DYNAMIC ANALYSIS OF CYBER-PHYSICAL SYSTEMS

BY

PARASARA SRIDHAR DUGGIRALA

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, 2015

Urbana, Illinois

Doctoral Committee:

Associate Professor Mahesh Viswanathan, Chair
Associate Professor Sayan Mitra, Co-Chair
Professor Jose Meseguer
Professor Rajeev Alur, University of Pennsylvania

Abstract

With the recent advances in communication and computation technologies, integration of software into the sensing, actuation, and control is common. This has lead to a new branch of study called Cyber-Physical Systems (CPS). Avionics, automotives, power grid, medical devices, and robotics are a few examples of such systems. As these systems are part of critical infrastructure, it is very important to ensure that these systems function reliably without any failures. While testing improves confidence in these systems, it does not establish the absence of scenarios where the system fails. The focus of this thesis is on formal verification techniques for cyber-physical systems that prove the absence of errors in a given system. In particular, this thesis focuses on *dynamic analysis* techniques that bridge the gap between testing and verification.

This thesis uses the framework of hybrid input output automata for modeling CPS. Formal verification of hybrid automata is undecidable in general. Because of the undecidability result, no algorithm is guaranteed to terminate for all models. This thesis focuses on developing heuristics for verification that exploit sample executions of the system. Moreover, the goal of the dynamic analysis techniques proposed in this thesis is to ensure that the techniques are sound, i.e., they always return the right answer, and they are relatively complete, i.e., the techniques terminate when the system satisfies certain special conditions. For undecidable problems, such theoretical guarantees are the strongest that can be expected out of any automatic procedure. This thesis focuses on safety properties, which require that nothing bad happens. In particular we consider invariant and temporal precedence properties; temporal precedence properties ensure that the temporal ordering of certain events in every execution satisfy a given specification.

This thesis introduces the notion of a discrepancy function that aids in dynamic analysis of CPS. Informally, these discrepancy functions capture the convergence or divergence of continuous behaviors in CPS systems. In control theory, several proof certificates such as contraction metric and incremental stability have been proposed to

capture the convergence and divergence of solutions of ordinary differential equations. This thesis establishes that discrepancy functions generalize such proof certificates. Further, this thesis also proposes a new technique to compute discrepancy functions for continuous systems with linear ODEs from sample executions.

One of the main contributions of this thesis is a technique to compute an over-approximation of the set of reachable states using sample executions and discrepancy functions. Using the reachability computation technique, this thesis proposes a safety verification algorithm which is proved to be sound and relatively complete. This technique is implemented in a tool called, Compare-Execute-Check-Engine (C2E2) and experimental results show that it is scalable.

To demonstrate the applicability of the algorithms presented, two challenging case studies are analyzed as a part of this thesis. The first case study is about an alerting mechanism in parallel aircraft landing. For performing this case study, the dynamic analysis presented for invariant verification is extended to handle temporal properties. The second case study is about verifying key specification of powertrain control system. New algorithms for computing discrepancy function were implemented in C2E2 for performing this case study. Both these case studies demonstrate that dynamic analysis technique gives promising results and can be applied to realistic CPS.

For distributed CPS implementations, where message passing, and clocks skews between agents make formal verification difficult to scale, this thesis presents a dynamic analysis algorithm for inferring global predicates. Such global predicates include assertions about the physical state and the software state of all the agents involved in distributed CPS. This algorithm is applied to coordinated robotic maneuvers for inferring safety and detecting deadlock.

To my parents, for their love and support.

Acknowledgments

This dissertation would not have been possible without the support and interactions of many others. First and foremost, I would like to thank my advisers Mahesh Viswanathan and Sayan Mitra. They have patiently helped me in every aspect of research, from choosing a research problem, methods to develop new solutions, and presenting the results. I am immensely grateful to them for giving me freedom to explore and learn new topics, and pursue research at my own pace. Their attention to clarity and detail allowed me to greatly improve my writing and presentation skills, and mature as an researcher. Special thanks to Mahesh Viswanathan for teaching me about the connections between logic and computer science and to Sayan Mitra for introducing me to verification of hybrid systems.

Thanks to Prof. Jose Meseguer, for not only being a mentor to me through out grad school and providing valuable advice, but also for being a part of my PhD committee. I would like to thank Prof. Rajeev Alur for being a part of my PhD committee and providing me inputs regrading the future directions of research.

I would like to thank Nitin Vaidya, Elsa Gunter, Jeff Erickson, Grigore Rosu, Gul Agha, Daniel Liberzon, Marco Caccamo and many others for having taught me courses that helped me immensely with research. I would like to thank Madhusudan Parthasarathy on his advice on various matters. I would like to thank Natrajan Shankar, Rakesh Kumar, Romit Roy Choudhury, and Pete Sauer for their valuable advice on career.

I would like to thank Aarti Gupta, Franjo Ivancic, and Khalil Ghorbal for allowing me to spend summer at NEC and Ashish Tiwari for acting as a mentor during my intrenship at SRI. I also thank Cesar Munoz and Le Wang for being a collaborator on verification of ALAS protocol. I would like to thank Adam Zimmerman for his help regarding StarL programing language. I would like to thank Matthew Potok and Bolun Qi for implementing various modules in developing C2E2.

I am glad to have interacted with Pavithra Prabhakar, Vijay Anand Reddy, Shruti

Bandhakavi, Rajesh Karmani, Taylor Johnson, and Stanley Bak, who gave me valuable insights about research and other aspects of life at graduate school. Many thanks to my lab mates Jayanand Ashok Kumar, Karthik Manamcheri, Aparna Sundar, Edgar Pek, Pranav Garg, Milos Gligorich, Dileep Kini, Zhenqi Huang, Debjit Pal, Adel Ahmadyan, Hongxu Chen, Chuchu Fan, Ritwika Ghosh, and others for providing me with an excellent working environment.

My stay at Champaign-Urbana has been delightful, thanks to my roommates, Virajith Jalaparti, Aditya Pabbaraju, and Naveen Cherukuri. Special thanks to Anirudh Vemula and Prannoy Suraneni for their hospitality in helping me get accustomed to life in Champaign. I would like to thank my friends Sneha, Sravanti, Rimpa, Varun, Foggy, Som, Preeti, Abhishek, Prakalp, Mayank, Ankur, Vivek, Ravi, Sonal, Sibin, Radha, Rekha, Sushmita, Bhushan, Neetika, Nisha, Spurti, Megha, Sarah, Jay, Gavin, Sandeep, Komal, Avinash, Bhargav, Raghavendra, Piyush, Neha, Rajhans, Raylan, and many others. Thank you all for all the great times.

This thesis would not have been possible without the love and support of my family. I am thankful to my parents for supporting my decision to pursue grad school and my aunts Manjula and Madhuri for making me feel like I am at home whenever I visited them. Special thanks to my brother Srinath and uncle Pallamraju for providing me with the most helpful advice at the times when I needed it the most. I dedicate this thesis to all my family members and especially to my niece Vidhatri. Finally, I would like to thank everyone else with whom I interact, as I'm sure I've failed to mention everyone explicitly.

Table of Contents

List of Tables

List of Figures

List of Algorithms

Chapter 1

Introduction

The hardware revolution in the past few decades has reduced the size of transistors exponentially and enabled the manufacturing of complex integrated circuits with billions of transistors. This has lead to the integration of software in various walks of life such as mobile phones, sensor networks, and embedded control systems. Recent developments in communication technologies such as Bluetooth, NFC, WiFi, and the Internet have enabled coordination among these software agents. Integrating software and communication technologies in traditional disciplines such as transportation, avionics, power grids, and medicine is expected to provide huge social and environmental benefits. This, in turn, lead to a new field of study called Cyber-Physical Systems (CPS).

Cyber-physical systems have two major characteristics. First, CPS interact with the physical environment, i.e., they sense and control physical quantities such as voltages in power grids, velocity of cars, etc. Control theory refers to the branch of science that studies such continuous systems. It focuses on designing controllers such that the system meets the desired qualities such as stability, robustness, disturbance rejection, etc. The second characteristic of CPS is that they are controlled by software. Unlike control theory, software is studied as a discrete transition system where the state of software evolves in discrete steps. As the study of CPS involves both continuous behaviors and discrete transitions, it is an interdisciplinary field at the intersection of control theory and computer science.

Many safety critical systems often involve both physical and software components. Examples of such systems are air-traffic control protocols, smart grids, autonomous cars, medical devices, etc. Faulty behavior of such systems can have catastrophic consequences such as loss of property and in extreme cases, life. Hence, it is very important to ensure that these CPS systems do not have any undesired behaviors (or bugs). While rigorous testing of CPS can help identify bugs, it does not prove their absence. One way to ensure that the system does not have any bugs is to perform

formal verification. Formal verification is a branch of science that checks all possible behaviors of the system and proves the absence of bugs.

One of the prerequisites for performing formal verification is that the system model and specification must be expressed in a mathematical framework with precise semantics. Given a model of a system and its specification, formal verification, by checking all possible behaviors of the system, either proves that the the specification is satisfied or presents a behavior that violates the specification. For example, to verify a specification that a *power grid system never fails*, the following steps are performed. First, we develop a mathematical model of a power grid, then, we define the set of behaviors that lead to *failed* states, and finally, prove that the set of behaviors that lead to failed states are not a part of the mathematical model. *In this thesis, we develop new techniques for verifying invariant and temporal properties of CPS.*

1.1 Challenges In Verification Of CPS

CPS involves interaction between continuous nature of physical environment and the discrete nature of software, which makes their verification challenging. Software verification, which establishes whether a given software satisfies a given specification is, in general, undecidable. Verification of CPS where software interacts with physical environment is hence, in general, undecidable. Although verifying properties of finite state discrete transition systems is known to be decidable, it suffers from *state space explosion* problem.

Verification of timed automata [13], which models the behavior of real-time systems, is shown to be decidable and is PSPACE-complete. Verification of rectangular hybrid automata (RHA), a modest extension of timed automata that generalizes the behavior of physical environment to rectangular dynamics, i.e., the rate of evolution of a continuous variable is an interval $\dot{x} \in [a, b]$, is shown to be undecidable in [88]. Although, verification of special classes of RHA were shown to be decidable in [149, 21, 150, 44], such classes impose restrictions on the nature of discrete transitions and are not suitable for modeling realistic CPS.

For CPS where the evolution of physical environment is defined as linear ordinary differential equation (ODE), decidability is only known for very restrictive class of systems [109, 147, 57]. Such results impose restrictions on the structure of ODEs and in some cases also on the property of interest. One of the main challenges

in verifying CPS where physical environment is modeled as linear ODEs is that the system behavior would involve either exponential or trigonometric functions (or both) and analysis of such systems is not known to be decidable. If the physical environment is specified as a nonlinear ODE, decidability results are not known. This is mainly because the closed form expression for solution of a nonlinear ODE may not exist and hence one has to largely rely on numerical techniques for verification.

Another aspect of CPS verification is that a bounded uncertainty in the initial set of a physical environment would lead to an uncountable number of behaviors. Hence, verifying such systems would require checking whether all of these uncountably infinite number of executions satisfy the specification or not. Searching for an execution that violates the specification within the uncountable number of executions would be like looking for a needle in a haystack. All these factors, i.e., interface between continuous and discrete components, complexity of physical environment, and uncountably infinite number of executions makes verification of CPS challenging.

1.2 Thesis Overview

For formally verifying whether a given CPS satisfies its specification, this thesis proposes new *dynamic analysis* techniques. Dynamic analysis, broadly, refers to a class of techniques that infer properties of a system from its sample executions. While *testing* these CPS systems involves checking whether sample behaviors of CPS satisfy the specification, dynamic analysis extends testing by performing formal analysis on the sample behaviors and then infers a property about all possible behaviors of the system. In practice, many CPS involve interaction with a physical environment that evolves according to an ordinary differential equation (ODE). In general, a closed form solution of an ODE does not exist and hence, one has to rely on numerical simulations to understand the behaviors of such systems, to a large extent. This makes dynamic analysis an attractive framework for verifying CPS as it can be applied to physical environments described by nonlinear ODEs and also because it can be easily integrated into the testing procedures, which are the standard industrial practice.

For continuous systems, neighboring trajectories get arbitrarily close to each other as the distance between the initial states of the trajectories vanishes. The dynamic analysis technique presented in this thesis exploits this continuity property. In Chapter 3, we introduce *discrepancy function* that formally expresses this continuity prop-

erty. We also prove that discrepancy functions generalize several proof certificates developed in control theory for establishing convergence or divergence of neighboring trajectories. Our technique requires formal models of CPS to be equipped with discrepancy functions.

The dynamic analysis technique for continuous systems presented in this thesis (Chapter 3) generates sample simulations and uses discrepancy functions to compute an overapproximation of all the behaviors, which can then be used for verifying whether the system satisfies the specification or not. In Chapter 4 we extend this dynamic analysis technique to CPS where the continuous state changes as a result of interaction with software. We also present the tool Compare-Execute-Check-Engine (C2E2) that implements the dynamic analysis technique. Several aspects of the tool such as the tool architecture, input and output format, and user experience are also discussed in Chapter 4.

To demonstrate the promise of the dynamic analysis technique presented in this thesis, we perform two case studies of realistic systems and present the verification results in Chapter 5. The first case study is an alerting system in a parallel aircraft landing protocol. In a parallel landing scenario, the alerting mechanism is supposed to be designed in such a way that an alert will be issued before the landing aircraft get too close to each other. The alerting mechanism designed by NASA has complex nonlinear functions and hence presents a challenge for verification. In this thesis, we overcome this challenge by presenting a new dynamic analysis algorithm and verify the alerting mechanism.

The second case study considered is a powertrain control system, which controls the engine subsystem in an automotive. The main goal of this system is to control the fuel/air ratio in the engine around an optimal value to achieve maximum fuel efficiency. The evolution of continuous variables in powertrain control system is given by a complex nonlinear ODE, making the analysis challenging. In this thesis, we implement new techniques for computing discrepancy function to overcome this challenge and present promising verification results.

While verifying distributed systems is challenging because of nondeterminism in communication, message delays, and concurrency, distributed CPS additionally have the complexity of interaction with a physical world. In Chapter 6, we present a new dynamic analysis technique for distributed CPS that helps in debugging implementations. This brings together the techniques presented in Chapters 3 and 4 and the techniques used in inferring global predicates in distributed systems. We ver-

ify various applications of a distributed robotic systems such as way-point tracking, geo-receive, and light painting, that are implemented in StarL framework.

1.3 Literature Review

Verification of CPS has enjoyed research attention for the past 20 years (check proceedings of HSCC [83, 62, 29] for recent developments). In the rest of this section, we present an overview of several modeling frameworks for CPS, the properties of interest, related research work, and the contributions of this thesis in the context of existing literature.

1.3.1 Modeling Cyber Physical Systems

The mixture of continuous and discrete behaviors in a CPS requires new modeling frameworks. A few popular frameworks for modeling CPS are Hybrid Automata [14, 15], Hybrid Programs [139, 140, 141], and Hybrid Input/Output Automata (HIOA) [121, 127]. These modeling frameworks differ from each other in the complexity of the physical environment and the notation used to describe the behaviors of CPS. Time automata [13] was first proposed as a framework to model the behavior of real time systems. The passing of time is modeled using *clock variables* that are either *on* or *off*. The software in real time system is modeled as a finite state machine that interacts with these clock variables. Hybrid automata [15] are extensions of timed automata to handle more involved behaviors of continuous variables. In a hybrid automaton, the continuous behaviors are not restricted to clocks, but can evolve according to differential equations. In this thesis, we use the Hybrid Input/Output Automata (HIOA) framework proposed in [121, 127]. HIOA uses trajectories that are real valued functions of time for modeling the evolution of physical environment in a CPS.

Hybrid automata have been extensively used in modeling many physical processes such as voltage behavior in circuits [9], biological systems such as heart [136, 85] and pancreas [152], and avionics systems such as aircraft behaviors [115].

There are several industrial scale tools available for modeling and developing/designing CPS. One of the most commonly used toolboxes is Simulink/Stateflow [4, 3] toolbox provided by Mathworks. Alternative modeling tools used in the industry are

Ptolemy [34] and Modelica [68]. While Ptolemy and Modelica have concrete semantics, more widely used Simulink/Stateflow toolbox does not have a formal semantics. In this thesis, we focus our attention on Simulink/Stateflow models and interpret them as hybrid input output automata [162].

1.3.2 Specification

System specification describe the requirements for a system. Although such specifications are typically given in natural language, for performing formal verification by rigorously checking all possible behaviors, one requires a formal semantics for each specification. Temporal logics introduce constructs that help express the natural language specification into logical formulas and are used extensively in formal verification. Linear Temporal Logic (LTL) [144] and Computational Tree Logic (CTL) [39] are two popular temporal logic frameworks for describing specification for hardware and software systems. As CPS have both continuous and discrete behaviors, new temporal logics such as Signal Temporal Logic (STL) [122] and Metric Temporal Logic (MTL) [107] were proposed to express specification in CPS domains. Formulas in these logics express requirements on behaviors of CPS, not just on the continuous behaviors, but also on the interaction with software.

Two important classes of specification widely studied in CPS verification are safety and liveness properties. A safety property (sometimes referred as invariant property) is said to be satisfied by a system if all the behaviors of the system remain in a set that is defined to be *safe*. One variant of safety property that we investigate in this thesis is *bounded time safety property*, where we consider executions of CPS for only bounded time. Typically, the *safe* set of states for a safety property are defined as the compliment of *unsafe* set, hence verifying such properties would require checking that none of the behaviors of the system reaches the *unsafe* set.

Liveness specification are specially important for reactive CPS, i.e., where all the behaviors run for unbounded time. An example liveness property is *region stability* which says that eventually all the behaviors of the system reach a given set of states reach a particular region and stay there. Other stability notions such as *Lyapunov stability* and *exponential stability* are commonly studied in control theory and are part of any standard text book such as [89, 105]

1.3.3 Relevant Results In Verification Of Cyber-Physical Systems

Verification of cyber-physical systems can be mainly categorized into two groups: algorithmic and proof theoretic. Proof theoretic verification requires using an automated theorem prover such as PVS [134] or Keymeara [143] and establishing that a hybrid automaton satisfies a specification by using a fixed set of axioms and proof rules such as in [128, 138, 35]. One commonly used technique to prove safety properties using proof theoretic techniques is to search for a candidate invariant (either using reachability analysis [76] or other heuristics [98, 161]) and then use the proof rules for establishing that the candidate invariant is indeed an invariant. Although proof theoretic techniques have been applied to verify moderately complex systems, they require domain expertise and manual effort.

Algorithmic verification techniques, on the other hand propose algorithms that take as input a hybrid automaton and a specification and return whether the specification is satisfied or not, by performing a pre-decided sequence of computations. This thesis proposes new algorithms for verification of CPS systems and hence falls into the second category.

A popular approach for verifying safety properties of CPS is *model checking* [42, 24]. Typical model checking approaches compute an overapproximation of all the states that can be reached by all the behaviors of a system. One of the main research questions in model checking CPS systems is to establish the decidability and complexity of these verification procedures. In [12] it was established that verifying an invariant property of timed automata is PSPACE-complete. These verification algorithms have been implemented in widely used tools such as UPPAAL [112] and Kronos [33]. In [88], undecidability of verifying invariant properties for a special type of hybrid automata called *rectangular hybrid automata* (RHA) was established and a model checker for verifying RHA was presented in [87]. Extending the dynamics of the system from intervals to linear ODEs gives us linear hybrid systems (sometimes also called affine hybrid systems). Decidability of very special fragments of linear hybrid systems was proved in [109, 147, 166, 57]. The decidability of hybrid systems with linear and nonlinear dynamics is still an open problem. However, δ-decidability of bounded time safety properties of hybrid systems was proved to be decidable in [71].

As realistic CPS systems do not often fall into the decidable classes, it is important to develop heuristics for general CPS models that are efficient and provide some theoretical guarantees. In the rest of this section we will present different approaches that have been used in developing scalable verification techniques for CPS.

Reachable Set Computation: Reachable set computation techniques compute the set of the states (or an overapproximation) reached by all the possible behaviors of the system. If this reachable set does not contain any state from the *unsafe* set of states, then we can conclude that the safety property is satisfied by the system. For timed automata and special classes of rectangular hybrid automata, the reachable set of states for unbounded time can be computed exactly. For the undecidable fragment of rectangular hybrid automata, as the safety verification problem is in general undecidable, the reachable set computation might not terminate. The efficiency of computing reachable set would depend on the data structure used for storing the reachable set. Discussion about efficiency of data structure representation for model checking timed automata can be found in [18, 111, 86].

For CPS with linear and nonlinear ODE, the decidability of reachable set computation is still an open problem. For physical dynamics described by linear ODEs, the continuous state of the system is given by matrix exponential. Computing matrix exponentials exactly is only possible with strict restrictions on the ODE and hence, a common approach for verifying such systems is to compute numerical overapproximations. As these numerical overapproximations cannot be computed for infinite time horizon, many of reachable set computation techniques for CPS with linear and nonlinear ODEs are restricted to *bounded time*. We now present an overview of some popular data structures or representations used in reachable set computation.

In Phaver [64, 67] the reachable set is represented as a convex polyhedron. As the number of vertices in a convex polyhedron might increase exponentially with respect to the number of equations (and dimensions), this representation does not work well for higher dimensional systems. Other data structures used include Zonotopes [77], Ellipsoides [31, 108], and Oriented Rectangular Hulls [159]. Support functions [113] have been recently proposed as a new data structure for computing reachable set. Support functions are equipped with a fixed point operator that helps in computing the reachable set of states for unbounded time. This data structure along with a few enhancements [66] is being used in SpaceEx [65], the current state of the art model checker for linear hybrid systems.

Unlike their linear counterparts, nonlinear ODEs might not have a closed form expression for the solution and hence numerical techniques are typically used in all reachable set computation techniques. One of the first steps in this direction was presented in [45]. Rigorous algebraic representation of sets were used for reachable set computation in [28]. Flow* [38] uses Taylor models and CORA [6] uses Polynomial

Zonotopes introduced in [5] for representing the reachable set. HyCreate [25] uses similar approach to [45], however performs optimizations in reducing error and can be applied for reachability analysis in real time [26]. In this thesis, we present a new representation of the reachable set called *reachtree* that we introduce in Chapter 3.

Abstractions and Approximations: The reachable set computation techniques presented earlier suffer from some disadvantages. First, is that as the number of dimensions increases, due to the *curse of dimensionality*, the number of steps for computing reachable set increases exponentially. Further, with increasing complexity of the differential equations or the interaction between the continuous and software systems, the efficiency of these techniques decreases. Hence, much research has been done on coming up with new abstractions and approximations for making the verification scalable.

Abstraction of a system, informally, is a simplification of the system that allows for more behaviors. If the abstraction of a system satisfies a property then the property is said to be satisfied by the system. However, the converse is not true, i.e., if the abstraction violates the property, the system might still satisfy the property. Counter Example Guided Abstraction Refinement (CEGAR) technique [41] has been a popular framework where a series of abstractions for a concrete system are constructed until an abstraction that satisfies the property is obtained or a violation of the property is observed. CEGAR technique has been applied to CPS verification in [40, 11]. Subsequent works that compute better abstractions and make the verification scalable were proposed in [151, 153, 163, 145]. Such abstraction refinement techniques are orthogonal to the dynamic analysis considered in this thesis. One can potentially develop new abstraction-refinement techniques using the tool C2E2 presented in Chapter 4 as the verification back end.

While reachable set computations for complex ODEs compute an overapproximation of the reachable set, a few other forms of approximations have also been considered in the literature. One body of work in this domain is approximate bisimulation [79, 81, 160]. These approaches compute a discrete approximation of the continuous behaviors of CPS and hence leverage the symbolic model checking [126] techniques for verification of discrete systems. Another branch of work in approximations is *hybridization* [19, 20]. In hybridization, the nonlinear ODE of the physical environment is approximated (while preserving soundness) as linear ODE in several regions of operation. This simplifies the reachability computation of nonlinear sys-

tems to more tractable computation for linear systems. Polynomial approximations of the behaviors of linear systems were also considered in [148, 146].

Dynamic Analysis: As explained earlier in this chapter, dynamic analysis refers to a class of techniques where the property is inferred to be satisfied by the system from only a sampled set of executions. This thesis primarily deals with dynamic analysis and hence we present a relatively more detailed overview of existing literature in dynamic analysis.

The first approach to bridge the gap between simulation and verification was proposed in [78]. The authors present a verification algorithm for continuous linear systems by sampling a few simulations and constructing a *Metric Transition System* [80]. This metric transition system gives us a finite state approximation of the reachable set which can then be used for invariant verification. Another approach for computing an overapproximation of the reachable set from sample simulations has been proposed in [51]. In [51], the authors leverage *sensitivity* for computing an overapproximation of the reachable set from a neighborhood. This requires the ordinary differential equation solver to be equipped with special procedures that compute sensitivity along a sample execution. It is shown that this overapproximation is sound for continuous behaviors that follow linear ordinary differential equations, but no such guarantee can be provided for systems with nonlinear ordinary differential equations.

Another important line of work in dynamic analysis is related to falsification techniques presented in [17, 133, 50]. Given a system as an executable model and specification given as a formula in metric temporal logic, these techniques generate sample executions and compute the robustness of these executions with respect to the specification provided. The search for executions that violate the specification is then performed using stochastic optimization techniques. The algorithm terminates with either an execution that falsifies the specification or does not find a counterexample within the given budget. Falsifying techniques hence only aid in bug finding but do not give a proof that the system satisfies the specification.

1.4 Contributions

In this thesis we bridge the gap between testing procedures that generate sample simulations and formal verification. There are two major advantages of the techniques proposed in this thesis compared to the other dynamic analysis techniques. First, while other dynamic analysis techniques work for linear ordinary differential equations, the techniques proposed in this thesis are also applicable to physical environment that follow nonlinear ordinary differential equations, which is a vastly more general class of systems. Second, while other reachable set computation techniques only guarantee soundness, we also guarantee relative completeness, which states that the dynamic analysis algorithm will always terminate if the system satisfies a specific condition. In this thesis, the condition for relative completeness is the robust satisfaction of the specification, which means that all the executions and their ϵ-perturbations also satisfy the specification. Such relative completeness guarantees are the strongest one can hope for given that in the invariant verification for general hybrid automata is undecidable.

To demonstrate the applicability of the proposed approach, we look at two verification case studies in this thesis. The first is a parallel aircraft landing protocol and second is a powertrain control system. We show that our approach can be extended to verify other temporal properties without losing the strong theoretical guarantees. Each of these case studies presents a unique challenge for verification and we present the advantages of dynamic analysis that made the verification feasible.

For distributed CPS implementations where uncertainties in message passing and clock drifts lead to state space explosion, we present a dynamic analysis technique for inferring the global predicates. For doing this, we bring together techniques used to infer global predicates in distributed systems and reachability computation techniques for hybrid systems. Such global predicates of distributed CPS systems help in inferring not just the software behavior of CPS implementations, but also the relation between the software and the physical state of the CPS.

1.5 Organization Of The Dissertation

We start with preliminaries in Chapter 2 which include the modeling framework of hybrid input output automata. In Chapter 3 we introduce the notion of discrepancy function, a certificate that generalizes other proof certificates from control theory that

first appeared in [55]. We also present a dynamic analysis algorithm for continuous systems and its theoretical guarantees. In Chapter 4 we present the extension of the dynamic analysis technique to general hybrid automata, that appeared in [56], and discuss the features of the tool C2E2 that implements the algorithm. In Chapter 5 we inspect the two verification case studies of parallel landing protocol (appeared in [58]) and powertrain control system(appeared in [53]) and present the challenges, extensions to the verification technique, and experimental results of the case studies. In Chapter 6 we present the technique for inferring global predicates of the distributed CPS implementations that appeared in [54]. We present our conclusions and directions for future research in Chapter 7.

Chapter 2

Preliminaries

This chapter introduces mathematical definitions and notations that are used in the rest of the thesis. In Section 2.1 we introduce the notations for vectors, sets, functions, and readers familiar with such notation may skip forward to Section 2.2, where we introduce the modeling formalism of Hybrid Input Output Automata (proposed in [127]) for modeling Cyber-Physical Systems.

2.1 Vectors, Sets, And Functions

For a vector $x \in \mathbb{R}^n$, $||x||$ denotes the ℓ^2 norm. For $x_1, x_2 \in \mathbb{R}^n$, $||x_1 - x_2||$ is the Euclidean distance between the points. A closed ball of radius δ centered at x_1 is defined as $B_\delta(x_1) \triangleq \{x \mid ||x_1 - x|| \leq \delta\}$. Unless otherwise stated, the default norm (denoted as $||\cdot||$) for elements in $\mathbb{R}^n$ is the ℓ^2 norm. Other norms such as ℓ^1 and ℓ^∞ are denoted as $||\cdot||_1$ and $||\cdot||_\infty$ respectively. Given a set S and a transition relation $R \subseteq S \times S$, the transitive closure of the relation is denoted as R^* and $(u, v) \in R^*$ if and only if $\exists u_1, u_2, \ldots, u_k \in S$ such that $(u, u_1) \in R$, $\forall 1 \leq i \leq k-1, (u_1, u_{i+1}) \in R$, and $(u_k, v) \in R$.

For a set $S \subseteq \mathbb{R}^n$, we denote its compliment as S^c. Given $\delta \geq 0$, the expansion of the set S by δ is defined as $B_\delta(S) = \cup_{x \in S} B_\delta(x)$. We will find it convenient to also define the notion of shrinking S by δ, i.e., for $\delta < 0$, and $S \subseteq \mathbb{R}^n$, $B_\delta(S) = \{x \in S \mid B_{-\delta}(x) \subseteq S\}$. Given a function $f : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$, the *$\delta$-sublevel set* of f around x, denoted as $B^f_\delta(x_0) \triangleq \{\, x \,|f(x, x_0) \leq \delta\}$.

A set S_1 is a *δ-overapproximation* of S_2, if $S_2 \subseteq S_1 \subseteq B_\delta(S_2)$. The maximum distance between two sets S_1 and S_2 is denoted as $dist_{max}(S_1, S_2) = sup_{x_1 \in S_1, x_2 \in S_2}\{||x_1 - x_2||\}$. For a bounded set S, a *δ-cover* of S is a finite collection of points $\mathcal{X} = \{x_i\}_{i=1}^m$ such that $S \subseteq \bigcup_{i=1}^m B_\delta(x_i)$. Its diameter $diameter(S) \triangleq \sup_{x_1, x_2 \in S} ||x_1 - x_2||$.

Given intervals I, I' over $\mathbb{R}$, the relation $I < I'$ holds iff $\forall u \in I, \forall u' \in I', u < u'$. Relation $I \leq I'$ holds iff $\forall u \in I, \exists u' \in I', u < u'$. Relations $>$ and $\geq$ are defined

similarly. For a real number b, $I - b = \{u - b \mid u \in I\}$. Subtraction operation over intervals is defined as, $I - I' = \{u - u' \mid v \in I, v' \in I'\}$. $I \times I' = \{u \times u' \mid u \in I, u' \in I'\}$.

A *predicate* over $\mathbb{R}^n$ is a computable function $P : \mathbb{R}^n \to \{\top, \bot\}$ that maps each state in $\mathbb{R}^n$ to either $\top$ (true) or $\bot$ (false). A predicate P is associated with the set $[[P]] = \{x \mid P(x) = \top\}$. When it is clear from context, we will overload the symbolic name of the predicate P to also denote its associated set $[[P]]$.

A *guarantee predicate* [124] $P(x)$ is a predicate of the form $[[P]] = \{\ x|\ \exists t > 0, f_p(x,t) > 0\}$, where $f_p : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is called a *lookahead function*. A guarantee predicate holds at a state x if there exists some future time t at which $f_p(x,t) > 0$ holds. Using a quantifier elimination procedure, a guarantee predicate can be reduced to an ordinary predicate without the existential quantifier. However, this can be an expensive operation, and more importantly, it is only feasible for restricted classes of real-valued lookahead functions with explicit closed form definitions.

A continuous function $f : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ is *smooth* if all its higher derivatives and partial derivatives exist and are also continuous. It has a Lipschitz constant $K \geq 0$ if for every $x_1, x_2 \in \mathbb{R}^n$, $||f(x_1) - f(x_2)|| \leq K||x_1 - x_2||$.

A non-negative function $g : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is a *class $\mathcal{K}$ function* if $g(x) \geq 0$ for $x \neq 0$, $g(0) = 0$ and $g(x) \to 0$ as $x \to 0$. A class $\mathcal{K}$ function g is called $\mathcal{K}_\infty$ if $g(x) \to \infty$ as $x \to \infty$. For example, the function $f(x) = \frac{x}{1+x}$ belongs to class $\mathcal{K}$ but not to $\mathcal{K}_\infty$. A function $g : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ is called a $\mathcal{KL}$ function, if and only if (1) for each $t \in \mathbb{R}_{\geq 0}$, $g_t(x) \triangleq g(x,t)$ is a $\mathcal{K}$ function and (2) for each $x \in {\mathbb{R}_{\geq 0}}^n$, $g_x(t) \triangleq g(x,t) \to 0$ as $t \to \infty$

2.1.1 Matrices

For an $n \times n$ matrix $A \in \mathbb{R}^{n \times n}$, the *norm* of matrix, denoted as $||A|| = \max_{x:||x||=1} ||x^T A x||$. A is *positive semi-definite*, written as $A \succeq 0$, if $\forall x \in \mathbb{R}^n$, $x^T A x \geq 0$. It is *positive definite*, $A \succ 0$, if the previous inequality is strict. It is *negative (semi) definite* if $-A$ is positive (semi) definite. A *matrix function* $\mathbf{M} : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{m \times m}$ maps a state $x \in \mathbb{R}^n$ and a time t to a matrix $\mathbf{M}(x,t)$. $\mathbf{M}$ is called an *uniform metric* if $\forall x \in \mathbb{R}^n, \forall t > 0$, $\mathbf{M}(x,t)$ is symmetric, positive definite and $\forall y \in \mathbb{R}^m, \exists\ v_2, v_1 \in \mathbb{R}$ such that $0 < v_2 < v_1$, $v_2 \cdot y^T y \leq y^T \mathbf{M}(x,t) y \leq v_1 \cdot y^T y$. Given a matrix M, its matrix exponential e^M is defined as

$$e^M = I + M + \frac{1}{2!}M^2 + \frac{1}{3!}M^3 + \ldots$$

2.2 Continuous And Hybrid System Models

Hybrid automata framework is proposed as a modeling framework (in [15]) for Cyber-Physical Systems that interact with the physical environment and are controlled by software. An executions of a hybrid automaton is a sequence of continuous trajectories and discrete transitions where the continuous trajectories model the evolution of the real valued variables with time (the physical parameters), and the discrete transitions model the evolution of software in discrete steps. In this section, we introduce the syntax, definitions, and semantics of Hybrid Input Output Automata (HIOA) framework proposed in [127].

2.2.1 Continuous Systems

Let us denote the set of all the real valued variables in the model as the set $\mathcal{V}$. For continuous systems, all the variables are real valued and hence the set of all values the variables can take, denoted as $val(\mathcal{V}) = \mathbb{R}^n$. A continuous behavior of the system is modeled as a *trajectory*. A trajectory τ is defined as a function $\tau : dom \rightarrow val(\mathcal{V})$ where dom is either $[0, T]$ for some $T > 0$ or is $[0, \infty)$ defines the evolution of the system with time. The domain of the function is referred as $\tau.dom$ and $\tau.dur = T$ if $dom = [0, T]$ and is ∞ otherwise. The state of the system along the trajectory at time t is defined as $\tau(t)$. Given a function $f : \mathbb{R}^n \rightarrow \mathbb{R}^n$, a trajectory τ is said to be a *solution* of the ordinary differential equation $\dot{x} = f(x)$ if $\forall t \in \tau.dom$, $\frac{d}{dt}\tau(t) = f(\tau(t))$. When f is a nice[1] function, the ODE has a unique solution for a given initial state and a duration. With different initial states and time bounds, an ODE defines a set of trajectories. For such a vector function $f(x) = \begin{pmatrix} f_1(x) \\ \vdots \\ f_n(x) \end{pmatrix}$, its Jacobian J_f is given as $\begin{pmatrix} \frac{\partial f_1}{\partial x_1} & \cdots & \frac{\partial f_1}{\partial x_n} \\ \vdots & \cdots & \vdots \\ \frac{\partial f_n}{\partial x_1} & \cdots & \frac{\partial f_n}{\partial x_n} \end{pmatrix}$. The Jacobian of f at x approximates the rate of change of each component of $f(x)$, with respect to changes in x.

[1] For example, Lipschitz continuous or smooth. A continuous function $f : \mathbb{R}^n \times \mathbb{R} \rightarrow \mathbb{R}$ is *smooth* if all its higher derivatives and partial derivatives exist and are also continuous. It has a Lipschitz constant $K \geq 0$ if for every $x_1, x_2 \in \mathbb{R}^n$, $||f(x_1) - f(x_2)|| \leq K||x_1 - x_2||$.

A continuous system $\mathcal{C}$ is denoted by the tuple $\langle \mathcal{V}, \mathcal{T} \rangle$ where $\mathcal{V}$ is the set of variables and $\mathcal{T}$ is a set of trajectories that are closed under the prefix, suffix, and concatenation operations. In this thesis, we restrict our attention to *deterministic* continuous systems. That is, if two trajectories τ_1 and τ_2 start from the same initial state, i.e., $\tau_1.\mathsf{fstate} = \tau_2.\mathsf{fstate}$, then either τ_1 is a prefix of τ_2 or vice versa. We say that a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ is defined by an ODE $\dot{x} = f(x)$ if and only if all the solutions of the ODE are in $\mathcal{T}$. In this thesis, we assume that the solution for the initial value problem of the ODE $\dot{x} = f(x)$ exists, and is unique, and hence $\mathcal{T}$ is deterministic. In this thesis, we are interested in only bounded time properties of continuous systems, i.e., given a time bound T, we consider trajectories τ such that $\tau.dom = [0, T]$. The set of reachable states of a continuous system starting from an initial set Θ for bounded time T is defined as follows:

Definition 1 *Given a continuous system $\mathcal{C} \triangleq \langle \mathcal{V}, \mathcal{T} \rangle$, initial set Θ, and time bound T, the set of reachable states of $\mathcal{C}$ from Θ for time bound T, is defined as $Reach(\mathcal{C}, \Theta, T) = \{x \mid \exists \tau \in \mathcal{T}, \exists t \in [0, T]$ such that $, \tau(0) \in \Theta,$ and $, \tau(t) = x\}$.*

Example 1: An RLC circuit consists of a resistor, capacitor, and inductor connected in series or in parallel (an example RLC circuit with series connection is shown in Figure 2.1). RLC circuits are ubiquitous in oscillators and power systems.

A second order differential equation modeling the dynamics of current in an RLC circuit is given by:

$$\frac{d^2 i}{dt^2} + \frac{R}{L}\frac{di}{dt} + \frac{i}{LC} = 0.$$

Using the variable transformation as $v \mapsto \frac{di}{dt}, u \mapsto i$, we can write the above second order differential equation as a linear ODE as

$$\begin{pmatrix} \dot{u} \\ \dot{v} \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -\frac{1}{LC} & -\frac{R}{L} \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}. \tag{2.1}$$

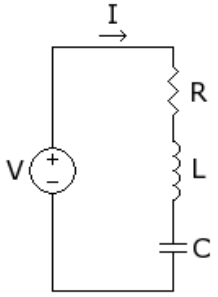


Figure 2.1: An example RLC circuit.

□

Consider a continuous system $\mathcal{C} = \langle \mathcal{V}, \mathcal{T} \rangle$, where the trajectories are solutions of a linear ODE (also commonly referred as Linear Time Invariant Systems (LTI systems))

given as $\dot{x} = Ax + B$ where A is an $n \times n$ matrix and B is a column matrix with n rows. The closed form expression for the state of a trajectory at time t is given by:

$$\tau(t) = e^{At}\tau(0) + \int_0^t e^{A(t-s)}Bds$$

where e^{At} is the matrix exponential.

For a continuous system $\mathcal{C} = \langle \mathcal{V}, \mathcal{T} \rangle$, where the trajectories are solutions of a linear ODE given as $\dot{x} = A(t)x + B(t)$ where $A(t)$ and $B(t)$ are time varying matrices (often called Linear Time Varying Systems (LTV systems)), a closed form expression similar to LTI systems does not exist. However, if $A(t)$ and $B(t)$ are continuous functions, there exists a state transition matrix which characterizes the trajectories of the system. A state transition matrix Φ where $\Phi : \mathbb{R} \times \mathbb{R} \to \mathbb{R}^{n \times n}$ returns a matrix $\Phi(t_1, t_2)$ given any two time instances t_1 and t_2. It satisfies the following properties for any t_1, t_2, t_3 in $\mathbb{R}$

1. $\Phi(t_1, t_1) = I$,
2. $\Phi(t_1, t_2)^{-1} = \Phi(t_2, t_1)$, and
3. $\Phi(t_1, t_3) = \Phi(t_1, t_2)\Phi(t_2, t_3)$

If $A(t)$ and $B(t)$ are continuous functions, then there exists a state transition matrix Φ such that all the trajectories of $\mathcal{C}$ satisfy

$$\tau(t) = \Phi(t, 0)\tau(0) + \int_0^t \Phi(t, s)B(s)ds.$$

Notice that the state transition matrix for LTV is a generalization of LTI. Substituting $\Phi(t_1, t_2) = e^{A(t_2 - t_1)}$ would give a solution to the LTI system. In this thesis, we assume that for linear systems considered, Φ is the state transition matrix that defines the trajectories.

2.2.2 Hybrid Systems

In addition to modeling the continuous behaviors, a hybrid system also models the software state using discrete variables. For hybrid systems the set of variables (both continuous and discrete) in denoted by $\mathcal{V}$. This set is the union of set of real valued continuous variables X and a special discrete variable *loc* that captures the software

state of a CPS. The set of all possible valuations of these variables is denoted by $val(\mathcal{V})$. The set of all possible valuations of *loc* is called the set of *locations* (*Loc*). The continuous evolution of the variables is modeled by *trajectories* similar to that of continuous systems.

The discrete transitions between the two locations are specified by a set A of actions. An action $a \in A$ is enabled at a state whenever the state satisfies a special predicate $Guard_a$ that is associated with the action. When the system takes a discrete transition, the new state of the system after the transition is defined by a function $Reset_a$ that maps the old state (pre-state) to a new state, also called the post-state (and possibly a new location). All of these components together define the behavior of the hybrid automaton as a sequence of alternating trajectories and transitions.

Definition 2 *A Hybrid Automaton (HA) $\mathcal{A}$ is a tuple $\langle \mathcal{V}, A, \mathcal{D}, \mathcal{T} \rangle$ where*

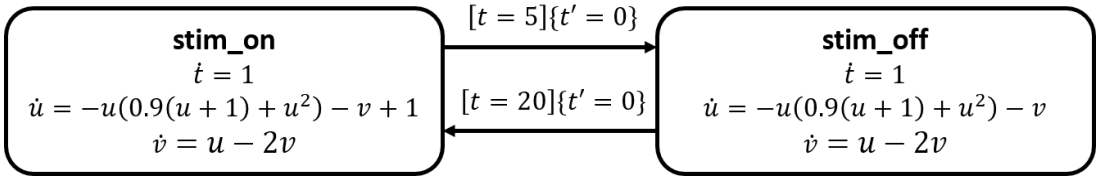
(a) $\mathcal{V} = X \cup \{loc\}$ *is a set of variables. Here loc is a discrete variable of finite type Loc. Valuations of loc are called* locations. *Each $x \in X$ is a continuous variable of type $\mathbb{R}$. Elements of $val(\mathcal{V})$ are called* states.

(b) A is a finite set of actions *or transition labels.*

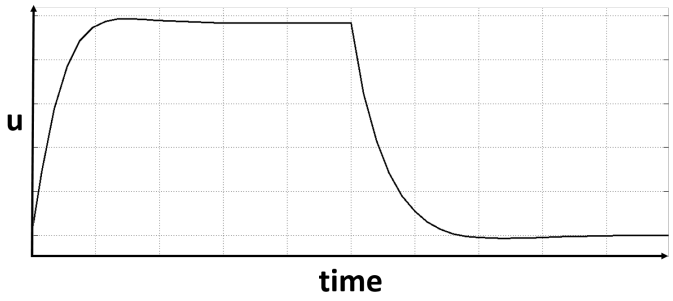
(c) $\mathcal{D} \subseteq val(\mathcal{V}) \times A \times val(\mathcal{V})$ *is the set of* discrete transitions. *A discrete transition $(\mathbf{v}, a, \mathbf{v}') \in \mathcal{D}$ is written as $\mathbf{v} \xrightarrow{a} \mathbf{v}'$. The discrete transitions are specified by finitely many guards and reset maps involving V.*

(d) $\mathcal{T}$ is a set of trajectories for X which is closed under suffix, prefix and concatenation (see [104] for details). For each $l \in Loc$, a set of trajectories $\mathcal{T}_l$ for location l are specified by differential equations E_l and an invariant $I_l \subseteq val(X)$. Over any trajectory $\tau \in \mathcal{T}_l$, loc remains constant and the variables in X evolve according to E_l such that at each time in the domain of τ, $\tau(t)$ satisfies the invariant I_l.

An execution of a hybrid automaton $\mathcal{A}$ records all the information (about its variables) over a particular run. Formally, an *execution* is an alternating sequence of trajectories and actions $\sigma = \tau_0 a_1 \tau_1 \ldots$ where each τ_i is a closed trajectory and $\tau_i(t) \xrightarrow{a_{i+1}} \tau_{i+1}(0)$, where t is the last time point in τ_i. The *duration* of an execution is the total duration of all the trajectories. $\sigma.dur = \Sigma \tau_i.dur$ if all the trajectories τ_i are finite or $\sigma.dur = \infty$ otherwise. The set of all executions is denoted as $execs(\mathcal{A})$. In this thesis, we only consider executions with a bounded number of transitions and



(a) Hybrid automaton model of a cardiac cell with a pacemaker.



(b) Sample behavior of the cardiac cell-pacemaker system from an initial state.

Figure 2.2: Figure depicting a hybrid automaton model of a cardiac cell-pacemaker system and a behavior of the hybrid automaton from an initial set.

bounded duration. Given a set of initial states $\Theta \subseteq val(V)$, the set of *executions from* Θ are those executions in $execs(\mathcal{A})$ with their first state, $\tau_0(0)$, in Θ. The set of executions starting from Θ of duration at most T and with at most N transitions will be denoted as $execs(\mathcal{A}, \Theta, T, N)$.

Example 2: A hybrid system that models the behavior of a cardiac cell-pacemaker system is given in Figure 2.2(a). The hybrid system has two locations, namely, **stim_on** and **stim_off**. The continuous variables u and v model certain electrical properties of the cardiac cell and t models the time spent in the current location. The system stays in **stim_on** location when the pacemaker gives a stimulus to the cell and is in **stim_off** location when the stimulus from pacemaker is absent.

Observe that the discrete transition from **stim_on** to **stim_off** is only enabled when the value of $t = 5$. After the discrete transition, the value of t is reset to 0, and the values of u and v are left unchanged. Observing the other discrete transition from **stim_off** to **stim_on** helps us realize that the hybrid system given in Figure 2.2(a) models the cardiac cell-pacemaker system where pacemaker gives stimulus periodically (with a time period of 25 units) for a duration of 5 units. The behavior of the continuous variable u with *time* is given in Figure 2.2(b). □

Definition 3 (Reachable Set) *Given a hybrid system $\mathcal{A}$, initial set Θ, bounded time T, and number of discrete transitions N, the set of all reachable states, denoted as $Reach(\mathcal{A}, \Theta, T, N) = \{ x \mid \exists \sigma \in execs(\mathcal{A}, \Theta, T, N), 0 \leq t \leq N$, such that $\sigma(t) = x\}$. We denote this set as $Reach(\Theta, T, N)$ when $\mathcal{A}$ is clear from the context.*

Definition 4 (Safe and Unsafe) *Given a hybrid automaton $\mathcal{A}$ with an initial set Θ, unsafe set U, time bound T, and transition bound N, is said to be* safe, *if $Reach(\mathcal{A}, \Theta, T, N) \cap U = \emptyset$. Otherwise, $\mathcal{A}$ is said to be* unsafe.

Definition 5 (Perturbing a Hybrid Automaton) *Given a hybrid automaton $\mathcal{A} = \langle \mathcal{V}, A, \mathcal{D}, \mathcal{T} \rangle$, we define an* ϵ-perturbation *of $\mathcal{A}$ as a new automaton $\mathcal{A}_\epsilon$ that has components identical to $\mathcal{A}$, except, (a) for each location $\ell \in Loc$, $I_\ell^{\mathcal{A}_\epsilon} = B_\epsilon(I_\ell^{\mathcal{A}})$ and (b) for each action $a \in A$, $Guard_a^{\mathcal{A}_\epsilon} = B_\epsilon(Guard_a^{\mathcal{A}})$.*

Here $I_{loc}^{\mathcal{A}}$ is the invariant of the location of a hybrid automaton $\mathcal{A}$ and $Guard_a^{\mathcal{A}}$ denotes the guard set for action a. The definition permits $\epsilon < 0$ for perturbation of a hybrid automaton. Informally, a positive perturbation of a hybrid automaton $\mathcal{A}$ bloats the invariants and guard sets and therefore enlarges the set of executions. A negative perturbation on the other hand, shrinks the invariants and the guards and therefore reduces the set of executions.

Definition 6 (Robust safety and unsafety) *Given a hybrid automaton $\mathcal{A}$ with an initial set Θ, unsafe set U, time bound T, and bound on discrete transitions N, it is said to be* robustly safe *if and only if $\exists \epsilon > 0$, such that $Reach(\mathcal{A}_\epsilon, B_\epsilon(\Theta), T, N) \cap B_\epsilon(U) = \emptyset$. It is said to be* robustly unsafe *if and only if $\exists \epsilon < 0$ such that $Reach(\mathcal{A}_\epsilon, B_\epsilon(\Theta), T, N) \cap B_\epsilon(U) \neq \emptyset$.*

2.3 Models For Distributed Cyber-Physical Systems

A distributed CPS consists of several agents, each of them interacting with physical environment, software, and with each other by sending and receiving messages through communication channels. For modeling distributed CPS, we will use a special type of *Hybrid Input/Output Automaton (HIOA)* [104, 127] that has constructs for communication channels and messages. Here, input/output transitions of the automata model the sending and receiving of messages. We first discuss the communication model among the agents, present the HIOA model of each of these agents, and finally give the semantics for the behavior of distributed CPS.

Communication Model The agents communicate through messages sent over an unreliable asynchronous channel. That is, messages can be arbitrarily delayed and dropped. It is standard to formally model this communication as a single automaton, Channel, which stores the set of all in-flight messages that have been sent, but are yet to be delivered. An agent sends a message m by invoking a $\mathsf{send}(m)$ action (more on this in the agent model). This action adds m to the *in-flight* set. At any arbitrary time, an *in-flight* message m is chosen by the Channel, which either delivers it to its recipient or removes it from the set. Let M be the set of all possible messages ever sent or received. We assume that all messages are unique and each message identifies its sender and recipient. We denote the set of messages sent and received by agent i by $M_{i,*}$ and $M_{*,i}$, respectively.

Agent Model Each agent in a distributed CPS interacts with the physical environment and the software. Hence we consider that each of the agents is a hybrid automaton as given in Definition 2 along with some extensions. In this thesis, we denote the hybrid system that models the behavior of i^{th} agent as $\mathcal{A}_i$. One specific feature of agents in distributed CPS is that each of the agents has its own clock variable clk_i. For distributed CPS models, this clk_i models the real-world properties such as drift and phase difference between clocks of different agents. Hence, it is not necessary that all the agents have the exact value of clk_i.

Hybrid automaton $\mathcal{A}_i \triangleq \langle \mathcal{V}_i, A_i, D_i, \mathcal{T}_i \rangle$ that models the behavior of the agent i is a hybrid automaton as given in Definition 2 along with a few extensions. These are given as follows:

a) The set of variables $\mathcal{V}_i$ consists of the set of continuous variables X_i (which includes the clk_i variable) and the set of discrete variables Y_i. The set of discrete variables Y_i includes a special variable $msghist_i$ that records all the sent and received messages. We denote the set of valuations for agent i as $Q_i \triangleq val(\mathcal{V}_i)$.

a) The set of actions A_i consists of the discrete transitions defined in Definition 2 and the set $\{\mathsf{send}_i(m) \mid m \in M_{i,*}\} \cup \{\mathsf{receive}_i(m) \mid m \in M_{*,i}\}$.

a) Discrete transitions include the transitions given in Definition 2 and also of actions $\mathsf{send}_i(m)$ and $\mathsf{receive}_i(m)$ actions in A_i. During the discrete transitions of $\mathsf{send}_i(m)$ and $\mathsf{receive}_i(m)$, the corresponding message is added to the $msghist_i$ discrete variable.

a) The trajectories of agents in distributed CPS are same as that of the hybrid automaton given in Definition 2.

System Model Let I be the set of unique identifiers for all the agents in the system. The complete system model is a TIOA called $\mathsf{System} = \langle \mathcal{V}, A, \mathcal{D}, \mathcal{T} \rangle$ that is obtained as the parallel composition of $\{\mathcal{A}_i\}_{i \in I}$ with $\mathsf{Channel}$. We refer the reader to [104] for the formal definition of the composition operator. Informally, each $\mathcal{A}_i$ synchronizes with $\mathsf{Channel}$ through the $\mathsf{send}_i(m)$ and the $\mathsf{receive}_i(m)$ actions. For a message $m \in M_{i,j}$ sent by $\mathcal{A}_i$ for $\mathcal{A}_j$, the $\mathsf{send}_i(m)$ is triggered by $\mathcal{A}_i$ and puts m in $\mathsf{Channel}$, then some time after that, $\mathsf{receive}_j(m)$ is (nondeterministically) triggered by $\mathsf{Channel}$, and causes m to be delivered at $\mathcal{A}_j$.

Semantics An *execution* of System models a particular run. Formally, an *execution* σ is an alternating sequence $\tau_0 a_1 \tau_1 \ldots \tau_k$, where each τ_j in the sequence is a trajectory, and each a_j is an action of System. The duration of an execution is defined as $\sigma.dur = \sum_{j=1}^{k} \tau_j.dur$. For any $t \in [0, \sigma.dur]$, $\sigma(t)$ denotes state of System at the end of in the longest prefix of σ with duration t. For a set of variables S, $\sigma(t).S$ is the valuation of the variables in S at the state $\sigma(t)$. A *global predicate* for System is a set $P \subseteq \prod_{i \in I} Q_i$. Often we will define P using a formula involving the variables in $\bigcup_{i \in I} V_i$. A predicate is *satisfied by an execution σ at time t* if $\sigma(t) \in P$.

2.3.1 Properties Of Distributed Cyber-Physical Systems

In this thesis, we are concerned with inferring global properties of sets of executions that *correspond to* recorded *observations*. First, we define recorded observations or traces.

Definition 7 *A* trace *for an agent $\mathcal{A}_i$ is a finite sequence $\beta_i = \mathbf{v}_i[1], \ldots, \mathbf{v}_i[k]$, where each $\mathbf{v}_i[j] \in Q_i$ is the j^{th} observed state of $\mathcal{A}_i$. We define length(β) to be k.*

We assume that for every agent $\mathcal{A}_i$, we know its initial state, denoted as $\mathbf{v}_{i,0}$. A trace for System is a collection $\beta = \{\beta_i\}_{i \in I}$, where each β_i is a trace for $\mathcal{A}_i$. We define $[\![\beta]\!]$ as the set of observations in β. Next, we formalize the notion of correspondence between traces and executions.

Definition 8 *Given a trace* $\beta = \{\beta_i\}_{i \in I}$*, an execution* σ *of* System *corresponds to* β *if*

$$\forall\ i \in I, \exists\ t_1 \leq t_2 \ldots \leq t_{length(\beta_i)},$$
$$\forall j \in \{1, \ldots, length(\beta_i)\}, \sigma(t_j).\mathcal{V}_i = \beta_i[j] \wedge \sigma(0).\mathcal{V}_i = \mathbf{v}_{i,0}.$$

The set of executions corresponding to trace β *is denoted by* $\mathsf{TraceInv}_\beta$*. Multiple executions may correspond to the same trace because the system model can be nondeterministic and there is loss of information in the trace observations.*

Finally, we state the two types of analysis we perform on distributed CPS in this thesis. Given the model of System $\mathcal{A}$, global predicate P, and a trace β, decide if

(a) there exists a time $t \in [0, \sigma.dur]$ such that for all $\sigma \in \mathsf{TraceInv}_\beta$, σ satisfies P at t, that is $\sigma(t) \in P$, and

(b) for all time $t \in [0, \sigma.dur]$ and all $\sigma \in \mathsf{TraceInv}_\beta$, σ satisfies P at t, that is $\sigma(t) \in P$.

We conclude this section by recalling the definition of Lamport's *happens before* relation on state observations [110]. For two state observations $\mathbf{v}_i[j]$ and $\mathbf{v}'_i[j']$ in a given trace β, $\mathbf{v}_i[j]$ is said to happen before $\mathbf{v}'_i[j']$, denoted by $\mathbf{v}_i[j] \rightsquigarrow \mathbf{v}'_i[j']$, iff one of the following holds: (i) $i = i'$ and $j < j'$, or (ii) $i \neq i'$, $\mathbf{v}_i[j]$ is the post state of a $\mathsf{send}_i(m)$ transition for some $m \in M_{i,i'}$, and $\mathbf{v}_{i'}[j']$ is the post-state of the corresponding $\mathsf{receive}_{i'}(m)$ transition. We identify the relation $\rightsquigarrow$ with its transitive closure. For an observation $\mathbf{v}$ in β, we define *before*$(\mathbf{v})$ as the set $\{\mathbf{u} \mid \mathbf{u} \rightsquigarrow \mathbf{v}\}$ and *after*$(\mathbf{v})$ as the set $\{\mathbf{u} \mid \mathbf{v} \rightsquigarrow \mathbf{u}\}$.

Chapter 3

Dynamic Analysis of Continuous Systems

In this chapter, we present a dynamic analysis technique for verifying bounded time safety properties for continuous systems. The dynamic analysis algorithm along with the model, requires as input a *discrepancy function* for the continuous system: a function that gives an upper bound on the distance between two trajectories. In this chapter, we show that discrepancy function generalizes well known proof certificates considered in control theory such as contraction metrics [118] and incremental Lyapunov functions [16]. Further, we present a new algorithm for synthesizing discrepancy functions for linear systems. We then present the dynamic analysis algorithm for bounded time safety verification of continuous systems. This algorithm iteratively computes *reachtree*, a data structure that records an overapproximation of the reachable set of states. We establish theoretical properties such as soundness and relative completeness of the verification algorithm and present experimental results that outperform other verification tools.

3.1 Introduction

Given a continuous system, the bounded time safety property answers the following question: "Given an initial set and an unsafe set, does there exist a trajectory which starts from the initial set and reaches the unsafe set within the bounded time T". For continuous systems where rate of change of a variable is a constant, or lies in a fixed interval, i.e. $\dot{x} = c$ or $\dot{x} \in [a, b]$, this question can be answered trivially when initial and unsafe sets are convex polyhedra. For general continuous systems with trajectories specified as solutions of ordinary differential equations $\dot{x} = f(x)$, the safety verification problem can be nontrivial. Recall from Section 2.2.1, that for linear systems given as $\dot{x} = Ax + B$, the closed form solution is given by the equation $\tau(t) = e^{At}\tau(0) + \int_0^t e^{A(t-\tau)}Bd\tau$. Hence, checking the bounded time safety property would require computing the *matrix exponential* e^{At}.

This presents us with two challenges for verification. First, computing the matrix exponential exactly (or its symbolic representation) is only possible for very specific matrices. To overcome this challenge, one has to rely on computing its bounded numerical approximation, several techniques for which, have been presented in [130]. Second, even if the matrix exponential can be computed numerically, one has to track all the trajectories from the initial set and prove (or disprove) that no trajectory reaches the unsafe set. One of the common techniques employed to overcome this challenge is computing the reachable set. In [64, 65, 77, 8, 31], the authors present new approaches that compute the reachable set of states for linear systems using numerical overapproximations of matrix exponential. While these techniques are sound, they suffer from curse of dimensionality and hence do not scale for large dimensional systems.

Unlike its linear counterparts, the closed form expression for the solution of a nonlinear ODEs may not exist. Hence, the techniques developed for safety verification of linear systems cannot be extended to nonlinear systems. Current state of the art techniques [38, 28] for checking safety verification of nonlinear systems rely on computing Taylor model flowpipes and do not scale to large dimensions.

In this chapter, we present a dynamic analysis algorithm for continuous systems (introduced in Section 2.2.1) with trajectories that are solutions of linear and nonlinear ODEs. Broadly, dynamic analysis refers to techniques that not only use the information about the automaton models (static), but also information from executions of the model or the real system for inferring a property of the system. Dynamic analysis techniques are specially suitable for continuous systems defined by nonlinear ODEs because the closed form solution for the ODE may not exist and hence inferring properties of such systems relies to a large extent on numerical simulations. Further, dynamic analysis techniques enjoy an advantage over static analysis techniques that they can produce a a counterexample (or violating behavior) in case the desired property is not satisfied by the system.

Assumption 1 *In this thesis, we make two assumptions about the safety verification problem of continuous systems. First, is that the initial set is compact and bounded and second, that the continuous system is deterministic. The assumption that the initial set is bounded and compact is commonly used in other works in literature such as [38, 64, 65]. While the Hybrid Input/Output Automata framework [127] allows for nondeterminism in trajectories, in this thesis, we focus on continuous systems where trajectories are given by solutions of ODEs. We restrict our attention to linear and*

nonlinear ODEs for which a solution exists and is unique and hence the continuous system is deterministic.

The dynamic analysis presented in this chapter exploits the continuity property of the trajectories. For "well-behaved'' continuous systems, two trajectories that start within close vicinity of each other, stay close to each other, and further, this distance vanishes as the distance between the starting states vanish. This intuition is captured in what we call *discrepancy function*. A given discrepancy function for a continuous system (a) bounds the distance between trajectories that start from neighboring states, and (b) converges to 0 as the distance between the initial states goes to 0. Proof certificates such as *contraction metric* [118] and *incremental Lyapunov function* [16] have been proposed in control theory in order to study convergence and divergence of neighboring trajectories. In this chapter we establish that discrepancy function is a generalization of such commonly used proof certificates. This helps us leverage the existing automatic techniques for proving convergence or divergence and thereby compute discrepancy function.

The dynamic analysis technique for safety verification presented in this chapter is an iterative process with 4 steps: *simulate*, *bloat*, *check*, and *refine* respectively. The verification algorithm proceeds as follows: 1) It computes a finite cover of the bounded and compact initial set according to partitioning parameter δ. This cover is a collection of neighborhoods that have diameter bounded by δ. 2) Next, the algorithm generates a numerical simulation from each of these neighborhoods. 3) It then computes an overapproximation of the reachable set of each neighborhood by bloating the numerical simulation using the discrepancy function. We call each of these bloated simulations as *reachtubes*. The bloating using discrepancy functions ensures that the set of reachable states from the initial set is contained in the union of *reachtubes*. 4) It then checks whether each of these *reachtubes* satisfies the safety property. 5) Finally, based on the results of checking safety, the algorithm either concludes that the system satisfies (or violates) the safety property or if it cannot infer safety (or its violation), it computes a new set of *reachtubes* by reducing the partitioning parameter δ.

We show that the proposed dynamic analysis technique can not only compute sound overapproximation, but also compute arbitrarily precise overapproximation of the reachable set from the initial set Θ. This helps us in establishing relative completeness which states that the algorithm will terminate with the right answer whenever the system is robustly safe or robustly unsafe.

The rest of the chapter is organized as follows, first we introduce the notion of discrepancy function and establish that discrepancy function generalizes well studied notions of proof certificates studied in control theory in Section 3.2. In Section 3.4, we present a new techniques for computing discrepancy functions for linear ODEs. We then present in Section 3.5, a dynamic analysis algorithm and prove its theoretical guarantees. We conclude and present the related work in Section 3.7 after reporting the verification results on standard benchmark examples in Section 3.6.

3.2 Discrepancy Function

The dynamic analysis procedure presented in this chapter requires what we call discrepancy function that we will now present. For now, we assume that discrepancy function is provided as an input by the end user, however, in Section 3.4 we discuss a few techniques to compute discrepancy functions automatically. Informally, a discrepancy function gives an upper bound on the distance between two trajectories as a function of the distance between their initial states and the time elapsed.

Definition 9 *Given an n-dimensional continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ with a set of real valued variables $\mathcal{V}$ and the set of trajectories $\mathcal{T}$ as defined in Section 2.2.1, a smooth function $V : \mathbb{R}^{2n} \to \mathbb{R}_{\geq 0}$ is called a* discrepancy function *for the continuous system if there are functions $\alpha_1, \alpha_2 \in \mathcal{K}_\infty$ and a uniformly continuous function $\beta : \mathbb{R}^{2n} \times \mathbb{R} \to \mathbb{R}_{\geq 0}$ such that for any pair of states $x_1, x_2 \in \mathbb{R}^n$, any pair of trajectories $\tau_1, \tau_2 \in \mathcal{T}$, and time $t \geq 0$:*

$$x_1 \neq x_2 \iff V(x_1, x_2) > 0, \tag{3.1}$$

$$\alpha_1(||x_1 - x_2||) \leq V(x_1, x_2) \leq \alpha_2(||x_1 - x_2||), \tag{3.2}$$

$$t \leq min\{\tau_1.dur, \tau_2.dur\}, V(\tau_1(t), \tau_2(t)) \leq \beta(\tau_1(0), \tau_2(0), t), \text{ and} \tag{3.3}$$

$$\beta(x_1, x_2, t) \to 0 \text{ as } ||x_1 - x_2|| \to 0. \tag{3.4}$$

A tuple $\langle \alpha_1, \alpha_2, \beta \rangle$ satisfying the above conditions is called a witness *to the discrepancy function.*

Condition 3.1 requires that the function $V(x_1, x_2)$ vanishes to zero if and only if the first two arguments (the initial states of the two trajectories) are identical. Con-

dition 3.2 states that the value of $V(x_1, x_2)$ can be upper and lower-bounded by functions α_2 and α_1 of the ℓ^2 distance between x_1 and x_2. Condition 3.3 requires that the function V applied to two trajectories τ_1 and τ_2 at time t (within the duration of both the trajectories) is upper bounded by the function β applied to the initial states of the two trajectories $\tau_1(0)$, $\tau_2(0)$, and the time t. Together with condition 3.4 on β, we have that the function V converges to 0 as the ℓ^2 distance between the initial states $||\tau_1(0) - \tau_2(0)||$ converges to 0. *Local discrepancy functions* [61] can be defined by restricting the trajectories $\mathcal{T}$ and functions V and β to a subset of $\mathbb{R}^n$ in Definition 9. Figure 3.1 illustrates discrepancy function using sample trajectories.

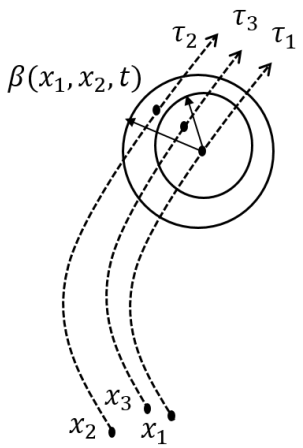


Figure 3.1: Figure illustrating the discrepancy function with sample trajectories τ_1, τ_2, τ_3 from initial states x_1, x_2, x_3 respectively.

Remark 1 An *exponential discrepancy function*, defined by a pair $\langle K, \gamma \rangle, K \geq 1, \gamma \in \mathbb{R}$ is a special type of discrepancy function with V, α_1, α_2, and β given as

1) $V(x_1, x_2) \triangleq ||x_1 - x_2||$,

2) $\alpha_1(||x_1 - x_2||) = \alpha_2(||x_1 - x_2||) \triangleq ||x_1 - x_2||$, and

3) $\beta(x_1, x_2, t) \triangleq Ke^{\gamma t}||x_1 - x_2||$.

It is easy to infer that β is uniformly continuous for a bounded time T, and conditions 3.1, 3.2, and 3.4 are satisfied. □

3.3 Discrepancy Function And Proof Certificates In Control Theory

In this section, we show that proof certificates used in control theory are in fact discrepancy functions. We consider continuous systems $\langle \mathcal{V}, \mathcal{T} \rangle$ as given in Section 2.2.1 with the trajectories satisfying the Ordinary Differential Equation

$$\dot{x} = f(x). \tag{3.5}$$

That is, for any $\tau \in \mathcal{T}$, $0 \leq t \leq \tau.dur$, $\frac{d}{dt}\tau(t) = f(\tau(t))$.

3.3.1 Lipschitz Continuity

Consider the continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ with $L > 0$ as the Lipschitz constant for $f(x)$, right hand side of differential equation. Lipschitz continuity is a common assumption on $f(x)$ as it guarantees the existence and uniqueness of trajectories. Further, Lipschitz constant for $f(x)$ can be computed automatically for linear, polynomial, and certain classes of trigonometric functions. A few empirical techniques for estimating Lipschitz constant for different classes of functions are given in [168]. We observe that for such systems, the function $V(x_1, x_2) \triangleq ||x_1 - x_2||$ is a discrepancy function.

Proposition 1 *For continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ that satisfies ODE given in Equation 3.5 with Lipschitz constant $L \in \mathbb{R}_{\geq 0}$ for $f(x)$, $V(x_1, x_2) \triangleq ||x_1 - x_2||$ is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where $\alpha_1(||x_1 - x_2||) = \alpha_2(||x_1 - x_2||) = ||x_1 - x_2||$ and $\beta(x_1, x_2, t) = e^{Lt}||x_1 - x_2||$.*

Proof: It is easy to check that the function $V(x_1, x_2) = ||x_1 - x_2||$ satisfies the first two conditions (conditions 3.1, 3.2) in Definition 9 with $\alpha_1(||x_1 - x_2||) = \alpha_2(||x_1 - x_2||) = ||x_1 - x_2||$. Consider a pair of trajectories $\tau_1, \tau_2 \in \mathcal{T}$ with $t \leq \tau_1.dur, t \leq \tau_2.dur$. If we take the derivative of the function V with respect to t:

$$\begin{aligned}
\frac{d}{ds}(V(\tau_1(s), \tau_2(s))) &= \frac{d}{ds}||\tau_1(s) - \tau_2(s)|| \\
&= \frac{\tau_1(s) - \tau_2(s)}{||\tau_1(s), \tau_2(s)||}(\frac{d}{ds}\tau_1(s) - \frac{d}{ds}\tau_2(s)) \\
&\leq ||\frac{d}{ds}\tau_1(s) - \frac{d}{ds}\tau_2(s)|| = ||f(\tau_1(s)) - f(\tau_2(s))|| \\
&\leq L||\tau_1(s) - \tau_2(s)|| = L\ V(\tau_1(s), \tau_2(s)).
\end{aligned}$$

Thus, for any pair of trajectories τ_1 and τ_2, we have

$$\begin{aligned}
& \dot{V} \leq L\ V \\
\Rightarrow\ & V(\tau_1(t), \tau_2(t)) \leq V(\tau_1(0), \tau_2(t))e^{\int_0^t L ds} \text{ using Grönwall's inequality} \\
\Rightarrow\ & V(\tau_1(t), \tau_2(t)) \leq V(\tau_1(0), \tau_2(0))e^{Lt}
\end{aligned}$$

This gives us a bound on V which grows exponentially with time t:

$$V(\tau_1(t), \tau_2(t)) \leq e^{Lt}V(\tau_1(0), \tau_2(0)) = e^{Lt}||\tau_1(0) - \tau_2(0)||.$$

Hence V is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where $\alpha_1 = \alpha_2 = ||x_1 - x_2||$

and $\beta(x_1, x_2, t) = e^{Lt}||x_1 - x_2||$. ■

We illustrate the computation of discrepancy function using Lipschitz constant using an example of RLC circuit.

Example 3: Consider the RLC circuit given in Example 1. For particular choice of the R, L, and C parameters, the system can be written as:

$$\begin{pmatrix} u \\ v \end{pmatrix} = \begin{pmatrix} 0 & 1 \\ -2 & -2 \end{pmatrix} \begin{pmatrix} u \\ v \end{pmatrix}. \tag{3.6}$$

The system with state space $\bar{u} = (u, v)$ is a damped oscillatory system that eventually stabilizes to the origin. The Lipschitz constant over this region can be computed by maximizing $\frac{||A_1[u_1,v_1]^T - A_1[u_2,v_2]^T||}{||[u_1-u_2 \ v_1-v_2]^T||}$ with (u_1, v_1) and (u_2, v_2) in $\mathbb{R}^2$, where A_1 is the matrix in Equation 3.6. For this example, the Lipschitz constant can be obtained by computing the matrix norm $||A_1||$ using MATLAB and it turns out to be 2.9208. It follows that, the discrepancy function is given as $V((u_1, v_1), (u_2, v_2)) = ||[u_1 - u_2 \ v_1 - v_2]^T||$ and $\beta((u_1, v_1), (u_2, v_2), t) = e^{2.9208t}||[u_1 - u_2 \ v_1 - v_2]^T||$. □

Although computing Lipschitz constant is a feasible way to compute the discrepancy function, we illustrate the disadvantage of using Lipschitz constant using Example 3. Consider two trajectories τ_1 and τ_2 starting δ distance apart, i.e., $||\tau_1(0) - \tau_2(0)|| = \delta$. The discrepancy function given in Example 3 estimates that $||\tau_1(10) - \tau_2(10)|| \leq e^{2.9208 \times 10}\delta \approx 4.84 \ 10^{12}\delta$. The overapproximation of the reachable set for the RLC circuit in Example 3 for the initial set $u \in [0, 0.1], v = 2$ computed using the Lipschitz constant is given in Figure 3.2(a). This estimate of distance is very coarse overapproximation and hence may be impractical for verification, especially for long time horizons. In the next few sections, we present other proof certificates which compute more precise overapproximations that are favorable for verification than Lipschitz constant.

3.3.2 Contraction Metrics

The discrepancy function from Lipschitz constant provides an overapproximation of the distance between trajectories that exponentially increases with time. For systems where trajectories exponentially converge towards each other, contraction metric (proposed in [118, 117]) is given as a proof certificate for establishing this convergence. In

this section we compute a discrepancy function from contraction metric, and unlike Lipschitz constant the overapproximation of distance between trajectories exponentially *decreases* with time.

Definition 10 (Definition 1 from [118]) *A region of the state space for ODE given in Equation 3.5 is called* contraction region *if the Jacobian of $f(x)$ given by $\frac{\partial \mathbf{f}}{\partial x}$ is uniformly negative definite in that region.*

In was shown in [118] that trajectories converge exponentially in a contraction region of an ODE. The rate of exponential convergence is bounded by the maximum eigen value of the symmetric part of Jacobian. Theorem 2 formalizes the exponential convergence of trajectories and helps in computing the discrepancy function in the contraction region of an ODE.

Theorem 2 (Theorem 1 from [118]) *The exponential rate of convergence of trajectories in* contraction region *for continuous system defined by Equation 3.5 is bounded by λ_{max} where λ_{max} is the maximum eigen value of symmetric part of Jacobian $(\frac{1}{2}(\frac{\partial \mathbf{f}}{\partial x}^T + \frac{\partial \mathbf{f}}{\partial x}))$. Hence, for any two trajectories τ_1 and τ_2 in contraction region, $||\delta x|| \leq ||\delta x_0|| e^{\lambda_{max} t}$, where $\delta x_0 = \tau_1(0) - \tau_2(0)$ and $\delta x = \tau_1(t) - \tau_2(t)$.*

Proposition 3 *For a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE given in Equation 3.5 with λ_{max} as the maximum eigen value of symmetric part of Jacobian, $V(x_1, x_2) \triangleq ||x_1 - x_2||$ is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where $\alpha_1 = \alpha_2 = ||x_1 - x_2||$ and $\beta(x_1, x_2, t) = ||x_1 - x_2|| e^{\lambda_{max} t}$.*

Proof: Follows from Theorem 2. ∎

A weaker condition than Theorem 2 for proving exponential convergence of trajectories is presented in Theorem 4.

Definition 11 (Definition 2 from [118]) *A uniform metric $\mathbf{M} : \mathbb{R}^n \times \mathbb{R}_{\geq 0} \to \mathbb{R}^{n \times n}$ is called a* contraction metric *for ODE given in Equation 3.5 if $\exists \beta_M \in \mathbb{R}_{\geq 0}$ such that*

$$\frac{\partial f}{\partial x}^T \mathbf{M}(x,t) + \mathbf{M}(x,t) \frac{\partial f}{\partial x} + \dot{\mathbf{M}}(x,t) + \beta_M \mathbf{M}(x,t) \preceq 0. \tag{3.7}$$

Theorem 4 (Theorem 2 From [118]) *For an ODE given in Equation 3.5 that admits a contraction metric* $\mathbf{M}$*, the trajectories converge exponentially with time, i.e.* $\exists\ k \geq 1, \gamma > 0$ *such that,* $\forall \tau_1, \tau_2 \in \mathcal{T}$*,* $\delta x^T \cdot \delta x \leq k \delta x_0^T \cdot \delta x_0 e^{-\gamma t}$*, where* $\delta x_0 = \tau_1(0) - \tau_2(0)$ *and* $\delta x = \tau_2(t) - \tau_2(t)$*.*

Proposition 5 *For continuous systems* $\langle \mathcal{V}, \mathcal{T} \rangle$ *defined by ODE in Equation 3.5 that admits a contraction metric* $\mathbf{M}$*,* $V(x_1, x_2) \triangleq (x_1 - x_2)^T (x_1 - x_2)$ *is a discrepancy function with witness* $\langle \alpha_1, \alpha_2, \beta \rangle$ *where* $\alpha_1 = \alpha_2 = (||x_1 - x_2||)^2$ *and* $\beta(x_1, x_2, t) = k(||x_1 - x_2||)^2 e^{-\gamma t}$ *where* k, γ *are from Theorem 4.*

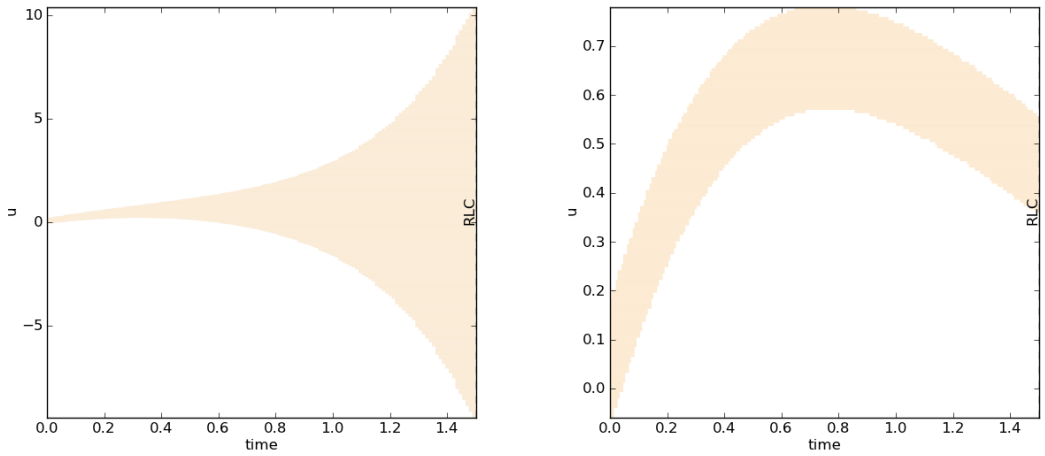
Proof: It is clear that the function $V(x_1, x_2)$ satisfies the first two conditions in Definition 9 with $\alpha_1(||x_1 - x_2||) = \alpha_2(||x_1 - x_2||) = (||x_1 - x_2||)^2 = (x_1 - x_2)^T (x_1 - x_2)$. Further, we have that:

$$\begin{aligned} V(\tau_1(t), \tau_2(t)) &= (\tau_1(t) - \tau_2(t))^T (\tau_1(t) - \tau_2(t)) \\ &= \delta x^T \delta x \text{ where } \delta x = \tau_1(t) - \tau_2(t) \\ &\leq k \delta x_0^T \delta x_0^T e^{-\gamma t}, \text{ from Theorem 4} \\ &= \beta(\tau_1(0), \tau_2(0), t). \end{aligned}$$

Hence, we have that V is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$. ■

Example 4: Continuing with the system in Example 3, we compute the Jacobian $\frac{\partial f}{\partial \bar{u}} = \begin{pmatrix} 0 & 1 \\ -2 & -2 \end{pmatrix}$ and observe that the maximum eigen value of its symmetric part is $\frac{\sqrt{3}-2}{2}$, thus $\lambda_{max} \leq \frac{-1}{10}$. From proposition 3, it follows that $V((u_1, v_1), (u_2, v_2)) = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}$ is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ such that $\alpha_1 = \alpha_2 = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}$ and $\beta((u_1, v_1), (u_2, v_2), t) = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2} e^{\frac{-t}{10}}$.

Also notice that, matrix function $\mathbf{M}(\bar{u}, t) \triangleq \begin{pmatrix} 2.5 & 0.5 \\ 0.5 & 0.75 \end{pmatrix}$ is a constant positive semi definite matrix function and hence is a uniform metric. Evaluating the left hand side of equation (3.7) with $\beta_M = 0.5$, we obtain $\frac{\partial f}{\partial \bar{x}}^T \mathbf{M}(\bar{u}, t) + \mathbf{M}(\bar{u}, t) \frac{\partial f}{\partial \bar{u}} + \dot{\mathbf{M}}(\bar{u}, t) + \beta_M \mathbf{M}(\bar{u}, t) = \begin{pmatrix} -0.75 & 0.25 \\ 0.25 & -1.625 \end{pmatrix} \prec 0$. Hence, $\mathbf{M}(\bar{u}, t)$ is a contraction



(a) Lipschitz constant as discrepancy function. (b) Contraction metric as discrepancy function.

Figure 3.2: Figure depicting the difference in reachable set (with u vs *time* plot) while using Lipschitz constant and contraction metric as discrepancy function. Comparing the scales on the two figures, it is clear that Lipschitz constant gives exponentially worse bound than contraction metric.

metric for Example 3. From Theorem 4, we have that $\exists k \geq 1, \gamma > 0$, such that $\delta\bar{u}^T \cdot \delta\bar{u} \leq k\delta\bar{u}_0^T \cdot \delta\bar{u}_0 e^{-\gamma t}$. From Proposition 5, the function $V((u_1, v_1), (u_2, v_2)) = [u_1 - u_2\ v_1 - v_2]^T [u_1 - u_2\ v_1 - v_2]$ is a discrepancy functionwith witness $\alpha_1 = \alpha_2 = ((u_1 - u_2)^2 + (v_1 - v_2)^2)$ and $\beta((u_1, v_1), (u_2, v_2), t) = k((u_1 - u_2)^2 + (v_1 - v_2)^2)e^{-\gamma t}$. Although the exponential rate of convergence can be derived from β_M, the multiplicative constant k can only be inferred for specific matrices M. □

The reachable set for the RLC circuit using the discrepancy function obtained from contraction metric as described in Example 4 for the initial set $u \in [0, 0.1], v = 2$ is given in Figure 3.2(b). Observing Examples 3 and 4 it can be inferred that several discrepancy functions can be computed for the same continuous system. The difference in the discrepancy functions would be in the order of overapproximation for the distance between trajectories. The difference in the order of overapproximation for computing the set of reachable states using different discrepancy functions becomes apparent when we compare Figure 3.2(a) and Figure 3.2(b).

3.3.3 Incremental Stability

While Lipschiz constant provides an upper bound of the distance between trajectories that increases exponentially, contraction metric gives an upper bound that decreases exponentially with time. However, in practice it need not be the case that the trajectories converge or diverge exponentially. For such systems contraction metric cannot be computed and Lipschitz constant gives a very coarse overapproximation. Incremental stability proposed in [16] expresses the property that the trajectories of the system converge towards each other, but the rate of convergence is not necessarily exponential. Incremental Lyapunov function is a proof certificate to establish incremental stability. Such guarantees impose less stricter conditions than contraction metrics and provide weaker guarantees on the rate of convergence of trajectories.

A generalization of incremental stability is the notion of incremental forward completeness proposed in [169]. While incremental stability requires that the trajectories converge towards each other with time, incremental forward completeness relaxes this notion and requires that the distance between trajectories to be bounded as time diverges. In this section we introduce the basics of incremental stability and incremental forward completeness and compute a discrepancy function whenever the system satisfies these conditions.

Definition 12 *A continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE given in Equation 3.5 is* incrementally stable *if there is a $\mathcal{KL}$ function β such that for any two trajectories τ_1 and τ_2 in $\mathcal{T}$, $||\tau_1(t) - \tau_2(t)|| \leq \beta(||\tau_1(0) - \tau_2(0)||, t)$.*

Theorem 6 (Theorem 1 from [16]) *If the system defined by ODE given in Equation 3.5 is incrementally stable, then there exists a smooth function $V : \mathbb{R}^n \times \mathbb{R}^n \to \mathbb{R}_{\geq 0}$ and $\exists \alpha_1, \alpha_2 \in \mathcal{K}_\infty$ and $\alpha \in \mathcal{K}$, such that for every pair of states $x_1, x_2 \in \mathbb{R}^n$*

$$\alpha_1(||x_1 - x_2||) \leq V(x_1, x_2) \leq \alpha_2(||x_1 - x_2||), \text{ and} \tag{3.8}$$

$$V(\tau_1(t), \tau_2(t)) - V(\tau_1(0), \tau_2(0)) \leq \int_0^t -\alpha(||\tau_1(s) - \tau_2(s)||)ds. \tag{3.9}$$

The function V is called an incremental Lyapunov function.

Proposition 7 *For a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE in Equation 3.5 with incremental Lyapunov function V, the function V is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where α_1 and α_2 are from Equation 3.8 and $\beta(x_1, x_2, t) =$*

$V(x_1, x_2) + \int_0^t -\alpha(||\tau_1(s) - \tau_2(s)||)ds$ *where* α *is from Equation 3.9 and* τ_1 *and* τ_2 *are trajectories starting from* x_1 *and* x_2 *respectively.*

Proof: From Equation 3.8, we can infer that V satisfies conditions 3.1 and 3.2. Further, from Equation 3.9 we can infer that V and β satisfy the conditions 3.3 and 3.4. Uniform continuity of β follows from uniform continuity of α as $\alpha \in \mathcal{K}$. Hence V is a discrepancy function with $\langle \alpha_1, \alpha_2, \beta \rangle$ as witness. ■

Remark 2 [Incremental Stability for Linear Systems] Consider a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by a linear differential equation $\dot{x} = Ax$ where A is a constant matrix. Suppose two $n \times n$ matrices P and Q satisfy the Lyapunov equation $A^T P + PA = Q$ and P is positive definite and Q is negative semi-definite. Then, $V(x_1, x_2) \triangleq (x_1 - x_2)^T P(x_1 - x_2)$ is an incremental Lyapunov function and hence a discrepancy function. This can be established as follows:

$$\begin{aligned}
\frac{d}{ds}(V(\tau_1(s), \tau_2(s))) &= \frac{d}{ds}(\tau_1(s) - \tau_2(s))^T P(\tau_1(s) - \tau_2(s)) \\
&= (\frac{d}{dt}(\tau_1(s) - \tau_2(s)))^T P(\tau_1(s) - \tau_2(s)) + \\
&\qquad (\tau_1(s) - \tau_2(s))^T P((\frac{d}{ds}\tau_1(s) - \tau_2(s))) \\
&= (\tau_1(s) - \tau_2(s))^T (A^T P + PA)(\tau_1(s) - \tau_2(s)) \\
&= (\tau_1(s) - \tau_2(s))^T Q(\tau_1(s) - \tau_2(s)) \\
&= -1 \times \alpha(\tau_1(s), \tau_2(s)).
\end{aligned}$$

Hence, $V(\tau_1(t), \tau_2(t)) = V(\tau_1(0), \tau_2(0)) + \int_0^t -\alpha(\tau_1(s), \tau_2(s))ds$ which is identical to Equation 3.9. Since P is a positive definite matrix, it follows that $\exists \alpha_1, \alpha_2$ such that Equation 3.8 is satisfied. Thus V is an incremental Lyapunov function for the ODE $\dot{x} = Ax$. Hence, V is a discrepancy function from Proposition 7. □

Remark 2 is of practical importance as it gives an automatic technique for computing discrepancy function for stable linear systems. The Lyapunov equation $A^T P + PA = Q$ can be solved by encoding it as Linear Matrix Inequalities (LMIs) [32] and using the related tools [116, 69].

Incremental Forward Completeness

Definition 13 (Definition 2.4 from [169]) *A continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE in Equation 3.5 is* incrementally forward complete *if it is Lipschitz continuous, and there exists continuous function $\beta : \mathbb{R}_{\geq 0} \times \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ such that for every $s \in \mathbb{R}_{\geq 0}, \beta(\cdot, s)$ is a $\mathcal{K}_\infty$ function and for any two initial trajectories τ_1 and τ_2 in $\mathcal{T}$, $||\tau_1(t) - \tau_2(t)|| \leq \beta(||\tau_1(0) - \tau_2(0)||, t)$. We refer to β as a witness for incremental forward completeness.*

Proposition 8 *If a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE in Equation 3.5 is Incrementally forward complete with witness function β_1, then the function $V(x_1, x_2) = ||x_1 - x_2||$ is a discrepancy function with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ as $\alpha_1 = \alpha_2 = ||x_1 - x_2||$ and $\beta(x_1, x_2, t) = \beta_1(||x_1 - x_2||, t)$.*

Proof: Follows from Definition 13. ■

Example 5: Continuing with the system in Example 3, here we start with a candidate quadratic incremental Lyapunov function $V((u_1, v_1), (u_2, v_2)) \triangleq [u_1 - u_2 \; v_1 - v_2]^T P [u_1 - u_2 \; v_1 - v_2]$, with $\alpha_1 \triangleq 0.375(u_1 - u_2)^2 + 0.375(v_1 - v_2)^2 + 0.5(u_1 - u_2)(v_1 - v_2)$, $\alpha_2 \triangleq 1.25(u_1 - u_2)^2 + 1.25(v_1 - v_2)^2 + 0.5(u_1 - u_2)(v_1 - v_2)$ and $\alpha \triangleq (u_1 - u_2)^2 + (v_1 - v_2)^2$, with $P = \begin{pmatrix} 1.25 & 0.25 \\ 0.25 & 0.375 \end{pmatrix}$. First we check that V is indeed an incremental Lyapunov function by computing $\dot{V}((u_1, v_1), (u_2, v_2))$ which turns out to be $[u_1 - u_2 \; v_1 - v_2]^T [A^T P + PA] [u_1 - u_2 \; v_1 - v_2] = -\alpha(u_1, v_1, u_2, v_2)$. Since $\alpha > 0$ whenever $(u_1, v_1) \neq (u_2, v_2)$, $(u_1, v_1) \neq (0, 0)$ and $(u_2, v_2) \neq (0, 0)$. Integrating both sides, we get $V(\tau_1(t), \tau_2(t)) - V((u_1, v_1), (u_2, v_2)) \leq \int_0^t -\alpha(||\tau_1(s) - \tau_2(s)||) ds$. Hence V is a discrepancy functionwith witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where $\beta((u_1, v_1), (u_2, v_2), t) = V((u_1, v_1), (u_2, v_2)) + \int_0^t -\alpha(||\tau_1(s) - \tau_2(s)||) ds$. □

Having seen different proof certificates in control theory to establish convergence and divergence of trajectories. We now inspect the difference between these proof certificates through some examples.

Example 6: Consider a two dimensional linear system: $\dot{u} = -u$; $\dot{v} = \frac{-v_0}{100}$, with initial state (u_0, v_0). A trajectory of the system is given by $\tau(t) = (u_0 e^{-t}, v_0 \frac{(1-t)}{100})$. The system converges to origin as $t \to 100$. In this example, the distance between two trajectories from different initial states decreases linearly and not exponentially. It can be verified that the function $V((u_1, v_1), (u_2, v_2)) = (u_1 - u_2)^2 + (v_1 - v_2)^2$ is

an Incremental Lyapunov function for duration $[0, 100]$ as it satisfies Equations 3.8 and 3.9. The Jacobian $\frac{\partial f}{\partial \bar{u}}$ is $\begin{pmatrix} -1 & 0 \\ 0 & 0 \end{pmatrix}$ and hence a uniform metric $\mathbf{M}(\bar{u}, t)$ that satisfies Definition 11 does not exist. □

Example 6 illustrates that incremental stability is a strictly more general condition than contraction metric. However, existence of contraction metric ensures exponential convergence of trajectories which is a much stronger guarantee than what is provided by incremental stability. Note that whenever the contraction metric is a constant matrix, an incremental Lyapunov function can be computed from the contraction metric. We now illustrate that discrepancy function is a strict generalization of both contraction metric and incremental Lyapunov function.

Example 7: Consider the system $\dot{u} = 1$; $\dot{v} = \frac{v_0}{100}$, where (u_0, v_0) is the initial state of the trajectory. The closed form solution of the above system is given as $u(t) = u_0 + t, v(t) = v_0(1 + \frac{t}{100})$. For two trajectories starting from (u_1, v_1) and (u_2, v_2), the distance between the trajectories after time t is given as $\sqrt{(u_1 - u_2)^2 + (1 + \frac{t}{100})^2 (v_1 - v_2)^2}$. Observe that the function $V((u_1, v_1), (u_2, v_2)) = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}$ with $\alpha_1 = \alpha_2 = \sqrt{(u_1 - u_2)^2 + (v_1 - v_2)^2}$ and $\beta((u_1, v_1), (u_2, v_2), t) = \sqrt{(u_1 - u_2)^2 + (1 + \frac{t}{100})^2 (v_1 - v_2)^2}$ satisfies all the conditions in Definition 9 and hence is a discrepancy function. □

Example 7 presents a system for which both contraction metric and incremental Lyapunov function do not exist, but has a bounded time discrepancy function. Detailed comparison of these proof theoretic techniques can be obtained from [100, 118, 16].

Remark 3 *[Comparison between Discrepancy Function and Sensitivity Analysis]* Sensitivity s_x for a continuous system defined by ODE $\dot{x} = f(x)$ is given by the differential equation $\dot{s_x} = \frac{\partial f}{\partial x} s_x$. The value of sensitivity at time t for an initial state x_0, denoted as $s_{x_0}(t)$ is obtained by solving the differential equation $\dot{s_x} = \frac{\partial f}{\partial x} s_x$ along the solution of $\dot{x} = f(x)$ from x_0. It has been observed in [52] that $||s_{x_1}(t)|| \cdot \epsilon$ gives the first order term in the Taylor series expansion of $||\tau_1(t) - \tau_2(t)||$ where $||\tau_1(0) - \tau_2(0)|| = \epsilon$.

Ignoring the higher order terms in the Taylor series expansion when $\epsilon \ll 0$, $||s_{x_1}(t)|| \cdot \epsilon$ gives us a close estimate of $||\tau_1(t) - \tau_2(t)||$. For linear systems, this estimate is exact as all the higher order terms vanish. However, for nonlinear systems, this estimate is

neither an overapproximation, nor an underapproximation. In contrast to sensitivity analysis, discrepancy function requires that $\beta(x_2, x_1, t)$ is an overapproximation of $V(\tau_1(t), \tau_2(t))$ for both linear and nonlinear systems. □

3.4 Algorithmic Techniques For Computing Discrepancy Functions

We have seen in Sections 3.3.1, 3.3.2, and 3.3.3 that discrepancy function generalizes proof certificates used for establishing convergence or divergence of trajectories. Typically, these proof certificates are derived manually. In [22], the authors present a technique to establish exponential convergence among trajectories using *Sum Of Squares* (SOS) techniques. Informally, it searches for a contraction metric that satisfies conditions given in Definition 11. The technique works as follows:

1) Select the degree of the polynomial d for contraction metric $M(x)$. That is, all the terms in the contraction metric are fixed degree polynomial terms in the n real variables. For example, the general form of $M(x)$ for a two dimensional system with variables u and v is given as $\begin{pmatrix} \Sigma a_{ij} v^i u^j & \Sigma b_{ij} v^i u^j \\ \Sigma c_{ij} v^i u^j & \Sigma d_{ij} v^i u^j \end{pmatrix}$.

2) Calculate $R(x) = \frac{\partial f}{\partial x}^T M(x) + M(x)\frac{\partial f}{\partial x} + \dot{M}(x)$ and enforce constraints on a_{ij}, b_{ij}, c_{ij} and d_{ij} such that $R(x)$ is symmetric.

3) Impose the restrictions that polynomials $y^T M(x) y$ and $-y^T R(x) y$ are sum of squares polynomials and solve for the feasibility using SOS tools. If the solution exists, then the SOS solver will find values of coefficients of polynomials.

4) If the solution is feasible, compute the exponential rate of convergence by computing the value of γ such that $\frac{\partial f}{\partial x}^T M(x) + M(x)\frac{\partial f}{\partial x} + \dot{M}(x) + \gamma M(x) \prec 0$.

5) If SOS solver returns infeasible, then increase the degree of the polynomial terms in M and repeat.

For a given nonlinear ODE, the existence of a sum of squares polynomial as contraction metric is not guaranteed and hence the technique need not terminate. A

variant of the above presented technique can be used to compute incremental Lyapunov function. In Remark 2, we presented a technique for computing discrepancy function for exponentially stable linear systems, however, cannot be applied to all linear systems (such as time varying linear systems or linear systems that are not exponentially stable).

To overcome the disadvantages of the existing techniques, in this section, we present a new technique for computing bounded time discrepancy functions for linear systems. The techniques presented relies on computing sample simulations of the system. We call these sample simulations as *validated simulations*. Typical numerical simulations return a sequence of sampled states of a trajectory at regular time instances. Validated simulations, on the other hand, return a sequence of regions that encloses the trajectory for specified time durations. We use these validated simulations not just for computing discrepancy functions, but also for dynamic analysis techniques presented in Section 3.5. In the rest of this section, we give a formal definition of validated simulation and present the algorithm for computing discrepancy function for any linear systems using validated simulations. We also discuss other algorithmic techniques for computing discrepancy function in literature such as [91, 90, 60].

3.4.1 Validated Simulations

Definition 14 *Consider a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by ODE in Equation 3.5. Given a trajectory $\tau \in \mathcal{T}$ starting from $x_0 \in \mathbb{R}^n$ of duration $T > 0$, time step $h > 0$, and an error bound $\epsilon > 0$, an (x_0, T, ϵ, h)-simulation of trajectory τ is a finite sequence $\psi = (R_1, [t_0, t_1]), \ldots, (R_k, [t_{k-1}, t_k])$ such that, $\forall i > 0, R_i \subseteq \mathbb{R}^n$, $t_i \in \mathbb{R}_{\geq 0}$ and*

(1) $\forall\ i \in \{0, \ldots k-1\}, t_{i+1} - t_i \leq h, t_0 = 0,$ *and* $t_k = T,$

(2) $\forall\ i > 0, \forall t \in [t_{i-1}, t_i], \tau(t) \in R_i,$ *and*

(3) $\forall\ i > 0, diameter(R_i) \leq \epsilon.$

Figure 3.3 illustrates an (x_0, T, ϵ, h)-simulation of the trajectory τ. As opposed to providing sample states of the trajectory at regular intervals, a validated simulation returns a sequence of time stamped regions as shown in Figure 3.3. The diameter of each of the regions is bounded by ϵ and for any given time instance $t \in [t_{i-1}, t_i], \tau(t) \in R_i$. Given any pair ϵ, h such a validated simulation for a trajectory τ need not exist.

For computing validating simulations, we instead do the following: fix the time step h and use numerical solvers such as CAPD [2] and VNODE-LP [132] to compute validated simulation that encloses the trajectory τ. We then compute the value of ϵ such that the returned validated simulation is an (x_0, T, ϵ, h)-simulation. Given an initial state $x_0 = \tau(0)$ and time bound $T = \tau.dur$, the function $\mathsf{valSim}(x_0, T, h, f)$ returns the pair $\langle \psi, \epsilon \rangle$ such that ψ is an (x_0, T, ϵ, h)-simulation. In this thesis, we assume that validated simulations can be made arbitrarily precise, i.e, $\epsilon \to 0$ as $h \to 0$. In practice, we have observed that CAPD indeed generates simulations that are arbitrarily precise up to the order of 10^{-7}. We now present the technique to compute discrepancy function using validated simulations for linear systems.

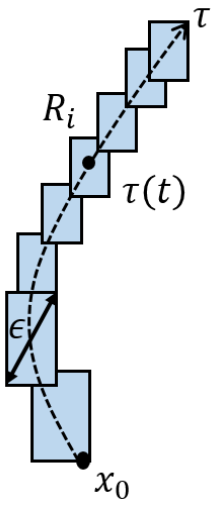


Figure 3.3: Illustration of a validated simulation for a trajectory τ.

3.4.2 Computing Discrepancy Function For Linear Systems From Validated Simulations

To compute discrepancy function for linear system from simulations, we exploit the linearity property of the solutions of ODEs. We define distance function for two validated simulations and then present an algorithm for computing a bounded time discrepancy function that uses the distance function for validated simulations.

Definition 15 *Let* $\psi_1 = (R^1_1, [t^1_0, t^1_1]), \ldots, (R^1_{k_1}, [t^1_{k_1-1}, t^1_{k_1}])$, *be an* (x_1, T, ϵ, h)*-simulation of trajectory* τ_1 *and* $\psi_2 = (R^2_1, [t^2_0, t^2_1]), \ldots, (R^2_{k_1}, [t^2_{k_1-1}, t^2_{k_1}])$ *be an* (x_2, T, ϵ, h)*-simulation of trajectory* τ_2*, the distance function* $d_{\langle \psi_1, \psi_2 \rangle} : \mathbb{R}_{\geq 0} \to \mathbb{R}_{\geq 0}$ *is such that given* $t \in [0, T]$

1. *If* $t = 0$, $d_{\langle \psi_1, \psi_2 \rangle}(0) = dist_{max}(R^1_1, R^2_1)$.
2. *If* $t \in (t^1_{l-1}, t^1_l], t \in (t^2_{m-1}, t^2_m]$, $d_{\langle \psi_1, \psi_2 \rangle}(t) = dist_{max}(R^1_l, R^2_m)$.

Informally, for two validated simulations ψ_1 and ψ_2, the distance function $d_{\langle \psi_1, \psi_2 \rangle}$ is a piecewise constant function such that, for a given time instance t, $d_{\langle \psi_1, \psi_2 \rangle}(t)$ gives the maximum distance between the two corresponding regions R^1_l and R^2_m that enclose the trajectories τ_1 and τ_2 at the time instance t respectively. Proposition 9

formalizes that such distance function gives an upper bound on the distance between trajectories.

Proposition 9 *Given* $\psi_1 = (R^1_1, [t^1_0, t^1_1]), \ldots, (R^1_{k_1}, [t^1_{k_1-1}, t^1_{k_1}])$, *an* (x_1, T, ϵ, h)*-simulation of trajectory* τ_1 *and* $\psi_2 = (R^2_1, [t^2_0, t^2_1]), \ldots, (R^2_{k_1}, [t^2_{k_1-1}, t^2_{k_1}])$, *an* (x_2, T, ϵ, h)*-simulation of trajectory* τ_2, *we have that* $\forall t \in [0, T], d_{\langle\psi_1,\psi_2\rangle}(t) \geq ||\tau_1(t) - \tau_2(t)||$.

Proof: The proof follows trivially for $t = 0$ as from Definition 15, $d_{\langle\psi_1,\psi_2\rangle}(0) = dist_{max}(R^1_1, R^2_1)$, and from Definition 14, we have that $\tau_1(0) \in R^1_1$ and $\tau_1(0) \in R^2_1$.

Consider $t \in (t^1_{l-1}, t^1_l], t \in (t^2_{m-1}, t^2_m]$. It follows from Definition 14 that $\tau_1(t) \in R^1_l$ and $\tau_2(t) \in R^2_m$. Since $d_{\langle\psi_1,\psi_2\rangle}(t) = dist_{max}(R^1_l, R^2_m)$. It follows that $d_{\langle\psi_1,\psi_2\rangle}(t) \geq ||\tau_1(t) - \tau_2(t)||$. ■

Intuitively, Proposition 9 states that the maximum distance between two regions that enclose the trajectories for a time instance gives an upper bound on the distance between the trajectories at that instance.

We now present Algorithm 3.1 that computes a discrepancy function for linear system by computing n distance functions. The algorithm computes the distance between two trajectories by exploiting the superposition principle for linear systems. The distance between trajectories is expressed as the sum of n distance functions from which an upper bound is computed.

input : $\langle\mathcal{V}, \mathcal{T}\rangle$, ϵ, h, T
output: Discrepancy function V, and its witness $\langle\alpha_1, \alpha_2, \beta\rangle$
1 Select an initial state x_0, and orthonormal basis $\{v_1, \ldots, v_n\}$;
2 $\langle\psi_0, \epsilon_0\rangle \leftarrow \mathsf{valSim}(x_0, T, h)$;
3 **for** *each* v_i **do**
4 $\quad \langle\psi_i, \epsilon_i\rangle \leftarrow \mathsf{valSim}(x_0 + v_i, T, h)$;
5 $\quad d_i \leftarrow d_{\langle\psi_0,\psi_i\rangle}$; `// Distance function between` ψ_i `and` ψ_0
6 **end**
7 $d \leftarrow \sum_{i=1}^{n} d_i$; `// Sum of distance functions in line 5`
8 **return** $V(x_1, x_2) \triangleq ||x_1 - x_2||$, $\beta(x_1, x_2, t) \triangleq ||x_1 - x_2||d(t)$;

Algorithm 3.1: Algorithm for computing discrepancy function for linear systems.

The algorithm 3.1 first selects an initial state x_0 and an orthonormal basis $\{v_1, \ldots, v_n\}$ for $\mathbb{R}^n$. In the loop from lines 3 to 6, it then generates $n + 1$ validated simulations

$\psi_0, \psi_1, \ldots, \psi_n$ of trajectories starting from $x_0, x_0+v_1, \ldots, x_0+v_n$ respectively. The distance functions $d_{\langle\psi_0,\psi_1\rangle}, \ldots, d_{\langle\psi_0,\psi_n\rangle}$ are added in line 7. This sum of distance functions $d(t)$ when multiplied with the initial distance between trajectories $||x_1 - x_2||$ returns the product $||x_1 - x_2||d(t)$. This product is an upper bound on the distance between trajectories τ_1 and τ_2 at time t, i.e. $||\tau_1(t) - \tau_2(t)|| \leq ||x_1 - x_2||d(t)$. Hence, the function $V(x_1, x_2) \triangleq ||x_1 - x_2||$ is a discrepancy function with $\beta(x_1, x_2, t) = ||x_1 - x_2||d(t)$. Proposition 11 establishes that Algorithm 3.1 computes a discrepancy function for any continuous system defined by a linear ODE.

Proposition 10 *Consider a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by a linear ODE $\dot{x} = A(t)x + B(t)$ where $A(t)$ and $B(t)$ are possibly time varying and Φ be the state transition matrix that defines the solution of ODE. Given any $t \in [0, T]$, we have that $||\Phi(t, 0)v_i|| \leq d_i(t)$ where v_i is the i^{th} vector in the orthonormal basis selected in line 1 and d_i is the distance function between ψ_0 and ψ_i that is computed in line 5 of Algorithm 3.1.*

Proof: As Φ is the state transition matrix that defines the solution of linear ODE $\dot{x} = A(t)x+B(t)$, for any trajectory $\tau \in \mathcal{T}$, we have $\tau(t) = \Phi(t, 0)\tau(0)+\int_0^t \Phi(t, s)B(s)ds$. If τ_0 and τ_i correspond to the trajectories starting from x_0 and $x_0 + v_i$, we have that $\tau_0(t) = \Phi(t, 0)\tau_0(0) + \int_0^t \Phi(t, s)B(s)\, ds$ and $\tau_i(t) = \Phi(t, 0)\tau_i(0) + \int_0^t \Phi(t, s)B\, ds$. It follows that

$$\begin{aligned} ||\tau_0(t) - \tau_i(t)|| &= ||\Phi(t, 0)\tau_0(0) + \int_0^t \Phi(t, s)B(s)\, ds \\ &\qquad - (\Phi(t, 0)\tau_i(0) + \int_0^t \Phi(t, s)B(s)\, ds)|| \\ &= ||\Phi(t, 0)(\tau_0(0) - \tau_i(0))|| \\ &= ||\Phi(t, 0)v_i||. \end{aligned}$$

Also, from Proposition 9 we have that $||\tau_0(t) - \tau_i(t)|| \leq d_i(t)$. Hence we have $||\Phi(t, 0)v_i|| \leq d_i(t)$. ■

Proposition 11 *Given a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$ defined by a linear ODE $\dot{x} = A(t)x + B(t)$ where $A(t)$ and $B(t)$ are possibly time varying. The function V and β returned by Algorithm 3.1 are such that V is a bounded time discrepancy function of the system with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ where $\alpha_1 = \alpha_2 = ||x_1 - x_2||$.*

Proof: Let Φ be the state transition matrix that defines the solution of the ODE $\dot{x} = A(t)x + B(t)$, i.e., for any trajectory $\tau \in \mathcal{T}$, $\tau(t) = \Phi(t,0)\tau(0) + \int_0^t \Phi(t,s)B(s)\,ds$. Consider two trajectories τ_1 and τ_2 in $\mathcal{T}$ such that $\tau_1(0) - \tau_2(0) = \eta_1 v_1 + \ldots + \eta_n v_n$, where $v_1, \ldots, v_n$ are the orthonormal bases selected in line 1 of Algorithm 3.1 and $\forall 0 \leq i \leq n, \eta_i \in \mathbb{R}$. Note that such scalars $\eta_1, \ldots, \eta_n$ exist because $\{v_1, \ldots, v_n\}$ is an orthonormal basis. Also note that as we use ℓ^2 norm, we have that $|\eta_i| \leq ||\tau_1(0) - \tau_2(0)||$. Now, using the solution of the linear ODE, we have that

$$
\begin{aligned}
||\tau_1(t) - \tau_2(t)|| &= ||\Phi(t,0)\tau_1(0) + \int_0^t \Phi(t,s)B(s)\,ds \\
&\qquad - (\Phi(t,0)\tau_2(0) + \int_0^t \Phi(t,s)B(s)\,ds)|| \\
&= ||\Phi(t,0)(\tau_1(0) - \tau_2(0))|| \\
&= ||\Phi(t,0)(\eta_1 v_1 + \ldots + \eta_n v_n)|| \\
&= ||\eta_1 \Phi(t,0)v_1 + \ldots + \eta_n \Phi(t,0)v_n|| \\
&\leq |\eta_1| \cdot ||\Phi(t,0)v_1|| + \ldots + |\eta_n| \cdot ||\Phi(t,0)v_n|| \\
&\leq |\eta_1| \cdot d_1(t) + \ldots + |\eta_n| d_n(t) \quad \text{from Proposition 10} \\
&\leq ||\tau_1(0) - \tau_2(0)|| \cdot d_1(t) + \ldots + ||\tau_1(0) - \tau_2(0)|| \cdot d_n(t) \\
&= ||\tau_1(0) - \tau_2(0)|| \cdot d(t) \\
&= \beta(\tau_1(0), \tau_2(0), t).
\end{aligned}
$$

Hence $V(\tau_1(t), \tau_2(t)) \leq \beta(\tau_1(0), \tau_2(0), t)$ which satisfies the third condition in the Definition 9. As $\beta(x_1, x_2, t) = ||x_1 - x_2||d(t)$, it follows that $\beta(x_1, x_2, t) \to 0$ as $||x_1 - x_2|| \to 0$. Uniform continuity of β follows from boundedness of $d(t)$ in $[0, T]$. Observing that the the first two conditions of Definition 9 are satisfied with $\alpha_1 = \alpha_2 = ||x_1 - x_2||$ is trivial. Hence V is a discrepancy function with $\langle \alpha_1, \alpha_2, \beta \rangle$ as witness. ■

Algorithm 3.1 can be used to compute discrepancy function for *any* linear system (both time invariant and time varying). Further, it only relies on the procedure valSim to generate validated simulations and hence is agnostic of the underlying model. Also notice that the discrepancy function obtained is a function of the orthonormal bases selected in line 1. Typically, these orthonormal bases can be the standard coordinate axes. The strategy to pick the orthonormal bases in order to get the tightest discrepancy function is still an open problem and is part of future work. As Algorithm 3.1 relies only on sample simulations to compute discrepancy function, it

enables *complete black box verification for linear systems* when applied together with the dynamic analysis presented in Section 3.5.

Remark 4 *Alternative approaches for computing discrepancy functions automatically*: Research on computing discrepancy functions (and other proof certificates in control theory) automatically has recently received much attention recently [60, 91, 90, 27, 103]. We highlight the contributions of relevant results and contrast it with the technique in Algorithm 3.1.

In [103], the authors propose a new technique for computing Lyapunov functions guided by the sample simulations of the system. What separates [103] from earlier computational approaches such as [46, 135] is that the search for Lyapunov function is guided by sample simulations. Similar technique that searches for contraction metrics using sample simulations is presented in [27]. These techniques require an underlying oracle to answer if the Lyapunov function (or the contraction metric) inferred is correct or not. Unlike these techniques, Algorithm 3.1 is agnostic of the underlying model and does not require any oracle.

In [91, 90], the authors present *input-to-state discrepancy function*, an extension of discrepancy to continuous systems with inputs. Using input-to-state discrepancy function, the authors present a new compositional approach for computing discrepancy function for a network of continuous and hybrid system from the input-to-state discrepancy function of the modules. Such techniques compliment the techniques present in this thesis.

In [60], the authors present a new technique for computing discrepancy function for nonlinear systems from sample simulations. Given a validated simulation ψ of τ, and radius δ, the technique presented in [60] returns a function β that gives an upper bound on the distance between trajectories starting at most δ distance away from $\tau(0)$. While technique in [60] can be applied to trajectories of nonlinear ODEs, it performs static analysis of the ODE and hence is not agnostic of the underlying model. □

3.5 Dynamic Analysis Of Continuous Systems

In this section we present the algorithm for performing dynamic analysis of a continuous system for which a discrepancy function and its witness are provided. Given an initial set of states denoted as Θ, the goal of dynamic analysis is to formally

prove whether a given property is satisfied (or violated) by all the trajectories starting from Θ by analyzing a finite number of simulations. Typically for continuous systems, the set Θ is uncountable and hence exhaustive simulation is impossible. Instead, for performing dynamic analysis, we rely on discrepancy function to compute overapproximations of trajectories in a neighborhood.

The dynamic analysis technique (Algorithm 3.4) has 4 main steps: *simulate*, *bloat*, *check*, and *refine* given as follows.

Simulate: A cover of the initial set Θ is computed as a union of neighborhoods, i.e. $\cup_{i=1}^{m} B_\delta(x_i)$ and a *simulation* is generated from each x_i.

Bloat: An overapproximation of all the behaviors in the neighborhood $B_\delta(x_i)$ is computed by *bloating* the simulation using the discrepancy function. This bloated simulation, which overapproximates the behaviors of all trajectories starting from a neighborhood is called *reachtube* (given in Definition 16).

Check: For each of these reachtubes, it is *checked* whether the property is satisfied or violated.

Refine: If the property is satisfied by all the reachtubes, or if a counterexample is discovered, we return the verification result, else, we *refine* the cover by computing smaller neighborhoods.

As the partition of the initial set gets finer, the order of overapproximation computed by rechtubes decreases. Hence, when the system robustly satisfies or violates the property, the overapproximation computed with finer partition would eventually either prove or disprove the property and the algorithm terminates.

For dynamic analysis, we save the overapproximations of all the trajectories from an initial set Θ in a *tree* data structure. This tree is called *reachtree* (given in Definition 17) and each element in this tree contains a *reachtube* for a given set of states. In this section, we present the definitions and procedures required for computing *reachtree* and present an algorithm for verifying the safety property of the continuous system with respect to a given unsafe set.

3.5.1 Reachtubes And Reachtrees

The definition of validated simulation that computes a sound overapproximation of trajectory has been given in Definition 14. In this section, we generalize that definition

to not just a single state, but a set of states and show how such reachtubes can be made arbitrarily precise.

Definition 16 *Given a continuous system* $\langle \mathcal{V}, \mathcal{T} \rangle$*, and an initial set of states* S*, an* (S, T, ϵ, h)-reachtube *of the trajectories starting from* S *for bounded time* T *is given as* $\phi = (R_1, [t_0, t_1]), \ldots, (R_k, [t_{k-1}, t_k])$ *such that,* $\forall i > 0, R_i \subseteq \mathbb{R}^n, t_i \in \mathbb{R}_{\geq 0}$*, and*

(1) $\forall\ i \in \{0, 1, \ldots k\}, t_{i+1} - t_i \leq h, t_0 = 0,$ *and* $t_k = T$*,*

(2) $\forall\ i > 0, \forall t \in [t_{i-1}, t_i], \forall \tau \in \mathcal{T},$ *such that* $\tau(0) \in S, \tau(t) \in R_i,$ *and*

(3) $\forall\ i > 0, diameter(R_i) \leq \epsilon$*.*

The i^{th} region in the reachtube ϕ, i.e. R_i is sometimes referred as $\phi[i]$ and the interval $[t_{i-1}, t_i]$ is referred as $\phi.dur[i]$. Similar to validated simulation, the reachtube from a given set of states S need not satisfy the error bound ϵ for a given time step h. Algorithm 3.2 defines a procedure called reachTube for computing a reachtube that uses valSim as a subroutine when the discrepancy function V with witness $\langle \alpha_1, \alpha_2, \beta \rangle$ are provided.

input : S, h, T, V, $\langle \alpha_1, \alpha_2, \beta \rangle$
output: $\langle \phi, \epsilon \rangle$ such that ϕ is an (S, T, ϵ, h)-*reachtube* from S
1 $x_0 \leftarrow center(S)$;
2 $\langle \psi, \epsilon_1 \rangle \leftarrow \mathsf{valSim}(x_0, T, h)$;
3 $\epsilon_2 \leftarrow sup_{x \in S, t \in [0,T]} \beta(x_0, x, t)$;
4 $\epsilon_3 \leftarrow sup_{x_1, x_2, \alpha_1(||x_1 - x_2||) \leq \epsilon_2} ||x_1 - x_2||$;
5 **for** *each* $(R_i, [t_{i-1}, t_i]) \in \psi$ **do**
6 | Add $(R'_i, [t_{i-1}, t_i])$ to ϕ such that $R'_i = B_{\epsilon_3}(R_i)$;
7 **end**
8 **return** $\langle \phi, \epsilon_1 + \epsilon_3 \rangle$;

Algorithm 3.2: Algorithm for computing *reachtubes* using valSim

The algorithm generates a validated simulation ψ from the center of the set of states S. It then computes the maximum distance between two trajectories starting from S as $\epsilon_1 + \epsilon_3$ computed in lines 3 and 4 using the witness of the discrepancy function β. It then bloats the validated simulation ψ by the maximum distance to compute the reachtube from the set of states S. An Illustration of the algorithm for computing a reachtube is shown in Figure 3.4.

Proposition 12 *The procedure* reachTube(S, T, h) *returns* $\langle \phi, \epsilon \rangle$ *such that* ϕ *is an* (S, T, ϵ, h)-reachtube.

Proof: Suppose that the validated simulation $\psi = (R_1, [t_0, t_1]), \ldots, (R_k, [t_{k-1}, t_k])$ and the reachtube given as $\phi = (R'_0, [t_0, t_1]), \ldots, (R'_k, [t_{k-1}, t_k])$. As ψ is a validated simulation, it follows that for $t \in [t_{l-1}, t_l]$ the trajectory starting τ_0 from x_0 is contained in region R_l, i.e. $\tau_0(t) \in R_l$.

Consider a trajectory τ starting from $x \in S$. From discrepancy function, we know that $V(\tau(t), \tau_0(t)) \leq \beta(x, x_0, t)$. As ϵ_2 in line 3 computes the supremum value of β, we have that $V(\tau(t), \tau_0(t)) \leq \epsilon_2$. From discrepancy function, we know that $V(\tau(t), \tau_0(t)) \geq \alpha_1(||\tau(t) - \tau_0(t)||)$, thus in line 4 the value of ϵ_3 computed is the maximum distance between the two trajectories i.e. $||\tau(t) - \tau_0(t)|| \leq \epsilon_3$. Hence $\tau(t) \in B_{\epsilon_3}(R_i) = R'_i$. Observe that $diameter(R'_i) \leq \epsilon_1 + \epsilon_3$. Hence ϕ is an (S, T, ϵ, h)-reachtube with $\epsilon = \epsilon_1 + \epsilon_3$. ■

Corollary 13 *Reachtubes can be made arbitrarily precise. That is,* $\forall \epsilon > 0$, $\exists \delta > 0, h > 0$ *such that whenever* $diameter(S) \leq \delta$, reachTube(S, T, h) *returns* ϕ *such that* ϕ *is an* (S, T, ϵ, h)*-reachtube.*

This observation follows from the fact that the diameter of reach region returned by valSim (ϵ_1) can be made arbitrarily small by decreasing the time step h. Further, it follows from the uniform continuity property of β that $\exists \delta > 0$ such that ϵ_3 computed in line 4 is less than $\epsilon - \epsilon_1$.

For computing an overapproximation of trajectories from a given set of initial states Θ, one can directly compute reachTube(Θ, T, h). However, this overapproximation, for some cases, may be too coarse to infer the property of interest. Hence, in order to improve the order of overapproximation, one might have to consider a collection of smaller neighborhoods in the set Θ, compute reachTube from each of the smaller neighborhoods and then combine these reachtubes together to infer whether all trajectories from Θ satisfy the property of interest. We introduce a notion of *reachtree* that helps in computing finer oveapproximations of all trajectories from Θ.

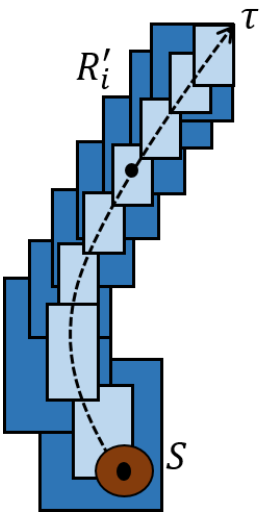


Figure 3.4: Illustration of a reachtube obtained by bloating a validated simulation.

We now introduce the notion of a *reachtree* for a given set of initial states Θ, time bound T, partitioning parameter $\delta > 0$, and time step h. A (Θ, T, δ, h)-reachtree has a root, denoted as #. Each of the child elements to the root has three attributes. They are given as

1. $\mathsf{InitialSet}$ refers to a set of states S such that $S = B_\delta(x)$ for some $x \in \mathbb{R}^n$ and $S \subseteq \Theta$.

2. tag is a value in $\{\mathsf{may}, \mathsf{must}\}$. It is set to must if $S \subset \Theta$ and is set to may otherwise.

3. RT stores $\mathsf{reachTube}(\mathsf{S}, \mathsf{T}, \mathsf{h})$, a reachtube that contains all the trajectories from the initial set S for bounded time T.

Formal definition of *reachtree* for continuous systems is given in Definition 17.

Definition 17 *Given a continuous system $\langle \mathcal{V}, \mathcal{T} \rangle$, an initial set Θ, a partitioning parameter $\delta > 0$, time bound T, and time step h, a $(\Theta, T, \delta, h)-$*reachtree *is a tree of depth 1 with root node denoted as # with child elements $E_1, E_2, \ldots, E_m$, where the child elements has three attributes* $\mathsf{InitialSet}$, tag, *and* RT *such that*

(1) $E_i.\mathsf{InitialSet} = B_\delta(x)$ *for some* $x \in \mathbb{R}^n$, $E_i.\mathsf{InitialSet} \cap \Theta \neq \emptyset$, *and* $\Theta \subseteq \cup_{i=1}^m E_i.\mathsf{InitialSet}$,

(2) $E_i.\mathsf{tag} = \mathsf{must}$ *if* $E_i.\mathsf{InitialSet} \cap \Theta^c = \emptyset$ *is* may *otherwise, and*

(3) $E_i.\mathsf{RT} = \mathsf{reachTube}(E_i.\mathsf{InitialSet}, T, h)$.

The algorithm to construct reachtree is given in Algorithm 3.3. It first computes a δ-cover of the initial set. For each element in the δ-cover, it creates a new child to the root node and sets the $\mathsf{InitialSet}$ attribute as the neighborhood $B_\delta(x)$. The tag attribute is set as must if the neighborhood is contained in the initial set Θ. The attribute RT stores the reachtube from $B_\delta(x)$ computed using $\mathsf{reachTube}$ procedure. An illustration of the reachtree is shown in Figure 3.5.

3.5.2 Dynamic Analysis For Safety Verification

In this section, we will present an algorithm for verifying a special class of properties called *safety* properties for continuous systems. A safety property is satisfied by a

input : Θ, T, δ, h
output: reachtree Δ
1 Compute $\mathcal{X} = \{B_\delta(x_1), \ldots, B_\delta(x_m)\}$, a δ-cover of Θ;
2 $\Delta \leftarrow Tree()$ with root node as #;
3 **for** *each* $B_\delta(x_i) \in \mathcal{X}$ **do**
4 Create node E_i with $E_i.\mathsf{InitialSet} = B_\delta(x_i)$;
5 **if** $B_\delta(x_i) \subset \Theta$ **then** $E_i.\mathsf{tag} = \mathsf{must}$;
6 **else** $E_i.\mathsf{tag} = \mathsf{may}$;
7 $E_i.\mathsf{RT} \leftarrow \phi_i$ where $\langle \phi_i, \epsilon_i \rangle \leftarrow \mathsf{reachTube}(B_\delta(x_i), h, T)$;
8 Set # as parent to E_i
9 **end**
10 **return** Δ

Algorithm 3.3: Algorithm to compute reachtree.

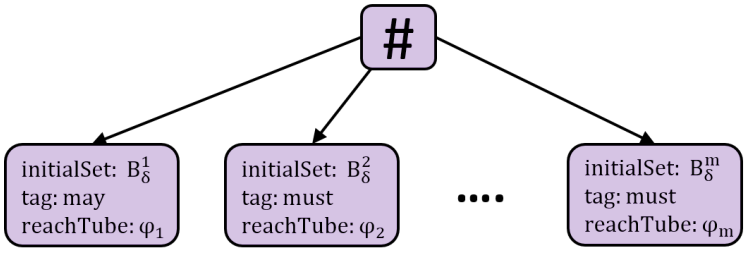


Figure 3.5: Illustration of a reachtree from an initial set Θ.

continuous system from an initial set of states, if all the trajectories starting from the initial set do not enter a specified *unsafe set*. The safety verification procedure iteratively computes reachtrees with increasing precision and checks whether the reachtree is safe or not. The loop terminates if the safety is established, or a counterexample to the safety property is found. In this section, we present the algorithm formally and prove that the algorithm gives theoretical guarantees, such as soundness and relative completeness.

Definition 18 *Given a reachtree Δ and an unsafe set U, the procedure* checkSafety *returns safe or unsafe or unknown if and only if*

- $\mathsf{checkSafety}(\Delta, U) =$ *safe if* $\forall E_i \in \Delta$, $\forall j, E_i.\mathsf{RT}[j] \cap U = \emptyset$
- $\mathsf{checkSafety}(\Delta, U) =$ *unsafe if* $\exists E_i \in \Delta$, *such that* $E_i.\mathsf{tag} = \mathsf{must}$ *and* $\exists j, E_i.\mathsf{RT}[j] \subset U$.
- $\mathsf{checkSafety}(\Delta, U) =$ *unknown otherwise.*

Observe that the procedure checkSafety returns *safe* only when all the reachtubes in the nodes of the tree are safe, and it returns *unsafe* only when a node that is tagged must has an element in reachtube that is completely contained within the unsafe set U. From propositions 12, it follows that whenever the checkSafety procedure returns *unsafe*, then there exist at least one trajectory τ from Θ that is unsafe, and whenever checkSafety returns *safe*, all the trajectories are indeed safe. The algorithm for safety verification of continuous systems is given in Algorithm 3.4.

```
   input  : Θ, T, ⟨V, T⟩
   output: If the continuous system is safe or unsafe
1  Select δ > 0, h ∈ [0, T];
2  while true do
3  |  Δ ← reachTree(Θ, T, δ, h);
4  |  if checkSafety(Δ, U) = safe then  return safe;
5  |  else if checkSafety(Δ, U) = unsafe then  return unsafe;
6  |  else δ ← δ/2; h ← h/2;
7  end
```

Algorithm 3.4: Dynamic analysis algorithm for safety verification of continuous systems.

The algorithm iteratively computes reachtrees of increasing precision that will either prove that the system is safe or unsafe. If the checkSafety procedure for the tree does not return that the tree is safe or unsafe, then a better approximation is computed by refining the values of δ and h.

Theorem 14 *Algorithm 3.4 for verifying safety of continuous systems is* sound, *i.e. if the algorithm returns safe, then the trajectories starting from Θ for bounded time T are safe, and if it returns unsafe, then there exists a trajectory starting from Θ that is unsafe within the time bound T.*

Proof: From proposition 12 and Definition 17, it follows that for any trajectory $\tau \in \mathcal{T}$, starting from Θ, $\exists E_i \in \Delta$ such that $\tau(0) \in E_i.\mathsf{InitialSet}$ and hence $\forall t \in [0, T], \tau(t) \in E_i.\mathsf{RT}$. Hence, if the checkSafety procedure returns safe, all the trajectories are safe.

If the algorithm returns unsafe, then the checkSafety procedure for the reachtree returned unsafe. Hence, $\exists E_i \in \Delta$, such that $E_i.\mathsf{tag} = \mathsf{must}$ and $\exists j, E_i.\mathsf{RT}[j] \subset U$. Thus, from Proposition 12, we have that $\forall \tau \in \mathcal{T}$ with $\tau(0) \in E_i.\mathsf{InitialSet}$, $\exists t \in E_i.\mathsf{RT}.dur[j], \tau(t) \in U$ and hence the system is unsafe. ■

Theorem 15 *Algorithm 3.4 for verifying safety of continuous systems is* relatively complete*, i.e. if all the trajectories are robustly safe (Definition 6), then the algorithm will terminate and return safe, and if the trajectories are robustly unsafe, then the algorithm will terminate and return unsafe.*

Proof: Suppose that the system is robustly safe. Hence, $\exists \epsilon_r > 0$ such that, $\forall \tau \in \mathcal{T}$ starting from Θ, $B_{\epsilon_r}(\tau) \cap U = \emptyset$. First, it follows from Theorem 14 that the algorithm will not return *unsafe*. Therefore, the algorithm either returns *safe* or iterates by reducing the value of δ and h. From Corollary 13, it follows that $\exists \delta_r, h_r > 0$ such that all the reachtubes computed in $\mathsf{reachTree}(\Theta, \delta_r, h_r, T)$, i.e. $E_i.\mathsf{RT}$ are such that the diameter of the regions in the reachtubes is less than ϵ_r, i.e. $diameter(E_i.(RT)[j]) < \epsilon_r$. Hence, $\forall E_i \in \Delta$, $\forall j, E_i.\mathsf{RT}[j] \cap U = \emptyset$. Hence, the algorithm will terminate when δ is less than δ_r and h less than h_r and returns that the system is *safe*.

If the system is robustly unsafe, then $\exists \epsilon_r > 0$, $\exists \tau \in \mathcal{T}$ starting within Θ and $\exists t \in [0, T]$ such that $B_{\epsilon_r}(\tau(t)) \in U$. Hence, $\exists \delta > 0$ such that $\forall \tau' \in \mathcal{T}$ starting from $B_\delta(\tau(0))$, the state at time t is unsafe, i.e. $\tau(t) \in U$. From Corollary 13 we have that $\exists \delta_r, h_r > 0$ such that $\exists E_i \in \Delta$ and $\exists j, E_i.\mathsf{RT}[j] \subset R$. Hence, the algorithm will return *unsafe*. ■

We remark that in the case when some trajectories from Θ are safe but not robustly safe, Algorithm 3.4 may not terminate. Observe that relative completeness of the above algorithm is achieved mainly because of Corollary 13 where reachtubes are computed with arbitrarily small δ and h. In practice this may not be feasible as CAPD implements finite precision arithmetic. However, even if we have finite precision arithmetic, relative completeness is achieved if the system is robustly safe with ϵ_0 greater than the precision of solver. In the landscape of CPS verification, given that safety verification is undecidable in general, such theoretical guarantees of soundness and relative completeness are the strongest one can hope for.

In summary, we have seen the algorithm for constructing reachtrees, inferring safety of reachtrees, and their iterative refinement to obtain a positive (or negative) answer for the safety verification. We have implemented these algorithms and the results of experimental evaluation are given in Section 3.6.

Benchmark	Vars.	TH	Refs.	Sims.	C2E2 (sec)	*Flow** (sec)	Ariadne (sec)
Moore-Greitzer Jet Engine [22]	2	10	12	36	1.56	10.54	56.57
Brussellator	2	10	33	115	5.262	16.77	72.75
VanDerPol	2	10	5	17	0.75	8.93	98.36
Coupled VanDerPol [22]	4	10	10	62	1.43	90.96	270.61
Sinusoidal Tracking [156]	6	10	12	84	3.68	48.63	763.32
Linear Adaptive	3	10	8	16	0.47	NA	NA
Nonlinear Adaptive	2	10	16	32	1.23	NA	NA
NonLin. Adpt. + Disturbance	3	10	22	48	1.52	NA	NA

Table 3.1: Experimental results for benchmark models with C2E2. Vars: Number of Variables, TH: Time Horizon for Verification, Refs: Number of Refinements, Sims: Total number of simulation traces required for proving safety.

3.6 Experimental Evaluation

For demonstrating the effectiveness of our technique which uses discrepancy functions and simulations for verification, we have built a tool called **C**heck **E**xecute **C**ompare **E**ngine (C2E2). More details about the tool C2E2 are given in Section 4.4. C2E2 accepts initial set and the unsafe set that are given as convex polyhedron. The validated simulations required by the dynamic analysis technique are generated using the numerical integration engine CAPD [2]. CAPD uses finite precision interval arithmetic and Taylor models for generating simulations described in Section 3.4.1. The tool is developed in C++ and uses GNU Linear Programing Kit (GLPK) to check the distance of simulation traces from the unsafe set. Observe that the algorithm presented in Section 3.5.2 requires arbitrary precision arithmetic for relative completeness, however in practice, we use finite precision arithmetic which requires the robustness to be greater than precision of the solver. In practice we terminate if the partitioning reaches 10^{-5} and return *unknown*.

Our evaluation has four parts: First, we verify a benchmark suite consisting of natural linear and nonlinear dynamical system models. Second, we verify several adaptive control examples where the executable system has a reference model with unknown parameters. Third, we evaluate scalability by increasing the time horizon for verification and the number of dimensions of two parameterized models. Finally, we study the effect of the initial partitioning.

3.6.1 Benchmarks And Comparison With Other Tools

We compare the performance of C2E2 against *Flow** [37] and Ariadne [28] using a benchmark suite consisting of linear and nonlinear models. Ariadne uses faithful geometric representation of sets for computing reachable set and *Flow** uses Taylor models. Results of comparing the performance with Breach are discussed later in this section. In Table 3.1, we report the time taken by *Flow** and C2E2 for computing reach set and checking safety with respect to the unsafe set, and the time taken by Ariadne for computing the reachable states. Considering that runtime for *Flow** depends on remainder errors and order of Taylor model, we use adaptive orders and set the remainder error to a value such that increasing it by a factor of 2 would change the verification result from "SAFE" to "UNKNOWN". The parameters for Ariadne is set to the same values as that for *Flow**. Table 3.1 show that C2E2 outperforms the other tools in all examples, and in most cases it is faster by at least an order of magnitude. All the experiments were performed on an i7 Quadcore machine with 8GB memory running Ubuntu 11.10.

Breach [50] is another tool against which we evaluated C2E2 . Breach is a toolbox developed in MATLAB with a GUI, making a fair comparison difficult. We ran Breach on Vanderpol coupled oscillator, Sinusoidal tracking, and tank examples, and measured the "wall clock" running time. In all cases C2E2 was faster; we don't report these numbers because of inaccuracies inherent in such numbers. One limitation of Breach is that we cannot specify polyhedral unsafe sets. Also Breach is neither sound nor complete for nonlinear models, but inaccuracies in the verification results for the examples were not observed.

Tools dReal [71] and dReach [106] are also commonly used for safety verification of nonlinear continuous and hybrid systems. The model of the system is encoded in as a formula (which includes constructs for continuous evolution through ODEs) and the safety verification problem is encoded as an SMT instance and given as input to dReal and dReach. Contrary to C2E2 that either proves the system to the safe or provides a counterexample, dReal and dReach return that either the formula is satisfied (in which case the system is safe) or return a δ-unsat formula (in which case the δ-perturbation of the model is unsafe). Due to this difference in the guarantees provided by the verification algorithms, we feel that comparing C2E2 with dReal/dReach would be unfair.

For the adaptive control system, the n-tank system and the nonlinear navigation system the discrepancy functions were derived manually, and for all the other exam-

ples the discrepancy functions were obtained from the papers [22, 156] in which the systems appeared.

3.6.2 Systems Models With Unknown Parameters

One prominent advantage of our approach is that it supports verification of executable systems where the reference model has unknown parameters. An illustrative example is the linear plant:

$$\dot{x} = 1$$
$$\dot{y} = \theta + u$$

where θ is an unknown parameter and u is the control input. Following a standard adaptive control technique for driving y to zero, a new variable $\hat{\theta}$—an estimator for θ—is introduced giving the new dynamics for y:

$$\dot{x} = 1$$
$$\dot{y} = -\sigma y + \theta - \hat{\theta}$$
$$\dot{\hat{\theta}} = \gamma y$$

Constants $\sigma, \gamma > 0$ are chosen by the designer. For the new system, $V((x_1, y_1, \hat{\theta}_1), (x_2, y_2, \hat{\theta}_2)) \triangleq \frac{1}{2}(x_1 - x_2)^2 + \frac{\gamma}{2}(y_1 - y_2)^2 + \frac{1}{2}(\hat{\theta}_1 - \hat{\theta}_2)^2$ is an incremental Lyapunov function because $\dot{V} = -\gamma\sigma(y_1 - y_2)^2 < 0$ and was used as discrepancy function for verification.

Consider the nonlinear system:

$$\dot{x} = \theta x + u$$

with control input u. Similar to the earlier example, we introduce $\hat{\theta}$ and define $u = -\hat{\theta}x - x$ such that the closed system becomes:

$$\dot{x} = (\theta - \hat{\theta})x - x$$
$$\dot{\hat{\theta}} = x^2$$

The Lyapunov function $\frac{1}{2}x^2 + \frac{1}{2}(\theta - \hat{\theta})^2$ establishes stability of the system. For com-

Benchmark	Refs.	Sims.	Time
12-Tanks	1	16	2.744
18-Tanks	4	76	15.238
24-Tanks	4	100	22.126
30-Tanks	4	124	28.824
12-Vehicles	0	32	5.477
16-Vehicles	0	64	12.238
20-Vehicles	0	128	25.144
24-Vehicles	0	256	54.236

(a) Scalability of verification of for n-Tank and n-vehicle systems.

Benchmark	δ	Refs.	Sims.	Time
Nonlin. Adpt.	0.5	16	32	1.01
Nonlin. Adpt.	0.2	9	20	0.91
Nonlin. Adpt.	0.05	5	13	0.58
Nonlin. Adpt.	0.01	0	26	1.32
VanDerPol	0.5	30	96	3.84
VanDerPol	0.2	5	17	0.75
VanDerPol	0.05	8	32	1.44
VanDerPol	0.01	0	120	5.87

(b) Dependence of running time on initial state covers.

Figure 3.6: Scalability and Initial state covers. Refs: Number of refinements, Sims: Number of simulation traces, Time: Running time of C2E2 in seconds.

puting a discrepancy function, we come up with a quartic (fourth degree) discrepancy function. A modified version of this example introduces unknown disturbance inputs. Traditional model checkers that model unknown parameters as unknown constant variable require partial information (such as range of values) for handling these systems which is not required by our technique.

3.6.3 Scalability With Time And Dimensions

To check the scalability of the approach with time horizon, we verified the VanDerPol and the Nonlinear Adaptive control benchmarks for time horizons 10, 20 and 40. The verification times of the former were 0.741, 1.662 and 4.373 seconds and for the latter were 1.245, 2.604 and 6.075 seconds respectively. This suggests that the verification time scales roughly linearly with the time horizon for these stable systems.

Table 3.6(a) shows the scaling of the verification times with the number of dimensions. We consider the a switched variant of nonlinear version of the n interconnected tank system of [7] and a switched nonlinear model with $n/4$ vehicles and each vehicle having 4 continuous variables. For n-tank the initial value of δ is large and this triggers several refinements and for n-vehicles the initial value of δ is small and this decrease (or eliminates) the need for refinements. There are two key observations. First, as the number of refinements increase, the size of the cover increases proportional to the number of dimensions. This is evident in the case of n-tanks, where for the $n = 12$, 1 refinement step required checking of 16 executions, whereas in the case of $n = 24$, 4 refinement steps required 100 executions. Second, as the number of

dimensions increase, a smaller value of initial partitioning parameter δ increases the size of the cover exponentially. This is evident in the case of n-vehicles, where adding each new vehicle increased the number of simulations by a factor of 2.

3.6.4 Dependence On Initial Set Partition

Table 3.6(b) shows the verification times for different δ-coverings of the initial set. If the value of δ is too small, then C2E2 generates a large initial partitioning and hence increases the number of simulations. On the other hand, if the initial δ is too large, then C2E2 needs to perform many refinements, and hence, takes more time. The value of optimum value of δ clearly depends on that robustness of the system and the relative distance of simulations from unsafe set. Observe that in Table 3.6(b), the partitioning with $\delta = 0.2$ takes less time for verification than $\delta = 0.5$ and $\delta = 0.1$. Searching for the optimal value of δ is an interesting direction of future work.

3.7 Conclusions And Related Work

In this chapter we first presented *discrepancy function* for a continuous system that captures the notion of divergence or convergence of trajectories. We have established that these discrepancy functions are generalizations of proof certificates such as contraction metric and incremental Lyapunov functions routinely used in stability analysis of dynamical systems in control theory. We have also presented an algorithms for computing discrepancy function for linear systems using validated simulations. We then introduced a notion of reachTube that overapproximates the set of reachable states from a given set of initial states. We then provided an algorithm for computing reachTree from validated simulations and discrepancy functions from a given set of initial states. The safety verification algorithm that iteratively computes reachtrees with increasing precision until the system is inferred to be safe or unsafe. Finally, we presented experimental evaluation on several benchmark problems to conclude that the approach holds promise.

Dynamic analysis techniques for safety verification has been studied before in [78, 101]. The authors present a technique to compute Metric Transition System from sample simulations of linear systems. This thesis differs from [78] as it does not compute discrete transition system and can also be applied for nonlinear systems.

Sensitivity analysis has been used in [52] for computing overapproximation of trajectories in a neighborhood and proposes a safety verification algorithm using sample simulations. This thesis uses discrepancy functions for computing overapproximation, which can be applied to nonlinear systems, as opposed to [52] which can only be applied to linear systems. Sample simulations of the systems were used to compute symbolic traces for Simulink/Stateflow models in [102], however do not tackle the safety verification problem that is analyzed in this thesis. As opposed to this thesis, STRONG [49] applies a lazy approach and partitions the initial set based the distance from unsafe set.

Proof certificates such as Incremental Lyapunov functions and Incremental forward completeness have been used in the literature to compute finite bisimulation models in [82] and [169]. However, this thesis is the first work to propose the usage of proof certificates for safety verification.

Chapter 4

Dynamic Analysis of Hybrid Systems

In this chapter we present a dynamic analysis algorithm for *hybrid* systems where, the continuous state evolves based on the current location and can be reset during discrete transitions. We extend the notion of reachtree from continuous systems presented in Chapter 3 to hybrid systems. This extension requires handling invariants of each location and guards and resets for each discrete transition that enable the change of location. This chapter introduces two routines. First, invariantPrefix for handling the invariants for each location and second, nextRegions for computing the overapproximation of states that encounter a discrete transition. We present the safety verification algorithm for hybrid systems, establish its correctness, and prove its relative completeness property. We also describe the tool Compare-Execute-Check-Engine (C2E2) that implements the algorithm, its architecture, and present some verification results.

4.1 Introduction

In this chapter we extend the dynamic analysis algorithm presented in Chapter 3 to hybrid systems. We assume that the user provides discrepancy functions for each of the locations of the hybrid system. The reachTube routine that computes an overapproximation of trajectories for continuous systems gets the current location *loc* as additional argument.

There are, however, two main steps in extending the dynamic analysis from continuous to hybrid systems. First, is to handle the invariants, i.e., every trajectory τ in a given location l ($\tau \in \mathcal{T}_l$) respects the invariant of the location, i.e., $\forall t \in [0, \tau.dur], \tau(t) \in Inv_l$. The routine invariantPrefix is presented to handle this issue. Second, is to handle the discrete transitions. This involves making the following checks while computing an overapproximation of the reachable states: a given discrete transition a is only enabled if the current state of the system x satisfies the guard

predicate, i.e., $Guard_a(x)$ is true. Further, once the discrete transition is enabled, the state after the discrete transition is given by a reset map $Reset_a(x)$. This reset map $Reset_a(x)$, maps a given state x to a set of states N_x. The system nondeterministically chooses the next state of the execution from N_x. For handling the discrete transitions, we present nextRegions routine.

Using these two procedures, we extend the notion of reachtree for continuous systems defined in Section 3.5.1 to hybrid systems. The depth of the reachtrees of hybrid systems can be greater than 1. Further, each element in the reachtree has additional attributes which track the current location, time when the execution entered the location, and the number of discrete transitions taken. Similar to continuous systems, the safety verification procedure for hybrid systems constructs reachtrees iteratively until either the safety of the system is inferred or an execution (or its overapproximation) that violates the safety property is discovered.

This chapter is organized as follows. We first present the two new routines for handling invariants for location and discrete transition in Section 4.2. We then present the reachtree computation for hybrid systems in Section 4.2.1. The safety verification algorithm, its proof of soundness and relative completeness is presented in Section 4.3. We then present the architecture of the C2E2 that implements the dynamic analysis algorithm and discuss its features and user experience in Section 4.4. Finally, we conclude this chapter after presenting the experimental results in Section 4.5.

4.2 Reachtrees With Invariants And Discrete Transitions

Before introducing the new routines that handle the invariants for locations and guards for discrete transitions, we briefly recall the execution of a hybrid system from Section 2.2.2. Given a hybrid system $\mathcal{A} = \langle \mathcal{V}, A, D, \mathcal{T} \rangle$, an execution starting from an initial state x_0 and initial location loc_0 is given as alternating sequence of trajectories and actions, i.e., $\sigma = \tau_0 a_1 \tau_1 \ldots$ such that each τ_i is a trajectory and a_i is a discrete transition. Further, each trajectory always lies within the invariant of the given location, i.e., $\forall i$, $\tau_i \in \mathcal{T}_{\tau_i.loc}$ and $\forall t \in [0, \tau_i.dom], \tau_i(t) \in Inv_{loc}$. Moreover, a discrete transition is taken only when the guard is enabled and the starting state from the next location is nondeterministically chosen from the reset map of the discrete transition, i.e., $\tau_{i-1}.\mathsf{lstate} \in Guard_{a_i}$ and $\tau_i.\mathsf{fstate} \in Reset_{a_i}(\tau_{i-1}.\mathsf{lstate})$. The location after the discrete transition is given as $nextLoc_{a_i}$.

In Section 3.5.1, we have introduced a notion of reachtube for continuous systems. The procedure $\mathsf{reachTube}(S, h, T)$ computes an overapproximation of all the trajectories starting from S for bounded time T of a continuous system using its discrepancy function. For hybrid systems, we assume that each location l is provided with a discrepancy function V^l and witness $\langle \alpha_1^l, \alpha_2^l, \beta^l \rangle$. We extend the reachtube routine as $\mathsf{reachTube}(S, h, T, l)$ that computes a reachtube for trajectories in location l given as $\mathcal{T}_l$ for a bounded time T in a similar manner. Notice that for hybrid systems, we require that the trajectories satisfy the invariant of the location Inv_l at all times. To compute new reachtubes that satisfy this restriction, we define a procedure $\mathsf{invariantPrefix}$ which only considers trajectories that respect the invariant Inv_l. Before introducing the procedure, we present a definition for tagging.

Definition 19 (Tagging) *Given a region R and set P, $\mathsf{tag}(R, P)$ returns either a* must *or* may *or* not *as follows:*

1. $\mathsf{tag}(R, P) = \mathsf{must}$ *if and only if* $R \subseteq P$.
2. $\mathsf{tag}(R, P) = \mathsf{not}$ *if and only if* $R \subseteq P^c$.
3. $\mathsf{tag}(R, P) = \mathsf{may}$ *otherwise.*

Definition 20 (Invariant Prefix) *Given a reachtube $\psi = (R_1, [t_0, t_1]), \ldots, (R_k, [t_{k-1}, t_k])$ and a set P, $\mathsf{invariantPrefix}(\psi, P)$ returns the longest sequence $\phi = \langle R_1, tag_1, [t_0, t_1] \rangle, \ldots, \langle R_m, tag_m, [t_{m-1}, t_m] \rangle$, such that $\forall 1 \leq i \leq m$, tag_i is decided as*

- $tag_i = \mathsf{must}$ *if and only if* $\forall j \leq i$, $\mathsf{tag}(R_j, P) = \mathsf{must}$.
- $tag_i = \mathsf{may}$ *if and only if* $\forall j \leq i$, $\mathsf{tag}(R_j, P)$ *is either* must *or* may *and* $\exists q \leq i$, $\mathsf{tag}(R_q, P) = \mathsf{may}$.

That is, if $m < k$, then $tag(R_{m+1}, P) = \mathsf{not}$.

Intuitively, a region R_i is tagged must in an invariant prefix, if all the regions before R_i (including itself) are contained within the set P. It is tagged may if all the regions before it have nonempty intersection with P, and at least one of them (including itself) is not contained within the set P. Given a reachtube ψ from a set S, $\phi = \mathsf{invariantPrefix}(\psi, Inv_{loc})$ returns an overapproximation of the valid set of trajectories from S that respect the invariant of the location *loc* of the hybrid system. Thus, the set of all reachable states from S by a valid trajectory are contained within

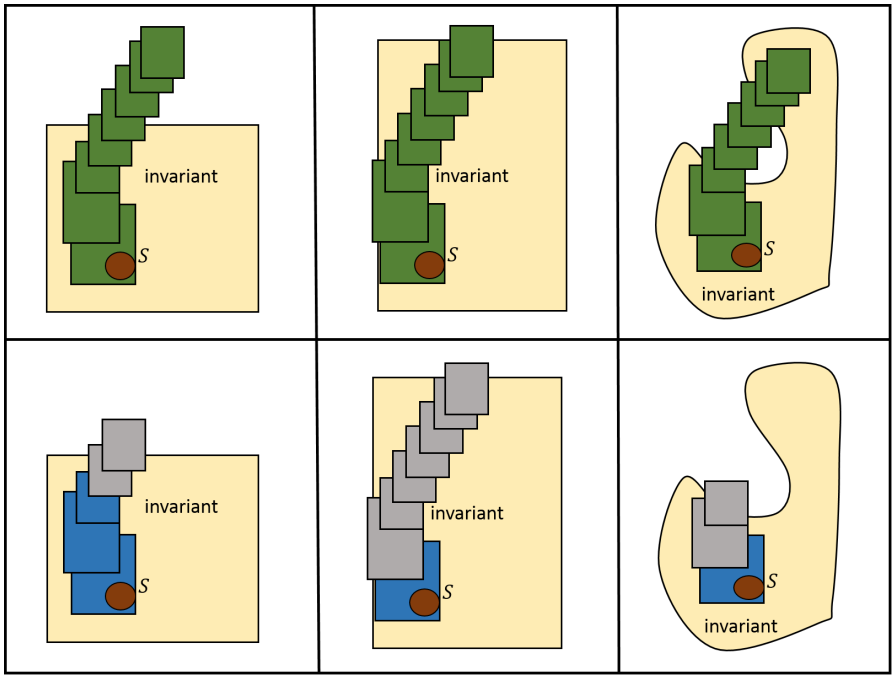


Figure 4.1: Figure illustrating the invariant prefix of reachtubes with respect to different invariants. The untagged regions in reachtube obtained from reachTube procedure are colored green. The regions in the invariant prefix that are tagged must are colored blue and the regions tagged may are colored gray.

ϕ. Also, from Definition 16 we have that, if a region R is tagged must, then there exists at least one trajectory starting from S that can reach R while respecting the invariant P. We abuse notation and say that $R \in \phi$ if $\langle R, tag, [t, t'] \rangle$ is an element of the sequence ϕ.

Illustrations of invariant prefix for a reachtube from S for several invariants is shown in Figure 4.1. Each of the figures in the top row shows a reachtube and an invariant, and the bottom row shows the corresponding invariant prefix. A region in invariant prefix is colored blue if it is tagged must and is colored gray if it is tagged may. The figures illustrate two things. First is that the invariant prefix does not contain the regions that violate the invariant (i.e., $\mathsf{tag}(R, P) = \mathsf{not}$) or the regions that appear after a region that violates the invariant. Second, is that the first region that is tagged may is not contained within the invariant and all the subsequent regions are tagged may.

We now introduce the routine nextRegions that computes overapproximation of the reachable states that serve as initial states in a new location after taking a discrete transition. A discrete transition a is enabled if the current state satisfies the guard

condition $Guard_a$. The state after the discrete transition is obtained by applying the reset map $Reset_a$. Given a sequence of tagged regions (obtained from invariantPrefix routine), the nextRegions routine returns the tagged set of reachable regions after a discrete transition.

Definition 21 (Next Regions) *Given* $\phi = \langle R_1, tag_1, [t_0, t_1]\rangle, \ldots, \langle R_m, tag_m, [t_{m-1}, t_m]\rangle$, *a sequence of tagged regions obtained from* invariantPrefix *of a location* l, *the subroutine* $nextRegions(\phi)$ *returns a set of tagged regions* $\mathbf{R}$. $\langle R', tag', loc', t'\rangle \in \mathbf{R}$ *if and only if there exists an action* a *from location* l *of the automaton and a region* R_i *in* ϕ *such that* $R' = Reset_a(R_i)$, $t' = t_{i-1}$, $loc' = nextLoc_a$ *and one of the following conditions hold:*

(a) $R_i \subseteq Guard_a, tag_i = \mathsf{must}, tag' = \mathsf{must}$.

(b) $R_i \cap Guard_a \neq \emptyset, R_i \nsubseteq Guard_a, tag_i = \mathsf{must}, tag' = \mathsf{may}$.

(c) $R_i \cap Guard_a \neq \emptyset, tag_i = \mathsf{may}, tag' = \mathsf{may}$.

A given tuple $\langle R', tag', loc', t'\rangle \in \mathbf{R}$ is tagged must only when the region R_i is tagged must and is contained within the $Guard_a$. In all other cases, i.e. when R_i is not completely contained within the guard, or if R_i is tagged may, the region is tagged may. This ensures that regions tagged must are indeed reachable after the discrete transition, and all the regions tagged may, may contain some reachable state after the discrete transition. Figure 4.2 illustrates the result of a next procedure on an invariant prefix. The reset map considered in the figure is the identity function. Notice that the regions that are tagged must by the nextRegions routine are tagged must in the invariant prefix and are contained in the guard. These regions are colored blue in the figure. The regions that are tagged may by the nextRegions routine are either tagged may in the invariant prefix or are not contained in the guard.

Assumption 2 *We restrict our attention to hybrid systems with reset maps that are uniformly continuous functions. For such systems, it follows that* $diameter(Reset_a(R)) \to 0$ *as* $diameter(R) \to 0$.

Notice that for each reachtube, multiple tagged regions are returned by nextRegions routine. For computing the overapproximation of the set of reachable states from a given initial set for a hybrid system, we construct reachtree. The elements of tree store the invariant prefixes, i.e., the set of states reachable from a trajectory that

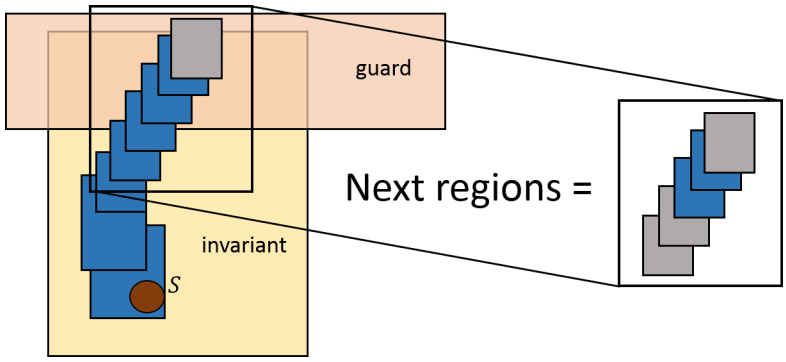


Figure 4.2: Figure illustrating the nextRegions procedure for an invariant prefix with respect to a guard. The regions in the invariant prefix that are tagged must are colored blue and the regions tagged may are colored gray. Same color convention is followed in labeling the regions returned by the nextRegions procedure.

respects the invariant of current location. If any discrete transitions are encountered by the trajectories, new child nodes are added to the current element that store the reachable set after a discrete transition (returned by nextRegions routine). The reachtree computation continues computing the reachtubes from initial set of child nodes and this process continues until the time bound or the bound on number of discrete transitions is reached. We now present the algorithm for computing the reachtree for hybrid systems.

4.2.1 Computing Reachtree For Hybrid Systems

The reachtree for hybrid systems is an extension of the reachtree for continuous systems presented in Section 3.5.1. In addition to the attributes InitialSet, tag, and RT, the elements in reachtree of hybrid systems have attributes location, time, and switches. These attributes track the location of the execution, the time of entering the location, and the number of discrete transitions taken to reach the location respectively. Additionally we have color attribute that helps in construction of the reachtree.

For a given system $\mathcal{A}$, reachtree algorithm takes as input the set of initial states Θ, the time bound T, the initial location of the system loc_0, and the bound on the number of discrete transitions N. Additionally, parameters $\delta > 0$, for constructing a cover of the initial set Θ and h, used as time step in computing reachtubes are also provided as inputs. The algorithm for computing reachtree is given in Algorithm 4.1.

The Algorithm 4.1 for hybrid systems is similar to the Algorithm 3.3. However, in

input : $\mathcal{A}$, Θ, δ, T, h, N, loc_0
output: reachTree Δ

Compute $\mathcal{X} = \{B_\delta(x_1), \ldots, B_\delta(x_m)\}$, a δ-cover of Θ;
$\Delta \leftarrow Tree()$ with root node as #;
for *each* $B_\delta(x_i) \in \mathcal{X}$ **do**
    $E_i \leftarrow newNode()$;
    E_i.InitialSet $\leftarrow B_\delta(x_i)$;
    if $B_\delta(x_i) \subset \Theta$ **then** E_i.tag $\leftarrow$ must;
    else E_i.tag $\leftarrow$ may;
    E_i.location $\leftarrow loc_0$;
    E_i.time $\leftarrow 0$;
    E_i.switches $\leftarrow 0$;
    E_i.color $\leftarrow gray$;
    Set # as parent to E_i;
end
for *each* $E \in T$ *with* E.color $= grey \wedge E$.switches $\leq N \wedge E$.time $\leq T$ **do**
    $\langle \psi, \epsilon \rangle \leftarrow$ reachTube(E.InitialSet, $T - E$.time, h, E.location);
    $\phi \leftarrow$ invariantPrefix(ψ);
    E.RT $\leftarrow \phi$;
    for *each* $\langle R', tag', loc', t' \rangle \in$ nextRegions(ϕ) **do**
        $F \leftarrow newNode()$;
        F.InitialSet $\leftarrow R'$;
        if E.tag $=$ may **then** F.tag $\leftarrow$ may **else** F.tag $\leftarrow tag'$;
        F.location $\leftarrow loc'$;
        F.time $\leftarrow t'$;
        F.switches $\leftarrow E$.switches $+ 1$;
        F.color $\leftarrow gray$;
        Set E as parent to F;
    end
    E.color $\leftarrow black$;
end
return Δ

Algorithm 4.1: Algorithm to compute reachtree for hybrid systems.

hybrid systems due to the presence of discrete transitions, the depth of the reachtree can be more than 1. The loop from lines 3 to 13 initiates the computation of reachtree. This loop computes the δ-cover of Θ and creates a node E_i for every element $B_\delta(x_i)$ in the cover. The attribute InitialSet is assigned as $B_\delta(x_i)$, tag is assigned must or may based on whether the $B_\delta(x_i)$ is completely contained within the initial set or not. The attributes time and switches are assigned 0 as no time elapsed and no discrete transitions are taken. Finally, the color attribute is assigned *gray* and the node is set as the child node to the root #.

The loop in lines 14 to 29 computes the rest of the reachtree. The loop selects an element E in the tree which is colored *gray* within the bounded time T and discrete transitions N. For such element, line 16 computes the invariant prefix of the reachtube starting from E.initialSet for $T - E$.time duration. It then computes the set of regions that can take a discrete transition by invoking the nextRegions routine in line 18. For each of the tagged regions returned by nextRegions, a new node is created, its attributes appropriately assigned, and the new node is added as the child node to the current node E in lines 19 to 26. Finally, the color of the current node is assigned *black*. This process is repeated until there are no nodes left that are colored gray with bound time T and discrete transitions N. The above procedure is guaranteed to terminate because of the bound on time and on the number of discrete transitions. Further, the depth of the reachtree is at most $N + 1$.

We abuse notation and say that $E \in \Delta$ to mean E is an element of the reachtree Δ. We also say that $R \in \Delta$ to mean that $\exists E \in \Delta$, such that $R \in E.\mathsf{RT}$. From the construction of reachtree, the following properties of reachtree can be established. We define the reachable set of Δ as the collection of regions in the reachtubes in Δ.

Definition 22 *Given a reachtree Δ, we denote the set of all regions in reachtubes of Δ as $ReachSet(\Delta)$. Formally, $ReachSet(\Delta) \triangleq \cup_{R \in \Delta} R$.*

Lemma 16 *Consider a reachtree Δ of hybrid system $\mathcal{A}$ for initial set Θ, time bound T, and bound on discrete transitions N, it follows that $Reach(\mathcal{A}, \Theta, T, N) \subseteq ReachSet(\Delta)$.*

Proof: Consider an execution σ of the hybrid system $\mathcal{A}$ for bounded time and bounded number of switches starting from initial set Θ. Let $\sigma = \tau_0 a_1 \tau_1 \ldots a_m \tau_m$. We now prove that $\forall t \leq \sigma.dur$, $\exists\ E_k \in \Delta$, $\exists R_j \in E_k.\mathsf{RT}$ such that $\sigma(t) \in R_j$. We prove this by induction on number of discrete transitions.

Base Case: Suppose σ does not have any discrete transition, i.e., $\sigma = \tau_0$. As $\tau_0(0) \in \Theta$, we have that $\exists E_i \in \Delta$, specifically, a child node of # such that $\tau_0(t) \in E_i.\mathsf{InitialSet}$ (line 1 computes the δ-cover of the initial set Θ). Notice that line 15 computes ψ, which is the reachtube for bounded time T starting from $E_i.\mathsf{InitialSet}$. Hence, we have that $\forall t \leq \tau_0.dur$, $\exists R_j \in \psi$, such that $\tau_0(t) \in R_j$. Moreover, since σ is a valid execution of $\mathcal{A}$, we have that $\forall t \leq \tau_0.dur$, $\tau_0(t) \in Inv_{loc_0}$, it follows that $\exists R'_j \in \phi$ such that $\tau_0(t) \in R'_j$ where ϕ is returned by $\mathsf{invariantPrefix}$ routine in line 16.

Induction Step: Suppose that for all executions σ' with $k-1$ discrete transitions ($k-1 \leq N-1$), the induction hypothesis is satisfied. We now have to establish that for all executions σ with k discrete transitions, the hypothesis is satisfied. Suppose that $\sigma' = \tau_0 a_1 \ldots a_{k-1}\tau_{k-1}$ and $\sigma = \tau_0 a_1 \ldots a_{k-1}\tau_{k-1}a_k\tau_k$, that is, σ is the execution obtained when σ' takes the discrete transition a_k and the trajectory τ_k.

From the induction hypothesis, we have that $\forall t \leq \sigma'.dur, \exists E \in \Delta, R \in E.\mathsf{RT}$ such that $\sigma(t) \in R$. Consider specifically the node that includes the last state of the execution σ'. Let $E' \in \Delta$ such that $\exists R' \in E'.\mathsf{RT}$ such that $\sigma'.\mathsf{fstate} \in R'$. As a_k is the discrete transition that is enabled after σ', it follows that $\sigma'.\mathsf{fstate} \in Guard_{a_k}$. Hence, we have that $R' \cap Guard_{a_k} \neq \emptyset$. Thus when $\mathsf{nextRegions}$ routine is invoked for E' in line 18, a new node E is created with initial set as R, the region reached after taking discrete transition a_k from R'. Therefore, we have that $\tau_k.\mathsf{fstate} \in E.\mathsf{InitialSet}$. Using the base case hypothesis, it follows that $\forall t \in [\sigma'.\mathsf{ltime}, \sigma.\mathsf{ltime}]$, $\tau_k(t) \in R_j$ for some $R_j \in E.\mathsf{RT}$. ■

Informally, Lemma 16 states that union of all the regions in reachtubes of a reachtree Δ contains the reachable set. Hence, to compute an overapproximation of the reachable set of states, one can compute a reachtree Δ and compute an overapproximation by considering the union of all the regions in Δ. In Chapter 6, we refer to this procedure as *Post*

Lemma 17 *Given a node $E \in \Delta$ such that $E.\mathsf{tag} = \mathsf{must}$, and region $R \in E.\mathsf{RT}$ that is tagged* must*, we have that $R \cap Reach(\mathcal{A}, \Theta, T, N) \neq \emptyset$.*

Proof: To prove this theorem, it is enough to prove that there exists an execution that reaches R if it meets the conditions stated. We prove this by induction on the depth of node E in reachtree Δ that contains R.

Base Case: Suppose that an element E is at depth 1 (as the root node # does not contain any reachtube) such that $E.\mathsf{tag} = \mathsf{must}$ and let $R \in E.\mathsf{RT}$ be a region that

is tagged must. It immediately follows from line 6 that $E.\mathsf{InitialSet} \subseteq \Theta$. As $E.\mathsf{RT}$ computes the invariant prefix of reachtube starting from the initial set, we have that all the trajectories τ that start from $E.\mathsf{InitialSet}$ for bounded time T are enclosed by the invariant prefix. Moreover, if region R is tagged must in the invariant prefix, it means that all the regions before it are completely contained within the invariant, hence $\exists \tau \in \mathcal{T}_{loc}$ such that $\tau(t) \in R$.

Induction Step: Suppose that the node E is at depth k such that $E.\mathsf{tag} = \mathsf{must}$ and let $R \in E.\mathsf{RT}$ be a region that is tagged must. Suppose that the parent node of E be E'. From the procedure nextRegions and from line 21 in Algorithm 4.1, it follows that 1) $E'.\mathsf{tag} = \mathsf{must}$ and 2) $\exists R' \in E'.\mathsf{RT}$ such that R' is tagged must and $\exists a \in A$ such that $R' \subseteq Guard_a$, $E.\mathsf{InitialSet} = Reset_a(R')$.

From induction hypothesis, we know that $\exists \sigma' \in execs(\mathcal{A})$ such that $\sigma'.\mathsf{fstate} \in R'$ (one can construct such an execution that ends in R' by considering only a prefix of the execution that reaches R'). As $R' \subseteq Guard_a$, it follows that the discrete transition a is enabled at $\sigma'.\mathsf{fstate}$. Applying the base case hypothesis, we have that $\exists \tau'$, a trajectory that starts from $E.\mathsf{InitialSet}$ and reaches R while satisfying the invariant of the location. Hence, the execution $\sigma = \sigma' a \tau'$ that is obtained by concatenating σ' with action a and trajectory τ' is a valid execution and reaches R. ■

Lemma 18 *Given a reachtree Δ, let $maxDiameter(\Delta) \triangleq max\ \{\ diameter(R) \mid R \in \Delta\}$. It follows that $maxDiameter(\Delta) \to 0$ as $\delta \to 0$ and $h \to 0$.*

Proof: Follows from Corollary 13 which states that all the regions in reachtubes can be made arbitrarily precise (from uniform continuity of discrepancy functions) and from Assumption 2 which establish the uniform continuity of reset maps. ■

4.3 Safety Verification For Hybrid Systems

The safety verification procedure is similar to that for continuous systems, where higher precision reachtrees are computed until the reachtree is proved to be safe or unsafe. We begin by first defining when a reachtree is safe.

Definition 23 *Given a reachtree Δ and an unsafe set U, the procedure* checkSafety *returns safe or unsafe or unknown if and only if*

- $\mathsf{checkSafety}(\Delta, U) = safe$ *if* $\forall E \in \Delta,\ \forall R_i \in E.\mathsf{RT},\ R_i \cap U = \emptyset$

- checkSafety(Δ, U) = *unsafe if* $\exists E \in \Delta$, *such that* E.tag = must *and* $\exists \langle R_i, tag_i, [t_{i-1}, t_i] \rangle \in E.\mathsf{RT}$, $R_i \subseteq U$ *and* tag_i = must.
- checkSafety(Δ, U) = *unknown otherwise.*

Similar to the safety of reachtree defined in Chapter 3, Definition 23 defines a reachtree to be safe only when all the reachtubes in all the nodes in the reachtree are safe. If there is an element E in reachtree that is tagged must and there is a region R in $E.\mathsf{RT}$ tagged must that is completely contained within unsafe set, then it is said to be unsafe. Else, one cannot infer whether the reachtree is either safe or unsafe, i.e., *unknown*. This happens because the overapproximation of reachtree is too coarse to either infer safety or its violation. To improve the order of overapproximation, we compute more precise reachtrees. The verification algorithm that computes more precise reachtrees for each iteration is presented in 4.2.

input : $\mathcal{A}$, Θ, U, T, N, loc_0
output: If $\mathcal{A}$ is safe or unsafe
1 Select $\delta > 0$, $h \in [0, T]$;
2 **while** *true* **do**
3 $\quad \Delta \leftarrow$ reachTree$(\Theta, T, N, \delta, h, loc_0)$;
4 $\quad$ **if** checkSafety$(\Delta, U) =$ *safe* **then** **return** *safe*;
5 $\quad$ **else if** checkSafety$(\Delta, U) =$ *unsafe* **then** **return** *unsafe*;
6 $\quad$ **else** $\delta \leftarrow \frac{\delta}{2}$; $h \leftarrow \frac{h}{2}$;
7 **end**

Algorithm 4.2: Dynamic analysis algorithm for safety verification of hybrid systems.

The correctness of the safety verification algorithm follows from the properties of reachtree given as lemmas 16 and 17. We first state the soundness property as follows:

Theorem 19 (Soundness) *Given a hybrid system* $\mathcal{A}$*, initial set* Θ*, unsafe set* U*, time bound* T*, and bound on discrete transitions* N*, if the algorithm 4.2 returns safe (or unsafe), then the system* $\mathcal{A}$ *is safe (or unsafe).*

Proof: Suppose that the Algorithm 4.2 returns safe. It follows that all the regions in the reachtree Δ (line 3) are safe. From Lemma 16 we have that all the reachable states from initial set Θ for bounded time and bounded number of discrete transitions are contained within the regions of reachtree. Hence, all executions with bounded time and bounded discrete transitions are safe.

Suppose that the algorithm 4.2 returns unsafe. It follows that $\exists E \in \Delta$ such that $E.\mathsf{tag} = \mathsf{must}$ and $\exists R \in E.\mathsf{RT}$ that is tagged must and $R \subseteq U$. From Lemma 17, it follows that there exists an execution $\sigma \in execs(\mathcal{A}, \Theta, T, N)$ such that $\sigma(t) \in R$. Hence the system is unsafe. ■

Lemma 20 *Given a reachtree Δ, let $\epsilon = maxDiameter(\Delta)$ as defined in Lemma 18. Given any region $R \in \Delta$, there is at least one execution of $\mathcal{A}_\epsilon$ from the initial set $B_\epsilon(\Theta)$ that reaches R within T time and N discrete transitions.*

Proof: Recall that $\mathcal{A}_\epsilon$ is the hybrid system that is obtained by bloating all the invariants and guards by ϵ. Similarly $B_\epsilon(\Theta)$ is obtained by bloating Θ by ϵ. Notice that all the regions that satisfy Lemma 17 are reachable by at least one execution of $\mathcal{A}$ from Θ and the proposition is automatically true.

We now consider the regions that are not covered by Lemma 17. For such regions, one of the following is true:

1. The region R is tagged may by the invariantPrefix routine or
2. The element E is tagged may by the nextRegions routine or
3. E is a child node to the root # and $E.\mathsf{InitialSet} \cap \Theta^c \neq \emptyset$ (line 6).

If the invariants, guards, and the initial set are bloated by ϵ, then the regions in the reachtree Δ would all be tagged as must regions during the reachtree computation from $\mathcal{A}_\epsilon$. Hence, from Lemma 17, it follows that all the regions in $\mathcal{R}$ are reachable for $\mathcal{A}_\epsilon$ with $B_\epsilon(\Theta)$ as initial set. ■

Theorem 21 (Relative Completeness) *Algorithm 4.2 will terminate and return correct answer when $\mathcal{A}$ is robustly safe or robustly unsafe.*

Proof: Suppose that the system is robustly safe. Recall from Definition 6 that if a system is robustly safe, then $\exists \epsilon > 0$ such that all executions of A_ϵ from initial set $B_\epsilon(\Theta)$ are safe from $B_\epsilon(U)$. From Theorem 19, we have that the algorithm never returns unsafe. Thus, the algorithm will continue computing more and more precise reachtrees until the safety of the system is proved. As the values of $\delta \to 0$ and $h \to 0$, the maximum diameter of regions in Δ also converges to 0. Let δ' and h' be the values such that the maximum diameter of regions in Δ is $\epsilon/2$. It follows from Lemma 20 that all such regions can be reached by executions starting from $B_{\epsilon/2}(\Theta)$ of $\mathcal{A}_{\epsilon/2}$. As

the system is robustly safe, i.e. do not contain any state from $B_\epsilon(U)$, the checkSafety procedure should return that Δ is safe because all the regions are at least $\epsilon/2$ distance from the unsafe set U.

Suppose that the system is robustly unsafe. From Definition 6. we have that $\exists \epsilon < 0$ such that all executions of A_ϵ from initial set $B_\epsilon(\Theta)$ are safe from $B_\epsilon(U)$. From Theorem 19, we have that the algorithm never returns safe. Thus, the algorithm will continue computing more and more precise reachtrees until the violation of safety is proved. Let δ' and h' be the parameters such that the diameter of all the regions computed in Δ are less than $\epsilon/2$.

Suppose that the unsafe execution σ of $\mathcal{A}_\epsilon$ starting from $B_\epsilon(\Theta)$ is given as $\sigma = \tau_0 a_1 \tau_1 \ldots a_m \tau_m$ where $\sigma.\mathsf{lstate} \in B_\epsilon(U)$. From Lemma 17 it follows that $\exists E_0, E_1, \ldots, E_m$ such that $E_i \in \Delta$ such that $\tau_i \in E_i.\mathsf{RT}$ and E_{i+1} is a child of E_i. As the diameter for each of the regions is less than $\epsilon/2$, it follows that each of the regions that contain trajectory τ_i in $E_i.\mathsf{RT}$ are completely contained within the invariant and are tagged must. Similarly, it follows that the regions where the discrete transition a_i is enabled are also completely contained within the guards and hence are also tagged must.

Consider the region $R \in \Delta$ such that $\sigma.\mathsf{lstate} \in R$. It follows that R is tagged must. Since $\sigma.\mathsf{lstate} \in B_\epsilon(U)$, we have that $R \subseteq B_{\epsilon/2}(U)$. Hence the checkSafety routine should return that Δ is unsafe. ■

In this section, we have presented a dynamic analysis technique for hybrid systems. We have seen the algorithm for computing the reachtree and presented an algorithm for safety verification using the reachtrees computed. In the following sections, we discuss the implementation of our technique in a tool and present some experimental results.

4.4 Compare-Execute-Check-Engine (C2E2): A Dynamic Analysis Tool For Hybrid Systems

The safety verification algorithm presented in Section 4.3 has been implemented in a tool called Compare-Execute-Check-Engine (C2E2). The tool takes as input the model of a hybrid system and a safety property. Further, discrepancy functions for each of the locations of the automaton are given by the user as model annotations. This is similar in spirit to the code contracts used in software. Having provided the

model, discrepancy function, and safety property, C2E2 verifies whether the safety property is satisfied or violated by the model. If violated, C2E2 also presents the counterexample (or its overapproximation) that violates the safety property. In this section we discuss the tool's architecture, its input format of the model description, format of the discrepancy function provided as model annotations, and user experience. We conclude this section by verifying standard benchmark examples with C2E2.

4.4.1 Architecture Of C2E2

The architecture of C2E2 is shown in Figure 4.3. The front end parses the input models, provides an editor for adding and verifying several safety properties, communicates the necessary information to the back-end for verification, and provides a plotter which helps visualize the reachable set. It is developed in Python and vastly extends the Hylink parser [123] for Stateflow models. The back end (verification engine that implements Algorithm 4.2) is developed in C++. The front end parses the input model file (.mdl or .hyxml) into an intermediate format and generates a C++ code for computing numerical simulations. The properties are obtained from the input file or from the user through the front end's GUI. The C++ code for generating numerical simulations is compiled using a validated simulation engine provided by Computer Assisted Proofs in Dynamic Groups (CAPD) library [2]. This compiled code and the property are read by the C2E2 verification back end which also uses the GLPK libraries. The verification result and the computed reachable set are read by the front end for display and visualization. This modular architecture allows us to extend the functionality of the tool to new types of models (such as Simulink models, systems with differential algebraic inequalities), different simulation engines (for example, Boost, VNODE-LP), and alternative checkers (such as Z3 and dReal [70]).

4.4.2 Models, Properties, And Annotations

Models & Properties: C2E2[1] takes as input annotated Stateflow models. Mathworks Simulink/Stateflow is a commonly used toolbox for modeling industrial scale CPS. It can express the continuous evolution of physical environment as solutions of ODEs (possibly nonlinear) and the software state can be described by different

[1] https://publish.illinois.edu/c2e2-tool/

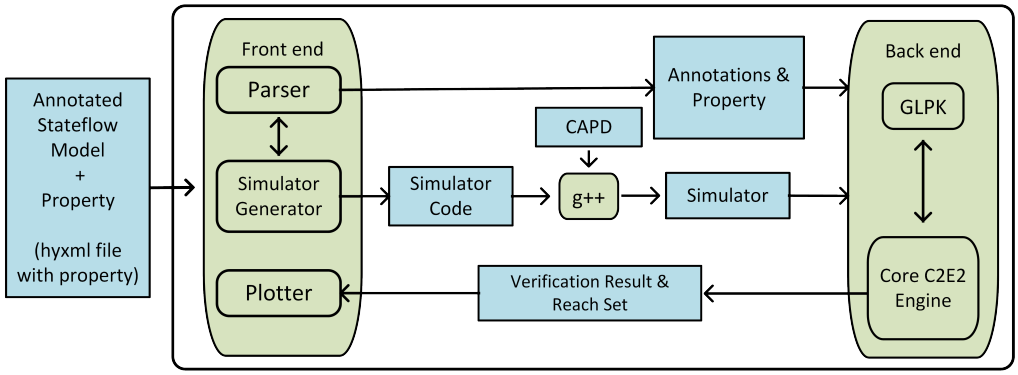


Figure 4.3: Architecture of C2E2

locations and with discrete transitions defined by guards and resets. The parser developed in Hylink [123] parses Stateflow models (given as files with .mdl extension). Alternatively, as Stateflow is a proprietary format, in order to facilitate academic researchers to develop models without having access to Simulink-Stateflow, we extended the Hylink parser to also accept as input an *eXtensible Markup Language* (XML) file with special tags for different elements of hybrid system. We restrict our attention to hybrid systems with polynomial predicates over the state variables as guards and invariants, and for the discrete transitions, the resets that are polynomial real-valued functions. The properties can be specified in the .hyxml model files or using the GUI. C2E2 can verify bounded time safety properties specified by a time bound, a polyhedral set of initial states and polyhedral set of unsafe states. Internally C2E2 does not bound the discrete number of transitions.

Annotations: The dynamic analysis technique presented crucially depends on discrepancy function. C2E2 requires that these discrepancy functions be given as model annotations. Specifically, C2E2 expects exponential discrepancy function for each location, given as tuple $\langle K, \gamma \rangle$ such that for a location l, given any two trajectories $\tau_1, \tau_2 \in \mathcal{T}_l$ and $t \in \tau_1.dur, t \in \tau_2.dur$, $||\tau_1(t) - \tau_2(t)|| \leq Ke^{\gamma t}||\tau_1(0) - \tau_2(0)||$. This pair $\langle K, \gamma \rangle$ is provided as comments in the Stateflow model and with special tags in hyxml file. We have seen in Sections 3.3.1 and 3.3.2 that ODEs that are Lipschitz continuous or ODEs that admit a contraction metric give us exponential discrepancy functions. In Example 8 we show how to obtain an exponential discrepancy function

for systems that have incremental Lyapunov function described in Section 3.3.3.
Example 8: Consider the differential equation

$$\dot{x} = 1 + x^2y - 2.5x;$$
$$\dot{y} = 1.5x - x^2y - y$$

By analyzing the auxiliary system (x_1, y_1) and (x_2, y_2), using the incremental stability [16], we have that

$$\frac{d}{dt}[(x_1 - x_2)^2 + 2(x_1 - x_2)(y_1 - y_2) + (y_1 - y_2)^2] = -2(x_1 - x_2 + y_1 - y_2)^2 < 0.$$

Therefore, for any two trajectories τ_1 and τ_2 starting from (x_1, y_1) and (x_2, y_2) respectively it follows that $(\tau_1(t).x - \tau_2(t).x)^2 + 2(\tau_1(t).x - \tau_2(t).x)(\tau_1(t).y - \tau_2(t).y) + (\tau_1(t).y - \tau_2(t).y)^2 \leq (x_1 - x_2)^2 + 2(x_1 - x_2)(y_1 - y_2) + (y_1 - y_2)^2$. The function $||\tau_1(t) - \tau_2(t)|| \leq 2||\tau_1(0) - \tau_2(0)||e^{0\times t}$ is an annotation and is specified as $K = 2, \gamma = 0$. □

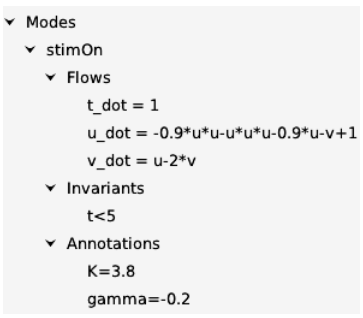
Remark 5 Example 8 illustrates the procedure to get an exponential discrepancy function for a particular incrementally stable system. More generally, for systems that are incrementally stable (as defined in Section 3.3.3) with incremental Lyapunov function V, we have that $\langle K, \gamma \rangle$ is an exponential discrepancy function where

$$K = sup_{x_2,x_1}\{\frac{V(x_1, x_2)}{||x_1 - x_2||}\} \text{ and } \gamma = 0.$$

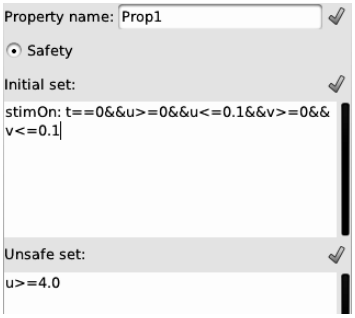
□

4.4.3 User Experience

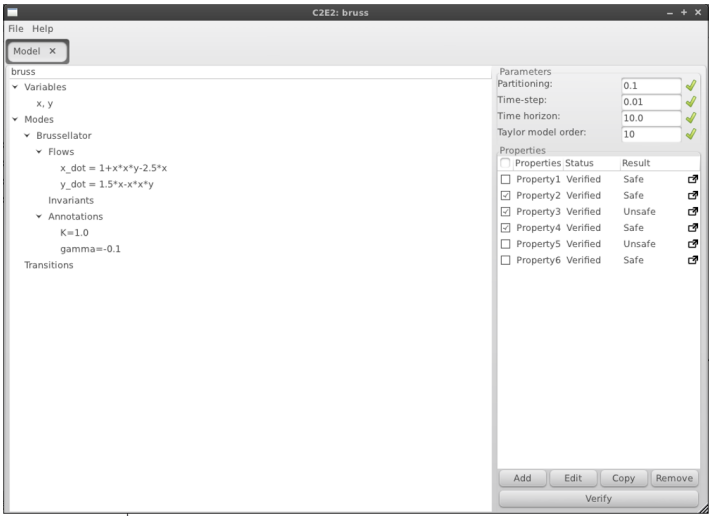
In this section, we discuss the C2E2 interface for verifying safety property and visualization of reachable set. Upon launching C2E2 and loading the model file, the users are greeted with a front end that enables them to explore the hybrid system in a tree-format as shown in Figure 4.4(a). The front end also facilitates the users to provide the safety property using GUI. The interface for specifying and verifying safety properties is shown in Figure 4.4(c). The interface has two main components. To the left is the model explorer which displays the model in a tree format and to the



(a) Model Explorer.



(b) Property Editor.

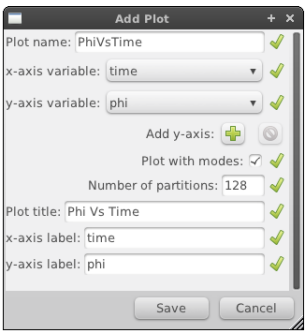


(c) GUI Front End.

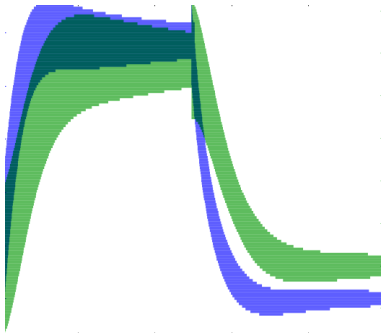
Figure 4.4: Figure showing a snippet of cardiac cell model in (a), property dialog for specifying properties in (b), the GUI for C2E2 that enables to verify properties and observe the models in (c).

right is the interface that allows the users to add, edit, copy, delete or verify several safety properties. The safety property for the hybrid system is defined by the initial set of states and an unsafe set of states. The GUI for specifying a new property is shown in Figure 4.4(b). As each property is edited, the *smart parser* provides real-time feedback about syntax errors and unbounded initial sets (Figure 4.4(b)). Once properties are edited the verifier can be launched.

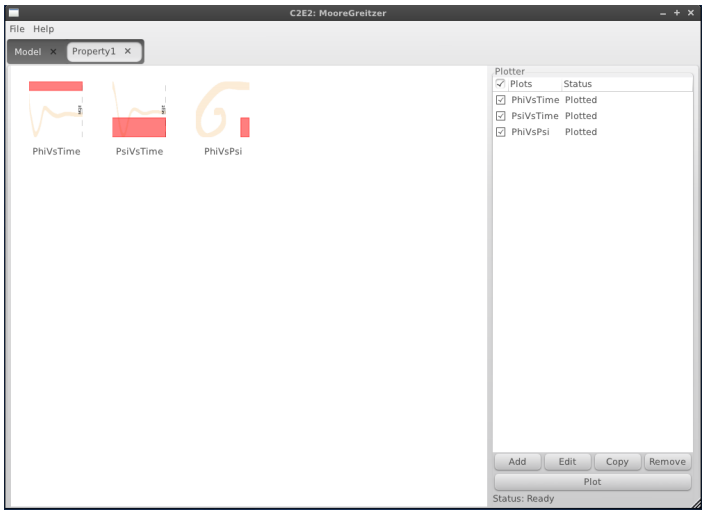
Visual representation of reachable states and locations can aid debugging process. To this end, we have integrated a visualizer into C2E2 for plotting the projections of the reachable states. Once a property has been verified, the user can plot the valuations of variables against time or valuations of pairs of variables (phase plots). Similar to the interface for safety verification, the visualizer also has 2 main components. On



(a) GUI for creating new plots.



(b) Plot of reachable set.



(c) GUI for plotting.

Figure 4.5: Figure showing the interface for adding a new plot in (a), reachable set of the cardiac cell model in (b), and the visualizer interface in (c).

the left is the interface that provides thumbnails of the phase or time plots (that are already plotted) and to the right is the interface that allows users to add, modify, delete, or plot additional time or phase plots. The screen shot of the interface is shown in Figure 4.5(c). The interface for adding a new plot is shown in Figure 4.5(a). The unsafe set is projected on the set of plotting variables. The property parser and visualizer uses the Parma Polyhedra Library[2] and matplotlib[3].

[2] http://bugseng.com/products/ppl/
[3] http://matplotlib.org/

4.4.4 Stateflow Model Semantics

It is often the case that using a typical formal verification tool requires a steep learning curve to understand the input format for models, the specification, and the interface. Designing C2E2 with a GUI was a conscious choice to alleviate part of the problem. Additionally, C2E2 accepts Stateflow models as input, which brings it closer to CPS system designers. However, these Stateflow models do not have a prescribed semantics. The simulations traces that MATLAB generates is considered as the default semantics of the model. There are two aspects of these models that should be discussed. First is that although the mathematical formalism of HIOA [127] allows for nondeterminism, such nondeterminism is not allowed in Stateflow models. Second, all the discrete transitions are interpreted as "urgent'', i.e. a discrete transition is taken by the system *as soon as* it is enabled. For example, if the guard for a discrete transition is given as $x \geq 5$, it is enabled whenever the value of variable x is greater than or equal to 5. However, in Stateflow, the transition is taken as soon as it is enabled i.e. $x = 5$. Under such interpretation, the guard sets are only allowed to be hyperplanes or hypersurfaces given by solutions of polynomials. For such models, the safety verification algorithm need not terminate. Therefore, we consider an ϵ perturbation of the guards. That is, the guard $x >= 5$ is interpreted as $x >= 5 - \epsilon \wedge x <= 5 + \epsilon$.

4.5 Experiments

Simulation based verification approach for annotated models has been demonstrated to outperform other verification tool such as Flow* and Ariadne in Section 3.6. In this chapter, we present the verification results for some of the nonlinear and linear hybrid systems benchmarks in Table 4.1. The annotations for each of these benchmarks have been obtained by techniques described in Sections 3.2 and 3.4. All the experiments have been performed on Ubuntu 11.10 systems with Intel i-7 Quad core processor with 8GB ram.

Algorithm 4.2 iteratively computes more precise reachtrees and checks if the reachtree is safe. However, the implementation optimizes this procedure by performing safety verification while reachtree is being computed. Therefore, if the system turns out to be unsafe, then the reachtree computation terminates. This is reflected in Table 4.1 where the systems that violate safety take less time for verification than their safe counterparts. On standard examples C2E2 can successfully verify these sys-

Benchmark	Vars.	Num. Loc.	TH	VT (sec)	Result
Cardiac Cell	3	2	15	17.74	safe
Cardiac Cell	3	2	15	1.91	unsafe
Nonlinear Navigation	4	4	2.0	124.10	safe
Nonlinear Navigation	4	4	2.0	4.94	unsafe
Inverted Pendulum	2	1	10	1.27	safe
Inverted Pendulum	2	1	10	1.32	unsafe
Navigation Benchmark	4	4	2.0	94.35	safe
Navigation Benchmark	4	4	2.0	4.74	unsafe

Table 4.1: Experimental Results for benchmark examples. Vars: Number of Variables, Num. Loc. : Number of discrete locations in hybrid automata, TH: Time Horizon for Verification, VT (sec) : Verification time for C2E2 in seconds, Result: Verification result of C2E2.

tems within an order of minutes and also handle nonlinear differential equations with trigonometric functions of inverted pendulum.

4.6 Conclusions

In this chapter we have presented the algorithm for computing reachTree for hybrid systems that overapproximates all the executions with bounded time and discrete transitions. For doing this, we have introduced two new routines called invariantPrefix and nextRegions. We then presented a safety verification algorithm for hybrid systems and proved its theoretical guarantees namely soundness and relative completeness. We then presented a tool C2E2 that implements the verification algorithm presented. We inspected several aspects of the tool C2E2 such as its architecture, its GUI interface, and visualization features. We conclude this chapter with verification results of standard benchmark examples.

Chapter 5

Verification Case Studies: Parallel Landing Protocol And Powertrain Control System

In this chapter we present two case studies for verification. First, is an alerting mechanism for parallel landing protocol and second, is a powertrain control system in automobiles. The parallel landing protocol considered in this case study relies on an alerting mechanism called Adjacent Landing Alerting System (ALAS) [96] for ensuring that a safe separation is maintained between the landing aircraft. In this chapter, we formulate the requirements of the alerting mechanism rigorously as temporal precedence properties and present a dynamic analysis technique for verifying such properties. We present the experimental results of the verification under different scenarios and runway geometries. The second case study is verification of powertrain control systems that has been proposed as a challenge problem by Toyota [94]. We verify important temporal properties for powertrain control system using the dynamic analysis technique for safety verification presented in Chapter 4. These two case studies demonstrate that dynamic analysis is a powerful technique for verifying realistic CPS.

5.1 Introduction

We now discuss the motivation behind the verification of parallel landing protocol and powertrain control system, the challenges encountered during verification, and outline the verification technique.

Parallel Landing Protocol: Steady increase in the air-traffic is projected to result in crowded airspaces and lower throughputs of airports. To improve the throughput of airports, Simplified Aircraft-based Paired Approach (SAPA) concept was proposed in [96] for enabling parallel landing of aircraft. SAPA relies on an alerting mechanism called Adjacent Landing Alerting System (ALAS) for ensuring that the safe separation between landing aircraft is always maintained. ALAS is designed to issue an alert if

it predicts that the safe separation between the landing aircraft would be violated in the near future.

It is hence pivotal to verify the correctness of ALAS protocol. ALAS should not only issue an alert before the aircraft violate safe separation, but it should issue this alert at least b time units before such violation happens. This time b is used as a buffer time for enabling the fail safe maneuvers in case of violation of safe separation. Formally, this property is specified as a *temporal precedence property* denoted as $P_1 \prec_b P_2$. A system is said to satisfy $P_1 \prec_b P_2$ if and only if all behaviors satisfy the predicate P_1 at least b time units before satisfying the predicate P_2. For ALAS, the requirement can be written in the form of *Alert* $\prec_b$ *Unsafe* where *Alert* predicate is satisfied by a state if ALAS issues an alert in that state and *Unsafe* predicate is satisfied when the safe separation between aircraft is violated.

Another challenging aspect of the ALAS is that, the predicate that issues the alert, falls under the category of *guarantee predicates*. Such predicates are of the form $\exists t, f_p(x,t)$. Here f_p is called the lookahead function and predicts the future behavior of the system. Although quantifier elimination can be used to handle a restricted form of such lookahead functions (such as polynomials), it is often an expensive operation. In the case of ALAS, this lookahead function is defined implicitly as a solution of a linear ordinary differential equation and hence involves trigonometric functions, for which, quantifier elimination is an expensive operation.

In this chapter, we verify the correctness of ALAS. We achieve this by presenting a new dynamic analysis algorithm for verifying temporal precedence properties. The algorithm relies on computing reachtrees for hybrid systems that was presented in Chapter 4. Further, we present a dynamic analysis technique for handling the guarantee predicate that is given as solution to ordinary differential equations. We prove the correctness of the algorithm, establish its theoretical properties and present verification results.

Powertrain Control System: As the targets for fuel efficiency, emissions, and drivability become more demanding, automakers are becoming interested in pushing the design automation and verification technologies for automotive control systems. The benchmark suite of powertrain control systems published in [95, 94] was posed as challenge problem for academic research. These benchmarks consists of a sequence of Simulink$^{\text{TM}}$/Stateflow$^{\text{TM}}$ models of the engine with increasing levels of sophistication and fidelity.

The complex of these models involves delay differential equations, periodically updated sensors, and look up tables. Such models are out of the scope of current verification tools. This complex model has been simplified to have ordinary differential equations, one with periodically updated sensors, and the other with polynomial ordinary differential equations. The challenge problems posed in [95, 94] includes verifying the conformance of one model with another. Important requirements of the system were specified in Signal Temporal Logic [122], a logical framework for specifying temporal properties of signals.

In this chapter, we present the results of verifying properties of powertrain control system model with polynomial dynamics. The key controlled quantity is the air to fuel ratio which influences the emissions, the fuel efficiency, and the torque generated. The properties verified in this case study are divided mainly into two categories. First, are the global properties that ensure that the air to fuel ratio always lies within the operating range and second, are the performance properties that enforce that the air to fuel ratio is within a bounded range close to the reference air to fuel ratio. In this chapter we present verification results of these properties using the reachTree computation presented in Chapter 4.

This chapter is organized as follows: we first present the case study of verifying temporal precedence protocol for parallel landing protocol in Section 5.2. Here, we present the verification algorithm (Section 5.2.2), prove its theoretical guarantees (Section 5.2.2), and experimentally evaluate the proposed approach under different scenarios (Section 5.3.2). We then present the case study of powertrain control systems in Section 5.4. We present the model considered for verification (Section 5.4.1), outline the verification algorithm (Section 5.4.2), and present the verification results of different properties (Section 5.5).

5.2 Temporal Precedence Checking For Parallel Landing Protocol

We now present the verification case study of Adjacent Landing Alerting System (ALAS). The Simplified Aircraft-based Paired Approach (SAPA) is an advanced operational concept proposed by the US Federal Aviation Administration (FAA) [96]. The SAPA concept supports dependent, low-visibility parallel approach operations to runways with lateral spacing closer than 2500 ft. A Monte-Carlo study conducted by

NASA has concluded that the basic SAPA concept is technically and operationally feasible [96]. SAPA relies on an alerting mechanism ALAS [137] to avoid aircraft blunders, i.e., airspace situations where an aircraft threatens to cross the path of another landing aircraft. The alerting mechanism designed to handle such scenarios should issue an alert preemptively before safe separation is violated. This requirement can be formally expressed as a *temporal precedence property* that we will introduce later in this section.

In the case of ALAS, this alerting system is designed as a *guarantee predicate* that is, the predicate is satisfied if the predicted behavior of the aircraft leads to the violation of safe separation. Further, the look ahead function in the guarantee predicate is given as implicit solution of differential equations. In this section we present a dynamic analysis technique for verifying temporal precedence property and handle guarantee predicates that is given implicitly as solution of differential equation. We conclude this section by presenting the results of verifying the ALAS protocol for different aircraft speeds and runway configurations.

5.2.1 Temporal Precedence And Guarantee Predicates

We assume that the motion of aircraft performing parallel landing is given as a hybrid system. The continuous motion of the aircraft is modeled as trajectories in each location of hybrid system. As the motion of each aircraft is dependent on the control input it receives from software, the discrete transitions in the hybrid system models the discrete transitions in software. Suppose that a hybrid system $\mathcal{A}$ models the behavior of aircraft in a parallel landing scenario.

An important property that should be satisfied by the ALAS system during parallel landing is that the alert should be issued at least a given time units b before the aircraft violate safe separation. This is modeled as temporal precedence property given formally as follows.

Definition 24 *Given a hybrid system $\mathcal{A}$, initial set Θ, predicates P_1 and P_2, and parameter $b > 0$, the property $P_1 \prec_b P_2$ is said to be satisfied by $\mathcal{A}$ from the set Θ if and only if $\forall \sigma \in execs(\mathcal{A}, \Theta)$, $\forall t_2 > 0$ with $\sigma(t_2) \in P_2$, $\exists t_1 > 0$ such that $\sigma(t_1) \in P_1$ and $t_1 < t_2 - b$. We say that $P_1 \prec P_2$ is satisfied if $\exists b > 0$ such that $P_1 \prec_b P_2$ is satisfied. The property $P_1 \prec_b P_2$ is said to be violated if and only if $\exists \sigma \in execs(\mathcal{A}, \Theta)$, such that, $\exists t_2 > 0$, such that $\sigma(t_2) \in P_2$ and $\forall t_1 > 0, t_1 \leq t_2 - b$ with $\sigma(t_1) \notin P_1$.*

Definition 25 (Robust satisfaction or violation) *Given a hybrid system $\mathcal{A}$, initial set Θ, predicates P_1 and P_2, and parameter $b > 0$, the property property $P_1 \prec_b P_2$ is said to be* robustly satisfied *if $\exists \nu > 0, \epsilon > 0$ such that all executions of $\mathcal{A}_\epsilon$ starting from $B_\epsilon(\Theta)$ satisfy the property $B_{-\epsilon}(P_1) \prec_{b+\nu} B_\epsilon(P_2)$. The property $P_1 \prec_b P_2$ is said to be* robustly violated *if $\exists \nu > 0, \epsilon > 0$ such that an executions of $\mathcal{A}_{-\epsilon}$ starting from $B_{-\epsilon}(\Theta)$ violates the property $B_\epsilon(P_1) \prec_{b-\nu} B_{-\epsilon}(P_2)$. Here $B_\epsilon(P_i)$ ($B_{-\epsilon}(P_i)$) refers to the bloating of the predicate P_i with the factor ϵ ($-\epsilon$) respectively.*

Definition 24 states that whenever the predicate P_2 is satisfied by the execution σ (say at time instance t_2), the predicate P_1 was satisfied at an earlier time instance (say t_1) at least b time units before. If the above condition does not happen, then it is said to violate the property. Notice that a system that satisfies the property $B_{-\epsilon}(P_1) \prec_{b+\nu} B_\epsilon(P_2)$ also satisfies $P_1 \prec_b P_2$ and hence $B_{-\epsilon}(P_1) \prec_{b+\nu} B_\epsilon(P_2)$ is a stronger requirement than $P_1 \prec_b P_2$. Similarly $B_\epsilon(P_1) \prec_{b-\nu} B_{-\epsilon}(P_2)$ is a weaker requirement than $P_1 \prec_b P_2$.

Suppose that the predicate to issue the alert in ALAS is given as *Alert* and the violation of safe separation is given by *Unsafe*, then the formal requirement of ALAS is given as *Alert* $\prec_b$ *Unsafe* where b is the desired buffer time. In this case study, we consider only executions of bounded time and bounded number of switches in Definition 24 as specification for temporal precedence. The next building block that is required in verification of ALAS is the notion of guarantee predicate. We recall the definition of guarantee predicate as follows:

Definition 26 *A predicate P is said to be a guarantee predicate with lookahead function $f_P : \mathbb{R}^n \times \mathbb{R} \to \mathbb{R}$ if $P \triangleq \exists t, f_P(x,t) < 0$. That is, a state $x \in P$ if and only if $\exists t, f_P(x,t) < 0$.*

A special case of this guarantee predicates is when this lookahead function is defined as solution of ODE. We consider a solution of an ODE as a continuous system $\mathcal{C}' \triangleq \langle \mathcal{V}, \mathcal{T}' \rangle$. Guarantee predicates defined with respect to such continuous systems are defined as follows:

Definition 27 *Given a continuous system $\mathcal{C}' \triangleq \langle V, \mathcal{T}' \rangle$ defined as the solution of an ODE $\dot{x} = g(x)$, a function $w_P : \mathbb{R}^n \to \{\top, \bot\}$, the guarantee predicate P defined by the ODE is given as $x \in P$ if and only if $\exists \tau' \in \mathcal{T}'$ with $\tau(0) = x$, $\exists t > 0$ such that $w_P(\tau(t)) = \top$.*

In this case study, we only consider guarantee predicates with bounded lookahead time, i.e., the trajectories in Definition 27 are restricted to bounded time (more on this in Section 5.2.3). Before presenting the verification algorithm for temporal precedence, we will introduce the definition of *may*, *must* and *not* regions for a reachtree. Recall that a reachtree (Section 4.2.1) is a tree with each element storing the invariant prefix of reachtubes that overapproximates the reachable set. Formally, the notion of may, must, and not regions is given as follows:

Definition 28 *Given a reachtree Δ, $\langle R, \mathsf{tag}, I\rangle \in \Delta$, and a predicate P, we have that*

$\langle R, \mathsf{tag}, I\rangle \in mustInt(P, \Delta)$ *if and only if* $R \subseteq P$.

$\langle R, \mathsf{tag}, I\rangle \in notInt(P, \Delta)$ *if and only if* $R \subseteq P^c$.

$\langle R, \mathsf{tag}, I\rangle \in mayInt(P, \Delta)$ *otherwise.*

We drop Δ as the second argument when it is clear from the context.

5.2.2 Dynamic Analysis For Temporal Precedence Verification

In this section, we present the dynamic analysis algorithm for verifying temporal precedence properties. This algorithm relies on computing reachtrees for hybrid systems that were introduced in Section 4.2. Similar to the verification of safety properties, this algorithm iteratively computes reachtrees with increasing precision. For each reachtree computed, it checks whether the reachtree satisfies the temporal precedence property and terminates if either the temporal precedence is satisfied, or, a violating execution is discovered. We now present the building blocks for the verification procedure. The first building block is a *reached before* relation among the tagged regions in the reachtree.

Definition 29 *Given a reachtree Δ and $\langle R_1, \mathsf{tag}_1, I_1\rangle, \langle R_2, \mathsf{tag}_2, I_2\rangle \in \Delta$, we say that $\langle R_1, \mathsf{tag}_1, I_1\rangle$ is* reached before *$\langle R_2, \mathsf{tag}_2, I_2\rangle$ if and only if one of two conditions is satisfied:*

1. *$\exists E \in \Delta$, such that $\langle R_1, \mathsf{tag}_1, I_1\rangle \in E.\mathsf{RT}$ and $\langle R_2, \mathsf{tag}_2, I_2\rangle \in E.\mathsf{RT}$ and $I_1 \leq I_2$, or*

2. *$\exists E_1, E_2 \in \Delta$, such that $\langle R_1, \mathsf{tag}_1, I_1\rangle \in E_1.\mathsf{RT}$ and $\langle R_2, \mathsf{tag}_2, I_2\rangle \in E_2.\mathsf{RT}$, and $\exists a$, a discrete transition such that E_2 is a child node of E_1 obtained after taking the discrete transition a from the region R_1, i.e., $E_2.\mathsf{initialSet} = Reset_a(R_1)$.*

Informally, $\langle R_1, \mathsf{tag}_1, I_1\rangle$ is reached before $\langle R_2, \mathsf{tag}_2, I_2\rangle$ if either they appear in the same reachtube and R_1 appears before R_2 in the reachtube, or, R_2 is part of a reachtube that is obtained after taking a discrete transition from R_1. We denote the reached before relation as $\langle R_1, \mathsf{tag}_1, I_1\rangle \curvearrowright \langle R_2, \mathsf{tag}_2, I_2\rangle$. The following corollary can be proved using the proof of Lemma 16.

Corollary 22 *Given a reachtree Δ of $\mathcal{A}$ from initial set Θ, time bound T, and bound on discrete transitions N, and given $\langle R_1, \mathsf{tag}_1, I_1\rangle, \langle R_2, \mathsf{tag}_2, I_2\rangle \in \Delta$ such that $\langle R_1, \mathsf{tag}_1, I_1\rangle \curvearrowright \langle R_2, \mathsf{tag}_2, I_2\rangle$, it follows that, if $\exists \sigma \in execs(\mathcal{A}, \Theta, T, N), \exists t \in I_2$ such that $\sigma(t) \in R_2$, then $\exists t' \in I_1$, $t' \leq t$ such that $\sigma(t') \in R_1$.*

Informally, Corollary 22 says that for any execution that reaches the region R_2, the execution also reaches R_1 before it reaches R_2 if they are related by reached before relation. This holds because either R_2 comes after R_1 in the same reachtube or R_2 is in a reachtube obtained after taking a discrete transition from region R_1. Notice that the corollary holds when $\langle R_1, \mathsf{tag}_1, I_1\rangle$ and $\langle R_2, \mathsf{tag}_2, I_2\rangle$ are also related by the transitive closure of reached before relation, i.e., $\langle R_1, \mathsf{tag}_1, I_1\rangle \curvearrowright^* \langle R_2, \mathsf{tag}_2, I_2\rangle$.

Definition 30 *Given a reachtree Δ, predicates P_1 and P_2, and $b > 0$, we have*

- *Δ satisfies the property $P_1 \prec_b P_2$, if and only if for all $\langle R, \mathsf{tag}, I\rangle \in mustInt(P_2) \cup mayInt(P_2)$, $\exists \langle R', \mathsf{tag}', I'\rangle \in mustInt(P_1)$, such that $\langle R', \mathsf{tag}', I'\rangle \curvearrowright^* \langle R, \mathsf{tag}, I\rangle$ and $I' < I - b$.*
- *Δ violates the property $P_1 \prec_b P_2$, if and only if, $\exists \langle R', \mathsf{tag}', I'\rangle \in mustInt(P_2)$, such that, $\forall \langle R, \mathsf{tag}, I\rangle \in \Delta$, such that $\langle R, \mathsf{tag}, I\rangle \curvearrowright^* \langle R', \mathsf{tag}', I'\rangle$, and $I' \leq I - b$, we have that $\langle R, \mathsf{tag}, I\rangle \in notInt(P_1)$.*
- *Δ neither satisfies nor violates $P_1 \prec_b P_2$ otherwise.*

Definition 30 defines whether a given reachtree Δ satisfies a given temporal precedence property $P_1 \prec_b P_2$ or not. Informally, it states that the property is satisfied if for every must or may region of P_2 in a given reachtree, there exists a must region of P_1 that is *reached before* at least b time units. Further, the reachtree is said to violate the property if there exists a must region of P_2, say R_2, such that regions that are *reached before* b time units from R_2 belong to $notInt(P_1)$. Otherwise, it cannot be inferred whether the temporal precedence property is satisfied or violated. The

procedure check$(\Delta, P_1 \prec_b P_2)$ returns $\top$ if Δ satisfies the property, $\bot$ if Δ violates the property and *unknown* otherwise.

We now present the verification algorithm for temporal precedence properties that uses reachtree computation.

input : Θ, T, N, $\mathcal{A}$, loc_0, $P_1 \prec_b P_2$
output: If $\mathcal{A}$ satisfies the property or not
1 Select $\delta > 0$, $h \in [0, T]$;
2 **while** *true* **do**
3 $\quad \Delta \leftarrow$ reachTree$(\Theta, \delta, h, T, N, loc_0)$;
4 $\quad$ **if** check$(\Delta, P_1 \prec_b P_2) = \top$ **then** **return** *satisfies*;
5 $\quad$ **else if** check$(\Delta, P_1 \prec_b P_2) = \bot$ **then** **return** *violates*;
6 $\quad$ **else** $\delta \leftarrow \frac{\delta}{2}$; $h \leftarrow \frac{h}{2}$;
7 **end**

Algorithm 5.1: Dynamic analysis algorithm checkTempPrec for verifying temporal precedence property of hybrid systems.

Algorithm 5.1 is similar to the safety verification algorithm presented in Algorithm 4.2. We now prove the theoretical guarantees provided by Algorithm 5.1.

Theorem 23 (Soundness) *Procedure* checkTempPrec *defined in Algorithm 5.1 is sound, i.e., if it returns that the system satisfies the property, then the property is indeed satisfied. If it returns that the property is violated, then the property is indeed violated by the system.*

Proof: Proof follows from Definition 24 and Corollary 22. ∎

Theorem 24 (Relative Completeness) *If the system $\mathcal{A}$ robustly satisfies the property $P_1 \prec_b P_2$ or if $\mathcal{A}$ robustly violates $P_1 \prec_b P_2$ then procedure* checkTempPrec *defined in Algorithm 5.1 terminates with the right answer.*

Proof: This proof is similar to the proof of Theorem 21 for proving relative completeness of the verification with respect to safety properties.

Property is robustly satisfied: Suppose that the system robustly satisfies the property. From Definition 25, let $\epsilon > 0, \nu > 0$ such that all executions of A_ϵ from $B_\epsilon(\Theta)$ satisfy $B_{-\epsilon}(P_1) \prec_{b+\nu} B_\epsilon(P_2)$. As the algorithm is sound, it will never return that the property is violated and it iterates while reducing the value of δ and h. It follows

from Lemma 18 that $\exists \delta' > 0, h' > 0$, such that the diameter of all the regions in Δ is less than $\epsilon/4$ and the intervals for all the regions is less than $\nu/4$.

Now consider $\langle R_2, \mathsf{tag}_2, I_2 \rangle \in mayInt(P_2) \cup mustInt(P_2)$. Consider an execution σ that reaches R_2, i.e., $\exists t \in I_2$, such that $\sigma(t) \in R_2$. Suppose that $\sigma(t) \in P_2$. Since the system robustly satisfies the property, from Definition 25, it follows that $\exists t' < t - b - \nu$ such that $\sigma(t') \in B_{-\epsilon}(P_1)$. Consider the region $\langle R_1, \mathsf{tag}_1, I_1 \rangle$ such that $\sigma(t') \in R_1$. As $diameter(R_1) < \epsilon/4$, we have that $\langle R_1, \mathsf{tag}_1, I_1 \rangle \in mustInt(P_1)$, and since $t' < t - b - \nu$, it follows that $\langle R_1, \mathsf{tag}_1, I_1 \rangle \curvearrowright^* \langle R_2, \mathsf{tag}_2, I_2 \rangle$ and $I_1 < I_2 - b$. Hence for every region in $mayInt(P_2) \cup mustInt(P_2)$, there is a region in $mustInt(P_1)$ at least b time units before. Therefore the algorithm will terminate and return that the system satisfies the property.

Property is robustly violated: Suppose that the system robustly violates the property. From Definition 25, let $\epsilon > 0, \nu > 0$ such that an execution of $A_{-\epsilon}$ from $B_{-\epsilon}(\Theta)$ violates $B_{\epsilon}(P_1) \prec_{b-\nu} B_{-\epsilon}(P_2)$. As the algorithm is sound, it will never return that the property is satisfied and it iterates while reducing the value of δ and h. It follows from Lemma 18 that $\exists \delta' > 0, h' > 0$, such that the diameter of all the regions in Δ is less than $\epsilon/2$ and the intervals for all the regions is less than $\nu/4$.

Now consider an execution σ in $execs(\mathcal{A}_{-\epsilon})$ that violates the property $B_{\epsilon}(P_1) \prec_{b-\nu} B_{-\epsilon}(P_2)$. Let t_2 be the time instance such that $\sigma(t_2) \in B_{-\epsilon}(P_2)$ and $\forall t_1 < t_2 - b + \nu$, we have that $\sigma(t_1) \notin B_{\epsilon}(P_1)$. Now consider $\langle R_2, \mathsf{tag}_2, I_2 \rangle \in \Delta$ such that $\sigma(t_2) \in R_2$. Since $diameter(R_2) < \epsilon/4$, we have that $\langle R_2, \mathsf{tag}_2, I_2 \rangle \in mustInt(P_2)$. Consider $\langle R_1, tag_1, I_1 \rangle \in \Delta$ such that $\langle R_1, tag_1, I_1 \rangle \curvearrowright * \langle R_2, tag_2, I_2 \rangle$ with $I_1 \leq I_2 - b - \nu/4$. From the falsifying execution, it follows that $\langle R_1, \mathsf{tag}_1, I_1 \rangle \in notInt(P_1)$. Hence, the procedure will return that the system violates the property. ■

5.2.3 Handling Guarantee Predicates

The final piece in the temporal precedence verification is to check whether a region is a must or a may region for a guarantee predicate given as implicitly as solutions of ODE. We consider a bounded time version of the guarantee predicate, i.e., given bounded time T_b, the predicate P is defined as $x \in P$ if and only if $\exists \tau' \in \mathcal{T}'$ such that $\tau'(0) = x$ and $\exists t > 0, t < T_b, w_p(\tau'(t)) = \top$. Given a region R and predicate P, for checking whether $R \subseteq P$, $R \subseteq P^c$, or $R \cap P \neq \emptyset \wedge R \cap P^c \neq \emptyset$, we compute the reachtree for continuous system C' starting from the set R and check that all the reachtubes in the nodes of reachtree eventually enter the region $w_p(x) = \top$. This

procedure checkGuarantee is given in Algorithm 5.2.

input : R, T_b, $\mathcal{C}'$, w_p
output: Whether $R \subseteq P$, $R \subseteq P^c$, or $R \cap P \neq \emptyset \wedge R \cap P^c \neq \emptyset$
1 . Select $\delta > 0$, $h \in [0, T_b]$;
2 **while** *true* **do**
3 $\quad \Delta \leftarrow \mathsf{reachTree}(\Theta, T_b, \delta, h)$;
4 $\quad$ **if** $\forall E_i \in \Delta, \exists R_j \in E_i.\mathsf{RT}, w_p(R_j) = \top$ **then return** $R \subseteq P$;
5 $\quad$ **else if** $\forall E_i \in \Delta, \forall R_j \in E_i.\mathsf{RT}, w_p(R_j) = \bot$ **then return** $R \subseteq P^c$;
6 $\quad$ **else if** $\exists E_i, E_j \in \Delta$, $E_i.\mathsf{tag} = E_j.\mathsf{tag} = \mathsf{must}$, $\exists R_k \in E_i.\mathsf{RT}$, $\exists R_l \in E_j.RT$, $w_p(R_k) = \top$, $w_p(R_l) = \bot$ **then return** $R \cap P \neq \emptyset \wedge R \cap P^c \neq \emptyset$;
7 $\quad$ **else** $\delta \leftarrow \frac{\delta}{2}$; $h \leftarrow \frac{h}{2}$;
8 **end**

Algorithm 5.2: Dynamic analysis technique for checking whether a region satisfies a guarantee predicate.

In lines 4 and 6, the function $w_p(R_j)$ would return $\top$ if and only if $\forall x \in R, w_p(x) = \top$. In line 5, the function $w_p(R_j)$ would return $\bot$ if and only if $\forall x \in R_j, w_p(x) = \bot$. The proof of soundness and relative completeness of the algorithm are very similar to proofs of soundness and relative completeness of safety verification given in Theorems 14 and 15. Similar to the safety verification procedure, this algorithm might not terminate if the region does not robustly satisfy the predicate. Hence, in practice, we terminate after reaching thresholds on δ and h and label the region as *unknown*.

5.3 Case Study: NASA's ALAS Protocol For Parallel Landing

ALAS is a pair-wise algorithm, where the two aircraft are referred to as *ownship* and *intruder* (as shown in Figure 5.1). When the ALAS algorithm is deployed in an aircraft following the SAPA procedure, the aircraft considers itself to be the ownship, while any other aircraft is considered to be an intruder. The alerting logic of the ALAS algorithm consists of several checks including conformance of the onwship to its nominal landing trajectory, aircraft separation at current time, and projected aircraft separation for different trajectories.

A formal static analysis of the ALAS algorithm is challenging due to the complexity of the SAPA protocol and the large set of configurable parameters of the ALAS

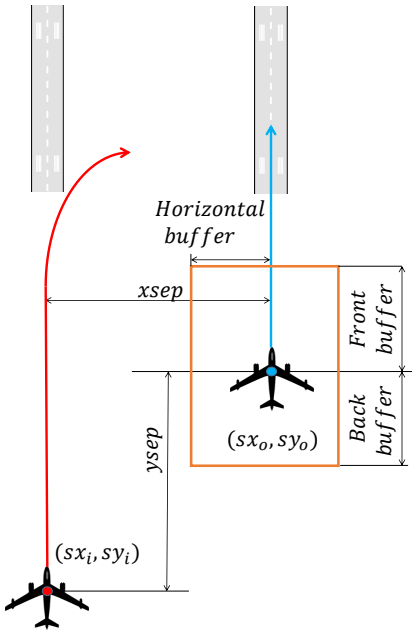


Figure 5.1: Possible blundering scenario during parallel approach of aircraft. Intruder (red) & ownship (blue).

algorithm that enable different alerting thresholds, aircraft performances, and runway geometries. In this case study, we consider the component of the ALAS alerting logic that checks violations of predefined separation minima for linear and curved projected trajectories of the current aircraft states. This component is one of the most challenging to analyze since it involves nonlinear dynamics. Safety considerations regarding communication errors, pilot and communication delays, surveillance uncertainty, and feasibility of resolution maneuvers are not considered in this case study.

For the analysis of the landing protocol, this case study considers a blundering scenario where the intruder aircraft turns towards the ownship during the landing approach. The dynamics of the aircraft are modeled as a hybrid system with continuous variables sx_i, sy_i, vx_i, vy_i and sx_o, sy_o, vx_o, and vy_o representing the position and velocity of intruder and ownship respectively. The hybrid system has two modes: *approach* and *turn*. The mode *approach* represents the phase when both aircraft are heading towards the runway with constant speed. The mode *turn* represents the blundering trajectory of intruder. In this mode, the intruder banks at an angle ϕ_i to turn away from the runway towards the ownship. The guard condition on the discrete transition determines the time of transition from *approach* to *turn*. In this mode, the differential equation of the ownship remains the same as that of *approach*, but the

intruder's turning motion with banking angle ϕ_i is

$$\begin{bmatrix} \dot{sx_i} \\ \dot{sy_i} \\ \dot{vx_i} \\ \dot{vy_i} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & 0 & 0 & \omega_i \\ 0 & 0 & -\omega_i & 0 \end{bmatrix} \begin{bmatrix} sx_i \\ sy_i \\ vx_i \\ vy_i \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \omega_i - c_y \\ \omega_i + c_x \end{bmatrix}, \tag{5.1}$$

where c_x and c_y are constant functions of the initial states of the ownship and intruder, and ω_i is the angular speed of intruder. Given the bank angle ϕ_i, the angular speed is given by $w_i = \frac{G|\tan(\phi_i)|}{\sqrt{{vx_i}^2+{vy_i}^2}}$, where G is the gravitational constant. The upper bound on the bank angle ϕ_i is denoted as ϕ_{max}.

The system starts in the *approach* mode with the initial position of the intruder at $sx_i = sy_i = 0$ and the ownship at $sx_o = xsep$ and $sy_o = ysep$, where $xsep$ denotes the lateral separation between the runways and $ysep$ denotes the initial longitudinal separation between the aircraft. The initial velocities of both aircraft along the x-axis are 0 and the initial velocities along the y-axis are parameters. The discrete transition from *approach* mode to *turn* mode is enabled when the time spent in approach mode is in the interval $[2.3, 2.8]$. These parameters and the initial values of the variables are constrained by the SAPA procedure [96].

5.3.1 Alerting Logic And Verification Of Temporal Precedence Property

The alerting logic of ALAS considered in this case study issues an alert when the aircraft are predicted to violate some distance thresholds called *Front* and *Back* [137]. To predict this violation, the alerting logic of ALAS projects the current state of the system with three different dynamics: first, the intruder does not turn, i.e., banking angle 0°, second, the intruder turns with the specified bank angle ϕ_i and third, the intruder turns with the maximum bank angle ϕ_{max}. If any of these projections violates the distance thresholds, then an alert is issued. The alert predicates for the each one of these projections are represented by $Alert_0$, $Alert_{\phi_i}$ and $Alert_{\phi_{max}}$, respectively. Thus, the alerting logic considered in this case study is defined as $Alert \equiv Alert_0 \vee Alert_{\phi_i} \vee Alert_{\phi_{max}}$.

The alert predicates $Alert_0$, $Alert_{\phi_i}$ and $Alert_{\phi_{max}}$ are guarantee predicates. The lookahead function for $Alert_\pi$ is defined as follows: from a given state x, it computes the projected trajectory of the aircraft when intruder turns at bank angle π. If these trajectories intersect, then it computes the times of intersection. That is, it computes t_i, t_o such that $sx'_i(t_i) = sx'_o(t_o)$ and $sy'_i(t_i) = sy'_o(t_o)$, where $sx'_i, sy'_i, sx'_o, sy'_o$ represent the positions of the intruder and ownship aircraft in the projected trajectory. If such t_i and t_o exist, the $Alert_\pi$ is defined as:

$$\begin{aligned} Alert_\pi(x) \equiv \text{ iff } t_i > t_o \,? \quad & (\Delta t^2 \times (vx_o^2 + vy_o^2) < Back^2) \\ : \quad & (\Delta t^2 \times (vx_o^2 + vy_o^2) < Front^2), \end{aligned}$$

where $\Delta t = t_i - t_o$. If such t_i and t_o do not exist, then $Alert_\pi(x) = \bot$. The expression $a \,?\, b \,:\, c$ is a short hand for $\textbf{if}(a)$ **then** b **else** c.

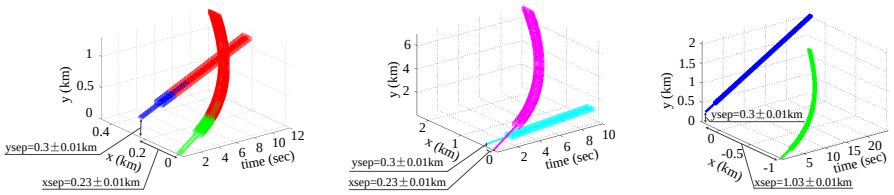
As the guarantee predicates cannot be handled by SMT solvers, we use the procedure checkGuarantee given in Algorithm 5.1 for handling them. In this case study, the proposed technique is used to resolve the nonlinearities of t_o and t_i in the $Alert_\pi$ predicate. As given in procedure checkGuarantee, the following steps are performed to resolve the nonlinear guarantee predicate. First, bounded time reachtree Δ is computed for the projected dynamics. Then, for each reachtube in the reachtree, the intervals T_o and T_i are computed such that $t_i \in T_i$ and $t_o \in T_o$. Finally, an overapproximation $Alert'_\pi$ of $Alert_\pi$ is computed as: $Alert'_\pi(x) = \top$ iff

$$\begin{aligned} T_i > T_o \,? \quad & (\Delta T^2 \times (vx_o^2 + vy_o^2) < Back^2) \\ : \quad & (\Delta T^2 \times (vx_o^2 + vy_o^2) < Front^2), \end{aligned}$$

where $\Delta T = T_i - T_o$. The numerical values of T_i and T_o computed simplify the $Alert'_\pi$ predicate and can be handled by SMT solvers.

A state of the system where the intruder aircraft is inside a safety area surrounding the ownship is said to be *Unsafe*. This case study considers a safety area of rectangular shape that is *SafeHoriz* wide, starts a distance *SafeBack* behinds the ownship and finishes a distance *SafeFront* in front of the ownship. The values *SafeHoriz*, *SafeBack* and *SafeFront* are given constants. Formally, the predicate *Unsafe* is defined as $Unsafe(x) \equiv (sy_i > sy_o \;?\; sy_i - sy_o < SafeFront : sy_o - sy_i < SafeBack)$ and $|sx_i - sx_o| < SafeHoriz$.

The specification for ALAS considered in this case study is that an alert is raised



(a) Scenario 1. (b) Scenario 2. (c) Scenario 3.

Figure 5.2: Figure depicting the set of reachable states of the system. Red color regions indicate that the safe separation is violated.

at least b seconds before the intruder violates the safety buffer. This can written as a temporal precedence property $\mathit{Alert} \prec_b \mathit{Unsafe}$. In this case study we consider scenarios where alert precedes unsafe configuration by at least 4 time units.

5.3.2 Verification Scenarios And C2E2 Performance

Algorithm 5.1 that verifies the temporal precedence property has been implemented as a prototype extension in the tool Compute Execute Check Engine (C2E2). C2E2 accepts Stateflow (SF) charts as inputs, translates them to C++ using CAPD for generating rigorous simulations. For checking SAT/SMT queries, it uses Z3 [47] and GLPK[1]. The discrepancy functions for the aircraft dynamics were obtained by computing incremental Lyapunov-like function using MATLAB [55]. The following experiments were performed on Intel Quad Core machine 2.33 GHz with 4GB memory.

The temporal precedence property $\mathit{Alert} \prec_b \mathit{Unsafe}$ is checked for several configurations of the system, i.e., values of parameters and initial values of state variables. For these experiments, the time bound for verification is set to 15 seconds and the time bound for projection is set to 25 seconds.

Scenario 1. The system configuration is specified by the following parameters and variables: $xsep \in [0.22, 0.24]$ km, $ysep \in [0.2, 0.4]$ km, $\phi_i = 30°$, $\phi_{max} = 45°$, $vy_o = 0.07$ km/s and $vy_i = 0.08$ km/s. With this configuration, C2E2 proves that the system satisfies the temporal precedence property $\mathit{Alert} \prec_4 \mathit{Unsafe}$ and an alert is generated 4.38 seconds before the safety is violated. The set of reachable states of the ownship and the intruder when the safety property is violated is shown in *red* and the safe states reached are shown in blue and green respectively in Figure 5.2(a).

[1] http://www.gnu.org/software/glpk

Scen.	$A \prec_4 U$	time (m:s)	Refs.	$A \prec_t U$
6	False	3:27	5	2.16
7	True	1:13	0	–
8	True	2:21	0	–
6.1	False	7:18	8	1.54
7.1	True	2:34	0	–
8.1	True	4:55	0	–
9	False	2:18	2	1.8
10	False	3:04	3	2.4
9.1	False	4:30	2	1.8
10.1	False	6:11	3	2.4

Table 5.1: Running times. Columns 2-5: Verification Result, Running time, # of refinements, value of b for which $A \prec_b U$ is satisfied.

Scenario 2. Increasing the intruder velocity to $vy_i = 0.11$ km/s, and bank angle $\phi_i = 45°$ from the configuration of Scenario 1 results in Scenario 2. In this case, the safe separation between the intruder and the ownship is always maintained as the intruder completes the turn behind the ownship. Also, the alarm is not raised and hence the property *Alert* $\prec_4$ *Unsafe* is satisfied.

Scenario 3. Changing the configuration by $vy_i = 0.11$ km/s, $xsep \in [1.02, 1.04]$ km, and $\phi_i = 45°$ from Scenario 1 results in Scenario 3. C2E2 proves that the simplified alerting logic considered in this case study issues a false-alert, i.e., an alert is issued even when the safety of the system is maintained. Though the property *Alert* $\prec_4$ *Unsafe* is not violated, avoiding such circumstances improves the efficiency of the protocol and C2E2 can help identify such configurations.

Scenario 4. Placing the intruder in front of ownship, i.e., $ysep = -0.3$ km and $vy_i = 0.115$ km/s from configuration in Scenario 1 results in Scenario 4. C2E2 proves that the simplified alerting logic considered in this case study misses an alert, i.e., does not issue an alert before the safety separation is violated. Such scenarios should always be avoided as they might lead to catastrophic situations. This demonstrates that C2E2 can aid in identifying scenarios which should be avoided and help design the safe operational conditions for the protocol.

Scenario 5. Reducing the $xsep \in [0.15, 0.17]$ km and $ysep \in [0.19, 0.21]$ km from configuration in Scenario 1 gives Scenario 5. For this scenario, C2E2 did not terminate in 30 mins. Since the verification algorithm (Algorithm 5.1) is sound and relatively complete only if the system robustly satisfies the property, it is conjectured that Scenario 5 does not satisfy the property robustly. The partitioning and the simulation parameters at the time-out were $\delta = 0.0005$ and time step $h = 0.001$. These values are an order of magnitude smaller than the typical values for termination, e.g., $\delta = 0.005$ and $h = 0.01$, which supports the conjecture that Scenario 5 does not satisfy the property robustly.

The running time of verification procedure and their outcomes for several other scenarios are presented in Table 5.1. Scenarios 6-8 introduce uncertainty in the initial velocities of the aircraft with all other parameters remaining the same as in Scenario 1. The velocity of the aircraft are changed to be $vy_o \in [0.07, 0.075]$ in Scenario 5, $vy_i \in [0.107, 0.117]$ in Scenario 6, and $vx_i \in [0.0, 0.005]$ in Scenario 7 respectively. Scenarios $S.1$ is similar to Scenario S (for S being 6,7,8), but with twice the uncertainty in the velocity. Scenario 9 is obtained by changing the runway separation to be $xsep = 0.5 \pm 0.01$. Scenario 10 is obtained by reducing the $xsep = 0.2 \pm 0.01$. Scenario $S.1$ is similar to Scenario S (for S being 9,10) however with twice the time horizon for verification and projection. These results suggest that the verification time depends approximately linearly on the time time horizon.

5.4 Powertrain Control System Verification

In this section, we present the verification case study of powertrain control systems. Due to the increased importance of fuel efficiency in cars, and the environmental impacts of reduced emissions, automotive companies are interested in integrating fine tuned control software in the powertrain control systems. Recently a suite of benchmarks were published in [95, 94] to introduce realistic industrial scale models to academics interested in formal verification. These benchmarks consist of Simulink$^{\text{TM}}$/Stateflow$^{\text{TM}}$ models (a popular modeling framework developed by Mathworks$^{\text{TM}}$) with increasing level of complexity and sophistication. These models capture the behavior of chemical reactions in internal combustion engine and hence hybrid automata are ideally suitable to capture both the discrete transitions of control software and the continuous parameters in these models.

The complex of the models captures all the interactions taking place in a physical process and faithfully models the control software. Hence, it contains several hierarchical components in Simulink/Stateflow with look-up tables, and delay differential equations. Currently, no state of the art formal verification tool can handle such complex system models. This complex model has been simplified to a model with periodic inputs to ordinary differential equations using several heuristics. These heuristics do not provide any theoretical guarantees, but as per the authors, seem to exhibit similar behavior (with respect to conformance of properties) of the complex model [95]. This model has been further simplified to a hybrid automaton with polynomial ordinary differential equations. In this case study, we verify the properties of polynomial hybrid automaton model of the powertrain control system.

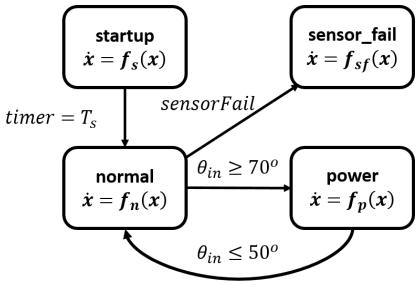
The rest of the section is as follows: we first introduce the model and its characteristic properties. We highlight the main challenge in verifying such system and present the details of the model that is verified in this case study. We then present the STL properties of interest and explain how the STL properties are encoded as safety properties. We then describe the optimizations used in verification procedure. Finally we present the verification results for different properties.

5.4.1 Nonlinear Hybrid Powertrain Model

The Simulink$^{\text{TM}}$ model for the powertrain control system is shown in Figure 5.3(a). The system has four continuous variables p, λ, p_e, i (see Figure 5.3(b)), and four modes of operation: *startup*, *normal*, *power*, and *sensor_fail*. The system also receives an input signal θ_{in} from the environment. This continuous signal models the throttle input given by the user. The mode switches (also called *transitions*) are brought about by changes in the input *throttle angle* θ_{in} or *failure* events.

The rest of the Simulink$^{\text{TM}}$ diagram defines polynomial differential equations that govern the evolution of the continuous variables in the four different modes. As an example, we reproduce the differential equation for normal mode of operation.

$$\begin{aligned}
\dot{p} =& c_1(2\theta_{in}(c_{20}p^2 + c_{21}p + c_{22}) - c_{12}(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p)) \\
\dot{\lambda} =& c_{26}(c_{15} + c_{16}c_{25}F_c + c_{17}c_{25}^2F_c^2 + c_{18}\dot{m}_c + c_{19}\dot{m}_c c_{25}F_c - \lambda) \\
\dot{p_e} =& c_1(2c_{23}\theta_{in}(c_{20}p^2 + c_{21}p + c_{22}) - (c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e)) \\
\dot{i} =& c_{14}(c_{24}\lambda - c_1 1).
\end{aligned}$$



(a) Hybrid automata model of powertrain control system.

Variable	Description
p	Intake manifold pressure
p_e	Intake manifold pressure estimate
λ	Air-fuel ratio
i	Integrator state, control variable
θ_{in}	Throttle angle

(b) Table of variables in powertrain control system.

Figure 5.3: Figure showing the model of the powertrain control in (a) and its variables in (b).

Here $F_c = \frac{1}{c_{11}}(1+i+c_{13}(c_{24}\lambda - c_{11}))(c_2 + c_3\omega p_e + c_4\omega p_e^2 + c_5\omega^2 p_e)$, $\dot{m}_c = c_{12}(c_2 + c_3\omega p + c_4\omega p^2 + c_5\omega^2 p)$, and all the c_i's are constant parameters of the model (from [95]).

Notice that these differential equations contain θ_{in} on the right hand side. The behavior of the system would hence depend upon the value of the input given. Although preliminary techniques verifying models with inputs (that treat the input as an unknown parameter) exist, such techniques cannot handle the complexity of the differential equations for the powertrain control system. Hence we construct a closed hybrid automata model by fixing a family of input signals given as θ_{in}. The family of input signals considered for this case study are modifications of the input signals considered in [95]. We select two such families of input signals that visit all the modes of the Stateflow model.

5.4.2 Specification Of Powertrain Control Systems

The required specification of powertrain control systems was given in [95] as a number of STL properties. Although, in general, STL properties might have nested temporal constructs (see [122]), for the sake of our case study, we do not need the full semantics of STL. In the rest of this section, we present only the semantics for properties of interest for powertrain control system.

Atomic propositions in STL are of the form $q \triangleq x_i \geq | = | \leq c$ where the value of continuous variable x_i is compared to the constant c. Extended propositions can be boolean combinations of such atomic propositions. An execution σ at time t satisfies the atomic predicate q if and only if $\sigma(t).x_i \geq | = | \leq c$. Extension for boolean combination of atomic propositions follows trivially.

STL has a temporal construct $\Box$ for specifying temporal properties of signals. A non-nested formula in STL is given as $\Box_{[a,b]}Q$ where Q is a boolean combination of atomic propositions. An execution σ is said to satisfy $\Box_{[a,b]}Q$ if and only if $\forall t \in [a,b], \sigma(t)$ satisfies Q.

Powertrain control system has two main goals: first is to maintain the air fuel ratio in an operating region and the second is to ensure that air fuel ratio quickly converges to a value close to the reference value of fuel air ratio, that is geared towards fuel efficiency. An example of the first requirement which ensure that the fuel air ratio is in an operating region is given as the following STL formula:

$$\Box_{[0,T]}(0.8\lambda_{ref} \leq \lambda \leq 1.2\lambda_{ref}). \tag{5.2}$$

The performance requirement requires that whenever the mode of operation changes, the air fuel ratio converges close to reference value in a given time interval. As this mode operation happens whenever the input throttle θ_{in} rises from 0 to 60, one such STL requirement is given as

$$rise \Rightarrow \Box_{(\eta,\zeta)}(0.98\lambda_{ref} \leq \lambda \leq 1.02\lambda_{ref}), \tag{5.3}$$

where η and ζ are parameters for the property.

For verifying such STL properties, we encode them as safety properties. For doing the same, we introduce a special variable called *clk* variable that keeps track of time elapsed. Using this, the violation of STL property can be encoded as reaching an unsafe set. For example, to verify the STL property $\Box_{[3,4]}x > 4$, we encode it as safety property as follows. We add the clock variable *clk* to the hybrid system, set its rate of change $\dot{clk} = 1$ in all the locations of hybrid system and leave its value unchanged for all the discrete transition. We say that the system satisfies the STL property if and only if it is *safe* with respect to the unsafe set $U \triangleq \{clk \leq 4 \wedge clk \geq 3 \wedge x < 4\}$.

5.4.3 Verification Algorithm And Optimizations

As we have encoded the verification of STL properties as safety verification, we can use the checkSafety procedure given in Algorithm 4.2 for verifying properties of powertrain control system. However, checkSafety procedure requires that a discrepancy function for each location be provided as a model annotation. For powertrain control system, existing proof theoretic certificates in control theory that use SOS tools (given in

first part of Section 3.4) could not produce a useful annotation. Hence, we use a technique for computing on-the-fly discrepancy function computation technique proposed in [60][2]. For implementing the on-the-fly discrepancy function procedure, C2E2 requires the ability to compute the the eigenvalues of the symmetric parts of the Jacobian and maximizing the norm of the interval matrices. The former is done using using Eigen library [59] and the latter is done using norm calculations with interval arithmetic. We now discuss some of the optimizations and engineering performed as a part of this case study.

Coordinate Transformation. An important technical detail that makes the implementation scale is the coordinate transformation proposed in [60]. For Jacobian matrices with complex eigenvalues the local discrepancy computed directly using the above algorithm can be a positive exponential even though the actual trajectories are not diverging. This problem can be avoided by first computing a local coordinate transformation and then applying the algorithm. Coordinate transformation provides better convergence, but comes with a multiplicative cost in the error term. This trade-off between the exponential divergence rate and the multiplicative error has to be tuned by choosing the time horizon over which the coordinate transformation is computed.

Model Reduction. In *start up* and *power* mode of the system, the differential equation does not update the value of the integrator variable i, i.e. i does not appear in the right hand side of the differential equations for variables p, λ, p_e. Moreover $\dot{i} = 0$. We take advantage of this observation, and consider only the dynamics of the variables p, λ, and p_e for computing local discrepancy. Without this optimization, the maximum eigen values of symmetric part of Jacobian would return 0 and the error computation would result in a very coarse approximation (exponential blow up) of the discrepancy function.

5.5 Experimental Results On Powertrain Challenge

We have implemented on-the-fly discrepancy function computation as a prototype extension of the tool C2E2. As the number of switching signals corresponding to driver behaviors is infinite, we analyze a subset of all possible switching signals. Specifically,

[2]The modified tool and related files are available from http://publish.illinois.edu/c2e2-tool/powertrain-challenge/

Property	Mode	Sat.	Sim.	Time
$\Box_{T_s,T}\lambda \in [0.8\lambda_{ref}, 1.2\lambda_{ref}]$	*all modes*	yes	53	11m58s
$\Box_{[0,T_s]}\lambda \in [0.8\lambda_{ref}, 1.2\lambda_{ref}]$	*startup*	yes	50	10m21s
$\Box_{[T_s,T]}\lambda \in [0.95\lambda_{ref}, 1.05\lambda_{ref}]$	*normal*	yes	50	10m28s
$\Box_{[T_s,T]}\lambda \in [0.8\lambda^{pwr}_{ref}, 1.2\lambda^{pwr}_{ref}]$	*power*	yes	53	11m12s
$\Box_{[0,T_s]}\lambda \in [0.98\lambda_{ref}, 1.02\lambda_{ref}]$	*startup*	no	2	0m24s
$\Box_{[T_s,T]}\lambda \in [0.9\lambda^{pwr}_{ref}, 1.1\lambda^{pwr}_{ref}]$	*power*	no	4	0m43s
$rise \Rightarrow \Box_{(\eta,\zeta)}\lambda \in [0.9\lambda_{ref}, 1.1\lambda_{ref}]$	*startup*	yes	50	10m40s
$rise \Rightarrow \Box_{(\eta,\zeta)}\lambda \in [0.98\lambda_{ref}, 1.02\lambda_{ref}]$	*normal*	yes	50	10m15s
$(\ell = power) \Rightarrow \Box_{(\eta^{pwr},\zeta)}\lambda \in [0.95\lambda^{pwr}_{ref}, 1.05\lambda^{pwr}_{ref}]$	*power*	yes	53	11m35s
$(\ell = power) \Rightarrow \Box_{(\eta^{s},\zeta)}\lambda \in [0.95\lambda^{pwr}_{ref}, 1.05\lambda^{pwr}_{ref}]$	*power*	no	4	0m45s

Table 5.2: Table showing the result and the time taken for verifying STL specification of the powertrain control system. Sat: Satisfied, Sim: Number of simulations performed. All the experiments are performed on Intel Quad-Core i7 processor, with 8 GB ram, on Ubuntu 11.10.

we pick two behaviors that cover almost all of the STL properties provided in [95]. Table 5.2 provides the results of verifying different STL properties.

The first six properties provided in Table 5.2 are invariant properties. These invariant properties can be global (i.e. correspond to all modes) or could be restricted to a certain mode of operation provided in the *Mode* column. The invariants assert that the air-fuel ratio should not go out of the specified bounds. Observe that C2E2 could not only prove that the given specification is satisfied, but also that a stricter version of invariants for *startup* and *power* modes is violated. The next four properties are about the settling time requirements. These requirements enforce that in a given mode, whenever an action is triggered, the fuel air ratio should be in the given range provided after η (or η^{pwr} for power mode) time units. Similar to the invariant properties, C2E2 could also find counterexample for a stricter version of the settling time requirement (η^s settling time instead of η) in *power* mode. When C2E2 finds an overapproximation that violates a given property, it immediately terminates and hence C2E2 takes less time when it finds counterexamples. The parameters used for verification are $\eta = \eta^{pwr} = 1$, $\eta^s = 0.5$, $T_s = 9$, $T = 20$, $\lambda_{ref} = 14.7$, $\lambda^{pwr}_{ref} = 12.5$, and $\zeta = 4$. Set of reachable states of the powertrain control system for a given driver behavior is provided in Figure 5.4.

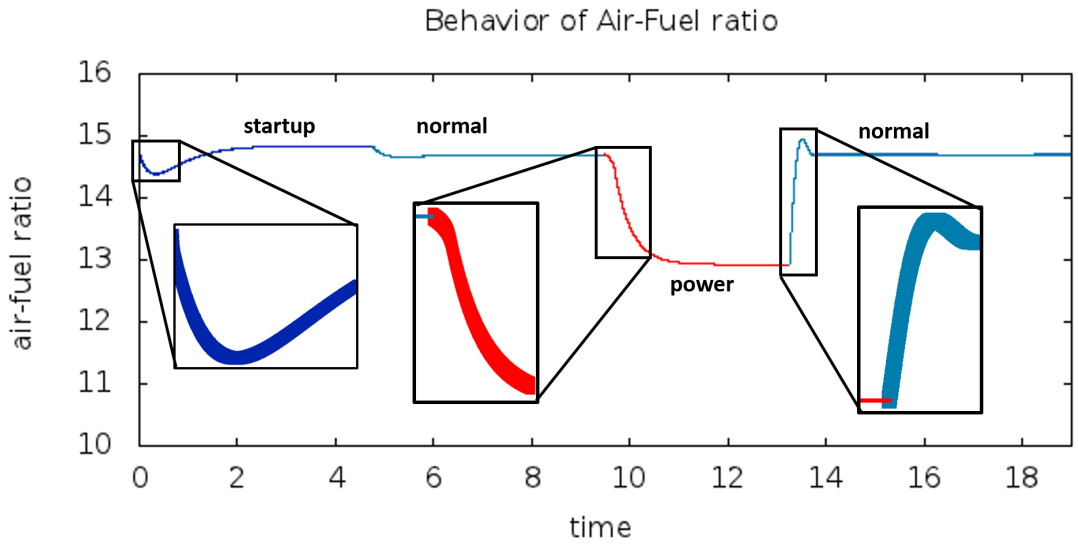


Figure 5.4: Figure showing the reachable set of the powertrain control system for a given user behavior that visits different modes.

5.6 Conclusion And Related Work

In this chapter we have presented two case studies, the first is an alerting mechanism in a parallel aircraft landing and the second is verifying STL properties of powertrain control systems. Both these systems present realistic examples of CPS models and highlight a challenging problem for verification. The challenge in verifying ALAS system is to handle the nonlinear guarantee predicate given implicitly as solution of ODE. We overcome this challenge by using a dynamic analysis technique for handling guarantee predicates. The challenge in verifying powertrain control system is the complexity of the differential equations that govern the dynamics. We overcome this challenge by implementing an algorithm that computes on-the-fly discrepancy function for powertrain control systems. These two case studies demonstrate that dynamic analysis techniques can be applied to realistic case studies without losing the theoretical guarantees.

Verification of air traffic control protocols has been an important application for CPS verification techniques [164, 165, 154, 148]. Verification of collision avoidance protocols such as TCAS and ACASX using Keymeara was done in [142] and [93] respectively. Verification of a Small Aircraft Transportation Systems (SATS) was performed using PVS in [131]. Parametrized distributed protocols have been verified in [120, 99]. The case study presented in this thesis is the first one to verify the ALAS

system. The benchmarks of powertrain control systems has been a very recent area of interest. In [95, 94] the authors present the verification challenges in powertrain system and present some falsification results. New falsification techniques have been developed recently to improve the search for falsifying execution [172, 171]. The case study on powertrain control systems is the first to *formally verify* the powertrain control system.

Chapter 6

Dynamic Analysis of Distributed CPS

In this chapter we present a dynamic analysis for inferring global predicates for a distributed Cyber-Physical Systems. Implementations of distributed CPS systems have uncertainties in sensor measurements, actuation parameters, and clocks drifts between different nodes. Thus, local recordings from nodes is insufficient for inferring a global predicate (on the software state or the physical state) of the distributed system. In this chapter we present a technique for inferring global predicates in the software state (such as deadlock detection), and physical state (such as safe separation between nodes) using the local recordings. We apply the technique on a several distributed robotics applications like waypoint tracking, geocasting, and distributed traffic control protocol, and verify global predicates such as safety and deadlock detection. The results presented in this chapter were presented originally in [54].

6.1 Introduction

Consider programming a group of mobile robots for collaborative construction. Each robot executes a program implementing one or more distributed algorithms, moves and manipulates its environment, and exchanges messages over an unreliable communication channel with other robots. In a realistic setting, numerous program parameters have to be configured for the system to work, there are failures, and moreover, there is no common notion of time. This is also the case with a variety of other real-time systems, such as autonomous vehicle platoons, industrial automation and warehouse management systems, and distributed process control systems. Even where the high-level, centralized and idealized control algorithm is well-understood, failures, timing-errors, and message delays make their implementation challenging. One barrier is that the distributed programmer, unlike her sequential compatriots, does not enjoy the benefits of a debugger that can deterministically replay an execution and identify the precise step where a program goes awry and later manifests as an incor-

rect behavior. The uncertainties in the message delays, concurrency, timing of events, and the interaction of the program with its physical environment, make deterministic replay impossible from the *traces* of the individual agents (robots or processes).

In this chapter, we present a procedure for answering the following types of queries. Given a trace β (given in Definition 7)—a sequence of state recordings for each of the individual agents—of the distributed system $\mathcal{A}$ and a global property P, does there exist a real-time t when all possible (or at least one) bounded executions of $\mathcal{A}$ that *correspond* to the recording β, satisfy the property P. The algorithm combines dynamic analysis of the trace β with symbolic overapproximation of bounded reach sets obtained through static analysis of $\mathcal{A}$. It is always sound, and it is complete when both the static analysis and the dynamically generated information are exact.

How does this procedure help in developing distributed cyber-physical systems? Suppose P describes a bad state (such as collision or deadlock) of the system and the algorithm answers 'yes' with a witnessing time t. Then the developer can learn that all the executions that resolve the uncertainties in a manner that is consistent with the recordings in β, demonstrate the violation of P at time t. She can then focus this particular set of executions around time t to isolate the bug. If the algorithm answers that there is no time when *any* execution satisfies P, then it establishes the complement of P for the bounded time interval. On the other hand, if the algorithm demonstrates that there is some time t when some overapproximation of the executions satisfy P, then a definite conclusion cannot be reached, and one has to either record a trace with more information or relax the property P. Under additional assumptions, the algorithm is complete—the overapproximation is exact—the procedure pinpoints an actual execution that violates the property; such traces can then be used for debugging and diagnosis.

Overview of the procedure: A *trace* β of a distributed system with a set I of agents is a collection of traces $\{\beta_i\}_{i\in I}$, where each trace β_i is a sequence of snapshots or *observations* recorded locally by agent $\mathcal{A}_i$. Our technique should generalize to traces with quantized and partial state observations, however, for simplifying the exposition we assume that each observation is a recording of the complete state of $\mathcal{A}_i$. Since this is a distributed system, the agents do not share any common knowledge about time. In fact, the only timing information available about an observation is the time stamp value from the agent's local clock which may not be synchronized with other agents or with any external clock. Since this is a cyber-physical system,

in between successive recordings, the state of the agents—for example, the positions of the robots—continue to evolve continuously.

The first step in our algorithm is to infer from the trace β, for each observation $\mathbf{v} \in \beta$, a real-time interval (called the *observation interval*) over which the observation *could* have been recorded. This inference is based on the timing information available about the local clocks and the time-bounds induced by the causality of the messages exchanged between agents.

From the observation intervals, we then compute the set of states that could be reached by agent $\mathcal{A}_i$ between successive observations. This step strongly relies on static analysis approaches available for individual agent models. In this chapter, we assume that behavior of the physical variables is modeled using rectangular hybrid automata [10], and as a consequence, we can symbolically compute expressions for forward and backward reach sets ($Pre(,)$ and $Post(,)$ defined formally in Definition 32) from a given set of states (for more details on reachable set computation for hybrid systems, please refer to Section 1.3). For more general dynamics, although it is difficult to compute the reachable set exactly, one can potentially use the numerical techniques present in literature (see Section 1.3). Alternatively, one can also use the dynamic analysis techniques presented in Chapters 3 and 4. In this chapter, we assume that the reachable set computed can be expressed as a logical formula in linear/nonlinear arithmetic solvers such as Z3 [48].

The final step combines the symbolic reachability computations of the individual agents and checks if there exists a time t when the complete distributed system violates the given property.

There are two sources of imprecision in the procedure. First, the $Pre(,)$ and $Post(,)$ expressions obtained through static analysis may not be exact. This may be the case either because the model of the system that is analyzed is not exact (e.g., the program or the dynamics are not known exactly), or because the model cannot be analyzed exactly. There exists a rich body of literature addressing the latter (see, Section 1.3 for more details and the recent the proceedings of HSCC [63, 83]). While there are classes of systems for which reach sets can be computed exactly, for most systems with complex dynamics, these computations are necessarily overapproximations of the actual dynamics. The second source of imprecision is the inaccuracy of the dynamic information recorded in the traces. Specifically, these inaccuracy arises from the lack of exact information about the timing of the recorded observations. For each observation, we calculate an interval of real time in which it *may* have occurred.

If these intervals are larger than what is actually possible, then the set of possible executions computed from the given (inaccurate) trace may subsume the actual set of possible executions. We show that the procedure is complete, if both these types of inaccuracies can be eliminated.

We have built a programming platform, called StarL [170, 114], for quickly developing distributed Android applications communicating over ad-hoc WiFi networks and Bluetooth. Using StarL, we have implemented applications such as peer-to-peer chat, geocasting, coordinated distributed search with mobile robots, and distributed traffic control [170][1]. We have implemented the procedure described in this chapter in a prototype tool that uses the Z3 SMT-solver [48] and have analyzed numerous traces from distributed Android applications with up to 20 agents. Analyzing a 10 second trace for 20 agents typically takes between a few seconds to a couple of minutes. These performance figures suggest that the procedure can scale. The traces analyzed in these experiments are obtained from three applications: waypoint following, geocasting, and a distributed traffic control application that requires mutual exclusion. For these traces, we are able to establish and find counterexamples for interesting properties, such as, (a) no two mobile robots ever came within a certain distance of each other, (b) every message that was geocast to a certain region was received by another agent iff the agent happened to be in that region during a certain time interval, and (c) there are no deadlocks at traffic intersections.

6.2 From Distributed Traces To Global Properties

In this section, we use the notations developed for modeling distributed CPS that were presented in Section 2.3. We present new definitions for observation intervals and present a procedure to infer the global predicates of distributed CPS by performing static and dynamic analysis on a trace.

6.2.1 Observation Intervals

Each recorded state $\mathbf{v}_i[j]$ in a trace β_i is associated with a local clock value $\mathbf{v}_i[j].clk$, but thus far we have not made any assumptions about clk apart from it being con-

[1]In fact, the errors encountered in developing these applications partly motivated the current work.

tinuous. In distributed computing systems, the impossibility of constructing globally synchronized clocks is a fundamental limitation [167]. Therefore, it is unrealistic to assume that all the $\mathbf{v}_i[j].clk$ values *equal* the real-time of the execution σ at which $\mathbf{v}_i[j]$ is recorded. At the same time, many embedded devices have local clocks that are synchronized up to some accuracy using distributed clock synchronization algorithms. The next definition specifies when an observation is synchronized to some accuracy.

Definition 31 *For a recorded state $\mathbf{v}_i[j] \in Q_i$ in a trace β of* System *and a positive constant $\mu \in \mathbb{R}_{\geq 0} \cup \{\infty\}$, $\mathbf{v}_i[j]$ is said to be* μ-synchronized *if for every execution $\sigma \in \mathsf{TraceInv}_\beta$, (1) there exists t in the real-time interval $J \triangleq [max(\mathbf{v}_i.clk - \mu, 0), \mathbf{v}_i.clk + \mu]$ such that, $\sigma(t).V_i = \mathbf{v}_i[j]$ and (2) for all $t \notin J$, $\sigma(t).V_i \neq \mathbf{v}_i[j]$. β_i is* μ-synchronized *if every observation in β_i is μ-synchronized.*

In other words, for any execution that corresponds to β, the state observation $\mathbf{v}_i[j]$ occurs within the interval J and only within that interval. If the agents implement a clock synchronization algorithm that guarantees that the local clocks are synchronized to the real-time with an accuracy of μ, then it is guaranteed that each β_i in the trace β is μ-synchronized. This definition is more general, in that, it allows different agents' clocks to be synchronized to real-time with different accuracy, and even different states within the same trace can be associated with local clocks that have different accuracy. Note that an ∞-synchronized recorded state provides no information about the real-time of the recording. Also, if $\mathbf{v}_i[j]$ is μ-synchronized in β, then it is also μ'-synchronized for any larger $\mu' \geq \mu$.

For a given trace β, the observation interval $[L(\mathbf{v}), U(\mathbf{v})]$ for a μ-synchronized observation $\mathbf{v}$ is defined inductively along the happens-before (happens-after) chain as follows:

$$L(\mathbf{v}) = \max\left(0, \mathbf{v}.clk - \mu, \max_{\mathbf{u} \in before(\mathbf{v})} L(\mathbf{u})\right), \text{ and} \tag{6.1}$$

$$U(\mathbf{v}) = \min\left(\mathbf{v}.clk + \mu, \min_{\mathbf{u} \in after(\mathbf{v})} U(\mathbf{u})\right). \tag{6.2}$$

We define the *duration of a trace* β, $dur(\beta)$, as $\max_{\mathbf{v} \in [\![\beta]\!]} U(\mathbf{v})$.

The following lemma is proved by induction on the length of the happens-before (and happens-after) chain of an observation.

Lemma 25 *For every execution $\sigma \in \mathsf{TraceInv}_\beta$, for every observation $\mathbf{v}$ in β of some*

agent i, (1) there exists t in the real-time interval $[L(\mathbf{v}), U(\mathbf{v})]$, such that $\sigma(t).V_i = \mathbf{v}$, and (2) for all $t \notin [L(\mathbf{v}), U(\mathbf{v})]$, $\sigma(t).V_i \neq \mathbf{v}$.

6.2.2 Filling in the Gaps

Throughout this section, we fix an automaton $\mathcal{A}_i$ and a trace $\beta_i = \mathbf{v}_i[1], \ldots, \mathbf{v}_i[k]$ of $\mathcal{A}_i$. We drop the suffix i from the observations $\mathbf{v}_i[j]$ and the initial state $\mathbf{v}_{i,0}$ in this section. The trace β_i records the state of $\mathcal{A}_i$ at certain time instants. The exact real-time of an observation $\mathbf{v}$ in β_i is unknown, but we compute the observation interval $[L(\mathbf{v}), U(\mathbf{v})]$ that satisfies Lemma 25. In what follows, we present a symbolic approximation algorithm that takes three inputs: (a) the hybrid system that models the behavior of agent i as $\mathcal{A}_i$, (b) a trace of $\mathcal{A}_i$ with length k and duration T, $\beta_i = \mathbf{v}[1], \ldots, \mathbf{v}[k]$, $\mathbf{v}_0$ and (c) the observation intervals for the observations in β_i. The algorithm computes a symbolic expression $reach_{\beta_i}(t)$, such that for each $t \in [0, dur(\beta_i)]$, $reach_{\beta_i}(t)$ overapproximates the states reached by $\mathcal{A}_i$ through the executions in $\mathsf{TraceInv}_{\beta_i}$.

Example 9: Our running example consists of three mobile robots in the plane, each attempting to move their x-coordinate to equal their numeric identifiers (e.g., eventually $x_1 = 1$, $x_2 = 2$, and $x_3 = 3$), without caring about the values of their y-coordinates. To accomplish this goal, each robot has several modes, each with different dynamics corresponding to a different controller. For ease of exposition for the example, we assume (a) the observation intervals are disjoint for each robot, (b) that mode switches only occur at the time an observation is recorded, and (c) that the robots have rectangular dynamics (that is, $\dot{x}_i \in [a_m, b_m]$ for some constants $a_m \leq b_m$ in each mode m). These assumptions are in contrast to the general case defined in Equation (6.3) and computed in the actual experiments.

clk	x	$\dot{x} \in [a, b]$	μ_l	μ_u
1.03	3.20	$[-1.6, -0.9]$	0.100	0.100
1.67	2.39	$[-1.6, -0.9]$	0.126	0.108
2.42	1.47	$[-1, 1]$	0.100	0.144
3.45	1.59	$[0.5, 0.5]$	0.144	0.100
4.08	1.86	$[-1.6, -0.9]$	0.126	0.100

Table 6.1: Example trace for robot 2 corresponding to Figure 6.1

A trace consisting of five valuations of a robot is shown in Table 6.1, and this corresponds to the middle robot 2 (in light green) in Figure 6.1. In the trace, clk is the local time recorded in the trace, x is the robot's x-coordinate in the corresponding interval $[clk - \mu_l, clk + \mu_u]$, and $\dot{x} \in [a, b]$ indicates the dynamics with which x is evolving. □

Before introducing the algorithm, we define several building-block expressions that are computed from static analysis of $\mathcal{A}_i$'s specification. For an expression S involving the variables of $\mathcal{A}_i$, $[\![S]\!] \subseteq Q_i$ denotes the subset of states that satisfy S. We now formally define the notation for $Pre(,)$ and $Post(,)$ functions.

Definition 32 *Given a hybrid system that models the behavior of agent i as $\mathcal{A}_i$, a set of states $S \subseteq val(V_i)$, and time $t > 0$. $Post_{\mathcal{A}_i}(S, t)$ is an expression involving $V_i \cup \{t\}$, such that for any execution σ of $\mathcal{A}_i$ with $\sigma.\mathsf{fstate} \in [\![S]\!]$, $\sigma(t) \in [\![Post_{\mathcal{A}_i}(S, t)]\!]$. The $Post_{\mathcal{A}_i}(S, t)$ expression is said to be* exact *if, for every $s \in [\![Post_{\mathcal{A}_i}(S, t)]\!]$, there exists an execution σ with $\sigma(0) \in S$ and $\sigma(t) = s$. $Pre_{\mathcal{A}_i}(S, t)$ is an expression involving $V \cup \{t\}$ such that for any execution σ of $\mathcal{A}_i$ with $\sigma(t) \in [\![S]\!]$, $\sigma.\mathsf{fstate} \in [\![Pre_{\mathcal{A}_i}(S, t)]\!]$. Exact $Pre_{\mathcal{A}_i}(S, t)$ expressions are defined analogously.*

One way to numerical compute $Post(,)$ and $Pre(,)$ is to use dynamic analysis techniques presented in Chapters 3 and 4. As we consider only rectangular hybrid automata models for modeling the interaction of agent i with the physical environment, several works in literature present techniques for computing these symbolic expressions (see Section 1.3).

For any time $t \in [0, dur(\beta_i)]$, $before(t)$ returns the $\mathbf{v} \in [\![\beta_i]\!]$ with the the largest $U(v) < t$, or $\mathbf{v}_0$ if no such $\mathbf{v}$ exists, and $after(t)$ returns the $\mathbf{v} \in [\![\beta_i]\!]$ with the the smallest $L(v) \geq t$, or it returns a special symbol $\top$ indicating that there is no such observation.

input : $\beta_i = \langle \mathbf{v}[1], \ldots, \mathbf{v}[k] \rangle$, $\mathbf{v}_0$
output: $reach_{\beta_i}(t)$
1 $TSeq \leftarrow Sort(\{L(\mathbf{v}), U(\mathbf{v}) \mid \mathbf{v} \in [\![\beta_i]\!]\})$;
2 $(l, u) \leftarrow (t_j, t_{j+1})$ `// such that` $t \in [l, u)$ `and` $t_j, t_{j+1} \in TSeq$;
3 $Seq \leftarrow \langle before(l), obs([l, u)), after(u) \rangle$;
4 $reach_{\beta_i}(t) \leftarrow Pred(Seq, [l, u))$;

Algorithm 6.1: Algorithm for computing the predicate for the set of states reached in the interval t_j, t_{j+1}.

In Line 5, $obs([l,u))$ is the sequence of observations, such that for each $\mathbf{v}$ in the sequence $[L(\mathbf{v}), U(\mathbf{v})] \cap [l,u) \neq \emptyset$. Seq prepends $before(l)$ to this sequence and appends $after(u)$ only if it is not $\top$. The predicate $Pred(Seq, [l,u))$ takes as input such a sequence and the time interval $[l,u)$, and computes the symbolic expression for the set of states reached in the interval $[l,u)$. If $after(u) \neq \top$, then $Pred(\langle s_1, \ldots, s_m \rangle, [l,u), t)$ is defined as $(l \leq t < u) \wedge \exists t_{s_1} < t_{s_2} < \ldots < t_{s_m}$

$$\begin{aligned} \bigwedge_{j=1}^{m} (L(s_j) \leq t_{s_j} \leq U(s_j)) \quad &\wedge \quad \bigwedge_{j=1}^{m-1} (t_{s_j} \leq t \leq t_{s_{j+1}} \Rightarrow \\ (Post(s_j, t - t_{s_j}) \quad &\wedge \quad Pre(s_{j+1}, t_{s_{j+1}} - t))). \end{aligned} \tag{6.3}$$

If $after(u) = \top$, then it is defined as $(l \leq t < u) \wedge \exists t_{s_1} < t_{s_2} < \ldots < t_{s_m}$

$$\begin{aligned} \bigwedge_{j=1}^{m} (L(s_j) \leq t_{s_j} \leq U(s_j)) \quad &\wedge \quad \bigwedge_{j=1}^{m-1} (t_{s_j} \leq t \leq t_{s_{j+1}} \Rightarrow \\ (Post(s_j, t - t_{s_j}) \quad &\wedge \quad Pre(s_{j+1}, t_{s_{j+1}} - t))) \wedge \\ (t \geq t_{s_m} \quad &\Rightarrow \quad (Post(s_m, t - t_{s_m}))). \end{aligned} \tag{6.4}$$

The next lemma states that $reach_{\beta_i}(t)$ contains all states that are reachable at time t through any execution in $\mathsf{TraceInv}_{\beta_i}$.

Lemma 26 *For any execution $\sigma \in \mathsf{TraceInv}_{\beta_i}$ of $\mathcal{A}_i$ and any $t \in [0, dur(\beta_i))$, $\sigma(t) \in [\![reach_{\beta_i}(t)]\!]$.*

Proof: Let us fix an execution σ of $\mathcal{A}_i$ in $\mathsf{TraceInv}_{\beta_i}$ and an instant of time $t \in [0, dur(\beta_i))$. Let $TSeq$ be the sorted sequence of unique observation interval endpoints for the observations in β as computed in Line 3. In Line 4, t uniquely defines the interval $[l,u)$ such that $t \in [t_i, t_{i+1})$. Let $Seq = \langle s_1, s_2, \ldots, s_m \rangle$ be the sequence of observations computed in Line 5. We will show that $\sigma(t) \in [\![Pred(Seq, [l,u))]\!]$.

First we consider the case where $s_m = after(u) \neq \top$ and $Pred$ is defined by Equation (6.3). From Lemma 25 (1), it follows that for each $j \in \{1, \ldots, m\}$, there exists

$$\exists\, t_{s_j} \in [L(s_i), U(s_i)], \alpha(t_{s_j}) = s_j. \tag{6.5}$$

From Definition 8, it also follows that for each $j \in \{1, \ldots, m-1\}$, $t_{s_j} \leq t_{s_{j+1}}$. From the construction of Seq, we have that s_1 happened before l, s_m happened after u, and $s_2, \ldots, s_{m-1}$ happened (in that order) within the time interval $[l,u)$. Therefore,

t must be within the $[t_{s_j}, t_{s_{j+1}})$ for exactly one $j \in \{1, \ldots, m\}$. We fix the j such that $t \in [t_{s_j}, t_{s_{j+1}})$. Using Equation (6.5), we have $\sigma(t_{s_j}) = s_j$ and $\sigma(t_{s_{j+1}}) = s_{j+1}$. Thus, $\sigma(t) \in [\![Post(s_j, t - t_{s_j})]\!]$ and $\sigma(t) \in [\![Pre(s_{j+1}, t_{s_{j+1}} - t)]\!]$.

For the case where $after(u) = \top$ $s_m \neq after(u)$ and $Pred$ is defined by Equation (6.4). The proof is identical to the previous case with the exception of the situation where $t \geq t_{s_m}$. In this case, there is no observation after t in σ, but there is only an observation s_m before t. Thus, $\sigma(t) \in Post(s_m, t - t_{s_m})$. ■

Example 10: In this part of the running example, we illustrate how the set of $Pre(,)$ and $Post(,)$ can be used for computing the reachable states between two consecutive observations. These sets of reachable states between observations are shown for the x-coordinate of the three robots plotted against real-time in Figure 6.1. The observations (x values) and the corresponding observation intervals are visualized

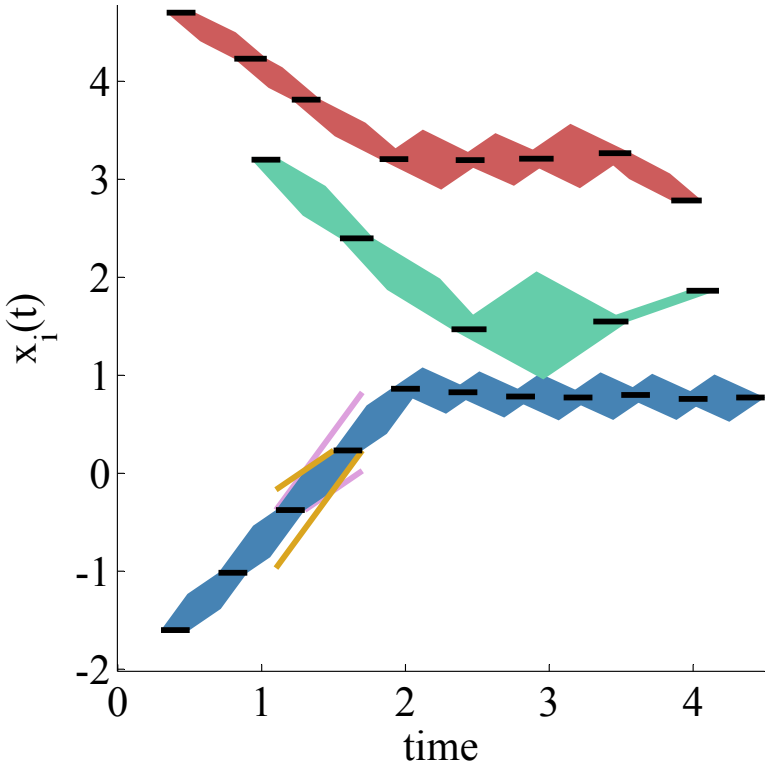


Figure 6.1: Example timeline and reach set computation between observations for the x-coordinates of three mobile robots in the plane.

as the black lines. From observation 3, the $Post(,)$ (the points between the purple lines) is computed up to the last time observation 4 could have occurred. Likewise, from observation 4, the $Pre(,)$ (the points between the orange lines) is computed up

to the earliest time observation 3 could have occurred. Instantiating (Equation (6.3)) for this case, we have:

$$\begin{aligned}\exists t_0 < t_m.\mathbf{v}[3].clk - \mathbf{v}[3].\sigma_l \leq t_0 \leq \mathbf{v}[3].clk + \mathbf{v}[3].\sigma_u \wedge \\ \mathbf{v}[4].clk - \mathbf{v}[4].\sigma_l \leq t_m \leq \mathbf{v}[4].clk + \mathbf{v}[4].\sigma_u \Rightarrow \\ Post(\mathbf{v}[3].x, t - t_0) \wedge Pre(\mathbf{v}[4].x, t_m - t).\end{aligned}$$

Under the assumption that the robots' positions evolve with rectangular dynamics, the $Post(\mathbf{v}[3].x, t - t_0)$ expression for robot 1 starting from observation 3 is:

$$\exists t.\mathbf{v}[3].x + \mathbf{v}[3].a(t - t_0) \leq x(t) \leq \mathbf{v}[3].x + \mathbf{v}[3].b(t - t_0).$$

As we have assumed every time a mode switch occurs a state observation is added to the trace, these $Pre(,)$ and $Post(,)$ expressions are exact. The final expression for $reach_{\beta_i}(t)$ for t between observations 3 and 4 computed by eliminating quantifiers. For a general automaton $\mathcal{A}_i$, these expressions are computed from the static analysis of the specification $\mathcal{A}_i$. □

Under the additional assumption that the observation intervals in β_i are exact and disjoint and that the $Pre(,)$ and $Post(,)$ computations are exact, we show that every state in $[\![reach_{\beta_i}(t)]\!]$ is reachable by some execution in $\mathsf{TraceInv}_{\beta_i}$ at time t.

Lemma 27 *Suppose $\mathcal{A}_i$ permits exact computation of $Post(,)$ and $Pre(,)$ expressions and for the given trace β, all the observation intervals are exact and disjoint. For any $t \in [0, dur(\beta))$ and any state $s \in [\![reach_\beta(t)]\!]$, there exists an execution $\sigma \in \mathsf{TraceInv}_\beta$ with $\sigma(t) = s$.*

Proof: Let us fix $t \in [0, dur(\beta_i))$, and a state $s \in [\![reach_{\beta_i}(t)]\!]$. Let $TSeq$ and Seq be the time and observation sequences computed in Lines 3 and 5 in Algorithm 6.1. Let $[l, u)$ be the interval in $Tseq$ which contains t. Let $before(l) = \mathbf{v}_1$ and $after(u) = \mathbf{v}_2$. Note that since the observation intervals are disjoint, for all but the last interval $after(u) \neq \top$. Based on the disjointedness of the observation intervals, there are two possible cases to consider:

Case 1: There is no observation with the observation interval is $[l, u]$. Therefore, $[l, u]$ cannot be the last interval and $after(u) \neq \top$. Since $s \in [\![reach_\beta(t)]\!]$, from Equation (6.3), it follows that there exists $t_1 \in [L(\mathbf{v}_1), U(\mathbf{v}_1)]$ and $t_2 \in [L(\mathbf{v}_2), U(\mathbf{v}_2)]$, such that $s \in [\![Post(\mathbf{v}_1, t - t_1)]\!]$ and $s \in [\![Pre(\mathbf{v}_2, t_2 - t)]\!]$. Since the $Pre(,)$ and $Post(,)$

computations are exact, it follows that there exists an execution fragment σ' with $\sigma'(0) = \mathbf{v}_1$, $\sigma'(t - t_1) = s$ and $\sigma'(t_2 - t_1) = \mathbf{v}_2$. Furthermore, since $[L(\mathbf{v}_1), U(\mathbf{v}_1)]$ is an exact observation interval, there exists an execution $\sigma_1 \in \mathsf{TraceInv}_{\beta_i'}$, where β_i' is the prefix of β_i up to $\mathbf{v}_1$, such that $\sigma_1.\mathsf{ltime} = t_1$ and $\sigma_1.\mathsf{lstate} = \mathbf{v}_1$. For the same reason, there exists another execution $\sigma_2 \in \mathsf{TraceInv}_{\beta_i''}$, where β_i'' is the suffix of β_i starting from $\mathbf{v}_2$. Concatenating we define $\sigma \triangleq \sigma_1\sigma'\sigma_2 \in \mathsf{TraceInv}_{\beta_i}$ and satisfies the requirement $\sigma(t) = s$.

Case 2: There exists a single observation $\mathbf{v} \in [\![\beta_i]\!]$ for which $L(\mathbf{v}) = l$ and $U(\mathbf{v}) = u$. Since $s \in [\![reach_{\beta_i}(t)]\!]$, from Equation (6.3), we know that there exists $t_1 \in [L(\mathbf{v}_1), U(\mathbf{v}_1)]$, $t_{\mathbf{v}} \in [l, u]$, and $t_2 \in [L(\mathbf{v}_2), U(\mathbf{v}_2)]$, such that one of the following conditions must hold (1) $t_1 \leq t \leq t_v$, $s \in [\![Post(\mathbf{v}_1, t - t_1)]\!]$, and $s \in [\![Pre(\mathbf{v}, t_{\mathbf{v}} - t)]\!]$, or (2) $t_{\mathbf{v}} \leq t \leq t_2$, $s \in [\![Post(\mathbf{v}, t - t_{\mathbf{v}})]\!]$, and $s \in [\![Pre(\mathbf{v}_2, t_2 - t)]\!]$. We consider the first sub case. Since the $Pre(,)$ and $Post(,)$ computations are exact, it follows that there exists an execution fragment σ' with $\sigma'(0) = \mathbf{v}_1$, $\sigma'(t - t_1) = s$ and $\sigma'(t_{\mathbf{v}} - t_1) = \mathbf{v}$. Furthermore, since $[L(\mathbf{v}_1), U(\mathbf{v}_1)]$ is an exact observation interval, there exists an execution $\sigma_1 \in \mathsf{TraceInv}_{\beta_i'}$, where β_i' is the prefix of β_i up to $\mathbf{v}_1$, such that $\sigma_1.\mathsf{ltime} = t_1$ and $\sigma_1.\mathsf{lstate} = \mathbf{v}_1$. For the same reason, there exists another execution $\sigma_2 \in \mathsf{TraceInv}_{\beta_i''}$, where β_i'' is the suffix of β_i starting from $\mathbf{v}$. Concatenating these executions, we define $\sigma = \sigma_1\sigma'\sigma_2 \in \mathsf{TraceInv}_{\beta}$ and satisfies the requirement $\sigma(t) = s$.

For the second sub case, the reasoning is similar except that we have to consider the possibility that $after(u)$ is $\top$, that is, $\mathbf{v}_2$ is undefined. In that situation, the execution prefix to $\mathbf{v}$ (at time $t_{\mathbf{v}}$) is concatenated with any execution fragment that starts from $\mathbf{v}$ and hits s after $t - t_{\mathbf{v}}$ time. ■

Summary of Sections 6.2.1 and 6.2.2 We have presented a procedure for computing observation intervals for each observation in a trace $\beta = \{\beta_i\}$ of a distributed system $\mathcal{A} = \|_i\mathcal{A}_i$. Furthermore, from these observation intervals and static analysis of each individual automaton $\mathcal{A}_i$, we can symbolically compute an expression $reach_{\beta_i}(t)$ that overapproximates the set of states of $\mathcal{A}_i$ that can be reached at a given time $t \in [0, dur(\beta_i))$. Under additional assumptions about the accuracy of the observation intervals and the static analysis, $reach_{\beta_i}(t)$ is the exact set of states reached at time t by executions of $\mathcal{A}_i$ that correspond to the trace β_i.

6.2.3 Global Predicate Detection

A *global property (or predicate)* is an expression P involving the variables $V = \cup_{i \in I} V_i$ of all the automata in the distributed system. Given a system $\mathcal{A} = \|_{i \in I} \mathcal{A}_i$, a trace $\beta = \{\beta_i\}_{i \in I}$, and a global predicate P, we can make the following two types of queries to an SMT-solver:

$$(eventually)\ \exists t \in [0, dur(\beta)) : (\wedge_{i \in I} reach_{\beta_i}(t)) \Rightarrow P \tag{6.6}$$

$$(always)\ \forall t \in [0, dur(\beta)) : (\wedge_{i \in I} reach_{\beta_i}(t)) \Rightarrow P. \tag{6.7}$$

If the eventuality-query has a satisfying t, then from Lemma 26 it follows that all executions of the system $\mathcal{A}$ corresponding to $\mathsf{TraceInv}_\beta$, satisfy P at time t. For example, if P captures an unsafe state of $\mathcal{A}$, then this implies that all the executions corresponding to β become unsafe at t, and therefore, points to a bug in $\mathcal{A}$. The negation of the always-query can be posed as an existential problem:

$$(always)\ \exists t \in [0, dur(\beta)) : (\wedge_{i \in I} reach_{\beta_i}(t)) \wedge \neg P. \tag{6.8}$$

If this query is unsatisfiable, then from Lemma 26 it follows that all executions of the system $\mathcal{A}$ corresponding to $\mathsf{TraceInv}_\beta$ satisfy P at all times in the interval $[0, dur(\beta))$. For example, if P captures a safety invariant, then this implies that all the executions corresponding to β are safe over this interval, and therefore, gives a proof of bounded safety. On the other hand, if Equation (6.8) has a satisfying solution t, *and* if β and $\mathcal{A}$ satisfy the requirements of Lemma 27 (completeness), then we can infer that there exists an actual execution corresponding to β that violates P at time t.

These queries involve real-arithmetic and their decidability and tractability depend on the nature of the $Pre(,)$, $Post(,)$ sub-formulas and the predicate P. For example, if the $\mathcal{A}_i$'s are hybrid automata with trajectories described by polynomial functions of time, and P is a polynomial in the state variables, then this problem is decidable. Our current strategy for solving these queries is to ship them to the SMT-solver Z3 [48].

Example 11: For global predicate detection in the running example, Figure 6.1 allows us to conclude that the x-coordinates of robots 2 (light green) and 3 (red) never come closer together than about 0.25 units from one another. However, we cannot reach the same conclusion for robots 1 (blue) and 2 (light green). In fact, since we assumed in the example that message delivery intervals are disjoint and the robots' dynamics are rectangular, then the assumptions of Lemma 27 are satisfied,

so we can conclude there is a real execution where the x-coordinates of robots 1 and 2 coincide. □

6.3 Experimental Evaluation

In this section, we describe the platform used for developing distributed hybrid systems and the type of applications used to evaluate our approach for verification of global predicates. We also demonstrate the sensitivity of the algorithm with respect to the different parameters computed both statically and dynamically. Further, we demonstrate the applicability of the algorithm by presenting a deadlock detection example.

6.3.1 Distributed Applications on StarL Android Platform

All the traces used in experimental evaluation of our technique are generated from distributed programs written for Android devices [1] controlling mobile robots. In our laboratory, we have implemented a Java-based framework, called StarL [170, 114], on top of the Android operating system. StarL has implementations of (a) unicast and multicast protocols, (b) a library of high-level functions for accomplishing common distributed tasks (for example, mutual exclusion, leader election, synchronization, etc.), (c) functions for accessing hardware sensors on the Android device (such as GPS sensors, accelerometers, and cameras), and (d) motion-control functions (used in applications where each Android device is paired via Bluetooth with a mobile robot, such as an iRobot Create). For the purposes of the experiments in this chapter, StarL provides a convenient abstraction for programming a swarm of mobile robots with many sensors, in Java, and over a standard operating system. Over the past year, we have used StarL to implement several applications such as peer-to-peer chat, geocasting, coordinated distributed search, flocking, and distributed traffic control [170]. In almost all cases, our initial implementations violated some of the expected safety and progress properties of the application. Often the violations were not reproducible, which made it difficult and time consuming to find their root cause. This experience, at least in part, motivated the work reported in this chapter of thesis.

In this thesis, we use traces generated from three StarL applications:

1. *Waypoint tracking (WT)*: Each robot is assigned a sequence of waypoints in the plane and they traverse these waypoints. The robots do not employ any collision avoidance and do not exchange messages. This simple building block is used in several other applications. The two global predicates of interest are *Separation* and *Collinear*. A set of robots satisfy *Separation*(d) at a given time if the minimum distance between all pairs is at least d. They satisfy *Collinear*(ϵ) if there exists a straight line passing within ϵ distance of their positions.

2. *Geocasting (GC)*: Each robot follows a sequence of waypoints as in WT and some of the robots geocast a message m to a circular area C in the plane. The key property of interest is that a robot receives this message iff it is located within C during the time interval $[t + a, t + b]$, where t is the time at which the message was sent. We call this property *Georeceive*(a, b, C).

3. *Distributed Traffic Control (DTC)*. Each robot has to traverse a specified sequence of segments as in WT, however, when two robots attempt to traverse intersecting segments simultaneously (representing traffic intersections), then one of them has to acquire a lock on that intersection. The robot succeeding in obtaining the lock traverses its segment, while the others wait. Each intersection lock is managed by the robots executing a distributed mutual exclusion algorithm. The important properties here are separation and deadlock freedom.

One important component of StarL is the *trace writer* that is called by functions and applications to record observations. Observations after each message send and receive are recorded by default. The frequency of other observations and the amount of state information that is recorded at each observation are controlled by the observation recording functions. In our experiments, we use two types of traces. First, for systems with four or fewer agents, the traces are recorded from the same number of Android devices executing application code. In these experiments, the position of each agent (wherever applicable) is obtained from a vision-based indoor positioning system. Secondly, for generating traces for systems with dozens of agents we have created a StarL *discrete event simulator* that can execute many instances of a StarL application. Specifically, the simulator executes the actual StarL application code combined with harness functions that substitute Android's hardware specific functions and physical sensors, such as the WiFi interface, the local clock, and the location sensors. For example, to generate traces for 20 robots, each robot is simulated on a PC by a process (with several threads). Each process obtains the location of the

simulated robot from a harness function that mimics the motion of the actual robot (in our case, the iRobot Create) in the simulated environment. Simulated robots can turn in place, move in straight lines, and travel in circular arcs. In a similar fashion, other harness functions provide the agent process access to simulated communication channels over which messages can be delayed and dropped. For the experiments performed, we also record the "true execution" that generates each trace. For the traces from the deployed system, the true execution is obtained from a log of the actual positions and states of all the robots that is updated at a high sampling rate (higher than the observation frequency) as is maintained in a central PC with an accurate real-time clock. For the simulation traces, the simulator itself records snapshots of the entire system as a record of the true execution.

Since StarL is written in Java for Android, real-time issues related to memory management and garbage collection are a potential issue. However, recall that for the soundness of our analysis (26), the only real-time requirement (from 31) is the following. If an observation $\mathbf{v}_i[j].clk$ appears in the trace β with sync accuracy $\mu > 0$, then the state $\mathbf{v}_i$ must have been visited within the real time interval $[ts - \mu, ts + \mu]$. Here ts is the time stamp derived from the (possibly inaccurate) local clock clk_i and μ is the synchronization accuracy. Similar assumptions are made for trace observations about messages sent and received.

When a *trace writer* instruction is executed in the Java program it first (1) creates the $\mathbf{v}_i[j]$ entry in memory (based on the current values of the program variables and local clock) and then (2) issues an instruction to write this observation to the trace stored on disk. Nondeterministic delays (such as the garbage collector starting, disk write delays, etc.) may indeed appear between steps (1) and (2), but these delays only affect when the $\mathbf{v}_i[j]$ observation is written to the trace file, and does violate the above assumption (31).

6.3.2 Scalability

The first set of experimental results serve to illustrate the scalability of the proposed approach. We have collected a large number of traces from the three StarL applications, WT, GC, and LP, with 4-20 participating agents. The automaton model of the application is obtained from the Java code, and includes details of the physical motion model. Since the robots have simple turn-and-move dynamics, their motion is modeled using simple rectangular differential inclusions (e.g., $a \leq \dot{x} \leq b$). Our

tool attempts to automatically check the relevant properties by first constructing appropriate formulas and then making the always or eventually queries to Z3.

Table 6.2 shows the total time and memory usage for checking global properties for typical traces. For each property, we report the performance numbers for $4, 12$, and 20 robots participating in the application. The size of the formula used to verifying the global predicate (i.e., the formula for global predicate detection) is given in the last column. In these traces, one observation is recorded roughly every 100 milliseconds, so a trace of length 100 approximately corresponds to 10 seconds in real-time. For all these traces with up to 20 robots, the properties are checked within a couple of minutes, and in most cases within a few seconds! These figures suggest that the approach can scale and in some cases may even be suitable for checking properties of distributed cyber-physical systems in real-time.

For the *Separation* property, we check that all the participating robots *always* maintain a minimum separation of d centimeters. This requires a pair-wise comparison of distances. Note that for the same trace with 12 (and 20) robots, $Separation(d = 2.5)$ holds, whereas $Separation(d = 10)$ does not. We examined the true execution for these traces (the fine-grained simulation log), and observed that, indeed, for a duration of about $200ms$ robots 6 and 11 came within 10cm of each other.

For the *Collinear* property, we check that all the robots *never* form a rough straight line (within ϵ distance of an ideal line). This property is violated by the robots, which implies that there exists a time, when, in some execution corresponding to the recorded trace, all the robots form a rough line. The Collinear property has more nonlinear terms than the Separation property and hence takes more time to verify (note, these traces have only 10 observations).

For the *Georeceive* property, we check that for any time t if a message m is geocast at t, then a robot receives m if and only it is located within a specified circle C during the interval $[t, t + 100ms]$. Hence robot i will receive a message if and only if, for *some* $t' \in [t, t + 100ms]$, the robot lies in C, and it does not receive the message if and only if, for the *entire* duration $[t, t + 100ms]$, it is outside the region. Checking whether the position of a robot is within the circle C at a given time involves checking satisfiability of formulas involving real-valued polynomials. In theory, this problem is decidable, but Z3 does not guarantee completeness. For some of the traces, Z3 returns "Unknown" even when the predicate actually holds. Checking *Georeceive* is much faster than Separation because it only involves checking the position of individual robots over short time intervals, as opposed to the distance between all pairs over the

Property	Num. agents	Trace len.	Result	Time (sec)	Memory (Mb)	Formula Size (Kb)
Always	4	100	Yes	1.6	3.07	3.9
Separation	12	100	Yes	14.1	8.66	14.9
($d = 25$)	20	100	Yes	81.1	18.6	31.6
Always	4	100	Yes	1.6	3.07	3.9
Separation	12	100	No	14.1	8.66	14.9
($d = 10$)	20	100	No	81.1	18.6	31.6
Always Not	4	10	No	4.66	5.83	3.7
Collinear	12	10	No	12.21	12.91	10.9
($\epsilon = 100$)	20	10	No	25.68	25.66	18.4
Always	4	100	Yes	1.28	1.24	3.2
Georeceive	12	100	Yes	1.77	3.67	9.5
	20	100	Yes	1.91	8.35	16

Table 6.2: Running time and memory requirements of distributed trace analysis.

Num. agents	Sampling Time 75 ms.	Sampling Time 150 ms.	Sampling Time 250 ms.	Sampling Time 500 ms.
8	92.57 sec	48.26 sec	22.07 sec	9.58 sec
12	4 min 6 sec	114.23 sec	34.16 sec	16.43 sec
16	9 min 58 sec	3 min 52 sec	49.36 sec	24.18 sec
20	20 min 26 sec	8 min 24 sec	67.82 sec	34.94 sec

Table 6.3: Running time (in seconds) for verification of separation property for different sampling periods.

entire duration.

The influence of sampling time (i.e. the time between two consecutive observations of a robot) over the total time taken for verification is shown in Table 6.3. The table reports the time taken to verify the $Separation(d = 10)$ property for distributed traces of real time 5 minutes. The verification time with sampling period of 500 ms are always lesser than a minute for the traces of 5 mins even with 20 robots. This suggests a promising approach for integrating this framework into run-time verification of distributed hybrid systems. It can be clearly observed from Table 6.3 that verification time is inversely proportional to the sampling time. This is because, as the sampling time increases, the number of observation intervals decrease and hence the lower is the verification time. Another key observation is that because of the soundness of the analysis, the outputs with different sampling periods were always consistent. Hence, one could adaptively change the sampling time based on the application in order to improve the scalability of the approach with the benifits of sound analysis.

6.3.3 Precision of Analysis

The precision of our analysis algorithm depends both on the precision of the static analysis (i.e., $Pre(,)$ and $Post(,)$ computations) and the accuracy of the dynamic information, namely the observation intervals inferred from the trace and bounds on variation of velocity. In this section, we discuss how different levels of precision about the static and the dynamic analysis can result in different answers. We consider traces for WT and the GC with 4 robots. The table consists of three parts. The first part shows the outcome of the analysis for *Separation* with $d = 10cm$. When the uncertainty in the dynamically computed observation interval (OI) increases from local clock $\pm 10ms$ to $\pm 20ms$, Separation gets violated. With increased uncertainty in the dynamically calculated values of the observation intervals, we get a more conservative overapproximation of the set of executions which violates Separation. Analogously, when the velocity bound (VB) increases from ± 20 to $\pm 50cm/s$, for the same observation interval, the property gets violated. Of course, whether there exists a real execution that violates the property depends on the tightness of the observation interval and the velocity bounds.

The second part of Table 6.4 shows the outcome of analysis for *Georeceive* with varying the minimum message delays (i.e. a) from $0ms$ to $50ms$ in $Georeceive(a, 100ms, C)$ while keeping the observation intervals constant at $\pm 5ms$. The final part of the table varies the observation interval from $\pm 5ms$ to $\pm 20ms$ while keeping the minimum message delay fixed at 0. The key observation here is that increasing either the static (model) or the dynamical (execution) uncertainties gives more conservative overapproximations.

6.3.4 Experience with Deadlock Detection

As described earlier, the LP application traverses segments as in WT, however when two robots attempt to traverse intersecting segments simultaneously, then one of them acquires a lock on the intersection. It is not hard to imagine that a deadlock might occur when a cycle is formed with each robot holding the lock on one intersection and waiting for the lock on the next. We used our procedure to check for the global predicate *always (no-deadlock)* . If the property is violated, then there is a potential execution that deadlocks[2].

[2]The feasibility of this execution will depend on whether the analysis is complete in this case.

Separation	VB = $\pm 0 cm/s$	VB = $\pm 20 cm/s$	VB = $\pm 50 cm/s$
OI = $\pm 5 ms$	yes	yes	no
OI = $\pm 10 ms$	yes	no	no
OI = $\pm 20 ms$	no	no	no
Georeceive	VB = $\pm 0 cm/s$	VB = $\pm 20 cm/s$	VB = $\pm 50 cm/s$
delay = $0 ms$	yes	yes	yes
delay = $20 ms$	yes	yes	no
delay = $50 ms$	no	no	no
Georeceive	VB = $\pm 0 cm/s$	VB = $\pm 20 cm/s$	VB = $\pm 50 cm/s$
OI = $\pm 5 ms$	yes	yes	yes
OI = $\pm 10 ms$	yes	yes	no
OI = $\pm 20 ms$	yes	no	no

Table 6.4: The same trace with different levels of precision in static and dynamic analyses yield different conclusions.

Our tool was able to automatically detect distributed deadlocks in two traces with 4 robots. In the first trace, the algorithm detected that the property is violated because robot 1 had obtained the token on all the intersections and it had halted because of its proximity to robot 2. Then, since all the tokens were taken by robot 1, all the other robots (i.e., robots 2, 3, and 4) were all waiting for robot 1. Examination of the true execution revealed that this was indeed the case. In a second trace, the tool detected a deadlock between robots 3 and 4. In this case, robot 3 had finished its traversal of waypoints and robot 4 could not reach its intended intersection waypoint because it was occupied by 3. These automatically discovered counterexamples illustrate that the procedure can be useful for finding corner cases that are commonly missed by programmers.

6.4 Conclusions And Related Work

In this chapter, we proposed a procedure for analyzing traces of distributed cyber-physical systems to infer global properties. The procedure combines static analysis of programs and analysis of traces generated at runtime. We showed that the procedure is sound, and presented additional conditions that ensure it is also complete. We have implemented the procedure in an automated software tool using the Z3 SMT solver for satisfiability checks. Our tool can verify interesting properties—such as correct geographic delivery in geocast and minimum separation—for 20 robots, in seconds.

The problem of detecting a global predicate from traces of purely asynchronous agents has been well-studied in the distributed computing literature [23, 72]. In these systems, the events (observations) are in no way associated with real-time and are related by Lamport's *happens-before* relation [110] alone. This relation induces a partial ordering on the events. The algorithms for detection of general global predicates rely on traversing the lattice of all possible interleavings of the asynchronous events (observations), and therefore, are exponential in the size of the trace [43, 125]. Thus, much of the research in this area has focused on identifying subclasses of predicates admitting efficient detection, such as conjunctions of local predicates [75, 92], *linear* predicates [73], and *regular* predicates [74]. For example, in [36], Chandy and Lamport present an algorithm for detecting *stable* predicates by taking global snapshots. For reducing the space of global states to be examined, techniques such as *symbolic* methods [157], *computational slicing* [129, 155] and *partial order* methods [158] have been investigated.

Though the same problem is studied in this chapter, our assumptions on agent behavior is motivated by the capabilities of current devices used for embedded and distributed computation, and these assumptions translate to solutions that use different strategies from those studied earlier. For example, our procedure uses static analysis for computing overapproximations of the reachable set between the observations (or events). This leads to a combination of static and dynamic analysis that has not been used earlier in the context of distributed systems.

Despite the challenges, in all of these cases, we would ideally like to build systems that guarantee certain safety and real-time properties. While there is a large body of work on performing verification for distributed cyber-physical systems [131, 119, 97], these works generally focus on model verification or static analysis. In contrast, in this work, we develop a method that can be used to help programmers debug during the designing phase for such distributed systems, and also that can be used as a means of performing runtime verification. Since traces correspond to actual executions, properties are verified in spite of non-idealities, such as imperfect local clock synchronization. Such non-idealities are essential to consider in realistic systems, and are not usually considered in model verification studies.

Chapter 7

Conclusions and Future Work

In this thesis we have presented new dynamic analysis techniques for verification of Cyber-Physical Systems. The dynamic analysis techniques presented assume that the hybrid system models of CPS are annotated with discrepancy functions. These discrepancy functions capture the continuity property that "executions starting close by stay close by''. We have shown that discrepancy functions generalize routinely used proof certificates in control theory for establishing convergence and divergence of trajectories. Furthermore, we presented a new technique to compute discrepancy function for time varying linear dynamical systems.

Using these discrepancy functions and validated simulations, we presented an algorithm that computes the reachtree for bounded time and bounded number of discrete transitions. We proved that this reachtree gives an overapproximation of the reachable set of states. The safety verification algorithm computes reachtrees of increasing precision until either the safety property is established or a violating counterexample is found. We showed that this algorithm gives theoretical guarantees such as soundness and relative completeness. Given that the safety verification problem of hybrid systems is undecidable, such theoretical guarantees are the strongest one can expect from any automatic procedure.

We implemented the dynamic analysis technique in a tool called Compare-Execute-Check-Engine (C2E2) and compared the verification results with state of the art tools. Experimental results show that C2E2 outperforms its competitors and shows promise for scalability. We presented the architecture of the tool, the input format of models, and user experience of the tool.

To demonstrate the applicability of the dynamic analysis technique, we performed two challenging case studies as a part of this thesis. The first case study was about alerting mechanism ALAS in parallel landing protocol and the second case study was about powertrain control systems. To verify the key property of alerting mechanism, we presented a dynamic analysis technique that verifies temporal precedence proper-

ties. Further, we extended the safety verification algorithm to handle the nonlinear guarantee predicates that is used in ALAS. Experimental results show that verification of alerting mechanism is scalable and C2E2 can identify scenarios such as *false alert* and *missed alert*. For verifying powertrain control systems, we encoded the specification as safety properties, implemented the algorithm for computing on-the-fly discrepancy function, and used the safety verification algorithm. This thesis is the first to *formal verify* powertrain control systems.

For distributed CPS implementations with message losses, clock skews, and sensor inaccuracies, we presented a dynamic analysis technique that infers global predicates of the system. For achieving this, we combined techniques from analysis of distributed systems and CPS verification. Experimental evaluation on several applications such as waypoint tracking, geo-cast, and distributed traffic control protocols demonstrate that the approach us promising and has the potential to be integrated into inferring global predicates in real-time.

To summarize, in this thesis we have investigated the potential of using dynamic analysis for verifying CPS. The results presented in this thesis suggest that dynamic analysis techniques are scalable and efficient without compromising on theoretical guarantees. These advantages present a wide scope for future work.

7.1 Future Work

In this section, we describe several directions for future work. We mainly classify these research directions into two groups: improvements and new avenues.

7.1.1 Improvements

In this section, we will present future research directions which are improvements of the dynamic analysis technique presented in this thesis.

Automatic discovery of discrepancy functions: The dynamic analysis technique presented requires that the hybrid system model of CPS be equipped with discrepancy functions as model annotations. While one could potentially use the proof certificates in control theory to compute discrepancy functions, for CPS modeling biological and chemical phenomenon, it is often the case that these proof certificates

cannot be computed automatically. A promising step in this direction is presented in Algorithm 3.1 where a discrepancy function for time varying linear dynamical system is computed using $n+1$ sample simulations. Computing the *tightest* possible annotation for such linear systems would provide theoretical insights into the behavior of linear dynamical systems and hence is an interesting direction of future work. Developing analogous techniques for general nonlinear dynamical systems would be a major breakthrough in verification of CPS. However, given that nonlinear dynamical systems are a more general class than linear dynamical systems, such a technique seems to be unlikely. Exploring such techniques for a restricted set of nonlinear systems would be an interesting direction of future work.

Another interesting direction for computing discrepancy functions is to apply techniques from machine learning and nonlinear real arithmetic solvers. While using nonlinear real arithmetic solvers to discover proof certificates has recently been investigated in [103, 27], we believe that applying machine learning to improve the search for a template for proof certificates would give rise to new techniques that are applicable to a large class of systems.

Battling the curse of dimensionality: The dynamic analysis technique presented relies on computing a finite δ-cover of the initial set and then computing the reachtree. Hence, as the number of real valued variables increases, the number of validated simulations to compute a reachtree would increase exponentially. Further, the refinement operations become progressively expensive as the number of dimensions increases. To battle this curse of dimensionality, we present three directions of future research.

Theoretical Lower Bounds: An interesting theoretical question to answer in dynamic analysis is the following. "Given a hybrid system (with or without discrepancy function), what is the minimum number of numerical simulations that have to be performed to compute a reachtree of ϵ accuracy." One can consider variants of the above problem such as, computing on-the-fly the minimum number of additional simulations based on the analysis of simulations generated until now. After carefully observing algorithm 3.1, we conjecture that $O(n)$ simulations could be sufficient for linear systems.

Leveraging Cloud Computing: One way to battle the curse of dimensionality is to develop parallel algorithms (or algorithms that leverage cloud computing)

and run these algorithms on supercomputers (or cloud computing platforms). Although the current dynamic analysis technique can leverage the embarrassing parallelism in generating sample simulations, subsequent computations seem to require communication among different threads. One technique to solve this problem is to leverage static analysis techniques and pre-determine the simulations that would require minimum (or zero) communication among threads.

Accelerated Counterexample Search: While falsification techniques presented in [17, 50] search for counterexamples that falsify the specification, we believe that it is possible to leverage discrepancy functions for accelerating the search for counterexamples. Such counterexample search strategies can also result in new results in theoretical lower bounds required for verification.

7.1.2 New Avenues

In this section, we will present the future avenues of research where dynamic analysis can yield promising results.

Properties: While in this thesis, we have presented techniques for safety verification using dynamic analysis, we believe that dynamic analysis can be extended to a more general set of properties. One such property of interest is *conformance checking*. Informally, given a model A of the system S, conformance checking inspects several behaviors of S, and infers whether A is indeed a faithful model of S. Another form of checking conformance is to check whether there exists a simulation relation among two models A and B. We believe that dynamic analysis techniques can be extended for checking simulation relation and in turn be applicable for developing new abstractions for hybrid systems. This could in-turn lead to new dynamic analysis based CEGAR techniques.

Generalized continuous behaviors: In this thesis, we have restricted our attention to trajectories that are deterministic and are defined as solutions of an ODE. As a direction of future work, it would be interesting to extend these techniques to systems with nondeterminism in trajectories. Delay differential equations and partial differential equations are commonly used to model a vast class of CPS. Extending dynamic analysis techniques to such systems would help understand the disadvantages of these techniques (if any).

Probabilistic systems: Stochastic hybrid systems are widely used to model biological systems, traffic patterns, and behavior of robots. Extending dynamic analysis to these systems would require developing new extensions of discrepancy functions with probabilities. We believe that such extensions would provide probabilistic guarantees that are fundamentally different from the guarantees obtained using Monte-Carlo techniques.

References

[1] Android developer's guide. http://developer.android.com/guide/index.html.

[2] Computer assisted proofs in dynamic groups (capd). http://capd.ii.uj.edu.pl/index.php.

[3] Decision logic with logic and state machines in simulink. http://www.mathworks.com/products/stateflow/.

[4] A model based design environment from mathworks. http://www.mathworks.com/products/simulink/.

[5] M. Althoff. Reachability analysis of nonlinear systems using conservative polynomialization and non-convex sets. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 173–182. ACM, 2013.

[6] M. Althoff. An introduction to CORA 2015. In *Proc. of the Workshop on Applied Verification for Continuous and Hybrid Systems*, 2015.

[7] M. Althoff, O. Stursberg, and M. Buss. Reachability analysis of nonlinear systems with uncertain parameters using conservative linearization. In *C.D.C.*, 2008.

[8] M. Althoff, O. Stursberg, and M. Buss. Computing reachable sets of hybrid systems using a combination of zonotopes and polytopes. *Nonlinear analysis: hybrid systems*, 4(2):233–249, 2010.

[9] M. Althoff, S. Yaldiz, A. Rajhans, X. Li, B. H. Krogh, and L. T. Pileggi. Formal verification of phase-locked loops using reachability analysis and continuization. In *2011 IEEE/ACM International Conference on Computer-Aided Design, ICCAD 2011, San Jose, California, USA, November 7-10, 2011*, pages 659–666, 2011.

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

[11] R. Alur, T. Dang, and F. Ivančić. Counter-example guided predicate abstraction of hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 208–223. Springer, 2003.

[12] R. Alur and D. Dill. Automata-theoretic verification of real-time systems. In *Formal Methods for Real-Time Computing*, Trends in Software, pages 55–82. John Wiley Sons Publishers, 1996.

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

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

[15] R. Alur, C. C. T. A. Henzinger, and P. H. Ho. Hybrid automata: an algorithmic approach to the specification and verification of hybrid systems. In R. L. Grossman, A. Nerode, A. P. Ravn, and H. Rischel, editors, *Hybrid Systems*, volume 736 of *LNCS*, pages 209–229. Springer-Verlag, 1993.

[16] D. Angeli. A lyapunov approach to incremental stability properties. *IEEE Trans. Automat. Contr.*, 2000.

[17] Y. Annapureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *TACAS*, 2011.

[18] E. Asarin, M. Bozga, A. Kerbrat, O. Maler, A. Pnueli, and A. Rasse. Data-structures for the verification of timed automata. In *Hybrid and Real-Time Systems*, pages 346–360. Springer, 1997.

[19] E. Asarin, T. Dang, and A. Girard. Reachability analysis of nonlinear systems using conservative approximation. In *Hybrid Systems: Computation and Control*, pages 20–35. Springer, 2003.

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

[21] E. Asarin and G. Schneider. Widening the boundary between decidable and undecidable hybrid systems*. In *CONCUR 2002Concurrency Theory*, pages 193–208. Springer, 2002.

[22] E. M. Aylward, P. A. Parrilo, and J.-J. E. Slotine. Stability and robustness analysis of nonlinear systems via contraction metrics and sos programming. *Automatica*, 2008.

[23] O. Babaoglu and M. Raynal. Consistent Global States of Distributed Systems: Fundamental Concepts and Mechanisms. In S. Mullender, editor, *Distributed Systems*, chapter 5, pages 97–145. Addison-Wesley, 1993.

[24] C. Baier, J.-P. Katoen, et al. *Principles of model checking*, volume 26202649. MIT press Cambridge, 2008.

[25] S. Bak and M. Caccamo. Computing reachability for nonlinear systems with hycreate. *Demo and Poster Session at Hybrid Systems: Computation and Control*, 2013.

[26] S. Bak, T. T. Johnson, M. Caccamo, and L. Sha. Real-time reachability for verified simplex design. In *RTSS 2014, Rome, Italy, December 2-5, 2014*, pages 138–148, 2014.

[27] A. Balkan, J. V. Deshmukh, J. Kapinski, and P. Tabuada. Simulation-guided contraction analysis. In *1st Indian Control Conference, ICC'15*, 2015.

[28] A. Balluchi, A. Casagrande, P. Collins, A. Ferrari, T. Villa, and A. Sangiovanni-Vincentelli. Ariadne: a framework for reachability analysis of hybrid automata. In *M.T.N.S.*, 2006.

[29] C. Belta and F. Ivancic, editors. *Proceedings of the 16th international conference on Hybrid systems: computation and control, HSCC 2013, April 8-11, 2013, Philadelphia, PA, USA*. ACM, 2013.

[30] A. Bemporad, A. Bicchi, and G. C. Buttazzo, editors. *Hybrid Systems: Computation and Control, 10th International Workshop, HSCC 2007, Pisa, Italy, April 3-5, 2007, Proceedings*, volume 4416 of *Lecture Notes in Computer Science*. Springer, 2007.

[31] O. Botchkarev and S. Tripakis. Verification of hybrid systems with linear differential inclusions using ellipsoidal approximations. In *Hybrid Systems: Computation and Control*, pages 73–88. Springer, 2000.

[32] S. P. Boyd, L. El Ghaoui, E. Feron, and V. Balakrishnan. *Linear matrix inequalities in system and control theory*, volume 15. SIAM, 1994.

[33] M. Bozga, C. Daws, O. Maler, A. Olivero, S. Tripakis, and S. Yovine. Kronos: A model-checking tool for real-time systems. In *Formal Techniques in Real-Time and Fault-Tolerant Systems*, pages 298–302. Springer, 1998.

[34] C. Brooks, E. Lee, S. Tripakis, et al. Exploring models of computation with ptolemy ii. In *Hardware/Software Codesign and System Synthesis (CODES+ ISSS), 2010 IEEE/ACM/IFIP International Conference on*, pages 331–332. IEEE, 2010.

[35] R. W. Butler. An elementary tutorial on formal specification and verification using PVS 2. NASA Technical Memorandum 108991, NASA Langley Research Center, Hampton, VA, June 1993. Revised June 1995. Available, with PVS specification files, at http://atb-www.larc.nasa.gov/ftp/larc/PVS-tutorial; use only files marked "revised.".

[36] K. M. Chandy and L. Lamport. Distributed snapshots: Determining global states of distributed systems. *ACM Transactions on on Computer Systems*, 3(1):63–75, 1985.

[37] X. Chen, E. Abraham, and S. Sankaranarayanan. Taylor model flowpipe construction for non-linear hybrid systems. In *R.T.S.S.*, 2012.

[38] X. Chen, E. Ábrahám, and S. Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, pages 258–263, 2013.

[39] E. Clarke, E. A. Emerson, and A. P. Sistla. Automatic verification of finite-state concurrent systems using temporal logic specifications. *ACM Transactions on Programming Languages and Systems*, 8(2):244–263, 1986.

[40] E. Clarke, A. Fehnker, Z. Han, B. Krogh, O. Stursberg, and M. Theobald. Verification of hybrid systems based on counterexample-guided abstraction refinement. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 192–207. Springer, 2003.

[41] 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.

[42] E. M. Clarke, O. Grumberg, and D. Peled. *Model checking*. MIT press, 1999.

[43] R. Cooper and K. Marzullo. Consistent Detection of Global Predicates. In *Proceedings of the Workshop on Parallel and Distributed Debugging*, pages 163–173, 1991.

[44] W. Damm, C. Ihlemann, and V. Sofronie-Stokkermans. Decidability and complexity for the verification of safety properties of reasonable linear hybrid automata. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 73–82. ACM, 2011.

[45] T. Dang and O. Maler. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, pages 96–109. Springer, 1998.

[46] E. Davison and E. Kurak. A computational method for determining quadratic lyapunov functions for non-linear systems. *Automatica*, 7(5):627–636, 1971.

[47] L. de Moura and N. Bjørner. Z3: An efficient SMT solver. In *Proceedings of the 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS 2008)*, volume 4963 of *Lecture Notes in Computer Science*, pages 337–340, Budapest, Hungary, 2008. Springer.

[48] L. De Moura and N. Bjørner. Z3: an efficient smt solver. In *Proc. of 14th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, TACAS'08/ETAPS'08, pages 337–340. Springer-Verlag, 2008.

[49] Y. Deng, A. Rajhans, and A. A. Julius. STRONG: A trajectory-based verification toolbox for hybrid systems. In *Proceedings of the 10th International Conference on Quantitative Evaluation of Systems (QEST 2013)*, volume 8054 of *Lecture Notes in Computer Science*, pages 165–168, Buenos Aires, Argentina, 2013. Springer.

[50] A. Donzé. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Proceedings of the 22nd International Conference on Computer Aided Verification (CAV 2010)*, volume 6174 of *Lecture Notes in Computer Science*, pages 167–170, Edinburgh, UK, 2010. Springer.

[51] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. In Bemporad et al. [30], pages 174–189.

[52] A. Donzé and O. Maler. Systematic simulation using sensitivity analysis. *H.S.C.C*, 2007.

[53] P. S. Duggirala, C. Fan, S. Mitra, and M. Viswanathan. Meeting a powertrain verification challenge. In *Computer Aided Verification - 27th International Conference, CAV 2015, San Francisco, CA, USA, July 18-24, 2015, Proceedings, Part I*, pages 536–543, 2015.

[54] P. S. Duggirala, T. T. Johnson, A. Zimmerman, and S. Mitra. Static and dynamic analysis of timed distributed traces. In *Proceedings of the 33rd IEEE Real-Time Systems Symposium, RTSS 2012, San Juan, PR, USA, December 4-7, 2012*, pages 173–182, 2012.

[55] P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *Proceedings of the 13th International Conference on Embedded Software (EMSOFT 2013)*, Montreal, Canada, 2013.

[56] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2E2: A verification tool for stateflow models. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 68–82, 2015.

[57] P. S. Duggirala and A. Tiwari. Safety verification for linear systems. In *Proceedings of the International Conference on Embedded Software, EMSOFT 2013, Montreal, QC, Canada, September 29 - Oct. 4, 2013*, pages 7:1–7:10, 2013.

[58] P. S. Duggirala, L. Wang, S. Mitra, M. Viswanathan, and C. A. Muñoz. Temporal precedence checking for switched models and its application to a parallel landing protocol. In *FM 2014: Formal Methods - 19th International Symposium, Singapore, May 12-16, 2014. Proceedings*, pages 215–229, 2014.

[59] Eigen. *a C++ template library for linear algebra*, (accessed February, 2015). http://eigen.tuxfamily.org.

[60] C. Fan and S. Mitra. Bounded verification using on-the-fly discrepancy computation. Technical Report UILU-ENG-15-2201, Coordinated Science Laboratory, University of Illinois at Urbana-Champaign, February 2015.

[61] C. Fan and S. Mitra. Bounded verification with on-the-fly discrepancy computation. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2015*, 2015.

[62] M. Fränzle and J. Lygeros, editors. *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*. ACM, 2014.

[63] M. Fränzle and J. Lygeros, editors. *17th International Conference on Hybrid Systems: Computation and Control (part of CPS Week), HSCC'14, Berlin, Germany, April 15-17, 2014*. ACM, 2014.

[64] G. Frehse. Phaver: Algorithmic verification of hybrid systems past hytech. In M. Morari and L. Thiele, editors, *HSCC*, volume 3414 of *Lecture Notes in Computer Science*, pages 258–273. Springer, 2005.

[65] G. Frehse, C. L. 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 Gopalakrishnan and Qadeer [84], pages 379–395.

[66] G. Frehse, R. Kateja, and C. L. Guernic. Flowpipe approximation and clustering in space-time. In Belta and Ivancic [29], pages 203–212.

[67] G. Frehse, B. H. Krogh, and R. A. Rutenbar. Verifying analog oscillator circuits using forward/backward abstraction refinement. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*, DATE '06, pages 257–262, 3001 Leuven, Belgium, Belgium, 2006. European Design and Automation Association.

[68] P. Fritzson. *Principles of object-oriented modeling and simulation with Modelica 2.1*. John Wiley & Sons, 2010.

[69] P. Gahinet, A. Nemirovskii, A. J. Laub, and M. Chilali. The lmi control toolbox. In *IEEE conference on decision and control*, volume 2, pages 2038–2038. INSTITUTE OF ELECTRICAL ENGINEERS INC (IEE), 1994.

[70] S. Gao, S. Kong, and E. M. Clarke. dreal: An SMT solver for nonlinear theories over the reals. In *Automated Deduction - CADE-24 - 24th International Conference on Automated Deduction, Lake Placid, NY, USA, June 9-14, 2013. Proceedings*, pages 208–214, 2013.

[71] S. Gao, S. Kong, and E. M. Clarke. Satisfiability modulo odes. In *Formal Methods in Computer-Aided Design, FMCAD 2013, Portland, OR, USA, October 20-23, 2013*, pages 105–112, 2013.

[72] V. K. Garg. Observation of global properties in distributed systems. In *Proceedings of International Conference on Distributed Computing*, pages 418–425, 1996.

[73] V. K. Garg, C. Chase, R. Kilgore, and J. R. Mitchell. Efficient Detection of Channel Predicates. *Journal of Parallel and Distributed Computing*, 45(2):134–147, 1997.

[74] V. K. Garg and N. Mittal. On Slicing a Distributed Computation. In *Proceedings of the International Conference on Distributed Computing Systems*, pages 322–329, 2001.

[75] V. K. Garg and B. Waldecker. Detection of Weak Unstable Predicates in Distributed Programs. *IEEE Transactions on Parallel and Distributed Systems*, 5(3):299–307, 1994.

[76] K. Ghorbal and A. Platzer. Characterizing algebraic invariants by differential radical invariants. In *Tools and Algorithms for the Construction and Analysis of Systems - 20th International Conference, TACAS 2014, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2014, Grenoble, France, April 5-13, 2014. Proceedings*, pages 279–294, 2014.

[77] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, pages 291–305. Springer, 2005.

[78] A. Girard. Verification using simulation. In *HSCC*, 2006.

[79] A. Girard and G. J. Pappas. Approximate bisimulations for nonlinear dynamical systems. In *Decision and Control, 2005 and 2005 European Control Conference. CDC-ECC'05. 44th IEEE Conference on*, pages 684–689. IEEE, 2005.

[80] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. In *IEEE Transactions on Automatic Control*, 2005.

[81] A. Girard and G. J. Pappas. Approximation metrics for discrete and continuous systems. *Automatic Control, IEEE Transactions on*, 52(5):782–798, 2007.

[82] A. Girard, G. Pola, and P. Tabuada. Approximately bisimilar symbolic models for incrementally stable switched systems. In M. Egerstedt and B. Mishra, editors, *HSCC*, volume 4981 of *Lecture Notes in Computer Science*, pages 201–214. Springer, 2008.

[83] A. Girard and S. Sankaranarayanan, editors. *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control, HSCC'15, Seattle, WA, USA, April 14-16, 2015*. ACM, 2015.

[84] G. Gopalakrishnan and S. Qadeer, editors. *Computer Aided Verification - 23rd International Conference, CAV 2011, Snowbird, UT, USA, July 14-20, 2011. Proceedings*, volume 6806 of *Lecture Notes in Computer Science*. Springer, 2011.

[85] R. Grosu, G. Batt, F. H. Fenton, J. Glimm, C. L. Guernic, S. A. Smolka, and E. Bartocci. From cardiac cells to genetic regulatory networks. In Gopalakrishnan and Qadeer [84], pages 396–411.

[86] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. *A user guide to HyTech*. Springer, 1995.

[87] T. A. Henzinger, P.-H. Ho, and H. Wong-Toi. Hytech: A model checker for hybrid systems. In *Computer Aided Verification (CAV '97)*, volume 1254 of *LNCS*, pages 460–483, 1997.

[88] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya. What's decidable about hybrid automata? *Journal of Computer and System Sciences*, 57:94–124, 1998.

[89] J. P. Hespanha. *Linear systems theory*. Princeton university press, 2009.

[90] Z. Huang, C. Fan, A. Mereacre, S. Mitra, and M. Z. Kwiatkowska. Invariant verification of nonlinear hybrid automata networks of cardiac cells. In *Computer Aided Verification - 26th International Conference, CAV 2014, Held as Part of the Vienna Summer of Logic, VSL 2014, Vienna, Austria, July 18-22, 2014. Proceedings*, pages 373–390, 2014.

[91] Z. Huang and S. Mitra. Proofs from simulations and modular annotations. In Fränzle and Lygeros [62], pages 183–192.

[92] M. Hurfin, M. Mizuno, M. Raynal, and M. Singhal. Eficient Detection of Conjunctions of Local Predictaes. *IEEE Transactions of Software Engineering*, 24(8):664–677, 1998.

[93] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 21–36. Springer, 2015.

[94] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts. Benchmarks for model transformations and conformance checking. In *1st International Workshop on Applied Verification for Continuous and Hybrid Systems (ARCH)*, 2014.

[95] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts. Powertrain control verification benchmark. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 253–262. ACM, 2014.

[96] S. C. Johnson, G. W. Lohr, B. T. McKissick, N. M. Guerreiro, and P. Volk. Simplified aircraft-based paired approach: Concept definition and initial analysis. Technical Report NASA/TP-2013-217994, NASA, Langley Research Center, 2013.

[97] T. T. Johnson and S. Mitra. Parameterized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In *ACM/IEEE 3rd International Conference on Cyber-Physical Systems*, Apr. 2012.

[98] T. T. Johnson and S. Mitra. Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In *2012 IEEE/ACM Third International Conference on Cyber-Physical Systems, Beijing, China, April 17-19, 2012*, pages 161–170, 2012.

[99] T. T. Johnson and S. Mitra. Parametrized verification of distributed cyber-physical systems: An aircraft landing protocol case study. In *Proceedings of the 2012 IEEE/ACM Third International Conference on Cyber-Physical Systems*, pages 161–170. IEEE Computer Society, 2012.

[100] J. Jouffroy and J.-J. E. Slotine. Methodological remarks on contraction theory. In *Decision and Control, 2004. CDC. 43rd IEEE Conference on*, volume 3, pages 2537–2543. IEEE, 2004.

[101] A. A. Julius, G. E. Fainekos, M. Anand, I. Lee, and G. J. Pappas. Robust test generation and coverage for hybrid systems. In *H.S.C.C.*, 2007.

[102] A. Kanade, R. Alur, F. Ivancic, S. Ramesh, S. Sankaranarayanan, and K. Shashidhar. Generating and analyzing symbolic traces of simulink/stateflow models. In *C. A. V.* 2009.

[103] J. Kapinski, J. V. Deshmukh, S. Sankaranarayanan, and N. Arechiga. Simulation-guided lyapunov analysis for hybrid dynamical systems. In Fränzle and Lygeros [62], pages 133–142.

[104] D. K. Kaynar, N. Lynch, R. Segala, and F. Vaandrager. *The Theory of Timed I/O Automata*. Synthesis Lectures on Computer Science. Morgan Claypool, November 2005. Also available as Technical Report MIT-LCS-TR-917.

[105] H. K. Khalil and J. Grizzle. *Nonlinear systems*, volume 3. Prentice hall New Jersey, 1996.

[106] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: δ-reachability analysis for hybrid systems. In *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, pages 200–205, 2015.

[107] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[108] A. B. Kurzhanski and P. Varaiya. Ellipsoidal techniques for reachability analysis: internal approximation. *Systems & control letters*, 41(3):201–211, 2000.

[109] G. Lafferriere, G. J. Pappas, and S. Yovine. A new class of decidable hybrid systems. In *In Hybrid Systems : Computation and Control*, pages 137–151. Springer, 1999.

[110] L. Lamport. Time, clocks, and the ordering of events in a distributed system. *Commun. ACM*, 21(7):558–565, 1978.

[111] K. G. Larsen, F. Larsson, P. Pettersson, and W. Yi. Efficient verification of real-time systems: compact data structure and state-space reduction. In *Real-Time Systems Symposium, 1997. Proceedings., The 18th IEEE*, pages 14–24. IEEE, 1997.

[112] K. G. Larsen, P. Pettersson, and W. Yi. Uppaal in a nutshell. *International Journal on Software Tools for Technology Transfer (STTT)*, 1(1):134–152, 1997.

[113] C. Le Guernic and A. Girard. Reachability analysis of hybrid systems using support functions. In *Computer Aided Verification*, pages 540–554. Springer, 2009.

[114] Y. Lin and S. Mitra. Starl: Towards a unified framework for programming, simulating and verifying distributed robotic systems. In *Proceedings of the 16th ACM SIGPLAN/SIGBED Conference on Languages, Compilers and Tools for Embedded Systems, LCTES 2015, CD-ROM, Portland, OR, USA, June 18 - 19, 2015*, pages 9:1–9:10, 2015.

[115] C. Livadas, J. Lygeros, and N. A. Lynch. High-level modeling and analysis of TCAS. In *Proceedings of the 20th IEEE Real-Time Systems Symposium (RTSS'99),Phoenix, Arizona*, pages 115–125, December 1999.

[116] J. Löfberg. Yalmip: A toolbox for modeling and optimization in matlab. In *Computer Aided Control Systems Design, 2004 IEEE International Symposium on*, pages 284–289. IEEE, 2004.

[117] W. Lohmiller and J.-J. Slotine*. Contraction analysis of non-linear distributed systems. *International Journal of Control*, 78(9):678–688, 2005.

[118] W. Lohmiller and J. J. E. Slotine. On contraction analysis for non-linear systems. *Automatica*, 32(6):683–696, 1998.

[119] S. M. Loos, A. Platzer, and L. Nistor. Adaptive cruise control: Hybrid, distributed, and now formally verified. In M. Butler and W. Schulte, editors, *Formal Methods*, LNCS. Springer, 2011.

[120] S. M. Loos, D. Renshaw, and A. Platzer. Formal verification of distributed aircraft controllers. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 125–130. ACM, 2013.

[121] N. A. Lynch, R. Segala, and F. W. Vaandrager. Hybrid I/O automata. *Inf. Comput.*, 185(1):105–157, 2003.

[122] O. Maler, D. Nickovic, and A. Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*, pages 475–505. Springer, 2008.

[123] K. Manamcheri, S. Mitra, S. Bak, and M. Caccamo. A step towards verification and synthesis from simulink/stateflow models. In *H.S.C.C*, 2011.

[124] Z. Manna and A. Pnueli. A hierarchy of temporal properties. In *Proceedings of the Sixth Annual ACM Symposium on Principles of Distributed Computing (PODC '87)*, pages 205–205, Vancouver, British Columbia, Canada, 1987. ACM.

[125] K. Marzullo and G. Neiger. Detection of Global State Predicates. In *Proceedings of the Workshop on Distributed Algorithms*, pages 254–272, 1991.

[126] K. L. McMillan. *Symbolic model checking*. Springer, 1993.

[127] S. Mitra. *A Verification Framework for Hybrid Systems*. PhD thesis, Massachusetts Institute of Technology, Cambridge, MA 02139, September 2007.

[128] S. Mitra and M. Archer. PVS strategies for proving abstraction properties of automata. *Electronic Notes in Theoretical Computer Science*, 125(2):45–65, 2005.

[129] N. Mittal and V. K. Garg. Techniques and applications of computation slicing. *Distributed Computing*, 17(3):251–277, 2005.

[130] C. Moler and C. Van Loan. Nineteen dubious ways to compute the exponential of a matrix, twenty-five years later. *SIAM review*, 45(1):3–49, 2003.

[131] C. Munoz, V. Carreño, and G. Dowek. Formal analysis of the operational concept for the small aircraft transportation system. In *Rigorous Development of Complex Fault-Tolerant Systems*, pages 306–325. Springer, 2006.

[132] N. Nedialkov. VNODE-LP: Validated solutions for initial value problem for ODEs. Technical report, McMaster University, 2006.

[133] T. Nghiem, S. Sankaranarayanan, G. Fainekos, F. Ivancić, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control (HSCC 2010)*, pages 211–220, Stockholm, Sweden, 2010. ACM.

[134] S. Owre, J. M. Rushby, and N. Shankar. Pvs: A prototype verification system. In *Automated DeductionCADE-11*, pages 748–752. Springer, 1992.

[135] A. Papachristodoulou and S. Prajna. On the construction of lyapunov functions using the sum of squares decomposition. In *Decision and Control, 2002, Proceedings of the 41st IEEE Conference on*, volume 3, pages 3482–3487. IEEE, 2002.

[136] Y. Pei, E. Entcheva, R. Grosu, and S. Smolka. Efficient modeling of excitable cells using hybrid automata. In *In Proc. of CMSB'05 Computational Methods in Systems Biology Workshop*, Edinburgh, UK, April 2005.

[137] R. B. Perry, M. M. Madden, W. Torres-Pomales, and R. W. Butler. The simplified aircraft-based paired approach with the ALAS alerting algorithm. Technical Report NASA/TM-2013-217804, NASA, Langley Research Center, 2013.

[138] A. Platzer. Differential logic for reasoning about hybrid systems. In Bemporad et al. [30], pages 746–749.

[139] A. Platzer. Differential dynamic logic for hybrid systems. *J. Autom. Reasoning*, 41(2):143–189, 2008.

[140] A. Platzer. *Logical Analysis of Hybrid Systems - Proving Theorems for Complex Dynamics*. Springer, 2010.

[141] A. Platzer. Logics of dynamical systems. In *Proceedings of the 27th Annual IEEE Symposium on Logic in Computer Science, LICS 2012, Dubrovnik, Croatia, June 25-28, 2012*, pages 13–24, 2012.

[142] A. Platzer and E. M. Clarke. *Formal verification of curved flight collision avoidance maneuvers: A case study*. Springer, 2009.

[143] A. Platzer and J.-D. Quesel. Keymaera: A hybrid theorem prover for hybrid systems (system description). In *Automated Reasoning*, pages 171–178. Springer, 2008.

[144] A. Pnueli. The temporal logic of programs. In *Foundations of Computer Science, 1977., 18th Annual Symposium on*, pages 46–57. IEEE, 1977.

[145] P. Prabhakar, P. S. Duggirala, S. Mitra, and M. Viswanathan. Hybrid automata-based cegar for rectangular hybrid systems. In *Verification, Model Checking, and Abstract Interpretation*, pages 48–67. Springer, 2013.

[146] P. Prabhakar and M. Viswanathan. A dynamic algorithm for approximate flow computations. In *Proceedings of the 14th international conference on Hybrid systems: computation and control*, pages 133–142. ACM, 2011.

[147] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud. A decidable class of planar linear hybrid systems. In *Hybrid Systems: Computation and Control, 11th International Workshop, HSCC 2008, St. Louis, MO, USA, April 22-24, 2008. Proceedings*, volume 4981 of *LNCS*, pages 401–414. Springer, 2008.

[148] P. Prabhakar, V. Vladimerou, M. Viswanathan, and G. E. Dullerud. Verifying tolerant systems using polynomial approximations. In *Real-Time Systems Symposium, 2009, RTSS 2009. 30th IEEE*, pages 181–190. IEEE, 2009.

[149] A. Puri and P. Varaiya. Decidability of hybrid systems with rectangular differential inclusions. In *Computer Aided Verification*, pages 95–104. Springer, 1994.

[150] A. Puri and P. Varaiya. Decidable hybrid systems. *Mathematical and computer modelling*, 23(11):191–202, 1996.

[151] S. Ratschan and Z. She. Safety verification of hybrid systems by constraint propagation based abstraction refinement. In *Hybrid Systems: Computation and Control*, pages 573–589. Springer, 2005.

[152] A. Roy and R. S. Parker. Dynamic modeling of exercise effects on plasma glucose and insulin levels. *Journal of diabetes science and technology*, 1(3):338–347, 2007.

[153] S. Sankaranarayanan and A. Tiwari. Relational abstractions for continuous and hybrid systems. In *Computer Aided Verification*, pages 686–702. Springer, 2011.

[154] S. Sastry, G. Meyer, C. Tomlin, J. Lygeros, D. Godbole, and G. Pappas. Hybrid control in air traffic management systems. In *Decision and Control, 1995., Proceedings of the 34th IEEE Conference on*, volume 2, pages 1478–1483. IEEE, 1995.

[155] A. Sen and V. K. Garg. Formal verification of simulation traces using computation slicing. *IEEE Trans. Computers*, 56(4):511–527, 2007.

[156] B. Sharma and I. Kar. Design of asymptotically convergent frequency estimator using contraction theory. *IEEE Trans. Automat. Contr.*, 2008.

[157] S. D. Stoller and Y. A. Liu. Efficient Symbolic Detection of Global Predicates. pages 357–368, 1998.

[158] S. D. Stoller, L. Unnikrishnan, and Y. A. Liu. Efficient Detection of Global Properies in Distributed Systems Using Partial-Order Methods. pages 284–279, 2000.

[159] O. Stursberg and B. H. Krogh. Efficient representation and computation of reachable sets for hybrid systems. In *Hybrid Systems: Computation and Control*, pages 482–497. Springer, 2003.

[160] P. Tabuada. *Verification and control of hybrid systems: a symbolic approach*. Springer Science & Business Media, 2009.

[161] A. Taly and A. Tiwari. Deductive verification of continuous dynamical systems. In *IARCS Annual Conference on Foundations of Software Technology and Theoretical Computer Science, FSTTCS 2009, December 15-17, 2009, IIT Kanpur, India*, pages 383–394, 2009.

[162] A. Tiwari. Formal semantics and analysis methods for Simulink Stateflow models. Technical report, SRI International, 2002. `http://www.csl.sri.com/-~tiwari/stateflow.html`.

[163] A. Tiwari. Hybridsal relational abstracter. In *Computer Aided Verification*, pages 725–731. Springer, 2012.

[164] C. Tomlin, I. Mitchell, and R. Ghosh. Safety verification of conflict resolution manoeuvres. *Intelligent Transportation Systems, IEEE Transactions on*, 2(2):110–120, 2001.

[165] C. Tomlin, G. J. Pappas, and S. Sastry. Conflict resolution for air traffic management: A study in multiagent hybrid systems. *Automatic Control, IEEE Transactions on*, 43(4):509–521, 1998.

[166] V. Vladimerou, P. Prabhakar, M. Viswanathan, and G. Dullerud. Specifications for decidable hybrid games. *Theoretical Computer Science*, 412(48):6770–6785, 2011.

[167] J. L. Welch and N. Lynch. A new fault-tolerant algorithm for clock synchronization. *Inf. Comput.*, 77(1):1–36, Apr. 1988.

[168] G. Wood and B. Zhang. Estimation of the Lipschitz constant of a function. *Journal of Global Optimization*, 8:91–103, 1996.

[169] M. Zamani, G. Pola, M. Mazo, and P. Tabuada. Symbolic models for nonlinear control systems without stability assumptions. *IEEE Trans. Automat. Contr.*, 2012.

[170] A. Zimmerman and S. Mitra. A programming environment for ad hoc wifi applications over android, 2012. https://wiki.cites.uiuc.edu/wiki/display/MitraResearch/StarL.

[171] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *2014 International Conference on Embedded Software, EMSOFT 2014, New Delhi, India, October 12-17, 2014*, pages 5:1–5:10, 2014.

[172] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *Proc. of IEEE Conference on Decision and Control*, 2013.