APPROXIMATION BASED SAFETY AND STABILITY VERIFICATION OF HYBRID SYSTEMS

BY

PAVITHRA PRABHAKAR

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2011

Urbana, Illinois

Doctoral Committee:

Associate Professor Mahesh Viswanathan, Chair and Director of Research
Professor Gul Agha
Professor Geir Dullerud
Aarti Gupta, PhD
Assistant Professor Sayan Mitra

ABSTRACT

With the advent of computers to control various physical processes, there has emerged a new class of systems which contain tight interactions between the "discrete" digital world and the "continuous" physical world. These systems which exhibit mixed discrete-continuous behaviors, called *hybrid systems*, are present everywhere - in automotives, aeronautics, medical devices, robotics - and are often deployed in safe-critical applications. Hence, reliability of the performance of these systems is a very important issue, and this thesis concerns automatic verification of their models to improve the confidence in these systems.

Automatic verification of hybrid systems is an extremely challenging task, owing to the many undecidability results in the literature. Automated verification has been seen to be feasible for only simple classes of systems. These observations have emphasized the need for simplifying the complexity of the system before applying traditional verification techniques. This thesis explores the feasibility of such an approach for verifying complex systems.

In this thesis, we take the approach of verifying a complex system by approximating it to a "simpler" system. We consider two important classes of properties that are required of hybrid systems in practice, namely, *safety* and *stability*. Intuitively, safety properties are used to express the fact that something bad does not happen during the execution of the system; and stability is used to capture the notion that small perturbations in the initial conditions or inputs to the system, do not result in drastic changes in the behaviors of the system.

For safety verification, we present two techniques for approximation, an error based technique which allows one to compute as precise an approximation as desirable in terms of a quantified error between the original and the approximate system, and a property based technique which takes into account the property being analysed in constructing and refining the approximate

system. With regard to error based approximation, we provide a technique for approximating a general hybrid system to a polynomial hybrid system with bounded error. The above technique is in general semi-automatic; and we present a fully automatic efficient method for analysing a subclass of hybrid systems by constructing piecewise polynomial approximations. In terms of property based approximation, we propose a novel "counterexample guided abstraction refinement" (CEGAR) technique called *hybrid CEGAR*, which constructs a hybrid abstraction of a system unlike the previous methods which were based on constructing finite state abstractions. Our method has several advantages in terms of simplifying the various subroutines in an iteration of the CEGAR loop. We have implemented the hybrid CEGAR algorithm for a subclass of hybrid systems called *rectangular hybrid systems* in a tool called HARE, and the experimental results show that the method has the potential to scale.

Though automated verification of safety properties has been well studied, verification of stability properties is still in its preliminary stages with respect to automation. In this thesis, we present certain foundational results which could potentially be used to develop automated methods for analysis of stability properties. We present a framework for verifying "asymptotic" stability or convergence of (distributed) hybrid systems which operate in discrete steps. We then ask a fundamental question related to approximation based verification, namely, what kinds of simplifications preserve stability properties? It turns out that stability properties are not invariant under bisimulation which is a canonical transformation under which various discrete-time properties (including safety) are known to be invariant. We enrich bisimulation with uniform continuity conditions which suffice to preserve various stability properties. These reductions are widely prevelant in the traditional techniques for stability analysis thereby emphasizing the potential use of these techniques for stability verification by approximation.

To papu

ACKNOWLEDGMENTS

The contents and form of this thesis have been the product of interactions with and influences of various people. I would like to take this opportunity to thank all who have made the realization of this thesis possible.

First and foremost, I would like to thank Mahesh Viswanathan for his wonderful guidance during the past five years. He has been responsible for initiating many ideas presented in this thesis, and nurturing them into results. I am grateful to him for having given me the independence to explore and learn new topics, and do research at my own pace. I have learnt from him not only the art of formulating and solving problems, but also presenting research ideas and results. He has influenced in many profound ways my growth as a researcher. His delightful enthusiasm towards research and teaching has always amazed and inspired me, and I will always strive to be an advisor, researcher and teacher like him.

I am thankful to Geir Dullerud for his long-term collaboration, especially for his inputs on the controls side of the research in the thesis. I am also grateful to him for his advice and encouragement at various times.

I would like to thank Sayan Mitra for his collaborations on various chapters of the thesis, and his input on the practical aspects of the research. More importantly, he has been a fine mentor in the last few years of my PhD.

I would like to thank my committee members Gul Agha and Aarti Gupta for their comments and suggestions on the thesis. I am thankful to Gul Agha, Lou Van den Dries, Michael Heath, Daniel Liberzon, Jose Meseguer, Sariel Har Peled, Manoj Prabhakaran, Slowamir Solecki, Mahesh Viswanathan, Douglas West and many other for having taught me the courses which built the necessary foundations for carrying out the research in this thesis. I would like to thank Madhusudan Parthasarathy for his advice on various matters.

I would like to thank Vladimeros Vladimerou for his collaborations in the earlier years of my PhD, which resulted in the work presented in Chapter

3. I am thankful to Parasara Sridhar Duggirala for his collaboration on the work on hybrid CEGAR and his patient implementation of the tool HARE.

I would like to thank Fangzhe Chang and Ramesh Viswanathan for allowing me to spend a few summers at Bell-labs; it not only provided a much needed diversion from the main line of research, but also an opportunity to learn many interesting things I would have otherwise missed.

I would like to thank Deepak D'Souza for having introduced me to the area of Verification and motivated me to do research in this area. He has also been responsible for teaching me the fundamentals of Verification, Logic and Timed Systems.

Urbana-Champaign would not have been the same without the friendly and encouraging environment provided by friends and colleagues - Sruthi Bandhakavi, Rohit Chadha, Sridhar Duggirala, Mike Katelman, Vijay Korthikanti, Rajesh Karmani, Edgar Pek, Camilo Rocha, Ralf Sasse, Payal Shah, Aparna Sundar and many others.

This thesis would not have been possible without the love and support of my family. I am thankful to my mother for her perpetual support and encouragement towards my education and career; and my father for his patient efforts towards my personal and emotional growth, and more importantly for having encouraged the independent thinker in me. This thesis is dedicated to my sister, Pavana, for being a kind and patient friend in this journey of life.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

CHAPTER 1

INTRODUCTION

The widespread use of computer-controlled devices has sparked the emergence of a new class of systems which have come to be named *Cyber Physical Systems* (CPSs). A unique feature of these systems is the tight interaction between control, communication and physical components. Cyber Physical Systems appear widely in various domains such as aeronautics, automotive, robotics, medical devices, manufacturing and many others. These system vary widely in size - they could be as small as a pacemaker or as large as a power grid. These systems exhibit a mixture of discrete and continuous behaviors, with the discrete part coming from the control and communication, and the continuous part from the physical processes. Such systems with mixed discrete-continuous behaviors have more traditionally been called *hybrid systems*.

With automation becoming an integral part of various applications due to reasons concerning reliability, safety and efficiency, we can expect a huge growth in the number and size of hybrid systems in the coming years. Since these systems are going to be deployed in safety critical situations, there is a pressing need for developing techniques and tools to aid reliable development of hybrid systems. *This dissertation addresses the problem of verifying the design of hybrid systems for conformance to the desired behavior, and develops techniques and tools to address this problem.*

We focus on the development of formal verification techniques for ensuring that a given design of a hybrid system satisfies a desired property of the behavior of the system. *Formal Verification* entails defining a formal or mathematical model of the system under study, a formal specification of the property with respect to which the model needs to be verified, and a method for checking that the model satisfies the specification. Verification techniques can be broadly classified as *algorithmic* - those based on exhaustive state space exploration called "model-checking" - and *deductive* - those

which reduce the problem to that of deciding the validity of a logical formula and use theorem proving to check the validity of these formulas. Both these methods have their own advantages and disadvantages. Traditionally, model-checking based techniques have been automatic, where as theorem proving based methods have been semi-automatic, often requiring user help. However, theorem proving based methods are generally able to handle a more expressive class of systems than what can be handled by model-checking based methods. We are interested in automatic verification of hybrid systems, and hence we focus on algorithmic techniques.

Analysis of hybrid systems brings new challenges due to the presence of both discrete and continuous components. Discrete systems have been the standard models for representing software and hardware systems; and have been extensively studied by computer scientists. In fact, the area of verification has its origin in computer science, with many advances in the theory and practice of verification of software and hardware in the last few decades. Continuous systems, on the other hand, have been investigated in the field of controls, with the focus on synthesis of efficient controllers optimizing various performance objectives. Since hybrid systems combine discrete dynamics with continuous dynamics, the study of these systems has its foundations in two different areas which have evolved fairly independently. One of the challenges in the analysis of hybrid systems is to combine techniques developed in computer science and control theory in a consistent manner. Also, the correctness criteria for a hybrid system is more involved; it needs to satisfy properties from both the discrete and the continuous worlds. We will study the verification problem with respect to two important classes of properties, namely, *safety* and *stability*.

Safety is a property of a system which requires that a bad event does not happen along any execution of the system. For example, in an aircraft coordination protocol, collision between two aircraft is a bad event and we would want any two aircraft to always maintain a minimum distance, which is a safety property expected of the protocol. Safety has traditionally been studied in the discrete setting; and is an important class of properties since various correctness criteria of systems can often be formulated as a safety property.

The second class of properties we study are stability properties. Intuitively, stability requires that when a system is started somewhere close to its desired

operating behavior, it will stay close to that desired operating behavior at all times. For example, we would expect the controlled behavior of a robot to depend gracefully on small variations to its starting orientation; more precisely, given any starting orientation there should be some neighborhood of this orientation for which all trajectories that start in this neighborhood remain close, and furthermore, it should be possible to ensure that the trajectories are as close as desired by making the neighborhood sufficiently small. Stability is one of the fundamental properties which is expected in the design of any control system, so much so that a system which is not stable is deemed useless.

In the next two sections, we provide an overview of the state of the art in the verification of safety and stability properties.

1.1 An Overview of Safety Verification

One of the important factors in the reliability of a system is to ensure that its behaviors fall into a desired set of behaviors. Often this set of behaviors corresponds to the absence of certain bad events during the execution of the system. Such requirements are referred to as safety requirements, since they ensure that the system operates with in a safety envelope described by the good set of behaviors. Safety verification has been studied extensively in the discrete setting in the context of software and hardware verification, and has resulted in the development of efficient techniques and tools for automated analysis. Many of these techniques have been borrowed into the hybrid world for the safety analysis of hybrid systems; however the addition of continuous dynamics brings new challenges into the safety analysis. Next, we will discuss some of the work on safety analysis of hybrid systems and explain how the work in this thesis relates to it.

1.1.1 Modelling Formalisms

A first step towards formal analysis is to obtain a formal or mathematical model of the system under study. There has been extensive research on formalisms or languages for describing hybrid systems. The standard formalism for modelling discrete systems, namely, finite state systems often do

not suffice to capture enough information about the continuous dynamics of a hybrid system. Hence the formalism of hybrid automata [2] has been proposed which combines finite state systems with differential equations that have been the standard framework for modelling physical processes in control theory and other engineering disciplines. These essentially generalize the timed automata model [4] which extend finite state automata with certain continuous variables whose values evolve with time at a constant rate of 1. Hybrid automata have been used to model and analyse certain distributed processes with drifting clocks, real-time schedulers and protocols for the control of manufacturing plants, vehicles and robots (see for example [6, 77]).

Hybrid Input/Output Automata [71] extend the formalism of hybrid automata with clearly defined input and output variables for the purpose of communication. Some of the other formalisms and languages for modelling hybrid systems include Modelica [43], CHARON [5] and Masaccio [55] which provide formalisms for hierarchical modelling of hybrid systems, Simulink/ Stateflow and HyVisual [19] which are graphical languages for specifying hybrid systems, SHIFT [35] which allows the modelling of dynamic networks of hybrid systems, and hybrid process algebras [31] which provides an algebraic approach to modelling and analysis of hybrid systems.

Depending on the system being modelled and the property being analysed, a particular formalism might be more suitable or convenient. In this thesis, we refrain ourselves from the modelling aspects. We choose hybrid automata as our modelling formalism, since, it is expressive enough to model most practical hybrid systems, and also has simple representation for the purpose of formal analysis.

1.1.2 Analysis

There are two broad approaches to automated analysis of safety properties, namely, *simulation* and *verification*. We will discuss the work related to these analysis techniques in the context of hybrid systems.

Simulation Simulation refers to the process of executing the system model starting from a single initial state and examining the observed behavior to determine if it conforms to the desired behavior. The de facto standard in industries, such as aeronautics and automotive, for modelling and simulation

of hybrid systems has been the MATLAB based tool set Simulink/ Stateflow. However, one of the main concerns in using this tool set for formal analysis is a lack of clearly defined formal semantics of the underlying simulation engines. Recently, there have been several attempts at formalizing the semantics of various fragments of the language [101]. Other formalisms which support simulation of the system models include Modelica [43], CHARON [5] and HyVisual [19].

In practice, simulation based analysis does not cover executions from every possible initial state, much less they allow one to observe the behaviour of the system only at certain time points. There has been some research on symbolic simulation where in the system is simulated starting from symbolic states which represent a possibly infinite set of states [61]. Developments in symbolic simulation techniques might go a long way in providing more robust simulation based methods.

Since simulation is in general is not exhaustive, it does not provide any formal guarantee about the correctness of the system. In contrast, verification based analysis, tries to formally prove that the system satisfies a certain property, and hence is more robust. The focus of this thesis is in developing verification techniques for proving safety of hybrid systems.

Algorithmic Verification The main focus towards verification of hybrid systems has been in developing algorithmic or model-checking based techniques (except for some recent efforts towards developing theorem proving techniques [87]). Some of the earliest work towards algorithmic analysis of hybrid systems includes the development of the theory of timed and hybrid automata [4, 1, 54, 52]. However, it was shown that safety verification is undecidable for a relatively simple subclass of hybrid systems [54]. This lead to a line of research consisting of identifying subclasses of hybrid systems with decidable properties, which we will discuss next.

The class of timed automata was introduced in [4] and the reachability problem for this class was shown to be PSPACE-complete. These are a class of models which extend finite state systems with a finite number of "clocks" which are variables whose values evolve with time at a constant rate of 1. Even this simple extension to the discrete models leads to an exponential blow-up in the complexity of verification [20]. Various tools have been developed for model-checking the class of timed automata, most

prominent of them being UPPAAL [13] and Kronos [104].

In [54], undecidability of the class of *rectangular hybrid automata* (RHA), was shown. These systems consist of variables whose evolution is such that their derivatives belong to a specified constant interval. The same work also identified a subclass of RHA called the initialized rectangular hybrid automata, which consists of RHA satisfying the constraint that during a mode change all variables with different dynamics in the modes before and after the change are reset, for which safety is decidable. HyTech [56] is a model-checker which implements a semi-decision procedure for the analysis of RHA.

Generalizing rectangular hybrid automata, one obtains the class of hybrid automata, where in, the gradient of the flow function belongs to a polyhedron, rather than a rectangular set. Decidability has been established for various subclasses of these hybrid automata [9, 91, 10, 11]. However, these results hold in very low dimensions, such as, two or three.

Recently, decidability of safety verification was shown for systems with certain restricted kinds of linear dynamics [66] ($\dot{x} = Ax$, where A is a constant matrix satisfying certain constraints), and for a class of systems which are definable in an o-minimal structure with a decidable theory [65, 18]. These systems allow rich continuous dynamics including polynomial and certain classes of exponential flows, but they decouple the dynamics in different modes by resetting the variables along a discrete transition. Various relaxations of the discrete dynamics have been shown to be decidable [103, 44].

It has been observed repeatedly that decidability is achieved for classes of systems with simple continuous or discrete dynamics. However, the fact remains that most real-life systems do not fall into these classes of decidable systems. This has lead to a broad range of approximation techniques, which essentially simplify a complex system into a simpler system on which analysis can be directly performed. The main focus of this thesis is also on developing approximation based techniques for analysis of systems. Below, we briefly review various existing techniques for approximation and how the results in this thesis compare with them.

Approximations of Hybrid Systems The class of approximation techniques can be broadly classified into two groups. The first class of techniques aim at *simplifying the continuous dynamics*, and the second class of tech-

niques perform *state-space reduction.*

Simplification of continuous dynamics. These techniques focus on constructing a simpler system by mainly simplifying the continuous dynamics. Further, there are two sets of techniques for simplifying the continuous dynamics. The first set focuses on simplifying the continuous dynamics by approximating the "flows" - solutions of the differential equations representing the complex continuous dynamics - by simpler flows which are amenable to efficient analysis. These techniques are typically applied to the classes of systems for which there exist closed form representations of their flow functions, such as, the class of linear dynamical systems (LDS) (where the continuous dynamics is specified by $\dot{x} = Ax$, A is a constant $n \times n$ matrix). Various techniques have been developed for the approximation of the solutions of linear dynamical systems. These techniques construct piecewise linear functions or ellipsoidal functions as approximations of the exponential flow of the LDS [32, 64, 73, 24, 45, 46, 51]. We present a novel algorithm which constructs piecewise polynomial functions. Our experimental results show improvements both in the time for constructing the approximation and the representation of the approximation, many a times by an order of 2 when compared with the previous algorithms.

The second set of techniques construct a simpler system directly from the differential equation or inclusion. More precisely, the state space of a complex dynamical system is partitioned into a finite number of parts and the continuous dynamics in each of the parts is approximated by a simpler differential equation or inclusion. This is popularly known as the *hybridization* approach; and has been widely studied for analyzing systems with non-linear dynamics for which generally closed-form solutions are not known [92, 7, 34].

In both sets of techniques, one can often quantify the error in the approximation, and approximation method is parametrized by this error. Hence, these methods can be called "error-based approximation methods".

State-space reduction. This refers to techniques in which the state-space of the approximate system is typically "smaller" than that of the original system. This has been the popular approach in verification of discrete systems with large number of states for which applying traditional verification techniques directly is not practically feasible. Here, an abstract transition system (hybrid or otherwise) that *simulates* the original hybrid system is first constructed, and the reachability computation is then carried out on

the abstract system. Typically, the abstract system is a finite state system, and is constructed, for example, by using predicate abstraction [100, 3, 26]. In some cases, the abstract system is also a hybrid system [60, 36].

In this method, the quality of the approximate solution cannot be quantitatively measured, and so, often this is compensated by repeatedly refining the abstract system as the analysis progresses. One of the techniques for refining an abstract system is based on analysing a counter-example of the current abstraction, and is popularly called the *Counter-Example Guided Abstraction Refinement* (CEGAR). CEGAR was first introduced in the context of software verification [28] for discrete systems; there have been several proposals for extending CEGAR to the hybrid setting [23, 3, 26, 27, 39, 97]. However, these techniques typically construct a finite state abstraction of the hybrid system. We propose a method called "Hybrid CEGAR" in which the abstract system is also a hybrid system. We argue that choosing an abstraction space of hybrid systems simplifies various tasks in a particular iteration of the CEGAR algorithm, thereby, ensuring the progress of the algorithm. We have implemented our algorithm for the class of RHA in a tool called HARE, and our experimental results suggest that the hybrid CEGAR approach has the potential to scale. These abstraction refinement techniques can be classified as "property based techniques", since they take into account the property being analysed both in constructing the approximate system and refining it. Some of the tools which perform approximation based verification include d/dt [8], PHAVer [41], CHECKMATE [25], MATISSE [48] and SpaceEx [42].

1.2 An Overview of Stability Verification

Stability is a property of the system which ensures that small perturbations to the initial state or input to a system does not lead to drastic changes to its future behaviors. This is an important property that a system is expected to possess to be practically useful, since in practice it is unreasonable to expect that a system start with a particular configuration exactly. In the design of control systems, stability is often a design goal. Stability has been a subject of study in the field of control theory for more than a century. There are many deep result on the analysis of stability of linear and non-linear systems. We will give a brief overview of some of the classical techniques for stability

analysis of control systems.

One of the most extensively studied classes of continuous dynamical systems is that of linear dynamical systems. There exist various results characterizing the stability of linear systems in terms of the eigen values of the representing matrix. For example, in the case of linear time-invariant systems with no inputs, the system is "asymptotically" stable iff all the eigen-values of the corresponding matrix are negative.

For the case of non-linear systems, there are two main methods for stability analysis. First method is called the *linearization* approach, in which a linear system which approximates the non-linear system around the equilibrium is constructed, and analysed to infer the stability properties of the non-linear system. The linear system that is construct is essentially the linear time-invariant system whose matrix is the Jacobian of the non-linear function evaluated at the equilibrium point. The second method, known as the Lyapunov's method (second), establishes the stability of a system by exhibiting a function of a certain form, called the Lyapunov function, on the state space such that the value of the function decreases along any execution of the system. These results can be found in any text book on state space methods for analysis of control systems, see for example [63].

Since stability has been primarily studied in the domain of controls, many of the results apply to the purely continuous setting and it is not always straightforward to extend these techniques to the mixed discrete-continuous setting. For example, let us consider the eigen value based analysis of linear systems. Stability analysis of a linear hybrid system cannot depend merely on the analysis of the eigenvalues of the matrices representing the continuous dynamics of the system, since it is fairly straightforward to exhibit two matrices (which are stable) and two systems which result from switching between the matrices in different manners, such that one of the systems is stable where as the other is unstable. However, there have been some efforts in the past decade to extend the stability analysis techniques to the hybrid setting [16, 69].

Though stability has been studied extensively, there is one aspect which has not received much attention, namely, automation. Most of the techniques for stability analysis are either manual or semi-automatic, and in general are not easily amenable to automation. For example, let us consider Lyapunov's second method for proving stability. Automating this proof technique re-

quires automation of two steps: (1) finding the right Lyapunov function, and (2) showing that the value of the Lyapunov function decreases along any trajectory of the system. In general, even manually figuring out the right Lyapunov function is an arduous task, and it is not clear how such a task could be automated. However, the technique can be automated for the class of linear systems, owing to the results which characterize the space of Lyapunov functions one needs to search for, and more importantly allow one to automatically derive such a function by solving certain Linear Matrix Inequalities. However, this can be done only for the simple class of linear systems. This brings us back to the idea of approximation; it seems that the only reasonable approach to achieve automated verification of complex systems is to approximate them to simpler system on which analysis can be automatically carried out.

In this thesis, we focus on approximation based automated verification of stability properties. Firstly, we provide a general framework for proving "asymptotic" stability of discrete-time linear systems. Secondly, we address a fundamental question aimed towards approximation based approach, namely, what kinds of simplifications preserve stability properties. These questions have not been studied thoroughly in the context of stability verification; and we believe that the results in the thesis will serve as the basis for developing automated techniques for stability analysis of hybrid systems.

1.3 Contributions

As pointed out in sections 1.1 and 1.2, from the point of view of automation, verification of hybrid systems is in general a difficult problem. It was observed that the problem of verification becomes undecidable for a relatively simple subclass of hybrid systems [54]. In fact, for system with complex continuous dynamics such as those specified by linear and non-linear differential equations or inclusions, even computing "one-step successors" which is the set of all states reached by one step of execution of the system (purely continuous executions without any discrete jumps) is a difficult problem. This is a basic operation on various state-space exploration based methods for analysis of systems. Hence, the only hope in being able to automatically verify these complex systems is to somehow "approximate" them into "simpler" systems

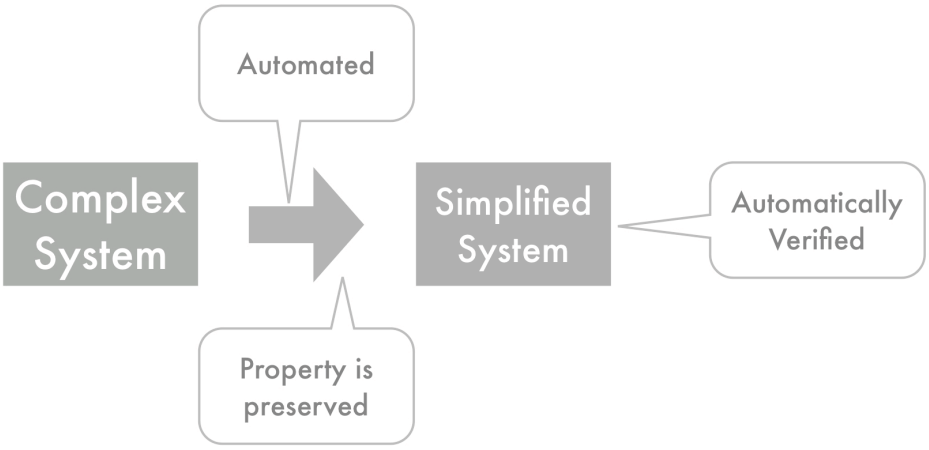

Figure 1.1: Approximation based Verification

which can be effectively analysed. This is the route we take in this thesis for analysing complex hybrid systems. Below we briefly summarize the process of approximation based verification.

1.3.1 Approximation based Verification of Complex Hybrid Systems

The first step in approximation based verification is to identify a target class of systems to approximate to, which are simple enough to be efficiently *automatically analysable*. Then we approximate the complex system into a simpler system in the target class, and analyse the simplified system. We should be able to infer the correctness of the original system from the analysis of this simpler approximate system. Hence, the approximate system constructed should be such that it *preserves the property* of interest. Finally, since we are interested in the automatic analysis of complex systems, we want the whole process to be automated, in particular, we want the *process of approximation to be automated*. The above objectives of approximation based verification are summarized in Figure 1.1. Next, we will present our results on approximation based verification of safety and stability properties.

1.3.2 Approximation Techniques for Safety Verification

We focus on the approximation part of the analysis and present two approximation techniques for safety analysis.

Error based approximation We explore the possibility of considering polynomial hybrid systems as the target class. Polynomial hybrid systems are a reasonable class of systems to approximate to since various bounded properties are decidable for the class.

First, we present a method that approximates a hybrid system with general continuous dynamics by systems with polynomial flows. The method takes the error in approximate system constructed as a parameter, and hence allows us to not only quantify the error in the approximation, but also construct as tight an approximation as desired. The approximation preserves the safety in one direction - if the approximate system is safe, then one can conclude that the original system is safe. We also show that for certain classes of systems with "tolerant" behaviors, safety is preserved in both directions.

This is a generic technique which applies to a large class of systems. However, the process of approximation is not automatic in general; and moreover, the degree of the polynomial grows with the precision of approximation, which in turn effects the cost of verification. We present a method for automating the process of approximation for the class of linear dynamical system, and in order to keep the degree of the polynomial from blowing up, we consider piecewise polynomial systems as the target class. Our experiments show that our piecewise polynomial approximation construction is better both in terms of the size of the final approximation and the time taken for constructing the approximation, when compared to various existing methods.

Property based abstraction Error based approximations do not take into account the property being analysed. Next, we focus on constructing abstractions based on the property. However, these techniques do not construct approximations with bounded error; and hence more and more precise approximations are obtained by refining the abstraction. One such technique for refinement which has been successfully used in software verification is based on the analysis of a "counterexample" - a witness to the violation

of the property - which is popularly known as "Counter Example Guided Abstraction Refinement" (CEGAR). There have been several proposals for applying CEGAR for hybrid systems. Most of these systems consider the class of finite state systems as the abstraction space. Our hypothesis is that the correct way to approximate a hybrid systems is to approximate it with a system which is also a hybrid system.

We propose a new method called "Hybrid CEGAR" in which the abstraction space is also a class of hybrid systems. This technique has various advantages in terms of simplifying the various tasks of the CEGAR loop, thereby guaranteeing progress. We have built a prototype tool called HARE implementing the above technique for the class of rectangular hybrid automata (RHA). Our experiments show the feasibility of such a technique for the verification of hybrid systems.

1.3.3 Approximation based Stability Verification

We present a general framework for reasoning about asymptotic stability of discrete-time hybrid systems, and explore pre-orders for reasoning about stability.

A technique for proving asymptotic stability Asymptotic stability is the property of a system which is addition to (Lyapunov) stability requires that the execution of the system eventually converges to the equilibrium. A discrete-time (distributed) hybrid system, where discrete transitions happen at discrete times, can be considered as a finite set of operators acting on the state space, and an execution of such a system can be considered as a sequence of operators applied on the state space. The sequences of operators arising from a discrete-time hybrid system form an ω-regular language. Tsitsiklis [102] has developed a method for proving convergence of systems with a finite set of operators, where in each execution every operator is assumed to be applied infinitely often. We extend this framework to the class of ω-regular languages. We present necessary and sufficient conditions for proving "convergence" or asymptotic stability of system where the operators interact in a regular fashion.

Pre-orders for reasoning about stability We explore a fundamental question in approximation based verification, namely, what kinds of simplifications preserve stability properties? It turns out that the classical notions of simulation and bisimulation, which have played an important role in the analysis of discrete systems with respect to various branching time properties, do not suffice for the purpose of analysing stability properties. We strengthen these notions by adding certain continuity constraints, namely, that of "uniform continuity", and show that this new notion suffices to preserve stability properties. We argue that this is a reasonable notion to consider by showing that various proofs of stability can be cast as proofs by uniformly continuous reductions to simpler systems with easy stability proofs.

1.4 Organization of the Dissertation

We start with some preliminaries in Chapter 2 including a formal definition of the hybrid automaton model. Chapter 3 contains the results on polynomial approximations of hybrid systems with general dynamics which first appeared in [90]. In Chapter 4, we present the piecewise polynomial approximation construction for linear dynamical systems which appeared in [89]. Chapter 5 consists of the hybrid CEGAR algorithm for rectangular hybrid automaton and experiments with our tool HARE. The results on stability verification are presented in Chapters 6 and 7. Chapter 6 presents the framework for proving convergence of discrete-time hybrid systems which appeared in [88] and Chapter 7 discusses pre-order for reasoning about stability. We present our conclusions and future research directions in Chapter 8.

CHAPTER 2

PRELIMINARIES

2.1 Basic Definitions

2.1.1 Notations

We denote the set of natural numbers, real numbers and non-negative real numbers by $\mathbb{N}$, $\mathbb{R}$ and $\mathbb{R}_{\geq 0}$, respectively. Let $\mathbb{R}_\infty$ denote the set $\mathbb{R}_{\geq 0} \cup \{\infty\}$, where ∞ denotes the largest element of $\mathbb{R}_\infty$, that is, $x < \infty$ for all $x \in \mathbb{R}_{\geq 0}$. Also, for all $x \in \mathbb{R}_\infty$, $x + \infty = \infty$.

Given $x \in \mathbb{R}^n$, let $(x)_i$ denote the projection of x onto the i-th component, that is, if $x = (x_1, \cdots, x_n)$, then $(x)_i = x_i$. Given $X \subseteq \mathbb{R}^n$, $(X)_i = \{(x)_i \mid x \in X\}$. Given a function $F : A \to \mathbb{R}^n$, let $F_i : A \to \mathbb{R}$ denote the function given by $F_i(a) = (F(a))_i$. Given a function $F : \mathbb{R}_{\geq 0} \to B$ and $[a, b] \subseteq \mathbb{R}_{\geq 0}$, let $F[a, b] : [0, b - a] \to B$ denote the function given by $F[a, b](c) = F(a + c)$. Given a function F: $A \to B$, and $A' \subseteq A$, $F(A')$ will denote the set $\{F(a) \mid a \in A'\}$. Given a function F, let $Dom(F)$ denote the domain of F.

A polynomial over a variable x of degree k, denoted $p(x)$, is a term of the form $a_0 + a_1x^1 + \cdots + a_kx^k$, where $a_i \in \mathbb{N}$ for all $1 \leq i \leq k$ and $a_k \neq 0$. Given a $v \in \mathbb{R}$, let $p(v)$ denote the value obtained by substituting v for x in the expression $p(x)$ and evaluating the resulting expression. A function $P : [a, b] \to \mathbb{R}^n$, for some $a, b \in \mathbb{R}$, is a *polynomial function* if there exist polynomials, $p_1, \cdots, p_n$ over x, such that for all $v \in [a, b]$, $P_i(v) = p_i(v)$. The degree of the polynomial function P is the maximum degree of the polynomials representing it, that is, degree of P is maximum of the degrees of $p_1, \cdots, p_n$ (the degree is unique). Note that polynomial functions are continuous functions. A piecewise polynomial function is a function whose domain can be divided into finite number of intervals such that the function restricted to each of these intervals is a polynomial function. A *piecewise*

polynomial function (*PPF*) is a continuous function $P : [a, b] \to \mathbb{R}^n$, where $a, b \in \mathbb{R}$, such that there exists a sequence $t_1, \cdots, t_k$ such that $a < t_1 < \cdots < t_k < b$ and $P[a, t_1], P[t_1, t_2], \cdots, P[t_k, b]$ are all polynomial functions.

Given a binary relation $R \subseteq A \times B$, R^{-1} denotes the set $\{(x, y) \mid (y, x) \in R\}$. Given an equivalence relation R on A the equivalence class of $a \in A$ is denoted by $[a]_R$. We will use R to also denote the function from A to the set of equivalence classes of R given by the mapping $a \mapsto [a]_R$. An *interval* is a set of the form $[a, b]$, $[a, \infty)$, $(-\infty, b]$ or $(-\infty, +\infty)$, where $a, b \in \mathbb{Z}$. The first kind is called a *bounded interval*. Given $n \in \mathbb{N}$, $[n]$ denotes the set $\{1, \cdots, n\}$.

2.1.2 Metric Spaces

An (extended) metric space $\mathcal{M}$ is a pair (M, d) where M is a set and $d : M \times M \to \mathbb{R}_\infty$ is a distance function such that for all m_1, m_2 and m_3,

1. (Identity of indiscernibles) $d(m_1, m_2) = 0$ if and only if $m_1 = m_2$.
2. (Symmetry) $d(m_1, m_2) = d(m_2, m_1)$.
3. (Triangle inequality) $d(m_1, m_3) \leq d(m_1, m_2) + d(m_2, m_3)$.

The following concepts are defined with respect to a fixed metric space $\mathcal{M} = (M, d)$. We define an open ball of radius ϵ around a point x to be the set of all points which are within a distance ϵ from x. Formally, an *open ball* is a set of the form $B_\epsilon(x) = \{y \in M \mid d(x, y) < \epsilon\}$. An *open set* is a subset of M which is a union of open balls. Given a set $X \subseteq M$, a *neighborhood* of X is an open set in M which contains X. Also, Y is a neighborhood of X if and only if for every $x \in X$, there exists an $\epsilon > 0$ such that $B_\epsilon(x) \subseteq Y$. Given a subset X of M, an ϵ-neighborhood of X is the set $B_\epsilon(X) = \bigcup_{x \in X} B_\epsilon(x)$. Given a set X, we define the shrink and expand of the set as follows. For $X \subseteq M$, $\mathit{shrink}_\epsilon(X) = \{x \in M \mid B_\epsilon(x) \subseteq X\}$, and $\mathit{expand}_\epsilon(X) = \{x \in M \mid B_\epsilon(x) \cap X \neq \emptyset\}$, $\mathit{expand}_\epsilon(X)$ is essentially an ϵ-neighborhood of X.

2.1.3 Transition Systems

A transition system $\mathcal{T} = (S, S_0, \mathit{Act}, \mathit{Lab}, \{\rightarrow_a\}_{a \in \mathit{Act}}, \langle\!\langle \cdot \rangle\!\rangle)$, where:

- S is a set of states,
- $S_0 \subseteq S$ is a set of initial states,
- Act is a set of action labels,
- Lab is a set of state labels,
- $\rightarrow_a \subseteq S \times S$ is the transition relation, and
- $\langle\langle\cdot\rangle\rangle : S \rightarrow Lab$ is a state labelling function.

Notation 1 *We will often write $q_1 \xrightarrow{a}_{\mathcal{T}} q_2$ to mean $(q_1, q_2) \in \rightarrow_a$ (the subscript $\mathcal{T}$ is dropped when the transition system is clear from the context). Also we will drop some of the components in the definition of transition system when they are not required for the presentation of a particular result.*

Given sets of states $S_1, S_2 \subseteq S$ and a symbol $a \in Act$, $Pre_{\mathcal{T}}(S_2, a)$ is defined as the set $\{s_1 \,|\, \exists s_2 \in S_2 : s_1 \xrightarrow{a}_{\mathcal{T}} s_2\}$ and $Post_{\mathcal{T}}(S_1, a)$ as $\{s_2 \,|\, \exists s_1 \in S_1 : s_1 \xrightarrow{a}_{\mathcal{T}} s_2\}$. Given a subset Act' of Act, $Pre_{\mathcal{T}}(S_2, Act') = \bigcup_{a \in Act'} Pre_{\mathcal{T}}(S_2, a)$ and $Post_{\mathcal{T}}(S_1, Act') = \bigcup_{a \in Act'} Post_{\mathcal{T}}(S_1, a)$.

A transition system is finite branching if the set of successor states for any state and action is a finite set. A transition system is *finite branching* iff $\forall q \in S, a \in Act$, the set $\{q' | q \xrightarrow{a} q'\}$ is finite. A metric transition system is a transition system whose state labels are equipped with a metric. A *metric transition system* is a pair $(\mathcal{T}, d)$, where $\mathcal{T} = (S, S_0, Act, Lab, \{\rightarrow_a\}_{a \in Act}, \langle\langle\cdot\rangle\rangle)$ is a transition system and (Lab, d) is a metric space.

2.1.4 Simulation

Given transition systems $\mathcal{T}_1 = (S_1, S_{01}, Act, Lab, \{\rightarrow_a^1\}_{a \in Act}, \langle\langle\cdot\rangle\rangle_1)$ and $\mathcal{T}_2 = (S_2, S_{02}, Act, Lab, \{\rightarrow_a^2\}_{a \in Act}, \langle\langle\cdot\rangle\rangle_2)$, $R \subseteq S_1 \times S_2$ is said to be a *simulation relation* or just *simulation* between $\mathcal{T}_1$ and $\mathcal{T}_2$ if and only if for all $(q_1, q_2) \in R$:

1. $\langle\langle q_1 \rangle\rangle_1 = \langle\langle q_2 \rangle\rangle_2$, and
2. if $q_1 \xrightarrow{a}_1 q_1'$ then there is a q_2' s.t. $q_2 \xrightarrow{a}_2 q_2'$ and $(q_1', q_2') \in R$.

We will say that q_1 is simulated by q_2 or q_2 simulates q_1, denoted $q_1 \preceq q_2$, if there is some simulation R such that $(q_1, q_2) \in R$. Also, we say that, $\mathcal{T}_1$ is simulated by $\mathcal{T}_2$ or $\mathcal{T}_2$ simulates $\mathcal{T}_1$, denoted by $\mathcal{T}_1 \preceq \mathcal{T}_2$, if there exists a simulation relation R between $\mathcal{T}_1$ and $\mathcal{T}_2$ which satisfies:

1. for every $q_1 \in S_{01}$, there exists a $q_2 \in S_{02}$ such that $(q_1, q_2) \in R$.

Remark 2 *When the transition systems do not specify the set of initial states, then the set of initial states is taken to be the empty set; hence the above condition is trivially true in that case.*

2.2 Hybrid Systems

We give a formal definition of a *hybrid automaton* or a *hybrid system*, which is a popular model for modelling systems with mixed discrete-continuous behaviors. The discrete behavior is specified by a finite state transition system and the continuous behavior is modelled using a finite set of variables whose values evolve continuously with time. In our definition, we will not have variables explicitly, but will implicitly represent them by having a continuous state space of the appropriate dimension.

2.2.1 Syntax of hybrid systems

A *hybrid automaton* or *hybrid system* is a tuple $\mathcal{H} = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv, flow, guard, reset, flab)$ where:

- Loc is a finite set of locations,
- Act_H is a finite set of action labels,
- Lab_H is a finite set of location labels,
- $Edges \subseteq Loc \times Act_H \times Loc$ is a set of edges,
- $Cont = \mathbb{R}^n$ is the set of continuous states; and n is called the *dimension* of $\mathcal{H}$,
- loc_0 is the initial location,

- $Cont_0 \subseteq Cont$ is the initial set of continuous states,
- $inv : Loc \rightarrow 2^{Cont}$ is the function which associates an invariant with every location,
- $flow : Loc \times Cont \rightarrow (\mathbb{R}_{\geq 0} \rightarrow Cont)$ is the flow function,
- $guard : Edges \rightarrow 2^{Cont}$ associates a guard with every edge,
- $reset : Edges \rightarrow 2^{Cont \times Cont}$ is the function which associates a reset relation with every edge, and
- $flab : Loc \rightarrow Lab_H$ is the location labelling function.

Remark 3 *The components Loc, Act_H, Lab_H, Edges, loc_0, and flab define a finite state transition system and is often referred to as the underlying "control graph" of the hybrid system. The dimension of the hybrid system specifies the number of continuous entities or variables present in the system. We choose to specify the continuous statespace as a subset of $\mathbb{R}^n$ instead of explicitly using the variables for the sake of simplicity of presentation. However, to make the specifications more readable, in the informal pictorial representations of hybrid systems, we choose to represent the continuous part using variables. The sets inv(l), guard(l), and reset(e) specify constraints on the continuous variables by specifying the set of valid valuations of the variables. At this point, we leave the exact representation of the different components of the hybrid system open.*

Example 4 *Shown on the left in Figure 2.1 is the behavior of a controller for an autonomous car. The function of the controller is to keep the car close to the centre of the road which is shown in the diagram by the dark gray region. The car initially moves at a constant velocity r with a heading angle γ, and when it hits one of the edges of the dark gray region, the car corrects its path back to the dark gray region by executing a right turn along a circular path which is generated by the application of an angular velocity ω to its heading angle.*

The controller can be modelled as a hybrid automaton with two locations: one corresponding to the mode of the car in which it moves along a straight line and the other corresponding to the mode of the car in which it turns right. These locations are named "Go Ahead" and "Turn Right" in hybrid

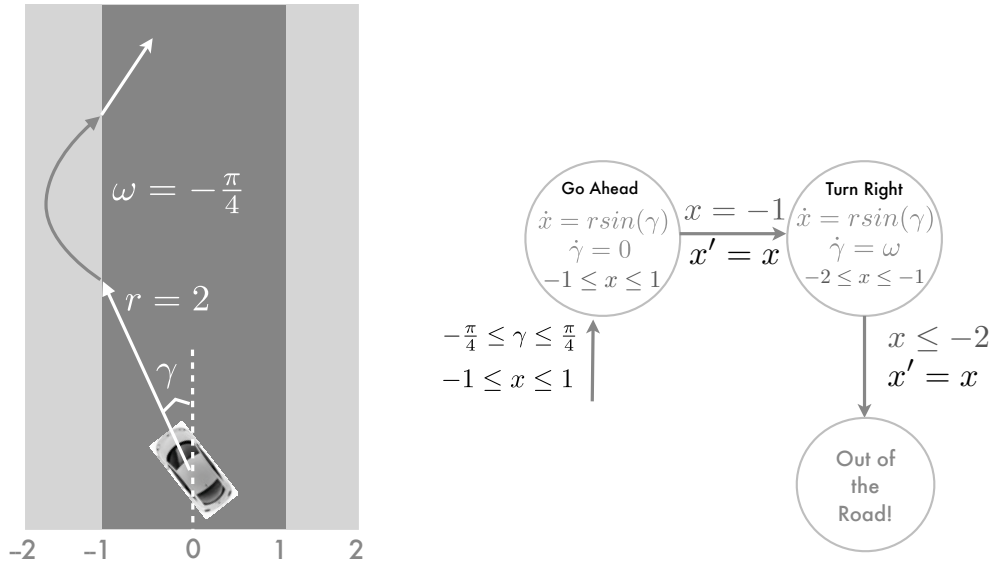

Figure 2.1: Car Controller and Hybrid Automaton Model

automaton model shown in Figure 2.1, respectively. The specification consists of two variables, namely, x and γ, where x represents the distance of the car from the center of the road, and γ represents the heading angle. Hence the continuous state space of the automaton is $\mathbb{R}^2$. It switches from the "Go Ahead" mode to the "Turn Right" mode when the car hits the edge of the dark gray region, which is reflected in the diagram by the guard $x = -1$ on the edge from "Go Ahead" to "Turn Right". The constraint $x = -1$ represents the set $\{(x, \gamma) \,|\, x = -1, \gamma \in \mathbb{R}\}$. The continuous dynamics is specified by the differential equations inside the circles of the corresponding locations. The solutions of these differential equations represent the flow functions in the corresponding locations. For example, the flow function associated with location "Go Ahead" is

$$flow(\text{“}GoAhead\text{”}, (x_0, \gamma_0), t) = (x_0 + r\sin(\gamma_0)t, \gamma_0).$$

That is, starting in the state (x_0, γ_0), the value of the continuous state after time t is $(x_0 + r\sin(\gamma_0)t, \gamma_0)$. Similarly, the

$$flow(\text{“}TurnRight\text{”}, (x_0, \gamma_0), t) = (x_0 + \frac{r}{\omega}\cos(\omega t), \gamma_0 + \omega t).$$

The invariant for the location "Go Ahead" is given by the constraint $-1 \leq x \leq 1$*, which represents the fact that the car remains in the center region of the road as long as the control is in location "Go Ahead". Similarly, the invariant for the location "Turn Right" is* $-2 \leq x \leq -1$*. The expression* $x' = x$ *represents the reset associated with the edge, where* x *is the value of the variable before the transition and* x' *is the value after the transition. Resets which are not shown in the diagram are assumed to be identity relations, that is,* $\gamma' = \gamma$ *on all the edges. In this example, the values of the variables don't change during a mode change, that is, the resets are all the identity relations. Note the the diagram only models a part of the controller for the car. A complete model will have additional locations, for example, to model the behavior of the car when it hits the left edge of the dark gray region. A safety property that we would expect the controller to satisfy is that it does not drive the car out of the road (shown by the light gray region in Figure 2.1), that is, it does not reach the "Out of the road!" mode in the model.*

2.2.2 Semantics of hybrid systems

Next we formally define the meaning of a hybrid automaton specification by presenting the transition system it represents. An execution of the model is then a path in this transition system.

Let us fix a hybrid system $\mathcal{H} = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv, flow, guard, reset, flab)$. The semantics of the hybrid system $\mathcal{H}$ is given in terms of a transition system whose statespace, denoted by $States(\mathcal{H})$, is given by $Loc \times Cont$. Given $(l, x) \in States(\mathcal{H})$, the location l is referred to as the discrete part of the state and x as the continuous part. The system can evolve in two ways, namely, by taking a discrete transition or by taking a continuous transition. A continuous transition does not change the discrete part; however, the continuous part changes according to the function $flow(l, x)$, remaining within the invariant set $inv(l)$ all along the evolution. A discrete transition could potentially change both the discrete and the continuous components of the state, and corresponds to executing an edge in *Edges*. The discrete transition is possible from a location l, if there is an edge e whose source is l, where for $e = (l, a, l')$, l is called the source of e, denoted $Source(e)$ and l' is called the target of e, denoted $Target(e)$. The

edge is enabled only if the continuous part of the state satisfies the guard of the edge. After taking an edge, the resulting state would consist of the target of the edge and the continuous state is such that the pair consisting of the continuous states before and after the transition satisfies the reset relation associated with the edge.

We present the formal semantics below. The semantics of $\mathcal{H}$ is given by the transition system $[\![H]\!] = (S, S_0, Act, Lab, \{\rightarrow_a\}_{a \in Act}, \langle\!\langle \cdot \rangle\!\rangle)$ where:

- $S = Loc \times Cont$,
- $S_0 = loc_0 \times Cont_0$,
- $Act = Act_H \cup \mathbb{R}_{\geq 0}$,
- $Lab = Lab_H \times \mathbb{R}^n$,
- $(l, x) \xrightarrow{a} (l', x')$ is either
  - a *discrete transition*, where $a \in Act_H$ and there exists $e = (l, a, l') \in Edges$ such that $x \in inv(l) \cap guard(e)$ and $(x, x') \in reset(e)$, or
  - a *continuous transition*, where $a \in \mathbb{R}_{\geq 0}$, $l = l'$ and there exist x_0, t_1 and t_2 such that $flow(l, x_0)(t_1) = x$, $flow(l, x_0)(t_2) = x'$, $a = t_2 - t_1$, and for all $t' \in [0, t_2]$, $flow(l, x_0)(t') \in inv(l)$ and
- $\langle\!\langle (l, x) \rangle\!\rangle = (flab(l), x)$.

Remark 5 *The above semantics of the transition system has been traditionally referred to as the timed semantics or the timed transition system of $\mathcal{H}$. The other popular semantics is the "time abstract semantics", which refers to the transition system which is similar to the above, except that the exact time on the continuous transitions is abstracted away, and this is achieved by replacing all transitions labelled by elements from $\mathbb{R}_{\geq 0}$ by a common symbol τ representing time evolution. We denote the time abstract semantics of $\mathcal{H}$ by $[\![\mathcal{H}]\!]_\tau$.*

We can associate the following natural metric on the state labels of $[\![\mathcal{H}]\!]$. Define metric d on $Lab = Lab_H \times \mathbb{R}^n$ as $d((p_1, x_1), (p_2, x_2)) = \infty$ if the location labels are not the same, that is, $p_1 \neq p_2$, and is equal to $\|x_1 - x_2\|$, the Euclidean distance between x_1 and x_2 otherwise. Then $([\![\mathcal{H}]\!], d)$ is a metric

transition system, with metric space (Lab, d). Unless specified otherwise, we will assume the above definition of d to be the default metric on the state labels of $[\![\mathcal{H}]\!]$.

CHAPTER 3

POLYNOMIAL APPROXIMATIONS

In this chapter, we present a technique for approximating a hybrid system with arbitrary flow functions by systems with polynomial flows; the verification of certain properties in systems with polynomial flows can be reduced to the first order theory of reals, and is therefore decidable. The polynomial approximations that we construct ϵ-simulate (as opposed to "simulate") the original system, and at the same time are tight. We show that for systems that we call *tolerant*, safety verification of a system can be reduced to the safety verification of the polynomial approximation. Our main technical tool in proving this result is a logical characterization of ϵ-simulations. We demonstrate the construction of the polynomial approximation, as well as the verification process, by applying it to an example protocol in air traffic coordination.

3.1 An Overview

In this section, we summarize the results of this chapter. Given a hybrid system $\mathcal{H}$ with arbitrary flows, we construct a hybrid system $poly_\epsilon(\mathcal{H})$ all of whose flows are polynomials[1], using the Stone-Weierstrass [95] theorem. Systems with polynomial flows are desirable, because for such systems, reachability in bounded executions can be reduced to the first order theory of reals, and is therefore decidable. The system $poly_\epsilon(\mathcal{H})$ that we construct, is not an abstraction of $\mathcal{H}$ in the traditional sense of exhibiting all the behaviors of $\mathcal{H}$. We show that $poly_\epsilon(\mathcal{H})$ ϵ-simulates (as introduced in [47]) $\mathcal{H}$. In other words, for every execution of $\mathcal{H}$, there is an execution of $poly_\epsilon(\mathcal{H})$ that remains within distance ϵ at all times. In addition, we show that our poly-

[1]Not only polynomials but any algebraically defined representations such as piece-wise polynomials or splines, etc.

nomial approximation is tight. More precisely, we show that $poly_\epsilon(\mathcal{H})$ itself is ϵ-simulated by an over-approximation of $\mathcal{H}$. Thus, $poly_\epsilon(\mathcal{H})$ has approximations to every behavior of $\mathcal{H}$ but not much more. The fact that $poly_\epsilon(\mathcal{H})$ is a tight approximation, allows us to conclude that verifying $poly_\epsilon(\mathcal{H})$ gives us a precise answer about the safety of $\mathcal{H}$, for certain special systems that we call *tolerant*.

An ϵ-tolerant system, intuitively is one where even if the invariants, guards and resets are perturbed slightly (by ϵ), the system remains safe. Tolerance is a desirable property of a system, and accounts for external disturbances and inaccuracies in modelling parameters. Usually good designs are tolerant. Our main result characterizes how the safety of tolerant systems can be determined by analyzing its polynomial approximation. We show that for a 2ϵ-tolerant system $\mathcal{H}$, $\mathcal{H}$ is safe if and only if $poly_\epsilon(\mathcal{H})$ is safe. Thus, in the case of tolerant systems, the flows can be reliably simplified without affecting the verification result.

This begs the question, how do we know if the system we start with is tolerant? We observe that even if the tolerance of a system $\mathcal{H}$ is unknown, analyzing $poly_\epsilon(\mathcal{H})$ gives useful information. Our proof shows that if $poly_\epsilon(\mathcal{H})$ is safe, then $\mathcal{H}$ is guaranteed to be safe, very much like the case of traditional abstractions. On the other hand, if $poly_\epsilon(\mathcal{H})$ is unsafe then it is either the case that $\mathcal{H}$ is unsafe or it is not 2ϵ-tolerant. Thus, if ϵ is small, it suggests that $\mathcal{H}$ is badly designed and must be modified, independent of whether it is actually safe.

Our result reducing the safety verification of tolerant systems to the verification of polynomial approximations, relies on a logical characterization of ϵ-simulations. Our characterization is remarkably similar to the logical characterization of (classical) simulation using Hennessy-Milner logics [72]. This is surprising in the light of the fact that ϵ-simulation is not a preorder as it is not transitive. Further, as in the case of simulations, our characterization is exact for finite branching transition systems. This logical characterization of ϵ-simulation maybe of independent interest.

Finally, we apply our technique to the verification of a protocol in air traffic coordination, demonstrating all the steps in our approach, including the construction of polynomial approximations and their verification.

We will discuss some of the techniques in literature on approximating complex continuous dynamics by simpler flows. In [93], a technique for approx-

imating arbitrary differential inclusions whose right hand side is a Lipschitz continuous function is given. The method produces rectangular hybrid systems approximating the continuous dynamics for a given precision. The approximation of a non-linear dynamics by a piecewise linear dynamics has been considered in [52]. Approximation of the continuous dynamics by polynomials has also been considered in [67]. This method is based on Taylor series approximation. Since, we do not insist on any particular method for constructing the polynomial approximation, our method applies to a more general class of systems than considered in [67]. Another difference between the two approaches is that our approximation technique approximates a general flow by a polynomial flow, which also preserves determinism of the flow function. However, no such guarantee is given by the method in [67].

The above approaches produce systems which "abstract" or "overapproximate" the original system. That is, the abstract system includes every behavior of the original system, and possibly many more. In contrast, our approximation method produces systems which have the property that for every execution of the original systems, there is an execution "close" to it in the approximate system. This notion of "close simulation" or ϵ-simulation was first introduced in [47], and a characterization of this notion in terms of simulation functions was given. In [49], the authors present a technique for computing approximately bisimilar systems using Lyapunov function.

3.2 Preliminaries

3.2.1 First-Order Logic

Let τ be a vocabulary and $\mathcal{A}$ a τ-structure. Let A be the domain of $\mathcal{A}$. A k-ary relation $S \subseteq A^k$ is definable in $\mathcal{A}$ if there is a first-order formula $\varphi(x_1, x_2, \ldots x_k)$ over τ with free variables $x_1, \ldots x_k$, such that $S = \{(a_1, \ldots, a_k) \,|\, \mathcal{A} \models \varphi[x_i \mapsto a_i]_{i=1}^k\}$. A k-ary function f will be said to be definable if its graph, i.e., the set of all $(x_1, \ldots, x_k, f(x_1, \ldots x_k))$, is definable. A *theory* $Th(\mathcal{A})$ of a structure $\mathcal{A}$ is the set of all sentences that hold in $\mathcal{A}$. $Th(\mathcal{A})$ is said to be *decidable* if there is an effective procedure to decide membership in the set $Th(\mathcal{A})$.

In this chapter, we consider the theory of real-closed fields, namely, the

set of all sentences true over $(\mathbb{R}, 0, 1, +, \cdot, <)$, denoted $Th(\mathbb{R})$, where $\mathbb{R}$ is the set of real numbers and 0, 1, +, . and < have the standard interpretations of the constant 0, constant 1, addition, multiplication and linear order over the real numbers. When we refer to a first-order formula over the reals, we mean a formula over $(\mathbb{R}, 0, 1, +, \cdot, <)$. We know from Tarski's theorem that $Th(\mathbb{R})$ admits quantifier elimination and hence it is decidable.

Theorem 6 (Tarski's theorem[99]) *The theory of real-closed fields $Th(\mathbb{R})$ is decidable.*

We say that a hybrid system $\mathcal{H}$ is definable in the structure $(\mathbb{R}, 0, 1, +, \cdot, <$), if the sets $Cont_0$, $inv(l)$ for all $l \in Loc$, $guard(e)$ and $reset(e)$ for all $e \in Edges$, and the functions $flow(l)$ for $l \in Loc$, are definable in the structure $(\mathbb{R}, 0, 1, +, \cdot, <)$. Note that a polynomial function is definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$.

3.2.2 Stone- Weierstrass Theorem

A *real function* on a set E is a function $f : E \to \mathbb{R}$. A family $\mathbb{A}$ of real functions defined on a set E is said to be an *algebra* if for all $f, g \in \mathbb{A}$ and $r \in \mathbb{R}$, $f + g \in \mathbb{A}$, $fg \in \mathbb{A}$, $rf \in \mathbb{A}$, where $(f + g)(x) = f(x) + g(x)$, $(fg)(x) = f(x).g(x)$ and $(rf)(x) = r.f(x)$. A sequence of functions $\{f_n\}, n = 1, 2, 3, \cdots$, *converges uniformly* on E to a function f if for every $\epsilon > 0$, there is an integer N such that $n \geq N$ implies $|f_n(x) - f(x)| < \epsilon$ for all $x \in E$. Let $\mathbb{B}$ be the set of all functions which are limits of uniformly convergent sequences of members of $\mathbb{A}$. Then $\mathbb{B}$ is called the uniform closure of $\mathbb{A}$. Let $\mathbb{A}$ be a family of functions on a set E. Then $\mathbb{A}$ is said to *separate points* on E if for every pair of distinct points $x_1, x_2 \in E$, there corresponds a function $f \in \mathbb{A}$ such that $f(x_1) \neq f(x_2)$. If for each $x \in E$, there corresponds a function $g \in \mathbb{A}$ such that $g(x) \neq 0$, $\mathbb{A}$ is said to *vanish at no point* in E.

Theorem 7 (Stone-Weierstrass) *Let $\mathbb{A}$ be an algebra of real continuous functions on a compact set $\mathbb{K}$. If $\mathbb{A}$ separates points on $\mathbb{K}$ and if $\mathbb{A}$ vanishes at no point of $\mathbb{K}$, then the uniform closure $\mathbb{B}$ of $\mathbb{A}$ consists of all real continuous functions on $\mathbb{K}$.*

Since the set of polynomial functions form an algebra, every arbitrary function is the limit of a uniformly converging sequence of polynomial functions.

Corollary 8 *Given any continuous function $f : \mathbb{R}^n \to \mathbb{R}^m$, a compact subset $\mathbb{K}$ of $\mathbb{R}^n$ and an $\epsilon > 0$, there exists a polynomial function $P : \mathbb{R}^n \to \mathbb{R}^m$ such that*

$$\|f(x) - P(x)\| < \epsilon, \forall x \in \mathbb{K}.$$

We will use this theorem to approximate arbitrary functions by polynomial functions.

Definition 9 *Given a function $f : \mathbb{R}^n \to \mathbb{R}^m$, a compact subset $\mathbb{K}$ of $\mathbb{R}^n$, we define $poly_\epsilon(f, \mathbb{K})$ to be some polynomial function obtained by the above Corollary.*

3.2.3 ϵ-Simulations

We now define the notion of approximate simulation, which is similar to the notion of simulation except that we do not require the state labels to match for related states but only require the distance between the state labels to be small. Given metric transition systems $\mathcal{T}_1 = (S_1, Act, Lab, \{\rightarrow_a^1\}_{a \in Act}, \langle\langle\cdot\rangle\rangle_1)$ and $\mathcal{T}_2 = (S_2, Act, Lab, \{\rightarrow_a^2\}_{a \in Act}, \langle\langle\cdot\rangle\rangle_2)$ with a distance function d on Lab, $R \subseteq S_1 \times S_2$ is said to be an ϵ-simulation between $\mathcal{T}_1$ and $\mathcal{T}_2$ if and only if for all $(q_1, q_2) \in R$:

1. $d(\langle\langle q_1 \rangle\rangle_1, \langle\langle q_2 \rangle\rangle_2) < \epsilon$, and
2. if $q_1 \xrightarrow{a}_1 q_1'$ then there is a q_2' s.t. $q_2 \xrightarrow{a}_2 q_2'$ and $(q_1', q_2') \in R$.

We will say that $q_1 \preceq_\epsilon q_2$ if there is some ϵ-simulation R such that $(q_1, q_2) \in R$.

3.3 Logical Characterization of Simulation

In this section we present the logical characterization of simulation in terms of safe Hennessy-Milner Logic and extend it to obtain a logical characterization of ϵ-simulation.

3.3.1 Safe Hennessy-Milner Logic

Given an alphabet *Act* and a set of labels *Lab*, we denote the *Safe Hennessy-Milner Logic* formulas over (Act, Lab) as $SHM(Act, Lab)$. The formulas in

SHM(*Act*, *Lab*) are defined inductively as:

$$\phi ::= p \mid [a]\phi \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2,$$

where $p \subseteq Lab$ is an atomic proposition and $a \in Act$.

The semantics of Safe Hennessy Milner is defined as follows. Given a transition system $\mathcal{T}$, a state q of it, and a formula ϕ over *SHM*(*Act*, *Lab*), where *Act* is the set of action labels and *Lab*, the set of state labels of $\mathcal{T}$, we define $\mathcal{T}$ at q satisfies ϕ, denoted $\mathcal{T}, q \models \phi$, inductively as:

$$\begin{array}{lcl}
\mathcal{T}, q & \models & p \text{ iff } \langle\langle q \rangle\rangle \in p, \\
\mathcal{T}, q & \models & [a]\phi \text{ iff } \forall q', \ q \xrightarrow{a} q' \Rightarrow \mathcal{T}, q' \models \phi, \\
\mathcal{T}, q & \models & \phi_1 \wedge \phi_2 \text{ iff } \mathcal{T}, q \models \phi_1 \wedge \mathcal{T}, q \models \phi_2, \\
\mathcal{T}, q & \models & \phi_1 \vee \phi_2 \text{ iff } \mathcal{T}, q \models \phi_1 \vee \mathcal{T}, q \models \phi_2.
\end{array}$$

We say that a transition system $\mathcal{T}$ satisfies a formula ϕ, denoted $\mathcal{T} \models \phi$ if $\mathcal{T}, q \models \phi$ for all initial states q of $\mathcal{T}$. For a state q in the transition system $\mathcal{T}$, define $[\![q]\!]_{\mathcal{T}} = \{\phi \in SHM(Act, Lab) \mid \mathcal{T}, q \models \phi\}$. Let q_1 be a state in $\mathcal{T}_1$ and q_2 be a state in $\mathcal{T}_2$. We say that q_1 is *SHM* simulated by q_2 denoted $q_1 \sqsubseteq_{SHM} q_2$, if $[\![q_2]\!]_{\mathcal{T}_2} \subseteq [\![q_1]\!]_{\mathcal{T}_1}$.

Remark 10 *When $Lab \subseteq \mathbb{R}^k$, we say that $\phi \in SHM(Act, Lab)$ is definable in $(\mathbb{R}, 0, 1, <, +, \cdot)$, if every proposition of ϕ is definable in $(\mathbb{R}, 0, 1, <, +, \cdot)$.*

Next, we present a logical characterization of simulation due to Milner.

Proposition 11 ([72]) *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two transition systems and let q_1 be a state of $\mathcal{T}_1$ and q_2 be a state of $\mathcal{T}_2$. Then:*

1. *$q_1 \preceq q_2$ implies $q_1 \sqsubseteq_{SHM} q_2$*
2. *$\mathcal{T}_2$ is finite branching and $q_1 \sqsubseteq_{SHM} q_2$ implies $q_1 \preceq q_2$.*

The proof is standard and skipped.

3.3.2 Logical Characterization of ϵ-Simulation

In this section we give a logical characterization of ϵ-simulation along the lines of that for simulation given by Milner. We require the notion of the

shrink of a formula. Intuitively, the shrink of a formula is satisfied by some valuation if the original formula is satisfied by all the valuations in an ϵ ball around it. Let (Lab, d) be a metric space. For a formula $\phi \in SHM(Act, Lab)$ we define $shrink_\epsilon(\phi)$ inductively as follows:

- $\phi = p$, where $p \subseteq Lab$: $shrink_\epsilon(\phi) = shrink_\epsilon(p)$. That is, shrink of the formula ϕ is the same as the shrink of the set p.
- $\phi = [a]\psi : shrink_\epsilon(\phi) = [a]shrink_\epsilon(\psi)$.
- $\phi = \psi_1 \wedge \psi_2 : shrink_\epsilon(\phi) = shrink_\epsilon(\psi_1) \wedge shrink_\epsilon(\psi_2)$.
- $\phi = \psi_1 \vee \psi_2 : shrink_\epsilon(\phi) = shrink_\epsilon(\psi_1) \vee shrink_\epsilon(\psi_2)$.

Observe that $shrink_\epsilon(shrink_\epsilon(\phi)) = shrink_{2\epsilon}(\phi)$. For a set of formulas Γ, $shrink_\epsilon(\Gamma) = \{\phi \mid shrink_\epsilon(\phi) \in \Gamma\}$.

We first generalize the notion of q_1 is *SHM* simulated by q_2 to q_1 is ϵ-*SHM* simulated by q_2 using the shrink of formulas. We assume for the rest of the section that $\mathcal{T}_1$ and $\mathcal{T}_2$ are metric transition systems with *Lab*, the set of state labels, *Act*, the set of action labels and d the distance function.

Definition 12 *For a state q_1 in $\mathcal{T}_1$ and a state q_2 in $\mathcal{T}_2$ we say $q_1 \sqsubseteq^\epsilon_{SHM} q_2$ iff $shrink_\epsilon([\![q_2]\!]_{\mathcal{T}_2}) \subseteq [\![q_1]\!]_{\mathcal{T}_1}$.*

We now logically characterize ϵ-simulation by relating it to ϵ-*SHM* simulation.

Theorem 13 *Let $\mathcal{T}_1$ and $\mathcal{T}_2$ be two metric transition systems. Let q_1 and q_2 be states in $\mathcal{T}_1$ and $\mathcal{T}_2$ respectively. Then*

1. *$q_1 \preceq_\epsilon q_2 \Rightarrow q_1 \sqsubseteq^\epsilon_{SHM} q_2$.*
2. *$\mathcal{T}_2$ is finite branching and $q_1 \sqsubseteq^\epsilon_{SHM} q_2 \Rightarrow q_1 \preceq_\epsilon q_2$.*

Proof

Proof of part (i). Let $q_1 \preceq_\epsilon q_2$. We will show by structural induction on ϕ that if $\mathcal{T}_2, q_2 \models shrink_\epsilon(\phi)$ then $\mathcal{T}_1, q_1 \models \phi$. Then we can conclude that $q_1 \sqsubseteq^\epsilon_{SHM} q_2$.

Base case: $\phi = p \subseteq Lab$. $T_2, q_2 \models shrink_\epsilon(\phi)$ implies $\langle\!\langle q_2 \rangle\!\rangle \in shrink_\epsilon(p)$. Since $q_1 \preceq_\epsilon q_2$ we know that $d(\langle\!\langle q_1 \rangle\!\rangle, \langle\!\langle q_2 \rangle\!\rangle) < \epsilon$ and hence $\langle\!\langle q_1 \rangle\!\rangle \in p$. Therefore, $\mathcal{T}_1, q_1 \models \phi$.

Induction step: In the case of $\phi = \psi_1 \vee \psi_2$ or $\phi = \psi_1 \wedge \psi_2$ the proof is straightforward. Hence we consider the case when $\phi = [a]\psi$. $shrink_\epsilon(\phi) = [a]shrink_\epsilon(\psi)$. Now suppose $q_1 \xrightarrow{a} q_1'$. Then $\exists q_2' \; . \; q_2 \xrightarrow{a} q_2' \wedge q_1' \preceq_\epsilon q_2'$. Further, since $q_2 \models [a]shrink_\epsilon(\psi)$, we have $q_2' \models shrink_\epsilon(\psi)$. By induction hypothesis $\mathcal{T}_1, q_1' \models \psi \Rightarrow q_1 \models [a]\psi$.

Proof of part (ii). Suppose $q_1 \sqsubseteq^\epsilon_{SHM} q_2$. We will show that $\sqsubseteq^\epsilon_{SHM}$ is an ϵ-simulation.

(a) Let $\langle\langle q_2 \rangle\rangle = l$. Clearly $q_2 \models shrink_\epsilon(B_\epsilon(l))$. Therefore $q_1 \models B_\epsilon(l) \Rightarrow d(\langle\langle q_1 \rangle\rangle, \langle\langle q_2 \rangle\rangle) < \epsilon$.

(b) Suppose $q_1 \xrightarrow{a} q_1'$. There must be some q_2' such that $q_2 \xrightarrow{a} q_2'$ and $q_1' \sqsubseteq^\epsilon_{SHM} q_2'$. If not, consider $NotSim = \{q_2' | q_2 \xrightarrow{a} q_2' \; and \; q_1' \not\sqsubseteq^\epsilon_{SHM} q_2'\}$. Now, for every $q_2' \in NotSim$, by definition of $\sqsubseteq^\epsilon_{SHM}$, there is a formula $\phi_{q_2'}$, such that $q_2' \models shrink_\epsilon(\phi_{q_2'})$ and $q_1' \not\models \phi_{q_2'}$. Take $\phi = [a] \bigvee_{q_2' \in NotSim} \phi_{q_2'}$. Now $shrink_\epsilon(\phi) = [a] \bigvee_{q_2' \in NotSim} shrink_\epsilon(\phi_{q_2'})$. Since $NotSim$ contains all a-successors of q_2, $\mathcal{T}_2, q_2 \models shrink_\epsilon(\phi)$. But since $q_1' \not\models \phi_{q_2'} \;\; \forall q_2' \in NotSim$, we have $q_1 \not\models \phi$, which contradicts the fact that $q_1 \sqsubseteq^\epsilon_{SHM} q_2$. ∎

Remark 14 *In this work we consider SHM logic instead of ACTL or ACTL* because we are interested in bounded-horizon verification. Theorem 13 can be extended in a straightforward manner to those logics.*

Remark 15 *In general, the transition systems arising from hybrid systems need not be finite branching due to non-determinism in either the discrete or continuous transitions. However, for the purpose of verification, we only need part (1) of Theorem 13.*

3.4 Polynomial Approximations and Approximate Simulations

In this section, we present a technique for constructing a polynomial hybrid system from a general hybrid system, and show that the approximate system approximately simulates the original system.

Let us fix a hybrid system $\mathcal{H} = (Loc, Act_H, Lab_H, \; Edges, Cont, loc_0, Cont_0, inv, flow, \; guard, reset, flab)$ for the rest of this section.

Definability We assume that various components of a hybrid system $\mathcal{H}$ are definable in the structure $(\mathbb{R}, 0, 1, +, \cdot, <)$. More precisely, the sets $Cont_0$, $inv(l)$ for $l \in Loc$, $guard(e)$ for $e \in Edges$, $reset(e)$ for $e \in E$ and the function $flow(l) : Cont \times \mathbb{R}_{\geq 0} \rightarrow Cont$ are all definable in the structure $(\mathbb{R}, 0, 1, +, \cdot, <)$.

Time Independence We will assume that the flow function is *time independent* by which we mean that if x_1 is the state reached starting from x_0 after time t_1 has elapsed and x_2 is the state reached starting from x_1 in time t_2, then the state reached starting from x_0 after time $t_1 + t_2$ is exactly x_2. Intuitively, the property implies that the behavior of the system from state x_1 is independent of how or when x_1 was reached. Formally, the flow function $flow : Loc \times Cont \rightarrow (\mathbb{R}_{\geq 0} \rightarrow Cont)$ is said to be *time independent* if for every $l \in Loc$ and $x \in Cont$, $flow(l, x)$ satisfies the following conditions:

1. $flow(l, x)$ is continuous and $flow(l, x)(0) = x$.
2. For every $t_1, t_2 \geq 0$, $flow(l, x)(t_1 + t_2) = flow(l, flow(l, x)(t_1))(t_2)$.

Time independent flows naturally arise as the solutions of differential equations whose right hand side is time invariant. The time independence of the flow function is desirable, since the existence of finite bisimulation and decidability is known for certain classes of polynomial systems exhibiting this property [17, 103]. In this chapter, we will only be considering systems with time independent flows.

3.4.1 ϵ-Polynomial Approximations

We give the definition of the polynomial approximation of a general hybrid system and prove some properties of this approximate system. We make use of the Stone-Weierstrass theorem to approximate complex continuous dynamics of the hybrid systems by polynomials. We assume that the invariants associated with the locations are compact; and that the flows are such that there is a bound on the time that can be spent in any location.

Assumptions 16

- *Let $\mathcal{H}$ be such that $inv(l)$ is compact for all l in Loc.*

- *There exists a time bound* $t_b \in \mathbb{R}_{\geq 0}$ *such for any* $l \in Loc$, $x \in Cont$, *and any time transition* $(l, x) \xrightarrow{t} (l, x')$ $t \leq t_b$.

We note that in many classes of systems such as timed automata, rectangular hybrid automata, and so on, the monotonicity of flows in a location ensures that there is a bound on the time spent in any location with compact invariant.

As we said earlier, time independence of the polynomial approximation is a desirable property since various classes of polynomial systems with this property are amenable to analysis [17, 103]. A straightforward approximation of the flow function to a polynomial function of n dimensions does not in general preserve this property. To preserve this property in our approximations, we consider a statespace with $2n + 1$ dimensions, where the extra $n + 1$ dimensions are used to remember the value of the continuous state of the current continuous execution at time 0, and the time elapsed since the beginning of the continuous evolution, respectively. We approximate the flows of the hybrid system with polynomial functions which are ϵ-close at all time using Stone-Weierstrass Theorem; this is possible since the invariants are compact and we have a bound on the time spent in any location. Note that the approximate flow can violate the invariant even when the corresponding original flow did not; similarly, the approximate flow can end in a state outside the guard of a set even when the original flow did not. However, the approximate flows will not deviate more than ϵ from the flows of $\mathcal{H}$, and hence we expand the invariants, guards and resets by an ϵ to accommodate these flows.

Definition 17 (ϵ-Polynomial Approximation) *Given a hybrid system* $\mathcal{H} = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv, flow, guard, reset, flab)$ *satisfying the Assumptions 16, we define the* ϵ*-polynomial approximation of* $\mathcal{H}$, *denoted by* $poly_\epsilon(\mathcal{H})$, *as the hybrid system* $(Loc, Act_H, Lab_H, Edges, Cont', loc_0, Cont'_0, inv', flow', guard', reset', flab)$ *where:*

- $Cont' = \mathbb{R}^{2n+1}$, *where* $Cont = \mathbb{R}^n$,
- $Cont'_0 = \{(x, 0, x) \mid x \in Cont_0\}$,
- $inv'(l) = inv(l) \times \{t \mid t \in \mathbb{R}_{\geq 0}, t \leq t_b\} \times expand_\epsilon(inv(l))$,
- $flow'(l, (x_0, t_0, x))(t) = (x_0, t_0 + t, poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, t_0 + t))$,

- $guard'(e) = inv(l) \times \{t \mid t \in \mathbb{R}_{\geq 0}, t \leq t_b\} \times expand_\epsilon(guard(e))$, *where* $e = (l, a, l')$,
- $reset'(e) = \{((x_1, t_1, y_1), (x_2, t_2, y_2)) \mid t_2 = 0, x_2 = y_2, \exists y_1', \|y_1 - y_1'\| < \epsilon \wedge (y_1', y_2) \in reset(e) \wedge y_2 \in inv(l')\}$, *for* $e = (l, a, l')$,

Remark 18 *We will assume that* $flow'(l, (x_0, 0, x_0))(0) = (x_0, 0, x_0)$ *or equivalently* $poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, 0) = x_0$. *(This can be achieved by considering the polynomial* $P - P_0$ *in the construction of* $flow'$*, where* P *is* $Poly_{\epsilon/2}(flow(l), inv(l) \times \{t_b\})$ *and* P_0 *is the polynomial obtained by substituting* t *by* 0 *in* P*.)*

The next proposition says that the polynomial approximation of $\mathcal{H}$ is time independent, given that $\mathcal{H}$ is time independent.

Proposition 19 *The* ϵ*-polynomial approximation* $poly_\epsilon(\mathcal{H})$ *has time independent flows.*

Proof It follows from the definition of $flow'$. More precisely,

$$\begin{aligned}
&flow'(l, flow'(l, (x_0, t_0, x))(t_1))(t_2) \\
&= flow'(l, (x_0, t_0 + t_1, poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, t_0 + t_1)))(t_2) \\
&= (x_0, (t_0 + t_1) + t_2, poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, (t_0 + t_1) + t_2))) \\
&= (x_0, t_0 + (t_1 + t_2), poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, t_0 + (t_1 + t_2)))) \\
&= flow'(l, (x_0, t_0, x))(t_1 + t_2).
\end{aligned}$$

∎

Next we show that $\mathcal{H}$ is ϵ-simulated by its ϵ-polynomial approximation. In the following, we will abuse notation to use $[\![poly_\epsilon(\mathcal{H})]\!]$ to mean the time transition system semantics of $poly_\epsilon(\mathcal{H})$ with the following difference: $\langle\!\langle (l, (x, t, x')) \rangle\!\rangle = (flab(l), x')$.

Theorem 20 *For all* $\epsilon > 0$, $[\![\mathcal{H}]\!] \preceq_\epsilon [\![poly_\epsilon(\mathcal{H})]\!]$.

Proof We define a relation $R \subseteq (Loc \times \mathbb{R}^n) \times (Loc \times \mathbb{R}^{2n+1})$ as follows:

$$((l, x), (l', (x_0, t_1, x_1))) \in R \text{ iff}$$

$$l = l', flow(l, x_0)(t_1) = x \text{ and } (x_0, t_1, x_1) = flow'(l, (x_0, 0, x_0))(t_1).$$

We will show that R is a simulation relation from $[\![\mathcal{H}]\!]$ to $[\![poly_\epsilon(\mathcal{H})]\!]$. Note that in the above $x_1 = poly_\epsilon(flow(l), inv(l) \times \{t_b\})(x_0, 0 + t_1))$, which implies

$\|\mathit{flow}(l,x_0)(t_1) - x_1\| < \epsilon$ or equivalently $d(\langle\!\langle (l,x)\rangle\!\rangle, \langle\!\langle (l,(x_0,t_1,x_1))\rangle\!\rangle) < \epsilon$ (here d is the metric on $Lab_H \times \mathbb{R}^n$ defined in Section 2.2).

Let us first consider the case of a continuous transition. Let $(l,x) \xrightarrow{t} (l,x')$ be a continuous transition. We need to show that for every (x_0,t_1,x_1) such that $(l,x)R(l,(x_0,t_1,x_1))$ there is a (x_0',t_1',x_1') such that $(l,(x_0,t_1,x_1)) \xrightarrow{t}$ $(l,(x_0',t_1',x_1'))$ and $(l,x')R(l,(x_0',t_1',x_1'))$. Let (x_0',t_1',x_1') be given by $\mathit{flow}'(l,$ $(x_0,0,x_0))(t_1+t)$. We claim that $(l,(x_0,t_1,x_1)) \xrightarrow{t} (l,(x_0',t_1',x_1'))$ and $(l,x')R(l,(x_0',t_1',x_1'))$ hold. From the definition $x_0' = x_0$, $t_1' = t_1 + t$ and $x_1' = \mathit{poly}_\epsilon(\mathit{flow}(l), \mathit{inv}(l) \times \{t_b\})(x_0, t_1+t)$. Note that for all $t_1 \le t' \le t_1 + t$, $d(\langle\!\langle \mathit{flow}(l,x_0)(t')\rangle\!\rangle, \langle\!\langle \mathit{flow}'(l,(x_0,t_1,x_1))(t')\rangle\!\rangle) < \epsilon$. The above claim follows from the above observation along with the fact that the invariants are expanded by ϵ.

Suppose that there is a discrete transition from (l,x) to (l',x') along edge $e = (l,a,l')$ and let $(l,x)R(l,(x_0,t_1,x_1))$. Then there is an edge from $(l,(x_0,t_1,x_1))$ to $(l',(x',0,x'))$. To see this, first observe that we can infer that $d(\langle\!\langle (l,x)\rangle\!\rangle, \langle\!\langle (l,(x_0,t_1,x_1))\rangle\!\rangle) < \epsilon$ from the definition of R. Further, since we expand the guards on the edges of $\mathcal{H}$ by ϵ in $\mathit{poly}_\epsilon(\mathcal{H})$, we have that $(l,(x_0,t_1,x_1))$ satisfies the guard of the edge e in $\mathit{poly}_\epsilon(\mathcal{H})$. Also, since $(x,x') \in \mathit{reset}(e)$, $((x_0,t_1,x_1),(x',0,x')) \in \mathit{reset}'(e)$ because $\|x - x_1\| < \epsilon$. Finally, since $(l,x')R(l,(x',0,x'))$, we are done. ∎

Next we use Theorem 13 to deduce that if the polynomial approximation satisfies a modified property then we can conclude that the original system satisfies the property.

Corollary 21 *Given a SHM formula ϕ over $(Act_H \cup \mathbb{R}_{\ge 0}, Lab_H \times \mathbb{R}^n)$, we have $[\![\mathit{poly}_\epsilon(\mathcal{H})]\!] \models \mathit{shrink}_\epsilon(\phi)$ implies $[\![\mathcal{H}]\!] \models \phi$.*

Remark 22 *Note that Theorem 20 also hold in the time-abstract semantics, that is, $[\![\mathcal{H}]\!]_\tau \preceq_\epsilon [\![\mathit{poly}_\epsilon(\mathcal{H})]\!]_\tau$. Hence, given a SHM formula ϕ over $(Act_H \cup \{\tau\}, Lab_H \times \mathbb{R}^n)$, we have $[\![\mathit{poly}_\epsilon(\mathcal{H})]\!]_\tau \models \mathit{shrink}_\epsilon(\phi)$ implies $[\![\mathcal{H}]\!]_\tau \models \phi$.*

A *SHM* formula ϕ over the set of labels $Lab_H \times \mathbb{R}^n$ is said to be definable in $(\mathbb{R},0,1,+,\cdot,<)$ if for each atomic formula $p \subseteq Lab_H \times \mathbb{R}^n$ occurring in ϕ, the set p_u is definable in $(\mathbb{R},0,1,+,\cdot,<)$ for each $u \in Lab_H$, where $p_u = \{x \mid (u,x) \in p\}$. The next proposition says that the model-checking problem is decidable for a hybrid system and a *SHM* formula which are definable in the structure $(\mathbb{R},0,1,+,\cdot,<)$, since the problem can be reduced to the

satisfiability problem of a formula in the structure $(\mathbb{R}, 0, 1, +, \cdot, <)$, which is decidable due to Tarski's theorem. This depends crucially on the fact that ϕ expresses a bounded property.

Proposition 23 *Given a hybrid system $\mathcal{H}$ and a SHM formula ϕ which are both definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$, the problem of whether $[\![\mathcal{H}]\!] \models \phi$ is decidable.*

Observe that if all the elements of $\mathcal{H}$ except *flow* are definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$ and a *flow'* defined in $poly_\epsilon(\mathcal{H})$ can be constructed, then $poly_\epsilon(\mathcal{H})$ is definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$. Similarly, $shrink_\epsilon(\phi)$ is definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$ provided ϕ is, and can be effectively constructed from ϕ. Hence, we obtain as a corollary of Proposition 23 that we can decide if $poly_\epsilon(\mathcal{H})$ satisfies ϕ.

Lemma 1 *Given $\mathcal{H}$, ϕ and ϵ which are definable in $(\mathbb{R}, 0, 1, +, \cdot, <)$, the problem of whether $[\![poly_\epsilon(\mathcal{H})]\!] \models shrink_\epsilon(\phi)$ is decidable.*

Using Corollary 21, we can conclude that a system is safe by constructing its polynomial approximation and checking whether it is safe (which can be automated due to Lemma 1).

Remark 24 *In fact, we can extend Theorem 13 to ACTL* formulas. But, ϕ in this case can potentially express an unbounded property and hence we can extend Proposition 23 directly for ACTL* formulas. However, if $\mathcal{H}$ has the "strong reset" property where resets are of the form $X_1 \times X_2$, $X_1, X_2 \subseteq \mathbb{R}^n$, then the resulting polynomial approximation has a special form (it is an o-minimal system) for which verification with respect to ACTL* is decidable [18].*

3.4.2 Tightness of the Polynomial Approximation

We show that our polynomial approximation is tight by showing that it is approximately simulated by an ϵ "expansion" of the original system. This implies that though the polynomial approximation has potentially more behaviors than the original system, it has does not have many more than a small perturbation of the original system. The ϵ-expansion of a hybrid system is obtained by expanding the invariants, guards and resets by ϵ, but leaving the flows untouched.

Definition 25 *Given a hybrid system* $\mathcal{H} = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv, flow, guard, reset, flab)$*, we define the* ϵ*-expansion of* $\mathcal{H}$*, denoted by* $expand_\epsilon(\mathcal{H})$*, as the hybrid system* $(Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv', flow, guard', reset', flab)$ *where:*

- $inv'(l) = expand_\epsilon(inv(l))$*, for all* $l \in Loc$*,*
- $guard'(e) = expand_\epsilon(guard(e))$*, for all* $e \in Edges$*, and*
- $reset'(e) = \{((x, y) \mid \exists x', d(x, x') < \epsilon \wedge (x', y) \in reset(e)\}$*, for all* $e \in Edges$*.*

It is easy to see that the ϵ-expansion of a system ϵ-simulates the system, since $[\![expand_\epsilon(\mathcal{H})]\!]$ is an "abstraction" in the traditional sense or has all the "behaviors" of the $[\![\mathcal{H}]\!]$.

Proposition 26 *For all* $\epsilon > 0$*,* $[\![\mathcal{H}]\!] \preceq_\epsilon [\![expand_\epsilon(\mathcal{H})]\!]$*.*

However, we can also show that the polynomial approximation of $\mathcal{H}$ can be approximately simulated by a "small" expansion of $\mathcal{H}$. This is formalized in the following theorem:

Theorem 27 *For all* $\epsilon > 0$*,*

$$[\![poly_\epsilon(\mathcal{H})]\!] \preceq_\epsilon [\![expand_{2\epsilon}(\mathcal{H})]\!].$$

Proof Let $\mathcal{H} = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv, flow, guard, reset, flab)$ be a hybrid system. Let $poly_\epsilon(\mathcal{H}) = (Loc, Act_H, Lab_H, Edges, Cont', loc_0, Cont_0', inv', flow', guard', reset', flab)$ and $expand_{2\epsilon}(\mathcal{H}) = (Loc, Act_H, Lab_H, Edges, Cont, loc_0, Cont_0, inv'', flow, guard'', reset'', flab)$ be the ϵ-polynomial approximation and 2ϵ-expansion of $\mathcal{H}$, respectively.

We consider a relation R' which is the inverse of R defined in the proof of Theorem 20, that is, $R' = R^{-1}$. Let $(l, (x_0, t_1, x_1))R'(l, x)$. Again from the definition of R', we have $\|x - x_1\| < \epsilon$, and $d(\langle\!\langle (l, (x_0, t_1, x_1)) \rangle\!\rangle, \langle\!\langle (l, x) \rangle\!\rangle) < \epsilon$.

Suppose that $(l, (x_0, t_1, x_1)) \xrightarrow{t_2} (l, (x_0, t_1 + t_2, x_2))$ is a continuous transition. Note that this implies that $t_1 + t_2 \leq t_b$. Then, for all $0 \leq t \leq t_1 + t_2$, $flow'(l, (x_0, 0, x_0))(t) \in inv'(l) = inv(l) \times \{t \mid t \in \mathbb{R}_{\geq 0}, t \leq t_b\} \times expand_\epsilon(inv(l))$, and $flow'(l, (x_0, 0, x_0))(t_1) = (x_0, t_1, x_1)$ and $flow'(l, (x_0, 0, x_0))(t_1 + t_2) = (x_0, t_1 + t_2, x_2)$. In particular, this implies that for all $0 \leq t \leq t_1 + t_2$,

$\mathit{flow}(l,x)(t) \in \mathit{expand}_{2\epsilon}(\mathit{inv}(l))$ (follows from the construction of flow'). Note that $x = \mathit{flow}(l,x_0)(t_1)$ and let $x_2' = \mathit{flow}(l,x_0)(t_1+t_2)$. Then $(l,x) \xrightarrow{t_2} (l,x_2')$ in $\mathit{expand}_{2\epsilon}(\mathcal{H})$ and we can deduce from the definitions of R' and x_2' that $(l,(x_0,t_1+t_2,x_2))R'(l,x_2')$.

Suppose that there is a discrete transition from $(l,(x_0,t_1,x_1))$ to $(l',(x_2,0,x_2))$ along edge $e = (l,a,l')$ and let $(l,(x_0,t_1,x_1))R'(l,x)$. Then there exists an x_1' such that $\|x_1 - x_1'\| < \epsilon$ and $(x_1',x_2) \in \mathit{reset}(e)$. We claim that $(l,x) \xrightarrow{a} (l',x_2)$ holds in $\mathit{expand}_{2\epsilon}(\mathcal{H})$. Since $(x_0,t_1,x_1) \in \mathit{guard}'(e)$, we have that $x_1 \in \mathit{expand}_{\epsilon}(\mathit{guard}(e))$ and since $\|x_1 - x_1'\| < \epsilon$, we have that $x_1' \in \mathit{expand}_{2\epsilon}(\mathit{guard}(e))$ or equivalently $x_1' \in \mathit{guard}''(e)$. Since $\|x_1 - x\| < \epsilon$ (from the definition of R'), we have that $\|x - x_1'\| < 2\epsilon$. Since $(x_1',x_2) \in \mathit{reset}(e)$, we have that $(x,x_2) \in \mathit{reset}''(e)$. Therefore $(l,x) \xrightarrow{a} (l',x_2)$. Since $(l',(x_2,0,x_2))R'(l',x_2)$ is trivially true, we have the desired result. ∎

The following corollary formalizes the tightness of the polynomial approximation by putting together theorems 20 and 27.

Corollary 28 *For all* $\epsilon > 0$, $[\![\mathcal{H}]\!] \preceq_\epsilon [\![\mathit{poly}_\epsilon(\mathcal{H})]\!] \preceq_\epsilon [\![\mathit{expand}_{2\epsilon}(\mathcal{H})]\!]$.

3.5 Verification of Tolerant Systems

As we saw in the previous section, if the polynomial approximation is approximately safe, then we can conclude that the original system is also safe. This is a property that is typically exhibited by traditional abstractions. However, what we can also show due to the tightness of our approximations is that if the polynomial approximation is not approximately safe, then it is either the case that the original system is unsafe or the system is not "tolerant" with respect to small perturbations. Next, we formalize the notion of tolerance and obtain the above statement by showing that for tolerant systems verifying the approximation is equivalent to verifying the original system.

Definition 29 (ϵ-tolerance) *A hybrid system $\mathcal{H}$ is said to be ϵ-tolerant for some $\epsilon > 0$ with respect to a property $\phi \in SHM$ if and only if $[\![\mathcal{H}]\!] \models \phi \Rightarrow [\![\mathit{expand}_\epsilon(\mathcal{H})]\!] \models \mathit{shrink}_\epsilon(\phi)$.*

Next we show that model-checking a system with respect to a problem is equivalent to model checking its polynomial expansion, if the system is tolerant.

Theorem 30 *Let ϕ be a formula in SHM logic. Let $\mathcal{H}$ be a hybrid system which is 2ϵ-tolerant with respect to ϕ. Then,*

$$[\![\mathcal{H}]\!] \models \phi \Leftrightarrow [\![poly_\epsilon(\mathcal{H})]\!] \models shrink_\epsilon(\phi).$$

Proof ($\Rightarrow$) Suppose $[\![\mathcal{H}]\!] \models \phi$. Since $\mathcal{H}$ is 2ϵ-tolerant, we can infer from the definition of 2ϵ-tolerance that $[\![expand_{2\epsilon}(\mathcal{H})]\!] \models shrink_{2\epsilon}(\phi)$ which is equivalent to $[\![expand_{2\epsilon}(\mathcal{H})]\!] \models shrink_\epsilon(shrink_\epsilon(\phi))$. Next, since $[\![poly_\epsilon(\mathcal{H})]\!] \preceq_\epsilon [\![expand_{2\epsilon}(\mathcal{H})]\!]$ (from Theorem 27), we obtain using the logical characterization of $\preceq_\epsilon$ in Theorem 13 and the fact $[\![expand_{2\epsilon}(\mathcal{H})]\!] \models shrink_\epsilon(shrink_\epsilon(\phi))$ that $[\![poly_\epsilon(\mathcal{H})]\!] \models shrink_\epsilon(\phi)$.
($\Leftarrow$) Suppose $[\![poly_\epsilon(\mathcal{H})]\!] \models shrink_\epsilon(\phi)$. Since $[\![\mathcal{H}]\!] \preceq_\epsilon [\![poly_\epsilon(\mathcal{H})]\!]$ (Proposition 20), we obtain from the logical characterization of ϵ-simulation in Theorem 13 that $[\![\mathcal{H}]\!] \models \phi$. ∎

Remark 31 *In practice, we would not know whether a system $\mathcal{H}$ is 2ϵ-tolerant with respect to ϕ. However, if ϵ is small and $[\![poly_\epsilon(\mathcal{H})]\!] \not\models shrink_\epsilon(\phi)$ then either $[\![\mathcal{H}]\!] \not\models \phi$ or $\mathcal{H}$ is not 2ϵ-tolerant with respect to ϕ. Either way it suggests that the design of $\mathcal{H}$ needs to be modified.*

3.6 Stone Weierstrass Theorem in Practice

Stone Weierstrass Theorem is not constructive in that it does not given an algorithm to compute an approximation of a function. In general, constructing polynomial approximations cannot be automated, since the problem is undecidable [85]. However there are various approximation techniques available in the literature which are efficient for certain classes of functions.

One popular approximation technique is Taylor approximation. In this, a smooth function is approximated by taking the first few terms of its Taylor series expansion. This method requires computing the values of the derivatives of the function at certain points. Instead, one can also use Bernstein polynomials to approximate continuous functions, provided one can compute the values of the function at certain points [96, 70].

These approximate the function by sampling it at various points, and do not require the function to be smooth. We will discuss Bernstein polynomials

in more detail in the next chapter. Another method of approximation is the Remez algorithm [94], which is an iterative minimax method. However such iterative methods are often expensive in terms of the computation time. Hence there is a trade-off between the computation time, accuracy and the size of the approximation.

Further, when the function is not available in closed form, but is given as the solution of a differential equation, there are methods known as *collocation method* to obtain polynomial approximations. At an abstract level, a form of the polynomial is selected and the differential equation is evaluated at various points to determine the coefficients of this polynomial. A method which is based on the above is the Picard operation [83]. Another efficient method, which improves upon the Picard operation, is the Parker-Sochacki method [81]. The Parker-Sochacki method can be carried out entirely symbolically and hence one can use a software package like *Maple* which supports manipulations of algebraic expressions. The LdeApprox package in *Mathematica* uses methods from [38] to find polynomial approximations to both symbolic and numerical forms of linear differential equations with or without boundary value constraints given a range for the input variable. The output of the algorithm is a closed form polynomial expression on the input and the symbols used as parameters.

3.7 An Application: Air Traffic Coordination Protocol

In this section we apply the approximation techniques introduced in this chapter to verify an air traffic coordination protocol. In [80], an optimal controller was synthesized for a similar collision avoidance protocol. The example was also considered in [86] and [84]. However the analyses used linear approximations without explicitly quantifying the error of approximation.

3.7.1 Problem Description

The system in Figure 3.1 describes a situation where two aircraft 1 and 2 which are flying in directions perpendicular to each other want to merge on to the x-axis. Aircraft 1 is initially at distance d_1 from the origin, i.e., at coordinates $(-d_1, 0)$ and aircraft 2 is at coordinates $(-d_2 - r, -r)$. Aircraft 1

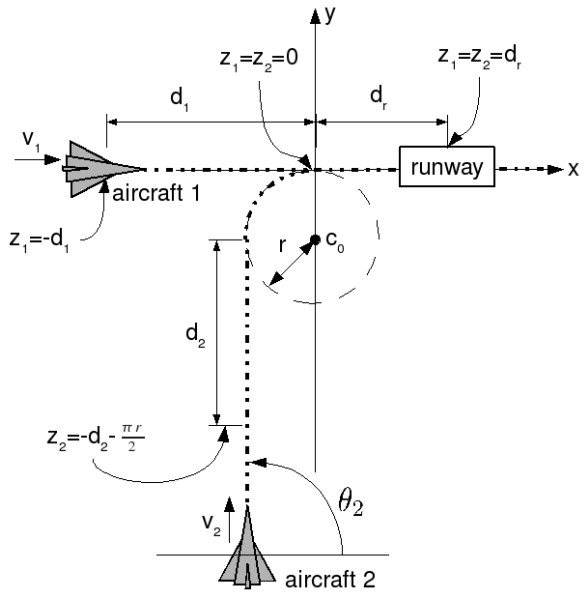

Figure 3.1: The smooth landing paths adopted from [80].

is moving along the positive x axis with velocity v_1 and aircraft 2 is moving along the positive y axis with velocity v_2. Aircraft 1 travels a total distance of $d_1 + d_r$. At some point within distance d_2 from its initial position aircraft 2 may choose to accelerate or decelerate at a constant rate a. Then its velocity changes till it reaches the point X which is at distance d_2 from the initial point. Let its velocity at point X by v. After this, the aircraft follows a circular trajectory with velocity v along the boundary of a circle with center $c_0 = (0, -r)$ and radius r till it reaches the origin. Then it continues to travel along the positive x-axis with velocity v.

We want to ensure that the two aircraft merge safely. We require that at any point of time when aircraft 1 has not reached its destination, the distance between the two aircraft is at least d_{safe}. We will solve the following problem: given a value of acceleration a, does there exist a time t to start the acceleration (or deceleration) so that the two aircraft merge safely?

As will be seen later, a formal model of this system will contain functions which are not polynomials. Our first step would be to construct an approximate system which would be an ϵ-approximation of the original system. In fact we calculate the value of ϵ for the approximated system. This quantification of the error is an interesting feature of our analysis. The abstractions

of the problem considered earlier did not explicitly quantify the error. For example, in [80] the authors consider a linear model of the above system, but do not provide any upper bounds on the error. After constructing the approximate system, we verify the safety property for this system. We know from our results that if the approximated system is safe, then the original system is safe.

3.7.2 Formal Model

The formal model of the system has four states, namely, *init*, *accel*, *turn* and *final*, corresponding to the different phases of aircraft 2. We have three variables z_1, z_2 and v. z_1 has the distance of aircraft 1 from the origin. z_2 has the distance of aircraft 2 to the origin along its trajectory. v_2 is the velocity of aircraft 2. For example, the initial value of z_1 is $-d_1$ and that of z_2 is $-d_2 - \frac{\pi r}{2}$. So $Loc = \{init, accel, turn, final\}$. $X_0 = \{-d_1, -d_2 - \frac{\pi r}{2}\}$.

- $inv(init) = z_1 \leq d_r \wedge z_2 \leq -d_2$.
- $inv(accel) = z_1 \leq d_r \wedge z_2 \leq -d_2$.
- $inv(turn) = z_1 \leq d_r \wedge z_r \leq 0$.
- $inv(final) = z_1 \leq d_r$.

The set of edges $Edges = \{(init, accel), (accel, turn), (turn, final)\}$.

- $guard(init, accel)$ is $z_2 \leq -\frac{\pi r}{2}$.
- $guard(accel, turn)$ is $z_2 = -\frac{\pi r}{2}$.
- $guard(turn, final)$ is $z_2 = 0$.

There are no resets in the system.

- $flow(init, (z_1, z_2, v_2))(t) = (z_1 + v_1 t, z_2 + v_2 t, v_2)$.
- $flow(accel, (z_1, z_2, v_2))(t) = (z_1 + v_1 t, z_2 + v_2 t + \frac{1}{2} a t^2, v_2 + at)$.
- $flow(turn, (z_1, z_2, v_2))(t) = (z_1 + v_1 t, z_2 + v_2 t, v_2)$.
- $flow(final, (z_1, z_2, v_2))(t) = (z_1 + v_1 t, z_2 + v_2 t, v_2)$.

We want to analyse the system for safety. In particular we want to verify that the distance between the two aircraft is always greater than d_{safe}. Hence let us define $dist(z_1, z_2)$, the distance between z_1 and z_2 as follows.

$$dist(z_1, z_2) = \begin{cases} \sqrt{(z_1 + r)^2 + (z_2 + \frac{\pi r}{2} - r)^2} \\ \text{when } z_2 \leq -\frac{\pi r}{2} \\ \sqrt{(z_1 + r\sin\theta_2)^2 + r^2(1 - \cos\theta_2)^2)} \\ \text{when } -\frac{\pi r}{2} < z_2 \leq 0 \\ \sqrt{(z_1 - z_2)^2} \\ \text{when } z_2 > 0 \end{cases}$$

$dist(z_1, z_2)$ can be thought of as just another variable which evolves as described above. The expression for $dist(z_1, z_2)$ is clearly not polynomial. In the next section we will approximate it by a polynomial.

We want to find the time t_s when we should switch such that the distance between the two aircraft is always at least d_{safe}. So we consider the time t_s as a parameter of the system. We expand the continuous statespace of the system by a component τ which evolves with time as $\tau(t) = \tau(0) + t$. We then allow the first discrete transition to happen *only* when $\tau = t_s$. This then restricts $guard(init, accel)$ to $guard(init, accel) \stackrel{redef}{=} \left(z_2 < -\frac{\pi r}{2} \wedge \tau = t_s\right)$.

The problem is then solved in the following way. We define an *SHM* formula $SAFETY(t_s)$ which says that the two aircraft are at safe distance if we start the acceleration at time t_s. We will define the formula $SAFETY(t_s)$ on the transition system which is similar to transition system of the above hybrid system except that it has only two action labels, namely, tt and dt, and every time transition, i.e, those labelled by $a \in \mathbb{R}_{\geq 0}$ is now labelled by tt and every discrete transition is now labelled by dt. The formula is then defined as:

$$\begin{aligned} SAFETY(t_s) := \quad & DIST \wedge [tt](DIST \wedge [dt] \\ & (DIST \wedge [tt](DIST \wedge [dt] \\ & (DIST \wedge [tt](DIST \wedge [dt] \\ & (DIST \wedge [tt]DIST)))))) \end{aligned}$$

where $DIST$ is an atomic proposition defining $dist(z_1, z_2) > d_{safe}$. We then need to verify if $\exists t_s SAFETY(t_s)$ is true.

As mentioned before, the above formula can be written as a first order formula with $DIST$ as an atomic formula. Let us call this formula $FO(SAFETY)$. Since $DIST$ is not an algebraic formula, $FO(SAFETY)$ is

not a formula over the structure of reals. $FO(SAFETY)$ can be written as:

$$FO(SAFETY)(t_s) := \bigwedge_{0 \leq i \leq 3} (reach_i(z_1, z_2) \Rightarrow dist(z_1, z_2) > d_s).$$

where $reach_i(z_1, z_2)$ is an expression which says that the value (z_1, z_2) is reachable by taking i discrete transitions. Since $dist(z_1, z_2)$ is not a formula over the reals, we cannot use the decidability of $Th(\mathbb{R})$ to verify $\exists t_s SAFETY(t_s)$. Hence we approximate *dist* and obtain an ϵ-approximation of the system. We then need to verify if $shrink_\epsilon(SAFETY)$ is true in the approximated system. Equivalently, we will need to check if $FO(shrink_\epsilon(SAFETY))$ holds, which can be done since this formula is algebraic. If this formula is true, then from Theorem 20 and Theorem 13, we have that the original system satisfies *SAFETY*. In the next section, we discuss details about the construction of the approximation.

3.7.3 Polynomial Approximations

In this section, we describe the approximation of the problem, quantification of the errors and some results we obtained.

As explained before, the general technique to approximate the system would involve approximating the flows, and expanding the guards, resets and the invariants. In the construction we need to expand the constraints to compensate for the error introduced due to approximation of the flows. We observe that for the problem at hand the flows are already algebraic except for that of the variable $dist(z_1, z_2)$. However this does not occur in any guards, resets or invariants. Hence it is easy to see that if we just approximate $dist(z_1, z_2)$ with a approximation error ϵ and do not change the guards, resets, invariants and the other flows, then the approximated system ϵ simulates the original system. Also in this case, $FO(shrink_\epsilon(SAFETY))$ will just be $FO(SAFETY)$ with *DIST* replaced by $DIST_\epsilon := Poly_\epsilon(dist(z_1, z_2)) > d_{safe} + \epsilon$, where $Poly_\epsilon(dist(z_1, z_2))$ is a polynomial approximation of $dist(z_1, z_2)$ with an ϵ upper bound on error in the range of interest.

Now let us turn to the approximation of $dist(z_1, z_2)$. The expression that needs to be approximated is $dist(z_1, z_2) = z_1^2 - 2z_1 r \sin \frac{z_2}{r} + r^2 - 2r^2 \cos \frac{z_2}{r}$ in the range $-\frac{\pi r}{2} < z_2 \leq 0$. In particular, we will need to approximate the functions

coshalfpi(y) := $\cos(\frac{\pi}{2}y)$ and sinhalfpi(y) := $\sin(\frac{\pi}{2}y)$ in the range $0 \leq y \leq 1$. We approximate these functions using Taylor expansions. For a 5-th order Taylor approximation around zero, we obtain an upper bound of 0.025 as the approximation error in this range. We find the error by plotting the error in *Mathematica* and visually identifying the maximum absolute error in the range of interest. This kind of analysis suffices when the function under consideration is smooth as in our case. We explain later a general grid based method to do the same. Then the approximation error for the whole function *dist* will be less than $\epsilon := |0.05 z_1 r| + |0.05 r^2|$. Given the values of z_1 and r, this gives us the value of ϵ. When $z_1 \leq r$, the approximation error will be $\epsilon := \frac{r^2}{10}$.

Now we turn to the issue of computation of approximation error. We note that computing the error of approximation is crucial to our analysis. This is because we are required to verify an approximate formula which depends on the approximation error of the approximated system. Unfortunately, it is rarely possible to exactly calculate the maximum approximation error throughout the approximation region. On the other hand, one can find upper bounds on the error which suffices for our analysis. There are analytic and grid based methods for this. Most of the methods are based on finding Lipschitz bounds for the function to be approximated. Here we explain a grid based method to compute a Lipschitz bound.

In order to find a bound for the maximum error, we divide the domain of the error function into a multidimensional grid of pitch δ. For each grid we find the accuracy which is proportional to the pitch value δ and the maximum gradient in each cell. Also we sample one point in every grid. The maximum error is then bounded from above by the sum of accuracy and the maximum sample value. We explain it through an example.

Let us consider the function sinhalfpi$(y) = \sin(\frac{\pi}{2}y)$. In the range $0 \leq y \leq 1$, we can easily see that $\frac{\pi}{2}$ is an upper bound for the derivative, i.e, $\sup_{0 \leq y \leq 1} \left| \frac{\partial \text{sinhalfpi}(y)}{\partial y} \right| \leq \frac{\pi}{2}$. The polynomial approximations maximum gradient within the range $[0, \frac{\pi}{2}]$ is 1.0. This can be verified by differentiating the approximating polynomials and finding the maximum of absolute value of the resulting polynomial derivatives. For example, for the 5th degree Taylor expansion of sinhalfpi the maximum absolute gradient (derivative) or Lipschitz constant within the range will be bounded by y given by quantifier elimina-

tion of the following: $\forall x\ (0 \leq x \leq 1) \Rightarrow y^2 < \left(\frac{\partial}{\partial x}(\frac{\pi x}{2} - \frac{\pi^3 x^3}{48} + \frac{\pi^5 x^5}{3840})\right)^2$. So the maximum error can be determined with accuracy 0.01 by sampling it on a grid of pitch equal to $0.01(\frac{\pi}{2}+1.0)$. To find a bound for the error we simply add the accuracy to the maximum error sample. After obtaining the error of the trigonometric functions we recursively find the maximum error for the whole approximation.

3.7.4 Verification Results

We now describe some results we obtained for the verification problem: Does there exists a time t_s to start the acceleration, such that the two aircraft maintain a safe distance d_{safe}.

We used the following constants for verification. $v_1 = 100$, $d_1 = d_2 = d_r = r = 1000$, $v_{2i} = 100$, $z_{1i} = -d_1$ and $z_{2i} = -d_2 - \pi r/2$, where z_{1i}, z_{2i} and v_{2i} are the initial values of z_1, z_2 and v_2 respectively. Resulting approximation errors for 3rd and 5th degree polynomial approximations were $\epsilon_3 = r^2$ and $\epsilon_5 = r^2/10$, respectively.

When we set the acceleration $a = 10$, and used the 3rd degree polynomial approximation, we obtained that the system is unsafe. Next we increased the degree of approximation to 5. In this case, the quantifier elimination in *Mathematica* lasted quite a few minutes and returned false again. For this value of a, we could not conclude if the system was safe. Then we tried $a = 40$. Again we did not succeed with a degree 3 polynomial. However when a degree 5 polynomial was used, the quantifier elimination returned the constraint $0 \leq t_s \leq 7.887784$ within a few minutes. Hence in this case we can conclude that the values of t_s returned is a conservative bound on the value of the time to start accelerating so that the aircraft maintain a safe distance.

In this section, we have illustrated how we can use our theoretical results of the earlier sections to verify a safety property. Our results in the earlier sections are quite general and do not specify the method to use for the approximation. In this section we saw that there are various methods for approximations and error computations, and one method may be better than the other depending on the system we are analysing. Once the approximation is obtained, we need to verify the approximated formula. Software tools

for quantifier elimination might not be able to handle large formulas, and hence in practice we might require some manual preprocessing and careful formulation of the problem as in our case.

3.8 Conclusions

In this chapter, we presented a technique for approximating a hybrid system with arbitrary flow function by a hybrid system with polynomial flow function. We showed that the approximate system ϵ-simulates the original system. This allowed us to conclude that we can prove the safety of a system by proving the safety of the approximate system. This is a property similar to that preserved by traditional abstractions. Interestingly, due to the tightness of our approximations, we were also able to show that if the approximate system is not safe, then it is either the case that the original system is not safe or it is not tolerant with respect to the safety property. This is a useful property, since if the approximate system is not safe, it suggest that the design is either faulty or not robust, and hence needs to be changed.

We applied our approximation technique to analyse an air-traffic coordination protocol. Experience suggests that verification by approximation to polynomial hybrid systems is feasible. This is a general technique which can be applied to a large class of systems. However, it has two shortcomings. Firstly, the construction of the polynomial approximation, in particular, the polynomial approximation of the flows is not automatic in general. Secondly, the degree of the polynomial grows quickly with the precision ϵ of the approximation. In the next chapter, we investigate the above issues, and present a technique for automating the construction of the polynomial approximation for a subclass of hybrid systems.

CHAPTER 4

PIECEWISE POLYNOMIAL APPROXIMATIONS

In this chapter, we present a method for automating the construction of the polynomial approximations of hybrid systems with continuous dynamics specified by Linear Dynamical Systems (LDS). Note that the main difficulty in automating the process of construction of the polynomial approximations, in the previous chapter, was in automating the construction of polynomial approximations for the flow function; since the transformation of other components was straight forward. Therefore, in this chapter, we focus on the approximation of the continuous dynamics.

Apart from the lack of automation, one of the other drawbacks of the approximation technique in the previous chapter was the increase in the degree of the polynomial approximating the flow function with the increase in the precision. To overcome this, we propose a method for constructing approximations which constrain the degree of the polynomials, by moving to a piecewise polynomial setting. More precisely, we consider the problem of approximating the solution of a LDS in the interval $[0, T]$ by a piecewise polynomial approximation of a given degree to within a given error bound ϵ. Fixing a degree d, our algorithm divides the interval $[0, T]$ into sub-intervals of not necessarily equal size, such that a polynomial of degree d approximating the function in that interval approximates the actual flow to within an error bound of ϵ. Our experimental evaluation of the algorithm when the degree d is fixed to be either 1 or 2, shows that the approach is promising, as it scales to large dimensional dynamical systems.

Piecewise linear approximations of the flow function of LDS (solutions of the LDS) have been extensively studied in the context of "Post Computation". Experimental comparisons of our algorithm with the existing algorithms for piecewise linear approximations suggest that our algorithm has huge advantages both in terms of the size of the representation of the approximation and the time for constructing the same, which in turn affect the

cost of verification. Next we present an overview of our algorithm in the context of post computation.

4.1 An Overview

Integral to the automatic verification of safety properties is the computation of the set of reachable states of a system. In the context of hybrid systems, the key challenge in reachability computation is to compute, for a given set of states X, all states that are reachable from X under the continuous dynamics, within (some) time T. While states reachable within a bounded time can be computed for some hybrid systems with simple dynamics [4, 1, 54, 65, 103], it must typically be approximated, since the problem of computing the reachable set is undecidable for most dynamical systems.

There are three principal techniques for computing an approximation to the reachable set of states. The first constructs an abstract transition system that *simulates* (in a formal sense) the dynamical system, and carries out the reachability computation on the abstract system [23, 3, 26, 27, 39, 60, 97]. In this method, the quality of the approximate solution cannot be measured, and so, often this is compensated by repeatedly refining the abstract transition system. The second approach, called *hybridization* [92, 7, 34], partitions the continuous state space, and approximates the continuous dynamics in each partition. Here one can explicitly bound the error between the reachable set of the hybrid system with simpler dynamics and the original dynamical system.

The third method is to directly compute the states reachable within time T. This has been carried out primarily for linear dynamical systems and a convex polytope as initial set X. For such systems, the algorithm proceeds as follows. First, the time interval $[0, T]$ is partitioned into equal intervals of size Δ. Then, the points reached at time Δ from the vertices of X are computed. The set of states reachable *within* time Δ is then approximated by the convex hull of the vertices of the set X and the points reached at time Δ from the vertices of X. Given Δ, T, and the dynamics, the error (or Hausdorff distance) between this convex hull and the actual set of states reachable within time Δ can be bounded. Based on this error bound, this convex hull is first "bloated" to contain all the reachable states and then approximated by a

data structure of choice. Different data structures that have been considered and found useful include griddy polytopes [32], ellipsoids [64], level sets [73], polytopes [24], zonotopes [45, 46], and support functions [51]. After this, the computation for the time interval $[0, \Delta]$ is *translated* to obtain the reachable states for the time interval $[i\Delta, (i+1)\Delta]$ — the states reached at time $i\Delta$ and $(i+1)\Delta$ are obtained by translating the vertices of X and those at time Δ, respectively, and then the "bloated" convex hull of all of these points is approximated by the data structure of choice. This method has been found to be scalable and successful, making the automated analysis of linear dynamical systems possible.

In this chapter, we take a slightly different stance on the problem of directly approximating the reachable set. Instead of trying to bound the error of a reachability computation, we view the problem as one where given an error bound ϵ, one has to compute an over-approximation of the reachable set whose Hausdorff distance from the actual set of reachable states is bounded by ϵ. This subtle change in perspective, immediately suggests some natural changes to the basic algorithm outlined in the previous paragraph. First, the discretization of the time interval $[0, T]$ need not be in terms of equal-sized intervals. We could change interval sizes, as long as the error of approximating the reachable states within that interval can be bounded by ϵ. Second, in the "basic algorithm", the convex hull of the points of the initial set and those at time Δ is taken to be the approximation of the set of states reachable within time Δ. Instead, we view the approximation process as first approximating the *flow* in the interval $[0, \Delta]$ by a polynomial, and then taking the "polynomial tube" defined by this dynamics to be the approximation of the reachable states — when the polynomial is taken to be a linear function, then it corresponds to taking the convex hull of the points, as done in the basic algorithm. Combining these ideas, the algorithm we plan to study in this chapter can be summarized as follows. Given an error bound ϵ and the degree d of the polynomials to be considered, we first find a time (hopefully, as large as possible) t such that dynamics in $[0, t]$ when approximated by degree d polynomials is within error bound ϵ of actual dynamics. The degree d polynomial approximating the actual dynamics can be found by constructing the appropriate Bernstein polynomial [70]. We approximate the set of reachable states in the interval $[0, t]$ by the degree d polynomial tube, and then repeat the process for the time interval $[t, T]$, each time *dynamically* fig-

uring out the appropriate discretization. Recently, a similar approach based on dividing the interval non-uniformly has been proposed for bounded error post computation of linear dynamical systems with inputs [42].

Our algorithm, when compared with the basic algorithm previously studied, has both perceptible advantages and disadvantages. On first glance, the basic algorithm seems to be computationally simpler and potentially faster. For a system with linear dynamics, computing the set of states reached at time Δ involves computing (or rather approximating) matrix exponentials. This is a significant computational overhead. In the basic algorithm, this cost is minimized, as it is performed once for the first $[0, \Delta]$ interval, and for subsequent intervals it is obtained by translation rather than direct computation. In our algorithm, this must be computed afresh for each sub-interval. Moreover, in our algorithm, the intervals need to be determined dynamically, which is an additional overhead to the computation in each sub-interval. However, on the flip side, dynamically determining intervals is likely to give us larger sub-intervals in some places and therefore result in fewer intervals overall to consider. This could be a potentially significant advantage. This is because the eventual approximation of the reach set for the interval $[0, T]$ is given as the union of basic sets (represented by the chosen data structure) that are computed for each sub-interval; thus, the number of terms in the union is as large as the number of sub-intervals. Subsequent steps in verifying safety properties (or other properties of interest) involve taking intersections, and checking emptiness and membership of states in these sets. The complexity of these set-theoretic operations depends on how many terms there are in the union — this is true no matter what the chosen data structure is. Thus, the time (and memory) used in verification is directly influenced by how many steps the time interval $[0, T]$ is divided into. Note, that the "quality" of the solution computed by the basic algorithm is not better than the one computed by our algorithm, even if it uses smaller sub-intervals and more of them, because the overall quality is determined by the quality of the solution in the "worst" interval, which is then built into the bloating factor used by both algorithms.

In order to evaluate these competing claims, we implemented both the basic algorithm and our algorithm in MATLAB. Observe that in both the basic algorithm and our algorithm, states reached at certain times must be computed, and then the convex hull or polynomial tube must be approximated

by the data structure of choice. In our experimental evaluation, we choose to be agnostic about the relative merits of different data structures, and we make no claims about which data structure should be chosen. Therefore, we only compute the states reached at certain time steps, and not the data structure representing the reachable states. Once a data structure is chosen, the computational overhead in constructing the desired set will be the same whether the basic algorithm or our method is used (provided linear flows are used to approximate the actual flow in our method). Thus, our experimental setup is to evaluate under what conditions (types of matrices and time bound T) does unequal intervals plus associated computation costs beat uniform intervals with computation minimized by translation. We also try to understand, when it makes sense to use polynomials that are not linear to approximate the flow. Our results apply no matter what your favorite data structure is.

Our experimental results are surprising. We evaluated the two methods on both "natural" examples that have been studied before, and randomly generated matrices, and for different time intervals and error bounds. First we observed that our algorithm is scalable as it computes the points for both large matrices (we tried it on 100×100 matrices) and for many time steps (requiring thousands of iterations). Second, surprisingly, our algorithm, approximating flow by linear functions, *most of the time* outperforms the basic algorithm, sometimes by a few orders of magnitude. The gap in the performance between the two algorithms only widens considerably as the time bound T is increased. This can be explained by the fact that as the number of iterations increases the significant reduction in the number of intervals dominates the computation costs. Our algorithm not only uses significantly fewer number of intervals for the same precision (as would be expected) but the size of the *minimum* interval is also significantly larger than the size of the uniform interval chosen by the basic algorithm. This suggests that our algorithm reaps the benefits of dynamic computation of error bounds, over static determination of them. Next, we compare the potential benefits of using non-linear flows to approximate the actual flow. We consider polynomials of degree 2, as they are appropriate when considering ellipsoids as the data structure. Theoretically, the size of a dynamically determined subinterval could be a factor of 2 larger when using polynomials of degree 2 when compared with linear functions. That, in turn, could translate to significant

(exponential) savings in terms of the number of intervals. However, these theoretical possibilities were not observed in our experiments — the number of intervals for degree 2 polynomials were at most a factor of 2 smaller. This could be explained by our observation that most of the sub-intervals in the linear approximation tend to be small, and they are roughly of the same size. In such a scenario the theoretical benefits of using quadratic approximations don't translate to visible gains.

We would like to remark that the approximations that we compute depend on the machine precision, since the approximations are constructed by sampling the function at certain points, and are only as precise as that of the function values computed for the sample points. There are other approaches for reachability analysis [82, 76, 65, 103] which rely on the decidability of the satisfiability problem for the first order theory of reals, which in contrast are algebraic techniques with infinite precision computation. However, for reachability analysis for the class of linear dynamical system, the above techniques need to first compute a polynomial approximation of the dynamics.

The rest of the chapter is organized as follows. Next, in Section 4.2, we outline our algorithm to compute post for general dynamical systems by approximating flows using Bernstein polynomials. In Section 4.3, we describe the specific algorithm for linear dynamical systems. We then give details of our experimental results (Section 4.4) before presenting our conclusions.

4.2 Post Computation by Flow Approximation

In this section, we describe a general algorithm to approximate the flow of a dynamical system by a piecewise polynomial function of any fixed degree within any approximation error bound. The approximations are based on Bernstein Polynomials. We will begin with some preliminaries.

4.2.1 Preliminaries

We will use ∞-norms for measuring the distance between two vectors. Given $x, y \in \mathbb{R}^n$, let $\|x - y\|$ denote the distance between x and y in the ∞-norm, that is, $\|x-y\| = max_{1\leq i\leq n}|(x)_i-(y)_i|$. Also, given two functions $G : A \to \mathbb{R}^n$ and $H : A \to \mathbb{R}^n$, the distance between the functions, denoted $\|G - H\|$, is

given by $\|G - H\| = \sup_{x \in A} \|G(x) - H(x)\|$. Given two sets $A, B \subseteq \mathbb{R}^n$, the Hausdorff distance between the two sets, denoted $d_H(A, B)$, is defined as

$$d_H(A, B) = max(\sup_{x \in A} \inf_{y \in B} \|x - y\|, \sup_{x \in B} \inf_{y \in A} \|x - y\|)$$

4.2.2 Bernstein Polynomial Approximations

Stone-Weierstrass Approximation theorem implies that any continuous function over a compact space can be approximated arbitrarily closely by a polynomial function. However, it does not provide an explicit method for constructing these polynomials. Bernstein polynomials are a class of polynomials which can be used to approximate a continuous function. They provide a constructive proof of Stone-Weierstrass theorem in that given a function F, and an $\epsilon > 0$, one can construct a Bernstein polynomial such that the distance between the function F and the corresponding polynomial is with in ϵ, provided one can compute the values of the function at certain points and estimate the values of certain parameters of the function. The approximate polynomial is constructed by evaluating the function F at certain finite number of points and using these values as coefficients of the polynomial. Let $F : [a, b] \rightarrow \mathbb{R}$ be a function. Then a Bernstein polynomial of degree n approximating F, denoted by $Bern_n(F)$ is given by:

$$(Bern_n(F))(x) =$$

$$\sum_{k=0}^{n} F(a + k(b - a)/n) * \binom{n}{k} *$$

$$((x - a)/(b - a))^k (1 - (x - a)/(b - a))^{n-k},$$

for all $a \leq x \leq b$.

The next two lemmas essentially show that the approximation error introduced by the polynomial approximation can be made arbitrarily small. In particular, given an $\epsilon > 0$, we can choose an n effectively such that the distance between the two functions is bounded by ϵ. Let us denote by F_{diff}, the absolute difference between the maximum and minimum values of F in its domain, i.e, $F_{diff} = \max_{x_1, x_2 \in [a,b]} |F(x_2) - F(x_1)|$. Then, we have the following from [70]:

Lemma 2 *Let $F : [a, b] \to \mathbb{R}$ be a continuous function and $\epsilon > 0$. Let $\delta > 0$ be such that for all $x_1, x_2 \in [a, b]$, $|x_2 - x_1| \leq \delta$ implies $|F(x_2) - F(x_1)| \leq \epsilon$. Then $|F(x) - Bern_n(F)(x)| \leq \epsilon$ if $n > F_{diff}/(\epsilon\delta^2)$.*

Lemma 3 *Let $F : [a, b] \to \mathbb{R}$ be a continuous function satisfying the Lipschitz condition $|F(x) - F(y)| < L|x - y|$ for $x, y \in [a, b]$. Then $|F(x) - Bern_n(F)(x)| < L/(2\sqrt(n))$.*

Note that both the lemmas give an n such that the error or distance between F and $Bern_n(F)$ is within ϵ. In particular, given an ϵ the first lemma tells us to choose an $n > F_{diff}/(\epsilon\delta^2)$, and the second lemma tells us that a choice of $n > (L/2\epsilon)^2$ would ensure that the error is within ϵ.

4.2.3 General Algorithm

Our aim is to approximate a flow F over a time interval $[0, T]$ by a polynomial of very low degree, such as a *linear* or a *quadratic* polynomial. Lemmas 2 and 3 give us a polynomial of a certain degree approximating the function over the interval $[0, T]$ ensuring the desired error bound. However, the degree of the polynomial can be large. Hence instead of approximating by a single polynomial of high degree, we present an algorithm which splits the interval $[0, T]$ into smaller intervals, and approximates the flow separately in each of the smaller intervals, thereby giving a piecewise continuous polynomial approximation of a fixed degree.

Consider the following dynamical system.

$$\dot{x} = f(x), x \in \mathbb{R}^n, x(0) \in X_0,$$

where X_0 is a set of initial vectors. We will assume that f is a 'nice' function (for example, Lipschitz continuous) such that it has a unique solution $\Phi : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$, satisfying $d/dt(\Phi(x_0, t)) = f(\Phi(x_0, t))$ for all $x_0 \in X_0$ and $t \in \mathbb{R}_{\geq 0}$. Note that Φ is assumed to be continuous and differentiable.

Let us fix an initial vector $x_0 \in X_0$, and a time $T \in \mathbb{R}_{\geq 0}$. Let $F : [0, T] \to \mathbb{R}^n$ be the function $F(t) = \Phi(x_0, t)$ for all $0 \leq t \leq T$. We will approximate each F_i by a piecewise polynomial function P_i of degree $\leq m$ within an error bound of ϵ. Hence $\|F - P\| < \epsilon$. The general algorithm is outlined below.

Algorithm 1 Varying Time Step Algorithm

Input: $m \in \mathbb{N}, \epsilon \in \mathbb{R}_{\geq 0}, F : [0, T] \to \mathbb{R}$
Output: Sequence of polynomials

$t := 0$
while $t < T$ **do**
  Choose $0 < t_i < T$ s.t.
  $\|Bern_m(F[t, t + t_i]) - F[t, t + t_i]\| \leq \epsilon$
  Output $Bern_m(F[t, t + t_i])$
  $t := t + t_i$
end while

Starting at time $t = 0$, find a time $0 < t_1 \leq T$ such that $\|Bern_n(F_i[0, t_1]) - F_i[0, t_1]\| < \epsilon$. There always exists such a t_1, since continuity of F implies that $F_i[0, t_1]_{diff}$ can be made arbitrarily small by taking t_1 to be sufficiently small, and therefore one can satisfy the condition $F_i[0, t_1]_{diff}/\epsilon\delta^2 < m$ in Lemma 2. This reduces the problem to finding a piecewise polynomial approximation of the function $F_i[t_1, T]$, and we proceed in the same manner to compute $t_2, t_3, \cdots$. Since the function values of the Bernstein polynomial and the function it is approximating match at the end-point, the piecewise polynomial function P, in any interval $[0, \sum_{i=1}^{k} t_i]$ is continuous. To ensure that the number of iterations is finite, we need to ensure that the we make progress. This can be guaranteed by ensuring that in each step, the t_i chosen is at least Δ, for some $\Delta > 0$. Note that there always exists a Δ, which can be chosen at any step which satisfies the condition in Lemma 2. To see this, let $\gamma = m\epsilon\delta^2$, where m is the degree of the polynomials we are considering, ϵ is the desired bound on approximation error, and δ is the parameter in the definition of continuity for F_i corresponding to ϵ. Since F_i is continuous and bounded, there exists a $\Delta > 0$ such that for all $t, t' \in [0, T]$, $|t - t'| \leq \Delta$ implies $|F_i(t) - F_i(t')| \leq \gamma$. Hence choosing Δ at any step ensures that we make progress. In order to materialize the above sketch of the algorithm, we need to be able to compute F_{diff} or some upper bound on it, which ensures progress. In the next section, we present two methods to compute the t_is for the class of linear dynamical systems.

4.3 Approximation of Linear Dynamical Systems

In this section, we consider linear dynamical systems and present our algorithm in detail. Consider the following system:

$$\dot{x} = Ax, x \in \mathbb{R}^n, x(0) \in X_0,$$

where $X_0 \subseteq \mathbb{R}^n$ is a bounded convex polyhedron. The solution of the above equation is given by:

$$\Phi(x_0, t) = e^{At}x_0, x_0 \in X_0, t \in \mathbb{R}_{\geq 0}.$$

Let us define $Post_\Phi(X, [0,T]) = \{\Phi(x,t) \mid x \in X, t \in [0,T]\}$.

We consider the problem of computing an over approximation of $Post_\Phi(X_0, [0,T])$ such that the error in the approximation is within an ϵ. More precisely, we wish to find a set $\widehat{Post_\Phi}(X_0, [0,T])$ such that $\widehat{Post_\Phi}(X_0, [0,T])$ is an over approximation, that is, $Post_\Phi(X_0, [0,T]) \subseteq \widehat{Post}_\Phi(X_0, [0,T])$, and the Hausdorff distance between the two sets is bounded by ϵ, that is, $d_H(Post_\Phi(X_0, [0,T]), \widehat{Post_\Phi}(X_0, [0,T])) \leq \epsilon$.

First we show that the flow function for a linear system preserves convexity and hence it suffices to approximate only the flows starting from the vertices of X_0.

Proposition 32 *Let $x = \alpha_1 x_1 + \cdots + \alpha_k x_k$ where $x_i \in \mathbb{R}^n$ and $\sum_{i=1}^k \alpha_k = 1$. Then $\Phi(x,t) = \alpha_1\Phi(x_1,t) + \cdots + \alpha_k\Phi(x_k,t)$.*

Let *Vertices*(X_0) denote the set of vertices of X_0. Let us fix a time T. Given a $v \in$ *Vertices*(X_0), let $F_v : [0,T] \to \mathbb{R}^n$ be the function $F_v(t) = \Phi(v,t)$ for all $t \in [0,T]$. For each $v \in$ *Vertices*(X_0), let $\hat{F}_v$ denote a function such that $\|\hat{F}_v - F_v\| \leq \epsilon$. Let $\hat{R} = \{\alpha_1\hat{F}_{v_1}(t) + \cdots + \alpha_k\hat{F}_{v_k}(t) \mid \alpha_1 + \cdots + \alpha_k = 1, t \in [0,T]\}$. The next lemma says that the Hausdorff distance between the exact post set and $\hat{R}$ is bounded by ϵ.

Lemma 4

$$d_H(Post_\Phi(X_0, [0,T]), \hat{R}) \leq \epsilon.$$

Proof Given $x \in Post_\Phi(X_0, [0,T])$ we will find an $x' \in \hat{R}$ such that $\|x - x'\| \leq \epsilon$ and vice versa.

Let $x \in Post_\Phi(X_0, [0,T])$.

Then $x = e^{At}x_0$ for some $x_0 \in X_0$ and $t \in T$.

Let $Vertices(X_0) = \{v_1, \cdots, v_k\}$.

Since X_0 is a bounded convex polyhedron,

$x_0 = \alpha_1 v_1 + \cdots + \alpha_k v_k$,

for some $\alpha_1 + \cdots + \alpha_k = 1$.

Then $x = e^{At}x_0 = e^{At}(\alpha_1 v_1 + \cdots + \alpha_k v_k)$

$= (\alpha_1 e^{At} v_1 + \cdots + \alpha_k e^{At} v_k)$.

Let $x' = \alpha_1 \hat{F}_{v_1}(t) + \cdots + \alpha_k \hat{F}_{v_k}(t)$.

Then $|x - x'| =$

$|(\alpha_1 e^{At} v_1 + \cdots + \alpha_k e^{At} v_k) - (\alpha_1 \hat{F}_{v_1}(t) + \cdots + \alpha_k \hat{F}_{v_k}(t))|$

$\leq \alpha_1 |e^{At} v_1 - \hat{F}_{v_1}(t)| + \cdots + \alpha_k |e^{At} v_k - hatF_{v_k}(t)|$

$\leq \alpha_1 \epsilon + \cdots + \alpha_k \epsilon = \epsilon$.

Similarly given an $x \in \hat{R}$, we can find a $x' \in Post_\Phi(X_0, [0,T])$ such that $|x - x'| \leq \epsilon$. ∎

The above proposition tells us that it suffices to approximate the flows starting at the vertices of the polyhedron. More precisely, if we approximate the flows at the vertices within an error bound of ϵ in an interval $[0,T]$, then at any time $t \in [0,T]$, the Hausdorff distance between the actual and approximate sets is with ϵ.

In the literature, various methods have been proposed to compute $\widehat{Post_\Phi}$.

These methods can be seen as consisting of the following two steps.

- Depending on the ϵ, a time step Δ is chosen. Let $V_0 = \mathit{Vertices}(X_0)$ and $V_i = \mathit{Post}_\Phi(V_0, [i\Delta, i\Delta])$ for $i > 0$ be the set of points reached from the vertices of X_0 after i time steps of size Δ. First $V_1 = \mathit{Post}_\Phi(V_0, [\Delta, \Delta])$ is computed. Then the convex hull C_0 of V_0 and V_1 is bloated by ϵ, and the resulting set is enclosed by a data structure of a certain form to obtain an overapproximation C_0' of $\mathit{Post}_\Phi(X_0, [0, \Delta])$.

- Similarly, to obtain an overapproximation of $\mathit{Post}_\Phi(X_0, [i\Delta, (i+1)\Delta])$, the convex hull C_i of V_i and V_{i+1} is bloated by ϵ, and enclosed in a data structure. However, instead of computing V_i directly from V_0, it is computed iteratively from V_{i-1}, that is, V_i is computed from V_{i-1} by a linear transformation using the matrix e^{Δ}.

We think of the above algorithm as first computing an approximation of the flow function, which is the piecewise linear function obtained by joining the corresponding points in V_is, and then enclosing the reach set given by the approximated flow function by a set of a certain form. The above algorithms compute a piecewise linear approximation of the flow function by dividing the interval $[0, T]$ into equal intervals of size Δ. Our main contribution is a novel algorithm for computing an approximation of the flow function, which does not divide the interval uniformly, but dynamically computes the next time step. The obvious advantage is the reduction in the number of times steps, since a time step chosen by the dynamic algorithm is always larger than the constant time step Δ chosen by the uniform time step algorithm. This in turn implies that the size of the final representation of the post set would be smaller, and the size plays a crucial role in further analysis. However, there is a overhead involved with the dynamic algorithm, which is in computing the set of vertices V_is at various time points, since these V_is can no more be computed iteratively by multiplication using a fixed matrix. Since the timesteps Δ keep changing, there does not exist a fixed matrix $e^{A\Delta}$ which can be used to obtain V_i from V_{i-1} for every i. So the new algorithm involves computing a new matrix exponential e^{Δ_i} at each step. However, as we will see in the next section, our experimental evaluations show that the overhead introduced due to computation of a new matrix exponential at each time step becomes negligible due to the huge decrease in the number of steps. In

other words, the cost of doing the large number of matrix multiplications in the constant time step algorithm is greater than the cost of computing new matrix exponentials followed by matrix multiplications for a small number of timesteps in the dynamic algorithm.

To compute the approximation of the flow function for a linear dynamical system, we instantiate Algorithm 1 to obtain an effective algorithm for linear dynamical systems. As mentioned in the discussion of Algorithm 1, we need to present a method to compute the t_is in each step such that progress is ensured. Next we present two methods for computing t_is.

4.3.1 Computing t_i: First Method

In this section we use Lemma 3 to compute the bound t_i. Let us fix an $x_0 \in \mathbb{R}^n$ and an $n \times n$ matrix A. Let $F : [0, T] \to \mathbb{R}^n$ be the function $F(t) = e^{At}x_0$. First we show that F satisfies the Lipschitz condition, and the Lipschitz constant can be bounded by a function of T.

Lemma 5 *Let $F : [0, T] \to \mathbb{R}^n$ be as defined above. Then for each $1 \leq i \leq n$, F_i is Lipschitz continuous with the Lipschitz constant $L = \|A\|e^{\|A\|T}\|x_0\|$.*

Proof First let us recall the following identity. Given $n \times n$ matrices X and Y,

$$\|e^{X+Y} - e^X\| \leq \|Y\|e^{\|X\|}e^{\|Y\|}.$$

W.l.o.g assume $t_2 > t_1$.

$$\begin{aligned}
&\frac{\|e^{At_2}x_0 - e^{At_1}x_0\|}{\|t_2 - t_1\|} \\
&= \frac{\|e^{At_1+A(t_2-t_1)}x_0 - e^{At_1}x_0\|}{|t_2 - t_1|} \\
&\leq \frac{\|A(t_2 - t_1)\|e^{\|At_1\|}e^{\|A(t_2-t_1)\|}\|x_0\|}{|t_2 - t_1|} \\
&= \frac{\|A\||t_2 - t_1|e^{\|A\||t_1|}e^{\|A\||t_2-t_1|}\|x_0\|}{|t_2 - t_1|} \\
&= \|A\|e^{\|A\|e(|t_1|+|t_2-t_1|)}\|x_0\| \\
&= \|A\|e^{\|A\|T}\|x_0\|
\end{aligned}$$

$$L \geq \max_{t_1,t_2} \frac{\|e^{At_2}x_0 - e^{At_1}x_0\|}{\|t_2 - t_1\|}$$
$$\geq \|A\|e^{\|A\|T}\|x_0\|$$

∎

The next lemma gives us a lower bound on the time step t_i that can be chosen at each step such that the approximate polynomial is within distance ϵ from the original function.

Lemma 6 *Let $F : [0,T] \rightarrow \mathbb{R}^n$ be as defined above. For $t_1 = \log_e(2\sqrt{m}\epsilon/\|A\|\|x_0\|)/\|A\|$, $\|F[0,t_1] - B_m(F[0,t_1])\| \leq \epsilon$.*

Proof The Lipschitz constant L for the function $F[0,t_1]$ is given by $L = \|A\|e^{\|A\|t_1}\|x_0\|$ from Lemma 5.

$\|F[0,t_1] - B_m(F[0,t_1])\| < L/(2\sqrt{m})$ from Lemma 3.

$L/(2\sqrt{m}) \leq \epsilon$ implies $\|A\|e^{\|A\|t_1}\|x_0\| \leq 2\sqrt{m}\epsilon$.

Hence for $t_1 \leq \log_e(2\sqrt{m}\epsilon/\|A\|\|x_0\|)/\|A\|$,

$\|F[0,t_1] - B_n(F[0,t_1])\| \leq \epsilon$. ∎

Using the t_i in the definition of Lemma 6 is desirable since it gives a closed form expression for computing the t_i. However, the problem with the above expression is that the expression being computed might not result in a positive number in which case we are in trouble. Next we present another method for computing lower bound for t_i, which always gives a positive answer.

4.3.2 Computing t_i: Second Method

In this section, we use Lemma 2 to compute a bound on the t_is.

Lemma 7 *Let $F : [0,T] \rightarrow \mathbb{R}^n$ be as defined above. Let t_1 be such that $e^{3\|A\|t_1}t_1 < m\epsilon^3/\|A\|^3\|x_0\|^3$. Then we have that $\|B_m(F[0,t_1]) - F[0,t_1]\| \leq \epsilon$.*

Proof From Lemma 2, we have that $\|B_n(F[0,t_1]) - F[0,t_1]\| \leq \epsilon$ if $n > F_{diff}/\epsilon\delta^2$.

We will find bounds on the values of F_{diff} and δ as a function of t_1.

$$F_{diff} = \max_{x,y\in[0,t_1]} \|F(x) - F(y)\|$$

$$= \max_{x,y\in[0,t_1]} \|e^{Ax}x_0 - e^{Ay}x_0\| \leq \|A\|e^{\|A\|t_1}\|x_0\|t_1$$

(See proof of Lemma 6.)

Next we need to find a lower bound on δ such that

$$\forall x, y \in [0,t_1], |x-y| \leq \delta \implies \|F(x) - F(y)\| \leq \epsilon.$$

Or equivalently

$$\max_{x,y\in[0,t_1],|x-y|\leq\delta} \|F(x) - F(y)\| \leq \epsilon.$$

However,

$$\max_{x,y\in[0,t_1],|x-y|\leq\delta} \|F(x) - F(y)\| \leq \|A\|e^{\|A\|t_1}\|x_0\|\delta$$

(again from the proof of Lemma 6).

Hence it suffices to choose a δ which ensures

$$\|A\|e^{\|A\|t_1}\|x_0\|\delta \leq \epsilon.$$

Hence we can choose

$$\delta = \epsilon/(\|A\|e^{\|A\|t_1}\|x_0\|).$$

We want to choose a t_1 so as to satisfy $m > F_{diff}/\epsilon\delta^2$. It suffices to satisfy

$$m > \frac{\|A\|e^{\|A\|t_1}\|x_0\|t_1)}{(\epsilon(\epsilon/(\|A\|e^{\|A\|t_1}\|x_0\|))^2)}.$$

Or,

$$m\epsilon^3 > \|A\|^3 e^{3\|A\|t_1}\|x_0\|^3 t_1.$$

For t_1 such that

$$e^{3\|A\|t_1}t_1 < n\epsilon^3/\|A\|^3\|x_0\|^3,$$

we have $\|F(x) - B_m(F(x))\| \leq \epsilon$ for all $0 \leq x \leq t_1$. ∎

There always exists a positive real number $t_1 \leq t$ satisfying the inequality $e^{at_1}t_1 < b$ where $a = 3\|A\|$ and $b = m\epsilon^3/\|A\|^3\|x_0\|^3$, since the function $e^{ax}x \to 0$ as $x \to 0$. Computing a t_1 such that $e^{at_1}t_1 = b$ might not be possible, instead one can obtain an upper bound on this value. For example, we know that $t_1 \leq t$. Hence we can consider $t_1 = b/e^{at}$. We use the following alternative bound. If $at_1 < 1$, then we can upper bound e^{at_1} by $1/(1 - at_1)$. This gives us a bound $t_1 < b/(1 + ab)$. Hence $t_1 < \min\{b/(1 + ab), 1/2a_1\}$ is a positive bound for t_1.

The algorithm for computing a piecewise polynomial approximation of a linear dynamical system is given in Algorithm 2.

Algorithm 2 Post Computation Algorithm for Linear Dynamical Systems

Input: $m \in \mathbb{N}, \epsilon \in \mathbb{R}_{\geq 0}, V_0 \subseteq_{Fin} \mathbb{R}^n, A \in \mathbb{R}^{n\times n}, T > 0$
Output: Sequence of a set of n polynomials

Let $F^x : [0, T] \to \mathbb{R}^n$ be $F^x(t) = e^{At}x$
for all $v \in V_0$ **do**
  $t := 0$
  $x := v$
  while $t < T$ **do**
    Choose $\tau_1 > 0$ s.t
    $e^{3\|A\|\tau_1}\tau_1 < m\epsilon^3/\|A\|^3\|x\|^3$
    Let $\tau_2 = \log_e(2\sqrt{m}\epsilon/\|A\|\|x\|)/\|A\|$
    Let $t_i = \max \tau_1, \tau_2$
    Output $Bern_m(F_j^v[t, t + t_i])$, for each $1 \leq j \leq n$
    $t := t + t_i$
    $x := e^{At}v$
  end while
end for

4.3.3 Termination of the Algorithm

In each step, we take as the next time step the maximum of the values obtained by methods in Lemma 6 and Lemma 7. This time step is always going to be positive, since Method 2 always gives a positive answer. Next we show that the time step we choose in any iteration has a positive lower bound. Hence, the algorithm always terminates.

Assume that in each step of method 2, we choose $t_i = \min\{b/(1+ab), 1/2a_1\}$.

$$b/(1+ab) = \frac{(m\epsilon^3/(\|A\|^3\|x_i\|^3))}{(1+(am\epsilon^3/(\|A\|^3\|x_i\|^3)))}$$

$$= \frac{(m\epsilon^3/(\|A\|^3))}{(\|x_i\|^3+(am\epsilon^3/(\|A\|^3)))}$$

$$\geq \frac{(m\epsilon^3/(\|A\|^3))}{(e^{aT}\|x_0\|^3+(am\epsilon^3/(\|A\|^3)))}$$

Therefore, the time steps t_i are lower bounded by a positive number.

Figure 4.1 and Figure 4.2 illustrate the difference between the constant step and varying step algorithms. For each algorithm, the points (t, y) are plotted, where t ranges over the times $\sum_{j=0}^{i} t_i$, where $t_1, t_2, \cdots$ is the sequence of time steps chosen by the algorithm, and y is given by $e^{At}x$. Observe that in the case of timestep varying algorithm, initially larger steps are chosen, and when the time approaches close to T, the timesteps taken by the constant time step algorithm and the varying time step algorithm become identical. Notice that the sampling rate of the varying time step algorithm depends on the value of the derivative of the function at various points, where as the constant time step algorithm makes no such distinction.

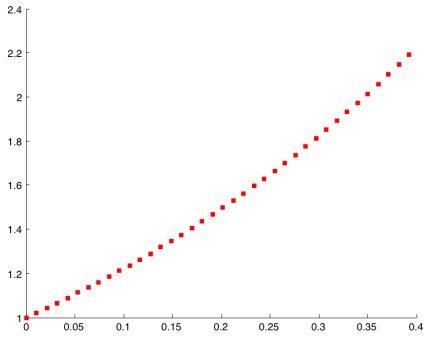

Figure 4.1: Constant time step Algorithm

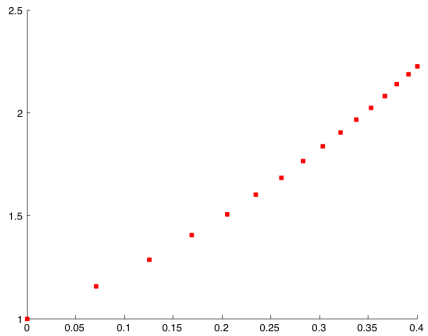

Figure 4.2: Varying time step Algorithm

4.3.4 Comparison with Other Polynomial Approximations

In this chapter, we considered Bernstein polynomials to approximate an arbitrary function by a polynomial function. Another popular technique to

obtain polynomial approximations is to use Taylor's series expansion of the function and truncate the infinite sum after some points to obtain a polynomial. There are a few caveats in using Taylor's approximation in general. First it assumes that the function is smooth, and the derivatives can be computed. Secondly, it does not give a closed form expression for m, the number of terms in the Taylor's expansion that should be considered to obtain an ϵ bound on the approximation error.

4.4 Experimental Evaluation

In this section, we explain our experimental set up for evaluating the performance of our algorithm and comparison with other methods. We implemented our algorithm in MATLAB 7.4.0, and the experiments were conducted on Mac OS X Version 10.4.11, with a 2.16 GHz Intel Core 2 Duo processor and 1GB SDRAM. We performed our experiments on linear and quadratic approximations. We will report and explain our results for both the approximations in the following sections.

4.4.1 Linear Approximation

Our experimental evaluation of the two algorithms chooses to be agnostic of the relative benefits of different data structures, and attempts to highlight the relative advantages of each algorithm, independent of the chosen data structure. No matter what the chosen data structure, the algorithms require computing the states reached at certain time steps. Once these points are computed, the data structure approximating a convex hull of the points needs to be computed. Thus, in our experimental evaluation we only compared the computational costs of finding the states reached at the required times using the two algorithms. In the basic algorithm, the interval size is fixed to be Δ, and then the states reached at time Δ are computed by multiplying initial set of states with matrix exponential $e^{A\Delta}$, but for subsequent times $i\Delta$ they are computed by translation that involves only multiplication with $e^{A\Delta}$ which is evaluated only once. In contrast, in our algorithm, the states reached at each of the designated time steps is computed from scratch using matrix exponentials, and the size of the next interval is found dynamically. Recall

Matrix	ϵ	T	n	m	t_{max}	t_{min}	Δ
$2DR$	4.93E-03	1	3.34E+02	8.53E+02	5.54E-03	1.78E-03	1.17E-03
$2DR$	6.78E-02	2	1.80E+01	1.60E+03	1.96E-01	7.81E-02	1.25E-03
$2DR$	3.60E-01	5	1.20E+01	4.27E+03	4.83E-01	4.83E-01	1.17E-03
$2DR$	1.85E+00	10	9.20E+01	8.47E+03	4.60E-01	9.42E-03	1.18E-03
$5DR$	4.92E-01	1	1.00E+01	5.65E+02	1.23E-01	1.07E-01	1.77E-03
$5DR$	3.14E+00	2	1.50E+01	1.14E+03	1.44E-01	1.44E-01	1.76E-03
$5DR$	2.11E+02	5	3.30E+01	2.99E+03	1.57E-01	1.57E-01	1.67E-03
$5DR$	3.04E+05	10	6.20E+01	6.15E+03	1.65E-01	1.65E-01	1.63E-03
$100DR$	2.01E-02	1	3.70E+02	9.16E+02	2.76E-03	2.61E-03	1.09E-03
$100DR$	2.56E-02	2	3.17E+02	1.84E+03	6.57E-03	5.93E-03	1.09E-03
$100DR$	6.13E-02	5	8.20E+01	4.59E+03	7.69E-02	4.97E-02	1.09E-03
$100DR$	2.65E-01	10	1.30E+01	9.17E+03	8.65E-01	8.22E-01	1.09E-03

Table 4.1: Random Martices: Comparison of the Number of Subintervals in the Approximation

Matrix	ϵ	T	RT_V	RT_C
$2DR$	4.93E-03	1	1.80E-01	2.79E-02
$2DR$	6.78E-02	2	1.77E-02	1.15E-01
$2DR$	3.60E-01	5	1.68E-02	4.99E-01
$2DR$	1.85E+00	10	5.64E-02	1.51E+00
$5DR$	4.92E-01	1	1.49E-02	4.69E-02
$5DR$	3.14E+00	2	1.96E-02	1.15E-01
$5DR$	2.11E+02	5	2.95E-02	4.97E-01
$5DR$	3.04E+05	10	4.81E-02	1.62E+00
$100DR$	2.01E-02	1	4.81E+00	5.33E-01
$100DR$	2.56E-02	2	4.42E+00	2.93E+00
$100DR$	6.13E-02	5	1.24E+00	2.14E+01
$100DR$	2.65E-01	10	4.05E-01	9.23E+01

Table 4.2: Random Matrices: Comparison of the Time Taken for Constructing the Approximation

from Section 4.3, that when the initial set is a convex polyhedron, reach set computation involves computing this approximated flow for each vertex. Thus, in our experimental evaluation, we start from a single point, as this will faithfully reflect the costs of starting from a polyhedron. Finally, the error bound chosen for the varying time step algorithm was the one guaranteed by the fixed time step algorithm.

To determine feasibility of the algorithms, we first ran them on some randomly generated matrices. The entries of the matrices were random values in the interval $[-1, 1]$. The results of our experiments are shown in Table 4.1 and Table 4.2. The rows labelled $2DR$ report results for 2×2 matrices,

those labelled $5DR$ for 5×5 matrices, and finally those labelled $100DR$ for 100×100 matrices. The columns reported in the table are as follows: ϵ gives the error bound; T gives the time bound chosen for the experiment; m and n are the number of sub-intervals used by the constant timestep algorithm and varying timestep algorithms, respectively; t_{max} and t_{min} are the largest and smallest time intervals considered by the varying timestep algorithm; Δ is the size of the interval used by the constant timestep algorithm; and RT_C and RT_V are the running times of the constant timestep and varying timestep algorithms, respectively. For these matrices, we chose a time step for the constant timestep algorithm to be of the order of 10^{-3}, and used the resulting error bound as ϵ. The results show that the varying timestep algorithm is scalable and has a running time comparable to the constant timestep algorithm; in many cases the varying timestep algorithm is faster by 2 orders of magnitude. The number of sub-intervals used by the varying timestep algorithm (n) is always less than that used by the constant timestep method (m) by either a magnitude or two orders of magnitude. The other surprising observation is that the smallest time interval used by the varying timestep method is in all cases larger than the interval used by the constant timestep algorithm.

We also experimented on benchmark examples considered in [45]. *Nav* is the navigation benchmark first suggested in [40], while $Z2$ and $Z5$ are the 2 dimensional and 5 dimensional examples from [45]. The matrices describing their dynamics is as follows.

$$Nav = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & -1.2 & 0.1 \\ 0 & 0 & 0.1 & -1.2 \end{bmatrix}, Z2 = \begin{bmatrix} -0.1 & -0.4 \\ 0.4 & -0.1 \end{bmatrix}$$

$$Z5 = \begin{bmatrix} -0.1 & -0.4 & 0 & 0 & 0 \\ 0.4 & -0.1 & 0 & 0 & 0 \\ 0 & 0 & -0.3 & 0.1 & 0 \\ 0 & 0 & -0.1 & -0.3 & 0 \\ 0 & 0 & 0 & 0 & -0.2 \end{bmatrix}$$

We tried to study the effect of increasing the time bound T on the running time of these algorithms and so we considered $T = 1, 2, 3$. Table 4.3 and Table

Matrix	ϵ	T	n	m	t_{max}	t_{min}	Constant
$Z2$	1E-01	1	1.30E+02	7.47E+02	1.13E-02	5.96E-03	1.34E-03
$Z2$	1E-01	2	1.86E+02	6.69E+03	2.69E-02	5.96E-03	2.99E-04
$Z2$	1E-01	3	2.42E+02	4.49E+04	2.69E-02	5.96E-03	6.68E-05
$Z5$	1E-01	1	1.04E+02	5.61E+02	1.34E-02	7.91E-03	1.78E-03
$Z5$	1E-01	2	1.52E+02	5.02E+03	3.82E-02	7.91E-03	3.98E-04
$Z5$	1E-01	3	1.87E+02	3.38E+04	3.82E-02	7.91E-03	8.89E-05
Nav	1E+00	1	7.00E+00	5.00E+00	1.92E-01	1.92E-01	1.92E-01
Nav	1E+00	2	1.20E+01	9.30E+01	1.92E-01	1.92E-01	2.14E-02
Nav	1E+00	3	1.70E+01	6.37E+03	1.92E-01	1.92E-01	4.71E-04

Table 4.3: Standard Examples: Comparison of the Number of Subintervals for Total Time $T = 1, 2, 3$

Matrix	ϵ	T	RT_V	RT_C
$Z2$	1E-01	1	7.43E-02	2.55E-02
$Z2$	1E-01	2	1.04E-01	9.95E-01
$Z2$	1E-01	3	1.33E-01	2.78E+01
$Z5$	1E-01	1	6.22E-02	4.00E-02
$Z5$	1E-01	2	8.75E-02	1.04E+00
$Z5$	1E-01	3	1.07E-01	4.33E+01
Nav	1E+00	1	1.47E-02	2.30E-02
Nav	1E+00	2	1.65E-02	2.30E-02
Nav	1E+00	3	1.95E-02	1.45E+00

Table 4.4: Standard Examples: Comparison of the Time for Constructing the Approximation for Total Time $T = 1, 2, 3$

4.4 show our results for these benchmark examples and varying time. It shows that as T increases, the varying timestep algorithm's relative performance improves both in terms of the number of sub-intervals considered and the running time, with the gap increasing to as much two orders of magnitude.

4.4.2 Quadratic Approximation

We also implemented the varying time step algorithm for approximation by piecewise quadratic approximation. Theoretically, a single timestep of the quadratic approximation could be twice as much as that of the linear approximation. Interestingly, this could lead to a huge reduction in the number of total time steps. Consider an exponentially growing function, for which the time steps chosen by the linear approximation decrease by a constant factor in consecutive time steps. For example, consider the following sequence of

Matrix	t_{min}	t_{max}	n	t^Q_{min}	t^Q_{max}	n_Q
$Z2$	5.96E-03	1.13E-02	1.30E+02	7.21E-03	2.20E-02	6.70E+01
$Z2$	5.96E-03	2.69E-02	1.86E+02	1.26E-05	5.10E-02	9.60E+01
$Z2$	1.00E-03	2.69E-02	2.42E+02	1.18E-02	5.10E-02	1.24E+02
$Z5$	2.48E-03	1.34E-02	1.04E+02	1.56E-02	2.55E-02	5.30E+01
$Z5$	1.60E-03	3.82E-02	1.52E+02	1.56E-02	6.45E-02	7.80E+01
$Z5$	7.91E-03	3.82E-02	1.87E+02	1.56E-02	7.02E-02	9.70E+01
Nav	1.92E-01	1.92E-01	7.00E+00	1.92E-01	1.92E-01	7.00E+00
Nav	1.92E-01	1.92E-01	1.20E+01	1.92E-01	1.92E-01	1.20E+01
Nav	1.92E-01	1.92E-01	1.70E+01	1.92E-01	1.92E-01	1.70E+01

Table 4.5: Comparison of the Number of Subintervals in the Linear and Quadratic Approximations

Matrix	RT_V	RT_Q
$Z2$	9.88E-02	4.74E-02
$Z2$	1.28E-01	5.98E-02
$Z2$	1.60E-01	7.49E-02
$Z5$	1.21E-01	4.10E-02
$Z5$	1.12E-01	5.17E-02
$Z5$	1.31E-01	6.25E-02
Nav	3.54E-02	1.10E-02
Nav	3.84E-02	1.44E-02
Nav	4.09E-02	1.57E-02

Table 4.6: Comparison of the Running Times in the Linear and Quadratic Approximations

timesteps $1, 1/2, 1/2^2, \cdots, 1/2^k$. A doubling of the time step in the quadratic approximation could lead to the skipping of k timesteps in the above example. However, we did not observe this phenomenon in our experiments which are tabulated in Table 4.5 and Table 4.6. The columns t_{min}, t_{max}, RT_V, and n report the smallest time interval, largest time interval, running time, and number of intervals when using a linear approximations; the columns with superscript or subscript Q refer to the same quantities for the quadratic approximation. The improvement for the quadratic approximation was not as dramatic as we hoped it might, and in the best case was better by a factor of 2.

4.5 Conclusions

In this chapter, we presented an algorithm for approximating the flow of a Linear Dynamical System by a piecewise polynomial function. This gives us a fully automatic method for safety verification of linear hybrid systems in combination with the approximation technique described in the previous chapter. Moreover, the dynamic nature of our algorithm in constructing the approximation gives us a scalable technique which constructs smaller approximations than those constructed by previously existing algorithms and is faster as seen in our experiments on standard and random examples. In the future, we intend to extend this algorithm to more general classes of hybrid systems.

CHAPTER 5

HYBRID CEGAR

Chapter 3 and Chapter 4 focused on developing techniques for constructing approximations of systems parametrized by a bound on the error between the original and the approximate system. These techniques provide a method for proving safety of systems by iteratively constructing more and more precise abstractions by reducing the error of approximation. However, they ignore the property being analysed in constructing these abstractions. Constructing and refining abstractions based on the property seems a natural approach to pursue, and one of the popular methods for property based abstraction refinement is a method called *counterexample guided abstraction refinement* (CEGAR) in which an abstraction is refined by examining a "counterexample" which is a witness to the violation of the property by the abstract system.

In this chapter, we present a framework for carrying out CEGAR for systems modelled as rectangular hybrid automata (RHA). We choose as the class of abstract systems, the subclass of RHA called the initialized rectangular hybrid automata (IRHA), for which safety verification is decidable. The main difference, between our approach and previous proposals for CEGAR for hybrid automata, is that we consider the abstractions to be hybrid automata as well. We show that the CEGAR scheme is complete, when applied to initialized rectangular automata, and is semi-complete for the class of RHA. We have implemented the CEGAR based algorithm in a tool called HARE, that makes calls to HyTech to analyze the abstract models and validate the counterexamples. Our experiments demonstrate the usefulness of the approach.

5.1 An Overview

As discussed earlier, direct model checking of realistic hybrid systems is in general undecidable and often foiled by the state-space explosion problem. Hence, one has to rely on some sort of abstraction. Finding the right abstraction is in itself a difficult problem. To this end, the *counterexample guided abstraction refinement* (CEGAR) [28] technique which combines automatic abstraction with model checking has gained preeminence in a number of contexts [12, 57, 29, 53] including in timed and hybrid systems [3, 26, 27, 98, 39, 97, 36, 60]. CEGAR begins with a coarse initial abstract model that is progressively refined based on model checking the abstraction and analyzing the counterexamples generated by the model checker, until either a valid counterexample is found or an abstraction satisfying the safety property is obtained.

The space over which CEGAR performs the abstractions and refinements is key in determining both the efficiency (of model checking) and the completeness of the procedure. For example, in [3, 26, 27, 98, 97] abstraction-refinement is carried out in the space of finite-state discrete transition systems. In other words, the (infinite) configuration space of the hybrid automaton is partitioned into finitely many equivalence classes, and the abstraction has as states these equivalence classes; the abstraction has no continuous dynamics. The partitioning of the configurations maybe based on predicates [3, 98, 97] as in predicate abstraction [50]. Computing the transitions for this abstract finite state machine involves computing the unbounded time reachable states from the equivalence classes of the concrete system, which is difficult in practice for hybrid systems with complex continuous dynamics. The second proposal [36, 60] is a CEGAR framework for hybrid automata where the abstractions are constructed by ignoring certain variables. Counterexamples are used to identify new variables to be added to the abstraction. This approach has been carried out for timed automata [36] and linear hybrid automata [60].

In this chapter, we generalize the approach in [36, 60] to present a *hybrid abstraction*-based CEGAR framework for rectangular hybrid systems. We abstract such automata using *initialized rectangular hybrid automata* [54], and not finite transition systems. The choice of initialized rectangular hybrid automata as the abstract space is motivated by the desire to have a rich

space of abstractions, with a decidable model checking problem, and effective tools to analyze them. We generalize the results in [36, 60] in several directions. First, in our abstract hybrid automata, we allow control states and transitions to be collapsed; thus, the control structure of the abstraction is not the same as that of the original automaton. In addition, the continuous variables in the abstract hybrid automaton may be a subset of the original variables or scaled versions of the original variables. Variable scaling changes the constants that appear in the abstract hybrid automaton which in turn can positively influence the efficiency of model checking the abstract automaton. Our refinement algorithm is more involved than for abstractions where only certain variables are dropped; it may involve splitting control states/transitions, and/or adding variables that may have new dynamics.

Our main result in this chapter is a hybrid abstraction-based CEGAR framework for the class of *rectangular hybrid automata* (RHA) [54]. These are systems in which the evolution of the continuous variables is constrained such that the derivative of the flows belong to a "rectangular constraint". Further, the invariants, guards and resets are also specified using rectangular regions. This is an interesting albeit simple class of systems since various classes of systems with complex continuous dynamics can be approximated to this class with arbitrary precision [93]. Unfortunately, safety verification is undecidable even for the class of RHA; and hence, we choose as our abstraction space, a subclass of RHA called *initialized rectangular hybrid automata* for which safety verification is decidable [54]. We present algorithms for counterexample analysis and refinement in this setting. If the concrete automaton is initialized, we prove that the procedure is complete, that is, it will terminate and either prove that the concrete automaton satisfies the given safety property or demonstrate a bug by constructing a valid counterexample. More generally, for the class of rectangular hybrid automata, we show that the method is semi-complete, that is, if the system is safe, it may or may not terminate, however, if the system is buggy, it is guaranteed to terminate.

We have implemented our CEGAR based algorithm for rectangular hybrid automata in a tool that we call **H**ybrid **A**bstraction **R**efinement **E**ngine (HARE). HARE makes calls to HyTech [56] to analyze abstract models and generate counterexamples; we considered PHAVer [41], but at the time of writing it does not produce counterexamples. We analyzed the tool on sev-

eral benchmark examples which illustrate that its total running time is comparable with that of HyTech, and on some examples HARE is a couple of orders of magnitude faster. Moreover, in some cases HARE can prove safety with dramatically small abstractions. Fair running-time comparison of HARE with other MATLAB-based tools, such as d/dt [33] and checkmate [26], is not possible because of the differences in the runtime environments. Experimental comparison of finite-state and hybrid abstractions was also not possible because to the best of our knowledge, the tools implementing finite-state abstractions are not publicly available. We believe that in terms of efficiency, the approaches of finite state abstractions and hybrid abstractions are incomparable, and we present examples where CEGAR using hybrid abstractions are demonstrably more efficient than CEGAR using finite-state counterparts.

Related Work We briefly distinguish our contribution from existing results in the literature. In [3, 27], the authors consider finite-state abstractions. When compared to using finite-state abstractions, using hybrid abstractions in a CEGAR framework requires carrying out computationally simpler tasks when constructing abstractions, refining them and validating counterexamples. Constructing finite-state abstractions requires computing time successors, which is computationally expensive. In contrast, constructing hybrid abstractions only involves making local checks about flow equations. Moreover when validating counterexamples in a hybrid CEGAR scheme, one is only required to compute time-bounded successors. Computing time-bounded successors is often computationally easier than computing time-unbounded successors (required in constructing discrete abstractions); for example, for automata with linear differential dynamics, time-bounded posts can be efficiently approximated, while no such algorithms exist for time-unbounded posts.

When compared to hybrid CEGAR schemes described in [36, 60], our abstractions (may) change both the control graph and variable dynamics, and are not restricted to only forgetting continuous variables. In contrast, the scheme in [60] considers a more general class of hybrid automata, though the abstractions in that scheme are not progressively refined.

Finally, in [37] hybrid systems with flows described by linear differential equations are approximated by rectangular HA. Even though, their scheme

progressively refines abstractions, the refinements are not guided by counter-examples.

5.2 Preliminaries

A *rectangular constraint* $B \subseteq \mathbb{R}^n$ is a Cartesian product of intervals whose finite end-points are in $\mathcal{I}$, that is, $B = \prod_{i=1}^{n} (B)_i$ and each $(B)_i$ is an interval whose end-points belong to $\mathcal{I} \cup \{-\infty, +\infty\}$. The set of all rectangular constraints in $\mathbb{R}^n$ is denoted by *RectConst*(n). A *multirectangular constraint* is a finite set of rectangular constraints of the same dimension. The set of all multirectangular constraints of dimension n is denoted by *MRectConst*(n). Given a multirectangular constraint $R = \{R_1, \cdots, R_m\}$, where each $R_i \in$ *RectConst*(n), $[\![R]\!]$ is the set $\bigcup_i R_i$. Given $R, S \in$ *MRectConst*(n), we say that $R \preceq S$ if for every $R' \in R$, there exists a $S' \in S$ such that $R' \subseteq S'$. Given $S \subseteq \mathbb{R}^n$, the *rectangular hull* of S, denoted *RectHull*(S), is the smallest rectangular constraint in *RectConst*(n) containing S. Sets which are rectangular, but are obtained from intervals whose endpoints are not necessarily integers but reals are called *rectangular regions*. A *multirectangular region* $R \subseteq \mathbb{R}^n$ is a finite set of rectangular regions. We denote by *RectReg*(n) the set of all rectangular regions which are subsets of $\mathbb{R}^n$, and by *MRectReg*(n) the set of all multirectangular regions which are subsets of $\mathbb{R}^n$. Given multirectangular regions R and S, $[\![R]\!]$ and $R \preceq S$ are defined similar to the case of multirectangular constraints.

In this chapter, we will consider transition systems in which states are not labelled. Hence, the transition system in this chapter will contain only 4 component as in $(S, S^0, Act, \{\rightarrow\}_{a \in Act})$, that is, we drop the components which specify the set of state labels and the state labelling function. The notion of simulation between transition systems $\mathcal{T}_i = (S_i, S_i^0, Act, \{\rightarrow_i\}_{a \in Act})$, for $i = 1, 2$ is the same as before except that the condition on state labels is ignored. Further, we call a function $\alpha : S_1 \rightarrow S_2$ a *simulation function* if the relation $\{(s, \alpha(s)) \mid s \in S_1\}$ is a simulation relation between $\mathcal{T}_1$ and $\mathcal{T}_2$, and we denote this fact by $\mathcal{T}_1 \preceq_\alpha \mathcal{T}_2$.

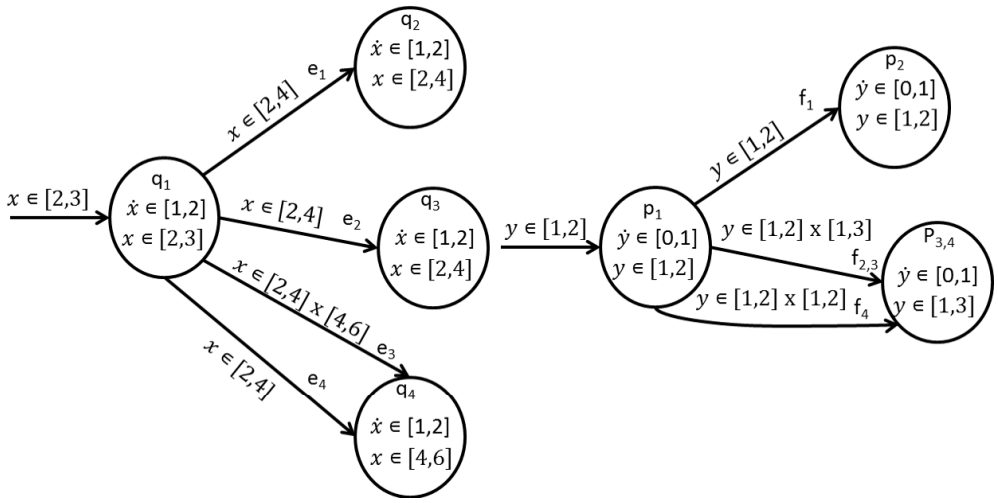

Figure 5.1: An example of a rectangular hybrid automaton

5.3 Rectangular Hybrid Automata (*RHA*)

In this chapter, we will consider a subclass of hybrid automata called rectangular hybrid automata in which all the guards, resets, invariants, and flows are described by rectangular constraints. We will consider unlabelled hybrid systems, that is, the edges and the locations do not have labels. Hence, we represent edges as a set of elements and specify the source and target of each element. Further, the flows of the automaton are non-deterministic, that is, the continuous state is allowed to evolve in more than one way starting from a particular state. The flow function is represented by specifying for each variable an interval in which the derivative of the flow corresponding to the variable is required to lie in. Figure 5.1 shows an example of a rectangular hybrid automaton with four discrete states and a variable x. The formal definition of a rectangular hybrid automaton is given below.

Formally, a *rectangular hybrid automaton* (*RHA*) $\mathcal{H}$ is a tuple (*Loc*, *Edges*, *Source*, *Target*, n, *init*, $Cont_0$, *inv*, *activity*, *reset*) where

- *Loc* is a finite set of *(discrete) control states* or *locations*.
- *Edges* is a finite set of *edges*.
- *Source*, *Target*: $Edges \rightarrow Loc$ are functions which associate a source and a target location to every edge, respectively.

- $n \in \mathbb{N}$ is the dimension denoted by $Dim(\mathcal{H})$, and $Cont = \mathbb{R}^n$ is the set of continuous states.
- $init \in Loc$ is the *initial location* and $Cont_0 \in RectConst(n)$ is the *initial set of continuous states.*
- inv: $Loc \rightarrow RectConst(n)$ associates with every location an *invariant.*
- *activity*: $Loc \rightarrow RectConst(n)$ associates with every location an *activity* set.
- *reset*: $Edges \times [n] \rightarrow (RectConst(1) \cup RectConst(2))$ associates with each edge and an index either a rectangular set of dimension 1, in which case it is called an identity reset, or a rectangular set of dimension 2 in which case it is called a strong reset.

The above is the standard definition of a RHA (e.g., [54]) except that here we have combined the guards and the reset maps into the *reset* function. This is explained with the aid of the example shown in Figure 5.1.

Example 33 *The set of locations* $Loc = \{q_1, q_2, q_3, q_4\}$ *and the set of edges in* $Edges = \{e_1, e_2, e_3, e_4\}$. *The source of edge* e_1 *is* q_1 *and the target is* q_2. *The activities in a location is given by an expression of the form* $\dot{x} \in I$ *and the invariant is given by an expression of the form* $x \in J$, *where* I *and* J *are intervals. The reset associated with the edge* e_1 *is an* identity reset *given by* $reset(e_1, 1) = [2, 4]$, *and belongs to* $RectConst(1)$. *Hence, if the execution is in location* q_1 *with value of* x *being* 2, *the edge* e_1 *is enabled since* 2 *belongs to the 1-dimensional reset interval* $[2, 4]$ *associated with the edge* e_1. *The value of* x *after the transition remains equal to* 2. *On the other hand, a reset specified as a product of two intervals corresponds to a* strong reset, *as in the case of the edge* e_3, *for which* $reset(e_3, 1) = [2, 4] \times [4, 6] \in RectConst(2)$. *Such an edge can be taken only if the value of the variable before taking the edge belongs to the first interval and the value of the variable after taking the edge is non-deterministically reset to some value in the second interval. For example, starting with value* 2 *for* x *in location* q_1, *the edge* e_3 *is enabled and after the edge is taken the value of* x *is reset to some value in* $[4, 6]$.

Notation 34 *We will use* $\mathcal{H}_1$, $\mathcal{H}_2$, $\mathcal{H}_A$, $\mathcal{H}_C$ *and so on to represent rectangular hybrid automata, and we will use subscripts to denote the components,*

for example, the set of locations of $\mathcal{H}_1$ is denoted by Loc_1 and the invariant function of $\mathcal{H}_A$ is denoted by inv_A.

The reset of an edge defines a binary relation on $\mathbb{R}^n$ containing pairs of continuous state changes allowed by the reset. Let us define the binary relation $ResetRel(e)$ to be the set of all points $(x, y) \in \mathbb{R}^n \times \mathbb{R}^n$ such that for all $i \in [n]$, if $reset(e, i) \in RectConst(1)$, then $(x)_i \in reset(e, i)$ and $(y)_i = (x)_i$, and if $reset(e, i) \in RectConst(2)$, then $((x)_i, (y)_i) \in reset(e, i)$.

The semantics of a *RHA* $\mathcal{H}$ is defined in terms of the transition system $[\![\mathcal{H}]\!] = (Q, Q^0, \Sigma, \{\rightarrow_a\}_{a\in\Sigma})$ over $\Sigma = \mathbb{R}_{\geq 0} \cup Edges$, where $Q = Loc \times Cont$, $Q^0 = init \times Cont_0$, and the transition relation $\{\rightarrow_a\}_{a\in\Sigma}$ is given by:

- *Continuous transitions* - For $t \in \mathbb{R}_{\geq 0}$, $(q_1, x_1) \xrightarrow{t} (q_2, x_2)$ iff $q_1 = q_2 = q$ and there exists a continuously differentiable function $f : [0, t] \rightarrow \mathbb{R}^n$, such that $x_1 = f(0)$, $x_2 = f(t)$ and for all $t' \in [0, t]$, $f(t') \in inv(q)$ and $f'(t) \in activity(q)$, where $f' = df/dt$ is the derivative of f with respect to time.

- *Discrete transitions* - For $e \in Edges$, $(q_1, x_1) \xrightarrow{e} (q_2, x_2)$ iff $q_1 = Source(e)$, $q_2 = Target(e)$, and $(x_1, x_2) \in ResetRel(e)$.

We will denote the set of states of $[\![\mathcal{H}]\!]$, namely Q, by $States_{\mathcal{H}}$. Given a set $S \subseteq States_{\mathcal{H}}$ and $q \in Loc$, $S|_q$ will denote the restriction of S to q, namely, the set $\{(q, x) \,|\, (q, x) \in S\}$, $Cont(S)$ will denote the continuous part of the states of S, that is, $Cont(S) = \{x \,|\, \exists q, (q, x) \in S\}$. When there is no ambiguity we will use $\mathcal{H}$ and $[\![\mathcal{H}]\!]$ interchangeably. For example, we use $Pre_{\mathcal{H}}$ and $Post_{\mathcal{H}}$ to denote $Pre_{[\![\mathcal{H}]\!]}$ and $Post_{[\![\mathcal{H}]\!]}$, respectively.

An *execution fragment* σ of $\mathcal{H}$ is a finite path in $[\![\mathcal{H}]\!]$ with alternating discrete and continuous transitions. An *execution* of $\mathcal{H}$ is an execution fragment starting with a state (q, x) in Q^0. We are interested in the *control state reachability problem* for hybrid systems, which is to determine, given a *HA* $\mathcal{H}$ and a location q, if there exists an execution reaching some state with location q. As a general rule we drop $\mathcal{H}$ from the notation whenever it is clear from the context.

For notational convenience, we will also need a slightly different class of automata called multireset rectangular hybrid automata (*MRHA*). A multireset *RHA* is similar to a *RHA* except that the reset function is now a function of the form $reset : Edges \times [n] \rightarrow MRectConst(1) \cup MRectConst(2)$, that is, the

resets are multirectangular sets instead of rectangular sets. The reset relation *ResetRel* associated with an edge e of a *MRHA* is defined in a similar manner to that for a *RHA*. Given an edge e of a *MRHA*, *ResetRel*(e) is given by the set of all points $(x, y) \in \mathbb{R}^n \times \mathbb{R}^n$ such that for every $i \in [n]$, if $\mathit{reset}(e, i) \in \mathit{MRectConst}(1)$, then $(x)_i \in R$ for some $R \in \mathit{reset}(e, i)$ and $(y)_i = (x)_i$, and if $\mathit{reset}(e, i) \in \mathit{MRectConst}(2)$, then $((x)_i, (y)_i) \in R$, for some $R \in \mathit{reset}(e, i)$. For example, in a one dimensional *MRHA* if $\mathit{reset}(e, 1) = \{[1, 2], [3, 4]\}$, then $\mathit{ResetRel}(e) = \{(x, x) \mid x \in [1, 2]\} \cup \{(x, x) \mid x \in [3, 4]\}$. If $\mathit{reset}(e, 1) = \{[1, 2] \times [3, 4], [3, 4] \times [4, 5]\}$, then $\mathit{ResetRel}(e, 1) = ([1, 2] \times [3, 4]) \cup ([3, 4] \times [4, 5])$. Note that every *RHA* can be converted to a *MRHA* by replacing every rectangular set in its reset function by a multirectangular set with one element. From now on, by an *RHA*, we mean an *MRHA* in which the reset function consists of only multirectangular sets which are singletons. Similarly, every *MRHA* can be converted to an *RHA* by replacing every edge by some finite number of edges depending on the number of elements in the multirectangular sets in the reset associated with that edge.

We will consider a special class of rectangular hybrid automata called initialized rectangular hybrid automata which have the property that identity resets are allowed on an edge for an index i only when the activity function associated with the corresponding variable doesn't change. We say that a *MRHA* $\mathcal{H}$ is *initialized* (*MIRHA*) if for every $e \in \mathit{Edges}$ and $i \in [n]$, $(\mathit{activity}(\mathit{Source}(e)))_i \neq (\mathit{activity}(\mathit{Target}(e)))_i$ implies $\mathit{reset}(e, i) \in \mathit{MRectConst}(2)$. Also, an initialized rectangular hybrid automaton is a *MIRHA* which is also an *RHA*. The reachability problem is decidable for the class of initialized rectangular hybrid automata [54]. Since every *MIRHA* can be translated to an initialized *RHA* (which is equivalent with respect to reachability verification) by using the same translation as from *MRHA* to *RHA*, the reachability problem is also decidable for the class of *MIRHA*.

5.4 CEGAR for Rectangular Hybrid Automata

In this section, we present a counterexample guided abstraction refinement method for the class of rectangular hybrid automata, and show that the method is complete for the class of initialized rectangular hybrid automata and semi-complete for the class of *RHA*. We will first describe the general

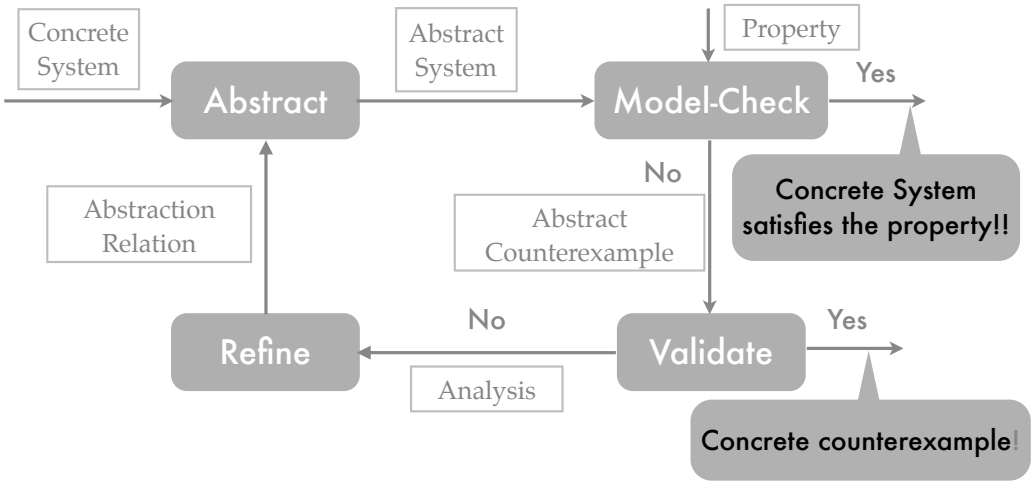

Figure 5.2: Counterexample Guided Abstraction Refinement Approach

CEGAR framework.

5.4.1 CEGAR Framework

A counterexample guided abstraction refinement algorithm consists of the four steps which are shown in Figure 5.3. CEGAR loop begins with the construction of an initial abstraction of the original system (also called concrete system). The abstract system is then model-checked to check if there are any executions leading to the unsafe control state. Such an execution if one exists is called an *abstract counterexample*. If the abstract system has no counterexamples, then it can be deduced from the properties of abstraction that even the concrete system does not have any counterexamples. However, if an abstract counterexample is returned in the model-checking phase, then it is validated to check if there are any concrete counterexamples corresponding to it. If a concrete counterexamples is found, then the concrete system is unsafe, and the concrete counterexamples is returned to the user. If no concrete counterexamples are found, then the analysis is used to construct a new abstract system which is a refinement of the current abstract system. The CEGAR algorithm continues with model-checking of the new abstract system. In general, the CEGAR algorithm might not terminate.

We will present a CEGAR algorithm for the class of *RHA* which is com-

plete for the class of initialized *RHA* and semi-complete for the class of *RHA*. The termination of each iteration of the CEGAR loop requires that the four steps mentioned above terminate. Since the safety verification is undecidable for the class of *RHA*, model-checking step may not terminate if the abstract system is some *RHA*. However, as mentioned earlier, safety verification is decidable for the class of *MIRHA*, and hence we choose it to be our abstraction space. We call our algorithm a "hybrid CEGAR" (HCEGAR) method since the class of abstraction is a class of hybrid automata rather than the class of finite state systems which is typically the case. Our HCEGAR algorithm is sketched in Algorithm 3. We will next explain the different operation involved in the algorithm.

Algorithm 3 Hybrid CEGAR

Input: $\mathcal{H}_C$ - concrete *RHA*, q_C- bad location
Output: "YES", if q_C is not reachable in $\mathcal{H}_C$ and "NO" otherwise.

$\alpha :=$ initial hybrid abstraction function
while *true* **do**
  //Constructing the abstract system
  $\mathcal{H}_A :=$ *Construct_IRHA*$(\mathcal{H}_C, \alpha)$
  $q_A = \alpha(q_C)$
  //Model Checking
  $\sigma' :=$ ModelCheck$(\mathcal{H}_A, q_A)$
  if σ' is empty **then**
    //Model checking successful
    Return "YES"
  else
    //σ' is an abstract counterexample
    Reach := Validate$(\mathcal{H}_C, \alpha, \sigma)$
    if *Reach* is a not empty **then**
      //A concrete counterexample exists; hence q_C is reachable in $\mathcal{H}_C$
      Return "NO"
    else
      //Refining the hybrid abstraction function α
      $\beta :=$ Refine(α, *Reach*)
      $\alpha := \beta$
    end if
  end if
end while

5.4.2 Hybrid Abstractions

The first step in the CEGAR algorithm is to construct an abstraction of the concrete system. We define an abstraction function which specifies the elements required in the construction of the abstract hybrid system. One of the goals of abstraction is to reduce the state space by "eliminating" irrelevant information. The abstract systems in our CEGAR algorithm are constructed by merging locations and/or edges of the original system. We also allow variables to be hidden/dropped and a scaling operation on the variables. The scaling operation on a variable allows the constants appearing in the constraints to be changed such that the maximum constant to which a variable is compared to is reduced by the scaling factor, which in turn affects the complexity of analysis of the system, since it reduces the number of regions in the region graph of the system [4, 54]. In short, the scaling has the effect of increasing the "granularity" of the system. Further, since we choose as the abstract space the class of *MIRHA*, we need a method to convert a non-initialized automaton to an initialized one. More precisely, let us call an edge e and a variable index i of a *MRHA non-initialized* if it has an identity reset, that is, $\mathit{reset}(e,i) \in \mathit{MRectConst}(1)$, and $\mathit{activity}(\mathit{Source}(e)) \neq \mathit{activity}(\mathit{Target}(e))$. Let us denote by $\mathit{NonInit}_{\mathcal{H}}$ the set of all non-initialized pairs (e,i) of a *MRHA* $\mathcal{H}$. A natural way to make (e,i) initialized is to associate the strong reset $\mathit{reset}(e,i) \times \mathit{reset}(e,i)$ with the pair (e,i). In general, such a transformation might be too coarse and we may need to refine it. Hence, our abstraction function has a function which specifies for every non-initialized pair (e,i), where e is not merged, a splitting of the constraint $\mathit{reset}(e,i)$ into a multi-rectangular constraint. Each of the rectangular constraints is then transformed into a strong reset as described above.

A *hybrid abstraction function* α on $\mathcal{H}$ is a tuple $(\alpha_L, \alpha_E, \alpha_X, \alpha_P)$ where

- α_L is an equivalence relation on *Loc*;
- α_E is an equivalence relation on *Edges* satisfying the constraint that for all $e_1, e_2 \in \mathit{Edges}$, $e_1 \alpha_E e_2$ implies $\mathit{Source}(e_1)\ \alpha_L \mathit{Source}(e_2)$ and $\mathit{Target}(e_1)\ \alpha_L \mathit{Target}(e_2)$;
- $\alpha_X : [n] \to \mathbb{N} \cup \{\perp\}$ is the *scaling function*; and

- α_P : $\mathit{NonInit}_{\mathcal{H}} \cap \mathit{Singleton} \rightarrow \mathit{MRectConst}(1)$, where *Singleton* is the set of pairs (e, i) such that the equivalence classes of $\mathit{Source}(e)$, e and $\mathit{Target}(e)$ are singleton sets, is such that $\alpha_P(e, i) \preceq \mathit{reset}(e, i)$ and satisfies $[\![\alpha_P(e, i)]\!] = [\![\mathit{reset}(e, i)]\!]$.

Next we describe a specific method *Construct_IRHA* for constructing a *MIRHA*, given an *MRHA* and a hybrid abstraction function. Before that, let us define some preliminaries.

Let *Id* be a predicate on a pair (e, i), where e is an edge and i is a variable index, such that $\mathit{Id}(e, i)$ holds iff $\mathit{reset}(e, i) \in \mathit{MRectConst}(1)$, that is, $\mathit{ResetRel}(e, i)$ is subset of the identity relation. Given $M \in \mathit{MRectConst}(n)$, define $\mathit{IdToStrong}(M)$ to be the set in $\mathit{MRectConst}(2n)$ corresponding to $\{M' \times M' \,|\, M' \in M\}$. It will be used to convert an identity reset to a strong reset.

The function $\alpha_X : [n] \rightarrow \mathbb{N} \cup \{\perp\}$ represents a transformation of the continuous state space. Given a tuple $x \in \mathbb{R}^n$, the function $\tilde{\alpha_X}(x)$ gives an element y in $\mathbb{R}^k$, where k is the size of the set $\{i \,|\, \alpha_X(i) \neq \perp\}$. For $j \in [k]$, $(y)_j = (x)_i / \alpha_X(i)$, where i is the least number in $\mathbb{N}$ such that the size of the set $\{i' \leq i \,|\, \alpha_X(i') \neq \perp\}$ is j. We will abuse notation and use $\tilde{\alpha_X}(z)$, where $z = (z_1, \cdots, z_{2n})$ to represent the element $w = (w_1, \cdots, w_{2k})$ where $(w_1, \cdots, w_k) = \tilde{\alpha_X}((z_1, \cdots, z_n))$ and $(w_{k+1}, \cdots, w_{2k}) = \tilde{\alpha_X}((z_{n+1}, \cdots, z_{2n}))$. Given a set of rectangular constraints $\{R_1, \cdots, R_t\}$ of dimension n (or $2n$) and a variable abstraction function $\alpha_X : [n] \rightarrow \mathbb{N} \cup \{\perp\}$, we define a merge operation $\mathit{Merge}(\{R_1, \cdots, R_t\}, \alpha_X)$ to be the rectangular constraint obtained by taking the union of the sets, applying α_X on it and taking the rectangular hull of the resulting set, that is, $\mathit{RectHull}(\tilde{\alpha_X}(\cup_i R_i))$. So the merge operation gives the tightest rectangular constraint which contains the set obtained by applying the variable transformation on the union of the input rectangular constraints.

Let $\mathcal{H}$ be a *MRHA*, and $\alpha = (\alpha_L, \alpha_E, \alpha_X, \alpha_P)$ be a hybrid abstraction function. We will define the *MIRHA* constructed from $\mathcal{H}$ and α, denoted $\mathcal{H}' = \mathit{Construct\_IRHA}(\mathcal{H}, \alpha)$. The underlying graph of $\mathcal{H}'$ is obtained by taking the quotient graph of the underlying graph of $\mathcal{H}$ with respect to α_L and α_E, that is, the locations of $\mathcal{H}'$ are the equivalence classes of α_L and the edges of $\mathcal{H}'$ are the equivalence classes of α_E. The invariant (activity) for an abstract location $[q]_{\alpha_L}$ is obtained by merging the invariants (activities) for

the concrete locations in the equivalence class of q using the *Merge* operation. The reset associated with an edge $[e]_{\alpha_E}$ is constructed in the following way.

- If $e \in \mathit{NonInit} \cap \mathit{Singleton}$, then $\mathit{reset}'([e]_{\alpha_E}, i) = \mathit{IdToStrong}(\alpha_P(e, i))$; and if $e \in \mathit{Singleton}/\mathit{NonInit}$, then $\mathit{reset}'([e]_{\alpha_E}, i) = \mathit{reset}(e, i))$.
- Otherwise, let $[e]_{\alpha_E} = \{e_1, \cdots, e_t\}$. For $1 \leq j \leq n$, $\mathit{reset}'([e]_{\alpha_E}, j) = \{\mathit{Merge}(B_1, \cdots, B_t\}, \alpha_X)\}$, where $B_i = [\![\mathit{IdToStrong}(\mathit{reset}(e_i, j))]\!]$ if $\mathit{Id}(e_i, j)$, and $[\![\mathit{reset}(e_i, j)]\!]$ otherwise.

Example 35 *Returning to the example in Figure 5.1, the MIRHA (in fact an initialized RHA) on the right, call $\mathcal{H}_A$, is obtained from the MRHA on the right, call $\mathcal{H}_C$ by using the function Construct_IRHA. More precisely, $\mathcal{H}_A = \mathit{Construct\_IRHA}(\mathcal{H}_C, \alpha)$, where α_L merges the locations q_3 and q_4 and leaves the rest as is, α_E merges the edges e_2 and e_3 and leaves the remaining as is, and α_X maps the index 1 corresponding to the variable x to 2. One can check that $\mathcal{H}_A$ is the abstraction of $\mathcal{H}_C$ using the α defined above. For example, invariant of the location $P_{3,4}$ would be obtained by first taking the union of $[2, 4]$ and $[4, 6]$ which is $[2, 6]$, then applying the scaling function thus obtaining the set $[1, 3]$ and then taking the rectangular hull of the set. The activity of the location P_1 is obtained by applying the scaling function to the activity of q_1, thus obtaining the set $[1/2, 1]$ and taking its rectangular hull, to obtain $[0, 1]$. Merging edges e_2 and e_3 results in a reset obtained in the following way. First the reset of e_2 is replaced by the set $[2, 4] \times [2, 4]$, and is then combined with the set $[2, 4] \times [4, 6]$ corresponding to the edge e_3, resulting in the set $[2, 4] \times [2, 6]$. Then the scaling function is applied on this set giving us the set $[1, 2] \times [1, 3]$, and finally taking the rectangular hull gives us the desired set $[1, 2] \times [1, 3]$.*

Notation 36 *Given a hybrid abstraction function, its elements are denoted using appropriate subscripts; for example, the components of a hybrid abstraction function α are denoted by α_L, α_E, α_X and α_P, respectively.*

Next we will define an ordering on the hybrid abstraction functions. Let α and β be two hybrid abstraction functions of $\mathcal{H}$. Then we say that $\alpha \preceq \beta$ iff

- $\alpha_L \subseteq \beta_L$;

- $\alpha_E \subseteq \beta_E$;
- for every $i \in [n]$, $\beta_X(i) \neq \bot$ implies $\alpha_X(i) \neq \bot$ and $\alpha_X(i)$ is a factor of $\beta_X(i)$; and
- $Dom(\beta_P) \subseteq Dom(\alpha_P)$ and for every $e \in Dom(\beta_P)$, $\alpha_P(e) \preceq \beta_P(e)$.

The next lemma states that we can obtain a refinement of an abstract system by choosing a smaller hybrid abstraction function.

Proposition 37 *Let α and β be hybrid abstraction functions of a MRHA $\mathcal{H}$. If $\alpha \preceq \beta$, then $[\![\mathcal{H}]\!] \preceq [\![Construct\_IRHA(\mathcal{H}, \alpha)]\!] \preceq [\![Construct\_IRHA(\mathcal{H}, \beta)]\!]$.*

Notation 38 *From now on, we will also use α_L (α_E) to mean the function mapping a location (edge) to its equivalence class, that is, $\alpha_L(l) = [l]_{\alpha_L}$ and $\alpha_E(e) = [e]_{\alpha_E}$ for a location l and an edge e. Also, we will use α_X for the function which maps an element $x \in \mathbb{R}^n$ to $\tilde{\alpha_X}(x)$. For a set $S \subseteq States_{\mathcal{H}}$, we will abuse notation and use $\alpha(S)$ to denote $\{(\alpha_L(q), \alpha_X(x)) \,|\, (q, x) \in S\}$, and similarly use $\alpha^{-1}(S')$ to denote $\{(q, x) \,|\, (\alpha_L(q), \alpha_X(x)) \in S'\}$. When S is a singleton $\{(q, x)\}$, we write $\alpha(q, x)$ and $\alpha^{-1}(q, x)$ instead of writing $\alpha(\{(q, x)\})$ and $\alpha^{-1}(\{(q, x)\})$, respectively. We will extend the domain of α_E to $Edges \cup \mathbb{R}_{\geq 0}$; which is the same function when restricted to Edges and $\alpha_E(t) = t$ for $t \in \mathbb{R}_{\geq 0}$. (This is to have a uniform α_E for all labels of $[\![\mathcal{H}]\!]$). In general, given a tuple containing elements from $Loc \cup Edges \cup \mathbb{R}^n$, applying α or α^{-1} to the tuple means applying the appropriate functions, α_L, α_E or α_X, or their inverses, to the components.*

Let us fix a concrete *MRHA* $\mathcal{H}_C$ for the rest of the chapter. In this chapter, we focus on verifying control state reachability properties. Given a location q_C, different from $init_C$, in the concrete automaton $\mathcal{H}_C$, we want to verify if some state with location q_C is reachable.

5.4.3 Initial Abstraction and Model-Checking

The HCEGAR loop will start with an *initial abstraction* $\mathcal{H}_A$ which is obtained by a hybrid abstraction function α in which q_C and $init_C$ are in different equivalence classes of α_L. $\mathcal{H}_A = Construct\_IRHA(\mathcal{H}_C, \alpha)$ is the initial abstraction. Let $q_A = \alpha_L(q_C)$. Note that q_A is different from $init_A$. The

model-checker then checks if q_A is reachable from $init_A$ in $\mathcal{H}_A$ which is shown in Algorithm 3 using the function ModelCheck($\mathcal{H}_A, q_A$). If q_A is not reachable in $\mathcal{H}_A$, then the HCEGAR algorithm terminates with a result that the concrete automaton is safe. This follows from the fact that $[\![\mathcal{H}_C]\!] \preceq [\![\mathcal{H}_A]\!]$. On the other hand, if the model-checker determines that q_A is reachable in $\mathcal{H}_A$, then it also returns an execution σ' of $\mathcal{H}_A$ reaching the abstract bad location q_A, which is called an *abstract counterexample*. The HCEGAR algorithm continues to the validation phase where the abstract counterexample σ' is validated.

5.4.4 Validation

We will now discuss how to validate an abstract counterexample returned by the model-checker.

Let $\sigma' = (q'_0, x'_0) \xrightarrow{a'_0} (q'_1, x'_1) \xrightarrow{a'_1} (q'_2, x'_2) \xrightarrow{a'_2} \cdots \xrightarrow{a'_{l-2}} (q'_{l-1}, x'_{l-1}) \xrightarrow{a'_{l-1}} (q'_l, x'_l)$ be an abstract execution fragment. Let us denote by $Concrete_\alpha(\sigma')$, the set of concrete execution fragments corresponding σ'. $Concrete_\alpha(\sigma')$ consists of execution fragments σ of $\mathcal{H}_C$ given by $(q_0, x_0) \xrightarrow{a_0} (q_2, x_2) \xrightarrow{a_1} (q_2, x_2) \xrightarrow{a_2} \cdots \xrightarrow{a_{l-2}} (q_{l-1}, x_{l-1}) \xrightarrow{a_{l-1}} (q_l, x_l)$ such that $\alpha(q_i, x_i) = (q'_i, x'_i)$ for all i; and $\alpha_E(a_i) = a'_i$.

Let us fix an abstract counterexample $\sigma' = (q'_0, x'_0) \xrightarrow{a'_0} (q'_1, x'_1) \xrightarrow{a'_1} (q'_2, x'_2) \xrightarrow{a'_2} \cdots \xrightarrow{a'_{l-2}} (q'_{l-1}, x'_{l-1}) \xrightarrow{a'_{l-1}} (q'_l, x'_l)$ of $\mathcal{H}_A$, , i.e, it is an execution fragment of $\mathcal{H}_A$ with $q'_0 = init_A$ and $q'_l = q_A$. Let $Exec_{\mathcal{H},q}$ represent the executions of $\mathcal{H}$ ending in q. *Validation* is the process of checking if $Concrete_\alpha(\sigma') \cap Exec_{\mathcal{H},q_C}$ is non-empty. In order to determine if $Concrete_\alpha(\sigma') \cap Exec_{\mathcal{H},q_C}$ is non-empty, we perform the classical backward reachability computation. We compute the set of all initial states which have a concrete counterexample starting from them corresponding to σ'. This is computed iteratively by computing for every k, the set $reach_k$ which is the set of all states (q, x) starting from which there is an execution fragment ending in q_C which belongs to $Concrete_\alpha(\sigma'_k)$, where σ'_k is the suffix of σ' starting from (q'_k, x'_k). Note that $reach_0 \cap (\{init\} \times Cont_0)$ is then the required set.

The following proposition summarizes how the *reach* sets can be used to validate the abstract counterexample.

Proposition 39 *$Concrete_\alpha(\sigma') = \emptyset$ iff $reach_k = \emptyset$ for some $0 \leq k \leq l$ or $reach_0 \cap (\{init\} \times Cont_0) = \emptyset$.*

The validation step requires the computation of the *reach* sets. The next proposition states that the *reach* sets can be computed and they have a special form. We will call a subset of states, a multirectangular region, if the continuous part corresponding to each location is a multirectangular region, that is, $S \subseteq States_{\mathcal{H}_C}$ is in $MRectReg(n)$ if $Cont(S|_q) \in MRectReg(n)$ for every location q.

Proposition 40 *For each k, $reach_k$ can be represented as an element of $MRectReg(n)$ and $reach_k$ can be computed.*

Proof (Sketch.) Proof is by induction on $reach_k$. Note that when a'_k is an edge, then computing $reach_k$ from $reach_{k+1}$ involves the operations of union and intersection on multirectangular regions. Since multirectangular regions are closed with respect union and intersection, $reach_k$ in this case can be represented as an element of $MRectReg(n)$. When a'_k is a time t, we first need to compute the set of states from which after time t elapse we can reach a specified multirectangular region. Let us consider a rectangular region for the sake of simplicity. So with each variable we can associate an interval (projection of the set to the dimension of the variable) such that the rectangular region is exactly the set of point obtained by choosing a point in the corresponding interval for each variable. The *Pre* computation can be computed for each of the variables independently, since the activity at which each variable evolves is independent of the others. Hence, the *Pre* set is just the rectangular region defined by taking *Pre* with respect to each variable (which can be computed by considering the end-points of the intervals in the initial set and those of the activity associated with the variable), and performing appropriate intersections with the invariants. ∎

If $Concrete_\alpha(\sigma')$ is not empty, then we can conclude that the system $\mathcal{H}_C$ is unsafe. Otherwise, we proceed to the refinement step.

5.4.5 Refinement

If the abstract counterexample σ' is found to be spurious in the validation step, that is, $Concrete_\alpha(\sigma') = \emptyset$, then we use the results of the analysis during

the validation to refine the abstract system.

Definition 41 *Given a transition systems $\mathcal{T}_C$ and $\mathcal{T}_A$ such that $\mathcal{T}_C \preceq \mathcal{T}_A$, a transition system $\mathcal{T}_R$ is said to be a refinement of $\mathcal{T}_A$ with respect to $\mathcal{T}_C$, if $\mathcal{T}_C \preceq \mathcal{T}_R \preceq \mathcal{T}_A$.*

We need to construct a refinement of $[\![\mathcal{H}_A]\!]$ with respect to $[\![\mathcal{H}_C]\!]$. We want to find an *MIRHA* $\mathcal{H}_R$ which is a refinement of $\mathcal{H}_A$. As seen from Proposition 37, if we can find a β such that $\beta \preceq \alpha$, then $[\![\mathcal{H}_C]\!] \preceq [\![\mathcal{H}_R]\!] \preceq [\![\mathcal{H}_A]\!]$, where $\mathcal{H}_R = \mathit{Construct\_IRHA}(\mathcal{H}_C, \beta)$. Also, we need to ensure that $\mathcal{H}_R$ makes progress by "eliminating" the spurious counterexample σ'.

Given a *MRHA* $\mathcal{H}$, a *potential execution fragment* of $\mathcal{H}$ is a sequence $\rho = (q_0, x_0)a_0(q_1, x_1)a_1 \cdots a_{l-2}(q_{l-1}, x_{l-1})a_{l-1}(q_l, x_l)$, where $(q_i, x_i) \in \mathit{States}_{\mathcal{H}}$ for $0 \leq i \leq l$, $a_i \in \mathit{Edges} \cup \mathbb{R}_{\geq 0}$ for $1 \leq i \leq l$ and a_is alternate between the sets *Edges* and $\mathbb{R}_{\geq 0}$. Given two *MRHA*s $\mathcal{H}_1$ and $\mathcal{H}_2$, a hybrid abstraction function β, such that $\mathcal{H}_2 = \mathit{Construct\_IRHA}(\mathcal{H}_1, \beta)$, and an execution $\sigma_2 = (q_0^2, x_0^2) \xrightarrow{a_0^2} (q_1^2, x_1^2) \xrightarrow{a_1^2} \cdots \xrightarrow{a_{l-2}^2} (q_{l-1}^2, x_{l-1}^2) \xrightarrow{a_{l-1}^2} (q_1^2, x_l^2)$ of $\mathcal{H}_2$, a potential execution fragment of σ_2 in $\mathcal{H}_1$ is a potential execution fragment of $\mathcal{H}_1$, $\rho_1 = (q_0, x_0)a_0(q_1, x_1)a_1 \cdots a_{l-2}(q_{l-1}, x_{l-1})a_{l-1}(q_l, x_l)$ such that $\beta(q_i, x_i) = (q_i^2, x_i^2)$ for $0 \leq i \leq l$ and $\beta_E(a_i) = a_i^2$ for $0 \leq i < l$. Let us denote by $\mathit{Potential}_\beta(\sigma_2)$ the set of a all potential execution fragments of σ_2 in $\mathcal{H}_1$. We define a predicate $\mathit{valid}_{\mathcal{H}}$ on potential execution fragments which evaluates to true only if the potential execution fragment corresponds to an actual execution fragment. $\mathit{valid}_{\mathcal{H}_1}(\rho_1)$ is true iff $(q_0, x_0) \xrightarrow{a_0} (q_1, x_1) \xrightarrow{a_1} (q_2, x_2) \cdots (q_{l-1}, x_{l-1}) \xrightarrow{a_{l-1}} (q_l, x_l)$ is an execution fragment in $\mathcal{H}_1$. We will use $\beta(\rho_1)$ to denote the sequence obtained by applying β to each element of the sequence ρ_1.

We want to ensure that the refinement that we construct makes progress by eliminating some counterexample. Let $\mathcal{H}_C$, $\mathcal{H}_A$, α and σ' be as defined before. We will find a hybrid abstraction function β such that $\beta \preceq \alpha$ and $\mathcal{H}_R = \mathit{Construct\_IRHA}(\mathcal{H}_C, \beta)$ satisfies:

*C*1 There exists a $\rho \in \mathit{Potential}_\alpha(\sigma')$ such that $\rho \notin \mathit{Potential}_\beta(\sigma'')$ for any counterexample σ'' of $\mathcal{H}_R$.

In validating σ', we found that either reach_k is empty for some k or $\mathit{reach}_0 \cap (\{\mathit{init}\} \times \mathit{Cont}_0)$ is empty. Let us consider the case where reach_k is empty for

some k. The refinement for the other case is similar. Let $\hat{k}$ be the largest integer such that $reach_{\hat{k}}$ is empty. Let $S_k = \alpha^{-1}(q'_k, x'_k)$. Observe that $reach_k = \emptyset$ iff $Post_{\mathcal{H}_C}(S_k, {\alpha_E}^{-1}(a'_k)) \cap reach_{k+1} = \emptyset$. Let R be the set of all the potential counterexamples $\rho = (q_0, x_0)a_0(q_1, x_1)a_1(q_2, x_2)\cdots(q_{l-1}, x_{l-1})a_{l-1}(q_l,\ x_l)$ in $Potential_\alpha(\sigma')$ such that $(q_{\hat{k}}, x_{\hat{k}}) \in S_{\hat{k}}$ and $(q_{\hat{k}+1}, x_{\hat{k}+1}) \in reach_{\hat{k}+1}$. We will ensure that none of the potential counterexamples in R occur in $\mathcal{H}_R$. Note that the set R of such potential counterexamples is not empty since none of the sets S_k or $reach_{\hat{k}+1}$ is empty. We will eliminate the sequences in R by ensuring that none of the sequences in $\beta(R)$ are valid in $\mathcal{H}_R$, that is, for all $\rho \in \beta(R)$, $valid_{\mathcal{H}_R}(\rho)$ is not true. This is ensured by the following condition:

$C2$ $Post_{\mathcal{H}_R}(\beta(S_{\hat{k}}), \beta(\alpha^{-1}(a_{\hat{k}}))) \cap \beta(reach_{\hat{k}+1}) = \emptyset$.

The procedure Refine in Algorithm 3 will essentially find a $\beta \preceq \alpha$ satisfying Condition $C2$.

Proposition 42 *There exists a $\beta \preceq \alpha$ which satisfies Condition $C2$, whenever Algorithm 3 reaches the Refine statement.*

Proof (Sketch.) Consider β to be the hybrid abstraction function, where the equivalence classes of β_L and β_E are singleton sets, and α_X is the function which maps every variable index i to 1. Note that if there are non-initialized edges, then $\mathcal{H}_R = Construct\_IRHA(\mathcal{H}_C, \beta)$ is the same as $\mathcal{H}_C$ and Condition $C2$ given by

$$Post_{\mathcal{H}_R}(\beta(S_{\hat{k}}), \beta(\alpha^{-1}(a_{\hat{k}}))) \cap \beta(reach_{\hat{k}+1}) = \emptyset$$

is equivalent to

$$Post_{\mathcal{H}_C}(S_{\hat{k}}, \alpha^{-1}(a_{\hat{k}})) \cap reach_{\hat{k}+1} = \emptyset$$

which is trivially true by the choice of $\hat{k}$. This reduction will not work when there are non-initialized edges in $\mathcal{H}_C$ since any definition of α_P will replace the identity resets on the non-initialized edges with strong resets; and hence the resulting system will not be identical to $\mathcal{H}_C$. For every edge $e \in \alpha^{-1}(a_{\hat{k}})$ which is not non-initialized, Condition $C2$ holds because the corresponding constraint is similar to that in $\mathcal{H}_C$ which we know holds. For those edges $e \in \alpha^{-1}(a_{\hat{k}})$ which are non-initialized, we know that the sets

$Post_{\mathcal{H}_C}(S_{\hat{k}}, e)$ and $reach_{\hat{k}+1}$ do not intersect, and moreover are both multi-rectangular (Proposition 40). Hence, we can choose $\beta_P(e)$ such that $Post_{\mathcal{H}_R}$ does not intersect $reach_{\hat{k}+1}$. ∎

5.4.6 Termination of Hybrid CEGAR

In this section, we show that the HCEGAR algorithm is complete for the class of *MIRHA*, that is, the algorithm terminates on all inputs which are *MIRHA*. We also show that the algorithm is semi-complete for the class of *MRHA* under certain fairness conditions on the generation of counterexamples by the model-checker, where semi-completeness refers to the fact that if the system is buggy, then the HCEGAR terminates in a finite number of steps, however, if the system is safe, then HCEGAR algorithm may or may not terminate.

Completeness of Hybrid CEGAR for *MIRHA*

Theorem 43 *Algorithm 3 is complete for the class of initialized rectangular hybrid automata, that is, it terminates on all inputs where the concrete automaton is an initialized rectangular hybrid automaton.*

Proof (Sketch.) First, observe that for an *MIRHA* the set *NonInit* is an empty set. Hence the domain of α_P, namely, $NonInit \cap Singleton$ is always $\emptyset$. Let α_i be the hybrid abstraction function at the beginning of the i-th iteration of Algorithm 3. Then $\alpha_{i+1} \preceq \alpha_i$ and $\alpha_{i+1} \neq \alpha_i$ (since the fact that the transition $(q'_{\hat{k}}, x'_{\hat{k}}) \xrightarrow{a'_{\hat{k}}} (q'_{\hat{k}+1}, x'_{\hat{k}+1})$ exists in $\mathcal{H}_A$ implies that Condition $C2$ does not hold for α_i). Therefore, for every i, either $\alpha_{Li+1} \subset \alpha_{Li}$, $\alpha_{Ei} \subset \alpha_{Ei}$ or $\alpha_{Xi+1}(j) < \alpha_{Xi}(j)$ for some $j \in [n]$ (where for every $k \in \mathbb{N}$, $k < \perp$). Since the number of locations and the number of edges in $\mathcal{H}_C$ is finite, the subset relation on the equivalence classes is a well-ordering. Also, since the $<$ on $\mathbb{N} \cup \{\perp\}$ is a well-ordering and the dimension of $\mathcal{H}_C$ is finite α_X can be refined only finitely many times. Therefore the number of iterations is finite. ∎

Semi-Completeness of Hybrid CEGAR for *MRHA*

Let us call a CEGAR algorithm *semi-complete*, if it terminates on all inputs where the concrete system is buggy, however may or may not terminate if the system is safe. We note that our HCEGAR algorithm is not semi-complete in general as shown by the following example.

Example 44 *Consider a 1-dimensional MRHA with two locations p and q, where p is the initial location and q the bad location. Let us call the only variable x. The invariants of p and q are $(0,1]$ and $[0,0]$, and the activities are $[0,0]$ and $[-1,-1]$, respectively. There is only one edge e in the system which goes from p to q. And the reset associated with it is the identity reset $[0,1]$, that is, the reset relation is given by $\{(x,x) \mid x \in [0,1]\}$. The abstract system is the same as the concrete system except that the reset is replaced by $[0,1]\times[0,1]$. Consider the abstract counter-example $(p, x=1) \xrightarrow{e} (q, x=0)$. One possible refinement step is to split $[0,1]$ into $[0,1/2] \cup [1/2,1]$. Next, suppose that the counter-example $(p, x=1/2) \xrightarrow{e} (q, x=0)$, followed by the refinement step which splits $[0,1/2]$ into $[0,/1/4],[1/4,1/2]$. The refinement process could continue infinitely, where in the i-th step the abstract counter-example $(p, x=1/2^i) \xrightarrow{e} (q, x=0)$ is output by the model-checker followed by a refinement step which refines the reset $[0,1/2^i]$ into $[01/2^{i-1}],[2^{i-1},2^1]$.*

Remark 45 *In the above example, all the counter-examples chosen were of the shortest length (length 1). This also shows that even the assumption that the model-checker returns the smallest counterexample does not suffice to ensure semi-completeness of Algorithm 3.*

However, if we assume that the model-checker outputs counter-examples according to some ordering, then we can show that Algorithm 3 is semi-complete.

A potential execution fragment $\sigma = (q_0,x_0)a_0(q_1,x_1)\cdots(q_k,x_k)$ of $\mathcal{H}$ is *rational* if $x_i \in \mathbb{Q}^n$ for $0 \leq i \leq k$ and $a_i \in \mathit{Edges} \cup \mathbb{Q}$ for $0 \leq i < k$. Let δ be a function which maps an execution fragment $\rho = (q_0,x_0) \xrightarrow{a_0} (q_1,x_1)\cdots(q_k,x_k)$ to the potential execution fragment $\rho = (q_0,x_0)a_0(q_1,x_1) \cdots (q_k,x_k)$. Similarly, an execution fragment ρ is called *rational* if $\delta(\rho)$ is rational.

Then next proposition says that it suffices to search for a rational counterexample of a *MRHA*.

Proposition 46 *Given a MRHA $\mathcal{H}$ with initial state init and a bad state q, $\mathcal{H}$ has a counter-example (an execution fragment from init to q) iff it has a rational counter-example (rational execution fragment from init to q).*

Proof (Sketch.) Let ρ be a counterexample of $\mathcal{H}$. Let P be the path (sequence of locations and edges) in $\mathcal{H}$ corresponding to ρ. The question of whether there exists a counterexample corresponding to P can be translated into a linear constraint with rational constants in which the variables correspond to the time spent in each location and the values of the continuous states before and after each time transitions, and the linear constraint holds for a valuation to the variables iff the values correspond to a valid execution. For a linear constraint with rational constants, if a solution exists, then there always exists a rational solution. And the constraint corresponding to P is satisfiable since ρ is an execution corresponding to P. Therefore, there exists a rational execution corresponding to P which is also a rational counterexample of $\mathcal{H}$. ∎

We will assume that the model-checker outputs counter-examples according to an ordering on the counter-examples.

Assumptions 47 *(Fairness of counterexample generation) Let F be the set of all rational potential execution fragments of a system $\mathcal{H}$. Then, there exists an ordering $\preceq_F$ on F (depending only on F) such that*

- *$\preceq_F$ is a partial ordering on F, and*
- *for any $\sigma \in F$, there exist only finitely many σ' such that $\sigma' \preceq_F \sigma$ or σ and σ' are incomparable.*

We will assume that the model-checker outputs the counter-example σ of $\mathcal{H}$ such that $\delta(\sigma)$ is a smallest element of $\{\delta(\sigma) \mid \sigma$ is a counterexample of $\mathcal{H}\}$.

For example, one such ordering on F is a well-ordering on potential execution fragments corresponding to their size of representation. Next, we show that HCEGAR is semi-complete for the class of *RHA*.

Theorem 48 *Under Assumption 47, Algorithm 3 is semi-complete for the class of rectangular hybrid automata, that is, it terminates on all inputs where the concrete automaton has a counterexample.*

Proof (Sketch.) We need to show that Algorithm 3 will terminate if $\mathcal{H}_C$ has a counter-example. Suppose not, that is, let ρ be a counterexample in $\mathcal{H}_C$, but the HCEGAR loop does not terminate. Then there is an infinite sequence of hybrid abstraction functions $\alpha_1 \succeq \alpha_2 \succeq \alpha_3 \succeq \cdots$, where α_i is the hybrid abstraction function used in the construction the i-th abstraction. Let $\mathcal{H}_j = \mathit{Construct\_IRHA}(\mathcal{H}_C, \alpha_j)$ for $j \geq 1$. First, observe that the first 3 components of the α_is cannot change infinitely often (See Proof of Theorem 43.) Hence there exists an i such that the first 3 components are identical in all the α_js for $j \geq i$. It follows from the definition of potential execution fragments that the set of rational potential execution fragments F of $\mathcal{H}_j$ is the same for $j \geq i$. Let $\preceq_F$ be the ordering the model-checker uses to output counter-examples. Let $\rho' = \alpha_i(\rho)$. Since the first 3 components of α_i remains the same for all α_j, $j \geq i$, $\alpha_j(\rho)$ is also ρ' for all $j \geq i$. Therefore, $\sigma' = \delta(\rho')$ belongs to F (in all the iterations) and $\mathit{valid}_{\mathcal{H}_j}(\sigma')$ holds for all $j \geq i$. We will show next that in each refinement the smallest valid element becomes invalid. More precisely, let $G_j = \{\mathit{valid}_{\mathcal{H}_j}(\sigma) \,|\, \sigma \in F\}$. Then $G_{j+1} \subseteq G_j / g_j$ where g_j is a smallest element of G_j. Since there are only finitely many elements which are smaller than or incomparable with σ', there exists a j such that G_j does not contain σ', a contradiction.

It remains to show that $G_{j+1} \subseteq G_j / g_j$. The model-checker outputs the counter-example ρ'' corresponding to g_j, that is, $\delta(\rho'') = g_j$. Let $(q'_{\hat{k}}, x'_{\hat{k}}) \xrightarrow{a'_{\hat{k}}} (q'_{\hat{k}+1}, x'_{\hat{k}+1})$ be transition in ρ'' that is refined. The refinement satisfies Condition $C2$:

$$\mathit{Post}_{\mathcal{H}_{j+1}}(\alpha_{j+1}(\alpha_j^{-1}(q'_{\hat{k}}, x'_{\hat{k}})), \alpha_{j+1}(\alpha_j^{-1}(a'_{\hat{k}}))) \cap \alpha_{j+1}(\mathit{reach}_{\hat{k}+1}) = \emptyset.$$

This is equivalent to

$$\mathit{Post}_{\mathcal{H}_{j+1}}((q'_{\hat{k}}, x'_{\hat{k}}), a'_{\hat{k}}) \cap \{(q'_{\hat{k}+1}, x'_{\hat{k}+1})\} = \emptyset,$$

since α_j and α_{j+1} match in the first three component. This also implies that g_j is not valid in $\mathcal{H}_{j+1}$, since there cannot be an execution fragment corresponding to $(q'_{\hat{k}}, x'_{\hat{k}}) \xrightarrow{a'_{\hat{k}}} (q'_{\hat{k}+1}, x'_{\hat{k}+1})$. ∎

5.4.7 A Specific Refinement Algorithm

Given the concrete system $\mathcal{H}_C$ and a hybrid abstraction function α, the method below computes a hybrid abstraction function $\beta \preceq \alpha$ which satisfies Condition $C1$.

Let $\hat{k}$ be the index in the abstract counterexample as defined in the Section 5.4.5. First let us consider the case where $a'_{\hat{k}}$ is an edge. The refinement takes place in two steps:

1. In the first step, we construct γ as defined below. Let R be the set of locations l such that $\mathit{reach}_{\hat{k}+1}|_l \neq \emptyset$. Let E be the edges e in $\alpha^{-1}(a'_{\hat{k}})$ such that $\mathit{Target}_C(e) \in R$. And let S be the set of locations in $\alpha^{-1}(q'_{\hat{k}})$ such that it is the source of some edge in E. Let γ_L be the maximal relation such that $\gamma_L \subseteq \alpha_L$ and each of the locations in $R \cup S$ appear in an equivalence class which is singleton set. Let γ_E be the maximal relation such that $\gamma_E \subseteq \alpha_E$, satisfies the constraints of a hybrid abstraction function, and each of the edges in E is in an equivalence class which is a singleton set. Let γ_X be the function which maps each i to 1. For each non-initialized pair (e, i), define $\gamma_P(e, i)$ to be splitting of $\mathit{reset}(e, i)$ such that it refines α_P and the sets $(\mathit{Cont}(\mathit{reach}_{\hat{k}+1}|_{\mathit{Target}_C(e)}))_i$ and $(\mathit{Cont}(\mathit{Post}_{\mathcal{H}_C}(S_{\hat{k}}, e)|_{\mathit{Target}_C(e)}))_i$ are separated.

2. In the second step we select the variables and set the appropriate scalings for the variables. Let β be same as γ except for β_X which is defined as follows. Iterate over all possible subsets of $[n]$ starting with those with fewer elements to find a subset X for which Condition $C2$ holds. Then for every index i, define v_i to be $\perp$ if $i \notin X$, and to be the g.c.d of all the constants appearing in the rectangular constraints of the resets of the edges in E and index i, if $i \in X$. Then set $\beta_X(i)$ is the g.c.d of $\alpha_X(i)$ and v_i, where g.c.d of $\perp$ and any element x is x.

Next, let us consider the case where $a'_{\hat{k}}$ is a time t. The first part is similar except that for each non-initialized edge (e, i) set $\gamma_P(e, i) = \alpha_P(e, i)$ if $\alpha_P(e, i)$ is defined, otherwise set $\gamma_P(e, i)$ to be $\mathit{reset}(e, i)$. The only difference in the second part is instead of considering resets of edges in e, we consider invariants and activities of locations in R.

Theorem 49 *Given a concrete MRHA $\mathcal{H}_C$ and a hybrid abstraction function*

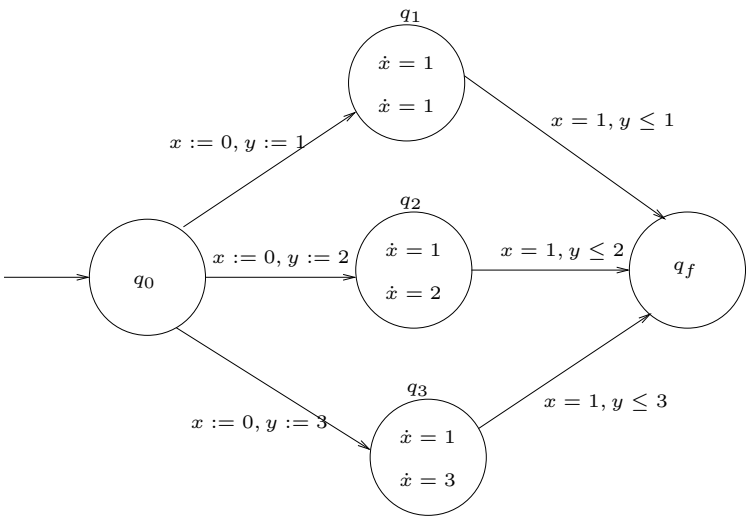

Figure 5.3: An illustrative example

α, the above method computes a β such that $\beta \preceq \alpha$ and β satisfies Condition C2.

5.4.8 Comparison with Discrete Abstraction based CEGAR

Consider the *RHA* in Figure 5.3. The expression $x := i$ on an edge is a short hand for the reset expression $x \in (-\infty, \infty) \times [i, i]$, an expression $x = i$ on an edge is a shorthand for the reset expression $x \in [i, i] \times (-\infty, \infty)$, and similarly an expression $y \leq i$ is a shorthand for the reset expression $y \in (-\infty, 1] \times (\infty, \infty)$. The invariants for all the locations is given by $x \in (-\infty, \infty), y \in (-\infty, \infty)$. It is easy to see that the location q_f is not reachable from q_0. For example, when location q_2 is entered, the value of x is 0 and y is 1. For the transition out of q_2 to be enabled x should be 1 and y should be ≤ 2. However if x is 1, then 1 time unit has been spent in location q_f. Then the value of y would by $1 + 1 * 2 = 3$. Hence, the transition out of q_2 would not have been enabled.

Assuming that the CEGAR algorithm is started with an abstraction given by the control flow graph of the automaton, predicate abstraction based refinement proceeds by model-checking the underlying control flow graph. Let us say, the path $q_0 \rightarrow q_1 \rightarrow q_f$ is returned as a counterexample. A backward analysis of this path concludes that the post of q_0 along the edge

from q_0 to q_1 given by the set $x = 0, y = 1$ has an empty intersection with the set defined by the predicate $y \leq x, x \leq 1$. To separate the two sets, a predicate $y \leq x$ could be added to the abstraction. Similarly, a analysis of the counterexample corresponding to the path $q_0 \rightarrow q_2 \rightarrow q_f$ would result in the addition of the predicate $2y \leq x$, and so on. If there are n states $q_1, \cdots, q_n$, such that the activity of the state q_i is given by $\dot{x} = 1, \dot{y} = i$, and the reset of the edge from q_i to q_f is $x = 1, y = i$, then to prove that the system is safe, n different predicates $i * y \leq x$, $1 \leq i \leq n$ could get added. This could lead to n^2 abstract states in the final abstraction and take n iterations of CEGAR to terminate.

On the other hand, our algorithm terminates with a final abstraction which is identical to the concrete automaton. To compare the number of abstract states in the final abstraction in the two approaches, let us determine the number of symbolic states explored by a symbolic reachability analysis tool such as HyTech. A forward reachability algorithm on the final abstraction (same as the concrete automaton) computes one symbolic state for each location, since a breadth-first search exploration would have at most two levels. Hence the number of symbolic states is n in this case. Alternately, observe that the automaton is an initialized *RHA*. One way to decide reachability of *IRHA* is to convert it to a timed automaton, and analyze the region graph of the timed automaton. The transformation of the above automaton to a timed automaton would result in the automaton in which all the paths $q_0 \rightarrow q_i \rightarrow q_f$ are identical to the path $q_0 \rightarrow q_1 \rightarrow q_f$. Only a constant number of regions corresponding to each location are reachable since the maximum constant appearing in the timed automaton is 1. The number of reachable regions of this graph is $O(n)$. In either case, the final abstract automaton returned by our algorithm has lesser number of states, for a large enough n. Further, our algorithm converges quickly, since the only operation performed for refining the abstraction are addition of variables and scaling of the variables.

5.5 Implementation and Experimental Results

The tool, which we call **H**ybrid **A**bstraction **R**efinement **E**ngine (HARE), implements the CEGAR algorithm in C++. HARE input consists of a hybrid automaton and an initial abstraction function to create an initial abstract

hybrid automaton. The default initial abstract automaton has no variables and has three locations—an initial location, an unsafe location, and a third location corresponding to all the other locations of the concrete automaton. This abstract automaton is automatically translated to the input language for HyTech [56] and then model-checked. If HyTech does not produce a counterexample, then HARE returns the current abstraction and the concrete hybrid automaton is inferred to be safe. Otherwise, the counterexample is parsed and validated. Consider an abstraction α with abstract counterexample $\sigma' = s'_1 \xrightarrow{a'_1} s'_2 \xrightarrow{a'_2} \ldots \xrightarrow{a'_{n-1}} s'_n$, where each s'_i is a set of states in the abstract hybrid automaton. As described in the previous section, validation proceeds backwards and involves checking that each $s'_i \xrightarrow{a'_i} s'_{i+1}$ corresponds to a reachable transition of the concrete automaton. A hybrid automaton which allows only the sequence of transitions in σ' is constructed and HyTech's Pre function is called on this automaton, starting from $\alpha^{-1}(s'_n)$, to check whether $Reach_k = \emptyset$ for some k. If such a k exists (i.e. the counter example is spurious), then the abstraction is refined as discussed in Section 5.4.5. In order to determine the set of variables v_s that are required to be added such that the projection of $Post(S_k, a_k)$ over v_s is disjoint from the projection of $Reach_{k+1}$ over v_s, a new hybrid automaton which allows the executions in $s'_k \xrightarrow{a'_k} s'_{k+1}$ is created. Thus, our implementation of the validation and the refinement steps involves construction of new hybrid automata and calls HyTech's Pre and $Post$ functions on these automata. These Pre and $Post$ are required because HyTech inherently doesn't support functions that can help in validating the counterexample. These calls to HyTech, at least in part, contribute to the relatively large time that HARE spends in the validation step for all the case studies.

5.5.1 Experimental Results

Our experimental evaluation of HARE (see Table 5.1 and Table 5.2) is based on four classes of examples:

1. BILL_n models a ball in an n-dimensional bounded reflective rectangle. The unsafe set is a particular point in the bounded rectangle.

2. NAV_n models the motion of a point robot in an $n \times n$ grid where each region in the grid is associated with a rectangular vector field. When

the robot is in a region, its motion is described by the flow equations of that region. The unsafe set is a particular set of regions. Nav_n_A and Nav_n_B represent the two different configurations of the vector fields, the initial and the unsafe regions. Nav_n_C corresponds to the model of two robots on the same the $n \times n$ grid with different initial conditions; the unsafe set being the set of states where the two robots simultaneously reach the same unsafe region.

3. SATS_n models a distributed air traffic control protocol with n aircraft presented in [75]. The model of each aircraft captures several (8 or 10) physical regions in the airspace where the aircraft can be located, such as the left holding region at 3K feet, the left approach region, the right missed-approach region, the runway, etc. The continuous evolution of the aircraft are described by rectangular dynamics within each region. An aircraft transitions from one to another region based on the rules defined by the traffic control protocol, which involves the state of this aircraft and also the other aircraft. Thus, the automata for the different aircraft communicate through shared variables. The safety property requires that the distance between any two aircraft is never less than a safety constant c. We have worked on two variants of SATS: SATS_n_S models just one side of the airspace and the full SATS_n_C has two sides.

4. ZENO is a variant of the well-known 2D bouncing ball system where the system has zeno executions.

It is clear from the above table that HARE produces relatively small abstractions: in some cases with two orders of magnitude reduction in the number of locations, and often reducing the continuous state space by one or two dimensions. In the extreme case of Nav_n_A, an abstraction with 6 discrete states is found in 4 iterations, independent of the size of the grid. This is not too surprising in hindsight because the final abstraction clearly illustrates why only a constant number of control locations can reach the unsafe region in this example, and it successfully lumps all the unreachable locations together. Yet, the total verification time is better for HyTech for Nav_n_A and Nav_n_B than HARE primarily because, as discussed earlier, HARE makes numerous calls to HyTech for validation, refinement and

Problem	Conc. size (locs, vars)	Abst. size (locs, vars)	Iterations
BILL_2_A	(6,2)	(4, 1)	1
BILL_3_A	(8,3)	(4, 1)	1
NAV_10_A	(100,2)	(6, 2)	4
NAV_15_A	(225,2)	(6, 2)	4
NAV_20_A	(400,2)	(6, 2)	4
NAV_10_B	(100,2)	(5, 1)	4
NAV_15_B	(225,2)	(5, 1)	4
NAV_20_B	(400,2)	(5, 1)	4
NAV_8_C	$(64^2,4)$	$(7^2, 4)$	5
NAV_10_C	$(100^2,4)$	$(7^2, 4)$	5
NAV_14_C	$(196^2,4)$	$(7^2, 4)$	5
SATS_3_S	$(8^3,3)$	$(5^3, 3)$	3
SATS_4_S	$(8^4,4)$	$(5^4, 4)$	3
SATS_3_C	$(10^4,4)$	$(5^4, 4)$	3
SATS_4_C	$(10^5,5)$	$(5^5, 5)$	3
ZENO_BOX	(7,2)	(5,1)	1

Table 5.1: The columns (from left) show the problem name, sizes of the concrete and final abstract hybrid automaton and number of CEGAR iterations

Problem	Conc. size (locs, vars)	Validation (sec)	Abstraction Refinement (sec)	HARE (sec)	HyTech (sec)
BILL_2_A	(6,2)	0.01	0.01	0.06	0.03
BILL_3_A	(8,3)	0.04	0.06	0.10	0.04
NAV_10_A	(100,2)	0.64	0.16	0.8	0.16
NAV_15_A	(225,2)	1.07	0.18	1.25	0.27
NAV_20_A	(400,2)	1.62	0.17	1.79	0.41
NAV_10_B	(100,2)	0.67	0.16	0.83	0.24
NAV_15_B	(225,2)	1.84	0.29	2.13	0.52
NAV_20_B	(400,2)	3.86	0.40	4.26	0.88
NAV_8_C	$(64^2,4)$	1.45	1.39	2.84	23.54
NAV_10_C	$(100^2,4)$	2.41	1.51	3.92	58.28
NAV_14_C	$(196^2,4)$	5.38	1.74	7.12	346.83
SATS_3_S	$(8^3,3)$	0.96	1.37	2.33	0.73
SATS_4_S	$(8^4,4)$	24.10	14.01	38.11	11.91
SATS_3_C	$(10^4,4)$	5.39	2.97	8.36	1.48
SATS_4_C	$(10^5,5)$	85.10	70.22	155.32	27.15
ZENO_BOX	(7,2)	0.04	0.04	0.08	—

Table 5.2: The columns (from left) show the problem name, sizes of the concrete automaton, time required for validation by HARE, time taken for verification of abstractions and refinement by HARE, total time taken by HARE and finally the time required for direct verification with HyTech

verification of abstract automaton. This is apparent as the time taken for abstraction refinement is relatively comparable to that of the time taken for direct verification by HyTech. The advantage of HARE is apparent in the case of Nav_C_*, where the system consists of several automata evolving in parallel on the $n \times n$ grid. Since the motion of each of the robots can be abstracted into a simpler automaton with less number of discrete locations, the state space of the composition of these abstract automaton is reduced dramatically (exponentially in the number of robots) and this is apparent in the differences in the running time.

The advantage of variable-hiding abstraction is apparent in ZENO (HyTech does not terminate in this case), as a subset of variables are sufficient to infer the safety of the system. We believe that in a complex hybrid automaton, with several components, adding the sufficient number of variables and abstracting the state space of hybrid automaton will yield better abstractions. All of this suggests, a direction of research, one we plan on pursuing, where the model-checker is more closely integrated with an abstraction refinement tool such as HARE.

5.6 Conclusions

In this chapter, we presented a property based abstraction refinement algorithm for analysing safety of rectangular hybrid automata which was fully automated. The novelty of our algorithm is that we consider as the abstraction space a class of hybrid automata unlike the previous approaches which typically considered finite state systems. This approach makes the different tasks in each iteration of the algorithm simpler. We have implemented the algorithm in a tool called HARE , and our experimental results suggest that this approach has the potential to scale. In particular, in many cases the tool produced safety proofs which were smaller than the size of the concrete system by a few orders. In the future, we intend to extend this approach to systems with more general continuous dynamics.

CHAPTER 6

A FRAMEWORK FOR PROVING CONVERGENCE OF DISCRETE-TIME HYBRID SYSTEMS

Stability is a property of a system which ensures that small perturbations in the initial state or input to the system result in only small changes to the future behavior or output of the system. In general, stability of a design is considered crucial for the system to be practically useful. In the design of control systems, stability is considered a fundamental property which any good design is expected to possess. In fact, often one of the objectives of feedback control is to design a control law which stabilizes an unstable system in addition to other performance optimization criteria.

Continuous dynamical systems have been studied extensively in the domain of control theory, and there exist various tools and techniques for stability analysis of these system. However, most modern day implementations of these control laws involve sensing, computation and actuation, which often include interactions between a digital computer and the physical system. This prompts the study of stability of systems with mixed discrete continuous behaviors.

In this part of the thesis, we focus on developing techniques for automated analysis of stability properties of hybrid systems. One of the main challenges in designing automated analysis techniques for stability is that existing techniques for stability analysis of purely continuous systems in control theory are not easily amenable to automation. For example, a classical technique for proving stability of continuous dynamical systems is the Lyapunov's (second) method. This requires one to identify a function called "Lyapunov function" which is a mapping from the state space to the positive reals such that the value of the function decreases along any execution of the system. Existence of such a function proves that the system is stable. Even in manual proofs, identifying the right Lyapunov functions often requires some cleverness, and in general, automating Lyapunov's method for stability analysis is possible for only certain classes of systems such as linear dynamical systems.

Our primary focus in this part of the thesis is to develop foundations for automated analysis, more precisely, approximation based analysis of stability properties. We believe that approximations are essential for designing efficient and scalable techniques for stability analysis. First we present a framework for proving "convergence" or asymptotic stability of hybrid systems which operate in discrete-steps. We provide necessary and sufficient conditions for establishing convergence of these systems. This characterization can potentially be used to develop automated verification methods for analysing convergence of simple systems. Next, we investigate a fundamental question in approximation based analysis, namely, what kinds of simplifications preserve stability properties? We observe that stability properties are not invariant under bisimulation, which is a canonical notion of equivalence with respect to various discrete-time properties. We equip bisimulation with some continuity requirements, more precisely, that of "uniform continuity" and show that stability properties are invariant under this notion. These results could potentially form the basis of automated techniques for approximation based stability analysis of complex systems.

In this chapter, we focus on the analysis of stability for discrete-time hybrid systems, namely, systems in which a transition is taken only at discrete times. We present a framework for analysing convergence or asymptotic stability of these systems. Intuitively, convergence is a property of a system which requires that executions starting close to an equilibrium point (a system state which does not change with time elapse) eventually converge to the equilibrium point.

A discrete-time hybrid system can be viewed as consisting of a finite set of operators over its statespace, corresponding to the discrete time-steps and the discrete transitions. Hence, such a system can be represented as a finite state automaton with transitions labelled by the operators. A sequence of operators given by a path of the automaton, corresponds to an execution of the system obtained by applying the operators in the sequence in order. In this chapter, we take this view of a hybrid system as a set of sequences of operators.

A closely related work is due to Tsitsiklis [102], who provides a framework for proving convergence of a sequence of operators under the assumption that each operator appears infinitely often in the sequence. The results of this chapter essentially extend Tsitsiklis' results by relaxing the above fairness

assumption on the sequences.

6.1 An Overview

Convergence or asymptotic stability is a key requirement of many concurrent and distributed systems that interact with physical processes. Roughly, a system A *converges* to a target state x^* if the state of A along infinite executions get closer and closer to x^*, with respect to some topology on the state space X, as time goes to infinity. While termination has been the defacto liveness property of interest for software systems, the more general convergence property becomes relevant for systems with both software and physical components. Examples of such systems include algorithms for mobile robots for forming a spatial pattern, synchronization of coupled oscillators, distributed control algorithms over switching networks [79](see for e.g. [58], [14], [15] and [78]). Convergence may indeed be viewed as a liveness property quantified over a (possibly infinite) sequence of shrinking predicates containing the target state.

Necessary and sufficient conditions for proving convergence of distributed systems which broadly fall under the category of *continuous consensus* have been studied extensively by control theorists for over three decades [79]. Specifically, two types of models of distributed computation have been considered. In the synchronous model, the state of the entire system $x \in X$ evolves according to some difference equation: $x_{k+1} = f(x_k)$ or differential equation $\dot{x} = f(x)$, where $f : X \to X$. Convergence conditions in this case are derived based on the eigenvalues of f. We refer the reader to [79] for a survey of the results of this type. In the asynchronous model, the evolution of the system is specified by a collection of transition functions $\{T_k\}$, where each $T_k : X \to X$, and an execution of the system is obtained by applying an infinite sequence σ of $T_k's$ to the starting state. In [102], Tsitsiklis has identified a general set of necessary and sufficient conditions for the convergence of executions that satisfy a particular fairness assumption.

Tsitsiklis' condition, informally, is as follows. He requires one to identify a collection of shrinking neighborhoods, indexed by a totally ordered index set, that converges to x^* and satisfies the following properties. First the neighborhoods are required to be invariant, i.e., for any neighborhood U,

$T_k(x) \in U$ for every $x \in U$ and every T_k. Second, for every neighborhood U, there must be a transition T_U that takes U to a strictly smaller neighborhood. Tsitsiklis shows that when such a neighborhood "system" exists, the system can be proved to converge to x^* in every execution where each transition T_k is applied *infinitely often*. Moreover, he shows that the convergence of a system also implies the existence of such a neighborhood system.

In this chapter, we generalize Tsitsiklis' observations as follows. We identify necessary and sufficient conditions for convergence under executions described by an *arbitrary* ω-regular language, instead of focusing on a particular set of executions that satisfy a specific fairness condition. While this is a philosophically natural extension of Tsitsiklis' investigations, it allows us to model a variety of asynchronous behavior, such as ordered execution of certain events, communication patterns between distributed agents over a dynamically evolving or unreliable communication network, and distributed network with nodes failing and recovering, that are not captured by Tsitsiklis' original formulation.

Our necessary and sufficient condition for convergence is remarkably similar to Tsitsiklis' condition. Let us assume that $\mathcal{A}$ is a Müller automaton that describes the set of valid executions. Once again, we require a collection of shrinking neighborhoods, indexed by a totally ordered index set, that converges to x^*. We also require these neighborhoods to be "invariant". However, since every finite sequence of operations need not be the prefix of a valid execution, our definition of invariance accounts for the state of the automaton $\mathcal{A}$. Next, like Tsitsiklis, we have a condition that ensures "progress towards" x^* is eventually made. This is captured by our insight that edges crossing "cuts" in accepting cycles of $\mathcal{A}$ are traversed infinitely often, and so for every neighborhood set U, there must be some cut that ensures progress. The proof showing that these conditions are sufficient, is very similar to Tsitsiklis' proof. To demonstrate the necessity of these condition for convergence is more challenging primarily because every finite sequence of operations need not be the prefix of a valid execution.

We conclude the chapter by demonstrating the application of the new set of conditions to prove the convergence of a simple continuous consensus algorithm. We consider a variety of scenarios ranging from a dynamically evolving communication network, to a situation where nodes in the distributed system can fail and recover.

Related Work. Tsitsiklis' result for the asynchronous model have been extended in several ways. For example, in [79] sufficient conditions have been given for proving convergence of distributed algorithms in which the communication graph of the participating agents is dynamic, but never permanently partitioned. More recently, in [74, 21] sufficient conditions for convergence have been derived for partially synchronous systems where messages may be lost or delayed by some constant but unknown time. All of these constrained executions can be modelled as ω-regular languages, and therefore the results of this chapter can be seen as a generalization of these observations.

6.2 Motivating Example

We model the behavior of a distributed system where agents starting at arbitrary positions on a line communicate based on an underlying dynamic graph to move closer to each other. In addition they can fail and join the system a finite number of times. However when they join they start at the same position in which they originally started. We show that the agents finally converge to a common point. We describe the protocol formally below.

Let $N \in \mathbb{N}$ denote the maximum number of agents that can ever be present in the system. Each agent has a unique identifier from the set $[N] = \{1, 2, \cdots, N\}$. We denote the state variable which stores the position of agent i, $i \in [N]$, by x_i, and it takes values in $\mathbb{R} \cup \{\perp\}$. For any $i \in [N]$, agent i is said to be failed if $x_i = \perp$; otherwise i is alive. We denote the collective states of all agents by vectors x, y etc.

Let $G = (V, E)$ be an undirected graph with $V = [N]$ and $E \subseteq V \times V$. Each vertex in the graph corresponds to an agent in the system. G is the underlying graph that remains fixed throughout our discussion. At a given point in the execution of the system, the actual communication graph G' is the subgraph of G restricted to the alive nodes.

Let us concentrate on a particular initial state (G, c) where $c = (c_1, \cdots, c_N)$. Let the current configuration of the system be (H, x), where $H = (V_H, E_H)$. There are three kinds of operators which can modify the configuration, namely, *join*, *fail* and *move*, and correspond to a node joining the system, a node failing and two nodes communicating to move to their average value.

We say that a configuration has reached a fixpoint if all the unfailed nodes

have the same value, that is, (H, x) is a fixpoint if for every $i, j \in [N]$, $i \neq j$, $x_i \neq \perp$ and $x_j \neq \perp$ implies $x_i = x_j$. When (H, x) is a fixpoint, for all $i, j \in [N]$, $\mathit{fail}_j((H, x)) = (H, x)$, $\mathit{join}_j((H, x)) = (H, x)$ and $\mathit{move}_{i,j}((H, x)) = (H, x)$. When (H, x) is not a fixpoint,

- $\mathit{fail}_j((H, x)) = (H', x')$ where $x'_j = \perp$ and $x'_i = x_i$ for $i \neq j$ and $H' = (V'_H, E'_H)$ where $V'_H = V_H - \{j\}$ and $E'_H = E \cap (V'_H \times V'_H)$.
- $\mathit{join}_j((H, x)) = (H', x')$ where $x'_j = c_j$ and $x'_i = x_i$ for $i \neq j$ and $H' = (V'_H, E'_H)$ where $V'_H = V_H \cup \{j\}$ and $E'_H = E \cap (V'_H \times V'_H)$.
- $\mathit{move}_{i,j}((H, x)) = (H, x')$ where $x' = x$ if either $x_i = \perp$ or $x_j = \perp$, otherwise $x'_i = x'_j = (x_i + x_j)/2$ and $x'_k = x_k$ for $k \notin \{i, j\}$.

We note that $\mathit{move}_{i,j}$ is defined only if $(i, j) \in E$, that is, communication between (i, j) is allowed. We want to show that an infinite sequence of operations converges to a point if it contains a finite number of fail_j and join_j operations and the set of edges (i, j) of E such that $\mathit{move}_{i,j}$ occurs infinitely often forms a connected graph. We will see later that this set of sequences is an ω-regular language. We will develop sufficiency conditions for proving such properties, and apply it to this example. Several generalizations of this type of consensus protocol has been presented in the literature (see for e.g. [22]).

6.3 Preliminaries

6.3.1 Directed and Undirected Graphs

A labelled directed graph (*LDG*) G is a triple (V, E, Σ), where V is a finite set of vertices, $E \subseteq V \times \Sigma \times V$ is a set of edges and Σ is a finite set of labels. Let $G = (V, E, \Sigma)$ be a *LDG*. Given $V' \subseteq V$, the restriction of G to V' is given by the *LDG* $G[V'] = (V', E', \Sigma)$ where $E' \subseteq E$ is the set $\{(u, a, v) \in E \mid u, v \in V'\}$. Given $E' \subseteq E$, $G - E' = (V, E - E', \Sigma)$. A *path* in G is a sequence of edges $e_1 \cdots e_n$ such that $e_i = (q_i, a, q_{i+1})$ for all i. We say that q_{n+1} is *reachable* from q_1. We say that G is *strongly connected* if for every $u, v \in V$, v is reachable from u. A set $V' \subseteq V$ is strongly connected

in G if $G[V]$ is strongly connected and is *maximally strongly connected* if in addition for all V'' such that $V' \subset V''$, $G[V'']$ is not strongly connected.

An undirected graph G is a pair (V, E) where $E \subseteq V \times V$ is a symmetric relation. Whenever we refer to a set of edges of an undirected graph it is assumed to be symmetric. A path in G and reachability of a vertex is defined as before. We say that a graph G is connected if every vertex is reachable from every other vertex. A cut in a connected graph G is a non-empty set of edges E such that $G - E$ is not connected. A subgraph of $G = (V, E)$ is a graph $G' = (V', E')$ such that $V' \subseteq V$ and $E' \subseteq E \cap (V' \times V')$.

6.3.2 Stability and Convergence

Let $\mathcal{X}$ be a set and $T_k : \mathcal{X} \to \mathcal{X}$ for $1 \leq k \leq K$ be a finite collection of functions ("operators"). Let $X_{fp} \subseteq \mathcal{X}$ be a set of common fixpoints, that is, $T_k(x) = x$ for all $1 \leq k \leq K$ and $x \in X_{fp}$. We will denote an infinite sequence of operators by σ. Let $\sigma = a_1 a_2 a_3 \cdots$, then $\sigma(i)$ denotes the i-th element of σ namely a_i, and $Pref(\sigma, i)$ denotes the finite sequence consisting of the first i elements, namely, $a_1 \cdots a_i$. Given an $x \in \mathcal{X}$, we denote the element obtained by applying the first n operators of σ to x in order by $\sigma(x, n)$. Formally $\sigma(x, 0) = x$, $\sigma(x, n) = \sigma(n)(\sigma(x, n-1))$, for $n \geq 1$.

Next we want to define the notion of convergence. We say that starting from x a sequence σ converges to some point in X_{fp} if it moves closer and closer to X_{fp} along σ. To make this notion precise we need to define a neighborhood system around X_{fp}.

Definition 50 *An $\mathcal{X}$-neighborhood system $\mathcal{U}$ around X_{fp} is a collection of subsets of $\mathcal{X}$ such that:*

Property 1 *$X_{fp} \subseteq U$, $\forall U \in \mathcal{U}$.*

Property 2 *For any $y \in \mathcal{X}$ such that $y \notin X_{fp}$, there exists some $U \in \mathcal{U}$ such that $y \notin U$.*

Property 3 *$\mathcal{U}$ is closed under finite intersections.*

Property 4 *$\mathcal{U}$ is closed under unions.*

We say that $\mathcal{U}$ has a *countable base* if there exists a sequence $\{U_n\}_{n=1}^{\infty}$ of elements of $\mathcal{U}$ such that for every $U \in \mathcal{U}$ there exists some n such that $U_n \subseteq U$.

We say that a sequence $\{x_n\}_{n=1}^{\infty}$ of elements of $\mathcal{X}$ *converges* to X_{fp} with respect to $\mathcal{U}$ if for every $U \in \mathcal{U}$ there exists a positive integer N such that $x_n \in U, \forall n \geq N$.

We want to converge not with respect to a single sequence but a set of sequences. Let us fix a set of operators $\Sigma = \{T_1, \cdots, T_k\}$. An infinite sequence of operators from Σ will also be called an infinite word. We will denote the set of all infinite words over Σ by Σ^{ω}. We will call a subset L of Σ^{ω} a language over Σ.

Definition 51 *Stability and convergence. Given a neighborhood system $\mathcal{U}$ and $L \subseteq \Sigma^{\omega}$, we say that L is* stable *with respect to $\mathcal{U}$ if $\forall U \in \mathcal{U}$, $\exists V \in \mathcal{U}$ such that $\forall x \in \mathcal{X}$, $\forall \sigma \in L$, if there exists $n_0 \in \mathbb{N}$ such that $\sigma(x, n_0) \in V$ then $\forall n \geq n_0$, $\sigma(x, n) \in U$.*

We say that L converges to X_{fp} with respect to $\mathcal{U}$ if $\{\sigma(x, n)\}_{n=1}^{\infty}$ converges to X_{fp} for all $x \in \mathcal{X}$ and $\sigma \in L$.

Remark 1 *Our definition of convergence is analogous to asymptotic stability used is control theory. However our definition of stability is slightly stronger than the classical notion of Lyapunov stability in that instead of requiring that for every U there exists a V such that any trajectory starting in V remains within U, we require that if a trajectory starting anywhere enters V, then we remain within U. This stronger condition is equivalent to the weaker condition, when the L we consider is a suffix closed language.*

Example 1 *Let us now try to formalize the convergence of the Example in Section 6.2. Let the agents have identifiers from $[N]$ and $G = (V, E)$ be the underlying undirected graph which changes when nodes fail and join. The state space $\mathcal{X}$ is the set of all pairs (H, x) where H is a subgraph of G restricted to the nodes i which have not failed. $\mathcal{X} = \{(H, x) \,|\, H = (V_H, E_H), V_H = \{i \,|\, x_i \neq \perp\}, E_H = E \cap (V_H \times V_H)\}$. We take X_{fp} to be the set of all fixpoints, which are configurations in which all the unfailed nodes have the same value. $X_{fp} = \{(H, x) \in \mathcal{X} \,|\, \forall i, j, (x_i \neq \perp, x_j \neq \perp) \Rightarrow x_i = x_j\}$.*

Next we need to define a neighborhood system which satisfies the properties 1, 2, 3 and 4. Before defining the neighborhood, we need to set some notation.

Let $\beta(n) = (1 - 1/(2n^3)$ *when* $n > 0$ *and* $\beta(0) = 0$. *Let* $alive(x)$ *is the size of the set* $\{i \mid x_i \neq \perp\}$. *We define a function* $f : (\mathbb{R} \cup \{\perp\})^N \to \mathbb{R}_{\geq 0}$ *given by* $f(x) = \sum_{j:x_j \neq \perp}(x_j - M)^2$, *where* $M = \frac{1}{alive(x)} \sum_{j:x_j \neq \perp} x_j$ *when* $alive(x) \neq 0$, *otherwise* $f(x) = 0$. *Let* $\mathbb{I}$ *be the set of all integers. We can now define the neighborhood as:*

$$\mathcal{U} = \{U_i\}_{i \in \mathbb{I}}, \text{where } U_i = \{(H, x) \in \mathcal{X} \mid f(x) \leq \beta(alive(x))^i\}.$$

$\mathcal{U}$ *is a neighborhood system. To see the Property 1 is satisfied, observe that for all* $(H, x) \in X_{fp}$, $f(x) = 0$ *and* $\beta(alive(x))^i \geq 0$. *Hence* $X_{fp} \subseteq U_i$ *for all* i. *Given any* $(H, x) \notin X_{fp}$, $f(x) > 0$. *And* $\beta(n)^i \to 0$ *as* $i \to \infty$ *for* $n \geq 0$. *Therefore* $(H, x) \notin U_i$ *for some* i. *In particular we have* $X_{fp} \subset \cdots \subset U_3 \subset U_2 \subset U_1 \subset U_0 \subset U_{-1} \subset U_{-2} \subset U_{-3} \subset \cdots \subset \mathcal{X}$, *and* $\bigcap_{i \in I} U_i = X_{fp}$. *Clearly Properties 3 and 4 are satisfied.*

We want to show that starting from any $(H, x) \in \mathcal{X}$, *we converge to* X_{fp} *on any sequence of operations of join, fail and move with finite number of join and fail operations and such that the moves form a connected component of the alive nodes. Let* $join = \{join_j \mid j \in [N]\}$, $fail = \{fail_j \mid j \in [N]\}$ *and* $move = \{move_{i,j} \mid (i, j) \in E\}$. *Formally* $\Sigma = join \cup fail \cup move$. *We can define a function* $Nodes-Alive : \Sigma^* \to 2^{[N]}$ *which takes a finite sequence of operators and returns the set of nodes alive after applying the operators in the sequence. We will use . for concatenation of two finite sequences or for concatenation of an infinite sequence to the end of a finite sequence.* $Nodes-Alive(\epsilon) = [N]$. $Nodes-Alive(\sigma.T) = Nodes-Alive(\sigma)$ *if* $T \in move$, $= Nodes-Alive(\sigma) - \{i\}$ *if* $T = fail_i$ *and* $= Nodes-Alive(\sigma) \cup \{i\}$ *if* $T = join_i$. $L_{converge} = \{\sigma \in \Sigma^\omega \mid \exists \sigma_1 \in \Sigma^*, \sigma_2 \in move^\omega, \sigma = \sigma_1\sigma_2, G_{\sigma_1,\sigma_2}$ *is connected*$\}$, *where* $G_{\sigma_1,\sigma_2} = (Nodes-Alive(\sigma_1), \{(i, j) \mid move_{i,j} \in \inf(\sigma_2)$ *or* $move_{j,i} \in \inf(\sigma_2)\})$.

Remark 2 *This system is not stable even in the classical sense because starting in any configuration, executing* $join_j$, $j = 1, \cdots, N$, *will result in the initial configuration. So given a* U *which is sufficiently small, for every* V, *there exists an* $x \in V$ *and* σ *such that* σ *takes* x *out of* U. *However we will prove the convergence using our sufficiency results.*

6.3.3 Muller Automata and ω-Regular Languages

A *Muller* automaton $\mathcal{A}$ over an alphabet Σ is a tuple $(Q, q_{init}, \delta, \{F_1, \cdots, F_k\})$ where:

- Q is a finite set of states.
- q_{init} is the initial state.
- $\delta \subseteq Q \times \Sigma \times Q$ is the transition relation (or the set of edges).
- $F_i \subseteq Q$ for $1 \leq i \leq k$ are accepting sets.

An automaton $\mathcal{A}$ defines a language over Σ. Given an infinite sequence sequence $\tau = \tau_1\tau_2\cdots$, we define $\inf(\tau) = \{\tau_i \mid \{j \mid \tau_j = \tau_i\}$ is an infinite set $\}$. A run of $\mathcal{A}$ on $\sigma \in \Sigma^\omega$ is an infinite sequence of states $\rho = q_1q_2\cdots$ such that $q_1 = q_{init}$ and $(q_i, \sigma(i), q_{i+1}) \in \delta$ for $i \geq 1$. A run ρ of $\mathcal{A}$ on σ is *accepting* if $\inf(\rho) = F_i$ for some i. An infinite word $\sigma \in \Sigma^\omega$ is *accepted* by $\mathcal{A}$ if there exists a run of $\mathcal{A}$ on σ which is accepting. The language accepted by $\mathcal{A}$, denoted $Lang(\mathcal{A})$ is the set of all infinite words accepted by $\mathcal{A}$. A language $L \subseteq \Sigma^\omega$ is *ω-regular* if there exists a Muller automaton whose language is L. We associate a labelled directed graph with $\mathcal{A}$ denote $Graph(\mathcal{A})$ and is defined as $Graph(\mathcal{A}) = (Q, \delta, \Sigma)$. Henceforth when we refer to a path of an automaton or a state being reachable, we refer to the underlying graph.

We call a Muller automaton $\mathcal{A} = (Q, q_{init}, \delta, \{F_1, \cdots, F_k\})$ *simple* if:

- Every state in Q is reachable from q_{init}, and every edge $e \in \delta$ is useful, that is, there exists $\sigma \in Lang(\mathcal{A})$ and an accepting run ρ of $\mathcal{A}$ on σ such that $e = (\rho(i), \sigma(i), \rho(i+1))$ for some i.
- $Graph(\mathcal{A})[F_i]$ is maximally strongly connected in $Graph(\mathcal{A})$ for all i.
- All edges going out of F_i go into F_i for all i, that is, $(q, a, q') \in \delta$ and $q \in F_i$ implies $q' \in F_i$.

The next proposition states that the class of languages accepted by simple Muller automata is exactly the class of ω-regular languages.

Proposition 52 *For every Muller automaton $\mathcal{A}$, there exists a simple Muller automaton $\mathcal{B}$ such that $Lang(\mathcal{A}) = Lang(\mathcal{B})$. Further $\mathcal{B}$ can be constructed in time polynomial in the size of $\mathcal{A}$.*

Proposition 53 *Given a simple Muller automaton $\mathcal{A}$ and a set of edges $E \subseteq \delta$ such that $(Graph(\mathcal{A}) - E)[F_i]$ is not strongly connected for every i, an accepting run ρ of $\mathcal{A}$ on any $\sigma \in \Sigma^\omega$ has infinitely many indices i such that $(\rho(i), \sigma(i), \rho(i+1)) \in E$.*

Example 1 *The language $L_{converge}$ of the Example in section 6.2 is ω-regular. It is accepted by the following automaton $\mathcal{A}_{converge} = (Q, q_{init}, \delta, \mathcal{F})$. Q consists of two types of states: the first set $Q_1 = 2^{[N]}$ stores the set of alive nodes, the second set $Q_2 = \{(S, E_S, e) \mid S \subseteq [N], (S, E_S)$ is a connected subgraph of G, $e \in E_S\}$. $q_{init} = [N]$. $\mathcal{F} = \{F_{S,E_S} \mid (S, E_S, e) \in Q_2\}$, where $F_{S,E_S} = \{(S, E_S, e) \in Q\}$. All nodes in F_{S,E_S} ensure that eventually the set of alive nodes will be S and those which will communicate infinitely often will be those in E_S. δ consists of three sets of transition:*

- $\delta_1 = \{(S, T, S \cup \{j\}) \mid S \in Q_1, T \in join\} \cup \{(S, T, S - \{j\}) \mid S \in Q_1, T \in fail\} \cup \{(S, T, S) \mid S \in Q_1, T \in move\}$.
- $\delta_2 = \{((S, E_S, e), T, (S, E_S, e')) \mid e = (i, j), T = move_{i,j}, e' \in E_S - \{e\}\}$.
- $\delta_3 = \{(S, T, (S, E_S, e)) \mid T \in move\}$.

6.3.4 $\mathcal{A}, \mathcal{X}$-Neighborhood System

For the rest of the chapter, let us fix some notation. Let $\mathcal{X}$ be a set and $\Sigma = \{T_1, \cdots, T_k\}$ a set of operators on $\mathcal{X}$. Let X_{fp} be a non-empty set of common fixpoints of $\mathcal{X}$ with respect to the operators in Σ, and $\mathcal{U}$ a neighborhood system around X_{fp}. Let $\mathcal{A}$ be a Muller automaton on Σ. Let $\mathcal{Y} = Q \times \mathcal{X}$.

We will define some concepts related to $\mathcal{Y}$. Given $Y \subseteq \mathcal{Y}$ and $q \in Q$, $Proj_q(Y) = \{x \mid (q, x) \in Y\}$ and $Proj(Y) = \cup_{q \in Q} Proj_q(Y)$. Given an edge $e = (q, T_i, q') \in \delta$, $func_e((q, x)) = (q', T_i(x))$. Given $Y \subseteq \mathcal{Y}$, $func_e(Y) = \{q'\} \times T_i(Proj_q(Y))$. A set $Y \subseteq \mathcal{Y}$ is said to be $\mathcal{A}$-invariant if for all $e \in \delta$, $func_e(Y) \subseteq Y$. We say that a state $y \in \mathcal{Y}$ is *reachable* if there exists an $x \in \mathcal{X}$, a $\sigma \in Lang(\mathcal{A})$ and a run ρ of $\mathcal{A}$ on σ such that $y = (\rho(i), \sigma(x, i-1))$ for some i. We say that y is reached from x using $w = Pref(\sigma, i-1)$. We will denote the set of all reachable states of Y by $Reachable(Y)$.

Let $Y_{fp} = Reachable(Q \times X_{fp})$. Note that Y_{fp} is an $\mathcal{A}$-invariant set. A $\mathcal{A}, \mathcal{X}$-neighborhood system around X_{fp} is a $Q \times \mathcal{X}$-neighborhood system $\mathcal{W}$

around Y_{fp}. When $\mathcal{X}$ is clear from the context we will drop the $\mathcal{X}$ and call it an $\mathcal{A}$-neighborhood system. $\mathcal{W}$ is said to be *finer* than $\mathcal{U}$, if for every $U \in \mathcal{U}$, there exists a $W \in \mathcal{W}$ such that $Proj(W) \subseteq U$. $\mathcal{U}$ is said to be *finer* than an $\mathcal{W}$, if for every $W \in \mathcal{W}$, there exists a $U \in \mathcal{U}$ such that W contains $Reachable(Q \times U)$. $\mathcal{U}$ and $\mathcal{W}$ are said to be *equivalent*, if $\mathcal{U}$ is finer than $\mathcal{W}$ and $\mathcal{W}$ is finer than $\mathcal{U}$. An $\mathcal{A}, \mathcal{X}$-neighborhood system $\mathcal{W}$ is said to be $\mathcal{A}$-*invariant* if every $W \in \mathcal{W}$ is $\mathcal{A}$-invariant.

Proposition 54 *Let $\mathcal{W}$ be an $\mathcal{A}$-neighborhood system equivalent to $\mathcal{U}$ such that every $W \in \mathcal{W}$ is a subset of $Reachable(Q \times \mathcal{X})$. Then $\mathcal{U}$ has a countable base if and only if $\mathcal{W}$ has a countable base.*

Proof Let $\{U_n\}_{n=1}^{\infty}$ be a countable base for $\mathcal{U}$. Let $W \in \mathcal{W}$. Then there exists $U \in \mathcal{U}$ such that $Reachable(Q \times U) \subseteq W$ since $\mathcal{U}$ is finer than $\mathcal{W}$. There exists $U_n \subseteq U$ by the definition of countable base. Also there exists W_n such that $W_n \subseteq Q \times U_n$ since $\mathcal{W}$ is finer than $\mathcal{U}$. Since $W_n \subseteq Reachable(Q \times \mathcal{X})$ by assumption, we have $W_n \subseteq Reachable(Q \times U_n)$. Therefore $W_n \subseteq W$. $\{W_n\}_{n=1}^{\infty}$ is a countable base for $\mathcal{W}$.

Similarly let $\{W_n\}_{n=1}^{\infty}$ be a countable base for $\mathcal{W}$. Let $U \in \mathcal{U}$. Then there exists $W \in \mathcal{W}$ such that $Proj(W) \subseteq U$, or there exists W_n such that $Proj(W_n) \subseteq U$. Also there exists $U_n \in \mathcal{U}$ such that $Reachable(Q \times U_n) \subseteq W_n$. Hence $Proj(Reachable(Q \times U_n)) \subseteq U$, or $U_n \subseteq U$. $\{U_n\}_{n=1}^{\infty}$ is a countable base for $\mathcal{U}$. ∎

6.4 Stability

From now on we will assume that $\mathcal{A}$ is a simple Muller automaton.

The following result generalizes the result of Tsitsiklis [102].

Theorem 55 *The following are equivalent.*

1. *$Lang(\mathcal{A})$ is stable with respect to an $\mathcal{X}$-neighborhood system $\mathcal{U}$ around X_{fp}.*

2. *There exists an $\mathcal{A}, \mathcal{X}$-invariant neighborhood system $\mathcal{W}$ around the set $Reachable(Q \times X_{fp})$ which is equivalent to $\mathcal{U}$.*

Proof The proof is along the lines of that given in [102]. Let $Lang(\mathcal{A}) = L$. $(1 \Rightarrow 2)$: We assume that L is stable with respect to $\mathcal{U}$ and we need to construct an $\mathcal{A}$-invariant neighborhood system $\mathcal{W}$ of subsets of $\mathcal{Y}$ which is equivalent to $\mathcal{U}$.

We do this as follows. Given $U \in \mathcal{U}$, we define W_U as the union of all $\mathcal{A}$-invariant subsets of $U' = Reachable(Q \times U)$. Note that $Y_{fp} = Reachable(Q \times X_{fp})$ is an $\mathcal{A}$-invariant subset of U', which shows that W_U is nonempty for all $U \in \mathcal{U}$. Also W_U is the largest $\mathcal{A}$-invariant subset of U'. Let $\mathcal{W}' = \{W_U : U \in \mathcal{U}\}$ and let $\mathcal{W}$ be the closure of $\mathcal{W}'$ under finite intersections and unions.

$\mathcal{W}$ is a neighborhood system. The first property is satisfied since $Q \times X_{fp} \subseteq W_U$ for all U implies $Q \times X_{fp} \subseteq W$ for all $W \in \mathcal{W}$. The sets in $\mathcal{W}$ are closed under finite intersections and arbitrary union by definition. Let $y = (q, x) \in \mathcal{Y} - Y_{fp}$. If y is not reachable then it does not belong to any W_U. Otherwise $x \notin X_{fp}$, and therefore $x \notin U$ for some $U \in \mathcal{U}$. Therefore $y \notin W_U$. Therefore for every $y \in \mathcal{Y} - Y_{fp}$, there exists $W \in \mathcal{W}$ such that $y \notin W$.

$\mathcal{W}$ is $\mathcal{A}$-invariant, since W_U are $\mathcal{A}$-invariant and finite intersections and arbitrary unions of $\mathcal{A}$-invariant sets are $\mathcal{A}$-invariant.

$\mathcal{W}$ is equivalent to $\mathcal{U}$. For every $U \in \mathcal{U}$, there exists $W_U \in \mathcal{W}$ such that $W_U \subseteq Q \times U$, hence $\mathcal{W}$ is finer than $\mathcal{U}$. To show that $\mathcal{U}$ is finer than $\mathcal{W}$, it is enough to show that $\mathcal{U}$ is finer than $\mathcal{W}'$, because the sets in $\mathcal{U}$ are also closed under finite intersections and arbitrary unions. Let $W \in \mathcal{W}'$, then $W_U = W$ for some $U \in \mathcal{U}$. Using the fact that L is *stable* $\exists V \in \mathcal{U}$ such that $\forall x \in \mathcal{X}$, $\forall \sigma \in L$, if there exists $n_0 \in \mathbb{N}$ such that $\sigma(x, n_0) \in V$ then $\forall n \geq n_0$, $\sigma(x, n) \in U$. Define $V' = \{y \mid y$ is reached from $x \in \mathcal{X}$ using $Pref(\sigma, j)$ for some $\sigma \in L$ and $\sigma(x, i) \in V$ for some $i \leq j\}$. In particular, V' contains $Reachable(Q \times V)$. Note that V' is an $\mathcal{A}$-invariant subset of U' (here we use that fact that every edge is useful). Hence $V' \subseteq W_U$. Therefore there exists $V \in \mathcal{U}$ such that $Reachable(Q \times V) \subseteq W$.

$(2 \Rightarrow 1)$: Given any $U \in \mathcal{U}$, there exists some $W \in \mathcal{W}$ such that $Proj(W) \subseteq U$, because $\mathcal{W}$ is finer than $\mathcal{U}$. Moreover, since $\mathcal{U}$ is finer than $\mathcal{W}$, there exists some $V \in \mathcal{U}$ such that $V' = Reachable(Q \times V)$ is contained in W. Consider an x, $n_0 \in \mathbb{N}$ and $\sigma \in L$ such that $\sigma(x, n_0) \in V$. Let ρ be an accepting run of $\mathcal{A}$ on σ. Then $(\rho(n_0 + 1), \sigma(x, n_0)) \in V' \subseteq W$. Since W is $\mathcal{A}$-invariant $(\rho(n+1), \sigma(x, n)) \in V'$ for all $n \geq n_0$. Therefore $\sigma(x, n) \in Proj(W) \subseteq U$ for all $n \geq n_0$. Hence L is stable with respect to $\mathcal{U}$. ∎

Remark 3 *We note that the $\mathcal{W}$ constructed in the first part of the proof is such that every $W \in \mathcal{W}$ is a subset of Reachable$(Q \times \mathcal{X})$.*

6.5 Convergence

In this section we present necessary and sufficient conditions for convergence of a ω-regular language L. Let $\mathcal{X}$, X_{fp}, Σ, $\mathcal{U}$, $\mathcal{Y}$ and Y_{fp} be as above. Let L be an ω-regular language over Σ such that $L = Lang(\mathcal{A})$, where $\mathcal{A}$ is a simple Muller automaton.

First, we present a sufficient condition for convergence.

Condition 1 *There exists a totally ordered index set I and a collection $\{X_\alpha : \alpha \in I\}$ of distinct subsets of $\mathcal{Y}$ containing Y_{fp} with the following properties:*

Property 1 *$\alpha < \beta$ implies $X_\alpha \subseteq X_\beta$.*

Property 2 *For every $U \in \mathcal{U}$, there exists some $\alpha \in I$ such that $Proj(X_\alpha) \subseteq U$.*

Property 3 *$\bigcup_{\alpha \in I} Proj_{q_{init}}(X_\alpha) = \mathcal{X}$.*

Property 4 *X_α is $\mathcal{A}$-invariant for all $\alpha \in I$.*

Property 5 *For every $\alpha \in I$ such that $X_\alpha \neq Y_{fp}$, there exists $E \subseteq \delta$ such that for every i $(Graph(\mathcal{A}) - E)[F_i]$ is not strongly connected, and for every $e \in E$, $func_e(X_\alpha) \subseteq \bigcup_{\beta<\alpha} X_\beta$.*

Property 6 *Every non-empty subset of I which is bounded below has a smallest element.*

Following theorem states that the above condition is sufficient for convergence.

Theorem 56 *If Condition 1 holds, then L converges to X_{fp} with respect to $\mathcal{U}$.*

Proof Let I, $\{X_\alpha : \alpha \in I\}$ have the properties in Condition 1. Suppose that we are given some $U \in \mathcal{U}$, $x_0 \in \mathcal{X}$ and $\sigma \in L$. We must show that $\sigma(x_0, n)$ eventually enters and remains in U.

Let us fix an accepting run $\rho = q_1 q_2 q_3 \cdots$ of $\mathcal{A}$ on σ. Let

$J = \{\alpha \in I : \exists n$ such that $(\rho(n), \sigma(x_0, n-1)) \in X_\alpha\}$.

Lemma 8 $J = I$.

Proof Since from Property 3, we have $\mathcal{X} = \bigcup_{\alpha \in I} Proj_{q_{init}}(X_\alpha)$, there exists some $\alpha \in I$ such that $(q_{init}, x_0) \in X_\alpha$. Hence J is nonempty. We consider two cases: we first assume that J is not bounded below. Then, for every $\alpha \in I$, there exists a $\beta \in J$ such that $\beta < \alpha$. Hence for every $\alpha \in I$, there exists some $\beta < \alpha$ and some integer n such that $(\rho(n), \sigma(x_0, n-1)) \in X_\beta \subseteq X_\alpha$ (Property 1). So, every $\alpha \in I$ belongs to J, and $I = J$.

Let us now assume that J is bounded below. Since it is nonempty, it has a smallest element from Property 6, denoted by β. If $X_\beta = Y_{fp}$, then β is also the smallest element of I, and $I = J$ follows. So, let us assume that $X_\beta \neq Y_{fp}$. Then from Property 5 there exists $E \subseteq \delta$ such that $(Graph(\mathcal{A}) - E)[F_i]$ is not strongly connected, and for every $e \in E$, $func_e(X_\beta) \subseteq \bigcup_{\gamma < \beta} X_\gamma$. From the definition of J, there exists some n_0 such that $(\rho(n_0), \sigma(x_0, n_0 - 1)) \in X_\beta$ and by invariance of X_β (Property 4), $(\rho(n), \sigma(x_0, n-1)) \in X_\beta$ for all $n \geq n_0$. Since $\sigma \in L$ and ρ is an accepting run, we have an $m > n_0$ such that $(\rho(m), \sigma(m), \rho(m+1)) \in E$ by Proposition 53. Since $(\rho(m), \sigma(x_0, m-1)) \in X_\beta$, we have $(\rho(m+1), \sigma(x_0, m)) \in \bigcup_{\gamma < \beta} X_\gamma$, or $(\rho(m+1), \sigma(x_0, m)) \in X_\gamma$ for some $\gamma < \beta$. Hence $\gamma \in J$, which contradicts the assumption that β was the smallest element of J. This completes the proof of the lemma. ∎

Given $U \in \mathcal{U}$, there exists some $\alpha \in I$ such that $Proj(X_\alpha) \subseteq U$ (Property 2). Since $J = I$, there exists some n_0 such that $(\rho(n_0), \sigma(x_0, n_0 - 1)) \in X_\alpha$. Since $func_e(X_\alpha) \subseteq X_\alpha$ for all e, it follows that $(\rho(n), \sigma(x_0, n-1)) \in X_\alpha$ for all $n \geq n_0$. Or $\sigma(x_0, n-1) \in Proj(X_\alpha)$ for all $n \geq n_0$. Hence $\sigma(x_0, n-1) \in U$ for all $n \geq n_0$, which completes the proof. ∎

Next we show that Condition 1 is a necessary condition when the system satisfies some additional properties.

Theorem 57 *If L is stable and converges to X_{fp} with respect to $\mathcal{U}$ and if $\mathcal{U}$ has a countable base, then Condition 1 holds.*

Proof Since L is stable with respect to $\mathcal{U}$, we have from Theorem 55 that there exists an $\mathcal{A}$-invariant neighborhood system $\mathcal{W}$ which is equivalent to $\mathcal{U}$. Since $\mathcal{U}$ has a countable base, from Proposition 54 and Remark 3, we have that $\mathcal{W}$ has a countable base as well $\{W_n\}_{n=1}^{\infty}$. Without loss of generality we

may assume that $W_{n+1} \subseteq W_n$ for all n. (Otherwise we could define a new countable base $W'_n = \bigcap_{k=0}^{n} W_k$.) Let W_0 be $Reachable(Q \times \mathcal{X})$.

Our proof consists of two main steps: for each $n \geq 0$ we construct a nested collection of subsets of $\mathcal{Y}$ which lie between W_n and W_{n+1}. Then we merge these collections to get a single nested collection.

Lemma 9 *Let $W', W'' \in \mathcal{W}$ such that $W' \subset W''$. Let Inv be the set of all $\mathcal{A}$-invariant subsets of W'' containing W'. Then there exists a function $f : Inv \to Inv$ and $g : Inv \to 2^{\delta}$ such that:*

- *For any $W \in Inv$, we have $W \subseteq f(W)$ and if $Proj_{q_{init}}(W) \neq Proj_{q_{init}}(W'')$ then $f(W) \subset W$.*
- *$func_e(f(W)) \subseteq W$ for all $e \in g(W)$.*
- *$(Graph(\mathcal{A}) - g(W))[F_i]$ is not strongly connected for all i.*

For the sake of continuity we prove continue with the proof of the theorem and prove the lemma later.

Let I_n be a well-ordered set with cardinality larger than that of $\mathcal{Y}$ and let $\alpha_{0,n}$ be its smallest element. We apply Lemma 9 with $W'' = W_n$ and $W' = W_{n+1}$ to obtain a function f_n satisfying the properties of the lemma above. We define a function $h_n : I_n \to Inv$ using the following transfinite recursion: $h_n(\alpha_{0,n}) = W_{n+1}$, and for all $\alpha > \alpha_{0,n}$, $h_n(\alpha) = f_n(\bigcup_{\beta<\alpha} h_n(\beta))$.

Notice that $W_{n+1} \subseteq h_n(\beta) \subseteq h_n(\alpha) \subseteq W_n$, for any α, β such that $\alpha > \beta$, and that if $Proj_{q_{init}}(h_n(\beta)) \neq Proj_{q_{init}}(W_n)$, then $h_n(\beta) \subset h_n(\alpha)$. Since I_n has cardinality larger than that of $\mathcal{Y}$, there exists some $\alpha \in I_n$ such that $Proj_{q_{init}}(h_n(\alpha)) = Proj_{q_{init}}(W_n)$. Let $\bar{\alpha}_n$ be the smallest such α and let $\bar{I}_n = \{\alpha \in I_n \,|\, \alpha < \bar{\alpha}_n\}$.

We now define $I = \{\top\} \cup \{(n, \alpha) \,|\, \alpha \in \bar{I}_n, n = 0, 1, \cdots\}$ with the following total order: $(n, \alpha) < (m, \beta)$ if either $n > m$ or $n = m$ and $\alpha < \beta$, and $(n, \alpha) < \top$ for all n and α. Finally, let $X_{(n,\alpha)} = h_n(\alpha)$ and $X_{\top} = W_0$.

We claim that the collection $\{X_\alpha \,|\, \alpha \in I\}$ satisfies all the properties of Condition 1. Property 1 is satisfied because $h_n(\beta) \subset h_n(\alpha)$ for every $\beta < \alpha$, where $\beta, \alpha \in \bar{I}_n$. Property 2 follows from the fact that our X_αs include the countable base $\{W_n\}_{n=1}^{\infty}$ we started with and $\mathcal{W}$ is equivalent to $\mathcal{U}$. Since $Proj_{q_{init}}(W_0) = \mathcal{X}$, Property 3 is true. Property 4 holds since all the new sets we introduce (basically in Lemma 9) are $\mathcal{A}$-invariant. Again Property

5 follows from Lemma 9, where the function g gives the set of edges E for every invariant set. Finally, since every non-empty subset of a well-ordered set has a least element, and countable concatenations of well-ordered sets is well ordered, we satisfy 6. ∎

6.5.1 Proof of Lemma 9

Given $Y \subseteq \mathcal{Y}$ and $e = (q, T, q')$, define $Reach_e(Y) = \{(q', x') \mid \exists (q, x) \in Y, x' = T(x)\}$. Given a path P in $Graph(\mathcal{A})$, $Reach_P(S)$ is defined inductively as follows. If $P = e \in \delta$, then $Reach_P(Y) = Reach_e(Y)$. Otherwise if $P = P'e$, and $Reach_P(Y) = Reach_e(Reach_{P'}(Y))$. Given an edge $e \in \delta$, $PathsEnd(e) = \{P \mid P = P'e\}$, is the set of all paths ending in e.

Given a set of edges $E \subseteq \delta$ and $W \in \mathcal{W}$, define $f_E(W) = \{(q, x) \mid \forall e \in E, P \in PathsEnd(e), Reach_P(\{(q, x)\}) \subseteq W\}$. $f_E(W)$ has the following properties.

- $W \subseteq f_E(W)$: Since W is invariant, for all P $Reach_P(W) \subseteq W$.
- $f_E(W)$ is $\mathcal{A}$-invariant: Let $x \in f_E(W)$, and $Y = f_E(\{x\})$. If there exists $y \in Y$ such that $y \notin f_E(W)$, then there exists $e \in E$ and $P \in PathsEnd(e)$ such that $Reach_P(\{y\}) \not\subseteq W$. Then $x \notin f_E(W)$, since there exists $e \in E$ and a path $e'P \in PathsEnd(e)$ such that $Reach_P(\{x\}) \not\subseteq W$.
- $func_e(f_E(W)) \subseteq W$ for all $e \in E$: The argument is similar to the previous.

Let $\mathcal{E} = \{E \subseteq (\delta \cap \cup_i (F_i \times \Sigma \times F_i)) \mid (Graph(\mathcal{A}) - E)[F_i]$ is not strongly connected for every $i\}$. Given $W \in Inv$, we claim that if $Proj_{q_{init}}(W) \neq Proj_{q_{init}}(W'')$, then $W \subset f_E(W)$ for some $E \in \mathcal{E}$. Suppose not. Then there exists $(q_{init}, x_0) \in W'' - W$ and $f_E(W) = W$ for all $E \in \mathcal{E}$. Therefore (q_{init}, x_0) does not belong to any $f_E(W)$.

We will construct a $\sigma \in Lang(\mathcal{A})$ and an accepting run ρ of $\mathcal{A}$ on σ such that $(\rho(i), \sigma(x_0, i-1)) \notin f_E(W)$ for all $E \in \mathcal{E}$ and $i \geq 1$. This contradicts the convergence of L as follows. There exists $U \in \mathcal{U}$ such that $Z = Reach(Q \times U) \subseteq W'$, since $\mathcal{U}$ is finer than $\mathcal{W}$. Since $Z \subseteq W' \subseteq W = f_E(W)$, $(\rho(i), \sigma(x_0, i-1)) \notin Z$ for all i. Therefore $\sigma(x_0, i) \notin U$ for all i, contradicting

the convergence of L with respect to $\mathcal{U}$. Hence $W \subset f_E(W)$ for some $E \in \mathcal{E}$. We set $f(W) = f_E(W)$ for some E for which $W \subset f_E(W)$.

It remains to construct a $\sigma \in L$ and ρ which satisfy the above condition. Given a path P starting in q and a singleton set $\{(q, x)\}$, $Reach_P(\{(q, x)\})$ is a singleton. Hence we will write this as just $Reach_P((q, x))$.

Let $E \in \mathcal{E}$ be non-empty. Let $s_0 = (q_{init}, x_0)$. Since $s_0 \notin f_E(W)$ there exists some P ending in an edge in E such that $Reach_P(s_0) \notin f_E(W) = W$. Let us call this P as P_1 and $Reach_P((q_{init}, x_0))$ as s_1. Let the last of edge of P belong to $F_{i^*} \times \Sigma \times F_{i^*}$. Since the automaton is simple any path starting from F_{i^*} will remain within F_{i^*}. We will assume $|F_{i^*}| \geq 2$, (a similar procedure can be used when $|F_{i^*}| = 1$.

The following procedure generates a sequence of P_js:

1. Let P_1 and s_1 be as defined above. Initialize j to 1.
2. Let $Q' = \emptyset$.
3. While $Q' \neq Q$ do:
   - Add the last state of P_j to Q'.
   - Consider $E = \delta \cap Q' \times \Sigma \times (Q - Q')$.
   - Increment j.
   - Set P_j to be a path ending in E such that $Reach_{P_j}(s_{j-1}) \notin f_E(W) = W$.
   - Set $s_j = Reach_{P_j}(s_{j-1})$.

Note that we maintain the invariant that $s_{j-1} \notin f_E(W)$ for some E and hence $s_{j-1} \notin W$. Therefore there always exists a path P_j ending in E such that $Reach_{P_j}(s_{j-1}) \notin f_E(W) = W$. Let $P' = P_1 P_2 \cdots$ be the sequence of edges $e_1 e_2 \cdots$ with $e_i = (q_i, a_i, q_{i+1})$. Define $\sigma = a_1 a_2 \cdots$ and $\rho = q_1 q_2 \cdots$. ρ is a run of $\mathcal{A}$ on σ. It is accepting because each P_j contains every state from F_{i^*} at least once, and only contains states from F_{i^*}, because the automaton is simple. We have infinitely many i such that $(\rho(i), \sigma(x_0, i-1)) \notin f_{\mathcal{E}}(W)$ for any E or equivalently $(\rho(i), \sigma(x_0, i-1)) \notin W$ (They correspond to s_js). Since W is invariant, if $(\rho(i), \sigma(x_0, i-1)) \in W$ for some i, then $(\rho(j), \sigma(x_0, j-1)) \in W$ for all $j \geq i$, which contradicts the previous statement. Therefore $(\rho(i), \sigma(x_0, i-1)) \notin W$ for all i.

6.6 An Application

In this section, we illustrate the application of our results to prove convergence of the Example in Section 6.2.

We have already defined $\mathcal{X}, X_{fp}, \mathcal{U}$ and $L_{converge}$. We will prove convergence using the simple Muller Automaton $\mathcal{A}_{converge}$. We will then point out how one can prove convergence given any simple Muller automaton for $L_{converge}$.

6.6.1 Properties of the Neighborhood System $\mathcal{U}$

Let us define $move_{i,j}(x) = x'$ as follows: If $x_i \neq \perp, x_j \neq \perp$, then $x'_i = x'_j = (x_i + x_j)/2$ and $x'_k = x_k$ for $k \notin \{i, j\}$, otherwise $x' = x$.

We recall the following two results from [74].

Proposition 58 $f(move_{i,j}(x)) \leq f(x)$.

Let $Sorted(x)$ from $[N] \rightarrow [N]$ be a one-one and onto function which satisfies for $i < j$, $x_{Sorted(x)(i)} < x_{Sorted(x)(j)}$, or $x_{Sorted(x)(i)} \leq x_{Sorted(x)(j)}$ and $Sorted(x)(i) < Sorted(x)(j)$. $Sorted(x)(i)$ will give the identifier of the agent with the i-th smallest value and when agents have same values, the value of the agent with the smaller identifier is considered smaller. Here $\perp$ is considered to have a value of ∞.

Proposition 59 *Given any x, there exists $k \in [N]$ such that $x_{Sorted(x)(k+1)} - x_{Sorted(x)(k)} > \frac{1}{alive(x)}\sqrt{\frac{f(x)}{alive(x)}}$. Given any i, j such that $1 \leq i \leq k$ and $k+1 \leq j \leq alive(x)$, $f(move_{i',j'}(x)) \leq \beta(alive(x))f(x)$, where $i' = Sorted(x)(i)$ and $j' = Sorted(x)(j)$.*

The above property says that if we start at some (H, x) in U_i, then there is partition of the nodes in H, such that for all edges (i, j) which go between the partitions, $move_{i,j}(H, x)$ will be in U_{i+1}. But we need more, we need one such partition which will work for all elements of U_i.

Define $Cut(x, k) = (A, B)$ where $A = \{Sorted(x)(i) \mid i \leq k\}$ and $B = \{Sorted(x)(j) \mid alive(x) \geq j \geq k + 1\}$. Define $Gap(x) = \{Cut(x, k) \mid 1 \leq k < alive(x), x_{Sorted(x)(k+1)} - x_{Sorted(x)(k)} > \frac{1}{alive(x)}\sqrt{\frac{f(x)}{alive(x)}}\}$.

Proposition 60 *For all $i, j \in [N]$ and x such that for all $(A, B) \in Gap(x)$, either $i, j \leq A$ or $i, j \geq B$, we have $Gap(x) \subseteq Gap(move_{i,j}(x))$.*

Let $\{Gap(x) \,|\, (H,x) \in U_i - U_{i+1}\} = \{C_1, \cdots, C_{i_n}\}$ such that if $C_i \subset C_j$ then $i < j$. Between U_i and U_{i+1} we define a finite number of sets as follows. $U_{i,0} = U_i$, $U_{i,j+1} = U_{i,j} - \{(H,x) \in \mathcal{X} \,|\, Gap(x) = C_{j+1}\}$ for $0 \leq j \leq i_n - 1$. We define the index set to be $J = \{(i,j) \,|\, 0 \leq j \leq i_n - 2\}$ with the ordering $(i,j) < (i',j')$ if $i > i'$ or $i = i'$ and $j > j'$. The required sets are $\{U_\alpha \,|\, \alpha \in J\}$. This set has the following property.

Proposition 61 *For all $\alpha \in J$, there exists $C \subseteq [N] \times [N]$ such that for all $(i,j) \in C$ and $(H,x) \in U_\alpha$, C is a cut in H and $move_{i,j}(H,x) \in U_\beta$ for some $\beta < \alpha$. Also, for all $\alpha \in J$, $i,j \in [N]$, $(H,x) \in U_\alpha$, we have $move_{i,j}((H,x)) \in U_\alpha$.*

Proof Given α, $U_\alpha = U_{m,n}$ for some m,n. Let $Gap(x) = Z$ which is the same for any $(H,x) \in U_\alpha - U_{\alpha+1}$ where $\alpha + 1 = m, n+1$ if $(m, n+1) \in J$, otherwise $\alpha + 1 = m+1, n$. The required cut $C = \{(i,j) \,|\, \exists (A,B) \in Z, i \in A, j \in B\}$. Then from Proposition 59, we have for all $(i,j) \in C$, $move_{i,j}((H,x)) \in U_{m+1,n}$ for all $(H,x) \in U_\alpha - U_{\alpha+1}$.

Given $(H,x) \in U_\alpha$, if $(i,j) \in C$, $move_{i,j}(H,x) \in U_\alpha$ from above. If $(i,j) \notin C$, then from Proposition 58 we have $move_{i,j}(H,x) \in U_{m',n'}$ where $m' \geq m$ and from 60, we have $n' \geq n$. Therefore $move_{i,j}((H,x)) \in U'_\alpha$ for some $\alpha' \leq \alpha$, hence also in U_α. ∎

6.6.2 Convergence Proof

We are now ready to define the invariant sets required to prove convergence.

Recall $\mathcal{A}_{converge} = (Q, q_{init}, \delta, \mathcal{F})$ with $Q = Q_1 \cup Q_2$ and $\delta = \delta_1 \cup \delta_2 \cup \delta_3$ and $\mathcal{F} = \{func_{S,E_S} \,|\, (S, E_S, e) \in Q_2\}$. Let $\mathcal{Y} = Q \times \mathcal{X}$, $Y_{fp} = Reachable(Q \times X_{fp})$. The index set $I = J \cup \{\top\}$, with $\top > j$ for all $j \in J$. For $\alpha \in J$, $Y_\alpha = (Q_2 \times X_\alpha) \cup Y_{fp}$, and $Y_\top = \mathcal{Y}$. The index set I with the sets $\{Y_\alpha \,|\, \alpha \in I\}$ satisfy all the properties of Condition 1. It is easy to see that Properties 1, 2, 3 and 6 are satisfied. $Y_\top$ is clearly invariant. For $\alpha \in J$ and any edge e not in δ_2, $func_e(Y_\alpha) = \emptyset \subseteq Y_\alpha$. For $\alpha \in J$ and $e = (q, a, q') \in \delta_2$, $a = move_{i,j}$ for some i,j and $func_e(Y_\alpha) \subseteq (\{q'\} \times move_{i,j}(X_\alpha)) \cup (Q \times X_{fp})$. Since $move_{i,j}(X_\alpha) \subseteq X_\alpha$ from Proposition 61, we have that Y_α is invariant. Now we show that Property 5 also holds. For $Y_\top$, we can choose E to be δ_2. $(Graph(\mathcal{A}) - E)[F_i]$ is not strongly connected for every i. We need to

show that for all $(q,(H,x)) \in Y_\top$, for all $e \in E$, $\mathit{func}_e((q,(H,x))) \in Y_\alpha$ for some $\alpha \in J$. We need to consider only $(q,(H,x)) \in Y_\top - \bigcup_{\alpha\in J} Y_\alpha$. But then $q \in Q_1$ and hence $\mathit{func}_e((q,(H,x))) = \emptyset$. For any other Y_α, we can choose $E = \{(q, \mathit{move}_{i,j}, q') \,|\, (i,j) \text{ or } (j,i) \in C\}$, where C is the cut associated with X_α given by Proposition 61. It is easy to see that $(\mathit{Graph}(\mathcal{A}) - E)[F_i]$ is not strongly connected since the labels of the edges in each F_i correspond to a connected subgraph on the unfailed nodes, and C is cut in the induced subgraph of G with unfailed nodes.

Since the system is not stable as mentioned before we cannot use Theorem 56 to guarantee existence of level sets to prove convergence of the system for any arbitrary automaton accepting L_{converge}. However we can always find such sets because of the following structure of any simple Muller automaton $\mathcal{A} = (Q, q_{\mathit{init}}, \delta, \Sigma, \{F_1, \cdots, F_k\})$ such that $\mathit{Lang}(\mathcal{A}) = L_{\mathit{converge}}$.

- There is no edge labelled by *join* or *fail* in any of the F_i, that is, there is no $a \in \mathit{join} \cup \mathit{fail}$ and $q, q' \in F_i$ for some i, such that $(q,a,q') \in \delta$. Because then we would have an accepting run with infinite *join* or *fail* operations.

- Let C be a cut of $G = (V,E)$, i.e, $G - C$ is not connected. Then removing the edges in $\mathcal{A}$ labelled by elements in C renders every F_i not strongly connected, i.e., for every i $(\mathit{Graph}(\mathcal{A}) - E')[F_i]$ is not strongly connected, where $E' = \{(q,a,q') \,|\, a = \mathit{move}_{i,j}, (i,j) \in C, q, q' \in \bigcup_j F_j\}$.

6.7 Conclusions

In this chapter, we presented a framework for proving convergence or asymptotic stability of a discrete-time hybrid system by viewing it as a set of sequence of opertors forming an ω-regular language. We essentially generalized Tsitsiklis's [102] result, which provided necessary and sufficient conditions for proving convergence of a sequence of operators in which each operator occurs infinitely often, to obtain necessary and sufficient conditions for systems where the infinite sequence of operators is a member of an arbitrary omega regular language. This enables us to apply our theory to distributed systems with changing communication topology, node failures and joins. We illustrated an application of the new set of conditions in verifying the con-

vergence of a simple (continuous) consensus protocol. Some of the interesting future directions are to investigate automated methods for generation of the invariants required in the proof of convergence; and extending the framework to the continuous time semantics of hybrid systems.

CHAPTER 7

PRE-ORDERS FOR REASONING ABOUT STABILITY

Pre-orders between processes, like simulation, have played a central role in the verification and analysis of systems. Logical characterization of such pre-orders have allowed one to verify the correctness of a system by analyzing an abstraction of the system. In this chapter, we investigate whether this approach can be feasibly applied to reason about stability properties of a system.

Stability is an important property of hybrid and dynamical systems, which requires that small perturbations from the initial state of a stable trajectory, result in system behaviors that are close to the stable behavior. We first show that stability is not invariant under the classical notion of bisimulation. We then present the notion of uniformly continuous simulations — namely, simulation with some additional continuity conditions — that can be used to reason about stability. Finally, we show that uniformly continuous simulations are widely prevalent, by recasting many classical results on proving stability of dynamical and hybrid systems as establishing the existence of a simple, obviously stable system that simulates the desired system through uniformly continuous simulations.

7.1 An Overview

Stability is a fundamental property expected from any well-designed continuous or hybrid system. Stability is not just a design goal, but is often the principal requirement, so much so that unstable systems are deemed "unusable". Intuitively, stability requires that when a system is started somewhere close to its desired operating behavior, it will stay close to that desired operating behavior at all times. For example, you would expect the controlled behavior of a robot to depend gracefully on small variations to its starting

orientation; more precisely, given any starting orientation there should be some (open) neighborhood of this orientation for which all trajectories that start in this neighborhood remain close, and furthermore, it should be possible to ensure that the trajectories are as close as desired by making the neighborhood sufficiently small.

In the discrete world, bisimulation [72] is the canonical congruence that is used to understand when two systems are intended to be equivalent. It is taken to be the finest behavioral congruence that one would like to impose, and correctness specifications are often invariant under bisimulation, i.e., if two systems are bisimilar then either both satisfy the property or neither one does. Given the efficiency of computing bisimulation quotients, bisimulation is often the basis of minimizing transition systems [68]. Variants of bisimulation, such as simulation are often used to *abstract* a system, and construct a simpler system that ignores some of the details of the system that maybe irrelevant to the satisfaction of the specification. Simulation and abstraction form the basis of verifying infinite state systems [28, 3].

However, stability is not bisimulation invariant. To see this, consider a standard dynamical system[1] D_1 with two state variables x, y taking values in $\mathbb{R}$, with the set of initial states being $\{(0, y) \mid y \in \mathbb{R}_{\geq 0}\}$. The dynamics of D_1 is given by a function $f((x, y), t)$ which describes the state at time t provided the state at time 0 was (x, y); let us take $f((0, y), t) = (t, y)$. Observe that such a system is stable with respect to the trajectory $\tau = [t \mapsto (t, 0)]_{t \in \mathbb{R}_{\geq 0}}$, as executions that start close to $(0, 0)$ remain close to τ at all times. Let us consider another dynamical system D_2 that has the same state space, and same initial states, but whose dynamics is given by the function $g((0, y), t) = (t, y(1+t))$. Observe that this system is not stable with respect to trajectory τ. On the other hand, the relation $R = \{((x_1, y_1), (x_2, y_2)) \mid x_2 = x_1 \text{ and } y_2 = y_1(1 + x_1)\}$ is a bisimulation between the systems D_1 and D_2.

Given these observations, the goal of this chapter is to identify the appropriate congruences and pre-orders to reason about stability. The stability requirement that small perturbations in initial state don't lead to large deviations from expected behavior, suggest that we need to enhance the classical notions of bisimulation (and simulation) with some continuity requirements on the witnessing bisimulation (or simulation) relation. However, identify-

[1] A standard dynamical system, in this chapter, refers to a hybrid system without any discrete transitions.

ing the right continuity assumptions turns out to be subtle. Consider once again the dynamical systems D_1 and D_2, and the bisimulation relation R, described in the previous paragraph. Both the relation R and its inverse R^{-1} are *upper semi-continuous* [59], and yet that does not ensure the preservation of stability. We observe that what is instead required is *uniform upper semi-continuity* of R and R^{-1}; we call such a congruence *uniformly continuous bisimulation*. We also introduce the notion of *uniformly continuous simulation*, which is a simulation relation R such that R and R^{-1} are uniformly upper semi-continuous. We show that stability is invariant under the new notion of bisimulation. Moreover we show that uniformly continuous simulations yield the right notion of abstraction for stability — if D_1 is uniformly simulated by D_2 and D_2 is stable then D_1 is also stable — yielding a mechanism for reasoning about stability.

Having established that uniformly continuous simulations and bisimulations are appropriate for reasoning about stability, we ask whether they arise naturally in practice. To substantiate the usefulness claim of the new relations, we investigate a number of classical results in control theory and hybrid systems, and show that the new pre-orders are widely prevalent and form the basis of stability proofs. The Hartman-Grobman theorem is an important result that says that the behavior of any dynamical system near a hyperbolic equilibrium point is qualitatively the same as the behavior of a linear system near the same equilibrium point. We observe that, in fact, the Hartman-Grobman theorem establishes that there is a uniformly continuous bisimulation between the dynamical system and its linearization.

Next we look at various results for establishing stability of dynamical and hybrid systems. The most general and useful result for establishing stability of dynamical systems is due to Aleksandr Lyapunov [63], which requires finding a (Lyapunov) function from the state space of the dynamical system to $\mathbb{R}$ that is positive definite, and decreases along every behavior of the dynamical system. We observe that a Lyapunov function constructs a dynamical system for which stability is simple to prove. Moreover, the properties of a Lyapunov function ensure that it is a uniformly continuous simulation from the original dynamical system to the one induced by the Lyapunov function. Thus, the proof of Lyapunov's theorem can be seen as constructing a simpler system which uniformly continuously simulates the original system and proving the stability of this simpler system. We also consider a technique for

establishing the stability of a hybrid system using multiple Lyapunov functions. Once again we demonstrate that the result can be recast as saying that the existence of multiple Lyapunov functions of certain kind imply that the dynamical system can be abstracted (via uniformly continuous simulations) into a system that for which stability can be proved easily, and therefore conclude the stability of the original hybrid system.

Related Work We briefly discuss some of the pre-orders that have been used in the analysis of hybrid systems. Bisimulation relations have been widely used in safety verification of hybrid systems, and are the main technical tools in proving decidability of several subclasses of hybrid systems including timed and o-minimal systems [4, 65, 18]. The notion of approximate simulations and bisimulations have been introduced and used for state space reduction of continuous and hybrid systems [47, 48, 49] for safety verification.

With respect to stability verification, the notion of of bi-continuous bisimulations [30] have been shown to preserve certain stability properties. The stability of the system is considered with respect to a single or a set of equilibrium point. Also, they do not seem to address "asymptotic" stability properties. In comparison, we consider stability of trajectories, which is a more general notion than stability of equilibrium points; and we show that bi-continuous bisimulations do not suffice to preserve stability of trajectories, and introduce the notion of uniformly continuous bisimulations under which stability of trajectories is invariant. Further, our notion preserves both Lyapunov and asymptotic stability properties.

7.2 Preliminaries

Notation Let $\mathbb{N}$ denote the set of all natural numbers $\{0, 1, 2, \cdots\}$, and let $[n]$ denote the first n natural numbers, that is, $[n] = \{0, 1, 2, \cdots, n-1\}$. A sequence σ is a function whose domain is either $[n]$ for some $n \in \mathbb{N}$ or the set of natural numbers $\mathbb{N}$. Let *SeqDom* denote the domain of sequences, that is, the set $\{[n] \,|\, n \in \mathbb{N}\} \cup \{\mathbb{N}\}$. Given a sequence σ, length of σ, denoted $|\sigma|$, is n if $Dom(\sigma) = [n]$ or ∞ otherwise. Given a sequence $\sigma : \mathbb{N} \to \mathbb{R}$, $\sum_{i=1}^{\infty} \sigma(i) = r$ for some $r \in \mathbb{R}$ if for every $\epsilon > 0$ in $\mathbb{R}$, there exists a $n \in \mathbb{N}$ such that for all $j > n$, $\sum_{i=1}^{j} \sigma(i) \in [r - \epsilon, r + \epsilon]$. Also, $\sum_{i=1}^{\infty} \sigma(i) = \infty$, if for every k, there

exists an $n > 0$ such that for all $j > n$, $\sum_{i=1}^{j} \sigma(i) \geq k$. We denote the suffix of σ starting from the i-th element by $\sigma^{i\rightarrow}$.

Set Valued Functions We consider set valued functions and define continuity of these functions. Let us fix a metric space (M, d) for the rest of this section. We choose not to treat set valued functions as single valued functions whose co-domain is a power set, since as argued in [59], it leads to strong notions of continuity, which is not satisfied by many functions. A *set valued function* $F : A \rightsquigarrow B$ is a function which maps every element of A to a set of elements in B. Given a set $A' \subseteq A$, $F(A')$ will denote the set $\bigcup_{a \in A'} F(a)$. Given a binary relation $R \subseteq A \times B$, we use R also to denote the set valued function $R : A \rightsquigarrow B$ given by $R(x) = \{y \,|\, (x, y) \in R\}$. Let $F : A \rightsquigarrow B$ be a set valued function, where A and B are extended metric spaces. We define upper semi-continuity of F which is a generalization of the "$\forall \epsilon, \exists \delta$ - definition" of continuity for single valued functions [59]. The function $F : A \rightsquigarrow B$ is said to be *upper semi-continuous* at $x \in Dom(F)$ if and only if for every neighborhood $\mathcal{U}$ of $F(x)$,

$$\exists \delta > 0 \,\text{such that}\, F(B_\delta(x)) \subseteq \mathcal{U}.$$

F is upper semi-continuous if F is upper semi-continuous at every $x \in Dom(F)$. Further, the function $F : A \rightsquigarrow B$ is said to be *uniformly upper semi-continuous* if and only if

$$\forall \epsilon, \exists \delta, \,\text{such that}\, \forall x \in Dom(A), F(B_\delta(x)) \subseteq B_\epsilon(F(x)).$$

Next we state some properties of upper semi-continuous and uniformly upper semi-continuous functions. Given a function $F : A \rightsquigarrow B$, $F^{-1} : B \rightsquigarrow A$ is the function which maps $b \in B$ to the set $\{a \in A \,|\, b \in F(a)\}$.

- If F is upper semi-continuous, then so is F^{-1}.
- If $F : A \rightsquigarrow B$ is upper semi-continuous and A is compact, then F is also uniformly upper semi-continuous.

Trajectories and Transitions Let *Int* denote the set of all intervals of the form $[0, T]$, where $T \in \mathbb{R}_{\geq 0}$ or $[0, \infty)$. Given a set S, a trajectory over S is a function $\tau : D \rightarrow S$, where $D \in Int$ is an interval. Let *Traj*(S)

denote the set of all trajectories over S. A transition over a set S is a pair $\alpha = (s_1, s_2) \in S \times S$. Let $\mathit{Trans}(S)$ denote the set of all transitions over S. Let $\mathit{Size} : \mathit{Traj}(S) \cup \mathit{Trans}(S) \to \mathbb{R}_{\geq 0}$ be a function which assigns a size to the trajectories and transitions. For $\tau \in \mathit{Traj}(S)$, $\mathit{Size}(\tau) = T$ if $\mathit{Dom}(\tau) = [0, T]$ and $\mathit{Size}(\tau) = \infty$ if $\mathit{Dom}(\tau) = [0, \infty)$. For $\alpha \in \mathit{Trans}(S)$, $\mathit{Size}(\alpha) = 0$. Let *First* and *Last* denote the first and last elements, respectively, of trajectories and transitions. For $\tau \in \mathit{Traj}(S)$, $\mathit{First}(\tau) = \tau(0)$, and if $\mathit{Size}(\tau) < \infty$, then $\mathit{Last}(\tau)$ denote the last value $\tau(\mathit{Size}(\tau))$. For $\alpha = (s_1, s_2) \in \mathit{Trans}(S)$, $\mathit{First}(\alpha) = s_1$ and $\mathit{Last}(\alpha) = s_2$.

7.2.1 Hybrid Transition Systems

Defining stability of hybrid systems requires more information about a continuous transition than the values of the state before and after the transition. Hence we introduce the notion of a hybrid transition system, which replaces a continuous transition of a hybrid system by a trajectory of the system.

A *hybrid transition system* (*HTS*) $\mathcal{H}$ is a tuple (S, Σ, Δ), where S is a set of states, $\Sigma \subseteq \mathit{Trans}(S)$ is a set of transitions and $\Delta \subseteq \mathit{Traj}(S)$ is a set of trajectories.

We will not formally define the hybrid transition system corresponding to a hybrid automaton $\mathcal{H}$, but S is essentially $\mathit{States}(\mathcal{H})$, Σ corresponds to the discrete transitions in $[\![\mathcal{H}]\!]$ and Δ corresponds to the flow functions used in defining the continuous transitions of $[\![\mathcal{H}]\!]$.

Notation 62 *We will denote the elements of a HTS using appropriate annotations, for example, the elements of $\mathcal{H}_i$ are $(S_i, \Sigma_i, \Delta_i)$, the elements of $\mathcal{H}'$ are (S', Σ', Δ') and so on.*

An execution of $\mathcal{H}$ is a finite or infinite sequence of trajectories and transitions which have matching end-points. Formally, an *execution* is a sequence $\sigma : D \to \mathit{Traj}(S) \cup \mathit{Trans}(S)$, where $D \in \mathit{SeqDom}$, such that for each $0 \leq i < |\sigma| - 1$, $\mathit{Last}(i) = \mathit{First}(i + 1)$. Let $\mathit{Exec}(\mathcal{H})$ denote the set of all executions of $\mathcal{H}$. Given a set of executions $T \subseteq \mathit{Exec}(\mathcal{H})$, we denote by $\mathit{First}(T)$ the set of initial points of trajectories starting from T, that is, $\mathit{First}(T) = \{\mathit{First}(\sigma(0)) \mid \sigma \in \mathit{Exec}(T)\}$. We will denote the set of states occurring in an execution σ as $\mathit{States}(\sigma)$. For $\alpha \in \Sigma$, $\mathit{States}(\alpha) =$

$\{First(\alpha), Last(\alpha)\}$, for $\tau \in \Delta$, $States(\tau) = \{\tau(t) \mid t \in Dom(\tau)\}$, and for an execution σ, $States(\sigma) = \bigcup_{i \in Dom(\sigma)} States(\sigma(i))$.

Compatibility and Unboundedness We say that two executions are compatible if at every index either both have a transition or both have a trajectory with the same domain. More precisely, an execution σ of $\mathcal{H}_1$ and an execution ρ of $\mathcal{H}_2$ are *compatible*, if $Dom(\sigma) = Dom(\rho)$, and for all $i \in Dom(\sigma)$, $\sigma(i) \in \Sigma_1$ iff $\rho(i) \in \Sigma_2$ and if $\sigma(i) \in \Delta_1$, then $Dom(\sigma(i)) = Dom(\rho(i))$. We will denote the fact that σ and ρ are compatible by the predicate $Comp(\sigma, \rho)$. An execution σ of $\mathcal{H}$ is said to be *unbounded*, denoted by the predicate $Unbounded(\sigma)$, if $\sum_{i=1}^{|\sigma|} Size(\sigma(i)) = \infty$.

Metric Hybrid Transition Systems A metric hybrid transition system is a hybrid transition system whose set of states is equipped with a metric. A *metric hybrid transition system* (MHS) is a pair $(\mathcal{H}, d)$ where $\mathcal{H} = (S, \Sigma, \Delta)$ is a hybrid transition system, and (S, d) is a metric space. Since, the state space has a metric associated with it, we can talk about the distance between states, trajectories, transitions and executions. We will extend the distance function d over S to trajectories, transitions and executions to be the maximum pointwise distance between two elements. Given two trajectories τ_1 and τ_2 in $Traj(S)$ with the same domain, the distance between τ_1 and τ_2, denoted $d(\tau_1, \tau_2)$, is given by $\sup_{t \in Dom(\tau_1)} d(\tau_1(t), \tau_2(t))$. Similarly, distance between two transitions $\alpha_1 = (p_1, q_1)$ and $\alpha_2 = (p_2, q_2)$ is given by $max\{d(p_1, p_2), d(q_1, q_2)\}$. We can then define the distance between two compatible executions to be the maximum of the distance between the corresponding elements of the sequences. Distance between two compatible executions σ and ρ, denoted $d(\sigma, \rho)$, is $\sup_{i \in Dom(\sigma)} d(\sigma(i), \rho(i))$. We define the distance between two incompatible executions to be infinity, that is, $\neg Comp(\sigma, \rho)$ implies $d(\sigma, \rho) = \infty$.

Convergence Two executions are said to converge, if the distance between the two decreases as we consider smaller and smaller suffixes. Let σ and ρ be two compatible and unbounded executions. If $Dom(\sigma) = \mathbb{N}$, then σ and ρ are said to *converge* if for every real $\epsilon > 0$, there exists an $n \in \mathbb{N}$ such that $d(\sigma^{n\rightarrow}, \rho^{n\rightarrow}) < \epsilon$. If $Dom(\sigma) = [n]$, then σ and ρ are said to converge if for every real $\epsilon > 0$, there exists $t \in \mathbb{R}_{\geq 0}$ such that $d(\sigma(n-1)(t'), \rho(n-1)(t')) < \epsilon$

for all $t' \geq t$.

7.2.2 Simulations and Bisimulations

We define simulation and bisimulation for hybrid transition systems along the lines of [62].

Let $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$ and $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$ be two hybrid transition systems. Let $R \subseteq S_1 \times S_2$ be a binary relation on the states of $\mathcal{H}_1$ and $\mathcal{H}_2$. We can extend this relation to the trajectories, transitions and executions in a natural way as follows. Given two trajectories $\tau_1 \in \Delta_1$ and $\tau_2 \in \Delta_2$, $R(\tau_1, \tau_2)$ holds if and only if $Dom(\tau_1) = Dom(\tau_2) = D$, and for all $t \in D$, $R(\tau_1(t), \tau_2(t))$. Similarly, $R(\alpha_1, \alpha_2)$ holds, where $\alpha_1 \in \Sigma_1$ and $\alpha_2 \in \Sigma_2$, iff $First(\alpha_1) = First(\alpha_2)$ and $Last(\alpha_1) = Last(\alpha_2)$. Given two executions $\sigma_1 \in Exec(\mathcal{H}_1)$ and $\sigma_2 \in Exec(\mathcal{H}_2)$, $R(\sigma_1, \sigma_2)$ holds iff $Comp(\sigma_1, \sigma_2)$ and for every $i \in Dom(\sigma_1)$, $R(\sigma_1(i), \sigma_2(i))$.

Given two hybrid transition systems $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$ and $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$, a binary relation $R \subseteq S_1 \times S_2$ is said to be a *simulation relation* from $\mathcal{H}_1$ to $\mathcal{H}_2$, denoted $\mathcal{H}_1 \preceq_R \mathcal{H}_2$, if for every $s_1 \in S_1$, there exists an $s_2 \in S_2$ such that $(s_1, s_2) \in R$, and for every $(s_1, s_2) \in R$, the following conditions hold:

- for every state s_1' such that $\Sigma_1(s_1, s_1')$, there exists a state s_2' such that $\Sigma_2(s_2, s_2')$ and $(s_1', s_2') \in R$; and
- for every trajectory $\tau_1 \in \Delta_1$ such that $First(\tau_1) = s_1$, there exists a trajectory $\tau_2 \in \Delta_2$ such that $First(\tau_2) = s_2$, $Dom(\tau_1) = Dom(\tau_2)$ and for all $t \in Dom(\tau_1) = Dom(\tau_2)$, $(\tau_1(t), \tau_2(t)) \in R$.

If there exists an R such that $\mathcal{H}_1 \preceq_R \mathcal{H}_2$, then we say that $\mathcal{H}_2$ simulates $\mathcal{H}_1$, denoted by $\mathcal{H}_1 \preceq \mathcal{H}_2$. Intuitively, if $\mathcal{H}_2$ simulates $\mathcal{H}_1$, then $\mathcal{H}_2$ has more behaviors than $\mathcal{H}_1$. $\mathcal{H}_2$ is also referred to as an abstraction of $\mathcal{H}_1$. Simulations preserve various discrete time properties, such as, safety properties, in that, if $\mathcal{H}_1 \preceq \mathcal{H}_2$ and $\mathcal{H}_2$ satisfies the property, then we can conclude that $\mathcal{H}_1$ satisfies the property as well.

A binary relation $R \subseteq S_1 \times S_2$ is a *bisimulation relation* between $\mathcal{H}_1$ and $\mathcal{H}_2$, denoted $\mathcal{H}_1 \sim_R \mathcal{H}_2$, if $\mathcal{H}_1 \preceq_R \mathcal{H}_2$ and $\mathcal{H}_2 \preceq_{R^{-1}} \mathcal{H}_1$. So a bisimulation relation preserves properties in both directions, in that, $\mathcal{H}_1$ satisfies a bisimulation invariant property iff $\mathcal{H}_2$ satisfies it.

7.3 Stability of Hybrid Transition Systems

In this section, we introduce various properties related to the stability of systems (a good introductory book is [63]). Intuitively, stability is a property which captures the notion that small changes in the initial state of the system result in only small changes in the behaviors of the system starting from those states.

We first define the notion of Lyapunov stability. Given a *HTS* $\mathcal{H}$ and a set of executions $T \subseteq \mathit{Exec}(\mathcal{H})$, we say that $\mathcal{H}$ is *Lyapunov stable* (*LS*) with respect to T, if for every $\epsilon > 0$ in $\mathbb{R}_{\geq 0}$, there exists a $\delta > 0$ in $\mathbb{R}_{\geq 0}$ such that the following condition holds:

$$\forall \sigma \in \mathit{Exec}(\mathcal{H}), d(\mathit{First}(\sigma(0)), \mathit{First}(T)) < \delta \Rightarrow \exists \rho \in T, d(\sigma, \rho) < \epsilon$$

The above statement says that for every execution σ of the system $\mathcal{H}$ which starts within a distance δ of some execution ρ' in T, there exists an execution ρ in T which is compatible with σ and is within a distance ϵ at all points of the execution.

Next we define a stronger notion of stability called asymptotic stability which in addition to Lyapunov stability requires that the executions starting close also converge as time goes to infinity. A *HTS* $\mathcal{H}$ is said to be *asymptotically stable* (*AS*) with respect to a set of execution $T \subseteq \mathit{Exec}(\mathcal{H})$, if it is Lyapunov stable and there exists a $\delta > 0$ in $\mathbb{R}_{\geq 0}$ such that

$$\forall \sigma \in \mathit{Exec}(\mathcal{H}), d(\mathit{First}(\sigma(0)), \mathit{First}(T)) < \delta \Rightarrow \exists \rho \in T, \mathit{Conv}(\sigma, \rho)$$

So a system $\mathcal{H}$ is asymptotically stable with respect to a set of its executions T if $\mathcal{H}$ is Lyapunov stable with respect to T and every execution starting within a distance of δ from the starting point of some execution in T converges to some execution in T.

Remark 63 *The reader might be familiar with the more standard definition of stability, which is defined with respect to an equilibrium point. An equilibrium point is a point in the state space whose value does not change with the evolution of time. A system is then said to be Lyapunov stable with respect to an equilibrium point if for every $\epsilon > 0$ there exists a δ ball around the equi-*

librium point such that every trajectory starting in this δ ball remains with in an ϵ ball around the equilibrium point. Similarly, in the case of asymptotic stability, the trajectories starting in a δ ball around the equilibrium point are required to converge to the equilibrium point, in addition to being stable in the sense of Lyapunov. Here, we consider the more general notion of stability with respect to a trajectory or more generally to a set of executions. In this framework, an equilibrium point is an execution whose value always remains the same.

Remark 64 *Note that since we consider stability with respect to a set of trajectories, we can specify stability with respect to trajectories starting from different locations.*

7.4 Uniformly Continuous Relations and Stability Preservation

The main focus of this chapter is to examine the right notions of pre-orders required to reason about stability properties. In the discrete setting, most interesting properties are known to be invariant under the classical notion of bisimulation. However, we show that stability is not invariant under bisimulation, that is, there are systems which are bisimilar, but one is stable where as the other is not. Then we introduce additional continuity requirements on the bisimulation relation which force invariance under stability.

7.4.1 Bisimulation is not enough

We will show that stability is not bisimulation invariant. We explain the example from the introduction in more detail. Consider a hybrid transition system $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$, where

- the state space S_1 is the set $\mathbb{R}^2_{\geq 0}$, which is the positive quadrant of the two dimensional plane;
- the set of transitions Σ_1 is the empty set; and
- Δ_1 is the set $\{f_m \,|\, m \in \mathbb{R}_{\geq 0}\}$, where for a particular $m \in \mathbb{R}_{\geq 0}$, $f_m : [0, \infty) \to \mathbb{R}^2_{\geq 0}$ is the trajectory such that $f(t) = (t, m)$.

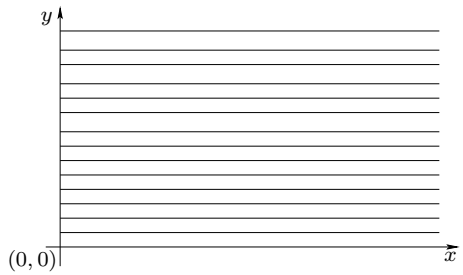

Figure 7.1: Lyapunov stable system

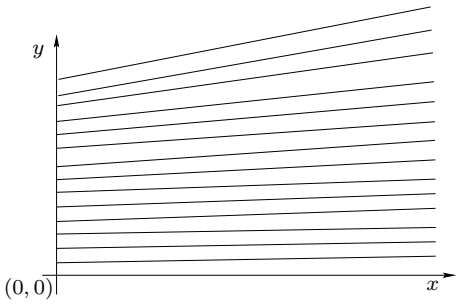

Figure 7.2: Unstable system

As shown in Figure 7.1, $\mathcal{H}_1$ consists of trajectories which start on the positive y-axis and evolve parallel to the positive x-axis. It is easy to see that $\mathcal{H}_1$ is Lyapunov stable with respect to the unique trajectory τ_1 which starts at the origin and moves along the x-axis.

Now let us consider another system $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$, shown in Figure 7.2, which is similar to $\mathcal{H}_1$, that is, $S_2 = S_1$ and $\Sigma_2 = \Sigma_1$, except that $\Delta_2 = \{f_m : [0, \infty) \to \mathbb{R}_{\geq 0} \,|\, f_m(t) = (t, m(1 + t)), m \in \mathbb{R}_{\geq 0}\}$. The trajectories of $\mathcal{H}_2$ start on the positive y-axis and evolve along a straight line whose slope is given by the y intercept. So they form a diverging set of straight lines. Consider the trajectory τ_2 which starts at the origin and evolves along the x-axis. Note that $\mathcal{H}_2$ is not Lyapunov stable with respect to $\{\tau_2\}$.

However we can show that there exists a bisimulation relation between $\mathcal{H}_1$ and $\mathcal{H}_2$, given by, $R = \{((x_1, y_1), (x_2, y_2)) \,|\, x_1 = x_2 \text{ and } y_2 = y_1(1 + x_1)\}$. This shows that Lyapunov stability is not invariant under bisimulation. In fact, observe that the relation R gives rise to the function which maps (x, y) to $(x, y(1 + x))$ which is a bi-continuous bijection. Hence, even with the additional constraint of continuity in both directions on the bisimulation relations, Lyapunov stability is not preserved. Hence we introduce the notions of uniformly continuous simulations and bisimulations, which impose sufficient constraints on simulation and bisimulation to force Lyapunov stability preservation.

Remark 65 *We can show in a similar fashion that bi-continuous bisimulations do not preserve asymptotic stability. For example, consider a system $\mathcal{H}_3$ which is similar to $\mathcal{H}_2$ except that f_m is defined as $f_m(t) = (t, me^{-t})$. Note that $\mathcal{H}_1$ is not asymptotically stable, where as $\mathcal{H}_3$ is. And there is a*

bi-continuous bisimulation relation given by $R = \{((x_1, y_1), (x_2, y_2)) \,|\, x_1 = x_2$ *and* $y_2 = y_1 e^{-x_1}\}$.

7.4.2 Uniformly Continuous Simulations and Bisimulations

Recall that a binary relation $R \subseteq A \times B$ defines in a natural way a set-valued function, namely, $F : A \rightsquigarrow B$ given by, for all $a \in A$, $F(a) = \{b \,|\, (a, b) \in R\}$. In the sequel, we use R also to denote the set-valued function associated with it.

Definition 66 *A* uniformly continuous simulation *from a HTS* $\mathcal{H}_1$ *to a HTS* $\mathcal{H}_2$ *is a binary relation* $R \subseteq S_1 \times S_2$ *such that* R *is a simulation from* $\mathcal{H}_1$ *to* $\mathcal{H}_2$*, and* R *and* R^{-1} *are uniformly upper semi-continuous functions.*

Given *HTS*s $\mathcal{H}_1$ and $\mathcal{H}_2$, and sets of executions $T_1 \subseteq Exec(\mathcal{H}_1)$ and $T_2 \subseteq Exec(\mathcal{H}_2)$, a binary relation $R \subseteq S_1 \times S_2$ is said to be *semi-complete* with respect to T_1 and T_2 if $R(First(T_1)) = First(T_2)$, and for every $\tau_2 \in T_2$, there is a trajectory in $\tau_1 \in T_1$ such that $R(\tau_1, \tau_2)$, and for every $x \in States(T_2)$, $R(x)$ is a singleton. And R is *complete* with respect to T_1 and T_2 if R and R^{-1} are semi-complete with respect to T_1 and T_2.

The next theorem states that uniformly continuous simulations preserve Lyapunov and asymptotic stability.

Theorem 67 *Let* $\mathcal{H}_1$ *and* $\mathcal{H}_2$ *be two hybrid transition systems and* $T_1 \subseteq Exec(\mathcal{H}_1)$ *and* $T_2 \subseteq Exec(\mathcal{H}_2)$ *be two sets of execution. Let* $R \subseteq S_1 \times S_2$ *be a uniformly continuous simulation from* $\mathcal{H}_1$ *to* $\mathcal{H}_2$*, and let* R *be semi-complete with respect to* T_1 *and* T_2*. Then the following holds:*

1. $\mathcal{H}_2$ *is Lyapunov stable with respect to* T_2 *implies* $\mathcal{H}_1$ *is Lyapunov stable with respect to* T_1*; and*

2. $\mathcal{H}_2$ *is asymptotically stable with respect to* T_2 *implies* $\mathcal{H}_1$ *is asymptotically stable with respect to* T_1*.*

Proof Given a uniformly upper semi-continuous function F, let us denote by $\delta_{F,\epsilon}$, the element in $\mathbb{R}_{\geq 0}/\{0\}$ such that for all $x \in Dom(F)$, $F(B_{\delta_{F,\epsilon}}(x)) \subseteq B_\epsilon(F(x))$. Similarly, let $\delta_{\mathcal{H},T,\epsilon}$ be the δ in the definition of Lyapunov stability

of $\mathcal{H}$ with respect to T corresponding to ϵ. Let us fix F to be the set valued function corresponding to the binary relation R.

Proof of part (1): Let $\mathcal{H}_2$ be Lyapunov stable with respect to T_2. We will show that $\mathcal{H}_1$ is Lyapunov stable with respect to T_1. Fix an $\epsilon > 0$. Let $\epsilon' = \delta_{F^{-1},\epsilon}$, $\delta' = \delta_{\mathcal{H}_2,T_2,\epsilon'}$ and $\delta = \delta_{F,\delta'}$. For the above ϵ, we will show that the above definition of δ satisfies the condition of Lyapunov stability of $\mathcal{H}_1$ with respect to T_1. Let $\sigma_1 \in \mathit{Exec}(\mathcal{H}_1)$ be such $d(\mathit{First}(\sigma_1(0)), \mathit{First}(T_1)) < \delta$. That is, there exists $\rho_1 \in T_1$ such that $d(\mathit{First}(\sigma_1(0)), \mathit{First}(\rho_1(0))) < \delta$. Let $\rho_2 \in T_2$ be such that $R(\rho_1, \rho_2)$. It follows from the uniform upper semi-continuity of F that $F(B_\delta(\mathit{First}(\sigma_1(0)))) \subseteq B_{\delta'}(F(\mathit{First}(\sigma_1(0))))$. Therefore $F(\mathit{First}(\rho_1(0))) \subseteq B_{\delta'}(F(\mathit{First}(\sigma_1(0))))$, which implies that $\mathit{First}(\rho_2(0)) \in B_{\delta'}(F(\mathit{First}(\sigma_1(0))))$. Therefore, there exists $s \in F(\mathit{First}(\sigma_1(0)))$ such that $d(s, \mathit{First}(\rho_2(0))) < \delta'$. Let σ_2 be an execution in $\mathcal{H}_2$ such that $\mathit{First}(\sigma_2) = s$ and $R(\sigma_1, \sigma_2)$. Such an execution exists since $R(\mathit{First}(\sigma_1), s)$ and R is a simulation. So σ_2 is an execution of $\mathcal{H}_2$ such that $d(\mathit{First}(\sigma_2(0)), \mathit{First}(T_2)) < \delta'$. Then from the Lyapunov stability of $\mathcal{H}_2$ we know that there exists an execution $\gamma_2 \in T_2$ such that $d(\sigma_2, \gamma_2) < \epsilon'$. Let γ_1 be the unique execution such that $R(\gamma_1, \gamma_2)$ and $\gamma_1 \in T_1$. We need to show that $d(\sigma_1, \gamma_1) < \epsilon$. We need to show that for any $i \in \mathit{Dom}(\sigma_1)$, $d(\sigma_1(i), \gamma_1(i)) < \epsilon$. Let us consider the case where $\sigma_1(i) \in \Delta_1$, the case where it belongs to Σ_1 is similar and skipped. Given any $t \in \mathit{Dom}(\sigma_1(i))$, we need to show that $d(\sigma_1(i)(t), \gamma_1(i)(t)) < \epsilon$. Note that $d(\sigma_2(i)(t), \gamma_2(i)(t)) < \epsilon'$. From the uniform upper semi-continuity of F^{-1}, it follows that $F^{-1}(B_{\epsilon'}(\gamma_2(i)(t))) \subseteq B_\epsilon(F^{-1}(\gamma_2(i)(t)))$. This implies that $F^{-1}(\sigma_2(i)(t)) \subseteq B_\epsilon(F^{-1}(\gamma_2(i)(t)))$ which further implies that $\sigma_1(i)(t) \in B_\epsilon(F^{-1}(\gamma_2(i)(t)))$. Since $F^{-1}(\gamma_2(i)(t)) = \{\gamma_1(i)(t)\}$ (due to R being complete), we have that $d(\sigma_1(i)(t), \gamma_1(i)(t)) < \epsilon$.

Proof of part (2): It is similar to the proof of part (1) except for the following differences. Given a δ' corresponding to the asymptotic stability of $\mathcal{H}_2$ with respect to T_2, we choose $\delta = \delta_{F,\delta'}$ as before. The proof is then similar to the previous part until we obtain γ_1 and γ_2. Then we need to show that σ_1 and γ_1 converge using the fact that σ_2 and γ_2 converge. Given an ϵ, we choose $\epsilon' = \delta_{F^{-1},\epsilon}$. Since σ_2 and γ_2 converge, there is a point in their executions after which the distance between them is within ϵ'. It can be argued similar to the previous case, that the distance between σ_1 and γ_1 starting from the same points is within ϵ. Since for every ϵ there exists a suffix of σ_1 and γ_1 such that the distance between the suffixes is within ϵ, we

know that σ_1 and γ_1 converge. ∎

The above theorem implies that the stability of a system $\mathcal{H}_1$ can be concluded by analysing a potentially simpler system $\mathcal{H}_2$ which uniformly continuously simulates $\mathcal{H}_1$.

As a corollary of Theorem 67, we obtain that Lyapunov stability and asymptotic stability are invariant under uniformly continuous bisimulations.

Definition 68 *A* uniformly continuous bisimulation *between two HTSs $\mathcal{H}_1$ and $\mathcal{H}_2$ is a binary relation $R \subseteq S_1 \times S_2$ such that R is a uniformly continuous simulation from $\mathcal{H}_1$ to $\mathcal{H}_2$ and R^{-1} is a uniformly continuous simulation from $\mathcal{H}_2$ to $\mathcal{H}_1$.*

Note that R is a uniformly continuous bisimulation iff R is a bisimulation between $\mathcal{H}_1$ and $\mathcal{H}_2$ and both R and R^{-1} are uniformly upper semi-continuous functions. The next corollary of Theorem 67 establishes the invariance of Lyapunov and asymptotic stability under uniformly continuous bisimulations.

Corollary 69 *Let $\mathcal{H}_1$ and $\mathcal{H}_2$ be two hybrid transition systems and $T_1 \subseteq Exec(\mathcal{H}_1)$ and $T_2 \subseteq Exec(\mathcal{H}_2)$ be two sets of execution. Let $R \subseteq S_1 \times S_2$ be a uniformly continuous bisimulation between $\mathcal{H}_1$ and $\mathcal{H}_2$, and let R be complete with respect to T_1 and T_2. Then the following holds:*

1. *$\mathcal{H}_1$ is Lyapunov stable with respect to T_1 if and only if $\mathcal{H}_2$ is Lyapunov stable with respect to T_2; and*
2. *$\mathcal{H}_1$ is asymptotically stable with respect to T_1 if and only if $\mathcal{H}_2$ is asymptotically stable with respect to T_2.*

7.5 Applications of Theorem 67

In this section, we show that various methods used in proving Lyapunov and asymptotic stability of systems can be formulated as constructing a simpler system which uniformly continuously simulates the original system and showing that the simpler system is Lyapunov or asymptotically stable, respectively.

7.5.1 Lyapunov Functions

In this section, we show that Lyapunov's direct method for proving stability of systems can be formulated as constructing a simpler system using Lyapunov functions which uniformly continuously simulates the original system and then establishing the stability of the simplified systems.

Consider the following time-invariant system,

$$\dot{x} = f(x), \qquad x \in \mathbb{R}^n, \tag{7.1}$$

where $f : \mathbb{R}^n \to \mathbb{R}$ and let $\bar{0}$ be an equilibrium point, that is, $f(\bar{0}) = \bar{0}$. Consider a C^1 (i.e., continuously differentiable) function $V : \mathbb{R}^n \to \mathbb{R}$. It is called *positive definite* if $V(0) = 0$ and $V(x) > 0$ for all $x \neq 0$. Let

$$\dot{V}(x) = \frac{\partial V}{\partial x} f(x),$$

and note that $\dot{V}$ is the time derivative of $V(x(t))$, where $x(t)$ is a solution of the Equation 7.1.

Theorem 70 (Lyapunov [63]) *Suppose that there exists a neighborhood Ω of $\bar{0}$ and a positive definite C^1 function $V : \mathbb{R}^n \to \mathbb{R}$ satisfying the algebraic condition:*

$$\dot{V}(x) \leq 0, \qquad \forall x \in \Omega. \tag{7.2}$$

Then System 7.1 is Lyapunov stable.

Furthermore, if $\dot{V}$ satisfies

$$\dot{V}(x) < 0, \qquad \forall x \in \Omega/\{0\}, \tag{7.3}$$

then System 7.1 is asymptotically stable.

A C^1 positive definite function satisfying inequality 7.2 is called a *weak Lyapunov function* and one satisfying 7.3 is called a *Lyapunov function.*

In the above theorem we can assume that the set Ω is a compact set containing an open neighborhood of 0. We now argue that the Lyapunov function over Ω and its inverse are uniformly upper semi-continuous continuous functions, and the conditions in 7.2 and 7.3 prove that a simpler system over $\mathbb{R}_{\geq 0}$ is Lyapunov stable and asymptotically stable, respectively.

Consider a hybrid transition system $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$, where $S_1 = \Omega$, $\Sigma_1 = \emptyset$, Δ_1 is the set of C^1 trajectories $\tau : D \to \Omega$ such that $d\tau(t)/dt = f(\tau(t))$ and $\tau(t) \in \Omega$ for all $t \in D$, that is, the set of solutions of 7.1 which remain with in Ω. Next consider another hybrid transition system $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$, where $S_2 = V(\Omega)$, $\Sigma_2 = \emptyset$ and Δ_2 is the set of trajectories $\tau : D \to S_2$ such that there exist $\tau' : D \to \Omega$ in Δ_1 and $\tau(t) = V(\tau'(t))$ for all $t \in D$.

Note that $V : S_1 \to S_2$ can be considered as a set valued function where for every point $s \in S_1$, $V(s)$ is a singleton set. First we show that this set valued function $V : S_1 \rightsquigarrow S_2$ is a uniformly upper semi-continuous function and so is V^{-1}. First note that V is upper semi-continuous on the whole of $\mathbb{R}^n$, and it follows from the property of upper semi-continuity that V^{-1} is upper semi-continous. V is a uniformly upper semi-continuous function since Ω is a compact set. Further $S_2 = V(\Omega)$ is a compact set. Hence V^{-1} is also a uniformly upper semi-continous function.

Let τ^* be the unique trajectory in $\mathcal{H}_1$ and $\mathcal{H}_2$ which starts at 0 and remains at 0 forever. Next we show that the Condition 7.2 implies that $\mathcal{H}_2$ is Lyapunov stable with respect to τ^* and Condition 7.3 implies that $\mathcal{H}_2$ is asymptotically stable with respect to τ^*. Condition 7.2 implies that for any $\tau \in \Delta_2$, $t_1 < t_2$ implies $\tau(t_1) \geq \tau(t_2)$. Therefore, given any ϵ by choosing a $\delta \leq \epsilon$ which is such that a δ ball around $\bar{0}$ is contained in Ω, we obtain that any trajectory in Δ_2 starting within a δ ball remain within an ϵ ball (in fact within a δ ball). Next let us consider Condition 7.3. For a trajectory $\tau : [0, \infty) \to S_2$ starting from a state x we need to show that the trajectory converges to 0. Since V is positive and decreasing along the corresponding solution, it has a limit $c \geq 0$ as $t \to \infty$. If $c = 0$, then we are done. Otherwise, the solution cannot enter the set $\{x : V(x) < c\}$. In this case the solution evolves in a compact set that does not contain the origin. Let the compact set be C. Let $d = max_{x \in S} \dot{V}(x)$; this number is well defined due to compactness of S and negative due to 7.3. We have $\dot{V} \leq d$, and hence $V(t) \leq V(0) + dt$. But then V will eventually become smaller than c, which is a contradiction.

So Lyapunov's theorem can be casted as reducing the original system to a simpler system by uniformly upper semi-continuous functions and proving the stability of the simpler system. The above steps give an alternate proof of Lyapunov stability using Theorem 67.

7.5.2 Multiple Lyapunov Functions

Let us consider a set of N dynamical systems, namely, $\dot{x} = f_i(x)$, $0 \leq i < N$. A *switching signal* is a piecewise constant function $\omega : [0, \infty) \to [N]$ such that ω has bounded number of discontinuities, called *switching times*, on every bounded time interval. We assume for concreteness that ω is continuous from right everywhere: $\omega(t) = lim_{r \to t^+} \sigma(r)$ for each $r \geq 0$. We assume for simplicity of presentation that there are infinitely many switching times. The set of N dynamical systems with a switching signal is called a *switched system.*

Let us fix the following switched system:

$$\dot{x} = f_i(x), i \in [N], x \in \mathbb{R}^n, \omega : [0, \infty) \to [N]. \tag{7.4}$$

The solution of this system is the set of functions $\sigma : [0, \infty) \to \mathbb{R}^n$ such that σ restricted to the interval between two switching times is a solution to the corresponding differential equation. More precisely, let $0 = t_0 < t_1 < \cdots$ be the switching times of ω. For two consecutive switching times t_i and t_{i+1} with $\omega(t_i) = j$, the function $g_i : [0, t_{i+1} - t_i] \to \mathbb{R}^n$ given by $g_i(t) = \sigma(t_i + t)$ is a solution to the differential equation $\dot{x} = f_j(x)$.

Theorem 71 (Multiple Lyapunov Method [16]) *Suppose there exists a neighborhood Ω of origin and N positive definite C^1 functions $V_i : \mathbb{R}^n \to \mathbb{R}$, $i \in [N]$, such that*

$$\dot{V}_i(x) = \frac{\partial V_i}{\partial x} f_i(x) \leq 0, \forall x \in \Omega$$

and for every pair of switching times t and t' such that $t > t'$, $\omega(t) = \omega(t')$ and for all switching times $t > t'' > t'$, $\omega(t) \neq \omega(t'')$,

$$V_i(\sigma(t)) \leq V_i(\sigma(t')),$$

for every solution σ of the switched system. Then the switched system is Lyapunov stable.

The above theorem can again be formulated as establishing a function from a *HTS* $\mathcal{H}_1$ to a simpler *HTS* $\mathcal{H}_2$ which is uniformly upper semi-continuous and proving that $\mathcal{H}_2$ is Lyapunov stable using the properties of V_is. We will sketch this formulation below.

Again we can assume that Ω is a compact set containing a neighborhood of origin. Here $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$, where $S_1 = \mathbb{N} \times \Omega$, $\Sigma_1 = \{((i,x),(i+1,x)) \,|\, i \in \mathbb{N}, x \in \Omega\}$, and Δ_1 consists of trajectories $\tau : [0, t_{i+1} - t_i] \to \{i\} \times \Omega$ for some $i \in \mathbb{N}$ such that $\tau' : [0, t_{i+1} - t_i] \to \mathbb{R}^n$, given by, $\tau(t) = (i, \tau'(t))$, is the solution of the differential equation $\dot{x} = f_j(x)$, where $j = \omega(t_i)$. And the simpler system $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$, where $S_2 = \bigcup_{i \in \mathbb{N}} \{i\} \times V_{\omega(t_i)}(\Omega)$, $\Sigma_2 = \{((i, V_{\omega(t_i)}(x)), (i+1, V_{\omega(t_i)}(x))) \,|\, ((i,x),(i+1,x)) \in \Sigma_1\}$ and Δ_2 is the set of trajectories $\tau : [0,t] \to \{i\} \times V_{\omega(t_i)}(\Omega)$ such that there exists $\tau' : [0,t] \to \{i\} \times \Omega$ in Δ_1 and $\tau(t) = (i, V_{\omega(t_i)}(x))$ where $\tau'(t) = (i,x)$. Then the function $h : S_1 \to S_2$ given by $h((i,x)) = (i, V_{\omega(t_i)}(x))$ can be shown to be a uniformly continuous simulation. Further $\mathcal{H}_2$ is a simpler system whose stability can be easily established from the properties of V_is.

7.5.3 Hartman-Grobman Theorem

We consider a theorem due to Hartman-Grobman which constructs linear approximations of non-linear dynamics and establishes a homeomorphism between the two dynamics. We show that the homeomorphic mapping from the non-linear dynamics to the linear dynamics is a uniformly continuous bisimulation. And hence one can use these reductions from non-linear to linear dynamics to potentially establish stability properties of non-linear dynamics by proving stability of the simpler linear dynamics, and using Theorem 67 to deduce the stability of the non-linear dynamics.

We need certain definitions to present the theorem formally. A function $f : A \to B$, where $A, B \subseteq \mathbb{R}^n$ is a *homeomorphism* if f is a bijection and both f and f^{-1} are continuous. A function $F : \mathbb{R}^n \to \mathbb{R}^m$ is given by m-real valued component functions, $y_1(x), \cdots, y_m(x)$, where $x = (x_1, \cdots, x_n)$. The partial derivatives of all these functions (if they exist) can be organized in a $m \times n$ matrix called the *Jacobian* of F, denoted by $DF(x)$.

$$DF(x) = \begin{bmatrix} \frac{\partial y_1}{\partial x_1} & \cdots & \frac{\partial y_1}{\partial x_n} \\ \vdots & \ddots & \vdots \\ \frac{\partial y_m}{\partial x_1} & \cdots & \frac{\partial y_m}{\partial x_n} \end{bmatrix}$$

Given an n-vector $a = (a_1, \cdots, a_n) \in \mathbb{R}^n$, $DF(a)$ is the matrix obtained by substituting x_i in the terms of the matrix $DF(x_1, \cdots, x_n)$ by a_i. A square

matrix A is *hyperbolic* if none of its eigen values are purely imaginary values (including 0).

Let $\Omega \subseteq \mathbb{R}^n$ be an open set and $F : \Omega \to \mathbb{R}^n$ be continuously differentiable. Suppose that $x_0 \in \Omega$ is a hyperbolic equilibrium point of the autonomous equation

$$\dot{x} = F(x), \tag{7.5}$$

that is, $A = DF(x_0)$ is a hyperbolic matrix. Let φ be the (local) flow generated by Equation 7.5, that is, $\varphi : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^n$ is a differentiable function such that $d\varphi(x,t)/dt = F(\varphi(x,t))$ for all $t \in \mathbb{R}_{\geq 0}$.

Theorem 72 (Local Hartman-Grobman Theorem for Flows)
Let Ω, F, x_0, A and φ be as defined above. Then there are neighborhoods U and V of x_0 and a homeomorphism $h : U \to V$ such that

$$\varphi(h(x), t) = h(x_0 + e^{tA}(x - x_0)) \tag{7.6}$$

whenever $x \in U$ and $x_0 + e^{tA}(x - x_0) \in U$.

We can replace the neighborhoods U and V in the above theorem by compact sets U and $V = h(U)$, respectively, such that U contains some neighborhood of x_0. We show that the homeomorphism h from U to $h(U)$ is essentially a uniformly continuous bisimulation between the hybrid transition system defined by the vector field F and its linearization A at x_0.

More precisely, $\mathcal{H}_1 = (S_1, \Sigma_1, \Delta_1)$ be a hybrid transition system, where $S_1 = U$, $\Sigma_1 = \emptyset$, Δ_1 is the set of C^1 trajectories $\tau : D \to U$ such that there exists $x \in U$ such that $\tau(t) = \varphi(x,t)$ and $\tau(t) \in U$ for all $t \in D$. Next consider another hybrid transition system $\mathcal{H}_2 = (S_2, \Sigma_2, \Delta_2)$, where $S_2 = h(U)$, $\Sigma_2 = \emptyset$ and Δ_2 is the set of trajectories $\tau : D \to S_2$ such that there exists $x \in S_2$ such that $\tau(t) = x_0 + e^{tA}(x - x_0)$ and $\tau(t) \in S_2$ for all $t \in D$ (which are essentially the solutions of the linearization $\dot{x} = A(x - x_0) + x_0$ of the system $\dot{x} = F(x)$). Since h is a continuous function with a compact domain, it is uniformly continuous, and similarly, h^{-1} is continuous function with a compact domain $h(U)$, and is therefore uniformly continuous. It is easy to see from Condition 7.6 that h is a bisimulation from $\mathcal{H}_1$ to $\mathcal{H}_2$. Again, we see that the reduction defined in Hartman-Grobman theorem from the non-linear dynamics to linear dynamics is a uniformly continuous bisimulation.

7.6 Conclusions

In this chapter, we investigated pre-orders for reasoning about stability properties of dynamical and hybrid systems. We showed that stability is not invariant under the classical notion of bisimulation. We introduced the notions of uniformly continuous simulations and bisimulations, which add an additional uniform continuity constraint on the simulation and bisimulation relations. We showed that the two standard notions of stability, namely, Lyapunov and asymptotic stability are invariant under uniformly continuous bisimulations. Further, we presented evidence of the fact that these pre-order can potentially be used to reason about stability by casting several proofs of stability analysis as constructing simpler systems which uniformly continuously simulate the original system, establishing the stability of these simpler systems, and thereby deducing the stability of the original system. In the future, we intend to investigate stability properties for systems with input and output.

CHAPTER 8

CONCLUSIONS AND FUTURE DIRECTIONS

In this thesis, we explored the idea of approximation based verification of hybrid systems. We focused on safety and stability properties, and developed various techniques and tools for approximation and analysis. We presented two approximation techniques for analysing safety properties. The first technique was an error based approximation technique for analysing bounded properties which applied to a general class of systems. Though, in general, the method is only semi-automatic - approximation is manual, but verification of the approximate system is automated; we presented a subclass of systems for which the whole process can be automated. For the case of linear dynamics, our method scales to even systems of 100 dimensions, and has practical benefits both in terms of space and time over various existing methods. The second technique concentrated on property based abstraction refinement of systems. Our contribution is the idea of hybrid CEGAR, where we focus on abstracting a complex hybrid system to a system which is another hybrid system as opposed to a finite state system considered in various approaches in the literature. This method has various advantages over the traditional discrete abstraction based CEGAR in terms of simplifying the various steps of the CEGAR loop, thereby guaranteeing progress in the abstraction refinement process. We have built a tool called HARE based on this technique for the class of rectangular hybrid systems; and our experimental results suggest that our method has the potential to scale. Future work in this direction will be on extending these automatic methods to larger classes of systems such as those with linear and non-linear dynamics.

In terms of stability analysis, we presented a framework for analysing asymptotic stability of discrete-time hybrid systems. We extended the framework of Tsitsiklis to handle ω-regular interactions between a finite set of operators, which suffices to model hybrid systems in discrete semantics. We also explored pre-orders for analysing stability properties, and introduced

the notion of uniformly continuous bisimulations under which various stability properties are invariant. As seen in Figure 1.1 of Chapter 1, identifying the right pre-orders is an important step towards developing approximations which preserve stability properties. These techniques provide a foundation for automated analysis of stability properties for hybrid systems. In the future, we intend to develop automated methods for analysis based on these observations and results.

REFERENCES

[1] R. Alur, C. Courcoubetis, N. Halbwachs, T. Henzinger, P.-H. Ho, X. Nicollin, A. Olivero, J. Sifakis, and S. Yovine. The algorithmic analysis of hybrid systems. *Theoretical Computer Science*, 138(1):3–34, 1995.

[2] R. Alur, C. Courcoubetis, T. A. Henzinger, and P. hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In *Hybrid Systems*, pages 209–229, 1992.

[3] R. Alur, T. Dang, and F. Ivancic. Counter-Example Guided Predicate Abstraction of Hybrid Systems. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–223, 2003.

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

[5] R. Alur, R. Grosu, Y. Hur, V. Kumar, and I. Lee. Modular specification of hybrid systems in charon. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 6–19, 2000.

[6] R. Alur, T. A. Henzinger, and P. hsin Ho. Automatic symbolic verification of embedded systems. *IEEE Transactions on Software Engineering*, 22:181–201, 1996.

[7] E. Asarin, T. Dang, and A. Girard. Hybridization methods for the analysis of nonlinear systems. *Acta Informatica*, 43(7):451–476, 2007.

[8] E. Asarin, T. Dang, and O. Maler. The d/dt tool for verification of hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*, pages 365–370, 2002.

[9] E. Asarin, O. Maler, and A. Pnueli. Reachability analysis of dynamical systems having piecewise-constant derivatives. *Theoretical Computer Science*, 138(1):35–65, 1995.

[10] E. Asarin, G. Schneider, and S. Yovine. On the decidability of the reachability problem for planar differential inclusions. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 89–104, 2001.

[11] E. Asarin, G. Schneider, and S. Yovine. Algorithmic analysis of polygonal hybrid systems, part I: Reachability. *Theoretical Computer Science*, 379(1-2):231–265, 2007.

[12] T. Ball and S. Rajamani. Bebop: A symbolic model checker for Boolean programs. In *Proceedings of the SPIN Workshop on Model Checking Software*, pages 113–130, 2000.

[13] J. Bengtsson, K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Uppaal - a tool suite for automatic verification of real-time systems. In *Hybrid Systems*, pages 232–243, 1995.

[14] V. Blondel, J. Hendrickx, A. Olshevsky, and J. Tsitsiklis. Convergence in multiagent coordination consensus and flocking. In *Proceedings of the Joint IEEE Conference on Decision and Control and European Control Conference*, pages 2996–3000, 2005.

[15] V. Borkar and P. Varaiya. Asymptotic Agreement in Distributed Estimation. *IEEE Transactions on Automatic Control*, 27(3):650–655, June 1982.

[16] M. S. Branicky. Stability of hybrid systems: state of the art. In *Conference on Decision and Control*, pages 120–125, 1997.

[17] T. Brihaye. *Verification and control of o-minimal hybrid systems and weighted timed automata*. PhD thesis, Academie Universitaire Wallonie-Bruxelles, 2006.

[18] T. Brihaye and C. Michaux. On the expressiveness and decidability of o-minimal hybrid systems. *Journal of Complexity*, 21(4):447–478, 2005.

[19] A. Cataldo, C. Hylands, E. A. Lee, J. Liu, X. Liu, S. Neuendorffer, and H. Zheng. *Hyvisual: A Hybrid System Visual Modeler*. 2003.

[20] R. Chadha, A. Legay, P. Prabhakar, and M. Viswanathan. Complexity bounds for the verification of real-time software. In *Proceedings of the International Conference on Verification, Model Checking, and Abstract Interpretation*, pages 95–111, 2010.

[21] K. Chandy, B. Go, S. Mitra, C. Pilotto, and J. White. Verification of distributed systems with local-global predicates. *Formal Aspects of Computing*, pages 1–31, 2010.

[22] K. M. Chandy, S. Mitra, and C. Pilotto. Convergence verification: From shared memory to partially synchronous systems. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, volume 5215 of *LNCS*, pages 217–231, 2008.

[23] A. Chutinan and B. Krogh. Infinite state transition system verification using approximate quotient transition systems. *IEEE Transactions on Automatic Control*, 46:1401–1410, 2001.

[24] A. Chutinan and B. Krogh. Computational techniques for hybrid system verification. *IEEE Transactions on Automatic Control*, 48(1):64–75, 2003.

[25] A. Chutinan and B. H. Krogh. Verification of polyhedral-invariant hybrid automata using polygonal flow pipe approximations. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 76–90, 1999.

[26] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Abstraction and Counterexample-Guided Refinement in Model Checking of Hybrid Systems. *International Journal on Foundations of Computer Science*, 14(4):583–604, 2003.

[27] E. Clarke, A. Fehnker, Z. Han, B. Krogh, J. Ouaknine, O. Stursberg, and M. Theobald. Verification of Hybrid Systems Based on Counterexmple-Guided Abstraction Refinement. In *Proceedings of the International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 192–207, 2003.

[28] E. Clarke, O. Grumberg, S. Jha, Y. Lu, and H. Veith. Counterexample-Guided Abstraction Refinement. In *Proceedings of the International Conference on Computer Aided Verification*, pages 154–169, 2000.

[29] J. Corbett, M. Dwyer, J. Hatcliff, S. Laubach, C. Pasareanu, Robby, and H. Zheng. Bandera: Extracting finite-state models from Java source code. In *Proceedings of the International Conference on Software Engineering*, pages 439–448, 2000.

[30] P. J. L. Cuijpers. On bicontinuous bisimulation and the preservation of stability. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 676–679, 2007.

[31] P. J. L. Cuijpers and M. A. Reniers. Hybrid process algebra. *Journal of Logic and Algebraic Programming*, 62(2):191–245, 2005.

[32] T. Dang and O. Maler. Reachability analysis via face lifting. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 96–109, 1998.

[33] T. Dang and O. Maler. The d/dt tool for verification of hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*, pages 365–370, 2002.

[34] T. Dang, O. Maler, and R. Testylier. Accurate hybridization of nonlinear systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 11–20, 2010.

[35] A. Deshpande, A. Göllü, and P. Varaiya. Shift: A formalism and a programming language for dynamic networks of hybrid automata. In *Hybrid Systems*, pages 113–133, 1996.

[36] H. Dierks, S. Kupferschmid, and K. Larsen. Automatic Abstraction Refinement for Timed Automata. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, pages 114–129, 2007.

[37] L. Doyen, T. A. Henzinger, and J. franois Raskin. Automatic rectangular refinement of affine hybrid systems. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, pages 144–161. Springer, 2005.

[38] V. K. Dzyadyk. *Approximation methods for solutions of differential and integral equations.* VSP, Utrecht, The Netherlands, 1995.

[39] A. Fehnker, E. Clarke, S. Jha, and B. Krogh. Refining Abstractions of Hybrid Systems using Counterexample Fragments. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 242–257, 2005.

[40] A. Fehnker and F. Ivancic. Benchmarks for hybrid systems verification. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 326–341, 2004.

[41] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In *Hybrid Systems*, pages 258–273, 2005.

[42] G. Frehse, C. Le Guernic, A. Donzé, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*, 2011 (to appear).

[43] P. Fritzson. *Principles of Object-Oriented Modeling and Simulation with Modelica 2.1.* Wiley-IEEE Computer Society Pr, 2003.

[44] R. Gentilini. Reachability problems on extended o-minimal hybrid automata. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, pages 162–176, 2005.

[45] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 291–305, 2005.

[46] A. Girard and C. Guernic. Zonotope/Hyperplane intersecion for hybrid systems reachability analysis. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 215–228, 2008.

[47] A. Girard, A. A. Julius, and G. J. Pappas. Approximate simulation relations for hybrid systems. *Discrete Event Dynamic Systems*, 18(2):163–179, 2008.

[48] A. Girard and G. J. Pappas. Approximate bisimulation relations for constrained linear systems. *Automatica*, 43(8):1307–1317, 2007.

[49] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 201–214, 2008.

[50] S. Graf and H. Saidi. Construction of abstact state graphs with PVS. In *Proceedings of the International Conference on Computer Aided Verification*, pages 72–83, 1997.

[51] C. Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Proceedings of the International Conference on Computer Aided Verification*, pages 540–554, 2009.

[52] T. Henzinger, P. Ho, and H. W. Toi. Algorithmic analysis of nonlinear hybrid systems. *IEEE Transactions on Automatic Control*, 43:540–554, 1998.

[53] T. Henzinger, R. Jhala, R. Majumdar, and G. Sutre. Lazy Abstraction. In *Proceedings of the ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages*, pages 58–70, 2002.

[54] T. Henzinger, P. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? In *Proceedings of the ACM Symposium on Theory of Computation*, pages 373–382, 1995.

[55] T. A. Henzinger. Masaccio: A formal model for embedded components. In *IFIP International Conference on Theoretical Computer Science*, pages 549–563, 2000.

[56] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Proceedings of the International Conference on Computer Aided Verification*, volume 1254 of *LNCS*, pages 460–483, 1997.

[57] G. Holzmann and M. Smith. Automating software feature verification. *Bell Labs Technical Journal*, 5(2):72–87, 2000.

[58] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *IEEE Transactions on Automatic Control*, 48(6):988–1001, 2003.

[59] H. F. Jean-Pierre Aubin. *Set-valued Analysis*. Boston : Birkhuser, 1990.

[60] S. Jha, B. Krogh, J. Weimer, and E. Clarke. Reachability for linear hybrid automata using iterative relaxation abstraction. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 287–300, 2007.

[61] A. Kanade, R. Alur, F. Ivancic, S. Ramesh, S. Sankaranarayanan, and K. C. Shashidhar. Generating and analyzing symbolic traces of simulink/stateflow models. In *Proceedings of the International Conference on Computer Aided Verification*, pages 430–445, 2009.

[62] D. K. Kaynar, N. A. Lynch, R. Segala, and F. W. Vaandrager. Timed I/O Automata: A Mathematical Framework for Modeling and Analyzing Real-Time Systems. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 166–177, 2003.

[63] H. K. Khalil. *Nonlinear Systems*. Prentice-Hall, Upper Saddle River, NJ, 1996.

[64] A. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 202–214, 2000.

[65] G. Lafferriere, G. Pappas, and S. Sastry. O-minimal Hybrid Systems. *Mathematics of Control, Signals, and Systems*, 13(1):1–21, 2000.

[66] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 137–151, 1999.

[67] R. Lanotte and S. Tini. Taylor approximation for hybrid systems. *Information and Computation*, 205(11):1575–1607, 2007.

[68] D. Lee and M. Yannakakis. Online Minimization of Transition Systems (Extended Abstract). In *Proceedings of the ACM Symposium on Theory of Computation*, pages 264–274, 1992.

[69] D. Liberzon. *Switching in Systems and Control*. Boston : Birkhuser, 2003.

[70] G. Lorentz. *Bernstein Polynomials*. University of Toronto Press, 1953.

[71] N. Lynch, R. Segala, and F. Vaandrager. Hybrid I/O automata. *Information and Computation*, 185:105–157, August 2003.

[72] R. Milner. *Communication and Concurrency*. Prentice-Hall, Inc, 1989.

[73] I. Mitchell and C. Tomlin. Level set methods for computation in hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 310–323, 2000.

[74] S. Mitra and K. M. Chandy. A formalized theory for verifying stability and convergence of automata in PVS. In *Theorem Proving in Higher Order Logics*, pages 230–245, 2008.

[75] C. A. Muoz, G. Dowek, and V. Carreo. Modeling and verification of an air traffic concept of operations. In *Proceedings of International Symposium on Software Testing and Analysis*, pages 175–182, 2004.

[76] V. Mysore, C. Piazza, and B. Mishra. Algorithmic Algebraic Model Checking II: Decidability of Semi-algebraic Model Checking and Its Applications to Systems Biology. In *Proceeding of the International Symposium on Automated Technology for Verification and Analysis*, pages 217–233, 2005.

[77] S. Nadjm-tehrani and J. erik Stromberg. Formal verification of dynamic properties in an aerospace application. In *Formal Methods in System Design*, 1999.

[78] R. Olfati-Saber. Flocking for multi-agent dynamic systems: algorithms and theory. *IEEE Transactions on Automatic Control*, 51(3):401–420, March 2006.

[79] R. Olfati-saber, J. A. Fax, and R. M. Murray. Consensus and cooperation in networked multi-agent systems. In *Proceedings of the IEEE*, page 2007, 2007.

[80] Y. Pang, M. P. Spathopoulos, and H. Xia. Reachability and optimal control for linear hybrid automata: A quantifier elimination approach. *IJC*, 80(5):731–748, May 2007.

[81] G. E. Parker and J. S. Sochacki. Implementing the picard iteration. *Neural, Parallel, and Scientific Computations*, (4):97–112, 1996.

[82] C. Piazza, M. Antoniotti, V. Mysore, A. Policriti, F. Winkler, and B. Mishra. Algorithmic Algebraic Model Checking I: Challenges from Systems Biology. In *Proceedings of the International Conference on Computer Aided Verification*, pages 5–19, 2005.

[83] C. E. Picard. *Traite D'Analyse*, volume 3. Guthier-Villars, Paris, France, 1922-28.

[84] A. Platzer and E. Clarke. Computing differential invariants of hybrid systems as fixedpoints. Technical Report CMU-CS-08-103, Pittsburg, PA, February 2008.

[85] A. Platzer and E. M. Clarke. The image computation problem in hybrid systems model checking. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 473–486, 2007.

[86] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. In *Proceedings of the International Conference on Computer Aided Verification*, pages 176–189, 2008.

[87] A. Platzer and J.-D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *International Joint Conference on Automated Reasoning*, pages 171–178, 2008.

[88] P. Prabhakar, S. Mitra, and M. Viswanathan. On convergence of concurrent systems under regular interactions. In *International Conference on Concurrency Theory*, pages 527–541, 2009.

[89] P. Prabhakar and M. Viswanathan. A dynamic algorithm for approximate flow computations. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 133–143, 2010.

[90] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. Dullerud. Verifying tolerant systems using polynomial approximations. In *Proceedings of the IEEE Real-Time Systems Symposium*, pages 181–190, 2009.

[91] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud. A decidable class of planar linear hybrid systems. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 401–414, 2008.

[92] A. Puri, V. Borkar, and P. Varaiya. ϵ-approximation of differential inclusions. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 362–376, 1995.

[93] A. Puri, V. Borkar, and P. Varaiya. ϵ-Approximation of differential inclusions. In *Proceedings of the International Conference on Hybrid Systems: Computation and Control*, pages 362–376, 1996.

[94] E. Y. Remez. *On the determination of polynomial approximations of a given degree*, volume 10. 1934.

[95] W. Rudin. *Principles of Mathematical Analysis*. McGraw-Hill, 1976.

[96] B. S. Dmonstration du thorme de weierstrass fonde sur le calcul des probabilities. *Communications of the Mathematical Society*, 13:1–2, 1912.

[97] M. Segelken. Abstraction and Counterexample-guided Construction of Omega-Automata for Model Checking of Step-discrete linear Hybrid Models. In *Proceedings of the International Conference on Computer Aided Verification*, pages 433–448, 2007.

[98] M. Sorea. Lazy approximation for dense real-time systems. In *Proceedings of Formal Modeling and Analysis of Timed Systems*, pages 363–378, 2004.

[99] A. Tarski. *A Decision Method for Elementary Algebra and Geometry.* University of California Press, 2nd edition, 1951.

[100] A. Tiwari. Abstractions for hybrid systems. *Formal Methods in System Design*, 32(1):57–83, 2008.

[101] S. Tripakis, C. Sofronis, P. Caspi, and A. Curic. Translating discrete-time simulink to lustre. *ACM Transactions on Embedded Computing Systems*, 4:779–818, November 2005.

[102] J. N. Tsitsiklis. On the stability of asynchronous iterative processes. *Mathematical Systems Theory*, 20(2-3):137–153, 1987.

[103] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. Dullerud. STORMED hybrid systems. In *Proceedings of the International Colloquium on Automata, Languages and Programming*, pages 136–147, 2008.

[104] S. Yovine. Kronos: A verification tool for real-time systems. (kronos user's manual release 2.2). *Software Tools for Technology Transfer*, 1:123–133, 1997.