rangle$ and $C^{q_2} = \langle q_2; e_2, e_1; q_2 \rangle$, were detected by the simulator when executing the hybrid automaton of the system in Figure 4.4, where e_1 is the transition from q_1 to q_2 guarded by the guard $\mathcal{G}(e_1) = \{x(t) \in \mathbb{R}^2 : x_2(t) \leq r_2\}$, and e_2 is the transition from q_2 to q_1 guarded by the guard $\mathcal{G}(e_2) = \{x(t) \in \mathbb{R}^2 : x_1(t) \leq r_1\}$.

The initial cycle 0 on the cyclic path $C^{q_1} = \langle q_1; e_1, e_2; q_1 \rangle$ was detected at the time instant $t = 7.8402$, and the initial cycle 0 on the cyclic path $C^{q_2} = \langle q_2; e_2, e_1; q_2 \rangle$ was detected at the time instant $t = 10.2723$.

The convergence of solution to the Zeno limit point was estimated to be at (0,0,2). Again the dense points near the Zeno time 20.0018 indicates that more and more computation steps were taken near the Zeno limit point. The simulation closely approaches the Zeno point before the simulator automatically switches to integrate the system with the sliding dynamics $(\dot{x}_1(t) = 0, \dot{x}_2(t) = 0)$, (i.e. sliding along the surface to which belong the Zeno state $x_\infty = \{x(t) \in \mathbb{R}^2 :\ x_1(t) = r_1\ \wedge\ x_2(t) = r_2\}$, carrying the solution past the Zeno limit point.

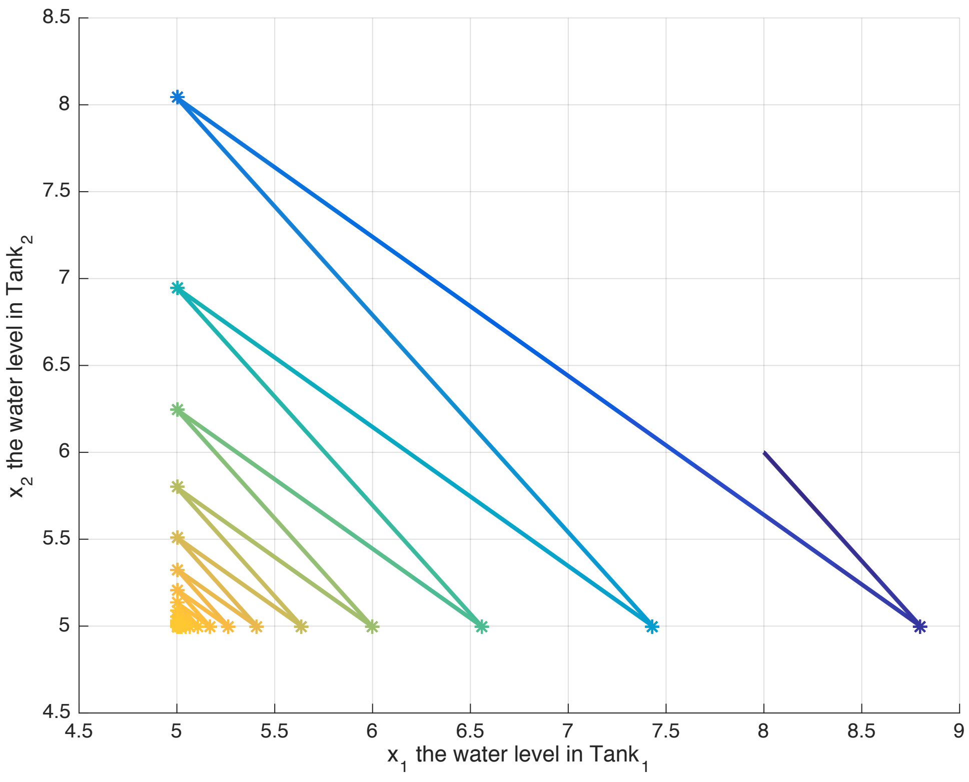


Figure 4.24: Zeno-free simulation of the two tanks model: x_2 versus x_1.

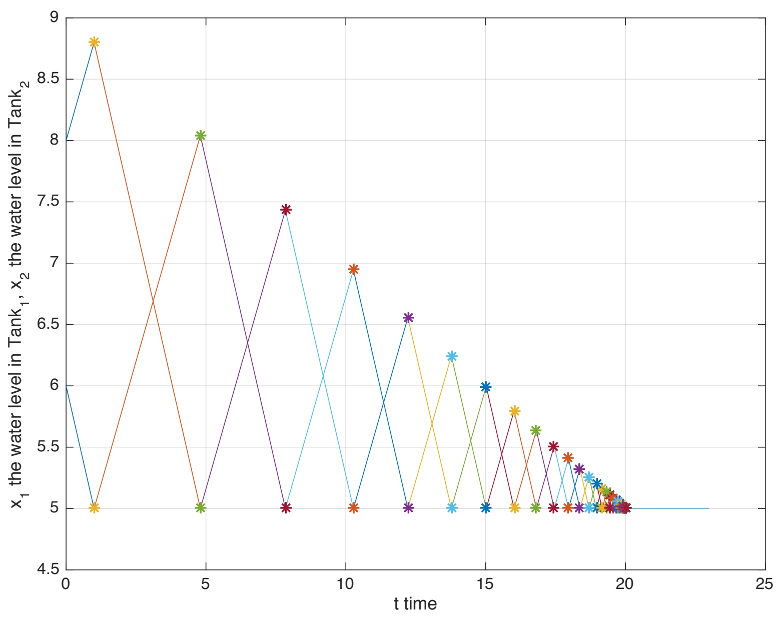


Figure 4.25: Zeno-free simulation of the two tanks model: The time evolution of the water levels x_1 and x_2.

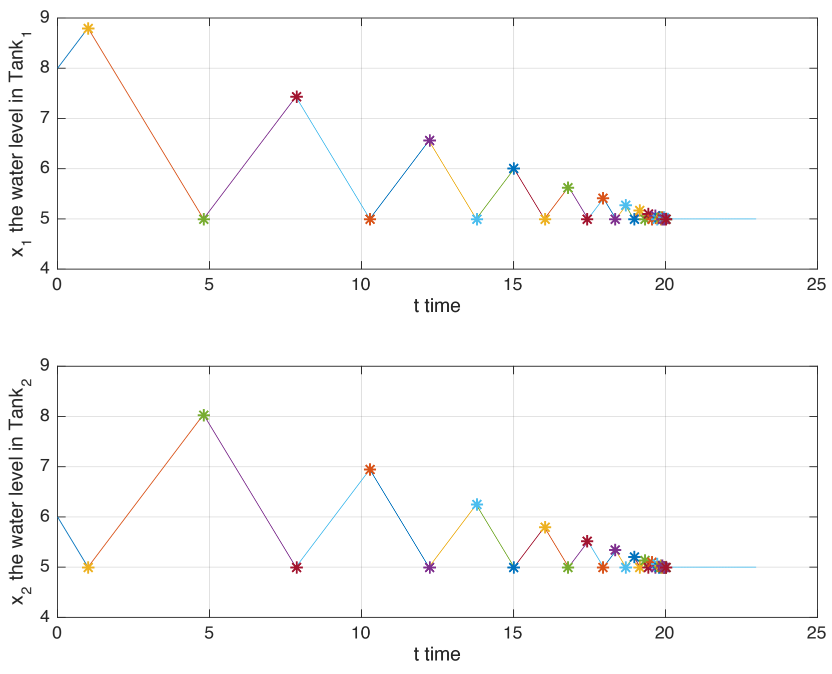


Figure 4.26: Zeno-free simulation of the two tanks model. Up: The time evolution of the water level x_1. Down: The time evolution of the water level x_2.

Chapter 5

Conclusions

This chapter summarizes the contents of this thesis and highlights several potential future research thrusts.

5.1 Summary

In this thesis, we have addressed the problem of both chattering-Zeno and geometric-Zeno behaviors of hybrid systems from a simulation perspective. We provided a static analysis of both types for Zeno in a non-standard hybrid time domain. We proposed a Zeno-free computational framework for Zeno detection and elimination "on the fly" based on an execution analysis and formally introduced conditions on when the simulated hybrid models exhibit chattering-Zeno or geometric-Zeno. We also provided methods for carrying solution beyond Zeno. A part of this thesis was also attributed to design, test, and validate Zeno-free implementations using the theory that we have proposed.

Non-Standard Static Analysis of Zeno Behavior. In Chapter 2 we gave a thorough introduction to hybrid dynamical systems, their model formalization, and executions. We discussed the limitation of the standard semantics of hybrid automata in the full treatment of the hybrid dynamics, especially for hybrid models exhibiting Zeno behavior. We proposed a non-standard semantics for Zeno executions of hybrid systems models, based on interpreting Zeno executions in a non-standard *densely ordered* hybrid time domain. The advantages of using non-standard semantics in the analysis of Zeno behavior is that the completeness in the space of the continuous dynamics and discrete dynamics is naturally introduced so that it allows for solutions of Zeno hybrid models to be well-defined beyond the Zeno limit points, while at the same time preserving the original semantics of the model:

1. The representation of dynamics based on non-standard analysis is complete.
2. In non-standard analysis, the continuous dynamics of the hybrid system is reduced to the recurrence equation that represents the infinite iteration of infinitesimal discrete changes with infinitesimal duration, and therefore, we can handle the hybrid dynamics based on fully discrete paradigm.

Chattering-Zeno Detection and Elimination. In Chapter 3, we have proposed a sliding mode computational framework with hierarchical application of convex combinations. The main benefit of the proposed framework is that chattering-Zeno detection and elimination is done "on the fly" without any need neither to add small hysteresis to the zero-crossing event functions, nor to solve stiff nonlinear equations for the computation of the equivalent sliding dynamics in case of chattering-Zeno on discontinuous surfaces intersections of high dimensions. The proposed hierarchical application of convex combinations provides a unique solution for chattering-free coefficients in general cases when the discontinuous switching manifold — on which chattering-Zeno occurs — takes the form of finitely many intersecting discontinuous surfaces.

Geometric-Zeno Detection and Elimination. In Chapter 4, we have proposed a new technique for detecting and eliminating "orbitally" geometric-Zeno behavior. We provided formal conditions for detecting cycles during execution of hybrid automata, and we derived sufficient conditions for the existence of geometric-Zeno behavior based on the existence — through the detected cycles — of a non-standard contraction map in a complete metric space, and the convergence of the solution to a geometric-Zeno limit point, through such map, according to a Cauchy sequence. Such map indicates when exactly a decision should be taken to transition the solution from pre-Zeno to post-Zeno, and thus eliminating Zeno behavior.

Prototype Implementations. A part of this dissertation was attributed to design, test, and validate Zeno-free simulator prototypical implementations. We have developed, tested, and validated two Zeno-free simulator implementations for chattering-Zeno detection and elimination. The first Zeno-free simulator implementation was developed in Acumen language, while the second one was developed conforming to Functional Mock-up Interface (FMI) standard for Model Exchange. Also we have developed, tested, and validated two Matlab/Simulink Zeno-free simulator implementations for geometric-Zeno detection and elimination. The Matlab implementation includes a stand-alone simulator written in Matlab code, while the Simulink implementation includes the basic ready-to-use Zeno-free Simulink library blocks that allow for geometric-Zeno detection and elimination of any Zeno hybrid model exhibiting geometric-Zeno.

5.2 Future Directions

The work described in this thesis can be continued in many different directions. In the following we suggest some ideas for possible future work:

- One of the possible future research directions would be to explore our theory in experimental case studies of complex hybrid systems having a large number of state variables (and thus a large space dimension), and whose models exhibit Zeno behavior.

- Another possible future research direction would be to extend our theory in such a way that it would be possible to measure as well as to control the sliding velocity (accelerating, slowing) without affecting the decision making for smooth exit from sliding.

- Our theory proposed in this thesis is purely an event-driven theory for hybrid systems simulation. A possible research direction thus would be to explore and to discuss how the methods proposed in this thesis would be used in a mixed hybrid simulation technique combining event-driven simulation and time-stepping simulation.

- Another future research thrust would be to extend our geometric-Zeno detection and avoidance technique presented in Chapter 4 in a way that includes an estimation of the geometric-Zeno limiting state using advanced state estimation techniques, as for example the use of state estimation with filtering (i.e. using Kalman Filetrs, Particle Filters, etc). It is worth noting that at the end of this dissertation we did not have enough time to discuss the computational load of our Matlab/Simulink prototype implementations — presented in Chapter 4 for geometric-Zeno detection and elimination — for larger case studies where the geometric-Zeno models contain a large number of state variables. This has been left for future research work.

Bibliography

[1] R. Alur, T. A. Henzinger, P. -H. Ho. Automatic symbolic verification of embedded systems. IEEE Transactions on Software Engineering, vol. 22, no. 3, pp. 181-201, 1996.

[2] J. Lygeros, C. Tomlin, and Sh. Sastry. Hybrid Systems: Modeling, Analysis and Control, Lecture Notes on Hybrid Systems, 2008.

[3] R. Alur, C. Coucoubetis, N. Halbwachs, T. A. Henzinger, P. -H. Ho, X. Nicollin, A. Olivero, J. Sifakis, S. Yovine. The algorithmic analysis of hybrid systems. Theoretical Computer Science, vol. 138, no. 1, pp. 3-34, 1995.

[4] P. J. Antsaklis. Special issue on hybrid systems: theory and applications a brief introduction to the theory and applications of hybrid systems. In proceedings of IEEE, vol. 88, no. 7, pp. 879-887, 2000.

[5] R. Alur, T. A. Henzinger. Modularity for timed and hybrid systems. In Proceedings of the 9th International Conference on Concurrency Theory, LNCS 1243, pp. 74-88, 1997.

[6] C. Cai, R. Goebel, R. Sanfelice, and A. Teel, Hybrid systems: limit sets and zero dynamics with a view toward output regulation, Springer-Verlag, 2008.

[7] Boris Golden, Marc Aiguier, and Daniel Krob. Modeling of complex systems II: A minimal- ist and unified semantics for heterogeneous inte- grated systems. Applied Mathematics and Com- putation, 218(16):8039–8055, 2012.

[8] J. Zhang, K. H. Johansson, J. Lygeros, and Sh. Sastry, Zeno hybrid systems, International Journal of Robust and Nonlinear Control, 11(05), pp.435-451, 2001.

[9] R. Alur, C. Courcoubetis, T. A. Henzinger, P. -H. Ho. Hybrid Automata: An algorithmic approach to the specification and verification of hybrid systems. In Hybrid Systems, LNCS 736, pp. 209-229, 1993.

[10] T. Blochwitz, M. Otter, J. Akesson, M. Arnold, C. ClauB, H. Elmqvist, M. Friedrich, A. Junghanns, J. Mauss, D. Neumerkel, H. Olsson, and A. Viel, Functional Mockup Interface 2.0: The Standard for Tool independent Exchange of Simulation Model, In Proceedings of 9th International Modelica Conference, Munich, Germany, 076(17), pp.173-184, 2012.

[11] M. di Bernardo, C. J. Budd, A. R. Champneys, Piotr Kowalczyk, A. B. Nordmark, G. O. Tost, and P. T. Piiroinen, Bifurcations in Nonsmooth Dynamical Systems, SIAM Review, 50(5), pp.629-701, 2008.

[12] K. H. Johansson, A.E. Barabanov, and K.J. Astrom, Limit cycles with chattering in relay feedback systems, IEEE Transactions on Automatic Control, 47(9), pp.1414-1423, 2002.

[13] D. Weiss, T. Kupper, and H.A. Hosham, Invariant manifolds for nonsmooth systems with sliding mode, Mathematics and Computers in Simulation, 110, March 2014.

[14] V. I. Utkin, Sliding Mode in Control and Optimization, Springer Berlin Heidelberg, ISBN: 978-3-642-06029-8, 2004.

[15] R. I. Leine, and H. Nijmeijer, Dynamics and Bifurcations of Non-Smooth Mechanical Systems, Springer Berlin Heidelberg, ISBN: 978-3-642-84379-2, 1992.

[16] V. I. Utkin. Sliding Modes in Control and Optimization. Springer, 1992.

[17] A.F. Filippov. Differential Equations with Discontinuous Righthand Sides, Springer Netherlands, ISBN: 978-94-015-7793-9, 1988.

[18] P. Schrammel. Logico-Numerical Verification Methods for Discrete and Hybrid Systems, PhD dissertation, 2012.

[19] J. P. Aubin, A. Cellina. Differential Inclusions, Springer, ISBN: 978-3-642-69514-8, Volume 264, 1984.

[20] M. Biák, T. Hanus, and D. Janovská, Some applications of Filippov's dynamical systems, Journal of Computational and Applied Mathematics, 254, pp.132–143, 2013.

[21] https://fr.mathworks.com/company/newsletters/articles/improving-simulation-performance-in-simulink.html.

[22] D. Harel, "Statecharts: A visual formalism for complex systems," Science of Computer Programming, vol. 8, p. 231:274, July 1987.

[23] P. Caspi, A. Curic, A. Maignan, C. Sofronis, and S. Tripakis, "Translat- ing discrete-time Simulink to Lustre," in Proc. of the Third Intl. Conf. on Embedded Software (EMSOFT). Philadelphia, PA, (R. Alur and I. Lee, eds.), (Berlin), pp. 84–99, Springer Verlag, October 2003.

[24] P. Fritzson. Principles of Object-Oriented Modeling and Simulation with Modelica 2.1. J. Wiley and Sons, 2004.

[25] P. Fritzson. Principles of Object Oriented Modeling and Simulation with Modelica 3.3: A Cyber-Physical Approach. J. Wiley and Sons, 2015.

[26] J.J. Moreau. Bounded variation in time. In J.J Moreau, P.D. Panagiotopoulos, and G. Strang, editors, Topics in Nonsmooth mechanics, pages 1–74, Basel, 1988. Bikh¨auser.

[27] J. Eker, J. W. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Neuendorffer, S. Sachs, and Y. Xiong. Taming heterogeneity: The Ptolemy approach. Proceedings of the IEEE, 91(1):127– 144, 2003.

[28] P. Fritzson. Introduction to Modeling and Simulation of Technical and Physical Systems with Modelica. Wiley-IEEE Press, 2011.

[29] Michael M. Tiller. Introduction to Physical Modeling with Modelica. Kluwer Academic Publishers", 2001.

[30] Thomas A. Henzinger. The theory of hybrid automata. In Proceedings of the 11th Annual IEEE Symposium on Logic in Computer Science, LICS'96, pages 278–292. IEEE Society Press, 1996.

[31] François E. Cellier, Ernesto Kofman, Gustavo Migoni, and Mario Bortolotto. Quantized state system simulation. Proceedings of Grand Chal- [14] lenges in Modeling and Simulation (GCMS'08), pages 504–510, 2008.

[32] Modelica Association. Modelica - a unified object-oriented language for physical systems modeling, language specification. December 2000.

[33] C. Hylands, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng. Hyvisual: A hybrid system visual modeler. Technial Report UCB/ERL M03/1, UC Berkeley, 2003.

[34] J. J. Moreau. Standard inelastic shocks and the dynamics of unilateral constraints. In Unilateral problems in Structural Analysis (G. del Piero and F. Maceri eds), Springer, New York, 1983, pp. 173-221.

[35] J. J. Moreau. Liaison unilatérales sans frottement et chocs inélastiques. C.R. Acad. Sc. Paris, 296:1473-1476, 1983.

[36] V. Acary and B. Brogliato. Higher order moreau's sweeping process. In P. Alart, O. Maisonneuve, and R.T. Rockafellar, editors, Non smooth Mechanics and Analysis: Theoretical and numerical advances, Colloquium in the honor of the 80th Birthday of J.J. Moreau (2003). Kluwer, 2005.

[37] A. Girard, A. A. Julius, G. J. Pappas. Approximate Simulation Relations for Hybrid Systems. Discrete Event Dynamic Systems, Springer, doi:10.1007/s10626-007-0029-9, Volume 18, Issue 2, pp 163–179, 2008.

[38] M. Schatzman. A class of nonlinear differential equations of second order in time. Non linear Analysis, 2(3):355–373, 1978.

[39] J. Eker, J.W. Janneck, E.A. Lee, J. Liu, J. Ludwig, S. Neuendorf- fer, S. Sachs, and Y. Xiong. Taming heterogeneity—The Ptolemy approach. Proceedings of the IEEE, 91(1):127–144, January 2003.

[40] J.J. Moreau. Numerical aspects of the sweeping process. Computer Methods in Applied mechanics and Engineering, 177:329–349, 1999. Special issue on computational modeling of contact and friction, J.A.C. Martins and A. Klarbring, editors.

[41] E.A. Lee and Y. Xiong. System-level types for component-based design. In T.A. Henzinger and C.M. Kirsch, editors, Embedded Software. Proceeding of the First International Workshop, EMSOFT 2001. Tahoe City, CA, volume 2211 of Lecture Notes in Computer Science, Berlin, October 2001. Springer Verlag.

[42] V. Acary, B. Brogliato, A. Daniilidis, and C. Lemaréchal. On the equivalence between complementarity systems, projected systems and unilateral differential inclusions. Systems and Control Letters, 2005.

[43] V. Acary, B. Brogliato. Numerical Methods for Nonsmooth Dynamical Systems: Applications in Mechanics and Electronics. Springer-Verlag Berlin Heidelberg, vol. 35, ISSN: 1613-7736, DOI:10.1007/978-3-540-75392-6, 2008.

[44] Goran Frehse, Colas Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang, and Oded Maler. SpaceEx: Scalable verification of hybrid systems. In Ganesh Gopalakr- ishnan and Shaz Qadeer, editors, Computer Aided Verification, volume 6806 of Lecture Notes in Computer Science, pages 379–395. Springer Berlin Heidelberg, 2011. doi: 10.1007/ 978-3-642-22110-1-30.

[45] V. Acary, B. Brogliato, and D. Goeleven. Higher order moreau's sweeping process : Mathematical formulation and numerical simulation. Mathematical Programming A, 2005.

[46] S. L. Campbell, J-P. Chancelier, and R. Nikoukhah. Modeling and Simulation in Scilab/Scicos with ScicosLab 4.4. Springer-Verlag New York, 2006.

[47] Edward A. Lee and Haiyang Zheng. Opera- tional semantics of hybrid systems. In Man- fred Morari and Lothar Thiele, editors, Hy- brid Systems: Computation and Control, volume 3414 of Lecture Notes in Computer Sci- ence, pages 25–53. Springer Berlin Heidelberg, 2005. ISBN 978-3-540-25108-8. doi: 10.1007/ 978-3-540-31954-2-2.

[48] R. Nikoukhah and S. Steer. SCICOS A dynamic system builder and simulator user's guide - version 1.0. Technical Report Technical Report 0207. INRIA, (Rocquencourt, France, June), 1997.

[49] H. Jerome Keisler. Foundations of Infinitesimal Calculus. On-line Edition, 2007. URL http://www.math.wisc.edu/keisler/ foundations.html.

[50] J. J. Moreau. Approximation en graphe d'une évolution discontinue. R.A.I.R.O. Analyse Numérique / Numerical Analysis, 12:75-84,1978.

[51] J. J. Moreau. Numerical aspects of the sweeping process. Computer Methods in Applied Mechanics and Engineering. 177 (1999) 329-349.

[52] C. Gomez. Engineering and Scientific Computing with Scilab. Birkhauser Verlag, 1999.

[53] R. Alur, T. Dang, J. Esposito, R. Fierro, Y. Hur, F. Ivancic, V. Ku- mar, I. Lee, P.

[54] Albert Benveniste, Timothy Bourke, Benoit Caillaud, and Marc Pouzet. Non-standard semantics of hybrid systems modelers. Journal of Computer and System Sciences, 78:877–910, May 2012. doi: 10.1016/j.jcss.2011.08.009.

[55] Peter Fritzson. Introduction to modeling and simulation of technical and physical systems with Modelica. Wiley-IEEE Press, 2011. ISBN 978- 1-118-01068-6.

[56] B. Brogliato. Impacts in Mechanical Systems – Analysis and Modelling. Springer–Verlag, New York, 2000. Lecture Notes in Physics, Volume 551.

[57] Timothy Bourke and Marc Pouzet. Zélus: A synchronous language with ODEs. In 16th International Conference on Hybrid Systems: Computation and Control (HSCC'13), pages 113–118, Philadelphia, USA, March 2013.

[58] B. Brogliato, A.A. ten Dam, L. Paoli, F. Génot, and M. Abadie. Numerical simulation of finite dimensional multibody nonsmooth mechanical systems. Applied Mechanics Reviews, 5:107–150, 2002.

[59] R. Alur, T. Dang, J. Esposito, Y. Hur, F. Ivancic, V. Kumar, I. Lee, P. Mishra, G. J. Pappas, and O. Sokolsky. Hierarchical modeling and analysis of embedded systems. Proceedings of the IEEE, November 2002.

[60] Bernard P. Zeigler and Jong Sik Lee. Theory of quantized systems: formal basis for DEVS/HLA distributed simulation environment. In SPIE Pro- ceedings, volume 3369, pages 49–58, 1998.

[61] J.J. Moreau. Unilateral contact and dry friction in finite freedom dynamics. In J.J. Moreau and P.D. Panagiotopoulos, editors, Nonsmooth mechanics and applications, number 302 in CISM, Courses and lectures, pages 1–82. Springer Verlag, 1988.

[62] R. Alur, A. Das, J. Esposito, R. Fierro, Y. Hur, G. Grudic, V. Kumar, I. Lee, J.P. Ostrowski, G. Pappas, J. Southall, J. Spletzer, and C.J. Taylor. A framework and architecture for multirobot coordination. In Proc. ISER00, 7th Intl. Symp. on Experimental Robotics, pages 289– 299, 2000.

[63] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specifica- tion of hybrid systems in charon. LNCS 1790. Springer-Verlag, 2000.

[64] Heinrich Rust. Operational Semantics for Timed Systems: A Non-standard Approach to Uniform Modeling of Timed and Hybrid Systems, vol- ume 3456 of Lecture Notes in Computer Sci- ence. Springer, 2005. ISBN 3-540-25576-1. doi: 10.1007/978-3-540-32008-1.

[65] A. Deshpande, A. Gollu, and P. Varaiya. Shift: a formalism and a programming language for dynamic networks of hybrid automata. In Hybrid Systems IV, LNCS 1273, pages 113–134. Springer-Verlag, 1997.

[66] Sébastien Furic. Enforcing model composability in Modelica. In Proceedings of the 7th Interna- tional Modelica Conference, Como, Italy, pages 868–879, 2009.

[67] A. Duracz, Ferenc A. Bartha, W. Taha. Accurate Rigorous Simulation Should be Possible for Good Designs. In Proceedings of the 2nd International Workshop on Symbolic and Numerical Methods for Reachability Analysis (SNR'16), April 11, 2016, Vienna, Austria.

[68] Abraham Robinson. Non Standard Analysis. North Holland, 1966.

[69] A. Deshpande, A. Gollu, and P. Varaiya. The shift programming language for dynamic networks of hybrid automata. IEEE Trans. on Automatic Control, 43(4):584–7, April 1998.

[70] R. G. Sanfelice, A. R. Teel. Dynamical Properties of Hybrid Systems Simulators. Automatica, Volume 46, Number 2, p.239–248, 2010.

[71] L. Semenzato, A. Deshpande, and A. Gollu. Shift reference manual. Technical report, California PATH, June 1996.

[72] Simon Bliudze and Daniel Krob. Modelling [11] of complex systems: Systems as dataflow machines. Fundamenta Informaticae, 91:1–24, 2009. doi: 10.3233/FI-2009-0001.

[73] M. Antoniotti and A. Gollu. Shift and smart ahs: a language for hybrid system engineering modeling and simulation. In Proceedings of the Conference on Domain-Specific Languages, Santa Barbara, CA, USA, Oct. 15-17 1997.

[74] A. Deshpande, D. Godbole, A. Gollu, L. Semenzato, R. Sengupta, D. Swaroop, and P. Varaiya. Automated highway system tool interface format. Technical report, California PATH Technical Report, January 1996.

[75] A. Deshpande, D. Godbole, A. Gollu, and P. Varaiya. Design and evaluation tools for automated highway systems. In Hybrid Systems III, Lecture Notes in Computer Science. Springer-Verlag, 1996.

[76] Stephen L Campbell, Jean-Philippe Chancelier, [13] and Ramine Nikoukhah. Modeling and Sim- ulation in Scilab/Scicos with ScicosLab 4.4. Springer, 2010. ISBN 978-1-4419-5527-2.

[77] Acumen 2015 Reference Manual. https://docs.google.com/document/d/1H1u64jmYAs7cbU7YWLw4HK2SEBjH1Y-Vi314a-8B-d8/edit.

[78] Michal Konecny, Walid Taha, Jan Duracz, Adam Duracz, and Aaron Ames. Enclosing the behavior of a hybrid system up to and beyond a Zeno point. In Cyber-Physical Systems, Networks, and Applications (CPSNA), 2013 IEEE 1st International Conference on, pages 120–125, 2013. doi: 10.1109/CPSNA.2013.6614258.

[79] A. Gollu and P. Varaiya. Smart ahs: a simulation framework for automated vehicles and highway systems. Mathematical and Computer Modeling, 27(9-11):103–28, May-June 1998.

[80] P. Varaiya. Smart cars on smart roads: problems of control. IEEE Trans. Automatic Control, 38(2), 1993.

[81] T. Simsek. Shift tutorial: A first course for shift programmers. Tech- nical report, UC Berkeley, 1999.

[82] D. Stewart. Convergence of a time-stepping scheme for rigid-body dynamics and resolution of Painlev´e's problem. Archives of Rational Mechanics and Analysis, 145:215–260, 1998.

[83] A. Duracz. Rigorous Simulation: Its Theory and Applications. PhD thesis, Halmstad University, 2016.

[84] F. Rocheteau and N. Halbwachs. Pollux, a Lustre-based hardware design environment. Conference on Algorithms and Parallel VLSI Architectures II. June 1991.

[85] Lionel Morel. Efficient Compilation of Array Iterators for Lustre. First Workshop on Synchronous Languages, Applications, and Programming, SLAP02, Grenoble, April 2002.

[86] P. Raymond, D. Weber, X. Nicollin, and N. Halbwachs. Automatic Testing of Reactive Systems. 19th IEEE Real-Time Systems Symposium. Madrid, Spain, December 1998.

[87] W. Taha, et al., Acumen: An Open-source Testbed for Cyber-Physical Systems Research, In Proceedings of EAI International Conference on CYber physiCaL systems, iOt and sensors Networks, October 2015.

[88] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet. A Hybrid Synchronous Language with Hierarchical Automata: Static Typing and Translation to Synchronous Code. In EMSOFT'11, Taiwan, Oct. 2011.

[89] A. Benveniste, T. Bourke, B. Caillaud, and M. Pouzet. Divide and recycle: types and compilation for a hybrid synchronous language. In LCTES'11, USA, Apr. 2011.

[90] A. Hindmarsh, P. Brown, K. Grant, S. Lee, R. Serban, D. Shumaker, and C. Woodward. SUNDIALS: Suite of nonlinear and differential/algebraic equation solvers. ACM Trans. Math. Soft., 31(3):363–396, Sept. 2005.

[91] G. Dahlquist and A. Bjorck. Numerical Methods in Scientific Computing: Volume 1. SIAM, 2008.

[92] N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata. Info. and Comp., 185(1):105–157, Aug. 2003.

[93] V. Kanovei, S. Shelah. Pang. A definable nonstandard model of the reals. Journal of Symbolic Logic, vol. 69, Issue , pp. 159-164, 2004.

[94] E. Levy. Non standard analysis as a functor, as local, as iterated. Technical report (https://arxiv.org/pdf/1601.00488.pdf). Department of Mathematics, Technion Israel Institute of Technology, 2016.

[95] J. Shen, J. S. Pang. Linear complementarity systems: Zeno states. SIAM Journal on Control and Optimization, vol. 44, no.3, pp. 1040–1066, 2005.

[96] A. Lamperski, A. D. Ames. Lyapunov theory for Zeno stability. IEEE Transactions on Automatic Control, vol. 58, no. 1, pp. 100–112, 2013.

[97] A.D. Ames, S. Sastry. Blowing up affine hybrid systems. In proceedings of the 43rd IEEE Conference on Decision and Control, vol. 1, pp. 473-478, 2004.

[98] L. Q. Thuan. Non-Zenoness of piecewise affine dynamical systems and affine complementarity systems with inputs. Control Theory and Technology, vol. 12, no. 1, pp. 35–47, 2014.

[99] M. K. Camlibel, J. M. Schumacher. On the Zeno behavior of linear complementarity systems. In Proceedings of the IEEE Conference on Decision and Control, 2001, vol. 1, pp. 346–351.

[100] A. Ames, A. Abate, S. Sastry. Sufficient Conditions for the Existence of Zeno Behavior. In Proceedings of the 44th IEEE Conference on Decision and Control and 2005 European Control Conference. CDC-ECC '05., Dec. 2005, pp. 696–701.

[101] Y. Zeng, C. Rose, W. Taha, A. Duracz, K. Atkinson, R. Philippsen, R. Cartwright, M. O'Malley. Modeling Electromechanical Aspects of Cyber-Physical Systems, Journal of Software Engineering for Robotics, 1(1), September 2015, 123-126, ISSN: 2035-3928.

[102] A. Lamperski, A. D. Ames. Lyapunov-Like Conditions for the Existence of Zeno Behavior in Hybrid and Lagrangian Hybrid Systems. In Proceedings of the 46th IEEE Conference on Decision and Control, 2007, pp. 115–120.

[103] K. H. Johansson, J. Lygeros, S. Sastry, M. Egerstedt. Simulation of Zeno hybrid automata. In proceedings of the 38th IEEE Conference on Decision and Control, vol. 4, pp. 3538-3543, 1999.

[104] A. Lamperski, A. D. Ames. On the existence of Zeno behavior in hybrid systems with non-isolated Zeno equilibria. In Proceedings of the 47th IEEE Conference on Decision and Control, pp. 2776–2781, 2008.

[105] R. Goebel, A. R. Teel. Lyapunov characterization of Zeno behavior in hybrid systems. In Proceedings of the 47th Conference on Decision and Control, 2008, pp. 2752-2757.

[106] A. D. Ames, H. Zheng, R. D. Gregg, S. Sastry. Is there life after Zeno? Taking executions past the breaking (Zeno) point. In Proceedings of American Control Conference, 2006, pp. 2652–2657.

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

[108] R.Goldblatt:Lecture on the Hyperreals: An Introduction to Nonstandard Analysis, New York, Springer (1988).

[109] G. Teschl. Ordinary Differential Equations and Dynamical Systems. vol.140, 356 pp, 2012, ISBN: 978-0-8218-8328-0.

[110] DOMNA: A Lite Matlab Simulator for Zeno-free Simulation of Hybrid Dynamical Systems with Zeno Detection and Avoidance in Run-Time, https://bil.inria.fr/fr/software/view/2691/tab.

[111] SEVAMA: A Simulink Toolbox and Simulator for Zeno-free Simulation of Hybrid Dynamical Systems with Zeno Detection and Avoidance in Run-Time, https://bil.inria.fr/fr/software/view/2679/tab.