BEHAVIORAL VALIDATION IN CYBER-PHYSICAL SYSTEMS: SAFETY VIOLATIONS AND BEYOND

Manish Goyal

A dissertation submitted to the faculty of the University of North Carolina at Chapel Hill in partial fulfillment of the requirements for the degree of Doctor of Philosophy in the Department of Computer Science.

Chapel Hill
2022

Approved by:

Parasara Sridhar Duggirala

Cynthia Sturton

Goran Frehse

Samarjit Chakraborty

Indranil Saha

Nikos Arechiga

ABSTRACT

Manish Goyal: Behavioral validation in Cyber-physical systems: Safety violations and beyond
(Under the direction of Parasara Sridhar Duggirala)

The advances in software and hardware technologies in the last two decades have paved the way for the development of complex systems we observe around us. Avionics, automotive, power grid, medical devices, and robotics are a few examples of such systems which are usually termed as Cyber-physical systems (CPS) as they often involve both physical and software components. Deployment of a CPS in a safety critical application mandates that the system operates reliably even in adverse scenarios. While effective in improving confidence in system functionality, testing can not ascertain the absence of failures; whereas, formal verification can be exhaustive but it may not scale well as the system complexity grows. Simulation driven analysis tends to bridge this gap by tapping key system properties from the simulations. Despite their differences, all these analyses can be pivotal in providing system behaviors as the evidence to the satisfaction or violation of a given performance specification. However, less attention has been paid to algorithmically validating and characterizing different behaviors of a CPS.

The focus of this thesis is on behavioral validation of Cyber-physical systems, which can supplement an existing CPS analysis framework. This thesis develops algorithmic tools for validating verification artifacts by generating a variety of counterexamples for a safety violation in a linear hybrid system. These counterexamples can serve as performance metrics to evaluate different controllers during design and testing phases. This thesis introduces the notion of complete characterization of a safety violation in a linear system with bounded inputs, and it proposes a sound technique to compute and efficiently represent these characterizations. This thesis further presents neural network based frameworks to perform systematic state space exploration guided by sensitivity or its gradient approximation in learning-enabled control (LEC) systems. The presented technique is accompanied with convergence guarantees and yields considerable performance gain over a widely used falsification platform for a class of signal temporal logic (STL) specifications.

I dedicate this thesis to allies, activists standing up for the rights and representation of LGBTQIA+ and other marginalized communities across the world, to generations of women striving for dignity and respect, to healthcare professionals for their unrelenting spirit during COVID-19 pandemic, and to my family for its love and support.

ACKNOWLEDGEMENTS

I am so very grateful for the many people who made this dissertation possible and supported me along the way. The completion of this dissertation would not have been possible without all of you.

First, I would like to thank my advisor, Parasara Sridhar Duggirala, who gave me the opportunity to join his research group. He always supported me, patiently guided me, and helped me navigate through graduate school by teaching nuances of scientific research, suggesting new research directions and providing various technical insights resulting into interesting solutions. I am grateful to him for his critical yet invaluable feedback on my writing and presentation on multiple occasions that has improved my confidence, strengthen my skill set and helped me mature as a researcher.

I would like to thank the late Oded Maler who introduced me to the field of hybrid systems. He went out of his way in helping me during my first visit at Verimag and he will forever remain one of most humble human beings I have ever met. I would also like to thank my Masters thesis advisor, Purandar Bhaduri with whom I had my first experience in research. Honestly, if it was not for their encouragement, I likely would not have considered pursuing a doctorate.

I would like to thank my dissertation committee members, Cynthia Sturton, Goran Frehse, Samarjit Chakraborty, Indranil Saha, and Nikos Arechiga, for all of their useful feedback, technical discussions, and for accommodating meeting requests in their schedules in spite of them being in different time zones.

I want to express my gratitude to all of my collaborators: Abolfazl Karimi (Abel), Muqsit Azeem, Miheer Dewaskar, Kumar Madhukar, David Bergman, R. Venkatesh, Tanya Amert, Catherine Nemitz, and Jim Anderson. This dissertation would not have been possible without their help. I want to especially recognize Abel, who has been a wonderful office-mate, a super helpful colleague, and a great friend. My special thanks to David, Miheer, and Muqsit for their contributions in bringing challenging projects to completion. I also appreciate the discussions and interactions with many other UNC and UConn graduate students: Meghan, Edward, Bashima, Tamzeed, Shareef, Clara, Sims, Jisan, Nathan, Nandan, Luke, Timothy, Chloe, Sohaib, Devon, and Reynaldo.

I appreciate the staff in the Computer Science Department, who have done so much to keep everything running smoothly, especially David Cowhig, Brandi Day, the late Bil Hays, Denise Kenney, and Deb Levin. I would also like to thank all of the faculty, and especially Ron Alteroviz, Montek Singh, Shahriar Nirjon, Kevin Jeffay, Leonard McMillan, and Jasleen Kaur.

I would like to acknowledge my mentors during internships - Kumar Madhukar at Tata Research Design and Development Center, Debamitro Chakraborti and Manu Chopra at Cadence Design Systems, and Arvind Raghunathan at Mitsubishi Electric Research Lab, for their supervision. I learned about the craft of scientific research, software development, and gained invaluable experience in handling industrial scale systems.

I would like to thank Anish Kumar Nayak for being a great mentor at Synopsys and helping me grow as a professional, and for always providing valuable advice on career. Thanks to Pravritti, an informal LGBTQIA+ group, which has played a crucial role in helping me accept myself. I am glad to have known Mahesh Agarwal who is an amazing friend, is very approachable, and interactions with him have taught me things about various aspects of life. Special thanks to a group of people whose company has kept me sane at different stages in my grad school life: Guneet, Amrita, Aman, Samapriya, Yanan, Miheer, Kunj, Wasim, and Wayne. My time as a graduate student has been made richer by many more friends: Sameera, Vikas, Harish, Shariq, Arshiah, Suresh, Krishna, Mathangi, Anand, Deepthi, Saketh, Preeti, and members of Tarang@UConn. I am also thankful for the lifelong friendships with those I have known well before starting my doctorate: Sidharth, Abhilash, Nilapratim, Krishna, Subhankar, Vallabh, Prabhat, Nirmal, Tushar, Parveen, and Naveen.

I am grateful to all my family members, especially my parents, my cousin: Deepika, my brother: Rajat, my sister-in-law: Kanika, my sister: Divya, my nephew: Samarth, and my nieces: Paakhi and Vidushi, for their love and support.

Last but not least, Vardhman - thank you for being there.

The research in this dissertation was funded by Air Force Office of Scientific Research award FA9550-19-1-0288 and National Science Foundation (NSF) grants CNS 1739936, CNS 1935724, CNS 2089630, and CNS 2038960. Any opinions, findings, and conclusions or recommendations expressed in this material are those of the author(s) and do not necessarily reflect the views of the United States Air Force or National Science Foundation.

"Experience has shown that counterexamples are the single most effective feature to convince system engineers about the value of formal verification. A counterexample can take you almost directly to the source of an error in a program or circuit. Some people use model checking primarily to find counterexamples."

An excerpt from the lecture on "Model Checking and the Curse of Dimensionality"
by
Edmund Melson Clarke, 2007 Turing Award Recipient,
@ 2013 Heidelberg Laureate Forum.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ABBREVIATIONS

CPS	Cyber-physical Systems
STL	Signal Temporal Logic
MTL	Metric Temporal Logic
HyLAA	Hybrid Linear Automata Analyzer
SMT	Satisfiability Modulo Theories
MILP	Mixed Integer Linear Programming
LCE	Longest Counterexample
DCE	Deepest Counterexample
RCE	Robust Counterexample
ODE	Ordinary Differential Equation
BDD	Binary Decision Diagram
LEC	Learning Enabled Control
ACC	Adaptive Cruise Control ...

CHAPTER 1: INTRODUCTION

Integration of software with embedded control systems has evolved as a field of Cyber-Physical Systems (CPS). It involves interaction between the continuous physical environment modeled as an ordinary differential equation (ODE) and discrete software systems. Designing a controller for these (potentially infinite) state systems is generally an iterative process which demands sensing and controlling physical quantities so that the system meets the desired behaviors such as stability, robustness, or at best safety. For a given system model and specification, the control designer uses tools in their repertoire to come up with a controller, generates a few test cases to check if the system satisfies the required specification and iteratively refines the controller. However, these test cases often do not generalize to the system behaviors at large. This is especially difficult if one has to consider all possible inter leavings of the continuous and discrete (i.e., hybrid) behaviors encountered by a modern CPS. Due to different sequence of mode changes, two neighboring states can potentially have divergent trajectories, thus extrapolating the behavior from one state to another becomes challenging. The problem is further exacerbated by sophisticated neural network based control algorithms. Since such neural network controllers are typically learned from a finite number of samples, a designer needs to perform additional checks for controller's behavior outside the test suite. However, such manual validation is not practically feasible.

1.1 Safety analysis based counterexamples

Control design for linear systems typically involves techniques such as pole placement and computing Lyapunov functions (Narendra & Balakrishnan 1994, Branicky 1998, Tanaka, Hori & Wang 2003, Lin & Antsaklis 2009). While such stability analysis tool are capable of providing intuitive information to the designer, similar tools for safety verification do not exist. Since most of the model checking approaches for safety verification focus on computing over-approximation of reachable set and hence establish the safety specification, they typically yield one counterexample as an evidence to safety violation. However, current model checkers do not have the capability to generate a variety of counterexamples that can give additional information to control designer. Such lack of information in artifacts from both stability

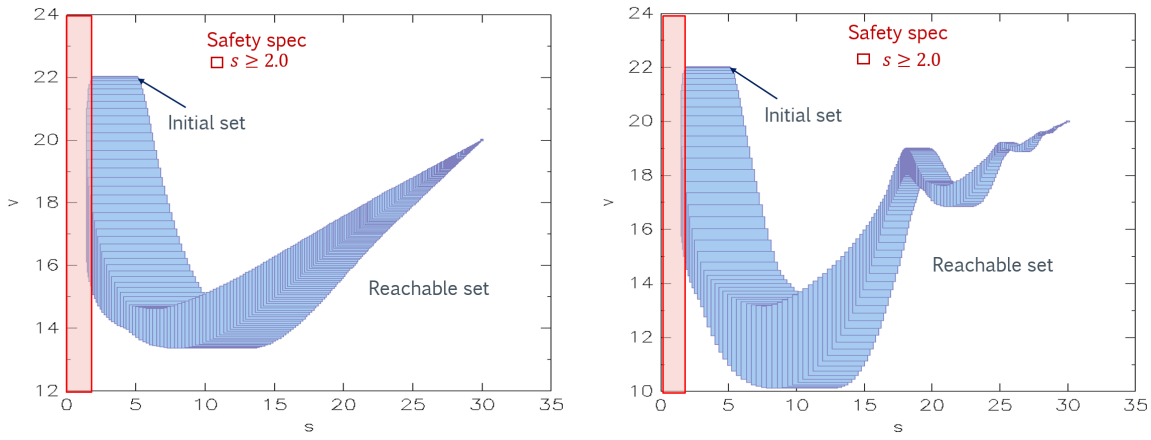


Figure 1.1: Reachable sets computed in SpaceEx for ACC with different controllers

and safety analysis prevents the designer from comparing different possible refinements of an existing controller across design iterations even for simple linear dynamical systems. These challenges are exacerbated when the system is a hybrid system and has several modes of operation.

Consider a 3 dimensional continuous-time linear system with two cars where the distance between two vehicles is s, the leading car is moving with a constant speed v_f, the follower's velocity is v and its acceleration is a. The differential equations for the adaptive cruise control (ACC) system deployed at the follower are as follows (Tiwari 2003):

$$\dot{s} = (v_f - v)$$

$$\dot{v} = a$$

$$\dot{a} = g_1 * a + g_2(v - v_f) + g_3(s - (v + 10))$$

Here, g_1, g_2 and g_3 are *gain* variables generally derived via a feedback control law. Consider the value of the gain variables as $g_1 = -3$, $g_2 = -3$ and $g_3 = 1$ for *Controller-I* and $g_1 = -3$, $g_2 = -3$ and $g_3 = 1$ for *Controller-II*. The stable equilibrium of the system is at $a = 0$, $v = v_f$, and $s = v_f + 10$. The collision avoidance (or, safety) specification is $\Box(s \geq 2)$ i.e., the follower should always maintain at least 2 units of distance from leader car to avoid collision. Reachable sets for these control systems computed in a reachability analysis tool, SpaceEx (Frehse, Le Guernic, Donzé, Cotton, Ray, Lebeltel, Ripado, Girard, Dang & Maler 2011), are demonstrated in Figure 1.1, which shows that both these control systems are stable and unsafe. As the evidence to the safety violation, a safety verification tool

typically spits out an execution (called *counterexample*) that violates the safety specification. However, such violations cease to provide any insights pertaining to the behavior of the controller(s) that can potentially assist a designer during synthesis process. The techniques for measuring the quality of the controller using counterexamples as the feature of a model checking tool would make the verification artifacts more useful to system engineers.

1.2 Testing and State Space Exploration

In recent years, advances in hardware and software have made it easier to integrate sophisticated (including neural network based-) control algorithms in embedded systems (Sutton & Barto 2018, Levine, Pastor, Krizhevsky, Ibarz & Quillen 2018). The control designers now often integrate multiple technologies and satisfy ever increasing behavioral specifications expected from complex CPS. However, these advanced control systems for characteristics and behavior, further augmented with complex specifications makes it difficult to predict the outcomes of perturbations in the state or the environment. The analysis of such systems particularly becomes more challenging because standard analytical tools for linear systems do not easily extend to them in the absence of closed form expression for non-linear ODEs. Since such neural network controllers are typically learned from a finite number of samples, a designer needs to perform additional checks for controller's behavior outside the test suite.

Testing is a simple commonly used technique tasked to determine whether the system model satisfies a given specification using a finite set of test cases. But it can not exhaustively evaluate an infinite state system and may not accurately reflect the manner in which the system will be used after deployment (Kapinski, Deshmukh, Jin, Ito & Butts 2016). Moreover, having tested a control algorithm for the satisfaction of a given safety specification by generating a finite test suite, the designer might like to generate test cases that are close to violating the specification. However, such manual validation is not practically feasible.

State space exploration, on the other hand, is aimed at systematically generate trajectories to explore desired (or undesired) outcomes. In some instances, the specification encoded as a temporal logic formula is used by an off-the-shelf *falsification* tool such as S-TaLiRo (Annpureddy, Liu, Fainekos & Sankaranarayanan 2011) for automatically generating a trajectory that violates (or close to violating) the specification. But such an approach has a few drawbacks. The search for an execution that violates the

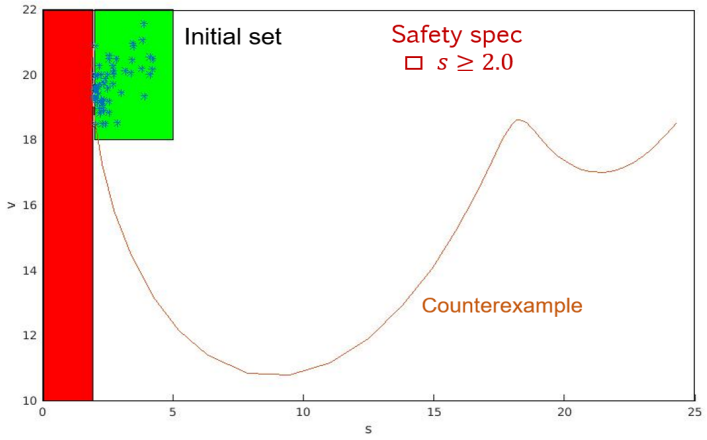


Figure 1.2: ACC controller falsification in S-TaLiRo

specification is performed using stochastic optimization and hence, would not yield much intuition for the control designer about the system behavior as shown in Figure 1.2. Second, if the control designer changes the specification during the design exploration, the results from the previous falsification analyses may no longer be useful. Third, existing falsification tools require the specification to be provided in a temporal logic such as signal temporal logic or metric temporal logic (STL/MTL). The designer needs to understand these specification language which despite being useful in the verification phase, may case hindrance during the design and exploration phases. Next, falsification tools are typically geared towards finding a violating trace for the given specification, not necessarily to help the control designer in exploring the state space. Finally, falsification tools like most of the existing state space exploration approaches lack any theoretical convergence guarantees.

1.3 Thesis Statement

The focus of this thesis is behavioral validation of Cyber-physical systems. *Behavioral Validation* attempts to validate that the CPS exhibits the behavior for a given characteristic or specification and possibly quantify these behaviors. It can be achieved by validating analytical procedures, evaluating their correctness or judging the quality of their results using some metric. In our discussion, we bring validation task under the purview of state space analysis. It includes exploring the utility of an existing CPS analysis technique to more applications, characterize its results, performing qualitative evaluation

using different validation measures, and, if possible, applying formal tools to argue about the performance or correctness of the analysis.

The underlying motivation for this thesis is that system artifacts and simulations contain important traits about system operations. In order to save computational resources, we should be able to extract as much information as possible from this data. My presented work shows that verification artifacts are useful for generating various counterexamples, and that a fixed number of available system traces can be used to approximating sensitivity functions for effective state space exploration.

This leads to the following thesis statement: *Simulation driven algorithmic tools can utilize knowledge of fundamental system characteristics to enable systematic state space analysis of complex Cyber physical systems.*

1.4 Contributions

The thesis is supported by the following contributions.

1.4.1 Generating a variety of counterexamples

A controller that is originally stable and safe, can become unsafe if the safety specification is tightened or the operating conditions are changed. While stability analysis tools of dynamical systems do not guarantee safety, present model checkers for linear hybrid systems lack the capability of classifying counterexamples in case of a safety violation. As not all counterexamples are equivalent to the control designer, finding a right counterexample is crucial to the control design process. We first define the notion of longest, deepest and robust counterexamples. We then discuss how such counterexamples can be used to compare different unsafe controllers in Adaptive Cruise Control system. Finally, we present techniques and frameworks to obtain these counterexamples and provide evaluation results on multiple benchmarks. We develop such counterexample generation capabilities (Goyal, Bergman & Duggirala 2020, Goyal & Duggirala 2018, Goyal & Duggirala 2020*a*) in an affine linear hybrid system verification platform, HyLAA (Bak & Duggirala 2017*a*).

1.4.2 Complete characterization of a safety violation

In spite of their application in evaluating different controllers, various types of counterexamples do not capture all modalities of a safety violation. The problem of computing complete characterization of counterexamples to a safety specification translates into efficiently representing all such modalities. We define the problem of complete characterization and present a technique to obtain a sub-optimal representation, in the form of Binary Decision Diagram (BDD), to represent complete characterization. We then propose an approach which uses Farkas' Lemma, to identify *isomorphic* nodes in order to reduce the size of the original decision diagram. We model the problem of finding isomorphic nodes as a Mixed Integer Linear Program (MILP) and perform rigorous evaluations on various linear systems without and with inputs to underscore the promise of our technique.

1.4.3 State space exploration of closed loop control systems

Given a safety specification and a test suite, present falsification tools perform stochastic optimization to find a violating trace. But existing state space exploration techniques would neither yield much intuition about the course of exploration to the designer nor do they provide any theoretical convergence guarantees. Our first work, NeuralExplorer, (Goyal & Duggirala 2020*b*, Goyal & Duggirala 2020*c*) is a state space exploration framework that uses neural networks to learn sensitivity function(s) from a fixed number of system trajectories. We demonstrate its utility in various applications such as falsification, predicting system trajectories, and supplementing reachability analysis. However, the framework suffers from high training time and lack of theoretical analysis.

The second work, NExG (Goyal, Dewaskar & Duggirala 2022), is an adaptation of NeuralExplorer, which attempts to approximate sensitivity functions for small perturbation. The new framework not only delivers better training performance but also achieves almost an order of magnitude gain over other state space exploration techniques. We also present a theoretical study that guarantees the convergence of our approach at a geometric rate, which is further supported by thorough evaluation on multiple closed loop control systems with neural network feedback controllers.

1.5 Organization

The remainder of this document is organized as follows. Chapter 2 presents terminology, notation, and related work. Chapter 3 defines the notion of longest, deepest and robust counterexamples and presents techniques to generate them in linear hybrid systems. Chapter 4 presents the study on complete characterization of a safety violation in linear dynamical systems. Chapter 5 presents NeuralExplorer, a state space exploration framework based on learning sensitivity functions. Chapter 6 proposes NExG which approximates the gradient of sensitivity functions, showcases its performance and applications. Chapter 7 summarizes the contributions and poses directions for future work.

1.6 Literature Review

CPS involves interaction between continuous nature of physical environment and the discrete nature of software, which makes their analysis challenging. Our work lies at the intersection of symbolic and analytical analyses, and its application spans various domains - verification, falsification, debugging, and control synthesis. It has the potential to supplement existing analysis techniques by generating interesting and desirable system behaviors (including specific counterexamples). While we review the literature for topics most relevant to this thesis, a detailed study on simulation based verification of embedded control systems (Kapinski et al. 2016) may serve as a good primer for the reader interested in further discussion on some of these core areas.

1.6.1 CPS verification

Verification of hybrid systems is, in general, undecidable.(Alur, Courcoubetis, Halbwachs, Henzinger, Ho, Nicollin, Olivero, Sifakis & Yovine 1995). Moreover, tractable verification techniques for certain classes of systems such as timed automata or decidable rectangular hybrid automata are not suitable for modeling realistic CPS with linear or nonlinear dynamics. A common approach for verifying such systems is to compute numerical over-approximations up to a bounded time (Goyal 2012). Using such numerical over-approximations, a reachable set computation tool computes an over-approximation of the set of states reached by all the possible behaviors of the system (Dang & Maler 1998, Alur, Dang & Ivančić 2003, Althoff 2015, Chen, Ábrahám & Sankaranarayanan 2013). If this reachable set does not overlap with the unsafe set of states, then it is concluded that the system is safe. The efficiency

of a reachability analysis technique generally depends on the data structure used for the symbolic representation of the reachable set and associated operations.

PHAVer (Frehse 2005), SpaceEx (Frehse, Le Guernic, Donzé, Cotton, Ray, Lebeltel, Ripado, Girard, Dang & Maler 2011, Frehse, Donzé, Cotton, Ray, Lebeltel, Goyal, Ripado, Dang, Maler, Guernic & Girard 2011) and HyLAA (Bak & Duggirala 2017*a*) are some well-known platforms for the reachability analysis of linear (hybrid) systems. In PHAVer, the reachable set is represented as a convex polyhedron, support functions are used in SpaceEx, whereas HyLAA uses generalized star representation for its reachable set. Systems with nonlinear ODEs might not have a closed form expression for the solution, hence they may require different techniques and representations for their reachable set. Some of the state of the art tools in this domain are CORA (Althoff 2015), Flow* (Chen, Ábrahám & Sankaranarayanan 2013), and C2E2 (Duggirala, Potok, Mitra & Viswanathan 2015). Flow* uses Taylor models, C2E2 uses Jacobian matrix and discrepancy functions, and CORA primarily makes use of zonotopes for representing the reachable sets. Another recent work (Adimoolam & Saha 2022) uses a non-convex set representation called *IoU zonotope* which is the intersection of unions of zonotopes to approximate the reachable set.

A recent work (Roehm, Oehlerking, Heinz & Althoff 2016) on STL model checking of hybrid systems considers an abstraction of the model to reduce the continuous time verification problem to a discrete-time problem for which the decision procedure is shown to be sound and complete. This method, however, does not rely on a specific representation of reachable sets and it can be used with any reachability analysis tool. Simulation based state space exploration (Donzé & Maler 2007, Dang, Donze, Maler & Shalev 2008) and verification (Huang & Mitra 2014, Fan & Mitra 2015) have also shown some promise by tapping the advantages of symbolic and analytical techniques. For example, DryVR (Fan & Mitra 2015) computes an upper bound on the sensitivity of the trajectories for computing the over-approximation of the reachable set. The work in (Dang et al. 2008) needs analytical model for guided random exploration of the state space together using sensitivity analysis to provide better coverage. However, these techniques might suffer due to high system dimensionality and complexity. That is, the number of required trajectories might increase exponentially with dimensions and complex dynamics.

Learning enabled control systems analysis: Given the rich history of application of neural networks in control (Miller, Sutton & Werbos 1991, Lewis, Yesildirak & Jagannathan 1998, Moore 2012) and

the recent advances in software and hardware platforms, neural networks are also being deployed in various control tasks. Consequently, many verification techniques are being developed for neural network based control systems (Ivanov, Carpenter, Weimer, Alur, Pappas & Lee 2021, Tran, Cai, Diego, Musau, Johnson & Koutsoukos 2019, Sun, Khedr & Shoukry 2019, Dutta, Chen, Jha, Sankaranarayanan & Tiwari 2019, Xiang, Tran, Yang & Johnson 2021, Dutta, Chen & Sankaranarayanan 2019, Rober, Everett & How 2022) and some other domains (Tjeng, Xiao & Tedrake 2019, Sun, Huang, Kroening, Sharp, Hill & Ashmore 2019, Huang, Kwiatkowska, Wang & Wu 2017). Some of the the recent neural network control analysis tools are Verisig (Ivanov et al. 2021), NNV (Tran et al. 2019), ReachNN (Huang, Fan, Li, Chen & Zhu 2019), Reach-SDP (Hu, Fazlyab, Morari & Pappas 2020), SyReNN (Sotoudeh & Thakur 2021), and Sherlock (Dutta et al. 2019).

1.6.2 Falsification

Falsification (Nghiem, Sankaranarayanan, Fainekos, Ivancić, Gupta & Pappas 2010, Abbas & Fainekos 2011, Deshmukh, Fainekos, Kapinski, Sankaranarayanan, Zutshi & Jin 2015, Zutshi, Deshmukh, Sankaranarayanan & Kapinski 2014) is geared towards finding an execution that violates a given specification. Given a specification of Cyber-Physical System in Metric Temporal Logic (MTL) (Koymans 1990) or Signal Temporal Logic (STL) (Maler, Nickovic & Pnueli 2008), falsification tools are aimed at discovering an execution that violates the given specification.

The authors in (Zutshi et al. 2014, Deshmukh et al. 2015) present falsification approaches that performs scatter-and-search over segmented (or spliced) trajectories in an abstract graph to find a likeliest counterexample by narrowing the gaps between adjacent segments. Similarly, the paper (Zutshi, Sankaranarayanan, Deshmukh, Kapinski & Jin 2015) combines the symbolic execution of the controller software with an approximation of the plant model, which is discovered on-the-fly using simulations, to discover abstract counterexamples to the given safety property. Classification and Coverage-Based Falsification for Embedded Control Systems (Adimoolam, Dang, Donzé, Kapinski & Jin 2017) presents a falsification technique which uses a coverage measure and support vector machine based classification to identify falsifying input traces. Another work (Bogomolov, Frehse, Gurung, Li, Martius & Ray 2019) uses symbolic reachability supplemented by trajectory splicing to scale up hybrid system falsification.

Some falsification methods perform stochastic optimizations to obtain violating executions to a safety specification based on *robustness* (Fainekos & Pappas 2009, Donzé & Maler 2010). S-Taliro (Annpureddy

et al. 2011) and Breach (Donzé 2010) are two notable falsification tools. However, present falsification techniques do not assist in state space exploration or may lack convergence guarantees due to their heuristics driven behaviors.

More recently, techniques were also developed that inductively strengthen the given property by focusing on some relevant aspects of the transition system or uncover deep bugs which would otherwise take a long time to discover (Bradley 2011).

The counterexample generated by a model checking tool may be a false alarm as a consequence of the over-approximation of the reachable set. For falsification of reach-avoid properties, the work (Goubault & Putot 2019) computes an inner approximation based on the notion of minimal reachable set in order to guarantee the existence of the safety violation.

The authors in (Singh & Saha 2020) argue that debugging a system requires precise information about the internal structure of the model, which eludes standard falsification approaches. For the given Simulink model and STL specification, their technique first obtains a falsifying execution, if any, and then employs a run time monitoring algorithm based on matrix analysis to identify a small subset of the signals (for debugging) that contribute to the falsification.

1.6.3 Counterexamples and their analysis

Counterexamples play an important role in model checking due to their practical relevance in understanding system under test. They can potentially provide intuition to the system designer about the reason why the system does not satisfy the specification. While control systems verification or falsification tools are useful for proving safety specification or its violation, generating counterexamples of interest in the domain of hybrid dynamical systems has hardly been explored. SpaceEx (Frehse, Le Guernic, Donzé, Cotton, Ray, Lebeltel, Ripado, Girard, Dang & Maler 2011) and HyLAA (Bak & Duggirala 2017*a*) spit out the counterexample that violates the safety specification at the earliest time and at the latest time respectively.

While model checking can find subtle errors, extracting the essence of an error may still require a great deal of human effort. This debugging is shown to benefit from using more than one counter-example (Groce & Visser 2003, Fey & Drechsler 2003) as well as by efficiently identifying crucial sites leading to the failure (Zeller 1999, Beer, Ben-David, Chockler, Orni & Trefler 2009, Jin, Ravi & Somenzi 2002, Groce & Visser 2003). To obtain a sub-optimal (minimal) set of candidate error

sites for all counter-examples in an erroneous design, (Fey & Drechsler 2003) introduce two greedy heuristics namely *maximum pairwise distance* and *efficient selection of error sites*. The work (Groce & Visser 2003) denotes the multiple variations of a single counterexample as positive and negative executions. It automates finding these executions and analyzing them to obtain a better summarized description of the erroneous elements. CLEAR tools (Barbon, Leroy & Salaün 2018, Barbon, Leroy & Salaün 2019) assist in debugging behavioral models of the concurrent systems. It attempts to improve the comprehension of counterexamples w.r.t. liveness properties by identifying erroneous parts of the model and highlighting such crucial decision points leading to the bug.

The notion of causality is used to visually explain failure sites on the counterexample trace in (Beer et al. 2009). On the other hand, (Jin, Ravi & Somenzi 2002) focuses on inevitability towards the failure to capture more of the error in the error trace. The trace is presented as an alternation of fated and free segments: the fated segments show unavoidable progress towards the error while free segments show choices that, if avoided, may have prevented the error. Another technique *delta debugging* (Zeller 1999) conducts binary search to discover and minimize difference between failing and succeeding runs of a program.

An alternating line of work (Kupferman & Vardi 2000, Beer, Ben-David, Eisner & Rodeh 1997) focuses on identifying vacuous satisfaction of a formula and generating an interesting witness for the same in ACTL and CTL* respectively. The motivation behind obtaining such witnesses is that a witness provides some confidence that the formal specification accurately reflects the intent of the user, one of the weak links in the practical application of formal verification to hardware design.

1.6.4 Application of counterexamples

The introduction of Counter-Example-Guided-Abstraction-Refinement (CEGAR) (Clarke, Grumberg, Jha, Lu & Veith 2000) changed the role of counterexamples from a mere feature to an algorithmic tool. In CEGAR, the counterexample acts as a primary guide to restricting the space of the possible refinements. In the domain of hybrid systems, different CEGAR based approaches (Dierks, Kupferschmid & Larsen 2007, Clarke, Fehnker, Han, Krogh, Stursberg & Theobald 2003, Prabhakar, Duggirala, Mitra & Viswanathan 2013, Duggirala & Mitra 2011, Alur, Dang & Ivancic 2006, Ratschan & She 2005, Frehse, Krogh & Rutenbar 2006) pursue various notions of counterexamples, but majority of them are restricted

to the domain of timed and rectangular systems. But recently there have been efforts in conducting CEGAR based falsification of hybrid systems (Zutshi et al. 2014, Zutshi et al. 2015).

In the automated control synthesis domain, some frameworks (Fan, Mathur, Mitra & Viswanathan 2018, Ding & Tomlin 2010, Huang, Wang, Mitra, Dullerud & Chaudhuri 2015, Solar-Lezama, Tancau, Bodík, Seshia & Saraswat 2006) synthesize correct-by-construction controller for reach-avoid specification; whereas, counterexample Guided Inductive Synthesis (CEGIS) frameworks (Raman, Donzé, Sadigh, Murray & Seshia 2015, Singh & Saha 2021), as the name suggests, leverage counterexamples from verification for inductive synthesis. The verification condition that the system satisfies an STL specification is encoded as a mixed-integer linear program (MILP) in (Raman et al. 2015). If the specification is violated, one can investigate the results of MILP to obtain counterexamples or try to gain an intuition for the system (Ghosh, Sadigh, Nuzzo, Raman, Donzé, Sangiovanni-Vincentelli, Sastry & Seshia 2016) failing to satisfy the specification. Another recent work (Singh & Saha 2021) proposes an algorithm for synthesizing multi-parameter complex feedback controllers by rigorous analysis of control parameters. The parameters are then selected based on their impact on the specification violation and tuned in an iterative manner.

A theoretical analysis of CEGIS based on different types of counterexamples in the domain of program synthesis is attempted in (Jha & Seshia 2014). It considers minimal and history bounded counterexamples which are aimed at localizing the error. The authors use these traces to investigate whether there are *good mistakes* that could increase synthesis power and conclude that none of the two counterexamples are strictly good mistakes.

Finally, counterexamples are also shown to be useful in learning control Lyapunov-like function which is used in synthesizing controllers for nonlinear dynamical systems (Ravanbakhsh & Sankaranarayanan 2019) or learning polyhedral Lyapunov functions for linear hybrid systems (Berger & Sankaranarayanan 2022).

1.6.5 Other related works

Binary Decision Diagrams: BDD's have been successfully used in computer aided design of logic circuits (Khatri, Narayan, Krishnan, McMillan, Brayton & Sangi 1996), formal verification (Bryant 1986, Hu 1995) and in recent years, for techniques using optimization (Behle 2007, Wegener 2000). At worst, the total number of nodes in a BDD can grow exponentially large in the number of decision variables.

Decision variables ordering plays a crucial role in determining the BDD size thus it is imperative to find an optimal ordering that minimizes the BDD size. However, the optimal ordering problem is shown to be computationally hard (Bollig & Wegener 1996, Tani, Hamaguchi & Yajima 1993, Bollig 2014). In practice, domain specific heuristics (Butler, Ross, Kapur & Mercer 1991, Chung, Hajj & Patel 1993, Lindenbaum, Markovitch & Rusakov 1999, Grumberg, Livne & Markovitch 2003) are proposed for an efficient ordering. We apply a different heuristic which is based on our observation specifically about the overlap of the reachable set with unsafe set.

Neural network based analysis: Multiple neural network based frameworks for learning the dynamics or their properties are also being proposed (Long, Lu, Ma & Dong 2018, Raissi, Perdikaris & Karniadakis 2018, Chen, Rubanova, Bettencourt & Duvenaud 2018), which further underlines the need of an efficient and structured state space exploration mechanism. In the model checking domain, neural networks have been used for state classification (Phan, Paoletti, Zhang, Grosu, Smolka & Stoller 2018) as well as reachability analysis by synthesizing barrier certificates (Zhao, Zeng, Chen & Liu 2020), learning state density distribution (Meng, Sun, Qiu, Waez & Fan 2021) or reachability function in NeuReach (Sun & Mitra 2022). In contrast, we exploit neural networks to learn sensitivity functions which we use to perform guided state space exploration.

CHAPTER 2: TECHNICAL PRELIMINARIES

We denote the set of Boolean values as $\mathbb{B}$. States and vectors are elements in $\mathbb{R}^n$ are denoted as x and v. The Inner product of two vectors $v_1, v_2 \in \mathbb{R}^n$ is denoted as $v_1^T v_2$. For $v \in \mathbb{R}^n$, $\|v\|$ denotes the standard Euclidean norm of the vector v. $|\cdot|$ operator denotes the cardinality of a given set. A linear constraint ϕ denotes a half-space in $\mathbb{R}^n$ is a mathematical expression $a^T x \leq \mathbf{b}$, where $a \in \mathbb{R}^n$ is vector of coefficients and $\mathbf{b} \in \mathbb{R}$ is a constant. Negation of a linear constraint $\phi \triangleq a^T x \leq \mathbf{b}$ is another linear constraint $\neg\phi \triangleq (-a)^T x \leq -(\mathbf{b} + \delta)$, where $0 < \delta \ll 1$. Given a sequence $seq = s_1, s_2, \ldots$, the i^{th} element in the sequence is denoted as $seq[i]$. Decision variables are elements in a set $\mathbb{B}$ of binary values 0 and 1. For $\delta \geq 0$, $B_\delta(x) \triangleq \{x' \in \mathbb{R}^n \mid \|x - x'\| \leq \delta\}$ is the closed neighborhood around x of radius δ.

2.1 Affine Linear Hybrid System with Constant Inputs

Definition 1 *An n-dimensional* time-invariant affine linear system with constant inputs $\mathcal{F}(X) \triangleq \langle \mathcal{A}, \mathcal{B} \rangle$ *is denoted as $\dot{x}(t) = \mathcal{A}x(t) + \mathcal{B}$ where:*

$X \subseteq \mathbb{R}^n$ *is the state space of the behaviors,*

$\mathcal{A} \in \mathbb{R}^{n \times n}$ *is the time-invariant dynamics matrix,*

$\mathcal{B} \in \mathbb{R}^{n \times 1}$ *is the time-invariant constant affine matrix.*

The closed form expression for its trajectory is given as $\xi(t) = e^{\mathcal{A}t}\xi(0) + \int_0^t e^{\mathcal{A}(t-\mu)}\mathcal{B}d\mu$, where $\xi(0) \doteq x_0 \in X$ is called the initial state. the system $\dot{x}(t) = \mathcal{A}x(t)$ with no inputs is called as the *autonomous* system.

Definition 2 *An n-dimensional* affine linear hybrid system $\mathcal{H}$ *modelled as a hybrid automaton is defined to be a tuple $\langle L, X, F, I, T, G \rangle$ where:*

L *is a finite set of locations (also called modes),*

$X \subseteq \mathbb{R}^n$ *is the state space of the behaviors,*

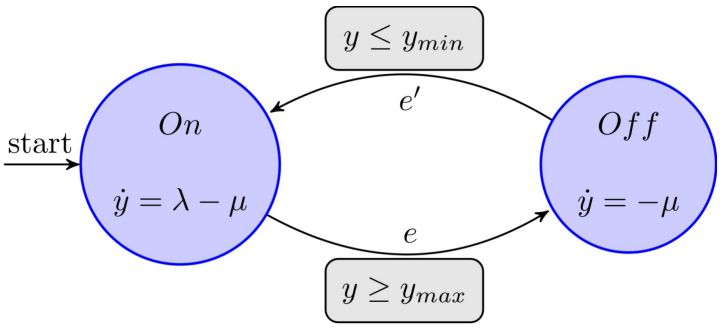


Figure 2.1: Hybrid automaton of a tank system.

$F : L \to \mathcal{F}(X)$ *assign a differential equation* $\dot{x}(t) = \mathcal{A}_l x(t) + \mathcal{B}_l$ *for location* l,

$I : L \to 2^{\mathbb{R}^n}$ *assigns an invariant set for each location of the hybrid system,*

$E \subseteq L \times L$ *is the set of discrete transitions,*

$G : E \to 2^{\mathbb{R}^n}$ *defines the set of states where a discrete transition is enabled.*

For a linear hybrid system, the invariants and guards are given as the conjunction of linear constraints.

The *initial set of states* Θ is a subset of $L \times 2^{\mathbb{R}^n}$, where second element in the pair is a conjunction of linear constraints. An *initial state* q_0 is a pair (L_0, x_0), such that $x_0 \in X$, and $(L_0, x_0) \in \Theta$.

Example 1 *The hybrid automaton for a tank system is described in Fig. 2.1.* y *is the water level in the tank,* λ *is the pump-on inflow, and* μ *is the outflow. The goal is to prevent the tank from emptying or filling up beyond some thresholds. The model has two discrete modes -* On *and* Off *for pump- on and off respectively. The continuous variable* y *is driven by different differential equations in different modes. There are two discrete transitions* e *and* e'*, and their respective guards are* $y \geq y_{max}$ *and* $y \leq y_{min}$*. Finally, the reset function for both transitions is* $y' = y$ *where* y *is the state of the system before transition and* y' *is the state after taking the transition.*

Definition 3 *Given a hybrid system* $\mathcal{H}$ *and an initial set of states* Θ*, an execution of* $\mathcal{H}$ *is a sequence of trajectories and transitions* $\xi_0 e_1 \xi_1 e_2 \ldots$ *such that (i) the first state of* ξ_0 *denoted as* q_0 *is in the initial set, i.e.,* $q_0 = (L_0, x_0) \in \Theta$*, (ii) each* ξ_i *is the solution of the differential equation of the corresponding location* L_i*, (iii) all the states in the trajectory* ξ_i *respect the invariant of the location* L_i*, and (iv) the state of the trajectory before each transition* e_i *satisfies guard* $G(e_i)$*.*

The set of states encountered by all executions that conform to the above semantics is called the *reachable set* denoted as $Reach_{\mathcal{H}}(\Theta)$ (or $Reach(\Theta)$ when it is clear from the context). A linear dynamical system can be considered as a hybrid system with one mode. We use the simulation engine that is described in (Bak & Duggirala 2017*b*) to generate system executions. This simulation engine also accounts for non-determinism induced due to discrete transitions. The closed form expression of a linear dynamical system execution involves matrix exponential; thus, we are better off using simulation engine that generates simulation as a proxy for an execution. For a unit time (also called the step), the hybrid system simulation starting from state q_0 is denoted as $\xi_{\mathcal{H}}(q_0)$.

Definition 4 *A sequence $\xi_{\mathcal{H}}(q_0) \doteq q_0, q_1, q_2, \ldots$, where each $q_i = (L_i, x_i)$, is a (q_0)-simulation of the hybrid system $\mathcal{H}$ with initial set Θ if and only if $q_0 \in \Theta$ and each pair (q_i, q_{i+1}) corresponds to either: (i) a continuous trajectory in location L_i with $L_i = L_{i+1}$ such that a trajectory starting from x_i would reach x_{i+1} after exactly unit time with $x_i \in I(L_i)$, or (ii) a discrete transition from L_i to L_{i+1} (with $L_{i-1} = L_i$) where $\exists e \in E$ such that $x_i = x_{i+1}$, $x_i \in G(e)$ and $x_{i+1} \in I(L_{i+1})$. Bounded-time variants of these simulations, with time bound T, are called (q_0, T)-simulations.*

If the pair (q_i, q_{i+1}) corresponds to a continuous trajectory, q_{i+1} is called the continuous successor of q_i, otherwise q_{i+1} is the discrete successor of q_i.

While talking about the continuous or discrete behaviors of simulations, we abuse notation and use x_i, the continuous component of the state instead of q_i.

Observations On Simulation Algorithm: We would like to make a few observations regarding the simulation algorithm that we have presented. First, the simulation engine allows the execution to make a discrete transition even when the invariant is violated. That is, if x_i and x_{i+1} are two successive states in the simulation, x_{i+1} can make a discrete transition to the new mode even when $x_{i+1} \notin I(L_i)$ as long as $x_{i+1} \in G(e)$. This is necessary to handle the common case where a guard is the complement of an invariant, and a sampled simulation jumps over the guard boundary during a single step. If these types of behaviors are not desired, the guard can be explicitly strengthened with the invariant of the originating mode.

If a guard is enabled and the invariant is still true, or if multiple guards are enabled, the simulation engine can make a non-deterministic choice. Consider that a one-dimensional system has two locations l_1 and l_2 such that $F(l_1) : \dot{x} = 1$, $I(l_1) : x \in [0, 50]$, transition $e = (l_1, l_2)$, and $G(e) : x \geq 45$. The

initial set is $\Theta \triangleq (l_1, x \in [0, 5])$. After the guard is enabled in l_1 i.e., $x \geq 45$, the simulation engine, in a non-deterministic manner, can either take a discrete transition to l_2 or continue evolving in l_1 as long as its invariant is true. At $x = 50$, the trajectory can no longer continue to stay in l_1 as the invariant will be violated. Hence, at $x = 50$, the engine is forced to take the transition to l_2.

Second, the simulation engine given in Definition 4 does not check if the invariant is violated for the entire time interval, but only at a discrete time instance. Computationally, it is very hard to give certainty about whether a predicate was satisfied during an entire time interval, and hence we consider this to a valid assumption. Readers familiar with industrial simulation engines can relate this to a feature of not detecting *zero crossings*.

Third, the discrete jumps are only enabled at time instances that are multiples of the unit time. For discrete transitions that are a result of change in controller input that is driven by software, such an assumption is valid as one can consider the control system providing actuation values at discrete instances of time. This notion might not accurately represent the discrete transitions that are a result of environmental impact such as impulse responses. However, we still argue that such a notion of execution is useful because of two reasons. First, it is impossible (except for some very specific cases) to finitely represent the execution trace when the discrete transition is a result of the environment. The closest we can get to such representation is to consider executions that are defined in Definition 4. Second, by reducing the time step, one can get arbitrarily close to the execution that is a result of impulse response.

Finally, in order to avoid Zeno executions, the simulation engine forces the system should spend at least unit time in each mode.

2.2 Metric Temporal Logic

Metric Temporal Logic (MTL) is defined over a finite set $\mathcal{P}$ of atomic propositions. Each proposition $p \in \mathcal{P}$ at discrete time $t \in \mathbb{N}$ takes a value from the boolean set $\{\top, \bot\}$. A timed word is defined as $\omega : \mathbb{N} \to 2^{\mathcal{P}}$, where $\omega[t] \in 2^{\mathcal{P}}$ is the set of propositions that are true at time t (Sadraddini & Belta 2016). The syntax of MTL formulas is recursively defined as:

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \varphi_1 \wedge \varphi_2 \mid \varphi_1 \mathrm{U}_\tau \varphi_2 \tag{2.1}$$

where $\neg$ and $\wedge$ are boolean negation and conjunction operators, respectively, and U is the timed *until* operator with a time interval $\tau \subseteq [t_1, t_2]$ where $0 \leq t_1 \leq t_2 \leq \mathrm{T}$ and $\mathrm{T} \in \mathbb{N}^+$ is the time bound. Other temporal operators are constructed using the syntax above. The temporal finally (*eventually*) is defined as $\Diamond_\tau \varphi := \top \mathrm{U}_\tau \varphi$ and temporal globally (*always*): $\Box_\tau \varphi := \neg \Diamond_\tau \neg \varphi$. Word ω satisfies MTL formula φ, denoted by $\omega \models \varphi$, if $\sigma_0 \models \varphi$, where σ_0 is a timed sequence "$\omega[0], \omega[1], \ldots$" starting at time 0. The language of φ is the set of all words satisfying φ. The semantics for MTL are inductively defined and can be referred in (Thati & Roşu 2005).

We consider specifications described using MTL with each of its atomic propositions is over a set of linear constraints

$$p := A_p x \leq b_p$$

where $A_p \in \mathbb{R}^{|p| \times n}$, $b_p \in \mathbb{R}^{|p|}$, and $|p|$ is the number of constraints in proposition p.

2.3 Safety and Counterexamples

We next define the safety property for simulations and for a set of initial states (from (Bak & Duggirala 2017*b*)).

Definition 5 *A given simulation $\xi_{\mathcal{H}}(q_0)$ is said to be safe with respect to an unsafe set $\Psi \subseteq \mathbb{R}^n$ if and only if $\forall q_i \doteq (L_i, x_i) \in \xi_{\mathcal{H}}(q_0)$, $x_i \notin \Psi$. An unsafe simulation is called a counterexample. Safety for bounded time simulations are defined similarly.*

Definition 6 *A hybrid system $\mathcal{H}$ with initial set Θ, time bound T, and unsafe set $\Psi \subseteq \mathbb{R}^n$ is said to be safe with respect to its simulations if all simulations starting from Θ for bounded time T are safe.*

For computing system simulations of interest, we use the simulation equivalent reachable set approach that is presented in (Duggirala & Viswanathan 2016, Bak & Duggirala 2017*b*).

We drop the subscript $\mathcal{H}$ from $\xi_{\mathcal{H}}$ as the work in this paper refers to the hybrid setting.

Remark 1 *Although we focus on safety specification in our counterexample generation because we are dealing with safety critical systems, Ψ can denote the violation of a general performance specification, such as an overshoot in the regulation control, an undesirable maneuver in an autonomous car, or a failure site in hardware design etc.*

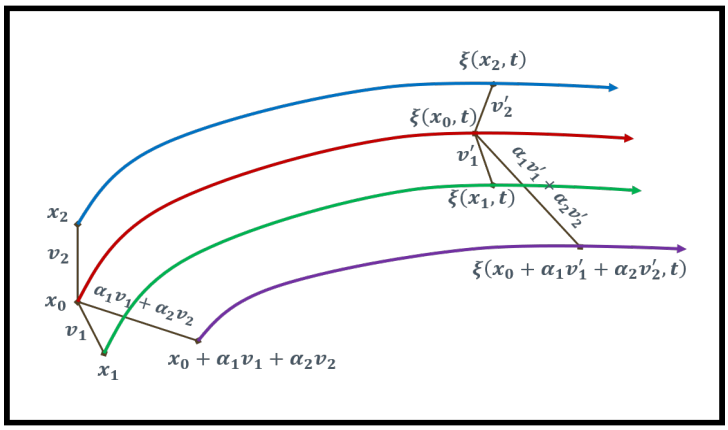


Figure 2.2: Illustration of the Superposition principle.

2.4 Reachable Set Computation

We now present some of the building blocks in computation of the reachable set (from (Bak & Duggirala 2017*b*)). There are three main aspects of the reachable set computation. First is the superposition principle, second is the generalized star representation that is used for representing the set of reachable states and finally, the reachable set algorithm for a single mode and the simulation-equivalent reachable set that is returned by Algorithm in (Bak & Duggirala 2017*b*).

Definition 7 *Given any initial state x_0, vectors $v_1, \ldots, v_m$ where $v_i \in \mathbb{R}^n$, scalars $\boldsymbol{\alpha_1}, \ldots, \boldsymbol{\alpha_m}$, the trajectories of linear differential equations in a given location l always satisfy*

$$\xi(x_0 + \Sigma_{i=1}^{m} \boldsymbol{\alpha_i} v_i, t) = \xi(x_0, t) + \Sigma_{i=1}^{m} \boldsymbol{\alpha_i} (\xi(x_0 + v_i, t) - \xi(x_0, t))$$

An illustration of the superposition principle for two vectors is shown in Figure 2.2. We exploit the superposition property of linear systems in order to compute the simulation-equivalent reachable set of states for a linear hybrid system. Before describing the algorithm for computing the reachable set, we introduce the data structure called a *generalized star* that is used to represent the reachable set of states.

Definition 8 *A* generalized star (or simply star) *S is a tuple $\langle c, V, P \rangle$ where $c \in \mathbb{R}^n$ is called the* center, *$V = \{v_1, v_2, \ldots, v_m\}$ is a set of m ($\leq n$) vectors in $\mathbb{R}^n$ called the* basis vectors, *and $P : \mathbb{R}^n \rightarrow \{\top, \bot\}$ is a predicate.*

A generalized star S defines a subset of $\mathbb{R}^n$ as follows.

$$[\![S]\!] \triangleq \{x \mid \exists \bar{\alpha} = [\boldsymbol{\alpha_1}, \ldots, \boldsymbol{\alpha_m}]^T \text{ such that } x = c + \Sigma_{i=1}^{m} \boldsymbol{\alpha_i} v_i \text{ and } P(\bar{\alpha}) = \top\}$$

Sometimes we will refer to both S and $[\![S]\!]$ as S. Additionally, we refer to the variables in $\bar{\alpha}$ as basis variables and the variables x as orthonormal variables. Given a valuation $\bar{\boldsymbol{\alpha}}$ of the basis variables, the corresponding orthonormal variables are denoted as $x = c + V \times \bar{\alpha}$.

Similar to (Bak & Duggirala 2017*b*), we consider predicates P which are conjunctions of linear constraints. This is primarily because linear programming is very efficient when compared to nonlinear arithmetic. We therefore harness the power of these linear programming algorithms to improve the scalability of our approach.

Example 2 *Consider a set $\Theta \subset \mathbb{R}^2$ given as Θ[1] $\triangleq \{(x_1, x_2) \,|\, x_1 \in [4, 6], x_2 \in [4, 6]\}$. The given set Θ can be represented as a generalized star in multiple ways. One way of representing the set is given as $\langle c_0, V_0, P_0 \rangle$ where $c_0 \doteq (5, 5)$, $V_0 \doteq \{[0, 1]^T, [1, 0]^T\}$ and $P_0 \doteq -1 \leq \boldsymbol{\alpha_1} \leq 1 \wedge -1 \leq \boldsymbol{\alpha_2} \leq 1$. That is, the set Θ is represented as a star with center $(5, 5)$ with vectors as the orthonormal vectors in the Cartesian plane and predicate where the components along the basis vectors are restricted by the set $[-1, 1] \times [-1, 1]$.*

Simulation-Equivalent Reachable Set for Linear Dynamical Systems: We briefly describe the algorithm for computing simulation-equivalent reachable set for a linear dynamical system (or, a hybrid system with one mode). This is primarily done to present some crucial observations which will later be used in the algorithms for generating specific counterexamples. Longer explanation and proofs for these observations and algorithms is available in prior work (Duggirala & Viswanathan 2016, Bak & Duggirala 2017*b*).

At its crux, the algorithm exploits the superposition principle of linear systems and computes the reachable states using a generalized star representation. For an n-dimensional system, this algorithm requires at most $n+1$ simulations. Given an initial set $\Theta \triangleq \langle c_0, V_0, P_0 \rangle$ with $V = \{v_1, v_2, \ldots, v_m\} (m \leq n)$, the algorithm performs a simulation starting from c_0 (denoted as $\xi(c_0, 0)$), and $\forall 1 \leq j \leq m$, performs a simulation from each $c_0 + v_j$ (denoted as $\xi(c_0 + v_j, 0)$). For a given time instance $i \geq 0$, the reachable

[1]We abuse the notation Θ to denote the initial set as well as its star representation.

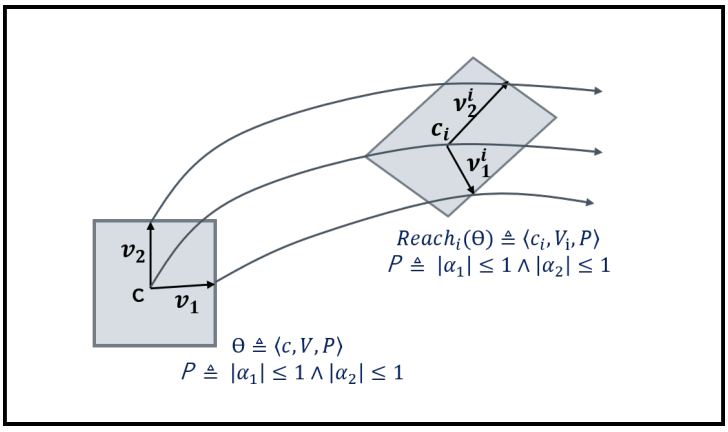


Figure 2.3: Reachable set computation using simulations and generalized star

set is defined as $\langle c_i, V_i, P_0 \rangle$ where $c_i = \xi(c_0, i)$ and $V_i = \langle v_1^i, v_2^i, \ldots, v_m^i \rangle$ where $\forall 1 \leq j \leq m, v_j^i = \xi(c_0 + v_j, i) - \xi(c_0, i)$. Notice that the predicate does not change for the reachable set, but only the center and the basis vectors are changed.

An illustration of this reachable set computation is shown in Figure 2.3. Here, as the system is 2-dimensional, a total number of three simulations are performed - one from *center* c_0, and one from each $c_0 + v_1$ and $c_0 + v_2$. The reachable set after time i is given as the star with center $c_i = \xi(c_0, i)$, basis vectors $v_1^i = \xi(c_0 + v_1, i) - \xi(c_0, i)$ and $v_2^i = \xi(c_0 + v_2, i) - \xi(c_0, i)$, and the same predicate P as given in the initial set.

The reachable set $Reach(\Theta)$ for a given linear dynamical system computed in this manner is a sequence (with its first element Θ) of generalized stars such that $Reach_i(\Theta) \doteq S_i \triangleq \langle c_i, V_i, P_0 \rangle$ is the set of states visited at a discrete time step i (Duggirala & Viswanathan 2016). Each subsequent element is the *successor* that corresponds to the system evolution after unit time step.

Simulation-Equivalent Reachable Set for Hybrid Systems with Linear Dynamics: The Algorithm presented in (Duggirala & Viswanathan 2016) has been extended in (Bak & Duggirala 2017*b*) as computeSimEquivReach, to compute the simulation equivalent reachable set for hybrid systems that accommodates for the invariants in each mode and the guard transitions for discrete mode jumps. This is achieved by introducing a new technique called *invariant constraint propagation* and *dynamic aggregation and de-aggregation*. Owing to our focus on generating interesting counterexamples, we apply fully-deaggregated version of the reachable set computation algorithm where each element in the reachable set is given as a generalized star.

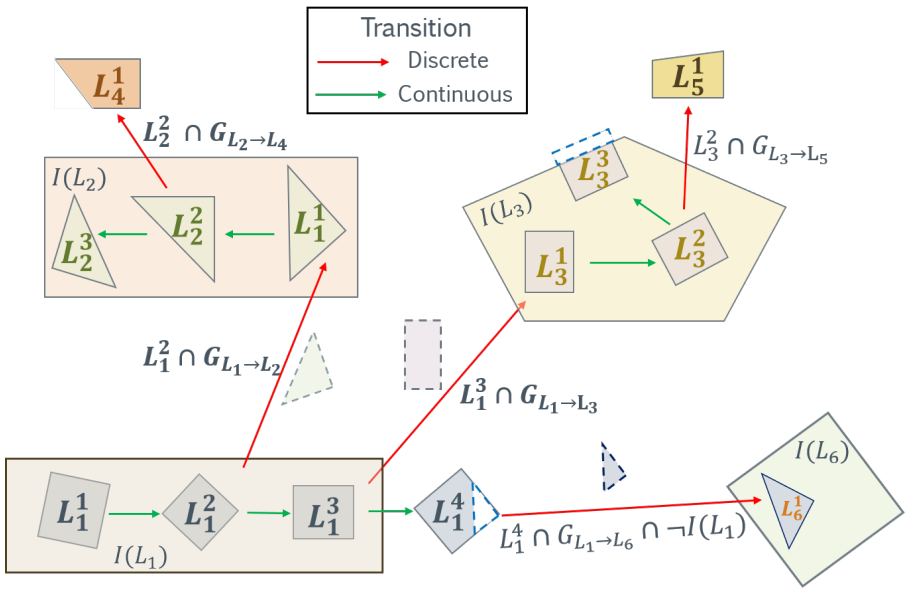


Figure 2.4: Illustration of $ReachTree$ construction.

Remark 2 (Propagation) *For a discrete transition $e_i \doteq L_i \to L_{i+1}$, a set of constraints $\bar{P}$ are propagated from a star $S_i \in L_i$ to $S_{i+1} \in L_{i+1}$ via guard $G(e_i)$ iff*

$$\bar{P} \doteq S_i \cap G(e_i) \neq \emptyset, \text{ and } \bar{P} \subseteq S_{i+1}$$

As a consequence of this propagation, the initial set for location L_{i+1} after the discrete transition e_i is the intersection of the set S_i with $G(e_i)$.

The algorithm computeSimEquivReach returns the reachable set in the form of a tree. The root node of the tree is the initial set Θ. Each node in this tree is a generalized star S_i of the form $S_i \triangleq \langle c_i, V_i, P_i \rangle$ corresponding to the set of states visited at a discrete step i. Notice that the predicate in S_i might be different from the predicate of the initial set Θ so as to accommodate the mode invariants and discrete transitions induced due to hybrid behavior. Each node in reach tree can have at most one *continuous successor* that corresponds to the evolution for unit time in the same mode, and have multiple *discrete successors* each corresponding to the reachable set after a discrete transition. We denote this tree form of the reachable set as $ReachTree$.

The construction of $ReachTree$ is illustrated in Figure 2.4. The part of the system shown has 6 modes (or locations) - L_1, L_2, L_3, L_4, L_5, and L_6. The invariants for the modes L_1, L_2, L_3 are $I(L_1), I(L_2), I(L_3)$ respectively. There are 4 nodes corresponding to mode L_1 where L_1^{i+1} is the

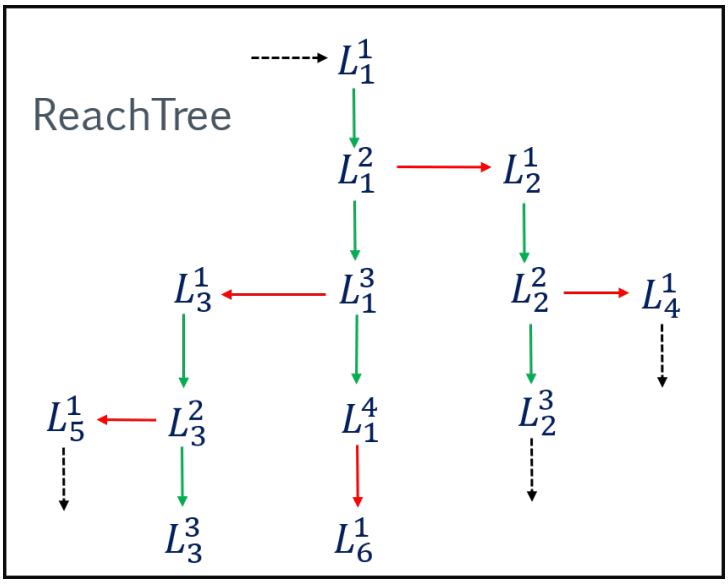


Figure 2.5: Representation of a $ReachTree$.

continuous successor of L_1^i, $1 \leq i \leq 3$. L_1^1 itself can be the root node or a successor - continuous or discrete - to some another node. A discrete transition $(L_i \rightarrow L_j)$ from mode L_i to mode L_j is active when its associated guard $(G_{L_i \rightarrow L_j})$ becomes enabled, and constraints $L_i \cap G_{L_i \rightarrow L_j}$ are propagated (Remark 2). Hence, during the transition from L_1^2 to L_2^1, predicates denoting the set $L_1^2 \cap G_{L_1 \rightarrow L_2}$ are propagated. It means that the initial set L_2^1 is the intersection of the set L_1^2 and the associated guard $G_{L_1 \rightarrow L_2}$.

As our reachable set construction algorithm explores all possible transitions, a node has as many discrete successors as the number of active discrete transitions, in addition to having at most one continuous successor. This behavior translates into 3 scenarios: 1) only continuous-, 2) only discrete-, 3) continuous- as well as discrete- successors. For instance, L_3^2 has one continuous and 2 discrete successors as it satisfies the invariant $I(L_3)$, and it has active transitions to both L_5 and L_6. L_3^1 does not have any discrete successor because there is no active discrete transition from L_3^1. In a similar fashion, L_1^4 has just one successor which is discrete because L_1^4 violates $I(L_1)$ but $G_{L_1 \rightarrow L_6}$ is enabled. The $ReachTree$ constructed in this manner is shown in Figure 2.5. The dashed transitions denote that there may or may not be a transition.

Definition 9 *Consider an initial set* Θ*, bound* T*, and the simulation equivalent reachable set represented as* $ReachTree$*. Given a star* $S_i \in ReachTree$*, we call a sequence of stars* $\sigma =$

$R_1, R_2, \ldots, R_m$ *a* chain *starting from* S_i *if and only if* $R_1 = S_i$ *and* $\forall 2 \leq j \leq m, R_j$ *is (either continuous or discrete) successor of* R_{j-1}.

Remark 3 *Given a star* $S_i \triangleq \langle c_i, V_i, P_i \rangle$ *in* $ReachTree$ *and its successor (either discrete or continuous)* $S_{i+1} \triangleq \langle c_{i+1}, V_{i+1}, P_{i+1} \rangle$*, observe that one has to either perform intersection with the invariant or with the guards for obtaining the predicate* P_{i+1}*. Hence* $P_{i+1} \subseteq P_i$*. Thus, given a valuation of* $\bar{\alpha}$ *such that* $P_{i+1}(\bar{\alpha}) = \top$*, it is true that all the stars that are the parents of* P_{i+1}*, the valuation of* $\bar{\alpha}$ *is contained in the predicate. Additionally, one can use this valuation of basis variables to generate the trace starting from the initial set* Θ *to* P_{i+1}*. We call the procedure that generates this execution as* $getExecution(\bar{\alpha}, S_{i+1}, ReachTree)$.

A side effect of the above observation is that all the trajectories that reach the star $S_{i+1} \triangleq \langle c_{i+1}, V_{i+1}, P_{i+1} \rangle$ *would originate from the subset* Θ' *of the initial set, where* $\Theta' \triangleq \langle c_0, V_0, P_{i+1} \rangle$.

Assumptions: Similar to the assumptions in earlier work (Bak & Duggirala 2017*b*), we assume that ODE solvers give the exact result. While theoretically unsound, such an assumption is adopted due to its practicality. Second, we use floating-point arithmetic in our computations and do not track the errors by floating point arithmetic. A user concerned about the inaccuracy of numerical simulation can either use validated simulations (e.g., Computer Assisted Proofs in Dynamic Groups (CAPD) library[2]) or compute the linear ODE solution as a matrix exponential to an arbitrary degree of precision. The algorithms presented are oblivious to the simulation engine used. We assume the initial set and unsafe region to be convex polytopes. However, generalized star provides flexibility to compute the reachable set even when the initial set is non-convex (Duggirala & Viswanathan 2016).

2.5 Constraint Propagation for Counterexamples

We briefly discuss how we can make use of constraint propagation to obtain desired executions. In particular, we focus on system executions that reach an unsafe set $\Psi \subseteq \mathbb{R}^n$ at time step i. That is, for a set Ψ specified as an STL/MTL formula, we need to find the valuations of basis variables $\bar{\alpha}$ such that $P(\bar{\alpha}) = \top$ and $(c_i + V_i \times \bar{\alpha}) \in \Psi$. We begin with computing the reachable set $Reach(\Theta)$ as a sequence of stars $S_0, S_1, \ldots, S_{\mathrm{T}}$ where $S_0 \doteq \Theta$, $Reach_j(\Theta) \doteq S_j \triangleq \langle c_j, V_j, P_0 \rangle$. Now, given star S_i, we represent Ψ

[2]http://capd.ii.uj.edu.pl/index.php

as another star $\langle c_i, V_i, P_i^\Psi \rangle$ by converting each constraint $(a^T x \leq \mathbf{b}) \in \Psi$ as $a^T(c_i + V_i\bar{\alpha}) \leq \mathbf{b}$. This gives us $P_i^\Psi \doteq A^T(c_i + V_i\bar{\alpha}) \leq b$ where $A \in \mathbb{R}^{n\times m}, b \in \mathbb{R}^m$, and m is the number of constraints in P_i^Ψ. We next check feasibility of the predicate $P_0 \wedge P_i^\Psi$. There is no simulation reaching Ψ at time i if $\left(P_0 \wedge P_i^\Psi\right)(\bar{\alpha}) = \perp$. Otherwise, we make use of the previous observation to propagate constraints so that all the executions that reach the unsafe region $S_i \cap U \triangleq \langle c_i, V_i, P_0 \wedge P_i^\Psi \rangle$ at time step i would originate from the set $\Theta_i \triangleq \langle c_0, V_0, P_0 \wedge P_i^\Psi \rangle \subseteq \Theta$. In other words, the predicate $P_0 \wedge P_i^\Psi$ denotes the set of α valuations for the counterexamples reaching Ψ at time step i. Considering $P_0 \doteq (H^T x \leq g)$, we solve a system of constraints $\left(A^T(c_0 + V_0\bar{\alpha}) \leq b \ \wedge \ H^T(c_0 + V_0\bar{\alpha}) \leq g\right)$ to find a satisfiable valuation $\bar{\alpha}$. Then we generate a desired counterexample as a simulation $c_0 + V_0 \times \bar{\alpha}, c_1 + V_1 \times \bar{\alpha}, \ldots$ where $(c_i + V_i \times \bar{\alpha}) \in \Psi$.

The extension of this approach to multiple time steps is straightforward. The executions that reach U at two different time steps i and i' would originate from $(\Theta_i \cap \Theta_{i'}) \triangleq \langle c, V, P \wedge P_i^\Psi \wedge P_{i'}^\Psi \rangle \subseteq \Theta$. We can solve the corresponding system of constraints to obtain a valuation $\bar{\boldsymbol{\alpha}}$ and generate a counterexample that overlaps with Ψ at both time steps i and i'.

2.6 Linear Affine Systems with Bounded Inputs

Most of the discussion in this section is adapted from (Bak & Duggirala 2017*c*) as it is. The reader interested in more details is referred to the original paper.

Definition 10 *An n-dimensional* time-invariant affine linear system with bounded inputs $\mathcal{F}(X) \triangleq \langle \mathcal{A}, \mathcal{B}, U \rangle$ *is denoted as $\dot{x}(t) = \mathcal{A}x(t) + \mathcal{B}u(t); u(t) \in U$ where:*

$X \subseteq \mathbb{R}^n$ *is the state space of the behaviors,*

$\mathcal{A} \in \mathbb{R}^{n\times n}$ *is the time-invariant dynamics matrix,*

$\mathcal{B} \in \mathbb{R}^{n\times m}$ *is the time-invariant affine matrix,*

$U \subseteq \mathbb{R}^{m\times 1}$ *is the set of possible inputs.*

Assuming that the system has m inputs, the input function $u(t)$ is given as $u : \mathbb{R}_{\geq 0} \to \mathbb{R}^m$. If $u(t)$ is an integrable function, the closed form expression for the trajectory of the above system is given as $\xi(u,t) = e^{\mathcal{A}t}\xi(u,0) + \int_0^t e^{\mathcal{A}(t-\mu)}\mathcal{B}u(\mu)d\mu$, where $\xi(u,0) \doteq x_0 \in X$ is called the initial state. If $u(t)$ is

a constant function set to the value of u_0, then the system can be represented as $\langle \mathcal{A}, \mathcal{B}_0 \rangle$ where $\mathcal{B}_0 \doteq \mathcal{B}u_0$ and is an affine linear system with constant inputs (Section 2.1).

We restrict our attention to inputs that are piece-wise constant. That is, the value of inputs are updated periodically with every unit time (also called step size h) and the inputs stay constant for the time duration $[k \times h, (k+1) \times h]$.

Definition 11 *Given an initial state x_0, a sequence of input vectors u, and a time period h, the sequence $\xi(x_0, u) \doteq x_0 \xrightarrow{u_0} x_1 \xrightarrow{u_1} \ldots$, is a (x_0, u)-simulation of above system if and only if all $u_i \in U$, and for each x_{i+1} we have that x_{i+1} is the state of the trajectory starting from x_i when provided with constant input u_i for unit time, $x_{i+1} \doteq \xi(x_i, u_i)$. Bounded-time variants are called (x_0, u, T)-simulations where T is called time bound. For simulations, the unit time is given as the step size h.*

The set of states encountered by a (x_0, u)-simulation is the set of states in $\mathbb{R}^n$ at the multiples of the time step h, $\{x_0, x_1, \ldots\}$. The relationship between x_i and x_{i+1} can be obtained, using the closed form expression of the trajectory, as the following

$$x_{i+1} = e^{\mathcal{A}h} x_i + G(\mathcal{A}, h)\mathcal{B}u_i \tag{2.2}$$

where $G(\mathcal{A}, h) = \sum_{i=0}^{\infty} \frac{1}{(i+1)!} \mathcal{A}^i h^{i+1}$.

Definition 12 *Given an initial set Θ, the simulation-equivalent reachable set for such system is the set of all states that can be encountered by any (x_0, u)-simulation starting from any $x_0 \in \Theta$, for any valid sequence of input vectors u. Time-bounded version of the reachable set is defined for a time bound T.*

As mentioned earlier, the simulation-equivalent reachable set is represented as the sequence of stars $\langle S_0, S_1, S_2, \ldots, S_k \rangle$ where the sets of states that all the simulations starting from $S_0 \doteq \Theta$ can encounter at time instances $i \times h$ is given as S_i. Using Equation 2.2, the relationship between S_{i+1} and S_i for the system with inputs can be expressed as $\Theta_{i+1} \doteq e^{\mathcal{A}h}\Theta_i \oplus G(\mathcal{A}, h)\mathcal{B}U$. Representing $\mathcal{U} = G(\mathcal{A}, h)\mathcal{B}U$ and expanding the above equation, we have

$$S_{i+1} = e^{\mathcal{A}(i+1)\times h} S_0 \oplus e^{\mathcal{A}(i)\times h}\mathcal{U} \oplus e^{\mathcal{A}(i-1)\times h}\mathcal{U} \oplus \ldots \oplus e^{\mathcal{A}\times h}\mathcal{U} \oplus \mathcal{U}. \tag{2.3}$$

where $e^{\mathcal{A}(i+1)\times h}S_0$ is the reachable set of the autonomous system and the remainder of the terms characterize the effect of inputs. Consider the j^{th} term in the remainder, namely, $e^{\mathcal{A}(j)\times h}\mathcal{U}$. This term is exactly same as the reachable set of states starting from an initial set $\mathcal{U}$ after $j \times h$ time units, and evolving according to the autonomous dynamics $\dot{x} = \mathcal{A}x$.

Furthermore, the set $\mathcal{U} = G(\mathcal{A}, h)\mathcal{B}U$ can be represented as a star $\langle c, V, P\rangle$ with m basis vectors, for an n-dimensional system with m inputs. This is done by taking the origin as the center c, the set $G(\mathcal{A}, h)\mathcal{B}$ as the star's $n \times m$ basis matrix V , and using the linear constraints U as the predicate P , replacing each input u_i with α_i. Now, the effect of the inputs after $i \times h$ time units is computed as the Minkowski sum of stars denoted as $\mathcal{U}_i \oplus \mathcal{U}_{i-1} \oplus \mathcal{U}_{i-2} \oplus \ldots \oplus \mathcal{U}_0$ where $\mathcal{U}_i$ is the effect of inputs at i^{th} time step and $\mathcal{U}_0 \doteq \mathcal{U}$.

Definition 13 (Minkowski Sum with Stars) *Given two stars $S = \langle c, V, P\rangle$ with m basis vectors and $S' = \langle c', V', P'\rangle$ with m' basis vectors, their Minkowski sum is a new star $\bar{S} = \langle \bar{c}, \bar{V}, \bar{P}\rangle$ with $m + m'$ basis vectors and (i) $\bar{c} = c + c'$, (ii) $\bar{V}$ is the list of $m + m'$ vectors produced by joining the list of basis vectors of S and S', (iii) $\bar{P}(\bar{\alpha}) = P(\bar{\alpha}_m) \wedge P'(\bar{\alpha}_{m'})$. Here $\bar{\alpha}_m \in \mathbb{R}^m$ denotes the variables in S, $\bar{\alpha}_{m'} \in \mathbb{R}^{m'}$ denotes the variables for S', and $\bar{\alpha} \in \mathbb{R}^{m+m'}$ denotes the variables for $\bar{S}$ (with appropriate variable renaming).*

Notice that both the number of variables in the star and the number of constraints grow with each Minkowski sum operation. Since we focus on bounded piece-wise constant inputs, for m inputs, the number of variables (or basis vectors) grow by m and the number of constraints grow by $2m$ after every unit time step to incorporate the effect of inputs. For instance, the set of states reachable exactly after i time steps is denoted as S_i which would have $n + (i \times m)$ basis vectors and $|P_0| + (i \times 2m)$ constraints where $|P_0|$ is the number of constraints in the initial star S_0.

2.7 Feedback Control Systems

We denote the dynamics of the plant as

$$\dot{x} = f(x, u) \tag{2.4}$$

where x is the state of the system which takes values in $\mathbb{R}^n$, and u is the input which takes values in $\mathbb{R}^m$. We will assume that the system with the controller $u = g(x)$ has unique trajectories starting from any initial state. This existence and uniqueness of trajectories is guaranteed if both f and g are functions with bounded Lipschitz constants.

Definition 14 (Unique Trajectory Feedback Functions) *A feedback function $u = g(x)$ is said to be a* unique trajectory feedback function *if the the initial value problem for the closed loop system $\dot{x} = f(x, g(x))$ is guaranteed to have a unique solution for all initial points $x_0 \in \mathbb{R}^n$.*

Definition 15 (Trajectories of Closed Loop System) *Given a unique trajectory feedback function $u = g(x)$, a trajectory of closed loop system $\dot{x} = f(x, g(x))$, denoted as $\xi_g(x_0, t)$ $(t \geq 0)$, is the solution of the initial value problem of the differential equation $\dot{x} = f(x, g(x))$ with initial condition x_0. We often drop the feedback function g when it is clear from the context.*

We extend the notion of trajectory to include backward time trajectories as well. Given $t > 0$, the backward time trajectory $\xi_g(x_t, -t) = x_0$ such that $\xi_g(x_0, t) = x_t$. We denote backward time trajectory as $\xi^{-1}(x_t, t)$.

Given $x_0, x_t \in \mathbb{R}^n$ and $t > 0$ such that $\xi(x_0, t) = x_t$, then $\xi^{-1}(x_t, t) = x_0$. It is trivial to observe that $\xi^{-1}(\xi(x_0, t), t) = x_0$.

Definition 16 (Sensitivity Functions) *Given an initial state x_0, vector v, and time t, the sensitivity $\Phi(x_0, v, t)$ of the trajectory, is defined as.*

$$\Phi(x_0, v, t) = \xi(x_0 + v, t) - \xi(x_0, t). \tag{2.5}$$

We extend the definition of sensitivity to backward time trajectories, denoted as inverse sensitivity, as

$$\Phi^{-1}(x_t, v, t) = \xi^{-1}(x_t + v, t) - \xi^{-1}(x_t, t). \tag{2.6}$$

The sensitivity functions are visually demonstrated in Figure 2.6. The sensitivity $\Phi(x_0, v, t)$ is the displacement between the respective states that the system reaches at time $t > 0$, when starting from states $x_0 + v$ and x_0 at time $t = 0$. The inverse-sensitivity $\Phi^{-1}(x_0, v, t)$ is the displacement between the states at $t = 0$ that will reach $x_0 + v$ and x_0, respectively, at time $t > 0$. In this work, we will primarily

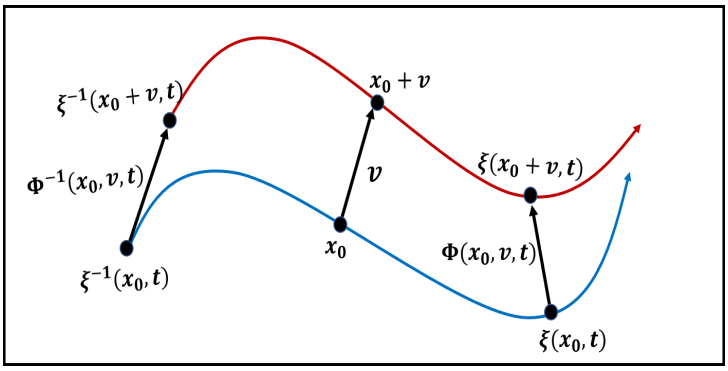


Figure 2.6: Visual description of the sensitivity functions Φ and Φ^{-1}. The blue and red curves, respectively, denote the unique trajectories that pass through the state of interest x_0 and its displaced state $x_0 + v$.

focus on using the inverse sensitivity function Φ^{-1} for performing systematic state space exploration, but an analogous analysis can also be conducted with the sensitivity function Φ.

For general nonlinear differential equations, analytic representation of the trajectories of the ODEs need not exist. If the closed loop system is a smooth function, the analytic representation of its inverse sensitivity function (2.6) can be given by its Taylor expansion

$$\Phi^{-1}(x_0, v, t) = \Phi^{-1}(x_0, 0, t) + \nabla_v \Phi^{-1}(x_0, 0, t)v + \frac{1}{2!} v^t \nabla_v^2 \Phi^{-1}(x_0, 0, t)v + \cdots \tag{2.7}$$

where $\nabla_v \Phi^{-1}$ denote its Jacobian matrix when considered a function only of its second argument v.

When the closed loop dynamics is linear, i.e., $\dot{x} = \mathcal{A}x$, it is easy to observe that $\Phi(x_0, v, t) = e^{\mathcal{A}t}v$, $\Phi^{-1}(x_0, v, t) = e^{-\mathcal{A}t}v$ where $e^{\mathcal{A}t}$ ($e^{-\mathcal{A}t}$) is the matrix exponential of the matrix $\mathcal{A}t$ ($-\mathcal{A}t$). Observe that for linear systems, the inverse sensitivity function is independent of the state x_0.

For nonlinear dynamical systems, one can truncate the infinite series up to a specific order and obtain an approximation. However, for hybrid systems that have state based mode switches, or for feedback functions where the closed loop dynamics is not smooth or is discontinuous, such an infinite series expansion is hard to compute. The central idea for this line of work is to approximate Φ and Φ^{-1} using a neural network and perform state space exploration using such neural networks.

CHAPTER 3: VARIETY OF COUNTEREXAMPLES

In this chapter, we describe the techniques for generating three types of counterexamples namely the deepest, the longest contiguous, absolute longest, and the robust counterexamples. [1] The reader is referred to (Goyal & Duggirala 2018, Goyal & Duggirala 2020*a*) for more details about these evaluations.

We give the definitions of these executions as follows.

Definition 17 *Given a hybrid system $\mathcal{H}$ with an initial set Θ, time bound T, unsafe set $\Psi \subseteq \mathbb{R}^n$, and direction $d \in \mathbb{R}^n$, the depth of a counterexample ξ in direction d is denoted as $depth(\xi, d) \triangleq d^T \cdot \arg\max_{x_i}\{d^T x_i \mid x_i \in \xi \wedge x_i \in \Psi\}$.*

The counterexample ξ with the maximum value of depth is called the deepest counterexample.

Definition 18 *Given a hybrid system $\mathcal{H}$ with an initial set Θ, time bound T, and unsafe set $\Psi \subseteq \mathbb{R}^n$, a counterexample ξ is said to be of* length *l if and only if $\exists$ consecutive states $x_i, x_{i+1}, \ldots, x_{i+l-1}$ in ξ such that $\forall i \leq j \leq i+l-1, x_j \in \Psi$.*

The counterexample of the maximum length *is called the longest contiguous counterexample.*

Definition 19 *Given a hybrid system $\mathcal{H}$ with initial set Θ, time bound T, and unsafe set $\Psi \subseteq \mathbb{R}^n$, a counterexample ξ starting from x_r is said to be* robust *with robustness δ if and only if $\forall x \in B_\delta(x_r) \triangleq \{\bar{x} \mid \|\bar{x} - x_r\| \leq \delta\}$, there exists at least one unsafe execution starting from x.*

Above definition states that any initial state within δ distance from x_r has at least one unsafe execution starting from it. The existential quantifier is introduced because of multiple active discrete transitions originating from same mode. Two executions starting from same initial state can be different

[1] Contents of this chapter previously appeared in preliminary form in the following papers:

Goyal, Manish and Parasara Sridhar Duggirala. 2018. On Generating a Variety of Unsafe Counterexamples for Linear Dynamical Systems. In *Proceedings of 6th IFAC Conference on Analysis and Design of Hybrid Systems.*

Goyal, Manish and Parasara Sridhar Duggirala. 2020. Extracting counterexamples induced by safety violation in linear hybrid systems. *Automatica*.

Goyal, Manish, David Bergman and Parasara Sridhar Duggirala. 2020. Generating Longest Counterexample: On the Cross-roads of Mixed Integer Linear Programming and SMT. In *American Control Conference.*

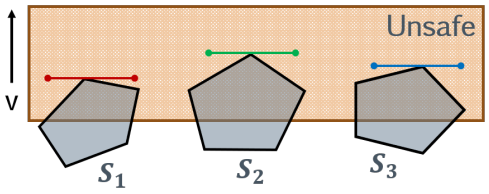


Figure 3.1: Illustration of the deepest counterexample in the direction of v.

if they correspond to different discrete transitions. That is, one execution can be safe while another is unsafe, where only the unsafe execution is used for computing the robust counterexample. If $\delta_1 < \delta_2$, then the robustness δ_2 of a counterexample trivially implies the robustness δ_1. Note that the robust counterexample may not be unique and is dependent on how δ is defined.

3.1 Deepest Counterexample

We now present the algorithm that would return the deepest counterexample for a safety specification and a direction. We illustrate the way to obtain the deepest counterexample using Figure 3.1.

Suppose that in the *ReachTree* computation, there are three stars S_1, S_2, and S_3 that overlap with the unsafe set Ψ. Given a direction $d \in \mathbb{R}^n$, the procedure to compute the deepest counterexample would be the following. 1. For each of the stars S_i, compute the maximum depth $depth_i$ of star S_i as $\max_x (d^T x)$ such that $x \in (S_i \cap \Psi)$. 2. Select the star S_j with maximum value of $depth_j$. 3. Extract the corresponding value of basis variables $\bar{\alpha}$ which achieves the maximum depth and generate the corresponding execution. The correctness of the algorithm trivially follows from Definition 17 and the correctness of the simulation-equivalent reachable set. The algorithm is presented formally in Algorithm 1.

The main loop in lines 2-12 iterates through all the stars in the reachable set given as $ReachTree$ and selects the stars that overlap with the unsafe set Ψ. The optimization problem for maximizing the value of the cost function $d^T x$ for the overlap with the unsafe set is formulated and solved in line 4. If the depth computed in line 4 is greater than the current maximum value (lines 6- 10), then the maximum value is updated and the value of basis variables corresponding to the optimal solution as well as the current star are stored. In line 14, the execution corresponding to the maximum depth is extracted using the value of $\bar{\alpha}_{max}$.

input : Simulation equivalent reachable tree $ReachTree$, direction d and unsafe set Ψ
output : Counterexample ce with maximum depth in direction d in the unsafe set Ψ

$depth_{max} \leftarrow -\infty$; $depthStar \leftarrow \perp$; $ce \leftarrow \perp$;
for *each star* S_i *in* $ReachTree$ **do**
  if $S_i \cap \Psi \neq \emptyset$ **then**
    $\bar{x} \leftarrow \arg\max_x d^T x$ given $x \in (S_i \cap \Psi)$;
    $depth_i \leftarrow d^T \bar{x}$;
    if $depth_i > depth_{max}$ **then**
      $depth_{max} \leftarrow depth_i$;
      $\bar{\alpha}_{max} \leftarrow getBasisVariables(\bar{x})$;
      $depthStar \leftarrow S_i$;
    end if
  end if
end for
if $depth_{max} \neq -\infty$ **then**
  $ce \leftarrow getExecution(\bar{\alpha}_{max}, ReachTree)$;
end if
return ce;

Algorithm 1: Algorithm which computes the deepest counterexample for a given direction d.

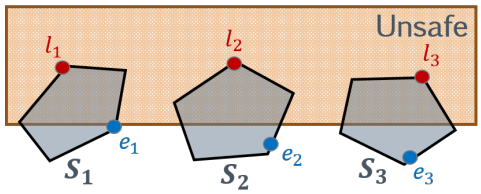


Figure 3.2: Illustration of the longest counterexample.

Analysis: If m is the number of stars overlapping with the unsafe set, we perform linear program optimization for each of these stars to obtain respective depth. Hence, the run time complexity for computing the deepest counterexample is $O(m)$.

3.2 Longest Contiguous Counterexample

We now describe the algorithm for obtaining the counterexample that spends the longest contiguous time in the unsafe set. For this purpose, we make use the observation provided in Section 2.5.

We illustrate the problem of finding the longest contiguous counterexample through Figure 3.2. Consider three consecutive stars S_1, S_2, and S_3 in the reachable set having overlap with the unsafe set as shown. If one picks the state $e_1 \in S_1$, then the *post* states of e_1, denoted as e_2 and e_3, do not lie in the unsafe set. However, if one picks $l_1 \in S_1$, then its *post* states, l_2 and l_3, lie in the unsafe set.

The key insight behind the generation of longest contiguous counterexample is that one has to select the *appropriate* state which visits the maximum number of contiguous overlaps between the unsafe set and the reachable set. In this instance, any state $x_1 \in S_1$ where $x_1 \in S_1 \cap \Psi$, with its successors x_2, x_3 such that $x_2 \in S_2 \cap \Psi$ and $x_3 \in S_3 \cap \Psi$ is the appropriate choice.

For finding such a state, we perform constraint propagation as explained in Section 2.5. Denote the sequence of stars as $\sigma \doteq S_1, S_2, S_3$. The objective is to identify a set of constraints, P_σ, so that $\forall \bar{\boldsymbol{\alpha}}$ such that $P_\sigma(\bar{\alpha}) = \top$, we have, $x_1 = c_1 + V_1 \times \bar{\alpha} \in (S_1 \cap \Psi)$, $x_2 = c_2 + V_2 \times \bar{\alpha} \in (S_2 \cap \Psi)$, and $x_3 = c_3 + V_3 \times \bar{\alpha} \in (S_3 \cap \Psi)$. We first convert the unsafe set U into the center and basis vectors of each of the stars S_1, S_2, and S_3. Thus, $S_i \cap \Psi \triangleq \langle c_i, V_i, P_i \wedge Q_i \rangle$, $1 \leq i \leq 3$. From Remark 3, we know that the set of states that reach $\langle c_i, V_i, P_i \wedge Q_i \rangle$ originate from $\langle c_0, V_0, P_i \wedge Q_i \rangle$. Hence, the set of states that would visit all the intersections of the unsafe set should originate from $\langle c_0, V_0, P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_3 \wedge Q_3 \rangle$. It follows that if the set of constraints $P_1 \wedge Q_1 \wedge P_2 \wedge Q_2 \wedge P_3 \wedge Q_3$ is satisfiable, then the trajectory corresponding to the basis variables that satisfy these constraints visits the unsafe set at all three consecutive time instances.

Building on the above discussion, the algorithm to compute the longest contiguous counterexample would iterate as follows. In each iterations, we consider the contiguous sequences of stars $\sigma \doteq S_1, S_2, \ldots, S_m$ that overlap with the unsafe set Ψ. We compute the set P_σ such that if P_σ is satisfiable, then there exists a trajectory that stays in the unsafe set for at least m duration. The satisfiable constraints set P_σ for the longest contiguous sequence of stars provides the desired counterexample trace. This procedure is formally defined in Algorithm 2.

The algorithm proceeds as follows: the main loop (lines 2-14) iterates over all stars having non-empty overlap with the unsafe set Ψ. The inner loop (lines 4-12) enumerates all the contiguous sequences σ starting with S_i and computes the set of constraints P_σ for the sequence. If the constraints are feasible, then the valuation of the basis variables that satisfies these constraints and the star S_i are stored. The length of the longest counterexample is also updated. In line 16, the execution corresponding to the longest contiguous counterexample is obtained using the valuation $\bar{\boldsymbol{\alpha}}_{len}$.

Theorem 1 *The execution returned by Algorithm 2 returns the longest counterexample.*

Proof 1 *We prove this by contradiction. Suppose that for the given initial set $\Theta \triangleq \langle c_0, V_0, P_0 \rangle$, the longest counterexample $\xi(x_0) \doteq x_0, x_1, \ldots, x_k$ spends duration m in the unsafe set Ψ. Consider*

input : Simulation equivalent reachable tree $ReachTree$ and unsafe set Ψ
output : Counterexample ce that spends longest contiguous time in Ψ

1 $length_{max} \leftarrow -\infty$; $lengthStar \leftarrow \perp$; $ce \leftarrow \perp$;
2 **for** *each star* S_i *in* $ReachTree$ **do**
3 **if** $S_i \cap \Psi \neq \emptyset$ **then**
4 **for** *each chain* σ *starting with* S_i **do**
5 Transform Ψ into $\langle c_i, V_i, Q_i \rangle$ where $\sigma[i] \triangleq \langle c_i, V_i, P_i \rangle$;
6 $P_\sigma \leftarrow \bigwedge_{i=1}^{|\sigma|} Q_i \wedge P_i$;
7 **if** $P_\sigma(\bar{\alpha}) = \top$ *and* $|\sigma| > length_{max}$ **then**
8 $length_{max} \leftarrow |\sigma|$;
9 $\bar{\alpha}_{max} \leftarrow \bar{\alpha}$;
10 $lengthStar \leftarrow S_i$;
11 **end if**
12 **end for**
13 **end if**
14 **end for**
15 **if** $length_{max} \neq -\infty$ **then**
16 $ce \leftarrow getExecution(\bar{\alpha}_{max}, ReachTree)$;
17 **end if**
18 **return** ce;

Algorithm 2: Algorithm which computes the longest contiguous counterexample.

that the states $x_j, x_{j+1}, \ldots, x_{j+m-1}$ in the execution ξ lie in the unsafe set. Additionally, suppose that the execution returned by Algorithm 2 returns a counterexample of length strictly less than m.

From the soundness and completeness result of simulation equivalent reachability (Bak & Duggirala 2017b), we have that $\exists$ stars $S_j, S_{j+1}, \ldots, S_{j+m-1}$ in $ReachTree$ such that $\forall j \leq k \leq j+m-1, x_k \in S_k$. Therefore, it should be the case that $\forall k, j \leq k \leq j+m-1$, $\Psi \cap S_k \neq \emptyset$. Additionally, since the trajectory ξ passes through $\Psi \cap S_k$, it should be the case that $\xi \in \langle c_0, V_0, P_k \wedge Q_k \rangle$ where $S_k \triangleq \langle c_k, V_k, P_k \rangle$ and $\Psi \triangleq \langle c_k, V_k, Q_k \rangle$. Therefore, the constraint P_σ that is computed for the sequence $\sigma \doteq S_j, S_{j+1}, \ldots, S_{j+m-1}$ should be feasible and would be updated as the longest counterexample in lines 7- 11. Which is a contradiction.

Analysis and Optimizations: In the $ReachTree$, a star can have at most one continuous successor and d discrete successors where d is the highest number of discrete transitions from any mode. If we consider the full tree with at least one step executed in each mode, the worst case possible number of sequences σ of length m would be $O((d+1)^m)$. Hence, the worst case time for computing the length would be to perform $O(m^2 \cdot (d+1)^m)$ linear program optimizations. However, in practice, such worst case bounds are not observed. In almost all of our experiments, the duration for overlap is not the order

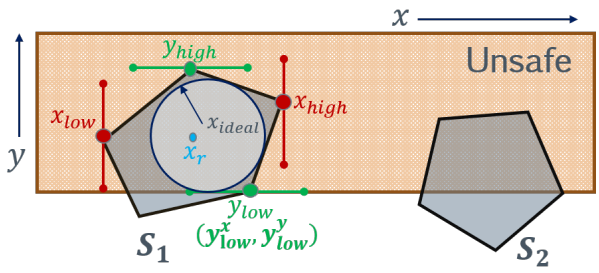


Figure 3.3: Illustration of the robust counterexample.

of m, each star has at most one active transition, and the number of sequences to be handled is at most one or two sequences of the maximum length.

One of the optimizations that can be performed for eliminating certain counterexamples is to conduct something similar to a binary search. That is, given a sequence $S_i, S_{i+1}, \ldots, S_{i+m-1}$ starting from star S_i that overlaps with Ψ, we can check if $S_{i+\lfloor \frac{m}{2} \rfloor}$ overlaps with Ψ. If there is no overlap, we can assert that the length of the longest unsafe sequence is less than $m/2$. However, this is a heuristic which may help in saving run time in some cases but not all.

3.3 Robust Counterexample

We now present the algorithm to obtain the robust counterexample. Recall that a counterexample starting from an initial state x_r is said to be δ-robust if and only if for all states $x \in B_\delta(x_r)$, there exists an unsafe execution starting from x. Informally, if we perturb the execution starting from x_r by less than δ, it remains unsafe. For obtaining this counterexample, we leverage the convexity property of reachable set.

For an unsafe star, the ideal robust counterexample is the center of the maximum ball inscribed inside the intersection of the star with the unsafe set. Since computing the maximum ball inscribed in a convex polytope is computationally hard (Xie, Snoeyink & Xu 2006, Allen Zhu, Liao & Orecchia 2014), we, therefore, compute a proxy as some internal state of the polytope. In our case, this is the centroid of extreme points in each orthonormal direction. We illustrate the approach using Figure 3.3 where $\bar{x}$ is the center of the maximum ball inscribed and x_r is its proxy. The generalization to the case of multiple stars intersecting with the unsafe set for the given sequence is trivial.

Consider a star S_1 having non-empty overlap with the unsafe set. After obtaining the set $S_1 \cap \Psi$, we compute states on the extreme ends in each direction by optimizing (maximizing and minimizing) the

cost function. Suppose these states are $x_{low}, x_{high}, y_{low}$ and y_{high}, respectively. Then the robust unsafe state is the centroid of these states.

$$x_r = (x_{low} + x_{high} + y_{low} + y_{high})/4$$

Remark 4 *For each state x in a convex set X, there exists $m \geq n+1$ states $x_1, \ldots x_m \in X$ such that the state $x \in X$ is represented as their convex combination. That is, $\exists$ scalars $\boldsymbol{\beta_1}, \ldots, \boldsymbol{\beta_m} \geq 0$ with $\sum_{i=1}^{m} \beta_i = 1$ such that*

$$x = \boldsymbol{\beta_1} x_1 + \boldsymbol{\beta_2} x_2 + \ldots + \boldsymbol{\beta_m} x_m.$$

Theorem 2 *If the intersection of the star S with the unsafe set Ψ is a non-empty convex set $S_\Psi \doteq (S \cap \Psi) \neq \emptyset$, then the robust unsafe state $x_r \in S_\Psi$.*

Proof 2 *For n orthonormal directions, we obtain $2n$ vertices of the convex set by maximizing and minimizing the cost function in each direction. The centroid, x_r, of these vertices can be represented as their convex combination with scalars $bm\beta_i = \frac{1}{2n} \geq 0$ such that $\sum_{i=1}^{2n} \boldsymbol{\beta_i} = 1$. This entails $x_r \in S_U$ as a consequence of Remark 4.*

The user can pick non-orthonormal directions as well to define cost function.

Remark 5 *There exists a set $B_\delta(x_r) \subseteq S_\Psi$, $\delta_r \geq 0$ where*

$$\delta_r = \arg\max_{\delta} B_\delta(x_r).$$

This follows from Theorem 2. The robust unsafe state $x_r \in S_\Psi$ is either on one of the hyper-planes defining S_Ψ or a state not on the edge. In the first case, $\delta = 0$, otherwise δ is the euclidean distance from x_r to its nearest vertex, which is positive.

We use the longest contiguous sequence of unsafe stars from Section 3.2 to find the robust counterexample.

In Algorithm 3, σ is the chain starting from $lengthStar$ and has the length of the longest counterexample. P_σ represents the intersection of unsafe set Ψ with stars in σ. In main loop (lines 7- 12), we formulate optimization problems to find the centroid $\bar{\alpha}^d$ in each orthonormal direction d. In line 13, the robust counterexample ce is obtained by using the centroid of all $\bar{\alpha}^d$ computed in the main loop. The user

input : Simulation equivalent reachable tree $ReachTree$, unsafe set Ψ, $lengthStar$ and $length_{max}$ as computed in Algorithm 2

output : Robust counterexample ce

$ce \leftarrow \perp$;
if $lengthStar \neq \perp$ **then**
  $S_1 \leftarrow lengthStar$;
  $\sigma \leftarrow S_1, S_2, \ldots, S_m$ where $m = length_{max}$;
  Transform Ψ into $\langle c_i, V_i, Q_i \rangle$ where $\sigma[i] \triangleq \langle c_i, V_i, P_i \rangle$;
  $S_\sigma \leftarrow \langle c_0, V_0, P_\sigma \rangle$ where $P_\sigma \doteq \bigwedge_{i=1}^{m} Q_i \wedge P_i$;
  for *each orthonormal direction* $d \in \mathbb{R}^n$ **do**
    $\bar{x} \leftarrow \arg\max_x d^T x$ given $x \in S_\sigma$;
    $\bar{\alpha}^d_{max} \leftarrow getBasisVariables(\bar{x})$;
    Similarly, $\bar{\alpha}^d_{min}$ is obtained by minimizing $d^T x$;
    $\bar{\alpha}^d \leftarrow (\bar{\alpha}^d_{max} + \bar{\alpha}^d_{min})/2$;
  end for
  $\bar{\alpha} \leftarrow (\sum_d \bar{\alpha}^d)/n$;
  $ce \leftarrow getExecution(\bar{\alpha}, ReachTree)$;
end if
return ce;

Algorithm 3: Algorithm which computes the robust counterexample such that a small perturbation yields a new counterexample.

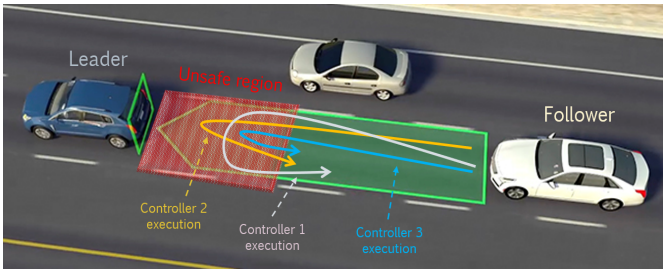


Figure 3.4: Unsafe execution profiles from 3 different controllers in Adaptive Cruise Control.

can provide additional directions for finding extreme points, which, in turn, may result into a different robust counterexample.

Runtime Analysis: Since the robust counterexample is obtained with respect to the longest unsafe sequence, the worst case complexity is proportional to computing the longest counterexample, that is $O(m^2 \cdot 2^m)$ as explained in Section 3.2. The heuristic approach based on conducting binary search applies here as well.

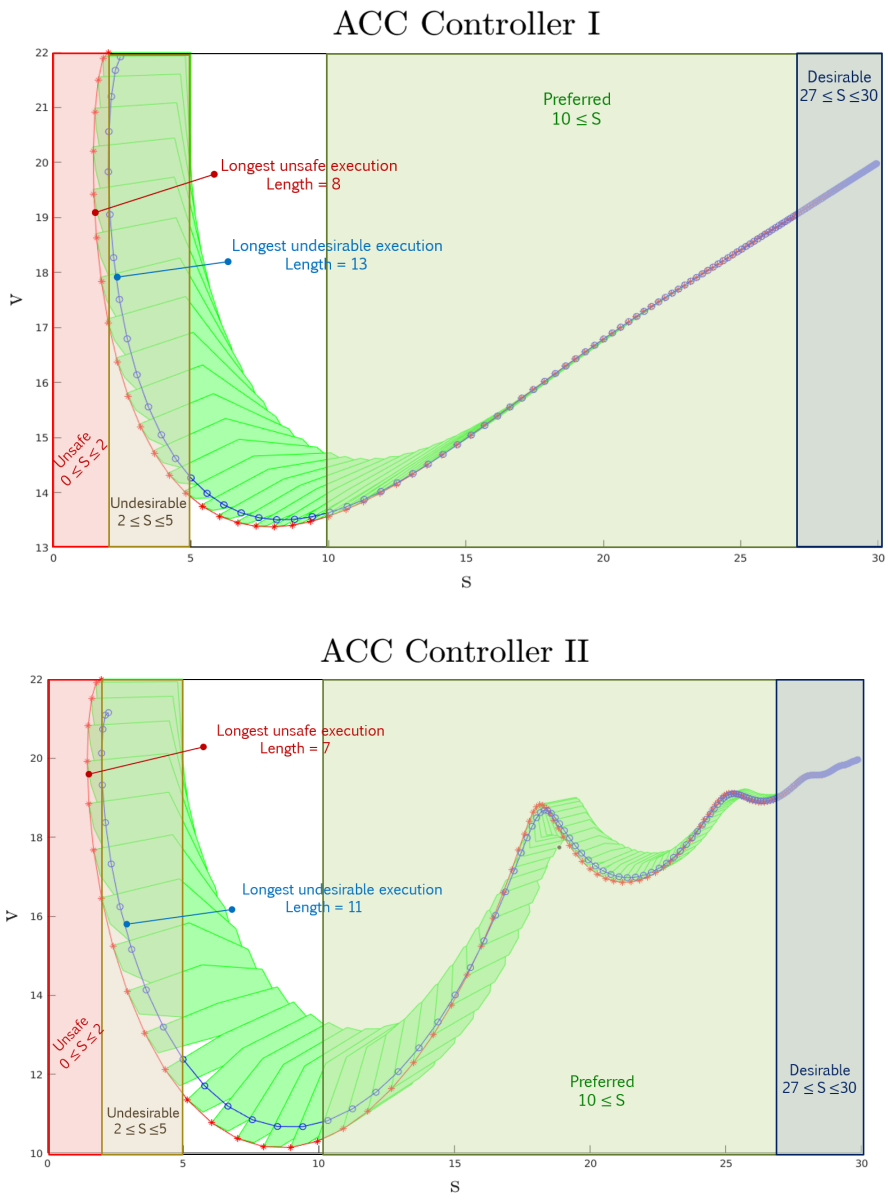


Figure 3.5: Illustration-I of controllers' performance in adaptive cruise control. Controller I gives longer unsafe and undesirable executions in comparison to controller II.

3.4 Analysis of Adaptive Cruise Controllers Using Counterexamples

In an adaptive cruise control system, the cars operate in autonomous manner. The leading car moves at a constant velocity; the following car slows down or speeds up automatically by sensing its velocity and the distance from the leading car. A control designer focuses on developing feedback controller for stabilizing this system. But a stable controller may not be safe for all initial states, where safety is defined as some minimum distance between these two vehicles or reasonable speed of the follower. As stated earlier, the objective is to evaluate the performance of controllers which violate the safety specification.

We provide an illustration of multiple adaptive cruise control algorithms in Figure 3.4 using their execution profiles[2]. The distance between the follower and the leader is shown in green, and the unsafe region is highlighted in red. Consider the execution profiles after applying 3 different stable controllers are given. Since all 3 controllers are unsafe as shown, these executions can be used in evaluating their performance. For instance, controller 2 execution ventures the most in the unsafe region in the direction of vehicles' movement. Although controller 1 execution is not the farthest in the unsafe region but it stays there for the longest time interval. Similarly, controller 3 execution is the most robust among all.

Consider the adaptive cruise control (ACC) system which was introduced in Section 1. This system is adopted from (Tiwari 2003).[3] However, given such a black-box scenario, our approach can be used to compare two controllers based on the safety specification. For convenience, we describe the system once again. It is 3 dimensional continuous time linear system with two cars where the distance between vehicles is s, the leading car's speed is v_f, the follower's velocity is v, and its acceleration is a. The differential equations deployed at the follower ACC system are as follows:

$$\dot{s} = (v_f - v)$$

$$\dot{v} = a$$

$$\dot{a} = g_1 * a + g_2(v - v_f) + g_3(s - (v + 10))$$

Here, g_1, g_2 and g_3 are *gain* variables. The values of gain variables for Controller-I are $g_1 = -3$, $g_2 = -3$, $g_3 = 1$ whereas, the gain variables for Controller-II are $g_1 = -1$, $g_2 = -3$, and $g_3 = 1$. The

[2]The image source is https://my.cadillac.com/learnAbout/adaptive-cruise-control

[3]This controller is not related to the execution profiles depicted in Figure 3.4 which is presented for only illustration purpose.

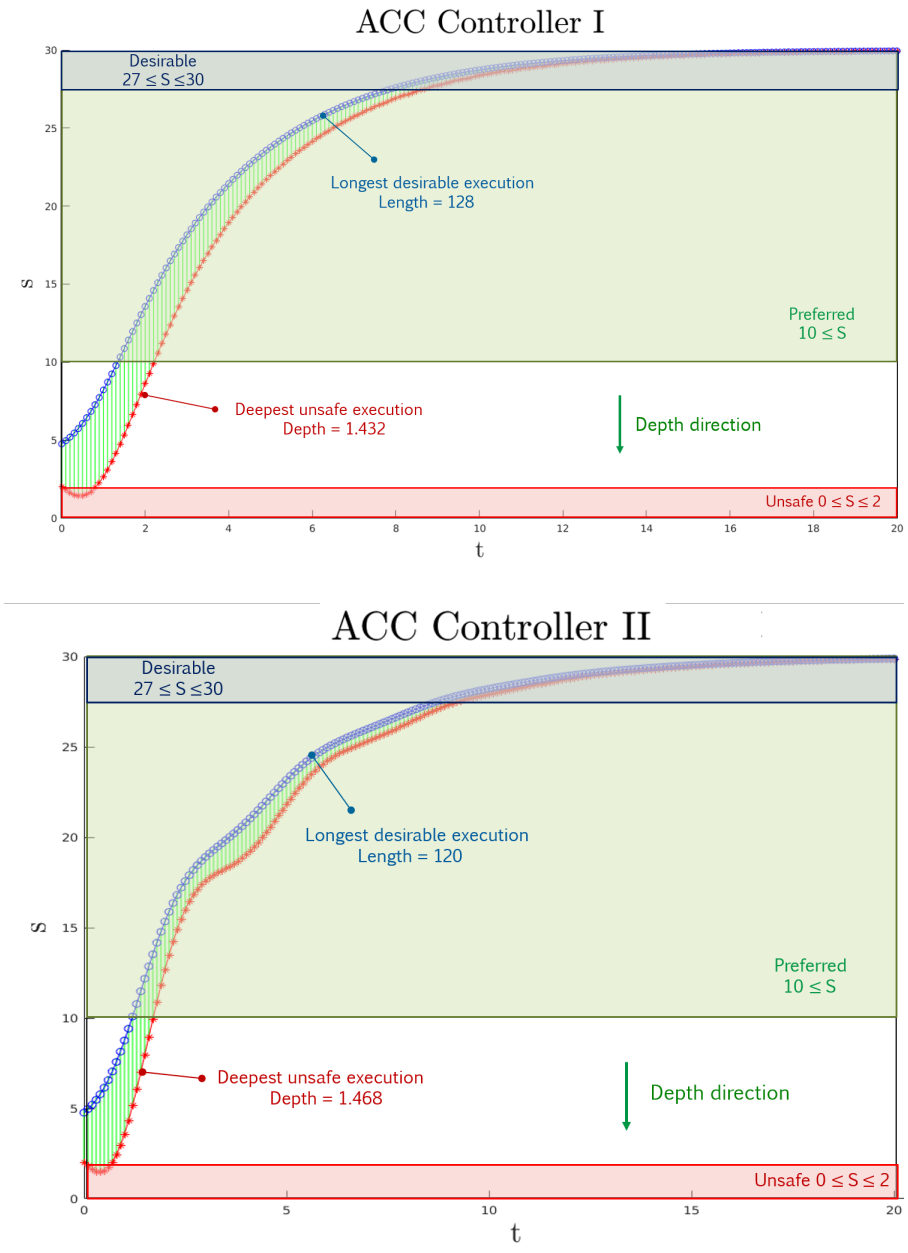


Figure 3.6: Illustration-II of controllers' performance in adaptive cruise control. Although the system with controller II gets more close to the leading car, it tries to stabilize faster once it is at the desirable distance.

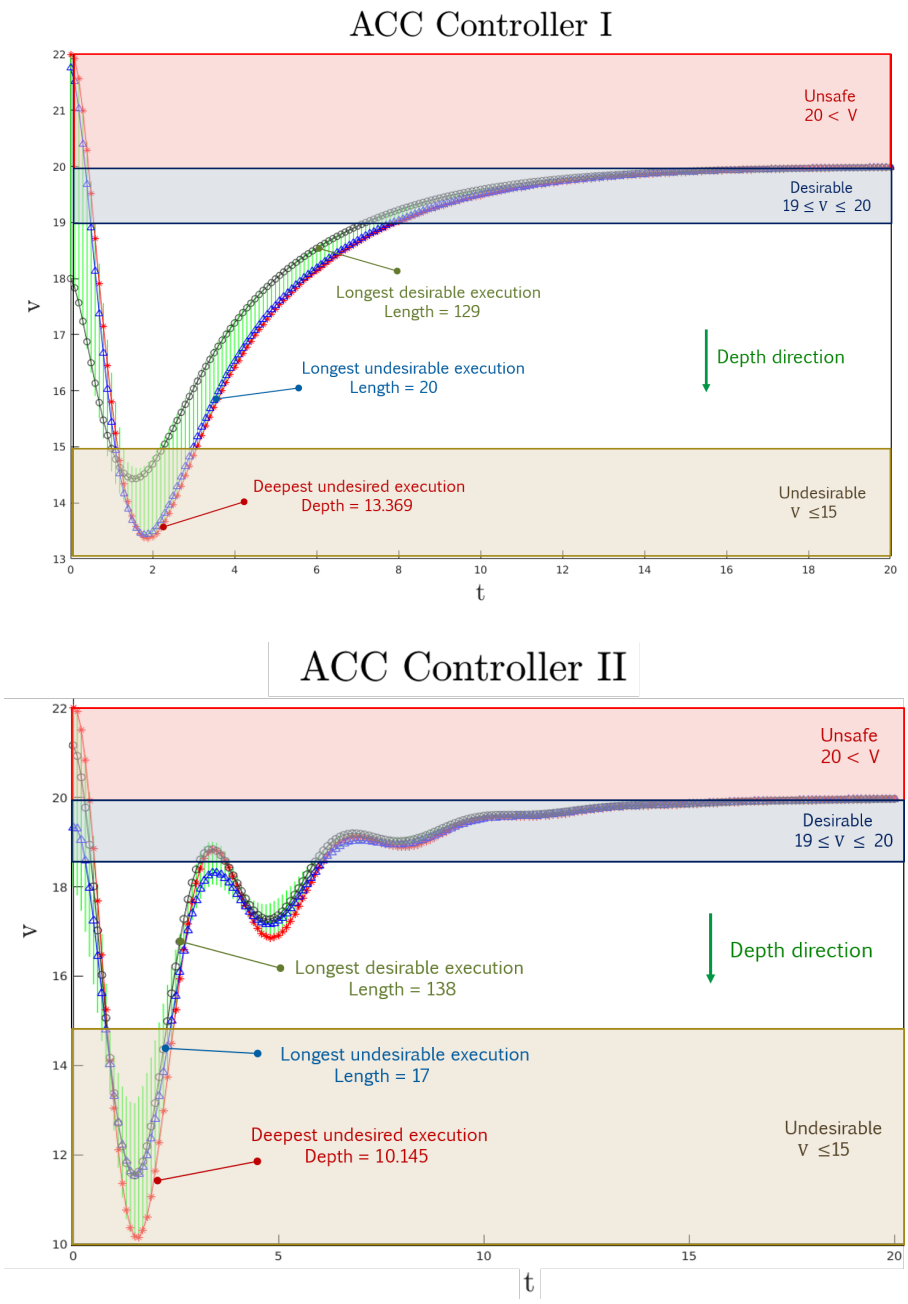


Figure 3.7: Illustration-III of controllers' performance in adaptive cruise control. Although the system with controller II slows down to an undesirable speed 10.145, it eventually achieves the desirable speed faster.

stable equilibrium of the system is at $a = 0$, $v = v_f$, and $s = v_f + 10$. The designer can use standard tools like SOSTOOLS (Papachristodoulou, Anderson, Valmorbida, Prajna, Seiler & Parrilo 2013) to find Lyapunov functions for proving controller stability. The original goal of adaptive cruise control is to keep the follower at a safe distance from the leader. Because not every stable controller is essentially safe, conducting a quantitative analysis of controllers would be of interest to the designer.

Given the initial set as $s \in [2, 5]$, $v \in [18, 22]$, $v_f = 20$, and $a \in [-1, 1]$, the reachable sets computed by HyLAA for above mentioned two adaptive cruise controllers (ACC) are shown in Figure 3.5. Although both systems eventually stabilize to $v = v_f = 20$ or $s = v_f + 10 = 30$, they are unsafe with respect to the safety specification $\square(s \geq 2)$. Notice that a more strict safety specification can be $\square(s \geq 0)$, but, during the design phase, one would want to work with specification that is conservative. As shown in Figure 3.5, the longest counterexample after applying controller I is of length 8 whereas its counterpart obtained from controller II has length 7. This means that controller II helps the system to recover faster from the unsafe region.

As an important side effect, our approach can also measure the extent to which a specification is satisfied. For instance, although $s \leq 2$ is certainly *unsafe*, the specification $2 \leq s \leq 5$ is *undesirable* as it can possibly render the system unsafe if the follower speeds up or the leader slows down. The longest undesirable execution obtained from controller I is of length 13 while controller 2 gives the longest undesirable execution to be of length 11. This re-emphasize that controller II makes the follower to get to the safe distance quicker as compared to controller I (Ref. Figure 3.5).

Building on above discussion, one might change the specification level to be *desirable* ($27 \leq s \leq 30$) because the system is required to be eventually stable i.e., $s = 30$. We plot distance s against time t in Figure 3.6. The longest desirable execution obtained from controller I is longer than the longest desirable execution generated from controller II. This would mean that the system with controller II tries to stabilize faster once it is at a desirable distance. Similarly, if we look at the maximum depth in the unsafe region, controller II is better.

To highlight that specifications over two different system variables may semantically differ, Figure 3.7 shows multiple specifications defined over v. As the given system stabilizes when $v = v_f = 20$, the specification $19 \leq v \leq 20$ is regarded as *desirable* and $v > 20$ as *unsafe*. Having the follower slowed down beyond a reasonable speed is also bad, therefore, the condition $v \leq 15$ is considered *undesirable*. The lengths of longest desirable executions indicate that the system with controller II obtains the desirable

speed faster than that with controller I. However, looking at the deepest undesirable executions reveals that controller II slows down the system to a speed 10.145 while controller I helps maintaining it above 13.

This exercise underlines the need for a software tool that can assist the designer in not only evaluating different controllers but also understanding their merits when the specification changes. The analysis will enable them to take action(s) to improve respective controllers.

3.5 Experimentation

The proposed algorithms have been implemented in a Python based verification tool, HyLAA; although, some of the computational libraries used may be written in other languages. Simulations for reachable sets are performed using `scipy's odeint` function, which can handle stiff and non-stiff differential equations using the FORTRAN library `odepack's lsoda` solver. Linear programming is performed using the `GLPK` library, and matrix operations are performed using `numpy`. The measurements were performed on a system running Ubuntu 18.04 with a 3.00GHz Intel Xeon E3-1505M CPU with 8 cores and 32 GB RAM.

HyLAA has a provision to perform verification in *aggregation* mode for better performance. We run HyLAA in *de-aggregation* mode for our experiments. HyLAA, by default, concludes its run as soon as it finds a counterexample. But, we let the tool run for the entire duration because we require to perform our analysis on all reachable stars intersecting with the unsafe set.

3.5.1 Benchmarks

The benchmarks for our study are taken from a standard benchmarks suite for continuous and hybrid systems [4] and (Beg, Davoudi & Johnson 2017). Most of these benchmarks are originally safe, however, in order to highlight counterexamples, we choose unsafe set such that the reachable set intersects with the unsafe set at multiple time instances. We further adjust the size of the unsafe set and observe that the intersection window of reachable set with the unsafe set differs proportionally. The variations in the unsafe set size are denoted as SU, MU, LU for **S**mall, **M**edium and **L**arge unsafe set respectively.

[4] https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

Model (Dims)	Longest Counter-example(LCE)	Actual Intersection Duration	LCE Duration	Deepest Counter-example(DCE)	Direction, Depth	Verification Time(sec)	LCE, DCE Gen. Times(sec)
Harmonic Oscillator (2)							
SU	[-5.373 0.0]	[5 10]	[6 10]	[-5.459 0.188]	$x_1 = 1, 2.0$	0.17	0.01, 0.00
MU	[-5.0 0.3968]	[4 10][33 44][66,74]	[33 44]	[-6 0.8829]	$x_2 = 1, 5.0$	0.22	0.03, 0.00
LU	[-5 0.296]	[3 10][29 49][59,100]	[59 100]	[-6 1]	$x_2 = 1, 5.288$	0.28	0.17, 0.01
Vehicle Platoon 1 (15)							
SU	$x_8 = 1.0475$ $x_{2,5} = 1.1$ $x_i = 0.9$	[27 41]	[29 41]	$x_1 = 1.071$ $x_2 = 0.993$ $x_i \in \{0.9, 1.1\}$	$x_2 = 1$, -0.1825	1.82	0.18, 0.11
MU	$x_{6,9} = 1.1, x_i = 0.9$ $x_{12} = 1.0761$	[27 73]	[27 73]	$x_i \in \{0.9, 1.1\}$	$x_2 = 1, 0.0170$	2.90	1.40, 0.39
LU	Same as above	[27 100]	[27 100]	$x_i \in \{0.9, 1.1\}$	$x_2 = 1, 0.0170$	3.51	3.78, 0.40
Vehicle Platoon 2 (30)							
SU	$x_9 = 0.9223$ $x_5 = 1.0204$ $x_i \in \{0.9, 1.1\}$	[42 48]	[44 48]	$x_5 = 0.9005$ $x_{23} = 1.0473$ $x_i \in \{0.9, 1.1\}$	$x_5 = 1$, -0.26347	4.86	0.23, 0.12
MU	$x_{19} = 1.0501$ $x_i \in \{0.9, 1.1\}$	[42, 53]	[45 53]	$x_2 = 0.91327$ $x_4 = 0.9389$ $x_5 = 1.1, x_i = 0.9$	$x_5 = 1$, -0.2217	5.20	0.43, 0.27
LU	$x_i = 0.9$	[36 100]	[36 100]	$x_i \in \{0.9, 1.1\}$	$x_5 = 1, 0.01745$	10.73	9.81, 1.87

Table 3.1: Longest contiguous and Deepest counterexamples in Linear Dynamical Systems for different sizes of the unsafe set

The linear continuous systems benchmarks - *Harmonic Oscillator*, *Vehicle Platoon 1* and *Vehicle Platoon 2* are simulated for maximum 100 time steps with step size 0.2 sec. The simulations for *Ball string* and *Two tanks* benchmarks are performed for maximum 200 time steps with step size 0.01 sec. The simulation for *Filtered oscillator* is carried out for maximum 100 time steps with step size 0.02 sec, and for *Forward converter* with step size 1×10^{-6}. The values of input variables in *Two tanks* benchmark are fixed to 0 which belongs to the actual interval [-0.1, 0.1]; whereas in *Forward converter*, the input (V_{in}) is fixed to 100 from the interval [98 102].

3.5.2 Evaluation Results and Analysis

The evaluation results for linear dynamical systems are provided in Table 3.1 and the results for linear hybrid systems are given in Tables 3.2 and 3.3.

Dims is the no. of dimensions, SU, MU, LU are variations of the unsafe set - **S**mall, **M**edium and **L**arge. Modes, in Table 3.2 is the number of system locations. *Longest Counterexample* (LCE) is a state in the initial set, simulation from which stays for the longest contiguous time in the unsafe set. In Table 3.1, x_i represents all the variables whose values are not explicitly given and $x_i \in \{0.9, 1.1\}$ denotes that the value is either 0.9 or 1.1. *Actual Intersection Duration* is the mode-wise ordered sequence of discrete time step intervals when reachable set intersects with the unsafe set. *LCE Duration* is the interval for the longest counterexample. *Verification Time* is the time HyLAA takes for verification, *LCE Gen. Time* is the time it takes to generate the longest counterexample and

Model	Dims, Modes	Unsafe Set Size	Longest Counterexample	Actual Inter. Duration	LCE Duration	Verification Time (sec)	LCE Gen Time (sec)
Ball String	2, 2			(ext, freefall)	(ext, freefall)		
		SU	[-0.9507 -0.15]	[18 20][21 23]	[18 20][21 23]	0.25	0.01
		MU	[-1.0191 -0.15]	[12 20][21 29]	[13 20][21 29]	0.33	0.07
		LU	[-0.9618 -0.15]	[7 20] [21 37]	[7 20][21 37]	0.38	0.22
Two Tanks	2, 4			(loc3, loc1)	(loc3, loc1)		
		SU	[1.763 1.1]	[21 26][27 40]	[24 26][27 40]	15.24	0.40
		MU	[2.407 1.077]	[16 28][33 78]	[-][34 77]	17.78	5.25
		LU	[2.497 1.1]	[7 30][31 81]	[15 30][31 81]	20.55	11.46
Filtered Oscillator	6, 4			(loc3, loc4)	(loc3, loc4)		
		SU	[0.294 0.0998 0...]	[39 64][65]	[39 64][65]	6.37	1.76
		MU	0.2938 0.1 0...]	[18 64][65 67]	[18 64][65 67]	6.96	7.14
		LU	[0.2938 0.1 0...]	[8 64][65 69]	[8 64][65 69]	7.10	12.70
Forward Converter	5, 5			(loc1, loc2, loc5)	(loc1, loc2, loc5)		
		SU	[0 0.399 0.223 0 0]	[8 11][12 16][17 18]	[8 11][12 16][17 18]	7.40	0.39
		MU	[0 0.4 0.2928 0 0]	[6 11][12 16][17 22]	[7 11][12 16][17 22]	7.79	0.83
		LU	[0 0.4 0.355 0 0]	[5 11][12 16][17 25]	[6 11][12 16][17 25]	8.84	1.32

Table 3.2: Longest contiguous counterexample in Linear Hybrid Systems.

Model	Deepest Counterexample	Direction	Depth	Verification Time (sec)	DCE Gen Time (sec)	Robust Counterexample	RCE Gen Time (sec)
Ball String	[-1.05 0.0691]	$x_2 = 1$	6.0	0.25	0.00	[-0.956, 0.0]	0.01
	[-1.045 -0.15]	$x_2 = 1$	7.0	0.33	0.00	[-1.019, -0.146]	0.08
	[-1.035 -0.15]	$x_1 = 1$	0.8	0.38	0.01	[-0.956, 0.0]	0.24
Two Tanks	[1.8995 1.0646]	$x_2 = 1$	0.1	15.24	0.02	[1.677, 1.016]	0.40
	[2.406 1.0282]	$x_2 = 1$	0.3	17.78	0.10	[1.731, 1.003]	5.27
	[2.225 1]	$x_1 = 1$	1.9	20.55	0.12	[2.326, 1.002]	11.50
Filtered Oscillator	[0.3 0.1 0...]	$x_6 = 1$	0.496	6.37	0.03	[0.2972 0.0993 0...]	1.77
	[0.3 0.0988 0...]	$x_6 = 1$	0.51	6.96	0.10	[0.2969 0.0994 0...]	7.16
	[0.3 0.1 0...]	$x_1 = 1$	0.67	7.10	0.10	0.2969 0.0994 0...]	12.73
Forward Converter	[0 0.4 0.4 0 0]	$x_3 = 1$	2.9056	7.40	0.01	[0.2 0.399 0.231 0 0]	0.40
	[0 0.4 0.2928 0 0]	$x_2 = 1$	0.3003	7.79	0.02	[0.2 0.396 0.346 0 0]	0.85
	[0 0.4 0.4 0 0]	$x_3 = 1$	2.9056	8.84	0.02	[0.2 0.397 0.378 0 0]	1.36

Table 3.3: Deepest and Robust counterexamples in Linear Hybrid Systems.

Direction is the direction in which the depth of the counterexample is obtained. For instance, in a 2-dimensional system (x, v), the direction $x_2 = 1$ represents a vector $[0, 1] \in \mathbb{R}^2$. *Deepest Counterexample* (DCE) is an initial state from which the simulation goes the deepest in given direction in the unsafe set. *DCE Gen. Time* is the time it takes to generate the deepest counterexample. *RCE Gen. Time* is the time taken for generating the robust counterpart. As we first obtain the predicates for LCE to compute the robust counterexample, it's generation time is inclusive of the longest counterexample generation time from Table 3.2.

The longest counterexample generation can be slower than the overall verification. As explained in Section 3.2, the combined number of constraints to be solved can become fairly large, which increases the counterexample generation time. Observe that the longest counterexample length is not necessarily

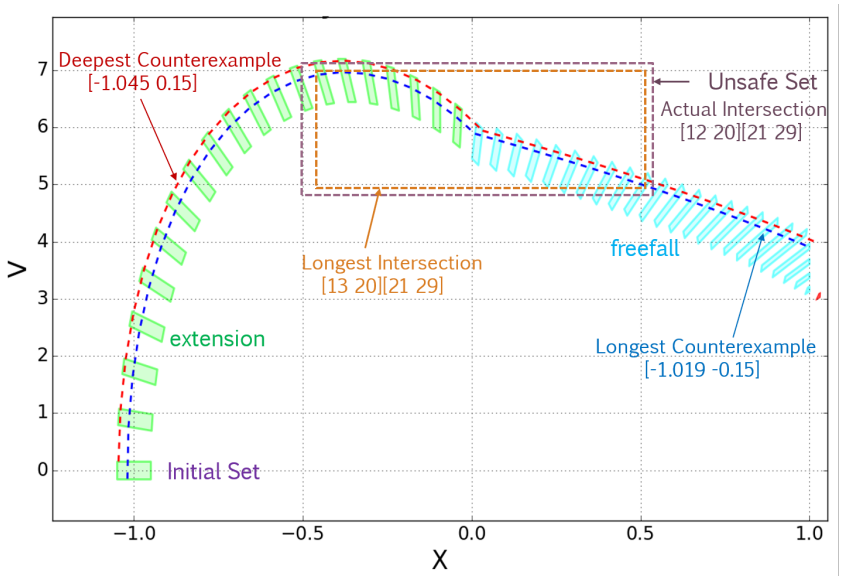


Figure 3.8: The longest contiguous and deepest counterexamples in *Ball string* benchmark. The actual intersection duration is [12 20][21 29] whereas that of the longest counterexample is [13 20][21 29].

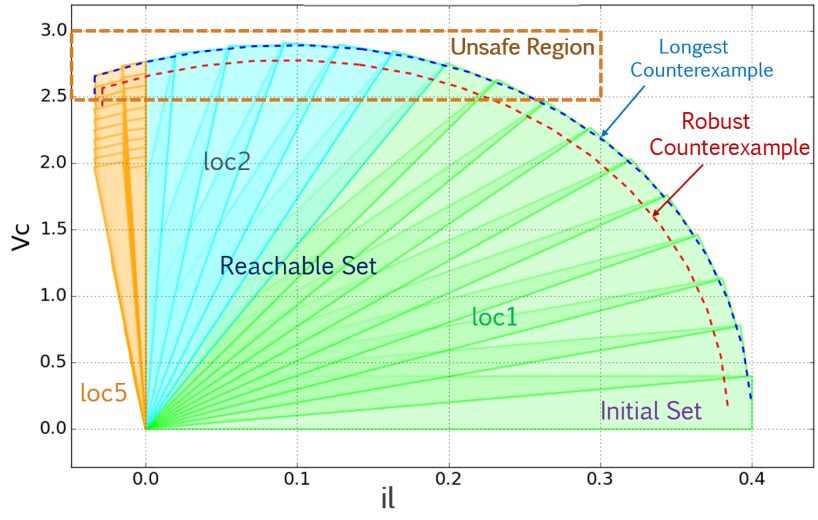


Figure 3.9: The longest contiguous and robust counterexamples in *Forward converter* benchmark. The longest counterexample duration is [8 11][12 16][17 18] which, in this case, is the actual intersection duration.

same as the actual duration of the overlap between the reachable and the unsafe set. This is a direct consequence of our approach: if a system of constraints during certain time interval is not feasible, we prune the list and again check for its feasibility until we find a solution.

We illustrate the longest contiguous and deepest counterexamples for MU configuration of the unsafe set in *Ball string* benchmark. This is a 2-dimensional system (x, v) having two modes - *extension* and *freefall*. The transition from extension to freefall occurs when $x = 0$. The unsafe set is [-0.5 0.5][5 7]. As shown, the actual intersection duration (in discrete time steps) is [12 20][21 29] whereas that of the longest counterexample is [13 20][21 29]. The deepest counterexample has depth 7.0 in V direction ($x_2 = 1$).

Another illustration is provided for *Forward converter* benchmark. This is a 5-dimensional system (il_m, il, v_c, u, t) with 5 modes. Each color in the reachable set corresponds to a different mode. The longest counterexample duration is [8 11][12 16][17 18] which, in this case, is the actual intersection duration.

Our evaluations exhibit that varying the unsafe set size not only add to the counterexample generation time but also may yield different counterexamples. The increase in the unsafe set size results in the increase in the number of stars overlapping with it. This, in turn, may lead to longer counterexamples, and higher counterexample generation time because every new star adds to the analysis time. We also notice that the variations in the depth direction can provide different deepest counterexamples. Finally, the time taken for generating deepest counterexample is relatively much less as compared to the longest and robust counterexamples. The reason being we need to scan through the list of unsafe stars only once to obtain the star with maximum depth.

3.6 Absolute Longest Counterexample

The reader is referred to (Goyal, Bergman & Duggirala 2020) for more details about these evaluations.

We illustrate the problem of finding the longest counterexample through an illustration in Fig. 3.10. Consider five consecutive stars, S_1, S_2, S_3, S_4 and S_5 in the reachable set have overlap with the unsafe set as shown. If one picks the state $e_1 \in S_1$, then some of the *post* states of e_1, denoted as e_2 and e_5 do not lie in the unsafe set, while e_3 and e_4 lie in the unsafe set. Similarly, if one picks the state $l_1 \in S_1$ as

shown, then the *post* states l_2, l_4 and l_5 lie in the unsafe set but l_3 does not. Thus, the execution starting from l_1 provides a longer counterexample than the execution starting from e_1.

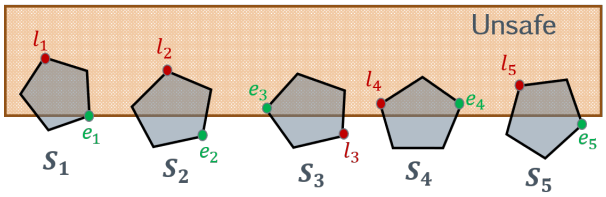


Figure 3.10: Illustration of the longest counterexample.

We formally state the absolute longest counterexample problem and describe the underlying technique used for obtaining these counterexamples.

3.6.1 Problem Statement

Definition 20 *For a set $\Psi \subseteq \mathbb{R}^n$, an indicator function*

$$\mathbb{1}_\Psi : Loc \times \mathbb{R}^n \to \{0, 1\}$$

is defined as

$$\mathbb{1}_\Psi(q) =\!\!=^{\Delta} \begin{cases} 1, & \text{if } x \in \Psi \\ 0, & \text{otherwise} \end{cases}$$

where $q \doteq (L_q, x)$ and $L_q \in L$.

Problem Definition Given the set of initial states Θ, the set of unsafe states Ψ and its indicator function $\mathbb{1}_\Psi$, compute

$$\arg\max_{q_0 \in \Theta} \sum_{t=0}^{\mathrm{T}-1} \mathbb{1}_\Psi(\xi(q_0)[t])$$

where ξ is the system simulation from Definition 4. For a simulation, the optimization problem aims to maximize the number of time steps where the simulation overlaps with the unsafe set.

As with the longest contiguous counterexample, the key insight behind the generation of absolute longest counterexample is that one has to select the *appropriate* state such that its corresponding execution has the maximum number of overlaps (not necessarily contiguous) with the unsafe set. In this instance,

any state $x_1 \in S_1 \cap \Psi$, with its successors x_2, x_4, x_5 such that $x_2 \in S_2 \cap \Psi$, $x_4 \in S_4 \cap \Psi$ and $x_5 \in S_5 \cap \Psi$ may be an appropriate choice. We, therefore, perform constraint propagation to identify a set of constraints, P, so that $\forall \bar{\alpha}$ such that $P(\bar{\alpha}) = \top$, we have, $x_1 = c_1 + V \times \bar{\alpha} \in (S_1 \cap \Psi)$, $x_2 = c_2 + V_2 \times \bar{\alpha} \in (S_2 \cap \Psi)$, $x_4 = c_4 + V_4 \times \bar{\alpha} \in (S_4 \cap \Psi)$, and $x_5 = c_5 + V_5 \times \bar{\alpha} \in (S_5 \cap \Psi)$.

We now present two computational frameworks to compute absolute longest counterexample.

3.6.2 MILP-based Framework

We formulate an MILP model for finding the longest counterexample as follows. For a given path Γ in the *ReachTree*, let Π be the set of stars S_i overlapping with the unsafe set Ψ, and $\boldsymbol{\alpha} \in \mathbb{R}^n$ be our basis variables such that $i = 1, \ldots, |\Pi|$ index the elements in Π. Additionally, consider $Q_i \wedge P_i$ be the set of linear constraints that need to be satisfied in order for S_i to be overlapping with the unsafe set. We can write each of these constraints as $\left(a^{i,k}\right)^T \bar{\alpha} \leq \mathbf{b}$, for $i = 1, \ldots, |\Pi|$, and $k = 1, \ldots, |Q_i \wedge P_i|$. We therefore have:

$$
\begin{aligned}
&\max \sum_{i=1}^{|\Pi|} z_i \\
&\text{s.t. } \left(a^{i,k}\right)^T \bar{\alpha} \leq \mathbf{b} + M\left(1 - z_i\right), \quad i = 1, \ldots, |\Pi|, \\
&\qquad\qquad\qquad\qquad\qquad\qquad\quad k = 1, \ldots, |Q_i \wedge P_i|, \\
&z_i \in \{0, 1\}, \qquad\qquad\qquad\qquad\quad i = 1, \ldots, |\Pi|.
\end{aligned}
$$

The z_i variable indicates whether all constraints in C_i are satisfied. Since this is a maximization problem on these decision variables, z_i will be 1 if it can, otherwise 0 if any one of constraints in C_i is not satisfied. Also note that this model requires the definition of an appropriate M, which in this instance can be set to

$$
\max_{i=1,\ldots,|\Pi|} \max_{k=1,\ldots,|Q_i \wedge P_i|} \left\{ \sum_{j=1}^{n} \left| a_j^{i,k} \right| \right\},
$$

noting that this can be refined if needed for each specific i and k. The formal procedure to find the longest counterexample using above MILP framework is provided in Algorithm 4.

3.6.3 SMT-based Framework

An SMT solver primarily answers the decision problem of whether a given logical formula is *satisfiable* i.e., if there exists some assignment to variables included in the formula such that this

input : Initial Set Θ, the simulation equivalent reachable tree $ReachTree$ and unsafe set Ψ
output : Trace ce that spends longest time in Ψ

$length_{max} \leftarrow -\infty$; $ce \leftarrow \perp$;
for *each path* Γ *in* $ReachTree$ **do**
  $\Pi \triangleq \{S_i | S_i \in \Gamma, S_i \cap \Psi \neq \emptyset\}$;
  Introduce $|\Pi|$ decision variables $z_1, z_2 \dots z_{|\Pi|}$;
  $\mathcal{C}_\Pi \leftarrow \emptyset$;
  Transform Ψ into $\langle c_i, V_i, Q_i \rangle$ where $\Pi[i] \triangleq \langle c_i, V_i, P_i \rangle$;
  $\mathcal{C}_\Pi \leftarrow \bigwedge_{i=1}^{|\Pi|} (\bigwedge_{p \in Q_i \wedge P_i} p + M(1 - z_i))$;
  $length_\Pi \leftarrow max \sum_i z_i$ while $C_\Pi(\bar{\alpha}) = \top$;
  if $length_\Pi > length_{max}$ **then**
    $length_{max} \leftarrow length_\Pi$;
    $\bar{\alpha}_{max} \leftarrow \bar{\alpha}$;
  end if
end for
if $length_{max} \neq -\infty$ **then**
  $ce \leftarrow getExecution(\bar{\alpha}_{max}, ReachTree)$;
end if
return ce;

Algorithm 4: MILP-based algorithm for the absolute longest counterexample.

assignment makes the logical formula evaluate to true. It either provides a satisfying assignment or declares that the formula is *unsatisfiable*. Certain SMT solvers also allow to encode linear optimization problems where the objective is to optimize a cost function while satisfying a given set of constraints. For this purpose, the solver provides the flexibility to specify constraints to be either soft or hard. A *hard* constraint is required to be asserted, whereas a *soft* constraint can be either satisfied or violated. Since a penalty is associated with the violation of soft constraint, the optimizer targets to minimize or maximize the overall penalty depending on the objective function. The approach to compute the longest counterexample using SMT is explained in Algorithm 5 where $\triangle$ designates soft constraints.

3.6.4 Evaluation and Discussion

For $ReachTree$ computation, we use a Python-based verification tool `HyLAA` (in *de-aggregation* mode) which uses `scipy's odeint` for simulating the differential equations, `GLPK` for linear programming, and `numpy` for matrix operations. The measurements were performed on a system running Ubuntu 18.04 with an 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM. We use

input : Initial Set Θ, the simulation equivalent reachable tree $ReachTree$ and unsafe set Ψ
output : Trace ce that spends longest time in Ψ

$length_{max} \leftarrow -\infty$; $ce \leftarrow \perp$;
for *each path* Γ *in* $ReachTree$ **do**
  $\Pi \triangleq \{S_i | S_i \in \Gamma, S_i \cap \Psi \neq \emptyset\}$;
  Introduce $|\Pi|$ binary variables $b_1, b_2 \dots b_{|\Pi|}$;
  $\mathcal{C}_\Pi \leftarrow \triangle_{i=1}^{|\Pi|} b_i$;
  Transform Ψ into $\langle c_i, V_i, Q_i \rangle$ where $\Pi[i] \triangleq \langle c_i, V_i, P_i \rangle$;
  $\mathcal{C}_\Pi \leftarrow C_\Pi \bigwedge_{i=1}^{|\Pi|} (b_i == (Q_i \wedge P_i))$;
  $length_\Pi \leftarrow Optimize_{SMT}(\mathcal{C}_\Pi)$ while $C_\Pi(\bar{\alpha}) = \top$;
  if $length_\Pi > length_{max}$ **then**
    $length_{max} \leftarrow length_\Pi$;
    $\bar{\alpha}_{max} \leftarrow \bar{\alpha}$;
  end if
end for
if $length_{max} \neq -\infty$ **then**
  $ce \leftarrow getExecution(\bar{\alpha}_{max}, ReachTree)$;
end if
return ce;

Algorithm 5: SMT-based algorithm for the absolute longest counterexample.

`Z3Py` (de Moura & Bjørner 2008) as an SMT solver, and Gurobi Optimizer (called from `C++`) for solving MILPs (Gurobi Optimization 2018).

The benchmarks for this study are taken from a standard benchmarks suite for continuous and hybrid systems [5], and (Nguyen & Johnson 2015) and (Sloth & Wisniewski 2011). As stated earlier, one can characterize various regions in the state space using specifications. We label that subspace as unsafe for our experiments. The designer can specify a region of interest in the same manner, and the longest execution obtained can be used to evaluate controllers with respect to that particular region/specification. Fig. 3.11 illustrates the experimental result for *Buck Converter*. The original benchmark has 2 locations but our model has 6 locations so as to incorporate transition resets. The figure shows the reachable set computed for locations - **closed1, open1,** and **closed2**.

The evaluations on various benchmarks are provided in Table 3.6.4. *Dims* is the number of system variables and *Modes* is the number of locations. The $ReachTree$ computed for a linear hybrid system can have multiple paths due to discrete transitions. Furthermore, each path may have multiple nodes overlapping with the unsafe set. *Longest Counterexample* is the valuation of basis variables such that

[5] https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

the corresponding execution has the maximum number of overlaps with the unsafe set. x_i represents all the variables whose values are not explicitly given. *Actual Intersection Duration* is the mode-wise ordered sequence of discrete time step intervals when the reachable set intersects with the unsafe set. *LCE Duration* is the interval for the absolute longest counterexample. *Verification Time* is the time `HyLAA` takes for verification which is exclusive of the counterexample generation time, *LCE Gen Time* is the time (in seconds) each framework takes to generate the longest counterexample.

As shown in the table, counterexample generation using SMT takes significantly more time than the time taken by MILP-based framework. Verification time of some benchmarks such as *Damped Oscillator* and *Ball String* is comparable to the SMT-based longest counterexample generation time.

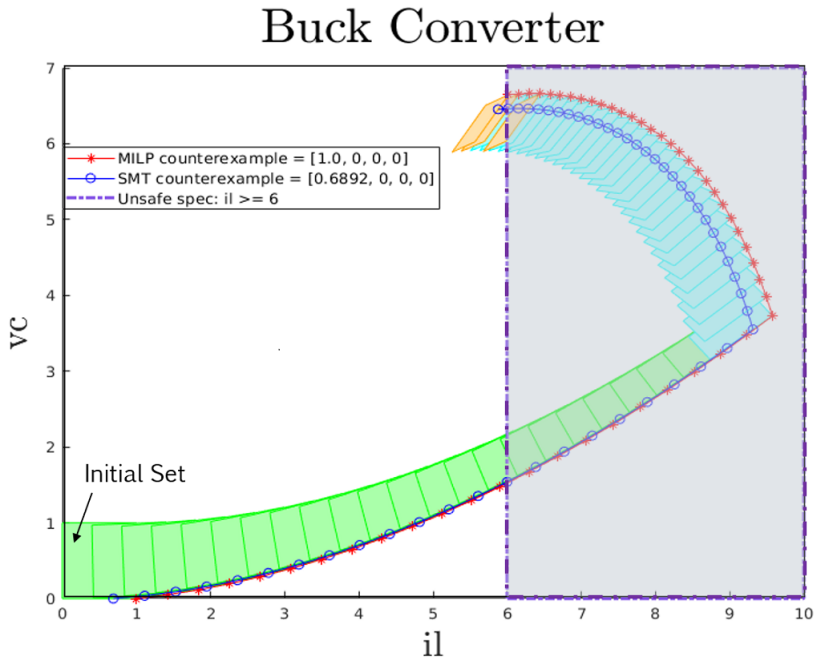


Figure 3.11: The absolute longest counterexample in Buck Converter.

Discussion: As MILP turns out to be the better alternative for generating the longest counterexample, an obvious question arises whether this framework is suitable in all cases. An issue with an MILP-based approach is numerical stability. The model requires the definition of M, which is larger than any value. Initial testing, even with a state-of-the-art commercial solver, led to a numerically unstable solution, returning a trajectory that isn't really a counterexample. Through iterating with `Z3` we were able to

Table 3.4: Evaluation results for absolute longest counterexample. *Longest Counterexample* (LCE) is the valuation of basis variables for an initial state from which the execution overlaps with the unsafe set at maximum time steps.

Model	Dims, Modes	Actual Inter. Duration	LCE Duration		Counterexample		Verification Time (sec)	LCE Gen. Time (sec)	
			MILP	SMT	MILP	SMT		MILP	SMT
Damped Oscillator 1	2, 1	[5 10] [34 44] [66 74]	[5 9] [35 44] [66 73]	[5 9] [35 44] [66 73]	[-5.28 0.764]	[-5.321 0.865]	0.44	0.04	0.51
Damped Oscillator 2	2, 1	[3 10] [29 49] [59 100]	[3 10] [30 49] [59 100]	[3 9] [29 49] [59 100]	[-5.0 0.398]	[-5.0 0.606]	0.59	0.04	0.55
Oscillating Particle	3, 1	[17 29] [51 58]	[18 21] [24 28] [52 56]	[18 21] [26 29] [52 57]	$x_1 = -0.0932$ $x_2 = 0.8193$ $x_3 = 0.9$	$x_1 = 0.0385$ $x_2 = 0.8877$ $x_3 = 0.9$	0.33	0.04	0.43
Vehicle Platoon 5	15, 1	[27 100]	[27 100]	[27 100]	$x_{11} = 0.96$ $x_i \in \{0.9, 1.1\}$	$x_i \in \{0.9, 1.1\}$	8.57	0.37	58
Vehicle Platoon 10	30, 1	[36 100]	[36 100]	[36 100]	$x_{23} = 1.029$ $x_i \in \{0.9, 1.1\}$	$x_{11} = 1.0$ $x_i \in \{0.9, 1.1\}$	31	0.91	360
Ball String	2, 2	**ext:** [12 13] **ext:** [15 20] **freefall:** [21 29]	[18 20]	[12 13] [16 20] [21 24]	[-1.008 -0.15]	[-0.95 -0.15]	0.31	0.03	0.3
Two Tanks	2, 4	**loc3:** [16 28] **loc1:** [33 78]	[34 77]	[35 78]	[2.399 1.079]	[2.407 1.077]	19.13	0.50	6.51
Buck Converter	4, 6	**cl1:** [13 21] **op1:** [22 50] **cl2:** [51]	[13 21] [22 50] [51]	[13 21] [22 50] [51]	il = 1.0 vc = 0 t = 0, gt = 0	il = 0.6892 vc = 0 t= 0, gt = 0	0.66	0.04	0.60
Filtered Oscillator	6, 4	**loc3:** [3 5] **loc3:** [7 21] **loc4:** [26]	[5] [7 21] [26]	[5] [7 21] [26]	[0.2069 0.07 0...]	[0.205 0.07 0...]	37	2.14	49

identify a suitable choice of M, but this can lead to an incorrect solution if sufficient care is not taken. On the other hand, `Z3` is slow but it doesn't suffer with the problem of numerical instability.

We also notice that the MILP-based approach and SMT-based approach return different counterexamples. This is compatible with our definition as longest counterexample need not be unique. In fact, the overlap with the unsafe set in the counterexamples returned by these two approaches can differ (as shown in the Table 3.6.4). Further, although the length of the longest counterexample is fixed, the counterexamples as well as their intersection intervals may differ across both frameworks. For instance, the longest counterexample length for *Ball String* is 11, and its respective duration generated with MILP and SMT is [18 20][21 28] and [12 13][16 20][21 24], respectively.

3.6.5 Counterexample for a Regular Expression

Consider a regular expression r over alphabet $\{0, 1\}$, which can be converted into a Deterministic Finite Automaton A_r whose language $L(A_r)$ is the set of all words expressed by r. For a given word $\omega \in L(A_r)$, one may be interested in finding an execution such that its overlapping pattern with the unsafe set conforms to ω where $\omega[i] \in \{0, 1\}$ is the character at index i. Here, $|\omega|$ is the number of times the reachable set overlaps with the unsafe set Ψ. For instance, given the overlap interval [6, 10] and a word "11010", the objective is to compute $q_0 \in \Theta$, if it exists, such that $q_6 \in \Psi \wedge q_7 \in \Psi \wedge q_8 \notin \Psi \wedge q_9 \in \Psi \wedge q_{10} \notin \Psi$, where $q_i \triangleq (x_i, L_i) = \xi(q_0)[i]$.

SMT Formulation The problem can be encoded in SMT by introducing $|\omega|$ decision variables b_i. If $\omega[i]$ is 1, assign b_i to be *true*; otherwise assign it to be *false*. Thus the problem is reduced to the satisfiability check of below constraints

$$C_\omega \triangleq \bigwedge_{j=1}^{l} \bigwedge_{k=1}^{m} b_j \wedge \bar{b}_k \text{ where } l + m = |\omega|.$$

Here, $b_i = Q_i \wedge P_i$ and $\bar{b}_i = \overline{Q_i \wedge P_i}$. If satisfiable, these constraints give a valid counterexample. Whereas unsatisfiability of C_ω implies that the given assignment to decision variables is not feasible and thus, there is no counterexample conforming to the pattern expressed by ω.

MILP Formulation We introduce $|\omega|$ binary variables z_i. Assign $z_i = 1$ if $\omega[i] = 1$, and $z_i = 0$ otherwise. Additional $|C_i|$ binary variables z_i^k are introduced for every set of constraints $C_i = Q_i \wedge P_i$. Each z_i^k value is tied to a constraint in C_i in such a way that any $z_i^k = 0$ makes the entire set C_i to be unsatisfiable. Now the model, for $i = 1, \ldots, |\omega|$ and $k = 1, \ldots, |C_i|$, becomes

$$
\begin{aligned}
&\max \sum_{i=1}^{|\omega|} z_i \\
&\text{s.t. } \left(a^{i,k}\right)^T \bar{\alpha} \leq \mathbf{b} + M\left(1 - z_i^k\right), \\
&\left(a^{i,k}\right)^T \bar{\alpha} \geq \mathbf{b} + \epsilon - M\left(z_i^k\right), \text{where } 0 < \epsilon \ll 1 \\
&\sum_k z_i^k \leq z_i + |C_i| - 1, \\
&z_i = \omega[i], \text{and } z_i \leq z_i^k, \\
&z_i \in \{0,1\}, z_i^k \in \{0,1\}.
\end{aligned}
$$

The problem is then reduced to an optimization problem over z_i's But this formulation does not guarantee a strict solution. That is, in a constraint $c^T x \leq \mathbf{b} + M(1 - z)$, by construction, assigning z to be 0 makes the original constraint ($c^T \leq \mathbf{b}$) relaxed and the solution may still satisfy it. Whereas, $0 \in \omega$ explicitly requires the corresponding constraint to be excluded[6].

3.7 Chapter Summary

This chapter has presented the notion of longest, deepest and robust counterexamples to a given safety specification in linear hybrid systems. The presented work builds on the prior work of computing a reachable set (Bak & Duggirala 2017*b*), which includes the set of states encountered by a simulation algorithm for systems with linear hybrid dynamics. The reachable set computation leverages the superposition principle and the generalized star representation. The counterexample generation mechanisms reuse the artifacts generated during the model checking process and employs constraint propagation. The chapter has also introduced two different frameworks for computing absolute longest counterexample and analyzes their performances. It has highlighted how these types counterexamples can provide some useful

[6]Excluding a constraint $c^T \leq \mathbf{b}$ is equivalent to satisfying $c^T \geq \mathbf{b}$

insights into the system behavior, thus assisting the control designer in comparing different controllers during control synthesis.

Acknowledgement. Muqsit Azeem (Technical University of Munich, Germany) and Aditya A. Shrotri (Rice University, Houston, TX) had provided valuable inputs regarding SMT formulation of the absolute longest counterexample.

CHAPTER 4: COMPLETE CHARACTERIZATION OF COUNTEREXAMPLES

Despite their utility in controllers' evaluation, the counterexamples introduced in the previous chapter cease to capture all modalities of a safety violation. Consider multiple executions of a system shown in Fig. 4.1. All executions cross the threshold where this overlap with the threshold could denote an overshoot in the regulation control, an undesirable maneuver in an autonomous car, a failure site in hardware design, or similar specification violation in a given system. Although it is important to observe that the system can go above the threshold, the control designer might be interested in executions crossing it multiple (possibly non-contiguous) time steps or more precisely how many times it was crossed or how many different ways (or characterizations) in which these executions can cross the threshold for a given system. The motivation to obtain these characterizations of a safety violation in control systems is inspired by failure identification in computer aided design, where extracting the essence of an error may still require a great deal of human effort. Yet debugging a design is shown to be greatly benefited from using more than one counterexample as well as by efficiently identifying crucial sites leading to the failure. This chapter attempts to compute all modalities of a safety violation in linear dynamical systems [1].

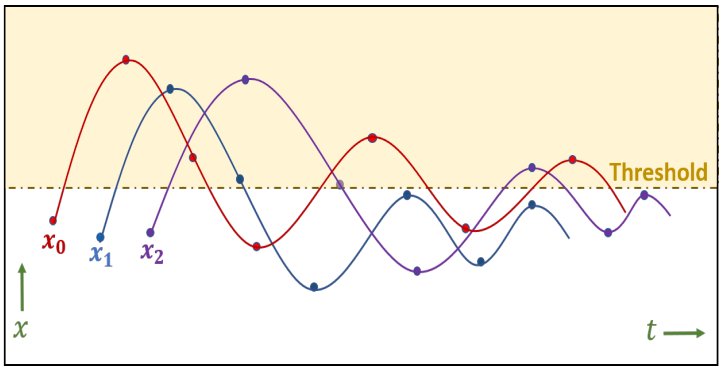


Figure 4.1: Different profiles of the system overshooting the threshold. The dots indicate executions states at certain discrete time steps.

[1] This work is currently under review.

We begin with a simple example that helps us in building the set up towards problem definition, proposed solution and related discussion.

Example 3 Oscillating Particle *(Sloth & Wisniewski 2011) is a 3-dimensional autonomous (without inputs) continuous time-invariant linear system, $\mathcal{F} \triangleq \langle \mathcal{A}, \mathcal{B} \rangle$, where:*

$$\mathcal{A} = \begin{pmatrix} -0.05 & -1.0 & 0 \\ 1.5 & -0.1 & 0 \\ 0 & 0 & -0.12 \end{pmatrix}, \mathcal{B} = \begin{pmatrix} 0 \\ 0 \\ 0 \end{pmatrix} \tag{4.1}$$

The initial set is $\Theta \doteq ([-0.1, 0.1], [-0.8, -0.4], [-1.07, -1])$.

Remark 6 *As per the definition of simulation-equivalent reachable set (Section 2.4), each generalized star in the reachable set for this example would have* 3 *basis vectors (or variables) and* 6 *constraints in its predicate such that, after each step, only the center and basis vectors change while the predicate remains the same. Initial star S_0 is denoted as $\langle c_0, V_0, P_0 \rangle$ where $c_0 = [0, 0, 0]^T$,*

$$V_0 = \begin{pmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{pmatrix}, P_0 \triangleq -0.1 \leq \boldsymbol{\alpha_1} \leq 0.1 \wedge -0.8 \leq \boldsymbol{\alpha_2} \leq 0.4 \wedge -1.07 \leq \boldsymbol{\alpha_3} \leq -1.0.$$

We consider $h = 0.6$ to be the unit time step. The set of states reachable exactly after one unit time step is denoted as a star $S_1 \triangleq \langle c_1, V_1, P_1 \rangle$ where $c_1 = [0, 0, 0]^T$,

$$V_1 = \begin{pmatrix} 0.72 & -0.52 & 0 \\ 0.78 & 0.69 & 0 \\ 0 & 0 & 0.91 \end{pmatrix}, P_1 \triangleq -0.1 \leq \boldsymbol{\alpha_1} \leq 0.1 \wedge -0.8 \leq \boldsymbol{\alpha_2} \leq 0.4 \wedge -1.07 \leq \boldsymbol{\alpha_3} \leq -1.0.$$

Similarly, the set of states reachable after exactly two unit time steps is a star $S_2 \triangleq \langle c_2, V_2, P_2 \rangle$ where $c_2 = [0, 0, 0]^T$,

$$V_2 = \begin{pmatrix} 0.11 & -0.74 & 0 \\ 1.11 & 0.07 & 0 \\ 0 & 0 & 0.86 \end{pmatrix}, P_2 \triangleq -0.1 \leq \boldsymbol{\alpha_1} \leq 0.1 \wedge -0.8 \leq \boldsymbol{\alpha_2} \leq 0.4 \wedge -1.07 \leq \boldsymbol{\alpha_3} \leq -1.0.$$

and so on.

The reachable set $Reach(\Theta)$ computed using generalized stars in this manner by the linear hybrid system reachability analysis tool, HyLAA (Duggirala & Viswanathan 2016), for $\mathrm{T} = 9.0$ is shown in Fig. 4.2. The safety specification is $\phi \triangleq \triangleq \square_{[0,9.0]} \neg p$, where $p \triangleq \triangleq y \geq 0.4$ is an atomic proposition in $\mathcal{P}$. The system violates ϕ because $Reach(\Theta)$ has non-empty intersection with the unsafe set $\Psi \doteq p$. Fig. 4.1 highlighted that different system executions can overshoot the threshold at multiple (possibly different) time instances. The reachable set illustration would give a much easier idea of the modalities of these executions instead of explicitly identifying individual overshoot profiles in a likely infinite state system.

Remark 7 *In this work, we focus on safety specification defined over one system variable. The extension to broader class of safety specifications is a part of future research.*

4.1 Introducing Characterization

For a given linear dynamical system and safety specification ϕ (a temporal logic formula over a set Ψ), consider that the reachable set violates the specification at k (not necessarily contiguous) time steps. The *characterization* of an execution $\xi(x_0)$ is defined as a function $\mathcal{C}_\Psi : (\mathbb{R}^n)^T \rightarrow \left(2^{\mathcal{P}}\right)^k$, which is a projection of $\xi(x_0)$ into the space of propositions defining the unsafe set Ψ. Thus the *complete characterization* is the set of all unique characterizations of a safety violation.

$$[\![\mathcal{C}_\Psi]\!] = \{\mathcal{C}_\Psi(\xi(x_0)) \mid \xi(x_0) \in Reach(\Theta)\}.$$

That is, $[\![\mathcal{C}_\Psi]\!]$ is set of all system executions mapped into the space $\left(2^{\mathcal{P}}\right)^k$ for the unsafe set. Sometimes we refer to both $\mathcal{C}_\Psi$ and $[\![\mathcal{C}_\Psi]\!]$ as $\mathcal{C}_\Psi$.

Problem Statement I: Given a linear dynamical system $\mathcal{F}$, initial set Θ, a safety specification defined over a set Ψ, and time bound T, a complete characterization of counterexamples $\mathcal{C}_\Psi$ is computing the set of unique strings that *correspond to* a violation of the specification ϕ.

Fig. 4.2 exhibits that the reachable set violates ϕ (i.e., satisfies proposition p) or has a non-empty overlap with Ψ during the time interval $[0, 9.0]$. The overlap specifically occurs at the following time steps: $3^{rd}, 4^{th}, 5^{th}, 12^{th}$ and 13^{th}. In the figure, a characterization of this safety violation, denoted as 11111, represents system executions that reach unsafe set Ψ at all i.e., $3^{rd}, 4^{th}, 5^{th}, 12^{th}$ and 13^{th} time

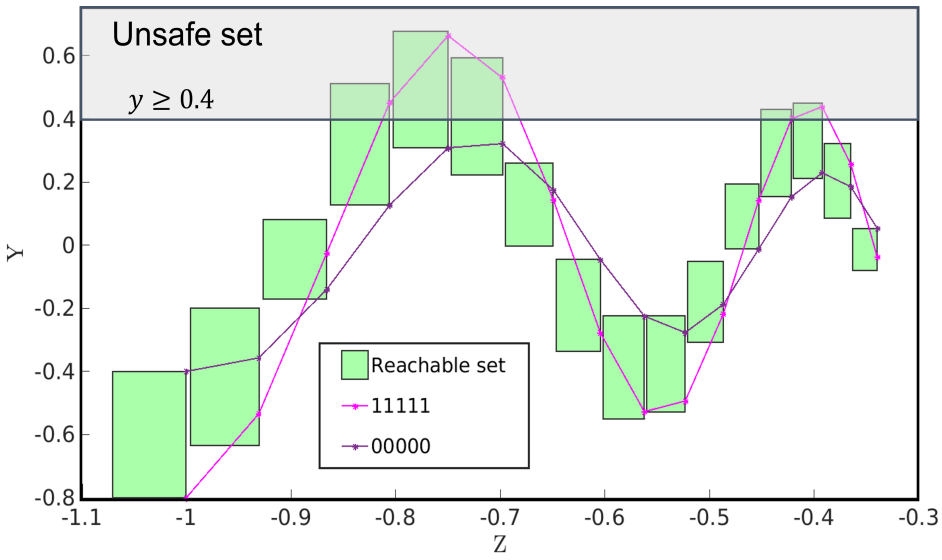


Figure 4.2: Simulation equivalent reachable set computed in HyLAA for oscillating article system with no inputs as described in 4.1.

steps. Whereas a characterization 01000 represents executions that violate the safety specification at only 4^{th} step. After having obtained these characterization, a question may arise whether there are executions that violate safety at any 2 of above time steps. The brute force way of obtaining these characterizations is to compute all binary strings of length 5, and check for the existence of a counterexample corresponding to each string. However, computing these exponential number of strings is not desirable by any practical means. Thus we would present an approach to efficiently compute the complete characterization of these counterexamples.

Before we introduce the approach to obtain complete characterization of a safety violation, we demonstrate the intuition behind characterizations of safety violation using constraint propagation.

4.2 Constraint propagation demonstration

Similar to the longest contiguous or absolute longest counterexample, we first require to perform constraint propagation. However, in contrast to only one set Π, we now maintain an additional set $\neg\Pi$ as explained next on our running example.

We define an *ordered* set $\Pi \triangleq \{S_i \cap \Psi \mid S_i \in Reach(\Theta), S_i \cap \Psi \neq \emptyset\}$. So, we have $\Pi = \{S_3 \cap \Psi, S_4 \cap \Psi, S_5 \cap \Psi, S_{12} \cap \Psi, S_{13} \cap \Psi\}$. Or, we simply write $\Pi = \{S_3, S_4, S_5, S_{12}, S_{13}\}$ where $S_i \doteq S_i \cap \Psi$. We define $\neg\Pi$ in a similar manner to obtain another ordered set $\neg\Pi = \{\neg S_3, \neg S_4, \neg S_5, \neg S_{12}, \neg S_{13}\}$

where $\neg S_i \doteq S_i \cap \neg\Psi$. These stars and their predicates denote valuations of the proposition in given temporal logic formula ϕ. Further, we only need to propagate constraints of the stars that are unsafe as explained in Section 2.5.

By using the technique described in Section 2.5, we propagate the constraints for elements in Π and $\neg\Pi$ to the initial set Θ; this results into sets Θ_Π and $\Theta_{\neg\Pi}$ respectively. Any execution that originates from $\Theta_\Pi[j] \triangleq \langle c_0, V_0, P^j\rangle \subseteq \Theta$ would reach $\Pi[j]$ (i.e., S_i) at i^{th} time step, where $P^j \doteq P \wedge P_i^U$. *Note that superscript j indexes an element in the ordered set Θ_Π while subscript i is the time step at which any execution starting from this j^{th} element reaches the unsafe set.* Similarly, an execution starting from $\Theta_{\neg\Pi}[j] \triangleq \langle c_0, V_0, \neg P^j\rangle \subseteq \Theta$ would reach $\neg\Pi[j]$ (i.e., $\neg S_i$) at i^{th} time step, where $\neg P^j \doteq P \wedge \neg P_i^\Psi$. It easily follows that $\forall j, \Theta_\Pi[j] \cap \Theta_{\neg\Pi}[j] \doteq \emptyset$ and $\Theta_\Pi[j] \cup \Theta_{\neg\Pi}[j] \doteq \Theta$. The elements in Θ_Π and $\Theta_{\neg\Pi}$ are specified in the space of same center c_0 and basis vectors V_0 of Θ. Therefore, for the ease of exposition, we can refer to the star $\Theta_\Pi[j]$ by its predicate P^j and star $\Theta_{\neg\Pi}[j]$ by $\neg P^j$. The resultant of constraint propagation step is shown in Fig. 4.3.

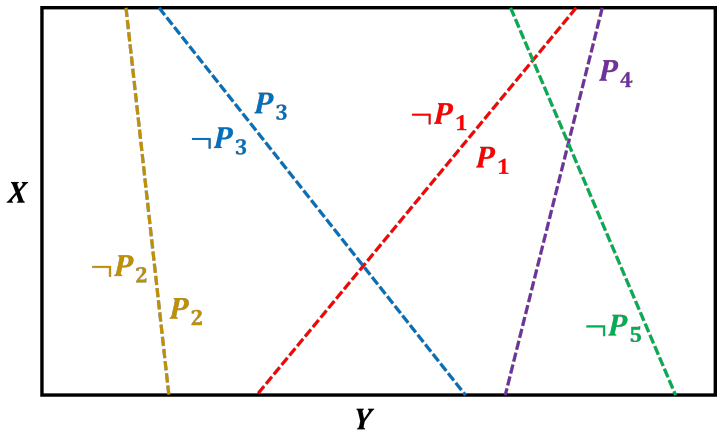


Figure 4.3: Initial set Θ after constraint propagation.

Now a system execution that visits the unsafe set Ψ at just $3^{rd}, 4^{th}, 5^{th}$ time steps can be obtained by finding a satisfiable valuation $\bar{\alpha}$ for the predicate $\left(P^1 \wedge P^2 \wedge P^3 \wedge \neg P^4 \wedge \neg P^5\right)(\bar{\alpha})$. The characterization of such counterexample is $\{\{p\},\{p\},\{p\},\{\neg p\},\{\neg p\}\}$ or $\{p, p, p, \neg p, \neg p\}$ or $\{\top, \top, \top, \bot, \bot\}$ or $\{1, 1, 1, 0, 0\}$. Similarly, the characterization of a counterexample that reaches U at only 4^{th} time instance is $\{0, 1, 0, 0, 0\}$, and the characterization of a longest counterexample is $\{1, 1, 1, 1, 1\}$. A few other characterizations of this safety violation are $\{1, 1, 0, 0, 0\}$, $\{0, 1, 1, 0, 1\}$, and $\{1, 0, 0, 1, 0\}$ etc. A characterization is *valid* if associated system of predicates is *feasible*. For example, the characteri-

zation $\{1, 1, 0, 1, 0\}$ is *valid* because $\left(P^1 \wedge P^2 \wedge \neg P^3 \wedge P^4 \wedge \neg P^5\right)(\bar{\alpha}) = \top$. But a characterization $\{1, 0, ?, ?, ?\}$ is not valid because $\left(P^1 \wedge \neg P^2\right)(\bar{\alpha}) = \bot$.

These multiple characterizations can provide some insights behind a safety violation thus assisting a control designer in state space exploration. Nonetheless, the main challenge remains is the combinatorial nature of the problem. The strings can be exponential in the number of times the reachable set enters the unsafe set. In a general case, there are $(2^m)^k$ binary strings for a violation of the specification with m propositions. But computing and checking validity of exponential number of strings is both computationally inefficient and practically undesirable. A binary decision diagram (BDD) yields a tractable and efficient representation of binary functions and strings. Therefore we use a decision diagram for generating complete characterization of counterexamples.

4.3 Binary Decision Diagram

A *Binary Decision Diagram* (BDD) (Akers 1978) is a graphical data structure for representing a boolean function. It compactly represents a set of satisfiable assignments (solutions) to decision variables z_i for a given boolean function $f : \mathbb{B}^n \to \mathbb{B}$ i.e.,

$$f(z_1, z_2, \ldots, z_n) \to \{0, 1\}, z_i \in \{0, 1\}.$$

It is typically represented as a *rooted*, *directed* and *acyclic* graph which consists of several (variable) nodes and two terminal (valued) nodes namely `False(0)` and `True(1)`. Each variable node is labeled by a decision variable z_i and has two children - *low* and *high*. The directed edge from variable node z_i to low represents an assignment 0 to z_i and the edge to $high$ denotes the assignment 1.

Definition 21 *A BDD is said to be* ordered *(OBDD) if the variables have fixed order along all paths from root to a terminal node in the graph.*

Definition 22 *A BDD is called* reduced *(RBDD) if following two rules have been applied to the graph.*

Isomorphism: *If two variable nodes are labeled by the same decision variable and have the same set of paths starting from them (to the terminal nodes), then these nodes are called* isomorphic.

One of them is removed and the incoming edges to the removed node are redirected to its isomorphic node.

Elimination: *If a variable node u has isomorphic children, then remove v and redirect its incoming edges to either of its children.*

A BDD which is both *ordered* and *reduced* is called Reduced Ordered Binary Decision Diagram (ROBDD). We only apply *isomorphism* rule to our OBDD. The term BDD refers to OBDD (not necessarily reduced) in this paper. In an ordered BDD, each non-terminal level is associated with only one decision variable. The size of a diagram is measured in its total number of nodes ($\mathcal{N}$) and its width ($\mathcal{W}$) which is defined as the maximum number of nodes at any level.

The ordering O for some $k \in \mathbb{N}^+$ is one of the permutations $k!$ of numbers $1, 2, \ldots, k$. Applying an ordering to a set of cardinality k results into its elements arranged in that order. We call this operation enumerate. Decision variable ordering plays a key role in determining the size of a decision diagram, which is observed in our evaluation results as well.

4.4 Computing complete characterization

In this section, we demonstrate how all modalities of a safety violation in linear dynamical systems can be represented using a binary decision diagram.

4.4.1 BDD construction

We first describe the basic idea. At each level (from root to terminal), a predicate (say P') is selected based on the the order that is pre-determined by O. It is then assessed whether P' renders the predicate (say P_u) of every individual node u at that level (in-)feasible. If the predicate ($P_u \wedge P'$) is evaluated to be feasible, it is added as the $high$ child of u; otherwise terminal 0 is assigned as its $high$ child. The same process is repeated with predicate $\neg P'$ for determining low child of u. Selection of predicate P' is analogous to assigning 1 to corresponding decision variable at that level while $\neg P'$ reflects the valuation 0 of the decision variable. We maintain two sets Π and $\neg\Pi$ for predicates and their negative counterparts. The union of all these (non-terminal) children computed at a particular level constitute the set of nodes for next iteration. The procedure terminates after k iterations where k is the number of decision variables.

input : Initial set: Θ, reachable set: $Reach(\Theta)$, unsafe set: Ψ, ordering: O
output : bdd: $\mathcal{G}$, its node count $\mathcal{N}$ and width $\mathcal{W}$, set of feasible paths: $\mathcal{C}_\Psi$

$\Pi \triangleq \{S_i \cap \Psi \mid S_i \in Reach(\Theta), S_i \cap \Psi \neq \emptyset\}$;
$\neg\Pi \triangleq \{S_i \cap \neg\Psi \mid S_i \in Reach(\Theta)\}$;
$\Pi, \leftarrow$ enumerate(Π, O);
$\neg\Pi \leftarrow$ enumerate$(\neg\Pi, O)$;
$\Theta_\Pi \leftarrow$ propagate_constraints(Π, Θ);
$\Theta_{\neg\Pi} \leftarrow$ propagate_constraints$(\neg\Pi, \Theta)$;
$\mathcal{G} \leftarrow$ init_bdd(Θ); // creates root, terminals
$\mathcal{N} \leftarrow 1$; // for *root* node
$\mathcal{V} \leftarrow [\mathcal{G}.root]$; // *root* is at level 0
$\mathcal{N} \leftarrow \mathcal{N} + 2, \mathcal{W} \leftarrow 2$; // for terminal nodes t_0, t_1
$k \leftarrow |\Theta_\Pi|$; // # of decision variables
for $1 \leq j \leq k$ **do**
    $\mathcal{V}' \leftarrow \emptyset$; // set of nodes at level j
    for $u \in \mathcal{V}$ **do**
        $\mathcal{V}' \leftarrow \mathcal{G}$.process_node$(u, j, \mathcal{V}', \Theta_\Pi, \Theta_{\neg\Pi})$;
    end for
    $\mathcal{N} \leftarrow \mathcal{N} + |\mathcal{V}'|$; // update nodes count
    if $\mathcal{V}' > \mathcal{W}$ **then**
        $\mathcal{W} \leftarrow |\mathcal{V}'|$; // update width
    end if
    $\mathcal{V} \leftarrow \mathcal{V}'$; // progress to next level
end for
$\mathcal{C}_\Psi \leftarrow$ traverse$(\mathcal{G}.root)$;
return $(\mathcal{G}, \mathcal{N}, \mathcal{W}, \mathcal{C}_\Psi)$;

Algorithm 6: construct_bdd algorithm.

Once bdd construction is completed, it enumerates the paths (from the beginning) in the diagram for generating all characterizations of the safety violation.

The algorithm for computing characterization of a safety violation can be divided into 2 main routines.

1. construct_bdd: Algorithm 6 computes the sets Π and $\neg\Pi$, enumerates their elements as per the given ordering O, and propagates constraints (lines 3-6). As a consequence of constraint propagation, the predicates of the elements in Θ_Π and $\neg\Theta_\Pi$ are now specified in center c_0 and basis vectors V_0. So a reference to a star or its predicate in this section implicitly considers c_0 as the center and V_0 as basis vectors. Further, we use the terms node and star interchangeably because each bdd node denotes a star.

The initialization step of bdd $\mathcal{G}$ (line 7) creates its *root* node (i.e., star $\langle c_0, V_0, P\rangle$) in addition to the terminal nodes t_0 (i.e., star $\langle c_0, V_0, \bot\rangle$) and t_1 (i.e., star $\langle c_0, V_0, \top\rangle$). The diagram is constructed in the breadth-first manner where *root* is the only node at level 0. $\mathcal{V}$ denotes the set of nodes at the current level (starting from 0), and $\mathcal{V}'$ maintains nodes at the next level. In each iteration of the outer loop (lines 12 - 22), the children of every node $u \in \mathcal{V}$ are computed via process_node, and added to the set $\mathcal{V}'$. After processing all of the nodes in $\mathcal{V}$, nodes count $\mathcal{N}$ and width $\mathcal{W}$ are updated (lines 17 and 19). Additionally, $\mathcal{V}$ is replaced with $\mathcal{V}'$ (line 21) for the next iteration.

2. process_node: Algorithm 7 computes $high$ and low children of a node u at level $(j-1)$ where $j \in [1, k]$. We now discuss the technique for computing the $high$ child. Initially a node $\bar{u}$ is created that represents a star with predicate $P_{\bar{u}} \doteq P_u \wedge P^j$. Informally, this step corresponds to assigning valuation 1 to the decision variable z_j. There are 4 scenarios that are evaluated. (i) If predicate $P_{\bar{u}}$ is infeasible, terminal t_0 is assigned as u's $high$ child (line 4). (ii) If $\bar{u}$ is at the terminal level (k) and $P_{\bar{u}}$ is feasible, t_1 is assigned as the $high$ child of u (line 7). The corresponding path (beginning from root) indicates a valid characterization. Otherwise, (iii) either an isomorphic node is identified in $\mathcal{V}'$ and assigned as u's $high$ child (lines 12- 15), or (iv) $\bar{u}$ is assigned as u's $high$ child and added to the set $\mathcal{V}'$ (lines 19 and 20). Same steps are performed for computing the low child of node u by creating a node $\neg\bar{u}$ which denotes a star with predicate $P_{\neg\bar{u}} \doteq P_u \wedge \neg P^j$. The isomorphs algorithm is discussed in the next section on bdd reduction.

Theorem 3 (Correctness) *The set $\mathcal{C}_\Psi$ returned by construct_bdd algorithm (without reduction) is the complete characterization of counterexamples of length $k \in \mathbb{N}$ to a given safety specification.*

Proof 3 *We note that $\mathcal{C}_\Psi$ is the set of all binary strings of length k accepted by the BDD. We denote $\mathcal{C}_\Psi^j$ to be the set of length $j \leq k$ prefixes of the elements in $\mathcal{C}_\Psi$, so we have $\mathcal{C}_\Psi = \mathcal{C}_\Psi^k$. We next define R which is a set of all feasible binary strings of length k.*

$$R \triangleq \{(b_1, b_2, \ldots b_k) \mid \bigwedge_{i=1}^{k} Q_i(b_i)(\bar{\alpha}) = \top, b_i \in \mathbb{B}\},\ \textit{where}$$

$$Q_i(b_i) = \begin{cases} P^i, & \textit{if } b_i = 1 \\ \neg P^i, & \textit{if } b_i = 0. \end{cases}$$

It suffices to prove that $\mathcal{C}_\Psi = R$ in order to show that all strings accepted by the BDD are indeed feasible and all strings that are not accepted are not valid counterexamples.

We first show that $R \subseteq \mathcal{C}_\Psi$. Consider an element $b \doteq (b_i, b_2, \ldots, b_k) \in R$. Assuming that $b \notin \mathcal{C}_\Psi$, there exists a prefix $(b_1, \ldots, b_j) \in \mathcal{C}_\Psi^j$ of b, where $j = min\{j' \in [1, k] \mid (b_1, \ldots, b_{j'}) \in \mathcal{C}_\Psi^{j'}$ and $(b_1, \ldots, b_{j'+1}) \notin \mathcal{C}_\Psi^{j'+1}\}$. Informally, $(b_1, \ldots, b_{j'+1})$ denotes the minimal infeasible prefix *of b. It also follows from the BDD construction that $(b_1, \ldots, b_{j'+1}) \notin \mathcal{C}_\Psi^{j'+1}$* iff *$\bigwedge_{i=1}^{j'+1} Q_i(b_i)(\bar{\alpha}) = \bot$. This further implies that $\bigwedge_{i=1}^{k} Q_i(b_i)(\bar{\alpha}) = \bot$ (i.e., $b \notin R$), which is a contradiction because $b \in R$. Therefore, all feasible strings of length k are accepted by the BDD.*

We next show that $\mathcal{C}_\Psi \subseteq R$. Consider an element $b \doteq (b_i, b_2, \ldots, b_k) \in \mathcal{C}_\Psi$. Now, assuming that $b \notin R$, we have $\bigwedge_{i=1}^{k} Q_i(b_i)(\bar{\alpha}) = \bot$ (by definition of R). This implies that $\exists j \in [1, k]$ such that $\bigwedge_{i=1}^{j} Q_i(b_i)(\bar{\alpha}) = \bot$ and $\bigwedge_{i=1}^{j-1} Q_i(b_i)(\bar{\alpha}) = \top$. It follows that $(b_1, \ldots, b_j) \notin \mathcal{C}_\Psi^j$. And since $(b_1, \ldots, b_j)$ is a prefix of b, and is infeasible, we have $b \notin \mathcal{C}_\Psi$, which is a contradiction. Therefore, all strings captured by the BDD are indeed feasible.

4.4.2 Feasibility Model

The feasibility of predicate $P_{\bar{u}}(\bar{\alpha}) \triangleq A\bar{\alpha} \leq \mathbf{b}$ in process_node algorithm is examined by solving following optimization problem.

$$\max\ \mathbf{d}$$

$$\text{s.t. } (a_m)^T \bar{\alpha} \leq \mathbf{b}_m, 1 \leq m \leq |P_{\bar{u}}|,$$

input : node: u, level: j, set of nodes at level j: $\mathcal{V}'$, ordered sets Θ_{Π} and $\Theta_{\neg\Pi}$
output : updated set of nodes at level j: $\mathcal{V}'$

1 $k \leftarrow |\Theta_{\Pi}|$; // # of decision variables
2 $\bar{u} \leftarrow$ create_node$(S_u \cap \Theta_{\Pi}[j])$; // for $z_j = 1$
3 **if** $(P_{\bar{u}}(\bar{\alpha})$ == $\bot)$ **then**
4 | $u.high \leftarrow \mathcal{G}.t_0$; // u's *high* child is t_0
5 **end if**
6 **else if** $(j == k)$ **then**
7 | $u.high \leftarrow \mathcal{G}.t_1$; // a valid characterization
8 **end if**
9 **else**
10 | $Q \leftarrow \Theta_{\Pi}[j+1 \rightarrow k]$;
11 | $\neg Q \leftarrow \Theta_{\neg\Pi}[j+1 \rightarrow k]$;
12 | **for** $u' \in \mathcal{V}'$ **do**
13 | | **if** isomorphs$(\bar{u}, u', Q, \neg Q)$ == $\top$ **then**
14 | | | $u.high \leftarrow u'$; // u', $\bar{u}$ are isomorphic
15 | | **end if**
16 | **end for**
17 | **if** $u.high == \bot$ **then**
18 | | $\mathcal{G}$.add_node$(\bar{u})$; // if no isomorphic node found
19 | | $u.high \leftarrow \bar{u}$;
20 | | $\mathcal{V}' \leftarrow \mathcal{V}' \cup \bar{u}$; // add $\bar{u}$ to $\mathcal{V}'$
21 | **end if**
22 **end if**
23 $\neg\bar{u} \leftarrow$ create_node$(S_u \cap \Theta_{\neg\Pi}[j])$; // for $z_j = 0$
24 repeat steps 3-22 with $\neg\bar{u}$ for adding low child at u;
25 **return** $\mathcal{V}'$;

Algorithm 7: process_node algorithm.

where $|P_{\bar{u}}|$ is the number of constraints in predicate $P_{\bar{u}}$. Here, $\mathbf{d} \in \mathbb{R}$ is a constant as certain optimization models require explicit specification of the objective function. This problem can be modeled as either an Integer Linear Program (ILP) or an SMT query. A satisfiable valuation of basis variables $\bar{\alpha}$ is an evidence to $P_{\bar{u}}$ feasibility. The feasibility of predicate $\neg P_{\bar{u}}$ is examined in a similar manner. The satisfiable valuation of $\bar{\alpha}$ obtained at terminal t_1 (line 7) gives a representative counterexample for corresponding characterization.

Demonstration: The BDD constructed for Example 3 with ordering [1, 2, 3, 4, 5] is shown in Fig. 4.4. At each variable node, the *solid* edge denotes $high$ child while *dashed* edge is for the low child. We omit terminal node t_0 and all of its incoming edges (due to the in-feasibility of predicate $P_{\bar{u}}$) for clarity. For example, the low child of the left P^2 node, which is not shown, is t_0 because $(P^1 \wedge \neg P^2)(\bar{\alpha}) = \bot$. These feasibility results can also be easily verified with the help of Fig. 4.3. The decision diagram has total ($\mathcal{N}$) 19 nodes (including t_0) and has width ($\mathcal{W}$) 6. The number of unique paths from *root* to terminal t_1 is 9 which is also the number of characterizations ($\mathcal{C}_\Psi$) of the safety violation under consideration. Some of these paths are $11111, 11101, 11000, 01100$, and 01000. The system executions generated from respective valuations of $\boldsymbol{\alpha}$ are illustrated in Fig. 4.5. For example, the counterexample labeled as 11111 signifies a characterization where the reachable set enters Ψ at all time steps; whereas the execution 00000 denotes the characterization for which the reachable set never enters the unsafe set Ψ.

The size of the bdd constructed in this way can still grow exponentially large in k in the worst case. The size also provides an estimate on the number of times that one needs to solve feasibility of a sub-characterization. Further, the number of feasibility instances is highly dependent on the order of decision variables. For our small running example, a random ordering results into the diagram with 24 nodes and width 9. There can also be isomorphic nodes at each level and applying the *isomorphism* rule can significantly reduce the number of feasibility instances to be solved as well as size of the diagram.

4.5 BDD Reduction

In this section, we discuss a technique to obtain a reduced bdd by applying *isomorphism* to the graph nodes. The isomorphism can be applied in two different ways. In *static* approach, isomorphic nodes are removed after constructing the decision diagram. Whereas, in *dynamic* technique, isomorphic nodes are discovered while building the diagram in the top-down fashion. We implement dynamic isomorphism

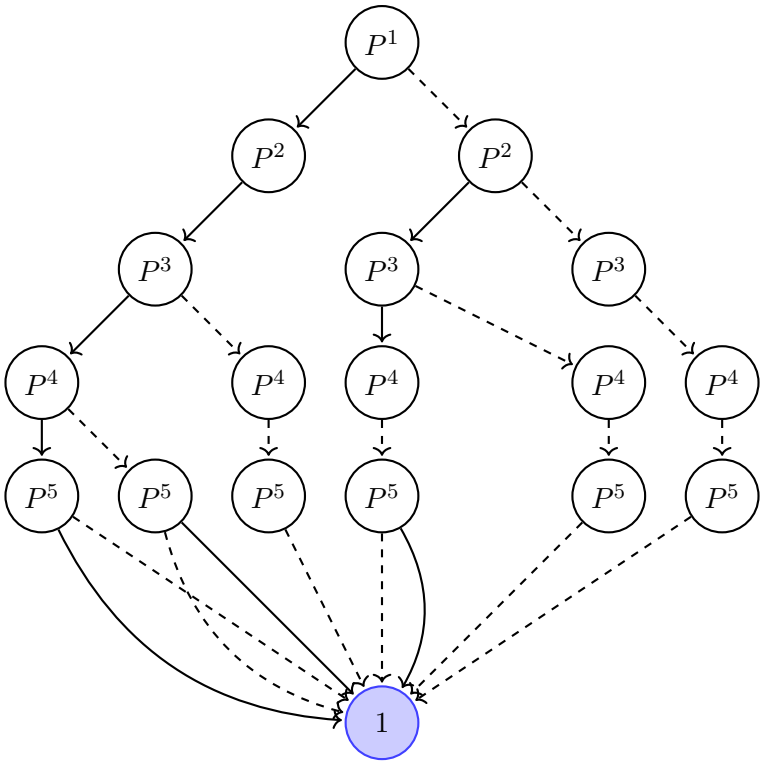


Figure 4.4: Decision diagram computed for default ordering of variables without isomorphism to represent the characterizations of counterexamples System 4.1.

for our work. We use a mathematical tool called *Farkas' Lemma* which is a theorem of alternatives for a finite set of linear constraints, for finding isomorphic nodes. It has several variants among which we adopt the one given below.

Lemma 1 (Farkas' Lemma) *Let $A \in \mathbb{R}^{m \times n}$ and $b \in \mathbb{R}^m$. Then exactly one of the following two assertions holds (Matouek & Gärtner 2006):*

1. *$\exists x \in \mathbb{R}^n$ such that $Ax \leq b$;*
2. *$\exists y \in \mathbb{R}^m$ such that $A^T y = 0, b^T y < 0$, and $y \geq 0$.*

Problem Statement II (Equivalence of predicates): Consider two predicates $\bar{P} \triangleq A_1 \bar{\alpha} \leq b_1$ and $P' \triangleq A'_1 \bar{\alpha} \leq b'_1$, and a set $\mathcal{I}$ of k predicates $P^j \triangleq A_j \bar{\alpha} \leq b_j$, $2 \leq j \leq k+1$. Is there a non-empty subset $I \subseteq \mathcal{I}$ such that the system of predicates $\bar{P} \cap I$ is *feasible* while the system with predicates $P' \cap I$ is *infeasible*, or vice-versa? If there is no such subset in both cases, we call predicates $\bar{P}$ and P' *equivalent with respect to* $\mathcal{I}$.

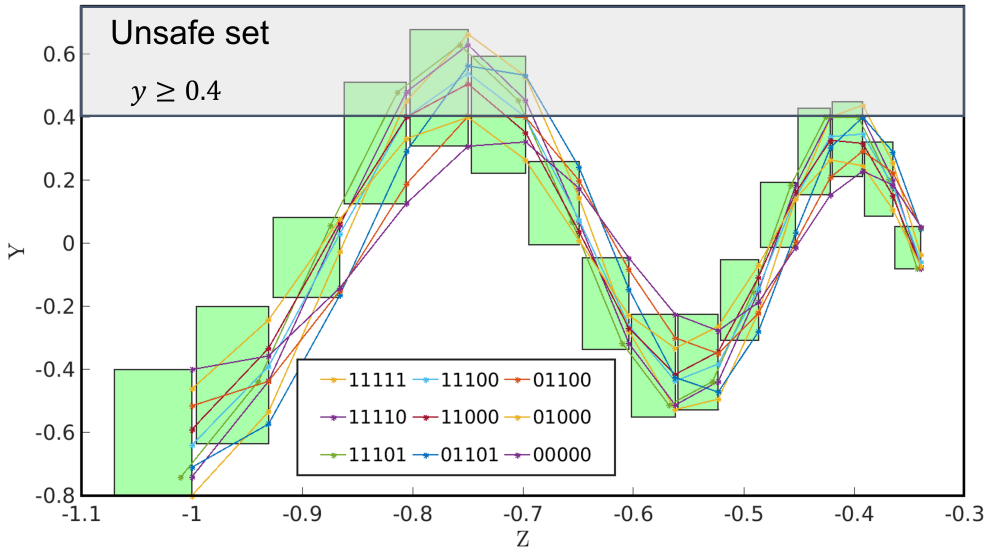


Figure 4.5: Representative executions for various characterizations of counterexamples in System 4.1.

4.5.1 System for Equivalence

The idea for checking equivalence between two predicates is as follows. Firstly, two systems are formed - one for the set of predicates $\bar{P} \cap \mathcal{I}$ while other for the set $P' \cap \mathcal{I}$. Then both systems are linked together with the help of decision variables such that feasible predicates in one of them are forced to violate in the other using Farkas' lemma. Finally, an optimization problem over those decision variables is solved to find $I \subseteq \mathcal{I}$. We now expand on this in a step-wise manner.

Step 1: The system of predicates $\bar{P} \cap \mathcal{I}$ can be represented as follows

$$\begin{pmatrix} A_1 \\ A_2 \\ A_3 \\ \vdots \\ A_{k+1} \end{pmatrix} \begin{pmatrix} \bar{\alpha} \end{pmatrix} \leq \begin{pmatrix} b_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{k+1} \end{pmatrix} \tag{4.2}$$

or,

$$H\bar{\alpha} \leq g. \tag{4.3}$$

System 4.3 is *feasible* if it has a satisfiable valuation of $\bar{\alpha} \in \mathbb{R}^n$. We denote $(H_I\bar{\alpha} \leq g_I) \doteq \bar{P}(\bar{\alpha}) \wedge \big(\bigwedge_{P^j \in I} P^j(\bar{\alpha})\big)$.

Similarly, the system with predicates $P' \cap \mathcal{I}$ is given as

$$\begin{pmatrix} A'_1 \\ A_2 \\ A_3 \\ \vdots \\ A_{k+1} \end{pmatrix} \left(\bar{\alpha}\right) \leq \begin{pmatrix} b'_1 \\ b_2 \\ b_3 \\ \vdots \\ b_{k+1} \end{pmatrix} \tag{4.4}$$

or,

$$H'\bar{\alpha} \leq g'. \tag{4.5}$$

System 4.5 is infeasible if $\exists y' \geq 0$ such that

$$H'^T y' = 0, g'^T y' < 0 \tag{4.6}$$

Step 2: Since number of variables in $\mathbf{y}'$ is same as the number of constraints in system 4.5, we have $y' = [y'_{1,1}, y'_{1,2}, \ldots, y'_{2,1}, y'_{2,2}, \ldots, y'_{k+1,|P^{k+1}|}]^T$. Further, we use y'_I to denote y' variables that correspond to the set $P' \cap I$. For example, $y'_I = [y'_{1,1}, \ldots, y'_{1,|P'|}, y'_{3,1}, y'_{3,2}, \ldots, y'_{k,|P^k|}]^T$ for $I = \{P^3, \ldots, P^k\}$. This is equivalent to dropping the rest of y' variables from system or assigning 0 to them, i.e., $y'_I = [y'_{1,1}, \ldots, y'_{1,|P'|}, 0, \ldots, 0, y'_{3,1}, y'_{3,2}, \ldots, y'_{k,|P^k|}, 0, \ldots, 0]^T$. This is essentially same as dropping columns related to predicates P^2 and P^{k+1} from matrix H'^T in system 4.6. As we are only interested in examining the feasibility w.r.t. the set I, this step ensures that the predicates in the set $\mathcal{I} \backslash I$ are dropped from both the systems.

Consequently, the problem of finding a set I in the first case is reduced to feasibility of the following system.

$$(H_I\bar{\alpha} \leq g_I), \qquad H'^T y' = 0, \qquad g'^T y' < 0, \qquad y'_I > 0. \tag{4.7}$$

Informally, a non-empty subset I indicates decision variables whose 1 valuations can make a path between $\bar{P}$ and terminal t_1 distinguishable from any path between P' and t_1. An empty I means the non-existence of such distinguishable path. But in order to ensure that $\bar{P}$ and P' are *equivalent*, we also need to check the existence of another set $I' \subseteq \mathcal{I}$. This set, if non-empty, indicates a distinguishable path (by valuation 1 of the variables in I') originating at P' from any valid path starting at $\bar{P}$. Here, I and I' can be different. By switching $\bar{P}$ and P' in equations (4.3)-(4.6), we can obtain a similar system as 4.7 to find whether such a set I' exist. Predicates $\bar{P}$ and P' are *equivalent w.r.t.* $\mathcal{I}$ if both these systems are infeasible (i.e., $I = I' = \emptyset$).

Lemma 2 *System 4.7 outputs $I = I' = \emptyset$ iff predicates $\bar{P}$ and P' are equivalent w.r.t. the set of predicates $\mathcal{I}$.*

Demonstration: Consider that $\bar{P} \doteq \neg P^1 \wedge P^2 \wedge \neg P^3$, $P' \doteq \neg P^1 \wedge P^2 \wedge P^3$, and $\mathcal{I} = \{P^4, P^5\}$ in Example 3 (The variables ordering is [1, 2, 3, 4, 5]). The solution to system 4.7 with these parameters would be $I = \emptyset$. However, the same system with $\bar{P} \doteq \neg P^1 \wedge P^2 \wedge P^3$, $P' \doteq \neg P^1 \wedge P^2 \wedge \neg P^3$ would output $I' = \{P^5\}$; hence these predicates $\bar{P}$ and P' are *not equivalent w.r.t.* the given $\mathcal{I}$. It follows that there is a valid path at node $\bar{P}$ that differs from the outgoing paths at node P' by assignment 1 to the decision variable for predicate P^5. This can also be verified using Fig. 4.4 that node $\bar{P} \doteq \neg P^1 \wedge P^2 \wedge P^3$ has 2 valid paths - "00", "01"; whereas node $P' \doteq \neg P^1 \wedge P^2 \wedge \neg P^3$ has only "00" as its valid path.

4.5.2 Equivalence-based Isomorphism

Before we discuss MILP formulation for system 4.7, we demonstrate the notion of *isomrophism* of two nodes centered on the equivalence of their predicates.

Suppose we are interested in checking the isomorphism between both children of the $root$ node in Fig. 4.6. The predicate for the $high$ child is P and that for low child is $\neg P$. For $\mathcal{I} \doteq \{\hat{P}\}$, it can be verified by solving our equivalence system that predicates P and $\neg P$ are equivalent w.r.t. $\mathcal{I}$. But we can also observe in the figure that both nodes are *not isomorphs* because the set of paths originating at respective nodes are not exactly same. We next explain how we can achieve isomorphism using equivalence.

Notice that the predicates in $\mathcal{I}$ correspond to 1 assignments to their related decision variables. And the predicates that are excluded from the solution I of System 4.7 are simply dropped and not negated. So

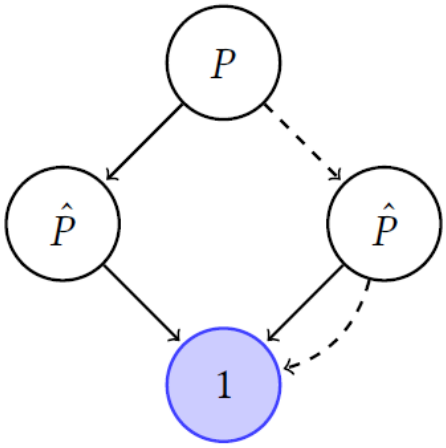


Figure 4.6: The predicates P and $\neg P$ are equivalent w.r.t. $\hat{P}$ but are not equivalent w.r.t. $\neg\hat{P}$. Thus both $\hat{P}$ nodes in the diagram are not isomorphs.

both predicates P and $\neg P$ are only shown to be equivalent for assignments 1 to the variables. However, we also need to probe whether they are equivalent for assignments 0 to the decision variables. In other words, we need to find a path, if it exists, between node $\bar{P}$ (or P') and terminal t_1 that can be distinguishable from any path originating at its counterpart by assignment 0.

For $\mathcal{I} \doteq \{\neg\hat{P}\}$, the equivalence check of predicates P and $\neg P$ fails as we obtain $I = \emptyset$ and $I' = \mathcal{I}$ by solving the equivalence system. This demonstrates that both children of the *root* node are *not isomorphs*. That is why the routine isomorphs at line 14 in Algorithm 7 has 4 arguments: nodes $\bar{u}, u'$ and sets $Q, \neg Q$. It examines the isomorphism of input nodes by performing equivalence check of predicates $P_{\bar{u}}$ and $P_{u'}$ for $\mathcal{I} \doteq \{Q\}$ as well as $\mathcal{I} \doteq \{\neg Q\}$.

Theorem 4 *For two nodes $\bar{u}, u'$ at the same level, the isomorphs algorithm returns $\top$ iff $\bar{u}$ and u' are isomorphic to each other.*

Proof 4 *We consider a simple case where both nodes have two outgoing paths (of length 1) labeled as 1 and 0 respectively. It means that we are interested in examining isomorphism between $\bar{u}$ and u' w.r.t. only one predicate (say Q) and its negation ($\neg Q$). The proof is generalizable to the paths of any finite length.*

($\Leftarrow$) Suppose isomorphs algorithm returns $\bot$ when $\bar{u}$ and u' are isomorphic. It follows from Lemma 2 that System 4.7 for examining equivalence between predicates $P_{\bar{u}}$ and $P_{u'}$ with $\mathcal{I} \doteq Q$ would output $I = I' = \emptyset$. The same system with $\mathcal{I} \doteq \neg Q$ would give $I = I' = \emptyset$. Therefore, isomorphs

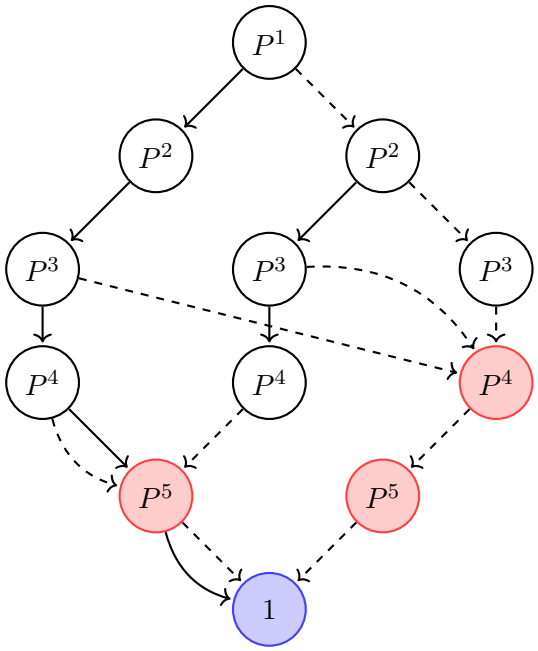


Figure 4.7: The reduced diagram for default ordering obtained upon performing *isomorphism* to express the modalities of the safety violation in System 4.1. The red-colored nodes are the ones identified as *isomorphs* to some other node(s) during BDD construction.

$(\bar{u}, u', Q, \neg Q)$ would return $\top$ which is a contradiction.

($\Rightarrow$) Suppose isomorphs returns $\top$ if $\bar{u}$ and u' are not isomorphic. However, isomorphs $(\bar{u}, u', Q, \neg Q)$ outputs $\top$ only when both predicates $P_{\bar{u}}$ and $P_{u'}$ are equivalent w.r.t. Q as well as $\neg Q$ i.e., no predicate in the set $\{Q, \neg Q\}$ can make one system feasible while making the other infeasible. This is analogous to both nodes having same set of feasible paths originating at them, thus concluding both nodes to be isomorphs which is a contradiction.

4.5.3 Linear Program for Equivalence

We model system 4.7 as a Mixed Integer Linear Program and feed it to an optimization solver. We introduce $k+1$ binary variables $z_1, z_2, \ldots, z_{k+1}$ for $k+1$ predicates in system 4.3. Then the model

(with sufficiently large M), $\forall i \in [1, k+1]$, becomes

$$\begin{aligned} &\max \sum_i z_i \\ &\text{s.t. } \left(a_{i,m}\right)^T \bar{\alpha} \leq \mathbf{b}_{i,m} + M\left(1 - z_i\right), \\ &z_i \in \{0,1\}, \bar{\alpha} \in \mathbb{R}^n, m = 1, \ldots, |P_i|. \end{aligned} \tag{4.8}$$

Note: The variables z_i that are assigned 1 by the optimization solver denote predicates in I. As we are interested in finding any non-empty subset $I \subseteq \mathcal{I}$, maximization over z_i variables subsumes all such cases. An alternative is a constant objective function with additional constraints explicitly specifying that at least one of the z_i variables is assigned 1 by the solver.

Next, the following set of constraints encodes system 4.6.

$$\begin{aligned} &\left(H'\right)^T y' \leq 0, \qquad &&\left(H'\right)^T y' \geq 0, \qquad &&g'^T y' \leq \epsilon \\ &\text{s.t. } y'_{i,m} \geq 0, \qquad &&0 < \epsilon \ll 1. \end{aligned} \tag{4.9}$$

Observe that z_i variables assigned 0 by the solver in model 4.8 are associated with the set $\mathcal{I} \backslash I$. The corresponding y'_i variables in model 4.9 are required to be assigned 0 as well. The next set of constraints enforces this requirement and completes the encoding for system 4.7 by combining models 4.8 and 4.9.

$$\begin{aligned} y'_{i,m} &\leq M * z_i, \\ y'_{i,m} &\geq \epsilon * z_i. \end{aligned} \tag{4.10}$$

The reduced binary decision diagram for ordering $[1, 2, 3, 4, 5]$ is shown in Fig. 4.7. As compared to the original BDD, the total number of nodes $\mathcal{N}$ is reduced from 19 to 13 (including t_0) and the width $\mathcal{W}$ is reduced from 6 to 3.

4.6 Extension to Systems with Bounded Inputs

Consider another version of our oscillating particle system that has one bounded input, denoted as $\mathcal{F} \triangleq \langle \mathcal{A}', \mathcal{B}', U \rangle$ where,

$$\mathcal{A}' \doteq \begin{pmatrix} -0.05 & -1.0 & 0 \\ 1.5 & -0.1 & 0 \\ 0 & 0 & -0.12 \end{pmatrix}, \mathcal{B}' \doteq \begin{pmatrix} 1 \\ 0 \\ 1 \end{pmatrix} \text{, and } U \subseteq [[-0.01, 0.01]]. \tag{4.11}$$

The input effects in the reachable set for systems with bounded inputs is computed using the Minkowski sum, and as a result, new basis variables and constraints are added to the star representation at each step. Assume that the set of states reachable after i unit time steps for system 4.11 is denoted as a generalized star $S'_i \triangleq \langle c'_i, V'_i, P'_i \rangle$. Recall from Remark 6 that every generalized star in the reachable set of our running example (without inputs) had 3 basis vectors (or state variables) and 6 constraints in its predicate (because initial set was given as a hyper-rectangle). Now from Definition 13 of Minkowski sum with stars, it follows that S'_i would have $3 + (i \times m)$ basis vectors in its basis matrix and $6 + (i \times (2 \times m))$ constraints in its predicate, where m is the number of bounded inputs (We have $m = 1$ in our case). The components for Stars S'_1 and S'_2 are as below.

$$c'_1 = [0, 0, 0, 0]^T, c'_2 = [0, 0, 0, 0, 0]^T,$$

$$V'_1 = \begin{pmatrix} 0.72 & -0.52 & 0 & 0.54 \\ 0.78 & 0.69 & 0 & 0.25 \\ 0 & 0 & 0.91 & 0.58 \end{pmatrix}, V'_2 = \begin{pmatrix} 0.11 & -0.74 & 0 & 0.26 & 0.54 \\ 1.11 & 0.07 & 0 & 0.60 & 0.25 \\ 0 & 0 & 0.86 & 0.54 & 0.58 \end{pmatrix},$$

$$\begin{aligned} P'_1 \triangleq & -0.1 \leq \boldsymbol{\alpha_1} \leq 0.1 \wedge -0.8 \leq \boldsymbol{\alpha_2} \leq 0.4 \wedge -1.07 \leq \boldsymbol{\alpha_3} \leq -1.0 \\ & \wedge -0.01 \leq \boldsymbol{u^1} \leq 0.01, \\ P'_2 \triangleq & -0.1 \leq \boldsymbol{\alpha_1} \leq 0.1 \wedge -0.8 \leq \boldsymbol{\alpha_2} \leq 0.4 \wedge -1.07 \leq \boldsymbol{\alpha_3} \leq -1.0 \\ & \wedge -0.01 \leq \boldsymbol{u^1} \leq 0.01 \wedge \wedge -0.01 \leq \boldsymbol{u^2} \leq 0.01, \end{aligned}$$

where $\boldsymbol{u^i}$ variable denotes the input applied at i^{th} time step. The simulation equivalent reachable set for system 4.11 computed for $h = 0.6$ and $\mathrm{T} = 9.0$ in HyLAA is illustrated in Figure 4.8. The reachable set violates given safety specification $\phi \triangleq \neg\Box_{[0,9.0]} y \geq 4.0$ at 5 discrete time steps:

$3^{rd}, 4^{th}, 5^{th}, 12^{th},$ and 13^{th}. The figure also illustrates executions which violate safety specification at either all 5 time steps or none.

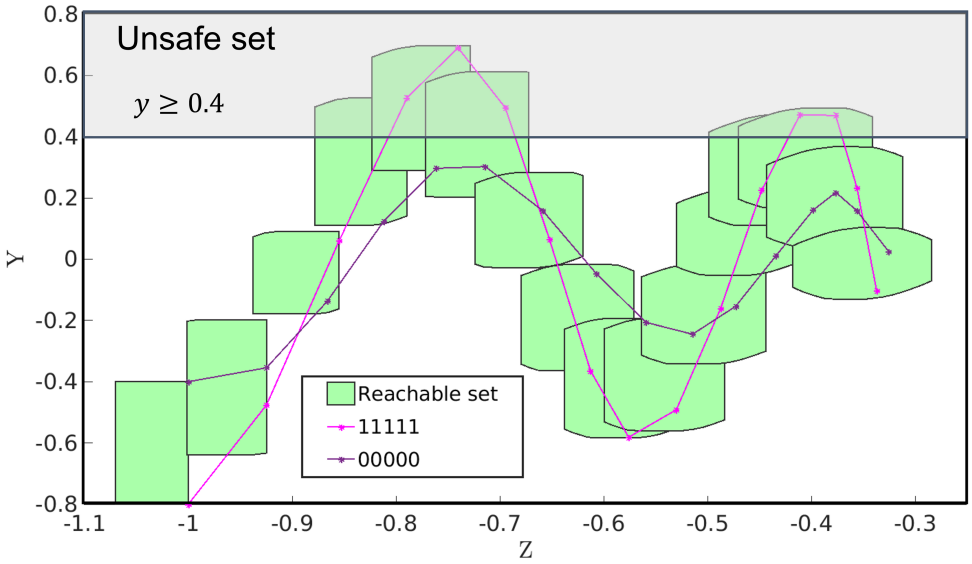


Figure 4.8: Simulation equivalent reachable set computed in HyLAA for oscillating particle system with bounded inputs as described in 4.11.

4.6.1 Transforming unsafe set constraints into star basis.

In order to compute counterexamples or their characterization, the constraints of the unsafe set Ψ need to be transformed into a star center and basis (Section 2.5 on constraint propagation). While constraints in Ψ are defined over n state variables, a star S'_i reachable after exactly i time steps has $n + (i \times m)$ basis vectors (or variables), where n is the number of dimensions of the original system and m is the number of bounded inputs. Therefore, every constraint in Ψ is first projected onto the $n + (i \times m)$ variables space by simply making the coefficients of all $(i \times m)$ variables 0.

We demonstrate this with an example. The unsafe set for the oscillating particle system is defined as a constraint $\Psi \doteq p \triangleq y \geq 0.4$, which can be denoted as $\begin{bmatrix} 0 \\ -1 \\ 0 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \end{bmatrix} \leq -0.4$. If we were to find the intersection of Ψ with star S'_2 which has 5 basis vectors (or variables), we would first project p

into the 5 variables space (by introducing dummy variables u' and u'') as $\begin{bmatrix} 0 \\ -1 \\ 0 \\ 0 \\ 0 \end{bmatrix}^T \begin{bmatrix} \boldsymbol{x} \\ \boldsymbol{y} \\ \boldsymbol{z} \\ \boldsymbol{u'} \\ \boldsymbol{u''} \end{bmatrix} \leq -0.4$, also written as $a^T x \leq \mathbf{b}$. Now this constraint can be transformed into the center and basis of the star S'_2 as $a^T(c'_2 + V'_2\bar{\alpha}) \leq \mathbf{b}$, where $\bar{\alpha} = [\boldsymbol{\alpha_1}, \boldsymbol{\alpha_2}, \boldsymbol{\alpha_3}, \boldsymbol{u^1}, \boldsymbol{u^2}]^T$.

4.6.2 BDD construction and reduction

Once constraints of the unsafe set Ψ are transformed into the center and basis of the each star that has an non-empty intersection with Ψ, an ordering O of these stars is chosen. Then Algorithm 6 is invoked to construct a BDD for representing the characterizations of the given safety violation. The decision diagram that demonstrates these characterization of the given safety violation ($y \geq 4.0$) is shown in Figure 4.9. The diagram has 23 nodes (including t_0), width as 9, and 12 unique paths or characterizations. The corresponding system executions for these 12 unique characterizations are illustrated in Figure 4.10.

However, performing BDD reduction with stars having different number of columns (or basis vectors or variables) requires an additional transformation step. From Minkowski sum operation for computing the effect of inputs (Definition 13, we know that the number of variables in the star grow at every step. However, constraints in System 4.3 or System 4.5 for finding equivalence of nodes in the BDD should be defined over same number of variables. That is, all A matrices in System 4.2 or System 4.4 should have same number of columns (or basis variables).

To address this issue, we pick $k \triangleq \arg\max_{j<\mathrm{T}}\{S'_j \mid S'_j \cap \Psi \neq \emptyset\}$ such that k denotes the largest time step at which the reachable set has a non-empty overlap with the unsafe set Ψ. Now, from the definitions of simulation-equivalent reachable set and Minkowski sum with stars, it follows that the star S'_k has the maximum number of variables among all stars having non-empty overlap with Ψ. To be precise, it has $n + k \times m$ number of variables (basis vectors) where we have $n = 3, m = 1$ for our example. For every S'_j such that $j < k$ and $S'_j \cap \Psi \neq \emptyset$, we add $(k - j) \times m$ many $\mathbf{0}$ columns to its set of constraints and $(k - j) \times m$ vectors to its basis matrix. This step makes the dimensions of each of these S'_j stars same as that of S'_k. In other words, all A matrices in System 4.2 or System 4.4 will now have same number of columns, thus making them suitable for performing equivalence as well as isomorphism

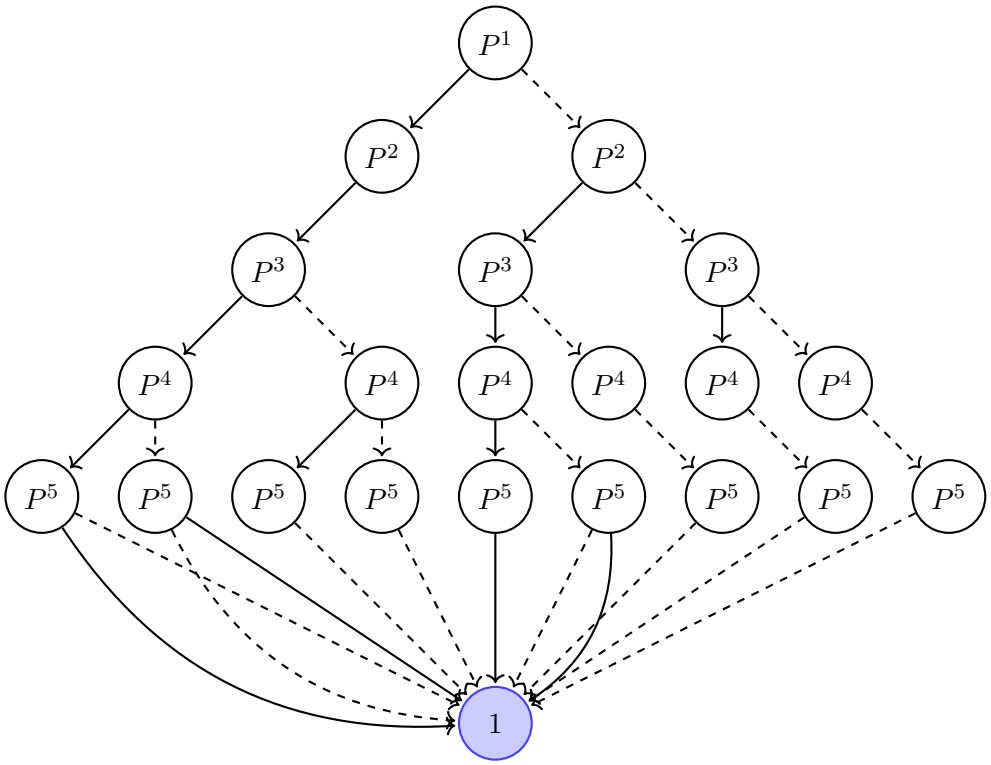


Figure 4.9: Decision diagram computed for default ordering of variables without isomorphism to represent the characterizations of counterexamples System 4.11.

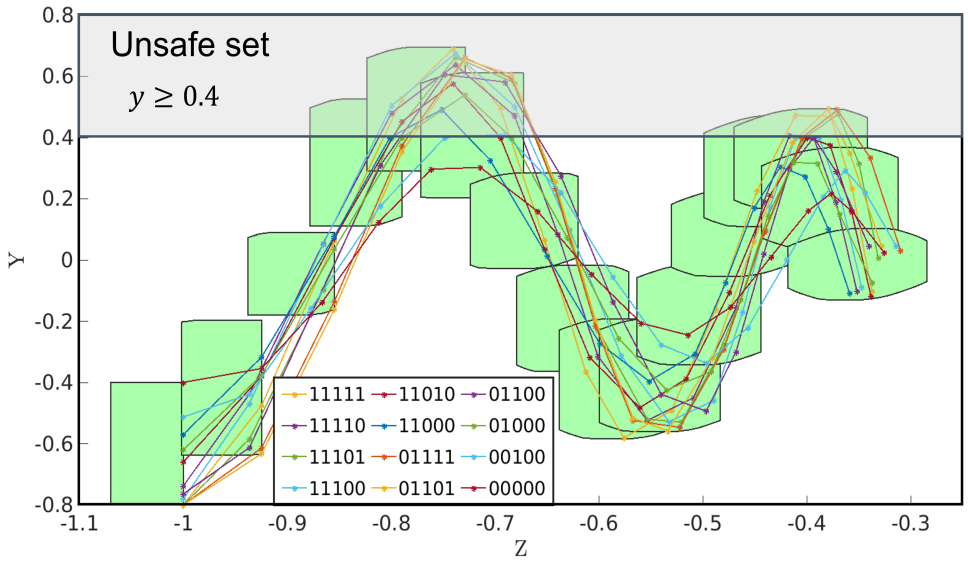


Figure 4.10: Executions for various characterizations of counterexamples in System 4.11.

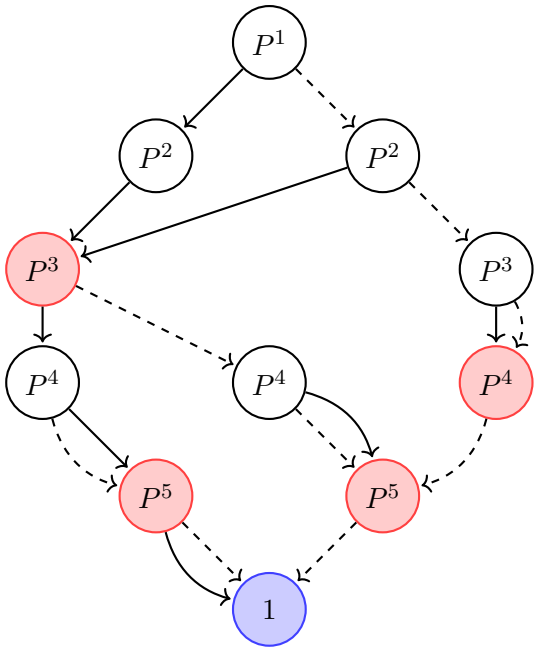


Figure 4.11: The reduced diagram for default ordering obtained upon performing *isomorphism* to express the modalities of the safety violation in System 4.11. The red-colored nodes are the ones identified as *isomorphs* to some other node(s) during BDD construction.

among BDD nodes. The final reduced bdd in Figure 4.11 has 12 nodes (including t_0), width as 3, and 14 unique characterizations. These additional 2 characterizations $(01110, 01010)$ are introduced due to merging of non-isomorphic nodes as a consequence of numerical in-stability in MILP frameworks. We discuss more on this issue in Section 4.8.

4.7 Evaluation

Implementation: The proposed algorithms have been implemented in a linear hybrid systems verification tool, `HyLAA`. Simulations for reachable sets are performed using `scipy's odeint` function, which can handle stiff and non-stiff differential equations. Linear programming is performed using the `GLPK` library, and matrix operations are performed using `numpy`. We use Gurobi (Gurobi Optimization 2018) as the optimization solver for MILP. The measurements are performed on a system running Ubuntu 18.04 with a 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM.

Benchmarks: We evaluate our techniques on various examples. "Oscillator" and "V-Platoon" are standard benchmarks from ARCH benchmark suite [2]; the ones annotated as "RS" are adopted

[2]https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

S. No.	System	Dims (n)	Verif. Time (sec)	Decision variables $\lvert\Pi\rvert$	$\lvert\mathcal{C}_\Psi\rvert$	OBDD			ROBDD			
						$\mathcal{N}$	$\mathcal{W}$	T_c (sec)	$\mathcal{N}$	$\mathcal{W}$	T_c (sec)	$\lvert\mathcal{C}_\Psi\rvert$
#1	RS-15	2	0.3	30	53	737 713 685	52 51	15 13 13	139 127 265	8 9 13	33 31 105	53
#2	RS-16	2	0.2	26	63	694 726 610	62 60	13 11 11	151 121 242	11 8 17	46 29 104	63
#3	Osc Particle	3	0.2	5	9	19 19 23	6 8	0.3	13 13 14	3 3 4	0.8 0.9 1.2	9
#4	RS-11	3	0.3	24	54	534 603 499	53	16 16 14	156 117 219	12 8 16	85 45 140	54
#5	RS-14	3	0.5	16	19	146 148 155	18	3	66 48 91	7 4 9	12 7 21	19
#6	RS-21	3	0.5	27	38	462 492 481	37	14 14 15	213 163 207	14 12 12	122 79 114	38
#7	RS-1	2	0.3	21	66	539 644 440	64 59	8 8 7	144 120 195	12 10 17	45 29 63	66
#8	RS-2	4	0.5	22	61	517 622 474	59	22 24 20	263 227 205	28 27 18	207 154 182	61
#9	RS-4	4	0.4	25	29	361 362 350	28	17 16 15	112 91 187	8 7 12	55 40 126	29
#10	RS-18	4	0.5	21	57	479 540 398	56 52	19 20 16	124 97 157	11 8 13	84 45 100	57 55 57
#11	ACC	5	0.5	18	63	460 501 336	62 53	24 24 20	118 120 142	11 12 13	80 84 118	63
#12	Quadcopter	6	0.5	19	42	413 347 292	41 39	29 25 21	60 60 95	5 4 7	39 39 72	42 40 41
#13	RS-6	8	0.6	13	14	93 93 93	13	7	53 43 54	6 5 7	40 26 43	14
#14	RS-8	8	0.6	16	25	174 186 177	24 23	18 18 17	77 57 93	8 6 10	98 53 116	25
#15	V-Platoon I	9	0.7	14	34	162 180 170	31 30	20 20 18	69 56 79	8 7 9	98 62 116	34
#16	RS-7	12	0.6	11	12	58 54 56	11	8	36 28 31	4 4 5	37 24 31	12
#17	V-Platoon II	15	0.9	24	16	110 74 138	14	40 35 61	88 52 82	4 4 6	230 116 246	16
#18	RS-9	16	1.0	15	33	205 232 204	32	68 71 62	74 63 88	9 7 10	320 221 377	33
#19	RS-10	20	1.0	14	23	145 162 140	22 21	68 71 66	65 51 65	8 6 7	318 218 364	23
#20	V-Platoon III	30	2.0	10	27	96 107 91	24 22	66 82 62	40 43 52	6 7 8	282 347 425	26 27

Table 4.1: Evaluation results for counterexamples' characterization in linear systems with no/constant inputs. The results highlighted in red are under-approximations incurred due to numerical in-stability.

S. No.	System	Dims (n)	Inputs (m)	Verif. Time (sec)	Decision variables $\lvert\Pi\rvert$	$\lvert\mathcal{C}_\Psi\rvert$	OBDD			ROBDD			
							$\mathcal{N}$	$\mathcal{W}$	$\mathrm{T_c}$ (sec)	$\mathcal{N}$	$\mathcal{W}$	$\mathrm{T_c}$ (sec)	$\lvert\mathcal{C}_\Psi\rvert$
#1	RS-1	2	1	5	18	63	447 523 352	62 55	334 361 263	103 91 138	10 8 13	1032 850 1620	63 59 63
#2	Osc Particle	2	1	1	5	12	23 24 26	9 10	2 2 2	12 13 13	3 3 3	5 6 7	14 13 12
#3	Osc Particle	2	2	1	6	17	39 40 47	16	12 12 15	17 18 17	4 4 4	30 37 38	17
#4	Osc Particle	2	3	2	6	23	43 44 52	19	29 32 39	16 18 19	4 4 4	76 89 133	23
#5	RS-14	3	1	15	16	23	171 177 166	22 21	85 87 85	73 63 95	7 7 10	400 291 614	23
#6	RS-3	4	2	8	19	35	279 302 249	34 32	424 476 481	102 77 119	9 7 11	2228 1294 2510	35
#7	RS-6	8	1	7	15	28	190 204 182	27	153 158 138	60 50 73	6 5 8	558 385 720	27
#8	RS-6	8	2	24	15	32	211 229 200	31	472 459 460	65 52 86	8 5 10	1684 1137 2815	32 31
#9	RS-8	8	1	4	18	33	246 269 228	31 30	371 440 365	98 73 102	9 7 9	1943 1208 2052	33
#10	V-Platoon I	9	1	7	15	64	338 399 360	61	581 588 580	74 70 116	9 7 15	1612 1239 3389	64 60 62
#11	RS-7	12	1	9	12	26	137 146 133	25	117 125 111	48 39 62	7 5 9	459 280 691	26
#12	RS-7	12	2	27	12	22	117 120 119	21	251 260 247	44 38 62	6 4 9	901 719 1740	22

Table 4.2: Evaluation results for complete characterization of counterexamples in linear systems with bounded inputs. The results highlighted in red are under-/over-approximations of complete characterization incurred due to numerical in-stability.

from `RealSyn` (Fan et al. 2018). "Adaptive Cruise Control (ACC)" and "Quadcopter" are adopted from (Tiwari 2003) and (Argentim, Rezende, Santos & Aguiar 2013), respectively. We take the dynamics ($\mathcal{A}$ and $\mathcal{B}$ matrices) from the original benchmark for each system while modifying the safety specification suitable for our work.

Variables ordering: We apply 3 different orderings of decision variables to construct 3 different decision diagrams for performance evaluation. For a monotonically increasing sequence of numbers $[1, 2, 3, 4, 5]$, the variables in default ordering O_d are arranged as per this sequence. Another ordering O_r is *random* where elements in a set are arranged in some random order. Additionally, we make an observation about the reachable set overlap with the unsafe set which yields third ordering O_m for our experiments. In most of the cases, we observe that the star which goes the deepest in the unsafe set lies in the middle of the sequence of the unsafe stars, thus it tends to have non-empty intersection with relatively more number of stars in its neighborhood. We use this intuition to obtain another ordering $[3, 2, 1, 4, 5]$ that has only the first half of the given sequence reversed which, in turn, brings the middle element to the front.

Performance: Table 4.1 demonstrates our evaluation results for system without (or constant) inputs. The results for systems with bounded inputs are demonstrated in Table 4.2. *Dims* is the number of system variables, *Verification Time* is the time `HyLAA` takes to compute the reachable set. $\mathrm{T_c}$ is the time taken by construct_bdd Algorithm. $\mathcal{C}_\Psi$ denotes the number of characterizations for a given safety violation in each system. Further, in each benchmark, the first row is for default ordering O_d, the second one is for O_m ordering, and the third row corresponds to the random ordering O_r. We measure the performance of our algorithms in terms of bdd creation time ($\mathrm{T_c}$), total number of nodes ($\mathcal{N}$), and its width ($\mathcal{W}$). The verification time for computing the reachable set is significantly less as compared to $\mathrm{T_c}$. Further, reduced bdd construction[3] takes much more time because we solve multiple optimization problems at each step to find isomorphic nodes.

We now make a few key observations. (i) For OBDD runs in both tables, the random ordering O_r turns out to be slightly better than its counterparts in the BDD size ($\mathcal{N}$ and $\mathcal{W}$). On the other hand, O_m fares better in ROBDD experiments in almost all cases, which supports our observation about the middle element probably having non-empty intersection with multiple elements. Figures 4.12 and 4.13 plotted

[3]Currently our implementation of the bdd reduction algorithm supports only *isomorphism* reduction rule.

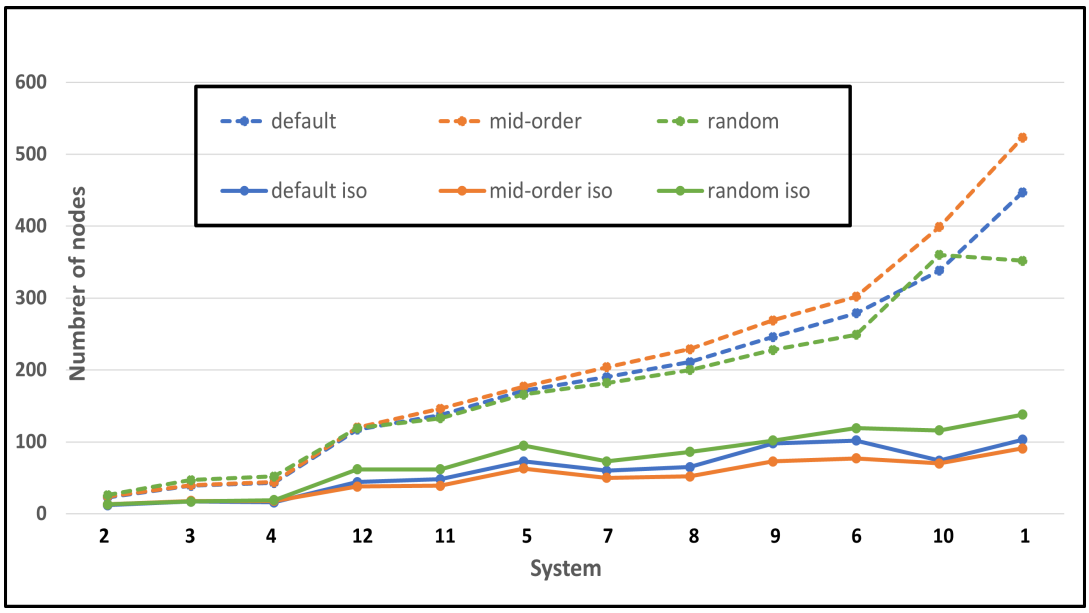


Figure 4.12: Number of nodes in diagrams for systems with bounded inputs. Systems are ordered in increasing number of BDD nodes. Dashed curves correspond to the diagrams without reduction, whereas solid ones correspond to their reduced counterparts.

for systems with bounded inputs support this observation where the *mid-order iso* scenario yields the most succinct representations. (ii) The figures also highlight that the reduced BDD size remains fairly small; that is, its size does not grow in proportion to the size of the original diagram. (iii) In some cases, the BDD size ($\mathcal{N}$) is reduced by upto 80% across OBDD and ROBDD runs, while in other cases, we only see a reduction by 50%. If we take the summation of the total number of nodes in all benchmarks across all orderings, we observe an overall reduction of $65-67\%$ in $\mathcal{N}$ across OBDD and ROBDD runs. Similarly, we notice an overall reduction of $74-76\%$ in width ($\mathcal{W}$) across all benchmarks and orderings.

4.8 Discussion

Numerical in-stability: Recall that we model node isomorphism as an optimization problem encoded as an MILP. An issue with an MILP-based approach even with a state-of-the-art commercial solver is numerical in-stability. The model requires the definition of M, which is larger than any value, and an arbitrarily small $\epsilon > 0$. An ϵ smaller than the solver's tolerance may translate into $\mathbf{y}'$ variables becoming 0 in system 4.10. This results into isomorphs algorithm returning $\perp$ even when two nodes are actually isomorphs. Consequently, ROBDD may end up having same size as OBDD. On the other hand, a higher ϵ value can sometimes lead to isomorphs returning $\top$ even when nodes are not isomorphic

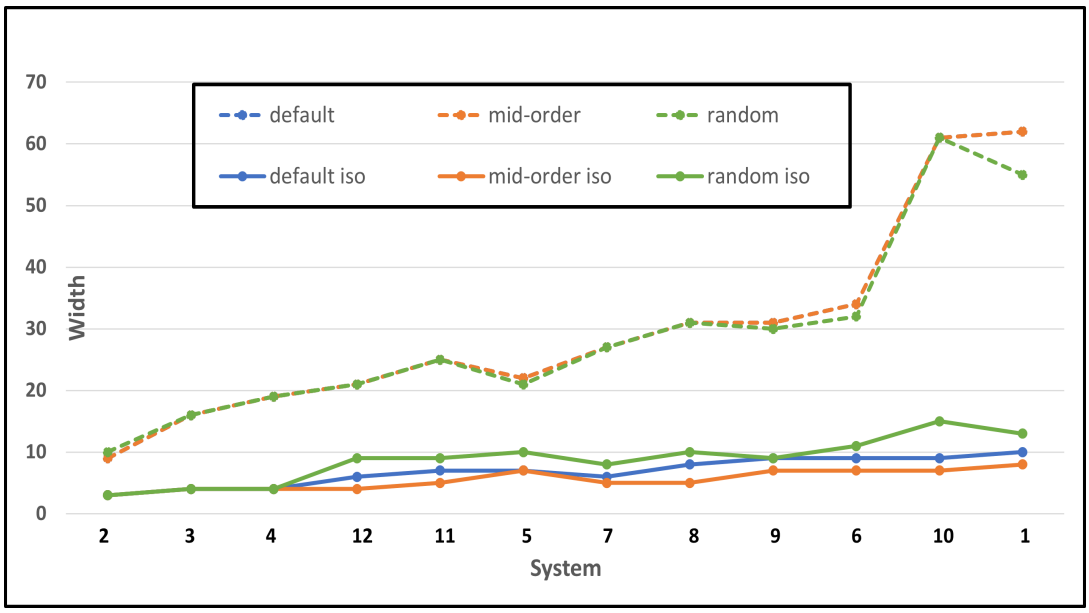


Figure 4.13: Width of the diagrams for systems with bounded inputs. Systems are ordered in the same order as in Figure 4.12. Dashed curves correspond to the diagrams without reduction, whereas solid ones correspond to their reduced counterparts. The curves for default and mid-order OBDDs completely overlap hence only one is visible in the plot.

to each other. As a result, a few infeasible characterizations can get introduced (over-approximation) and/or some feasible ones may get dropped (under-approximation) from the diagram. Our evaluations highlight that such approximations occur only in a few cases, that too with a minor difference from the complete solution. Another key point to observe is that there is no correlation between variables ordering and numerical in-stability because it is reported in all 3 orderings for different systems.

Variations in the size: We report in Table 4.3 the $\%$-variation σ in the BDD size ($\mathcal{N}$) for systems with no/constant inputs. We define $\sigma \triangleq ((\mathcal{N}_{max} - \mathcal{N}_{min}) \div \mathcal{N}_{max}) \times 100$, where $\mathcal{N}_{max}$ (or $\mathcal{N}_{min}$) is the highest (or lowest) number of nodes among any of the orderings O_d, O_m and O_r. The table underlines the importance of the decision variables ordering. It demonstrates that the BDD size can vary significantly across different orderings, and one can potentially achieve $25\% - 50\%$ reduction in the BDD size with appropriate ordering.

Application to additional specifications: We have maintained so far that the diagrams 4.4 or 4.7 represent complete characterization of the violation of the safety specification $\phi \triangleq \mathrm{G}_{[0,20]}\neg p$. We argue that the same diagrams also subsumes the characterization of the violation of some other safety specifications such as $\phi' \triangleq \mathrm{F}_{[0,5]}p \rightarrow \mathrm{G}_{[6,15]}\neg p$ and $\phi'' \triangleq \mathrm{G}_{[0,10]}\neg p \rightarrow \mathrm{F}_{[11,15]}\neg p$. The specification ϕ'

Table 4.3: $\%$-variations (σ) in the number of BDD nodes ($\mathcal{N}$) due to different orderings in systems with no/constant inputs

System	σ		System	σ	
	OBDD	ROBDD		OBDD	ROBDD
#1	7.1	52.1	#11	32.9	17.0
#2	16.0	50.0	#12	29.3	42.9
#3	16.7	0	#13	0	28.6
#4	17.2	46.6	#14	6.5	40.0
#5	5.8	47.2	#15	10.0	22.2
#6	6.1	23.5	#16	6.9	20.0
#7	31.7	38.5	#17	46.4	33.3
#8	23.8	22.1	#18	12.1	30.0
#9	3.3	51.3	#19	13.6	25.0
#10	26.3	38.2	#20	15.0	25.0

states that if something bad happens within first 5 time steps then nothing bad would happen between time interval $[6, 15]$. On the other hand, ϕ'' states that if nothing bad happens in first 10 time steps then something good will eventually happen in the interval $[11 - 15]$. This observation is important for 2 key reasons - (i) existing verification artifacts can be reused for safety verification w.r.t. such specifications, and (ii) it is not required to construct individual decision diagram for each such specification.

Extension to hybrid systems: A linear hybrid system has multiple *modes* (also called *locations*) of operation and discrete transitions as switching function between modes. These transitions labelled by *guard* condition induce non-determinism which can possibly lead to multiple paths in the reachable set. Therefore, the reachable set in linear hybrid systems is denoted in the form of a tree (rooted at Θ) called $ReachTree$. We can construct a bdd for each path in $ReachTree$ for generating respective characterization of counterexamples. The detailed explanation on $ReachTree$ computation and counterexample generation in linear hybrid systems is present in (Goyal & Duggirala 2020*a*).

4.9 Chapter Summary

This chapter has introduced the notion of complete characterization of counterexamples to a given safety specification in linear hybrid systems. The presented work reuses the artifacts generated during the model checking process and employs constraint propagation to construct a a graphical structure called *binary decision diagram* (BDD) for representing all modalities of a safety violation. It has introduced an approach for examining node isomorphism leading to a much more succinct BDD representation. It has provided a Mixed Integer Linear Program (MILP) formulation of the system that models node

isomorphism and conducted an exhaustive evaluation on numerous benchmarks to exhibit the performance of our approach.

Acknowledgement. David Bergman (Associate Professor, Business School, UConn) has been involved throughout the project, and in particular, helped in encoding an MILP to perform node isomorphism in a BDD.

CHAPTER 5: NEURALEXPLORER: STATE SPACE EXPLORATION OF CLOSED LOOP CONTROL SYSTEMS USING SENSITIVITY APPROXIMATION

The analysis of non-linear and learning enabled control systems is particularly challenging because standard analytical tools for linear systems do not easily extend to them in the absence of closed form expression for non-linear ODEs. In this chapter, we propose a framework for performing state space exploration of complex closed loop control systems. Our approach involves approximating sensitivity and a newly introduced notion of inverse sensitivity by a neural network. We show how the approximation of inverse sensitivity can be used for performing state space exploration by generating trajectories that reach a neighborhood. We also discuss a sensitivity approximation based technique for estimating the trajectories of the given system. [1]

5.1 Learning the inverse sensitivity function using observed trajectories

For testing the system operation in a given domain $\mathbb{D} \subseteq \mathbb{R}^n$, one may wish to generate a finite set of trajectories. Often, these trajectories are generated using numerical ODE solvers which return system simulations sampled at a regular time step. The step size, time bound, and the number of trajectories are specified by the user. Given a sampling of a trajectory with step size h, i.e., $\xi(x_0, 0)$, $\xi(x_0, h)$, $\xi(x_0, 2h)$, ..., $\xi(x_0, kh)$, we make a few observations. First, any prefix of this sequence is also a trajectory of a shorter duration. Hence, from a given set of trajectories, one can truncate them to generate more trajectories having shorter duration. Second, given two trajectories starting from states x_0 and x_0',

[1] Contents of this chapter previously appeared in preliminary form in the following papers:

Goyal, Manish and Parasara Sridhar Duggirala. 2020. **NeuralExplorer:** State Space Exploration of Closed Loop Control Systems Using Neural Networks. In *Proceedings of the 2nd Annual Conference on Learning for Dynamics and Control*.

Goyal, Manish and Parasara Sridhar Duggirala. 2020. **NeuralExplorer:** State Space Exploration of Closed Loop Control Systems Using Neural Networks. In *Proceedings of the Automated Technology for Verification and Analysis*.

($x_0 \neq x'_0$), we can compute the following values for the sensitivity functions:

$$\Phi(x_0, x'_0 - x_0, t) = \xi(x'_0, t) - \xi(x_0, t) = x'_t - x_t \tag{5.1}$$

$$\Phi^{-1}(x_t, x'_t - x_t, t) = x'_0 - x_0 \tag{5.2}$$

Note that we can estimate values of Φ^{-1} based only on samples from a forward simulator ξ (rather than requiring a simulation from ξ^{-1}).

Let us explain how we generate values of the function $\Phi^{-1}(x_t, v, t)$ (or $\Phi(x_0, v, t)$) in order to learn an approximator $N_{\Phi^{-1}}(x_t, v, t)$ (or $N_{\Phi}(x_0, v, t)$). First, start with a set of reference trajectories generated from randomly sampled initial points, we generate prefixes of these reference trajectories. Then we iterate over each pair of these prefixes and use Equations 5.1 and 5.2 for generating tuples $\langle x_0, v, t, v_+ \rangle$ and $\langle x_t, v, t, v_- \rangle$ such that $v_+ = \Phi(x_0, v, t)$ and $v_- = \Phi^{-1}(x_t, v, t)$. We use these tuples to train either a forward sensitivity approximator denoted as N_Φ or an inverse sensitivity approximator $N_{\Phi^{-1}}$.

The training performance for various benchmark systems and neural network architecture is detailed in the following section.

5.2 Benchmarks and Training Performance

For approximating the sensitivity and inverse sensitivity functions, we pick a standard set of benchmarks consisting of nonlinear dynamical systems, hybrid systems, and a few control systems with neural network feedback functions. Most of the benchmarks are taken from a standard hybrid systems benchmark suite[2] and (Immler, Althoff, Chen, Fan, Frehse, Kochdumper, Li, Mitra, Tomar & Zamani 2018, Bak, Beg, Bogomolov, Johnson, Nguyen & Schilling 2019). The benchmarks *Brussellator, Lotka, Jetengine, Buckling, Vanderpol, Lacoperon, Roesseler, Steam, Lorentz* and *Coupled vanderpol* are continuous nonlinear systems, where *Lorentz* and *Roesseler* are *chaotic* as well. *SmoothHybrid-/Hybrid-oscillator* are nonlinear hybrid systems. The remaining 2 benchmarks *Mountain Car* and *Quadrotor* are selected from (Ivanov, Weimer, Alur, Pappas & Lee 2019), where the state feedback controller is given in the from of neural network. In *Quadrotor* benchmark, we induce determinism by fixing the control law based on 8 control actions generated by the controller.

[2]https://ths.rwth-aachen.de/research/projects/hypro/benchmarks-of-continuous-and-hybrid-systems/

For each benchmark, we generate a given number (N is typically 30 or 50) of trajectories, where the step size for ODE solvers (s) and the time bound are provided by the user. The data used for training the neural network is collected as described in 5.1. We use 90% of the data for training and 10% for testing. We use the Python Multilayer Perceptron implemented as Sequential model in `Keras` (Chollet et al. 2015) library with Tensorflow as the backend. The network has 8 layers with each layer having 512 neurons. The optimizer used is stochastic gradient descent. The network is trained using Levenberg-Marquardt backpropagation algorithm optimizing the mean absolute error loss function, and the Nguyen-Widrow initialization.

The activation function used to train the network is **relu** for all benchmarks except *Mountain car* for which **sigmoid** performs better because the NN controller is trained with **sigmoid** activation. Note that the choice of hyper-parameters such as number of layers and neurons, the loss and activation functions is empirical, and is motivated by some prior work (Goyal & Duggirala 2019). We evaluate the network performance using root mean square error (MSE) and mean relative error (MRE) metrics. The training and evaluation are performed on a system running Ubuntu 18.04 with a 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM. The network training time, MSE and MRE are given in Tables 5.1 and 5.2 respectively. Time bound is number of steps for which the system simulation is computed.

Benchmark		Dims	Step size (sec)	Total steps	Training Time (min)	MSE	MRE
Continuous Nonlinear Dynamics	Brussellator	2	0.01	500	40.0	0.14	0.34
	Buckling	2	0.01	500	25.0	2.38	0.18
	Lotka	2	0.01	500	27.0	0.38	0.31
	Jetengine	2	0.02	300	35.0	0.086	0.63
	Vanderpol	2	0.01	500	75.50	0.15	0.29
	Lacoperon	2	0.1	500	66.0	0.12	0.33
	Roesseler	3	0.02	500	42.0	0.58	0.087
	Lorentz	3	0.01	500	22.0	1.08	0.11
	Steam	3	0.01	500	35.0	0.34	0.07
	C-Vanderpol	4	0.01	500	70.0	0.18	0.15
Hybrid/ NN Systems	HybridOsc.	2	0.01	500	80.0	0.35	0.11
	SmoothOsc.	2	0.01	500	38.5	0.40	0.096
	Mountain Car	2	-	100	12.5	0.015	0.79
	Quadrotor	6	0.01	120	40.0	0.064	0.20

Table 5.1: Training results for sensitivity function approximator N_Φ in NeuralExplorer. Dims is the number os system variables. MSE and MRE are respectively mean squared error and mean relative error.

Benchmark		Dims	Step size (sec)	Total steps	Training Time (min)	MSE	MRE
Continuous Nonlinear Dynamics	Brussellator	2	0.01	500	67.0	1.01	0.29
	Buckling	2	0.01	500	42.0	0.59	0.17
	Lotka	2	0.01	500	40.0	0.50	0.13
	Jetengine	2	0.01	300	34.0	1.002	0.26
	Vanderpol	2	0.01	500	45.50	0.23	0.23
	Lacoperon	2	0.2	500	110.0	1.8	0.46
	Roesseler	3	0.02	500	115.0	0.44	0.07
	Lorentz	3	0.01	500	67.0	0.48	0.08
	Steam	3	0.01	500	58.0	0.13	0.057
	C-Vanderpol	4	0.01	500	75.0	0.34	0.16
Hybrid/ NN Systems	HybridOsc.	2	0.01	1000	77.0	0.31	0.077
	SmoothOsc.	2	0.01	1000	77.5	0.23	0.063
	Mountain Car	2	-	100	10.0	0.005	0.70
	Quadrotor	6	0.01	120	25.0	0.0011	0.16

Table 5.2: Training results for inverse sensitivity function approximator $N_{\Phi^{-1}}$ in NeuralExplorer.

5.3 Space Space Exploration Using an Approximator

We now present various applications in the domain of state space exploration using the neural network approximations of sensitivity and inverse sensitivity. The goal of state space exploration is to search for trajectories that satisfy or violate a given specification. We primarily concern ourselves with a safety specification, that is, whether a specific trajectory reaches a set of states labelled as *unsafe*. However, this unsafe set can denote the violation of any performance specification (Remark 1). In order to search for such trajectories, we present various empirical techniques that use both forward and inverse sensitivity.

5.3.1 Reaching a Specified Destination Using Inverse Sensitivity Approximator

In the course of state space exploration, after testing the behavior of the system for a given set of test cases, the control designer might choose to explore the behavior of a system that reaches a destination or approaches the boundary condition for safe operation. Given a domain of operation D, we assume that the designer provides a desired target state z (with an error threshold of δ) that is reached by a trajectory at time t. Our goal is to generate a trajectory ξ such that it visits a state in the δ neighborhood of the target z at time t.

Our approach for generating the target trajectory is as follows. First, we generate an anchor trajectory ξ_A from a randomly sampled state in the initial set Θ, and compute the difference vector of target state z

and $\xi_A(t)$. We now use the inverse sensitivity approximator $N_{\Phi^{-1}}$ to estimate the perturbation required at the initial state such that the trajectory after time t goes through z. Since the neural network can only approximate the inverse sensitivity function, the trajectory after the perturbation need not visit δ neighborhood of the destination. However, we can repeat the procedure until a threshold on the number of iterations $(\mathcal{K})$ is reached or the δ threshold is satisfied. The pseudo code of this procedure called *ReachTarget* abbreviated as $\mathcal{RT}$, is given in Algorithm 8. To summarize, it estimates the inverse sensitivity at each step to displace the initial state, generates the simulation from modified initial state, and treats this simulation as the new anchor. k is the number of simulations generated.

input : simulator: ξ, time instance: $t \leq T$, reference trajectory: ξ_A, destination: $z \in \mathbb{D}$, Iterations bound: $\mathcal{K}$, function $N_{\Phi^{-1}}$ that approximates Φ^{-1}, initial set: Θ, and threshold: δ.

output : iterations: k, final trace: $\xi(x_0^k, \cdot)$, final distance: d_a^k, final relative distance: d_r

1 $x_0^0, x_t^0 \leftarrow \xi_A(0), \xi_A(t)$; // states at time 0 and t

2 $v^0 \leftarrow z - x_t^0$; // initial vector difference with z

3 $d_{init} \leftarrow d_a^0 \leftarrow \|v^0\|$; // initial distance

4 $k \leftarrow 0$;

5 **while** $(d_a^k > \delta)$ & $(k < \mathcal{K})$ **do**

6 $\quad \hat{v}_-^k \leftarrow N_{\Phi^{-1}}(x_t^k, v^k, t)$; // predict v_-^k

7 $\quad x_0^k \leftarrow x_0^k + \hat{v}_-^k$; // perturb x_0^k

8 $\quad x_0^{k+1} \leftarrow x_0^k$;

9 $\quad \xi_A^{k+1} \leftarrow \xi(x_0^{k+1}, \cdot)$; // new anchor

10 $\quad x_t^{k+1} \leftarrow \xi_A^{k+1}(t)$ $v^{k+1} \leftarrow z - x_t^{k+1}$; // new vector difference

11 $\quad d_a^{k+1} \leftarrow \|v^{k+1}\|$; // update distance to z

12 $\quad k \leftarrow k + 1$; // increment corrections by 1

13 **end while**

14 $d_r \leftarrow d_a^k / d_{init}$; // update relative distance

15 **return** (k, ξ_A^k, d_a^k, d_r);

Algorithm 8: *ReachTarget* $(\mathcal{RT})$ algorithm

5.3.2 Evaluation of *ReachTarget* on Standard Benchmarks

We evaluate the performance of *ReachTarget* (or, $\mathcal{RT}$) algorithm by randomly sampling a target state z in the domain of interest and let $\mathcal{RT}$ generate a trajectory that goes through the δ-neighborhood of the target at a specified time t. Typically, $\mathcal{RT}$ executes the loop in lines 2-12 for ~ 10 times before arriving in the δ-neighborhood of the target.

Benchmark	Dims	Iteration count = 1			Iteration count = 5		
		d_a	d_r	Time (ms)	d_a	d_r	Time (ms)
Brussellator	2	[0.19 - 1.87]	[0.23 - 0.74]	11.38	[0.003- 0.22]	[0.01 - 0.12]	31.34
Buckling	2	[1.67 - 11.52	[0.17 - 0.45]	13.61	[0.36- 2.09]	[0.06 - 0.31]	34.51
Lotka	2	[0.08 - 0.24]	[0.21 - 0.45]	12.38	[0.02 - 0.07]	[0.09 - 0.22]	34.28
Jetengine	2	[0.05 -0.20]	[0.19 - 0.28]	15.96	[0.0004 - 0.05]	[0.006 - 0.14]	38.26
Vanderpol	2	[0.29 - 0.58]	[0.16 - 0.66]	12.34	[0.03 - 0.18]	[0.04 - 0.16]	34.02
Lacoperon	2	[0.03 - 0.13]	[0.12 - 0.28]	17.18	[0.003 - 0.03]	[0.02 - 0.16]	37.34
Roesseler	3	[0.72 - 2.02]	[0.20 - 0.34]	16.08	[0.21 - 0.63]	[0.06 - 0.14]	38.26
Lorentz	3	[1.24 - 5.60]	[0.29 - 0.58]	24.72	[0.20 - 0.70]	[0.05 - 0.17]	60.18
Steam	3	[1.59 - 5.21]	[0.31 - 0.67]	8.68	[0.41 - 1.8]	[0.08 - 0.30]	69.80
C-Vanderpol	4	[0.87 - 1.72]	[0.34 - 0.60]	17.44	[0.20 - 0.40]	[0.07 - 0.18]	44.86
HybridOsc.	2	[0.28 - 0.92]	[0.13 - 0.29]	16.70	[0.03 - 0.31]	[0.01 - 0.10]	45.82
SmoothOsc.	2	[0.37 - 1.09]	[0.13- 0.23]	52.22	[0.04 - 0.42]	[0.02 - 0.18]	136.72
Mountain Car	2	[0.004 - 0.24]	[0.08 - 0.22]	138.90	[0.0002 - 0.005]	[0.03 - 0.12]	266.76
Quadrotor	6	[0.014 -1.09]	[0.10 - 0.67]	284.96	[0.004 - 0.04]	[0.02 - 0.13]	668.78

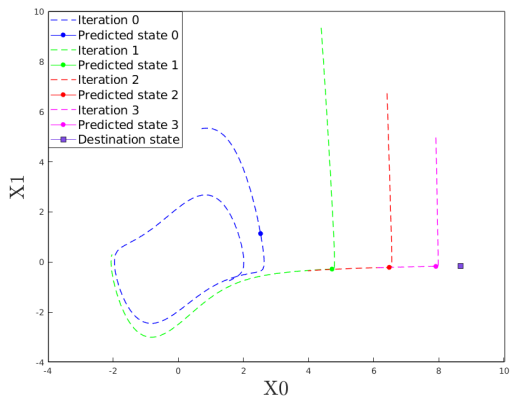
Table 5.3: The evaluation results of $\mathcal{RT}$ after 1 and 5 iterations.

For evaluations, we run $\mathcal{RT}$ 500 times by selecting random anchor trajectory, time step, and target state for each run. In every run, we compute the relative d_r and absolute distance d_a between the target and the state of the trajectory generated by $\mathcal{RT}$ after one or five iterations of the main loop. Finally, we report in Table 5.3 the range of values obtained for d_a and d_r over those 500 runs. The demonstrations on a couple of benchmarks are shown in Figure 5.1. Iteration 0 refers to initial anchor ξ_A. Subsequent 3 anchors/simulations are labeled as Iteration 1, 2 and 3 respectively. As shown, the course gets closer to the target with each iteration. In this way, our algorithm can also be used to generate multiple trajectories that reach within a certain neighborhood of a given target (Figure 5.2) or a set of target states (Figure 5.3).

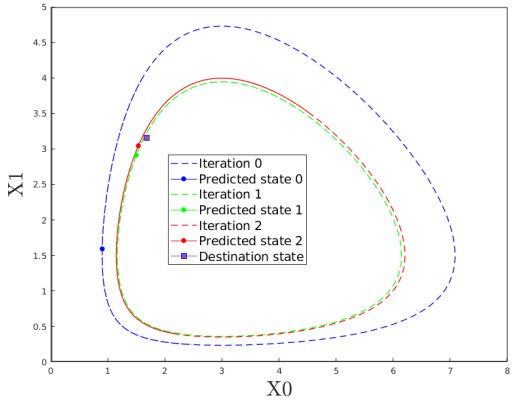
Discussion: It can be observed from Table 5.3 that our technique is capable of achieving below 10% relative distance in almost all cases after 5 iterations. That is, the trajectory generated by $\mathcal{RT}$ algorithm after 5 iterations is around 10% away from the target than the initial trajectory. This was the case even for chaotic systems, hybrid systems, and for control systems with neural network components. While training the neural network might be time taking process, the average time for generating new trajectories that approach the target is very fast (less than a second for all cases). The high relative distance in some cases might be due to high dimensionality or large distance to the target which may be reduced further with more iterations.

We now discuss and demonstrate a few variations of our algorithm.

1. *Uncertainty in time:* For many practical purposes, it may not be possible to know the exact time instance at which the target is reachable from given initial configuration. In such cases, one can provide a time interval, and invoke the algorithm to iteratively find a time instance where the

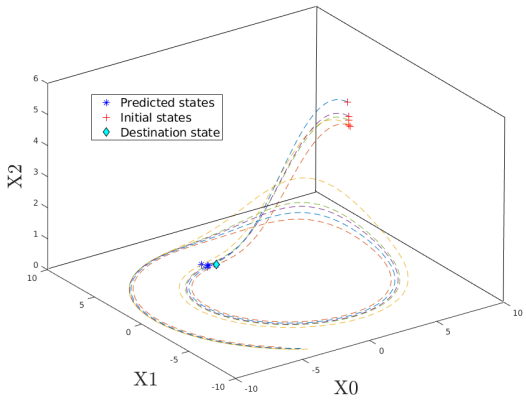


(a) Coupled Vanderpol

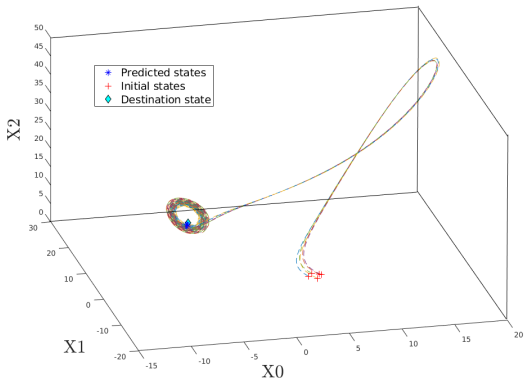


(b) Lotka

Figure 5.1: Basic demonstrations of $\mathcal{RT}$ routine.

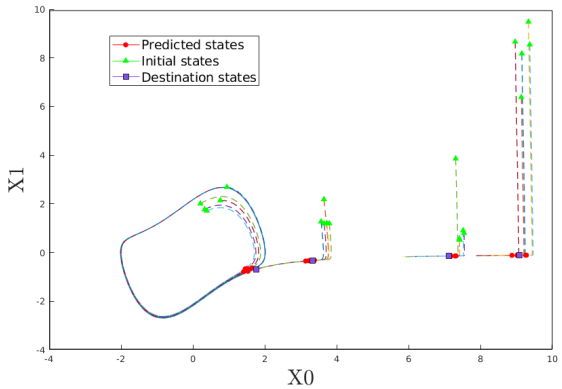


(a) Roesseler

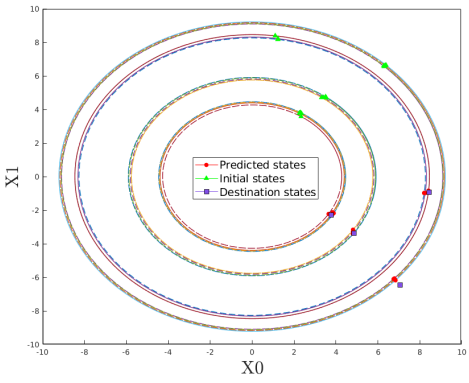


(b) Lorentz

Figure 5.2: Generating multiple executions arriving in a neighborhood of a given target



(a) Vanderpol



(b) Hybrid Linear Oscillator

Figure 5.3: Generating executions arriving in the respective neighborhoods of multiple destinations

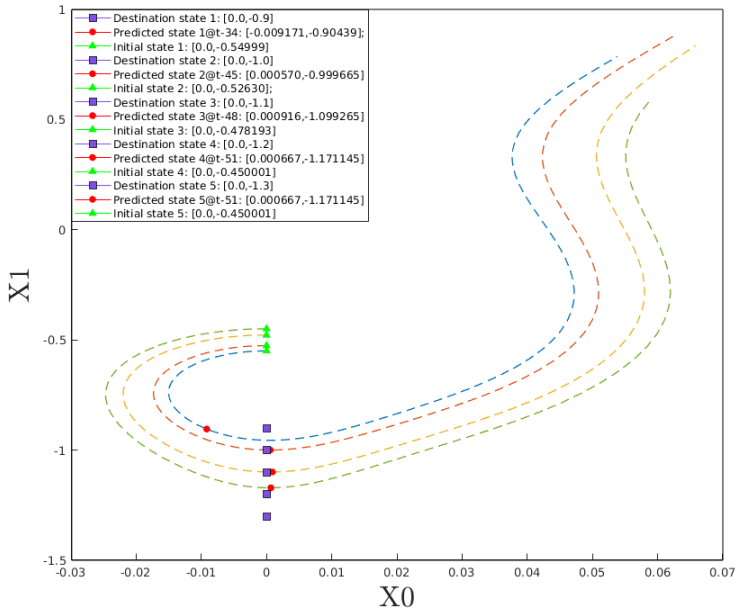
simulation gets reasonably close to the target. Consider the designer is interested in finding the maximum distance (or, height) the car can go to on the left hill in *Mountain Car*. By providing an ordered list of target states and a time interval, she can obtain the maximum distance as well the time instance at which it achieves the maxima (Figure 5.4a). If there is no state in the given initial set from which the car can go to a particular target, the approach, as a side effect, can also provide a suitable initial candidate that takes the car as close as possible to that target. In *Quadrotor*, one can find an initial configuration from which the system can go to a particular location during a given time interval (Figure 5.4b). Note that the *Quadrotor* benchmark has 5 discrete modes and each linear segment in the figure corresponds to the system evolution in a particular mode.

2. *Generalization:* Based on our *Mountain Car* experiment, we observed that, for the given initial set, the maximum distance the car can achieve on the left hill is approx. 1.17. However, even after expanding the initial set from originally [-0.55, -0.45][0.0, 0.0] to [-0.60, -0.40][0.0, 0.0], our approach finds the maximum achievable distance (1.3019) such that the car can still reach on the top of the right hill (shown in Fig. 5.5). This shows that our neural network is able to generalize the inverse sensitivity over trajectories that go beyond the test cases considered during the training process.

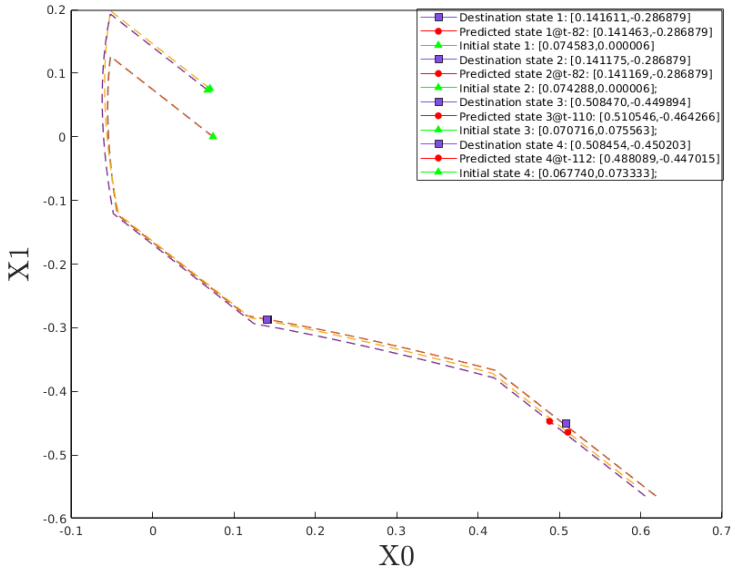
3. *Evaluating MRE for Random Targets:* So far we have evaluated our technique with respect to the target states that are reachable. We also perform the evaluation of *ReachTarget* by generating a random trajectory from the domain and change its course at a provided time interval [25, 70] by a randomly generated vector. We generate a large number of vectors in the unit sphere, scale them on the scale 1-10, and compute the absolute as well as relative error after first iteration at time instances in the given interval. The results for *Roesseler* system are demonstrated in Figure 5.6. The horizontal axis denotes the vector norm. Each colored-plot corresponds to a particular time instance. The absolute error increases linearly with vector size, whereas the relative error appears to be converging. Similar behavior is observed for other systems as well.

5.3.3 Falsification of Safety Specification

One of the widely used methods for performing state space exploration are falsification methods (Sankaranarayanan & Fainekos 2012, Nghiem et al. 2010). Here, the specification is provided in some



(a) Mountain Car



(b) Quadrotor

Figure 5.4: Illustration on how to use $\mathcal{RT}$ routine for time given as an interval.

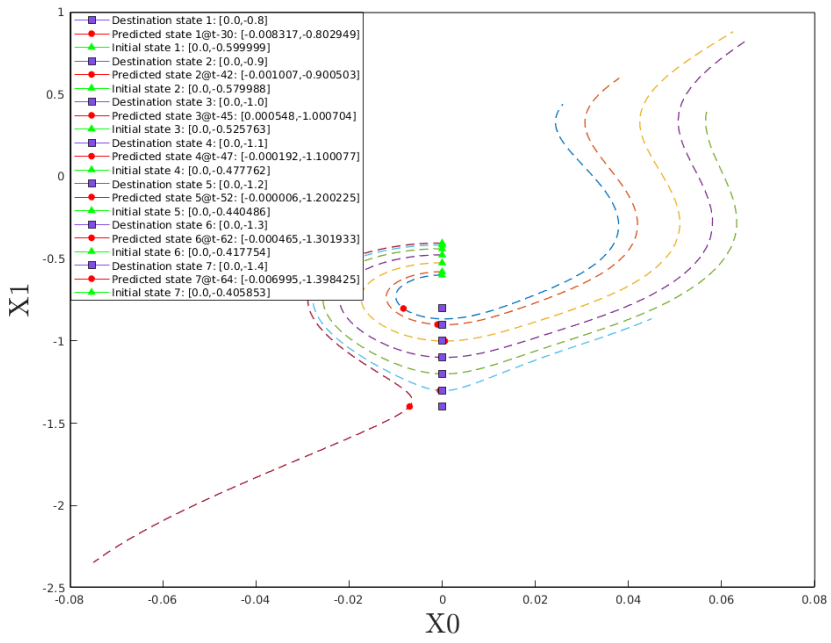
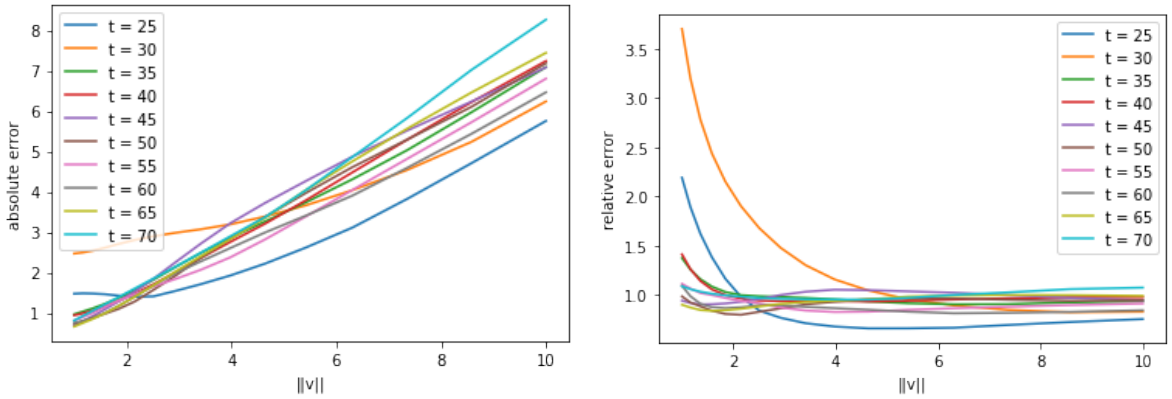


Figure 5.5: Generalizability. This plot for MC benchmark demonstrates that $\mathcal{RT}$ is capable of generalizable to the behaviors outside of the test suite.



(a) Absolute error

(b) Relative error

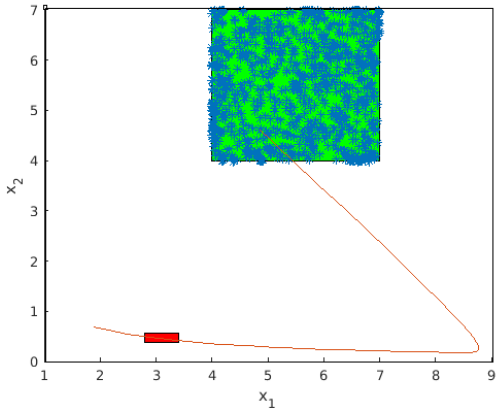
Figure 5.6: $\mathcal{RT}$ demonstrations for random destinations. Figures depict that the approximation error changes in proportion to the magnitude of the inverse sensitivity.

temporal logic such as Signal or Metric Temporal Logic (Maler, Nickovic & Pnueli 2008, Koymans 1990). The falsifier then generates a set of test executions and computes the *robustness* of trajectory with respect to the specification. It then invokes heuristics such as stochastic global optimization for discovering a trajectory that violates the specification.

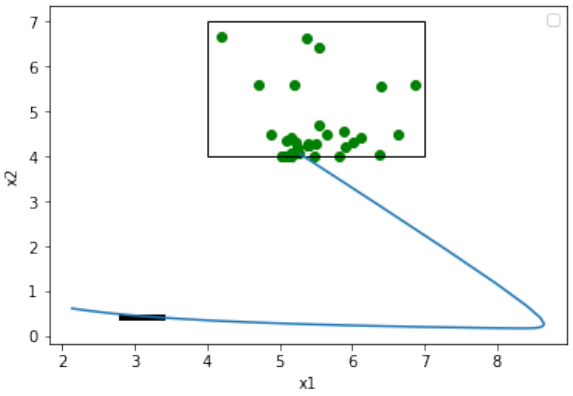
Given an unsafe set U, we provide a simple algorithm to falsify safety specifications. We generate a fixed number (m) of random states in the unsafe set U. Then, using the *ReachTarget* sub-routine, generate trajectories that reach a vicinity of the randomly generated states in U. We terminate the procedure when we discover an execution that enters the unsafe set U. We compare the number of trajectories generated by STL with the number of trajectories generated by *ReachTarget* in Figure 5.7. The box in each of the figures denotes the initial set and the red box represents the unsafe set. Each of the points in the initial set represents a sample trajectory generated by the falsification engine. When a falsification tool fails to find a counterexample, NeuralExplorer may help (using much less number of samples) the system designer by providing geometric insight into the reason why the property is not satisfied.

Falsification using approximation of inverse sensitivity enjoys a few advantages over other falsification methods. First, since our approach approximates the inverse sensitivity, and we use the *ReachTarget* sub-routine; if the approximation is accurate to a certain degree, each subsequent trajectory generated in *ReachTarget* would make progress towards the destination. Second, if the safety specification is changed slightly the robustness of the trajectories with respect to new specification and the internal representation for the stochastic optimization solver has to be completely recomputed. However, since our trajectory generation does not rely on computing the robustness for all the previously generated samples, our algorithm is effective even when the safety specification is modified. The third and crucial advantage of our approach lies when the falsification tool does not yield a counterexample. In those cases, the typical falsification tools cannot provide any geometric insight into the reason why the property is not satisfied. However, using an approximation of inverse sensitivity, the system designer can envision the required perturbation of the reachable set in order to move the trajectory in a specific direction. This geometric insight would be helpful in understanding why a specific trajectory does not go into the unsafe set.

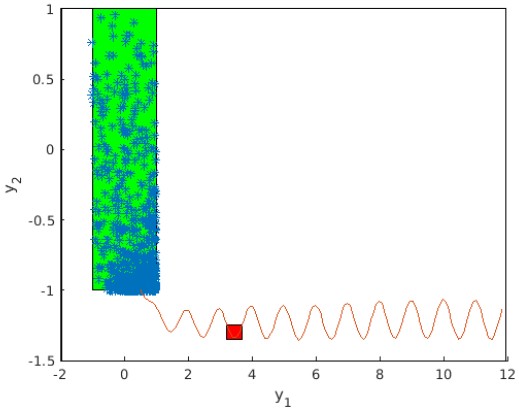
Considering these advantages, the results demonstrated in Figures 5.7a, 5.7b, 5.7c and 5.7d should not be surprising. We also would like to mention that these advantages come at the computational price of training the neural networks to approximating the inverse sensitivity. In addition to the examples shown above, we have included Figures 5.7e and 5.7f where S-TaLiRo terminates with a falsification



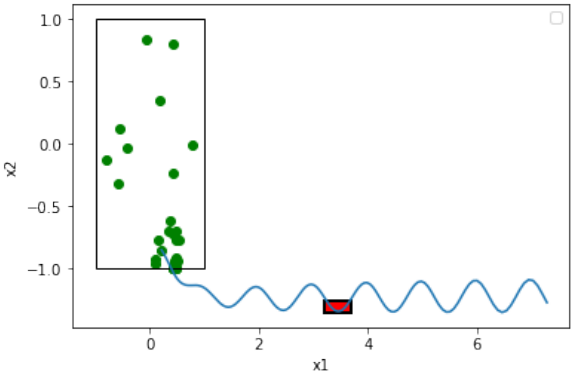
(a) S-TaLiRo



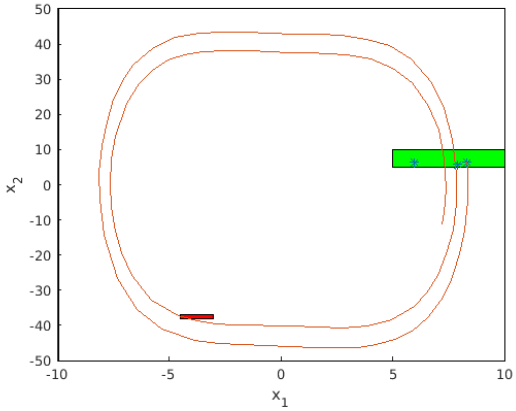
(b) NeuralExplorer



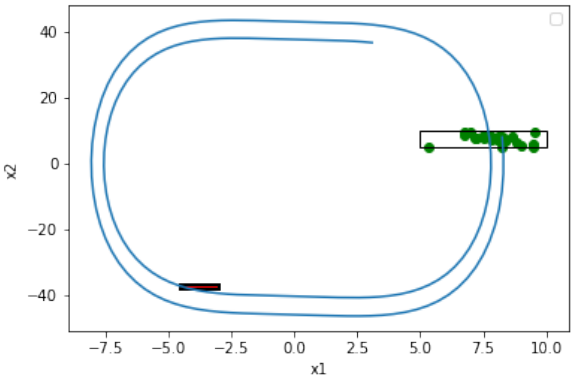
(c) S-TaLiRo



(d) NeuralExplorer



(e) S-TaLiRo: Buckling



(f) NeuralExplorer: Buckling

Figure 5.7: Falsification demonstrations in NeuralExplorer and S-TaLiRo for Brussellator, SA and Buckling benchmarks.

trajectory faster than our approach. The reasons for such cases and methods to improve falsification using NeuralExplorer are a topic of future work.

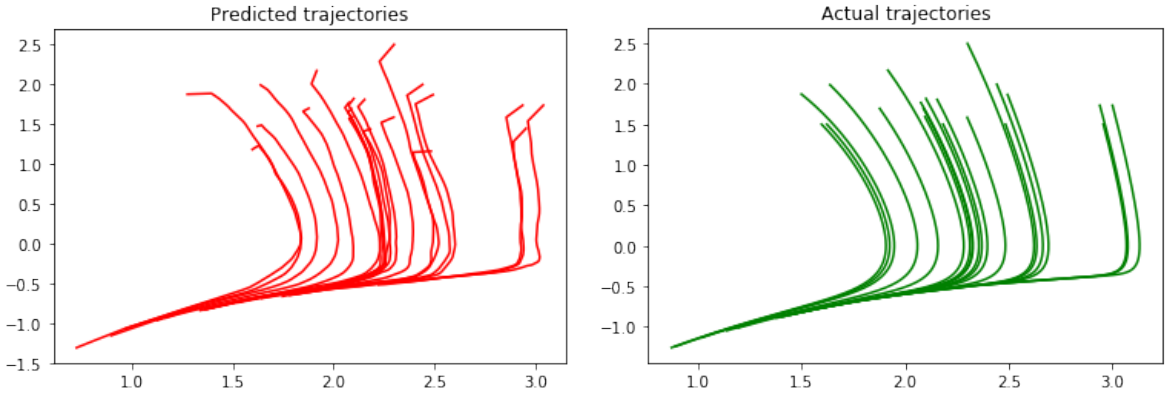
5.4 Generating trajectories for Reachability

Another commonly used technique for performing state space exploration is generation of trajectories from a set of random points generated using an apriori distribution. Based on the proximity of these trajectories to the *unsafe* set, this probability distribution can further be refined to obtain trajectories that move closer to the unsafe set. However, one of the computational bottlenecks for this is the generation of trajectories. Since the numerical ODE solvers are sequential in nature, the refinement procedure for probability distribution is hard to accelerate.

For this purpose, one can use the neural network approximation N_Φ of sensitivity to *predict* many trajectories in an embarrassingly parallel way. Here, a specific set of initial states for the trajectories are generated using a pre-determined distribution. Only a few of the corresponding trajectories for the initial states are generated using numerical ODE solvers. These are called as *anchor trajectories*. The remainder of trajectories are not generated, but rather predicted using the neural network approximation of sensitivity and anchor trajectories as $\xi(x_i, t) + N_\Phi(x_i, x_j - x_i, t)$. Additionally, the designer has the freedom to choose only a subset of the initial states for only a specific time interval for prediction and refine the probability distribution for generating new states. This would also allow us to specifically focus on a time interval or a trajectory without generating the prefix of it. Figures 5.8a and 5.8b show the projected and actual trajectories, respectively, and for a particular cluster of points.

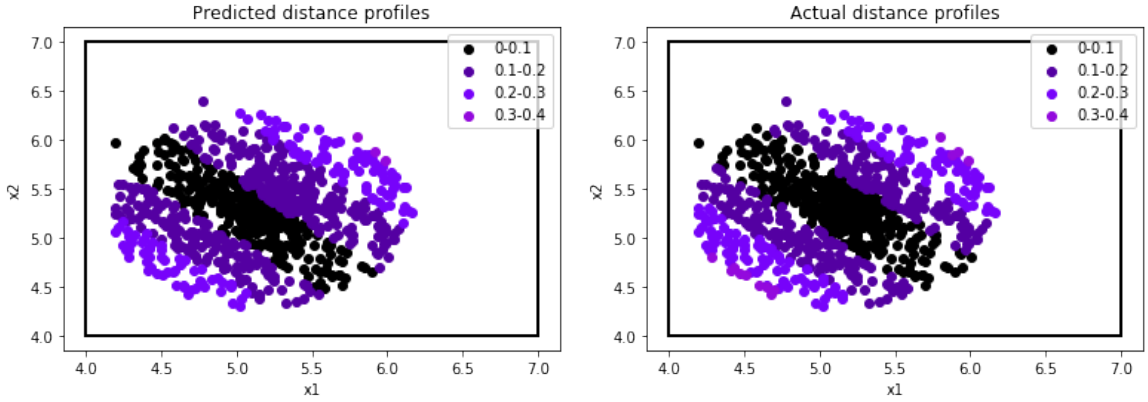
5.5 Discussion on Density based profiling

Similar to the inverse sensitivity based falsification, one can use the density based search space method for generating trajectories that reach a destination and violate a safety specification. The search procedure would work as follows. First, an anchor trajectory is generated and time intervals of its trajectory that are closer to the unsafe set are identified. Then a set of new initial states are generated according to an apriori decided distribution. Instead of generating the trajectory from these initial states, the *predicted trajectories* using the anchor trajectory and neural network approximation of sensitivity is generated specifically for the time intervals of interest. Then, the initial state with the predicted trajectory



(a) (b)

Figure 5.8: Predicting trajectories using sensitivity approximation

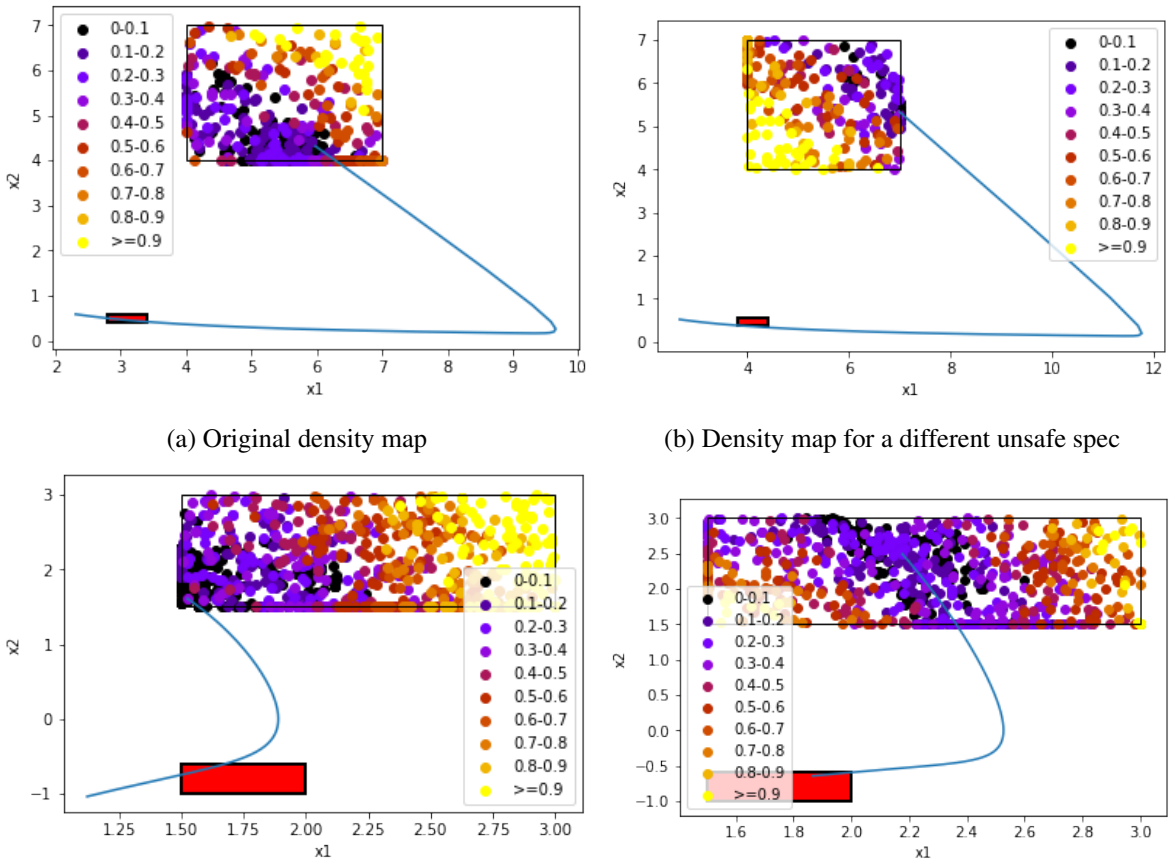


(a) Brussellator (b) Brussellator

Figure 5.9: Density based profiling using sensitivity approximation for falsification. The initial states explored in this iterative process are classified based on the distance between their respective trajectories and the unsafe state.

that is closest to the unsafe set is (*greedily*) chosen and a new anchor trajectory from the selected initial state is generated. This process of generating anchor trajectory, new distribution of initial states, and moving closer to the unsafe set is continued until you reach within the threshold that is generated by the user. Demonstration of this procedure on *Brussellator* system is shown in Figure 5.9.

Notice that this approach gives an underlying intuition about the geometric behavior of neighboring trajectories. A similar method for density based estimation using inverse sensitivity approximation can also be devised. Instead of sampling the initial set, the density based method for inverse sensitivity generates random states with the given unsafe set and then generates a density map. The routine, starting from a reference trajectory, attempts to iteratively find a falsifying trajectory for a given unsafe set.



(a) Original density map

(b) Density map for a different unsafe spec

(c) Original Density map

(d) Density map for a different time instance

Figure 5.10: Density based profiling of the initial set using inverse sensitivity approximation. Notice the difference in the distance profiles (color densities) as we select a difference unsafe spec in *Brusselator* (Figures 5.10a and 5.10b) or change the time instance in *Vanderpol* (Figures 5.10c and 5.10d), thus providing useful insights to the designer during falsification.

The initial states explored in this process are colored according to the distance between their respective trajectories and the unsafe states. These color densities help in identifying regions in the initial set potentially useful for falsification. An example of such a density map generated is given in Fig. 5.10.

5.6 Chapter Summary

This chapter has presented NeuralExplorer framework for state space exploration of closed loop control systems using neural network. The framework learns two key properties of a dynamical system called sensitivity and inverse sensitivity in the form of neural network approximations. It has discussed how such sensitivity functions approximations can be utilized in performing falsification, predicting system trajectories, profiling initial set, and generating some corner case executions.

It has shown that such a state space exploration technique can give a geometric insight into the behavior of the system and provide more intuitive information to the user, unlike earlier black box methods. The evaluation results have demonstrated that our method can not only be applied to standard nonlinear dynamical systems but also for control systems with neural network as feedback functions.

CHAPTER 6: NEXG: PROVABLE AND GUIDED STATE SPACE EXPLORATION OF NEURAL NETWORK CONTROL SYSTEMS USING LOCAL SENSITIVITY APPROXIMATION

The previous chapter presented the state space exploration framework using sensitivity approximation. While achieving the desired goal of systematic exploration based on learning inherent system characteristics, the framework suffers from high training time, sub-optimal coverage and lack of theoretical analysis on its convergence.

This chapter [1] introduces NExG framework which involves approximating the local sensitivity (i.e., for small perturbation) of the trajectories of the closed loop dynamics. Additionally, the theoretical framework is presented which establishes that the new method will produce a sequence of trajectories that converge to the target state at a geometric rate. The thorough evaluation on various systems exhibits that the state space exploration based on local sensitivity approximation outperforms NeuralExplorer and achieve significant improvement in both the quality (coverage) and performance (convergence rate).

We begin with a sub-routine to show how to use an inverse sensitivity approximator $N_{\Phi^{-1}}(x_t, v, t)$ for small values of $\|v\|$ in order to perform guided state space exploration given a destination state. We then theoretically analyze the convergence of the presented technique followed by the suggestions on guiding better approximators. Towards the end, we adapt our approach to falsify an MTL safety specification.

6.1 Reaching a destination at specified time

As discussed earlier, in the course of state space exploration, the designer might want to explore the system behavior that reaches a given destination or approaches the boundary condition for safe operation. Given a domain of operation, and a sample trajectory ξ, the control system designer desires to generate

[1] The content of this chapter is previously appeared in preliminary form in the following preprint:

Goyal, Manish, Miheer Dewaskar, and Parasara Sridhar Duggirala. 2022. **NExG:** Provable and Guided State Space Exploration of Neural Network Control Systems using Local Sensitivity Approximation. In *arXiv:2207.03884*.

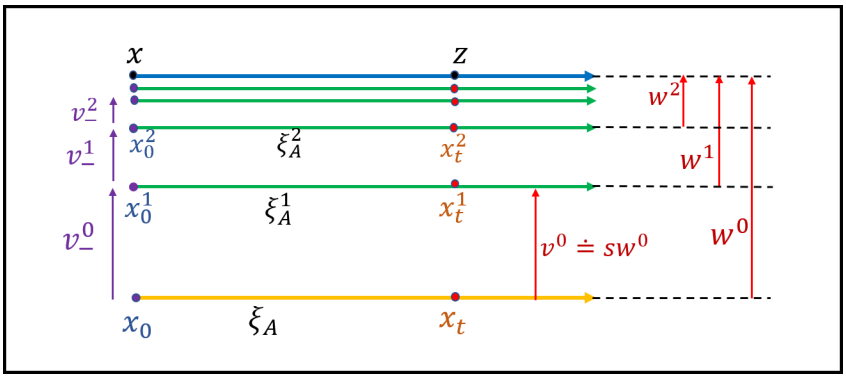


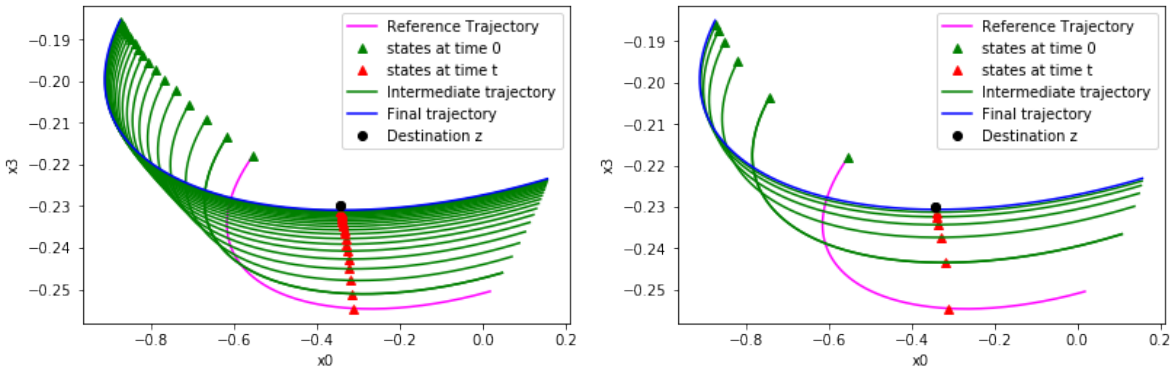
Figure 6.1: Toy execution of Algorithm 9 for a system consisting of a constant horizontal vector field in $\mathbb{R}^2$.

a trajectory that reaches a destination state z (with an error threshold of δ) at time t. Using previous notation, our goal is to find a state x such that the state $\xi(x,t)$ lies in the δ-neighborhood of z.

A toy illustration of our the state space exploration technique with oracle access to the exact inverse sensitivity function is shown in Figure 6.1. Given an initial point x_0, a destination z, and time t, we successively move the initial point in small steps in the direction specified by Φ^{-1}, so that the trajectory starting from the new initial point at time t moves closer to that target z with each step. In practice, since the exact inverse sensitivity function is unknown, we use a neural-network based approximation instead.

Formally, given an *anchor* trajectory starting from initial state $x_0 \in \theta$ (typically chosen at random), we first compute the vector $w^0 \doteq z - x_t^0$ where $x_t^0 \doteq \xi(x_0, t)$. Next, we estimate the inverse sensitivity $\hat{v}_-^0 \doteq N_{\Phi^{-1}}(x_t^0, sw^0, t)$ required at x_0 to move towards z, and then move x_0 by $\hat{v}_-^0$. Here, the input $s \in (0,1)$, called as the *scaling factor*, controls the magnitude of movement at each step. This process is again repeated: move the new initial state $x_0^1 \doteq x_0 + \hat{v}_-^0$ by the vector $\hat{v}_-^1 = N_{\Phi^{-1}}(x_t^1, sw^1, t)$, where $w^1 \doteq z - x_t^1$ and $x_t^1 \doteq \xi(x_0^1, t)$ is the point reached at time t by a new simulated trajectory for the system starting from initial state x_0^1. This process is repeated until x_t^k reaches a pre-specified neighborhood of z.

Since $N_{\Phi^{-1}}$ is only an approximation of Φ^{-1}, the repeated application of the former will typically compound the approximation error. Hence periodically simulating system trajectories starting from intermediate initial states – a step that we term *course correction* – is important to keep the exploration on track. Course correction steps not only confirm that the estimates at time t of the trajectory are indeed close to the z, but they also allow our procedure to make suitable adjustments if that is not indeed the case.



(a) Correction after every step. The number of corrections performed (k) is 23.

(b) Correction after every 4 steps. The number of corrections performed (k) is 7.

Figure 6.2: Periodically correcting the course of exploration (i.e., simulating a new trajectory to aid the search) at different periods.

Since system simulation is expensive, our framework allows for course correction to be performed more or less frequently as desired. The parameter p is designated as *correction period* because the new anchor trajectory attempts to correct the course once for every p invocations of $N_{\Phi^{-1}}(\cdot)$. Observe the effect of performing course corrections at every step (i.e. $p = 1$) and every 4 steps (i.e. $p = 4$) in Figures 6.2a and 6.2b respectively. Algorithm 9, which we call Reach Destination (abbreviated as $\mathcal{RD}$), provides further details of the our procedure. After termination, algorithm $\mathcal{RD}$ returns a 4-tuple consisting of: the number of course corrections k, the trace ξ_A^k of the last anchor trajectory, the absolute distance d_a^k between the target z and ξ_A^k(t), and the relative distance d_r.

Notice that course corrections count is same as the number of trajectories (simulations) generated. If we were to consider physically simulating the plant (which can be expensive) as a part of the operational cost, it would make sense to minimize the number of trajectories we simulate. Limiting the number of simulated trajectories also makes the exploration algorithm more user-friendly by saving their time. Thus, we choose the number of course corrections as the primary metric for performance evaluation.

6.2 Theoretical analysis of the convergence of *ReachDestination*

We now discuss the convergence of Algorithm 9. As seen in Figure 6.1, the distance between x_t^i and the target z contracts by a factor of $0 < 1 - sp < 1$ in each iteration if the exact inverse sensitivity is used. That is,

input : simulator: ξ, time instance: $t \leq T$, reference trajectory: ξ_A, destination: $z \in \mathbb{D}$, course corrections bound: $\mathcal{K}$, function $N_{\Phi^{-1}}$ that approximates Φ^{-1}, initial set: θ, correction period: p, scaling factor: s, and threshold: δ.

output : course corrections: k, final trace: $\xi(x_0^k, \cdot)$, final distance: d_a^k, final relative distance: d_r

$x_0^0, x_t^0 \leftarrow \xi_A(0), \xi_A(t)$; // states at time 0 and t

$w^0 \leftarrow z - x_t^0$; // initial vector difference with z

$d_{init} \leftarrow d_a^0 \leftarrow \|w^0\|$; // initial distance

$k \leftarrow 0$;

while $(d_a^k > \delta)$ & $(k < \mathcal{K})$ **do**

  $v^k \leftarrow s \times w^k$;

  for $1 \leq j \leq p$ **do**

    $\hat{v}_-^k \leftarrow N_{\Phi^{-1}}(x_t^k, v^k, t)$; // predict v_-^k

    $x_0^k \leftarrow \hat{x}_0^{k,\theta} \leftarrow proj_\theta(x_0^k + \hat{v}_-^k)$; // perturb x_0^k

    $x_t^k \leftarrow x_t^k + v^k$; // progress x_t^k

  end for

  $x_0^{k+1} \leftarrow x_0^k$;

  $\xi_A^{k+1} \leftarrow \xi(x_0^{k+1}, \cdot)$; // new anchor

  $x_t^{k+1} \leftarrow \xi_A^{k+1}(t)$; // course correction

  $w^{k+1} \leftarrow z - x_t^{k+1}$; // new vector difference

  $d_a^{k+1} \leftarrow \|w^{k+1}\|$; // update distance to z

  $k \leftarrow k + 1$; // increment corrections by 1

end while

$d_r \leftarrow d_a^k / d_{init}$; // update relative distance

return (k, ξ_A^k, d_a^k, d_r);

Algorithm 9: *ReachDestination* ($\mathcal{RD}$) algorithm

$$\|x_t^k - z\| \leq (1 - sp)^k \|x_t^0 - z\| \tag{6.1}$$

Hence, the generated trajectory will reach the desired destination within an error of δ after k^* iterations where,

$$k^* = \frac{\log(\|x_t^0 - z\|/\delta)}{-\log(1 - sp)}. \tag{6.2}$$

However, in $\mathcal{RD}$ algorithm, instead of exact inverse sensitivity, we use its approximation. In this section, we show that it is possible to achieve a similar geometric rate of convergence even with an approximation. However the convergence of $\mathcal{RD}$ can fail badly in cases when the system is chaotic or the approximation error is large. To this end we now make assumptions on the regularity of the system and the magnitude of the approximation error that will ensure convergence.

Assumption 1 *Suppose there are functions $\eta_1, \eta_2 : [0,T] \to [0,\infty)$ so that*

$$\eta_1(t)\|x - x'\| \leq \|\xi(x,t) - \xi(x',t)\| \leq \eta_2(t)\|x - x'\| \tag{6.3}$$

for each $x, x' \in \mathbb{D}$ and $t \in [0,T]$.

The functions η_1 and η_2, sometimes called as witnesses to the discrepancy function (Duggirala, Mitra & Viswanathan 2013), provide worst-case bounds on how much the distance between trajectories expand or contract starting from different initial states. These functions (and their ratios) can be considered as a measure of the regularity of the system. Although in practice it may be hard to obtain the values η_1 and η_2 for the system at hand, exponential lower bound for η_1 and a similar upper bound for η_2 can be obtained using Grönwall's inequality under a Lipschitz continuity assumption on the vector field. As shown in the following lemma, Assumption 1 also ensures that $\Phi^{-1}(x,v,t)$ is a Lipschitz function of its inputs x and v. This is important as such functions can be approximated by Neural networks of bounded depth (see e.g. (Gühring, Raslan & Kutyniok 2020, Theorem 4.5)).

Lemma 3 *If Assumption 1 is satisfied then for any $t \in [0,T]$*

$$\|\Phi^{-1}(z',v',t) - \Phi^{-1}(z,v,t)\| \leq \left(2\|z - z'\| + \|v - v'\|\right)/\eta_1(t)$$

Proof 5 *By taking $(x,x') = (\xi^{-1}(y,t), \xi^{-1}(y',t))$ in Assumption 1, note that $\|\xi^{-1}(y,t) - \xi^{-1}(y',t)\| \leq \|y - y'\|/\eta_1(t)$ for any $y, y' \in \mathbb{D}$. The Lemma now follows by suitably applying triangle inequality and the definiton of Φ^{-1}.*

In general, we will use the following model to measure the approximation error of $N_{\Phi^{-1}}$. The separate roles played by the relative error ε_{rel} and the absolute error ε_{abs} will become more clear in the context of Theorem 5.

Definition 23 *$N_{\Phi^{-1}}$ is called an $(\varepsilon_{rel}, \varepsilon_{abs})$-approximator of Φ^{-1} upto radius r and time T if*

$$\|N_{\Phi^{-1}}(x_t,v,t) - \Phi^{-1}(x_t,v,t)\| \leq \varepsilon_{rel}\|\Phi^{-1}(x_t,v,t)\| + \varepsilon_{abs}$$

for any $x_t \in \mathbb{D}$, $t \in [0,T]$ and $v \in \mathbb{R}^n$, with $\|v\| \leq r$.

We are now ready state Theorem 5 which bounds the distance between z and iterate x_t^k in the kth iterations of the outer loop in $\mathcal{RD}$ when the system satisfies Assumption 1 and $N_{\Phi^{-1}}$ satisfies Definition 23 with sufficiently small error terms $(\varepsilon_{\text{rel}}, \varepsilon_{\text{abs}})$. To further interpret Theorem 5, note that:

1. When the additive error $\varepsilon_{\text{abs}} \approx 0$ is negligible, the relative error $\varepsilon_{\text{rel}} < \eta_1(t)\eta_2(t)^{-1}$ is suitably small depending on the system regularity, and s takes a value close to its upper bound, then Equation (6.3) holds for any $k \in \mathbb{N}$ with $r_{\boldsymbol{\varepsilon}}(t)/s \approx 0$. Hence, in this case, a geometric convergence similar to that described for the toy example from above continues to hold with a slightly slower convergence rate (i.e. $-\log(1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))$ instead of $-\log(1 - sp)$).

2. On the other hand, when ε_{abs} is small (so that $r_{\boldsymbol{\varepsilon}}(t) \leq r$) but non-negligible, the last term in Equation 6.3 cannot be ignored. In this case, if the assumptions of Theorem 9 are satisfied, one obtains the guarantee that $\lim_{k\to\infty} d_a(k) \leq r_{\boldsymbol{\varepsilon}}(t)/s$. Hence if $r_{\boldsymbol{\varepsilon}}(t)/s < \delta$, the termination condition $x_t^k \in B_\delta(z)$ will eventually be satisfied. More precisely, $\mathcal{RD}$ will find an x_t^k in the δ-neighborhood of z after at most $k \leq k^* = \lceil \log(\frac{\delta - r_{\boldsymbol{\varepsilon}}(t)/s}{d_{init}}) / \log(1 - sp\gamma_{\boldsymbol{\varepsilon}}(t)) \rceil$ iterations.

Theorem 5 (Convergence of $\mathcal{RD}$) *Fix the domain $\mathbb{D} = \mathbb{R}^d$ and a time $T > 0$. Suppose*

1. *The system satisfies Assumption 1, and*
2. *$N_{\Phi^{-1}}$ is an $(\varepsilon_{rel}, \varepsilon_{abs})$-appoximation of Φ^{-1} for radius r and time T.*
3. *$\varepsilon_{rel}, \varepsilon_{abs} \geq 0$ values small enough so that for each $t \in [0, T]$, $\gamma_{\boldsymbol{\varepsilon}}(t) \doteq 1 - \varepsilon_{rel}\eta_2(t)\eta_1(t)^{-1} > 0$ and $r_{\boldsymbol{\varepsilon}}(t) \doteq \varepsilon_{abs}\eta_2(t)/\gamma_{\boldsymbol{\varepsilon}}(t) \leq r$.*

Suppose the inputs $\theta = \mathbb{D}$, $t \in [0, T]$, and the destination $z \in \mathbb{D}$ to Algorithm 9 are given. For sufficiently small s, the distance between the trajectory generated by Algorithm 9 after k iterations of the outer loop is given as

$$\|x_t^k - z\| \leq (1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))^k \|x_t^0 - z\| + \frac{r_{\boldsymbol{\varepsilon}}(t)}{s} \tag{6.4}$$

for any $k \in \mathbb{N}$.

The proof of Theorem 5, particularly for the case of $\varepsilon_{\text{abs}} = 0$, can be seen to be a suitable contraction argument. Formal details are given below.

Proof 6 (Proof of Theorem 5) *In this proof, for mathematical clarity, we slightly the change notation for the variables used in Algorithm 9. For each $k \geq 0$, let $x_0(k), x_t(k), w(k)$ and $d_a(k)$ denote the values of the variables x_0^k, x_t^k, w^k and d_a^k after k executions of the outer loop in Algorithm 9. Hence the equalities $w(k) = z - x_t(k)$, $d(k) = \|w(k)\|$, and $x_t(k) = \xi(x_0(k), t)$ are satisfied for any $k \geq 0$.*

Since $\theta = \mathbb{D}$, unwinding the inner loop in Algroithm 9, note

$$x_0(k+1) = x_0(k) + \sum_{l=1}^{p} N_{\Phi^{-1}}(x_t(k) + (l-1)sw(k), sw(k), t). \tag{6.5}$$

Let $\tilde{y}(k) \doteq x_t(k+1) - x_t(k)$ denote the increment between the k and $(k+1)$th iteration, and using the fact that $x_t(k) = \xi(x_0(k), t)$ note

$$\tilde{y}(k) = \xi\left(x_0(k+1), t\right) - \xi(x_0(k), t). \tag{6.6}$$

The quantity $\tilde{y}(k)$ approximates the true target increment given by

$$y(k) \doteq \xi\left(x_0(k) + \Phi^{-1}(x_t(k), spw(k), t), t\right) - \xi(x_0(k), t). \tag{6.7}$$

Further, note using Definition (2.6) *of Φ^{-1} that*

$$\begin{aligned} y(k) &= \xi\left(x_0(k) + \xi^{-1}(x_t(k) + spw(k), t) - \xi^{-1}(x_t(k), t), t\right) - \xi\left(x_0(k), t\right) \\ &= \xi\left(x_0(k) + \xi^{-1}(x_t(k) + spw(k), t) - x_0(k), t\right) - x_t(k) \\ &= x_t(k) + spw(k) - x_t(k) = spw(k). \end{aligned} \tag{6.8}$$

Subtracting (6.7) *from* (6.6)*, and using the upper bound from* (6.3)

$$\begin{aligned}
\|\tilde{y}(k) - y(k)\| &= \left\|\xi\left(x_0(k+1), t\right) - \xi\left(x_0(k) + \Phi^{-1}(x_t(k), spw(k), t), t\right)\right\| \\
&\leq \eta_2(t)\|x_0(k+1) - x_0(k) - \Phi^{-1}(x_t(k), spw(k), t)\| \\
&= \eta_2(t)\left\|\sum_{l=1}^{p} N_{\Phi^{-1}}(x_t(k) + (l-1)sw(k), sw(k), t)\right. \\
&\qquad \left. - \sum_{l=1}^{p} \Phi^{-1}(x_t(k) + (l-1)sw(k), sw(k), t)\right\| \\
&\leq p\eta_2(t) \max_{l=1,\ldots,p} \|N_{\Phi^{-1}}(x_t(k) + (l-1)sw(k), sw(k), t) \\
&\qquad -\Phi^{-1}(x_t(k) + (l-1)sw(k), sw(k), t)\| \qquad (6.9)
\end{aligned}$$

where the second equality is obtained by rewriting $\Phi^{-1}(x_t(k), spw(k), t)$ *as a telescoping sum and using* (6.5)*. To bound the terms under the maximum in* (6.9)*, we now use that* $N_{\Phi^{-1}}$ *is an* $(\varepsilon_{rel}, \varepsilon_{abs})$*-approximator of* Φ^{-1}*. Using Definition 23 followed by the lower bound in* (6.3)*, we obtain*

$$\begin{aligned}
\|N_{\Phi^{-1}}(x, v, t) - \Phi^{-1}(x, v, t)\| &\leq \varepsilon_{rel}\|\Phi^{-1}(x, v, t)\| + \varepsilon_{abs} \\
&\leq \varepsilon_{rel}\eta_1(t)^{-1}\|v\| + \varepsilon_{abs}
\end{aligned} \qquad (6.10)$$

as long as $x \in \mathbb{D}$*,* $\|v\| \leq r$ *and* $t \in [0, T]$*. Using* $sd_{init} \leq r$*, we have*

$$\|\tilde{y}(k) - y(k)\| \leq sp\|w(k)\|\varepsilon_{rel}\eta_2(t)\eta_1(t)^{-1} + p\eta_2(t)\varepsilon_{abs} \qquad (6.11)$$

whenever $\|w(k)\| \leq d_{init}$*. With the above estimate, we obtain using* (6.8)*:*

$$\begin{aligned}
w(k+1) - w(k) = -x_t(k+1) + x_t(k) \doteq -\tilde{y}(k) &= -y(k) + y(k) - \tilde{y}(k) \\
&= -spw(k) + y(k) - \tilde{y}(k).
\end{aligned}$$

Recall $d_a(k) \doteq \|w(k)\| = \|z - x_t(k)\|$. *Combining the above display with* (6.11) *shows the following recursive inequality for* $d_a(k)$ *whenever* $d_a(k) \leq d_{init}$*:*

$$\begin{aligned} d_a(k+1) = \|w(k+1)\| &= \|(1-sp)w(k) + y(k) - \tilde{y}(k)\| \\ &\leq (1-sp)\|w(k)\| + \|\tilde{y}(k) - y(k)\| \\ &\leq (1 - sp\{1 - \varepsilon_{rel}\eta_2(t)\eta_1(t)^{-1}\})d_a(k) + p\eta_2(t)\varepsilon_{abs} \\ &= (1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))d_a(k) + p\eta_2(t)\varepsilon_{abs} \end{aligned}$$

where we have used the assumption that $sp \leq 1$ *in first inequality,* (6.11) *in the second inequality, and the notation* $\gamma_{\boldsymbol{\varepsilon}}(t) \doteq 1 - \varepsilon_{rel}\eta_2(t)\eta_1(t)^{-1}$ *in the final equality. From the assumed lower bound on* s*, we have* $p\eta_2(t)\varepsilon_{abs} \leq sp\gamma_{\boldsymbol{\varepsilon}}(t)d_{init}$*, and the hence the condition* $d_a(k) \leq d_{init}$ *continues to holds for any* $k \in \mathbb{N}$ *by induction. Hence recursively applying the inequality in the last display we obtain*

$$\begin{aligned} d_a(k) &\leq (1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))^k d_a(0) + p\eta_2(t)\varepsilon_{abs}\sum_{i=0}^{k-1}(1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))^{k-1} \\ &\leq (1 - sp\gamma_{\boldsymbol{\varepsilon}}(t))^k d_{init} + \frac{r_{\boldsymbol{\varepsilon}}(t)}{s}. \end{aligned} \tag{6.12}$$

where, since $sp\gamma_{\boldsymbol{\varepsilon}}(t) \in [0, 1)$*, we have used the formula for the infinite geometric sum to obtain the last inequality.*

6.2.1 Guidance on designing better approximators

Theorem 5 also provides guidance on how to design good approximators to use with $\mathcal{RD}$. For various approximators which one may consider that satisfy Definition 23, ε_{abs} will typically be non-zero. Therefore, Theorem 5 suggests that approximators with small additive error ε_{abs} will have better reachability guarantees when used within $\mathcal{RD}$. This naturally raises the question of how to design approximators with a small additive error ε_{abs}. One important aspect of this is the evaluation radius $r > 0$. For any given approximator $N_{\Phi^{-1}}$, the additive error $\varepsilon_{\text{abs}} = \varepsilon_{\text{abs}}(r)$ in Definition 23 can be considered as an increasing function of the evaluation radius r. Therefore, one may hope to obtain estimators with better values of $\varepsilon_{\text{abs}}(r)$ by evaluating for small values of the radius r.

In Figure 6.3, we used testing data to empirically estimate the absolute error $\varepsilon_{\text{abs}}(r)$ for Neural-Explorer trained on four benchmarks, evaluated for various values of r. Even as $r \to 0$, the $\varepsilon_{\text{abs}}(r)$

values of NeuralExplorer seem to approach a non-zero value $\delta_0 = \lim_{r\to 0} \varepsilon_{\text{abs}}(r) > 0$. In the light of Theorem 5, this might explain the lack of convergence of NeuralExplorer that we have observed in certain empirical examples. Indeed, as the iterates x_t^k in the NeuralExplorer approach the target z, the error in the approximation of $N_{\Phi^{-1}}$ might possibly be dominating the increment $\Phi^{-1}(x_t, s(z - x_t), t)$ needed to proceed towards the target.

Motivated by the above discussion, in this work, we introduce approximators based on neural networks $\tilde{N}_{\Phi^{-1}}(x_t, v/\|v\|, t)$ that learn for small values of $\|v\|$ only the direction (and not the magnitude) of the vector $\Phi^{-1}(x_t, v, t) \approx \nabla_v \Phi^{-1}(x_t, 0, t)v$. By focusing only on learning the direction, we avoid numerical issues involved in learning small values. Intuitively, if the value $\|\Phi^{-1}(x_t, v, t)\|$ was indeed known, we can use the oracle-estimator

$$N_{\Phi^{-1}}(x_t, v, t) = \tilde{N}_{\Phi^{-1}}(x_t, \frac{v}{\|v\|}, t) \cdot \|\Phi^{-1}(x_t, v, t)\| \tag{6.13}$$

to approximate $\Phi^{-1}(x_t, v, t)$ for small values of $\|v\|$. Figure 6.3 shows the estimate of the $\varepsilon_{\text{abs}}(r)$ versus r plot for the oracle estimator 6.13. However, note that outside a testing scenario like that in Figure 6.3, $N_{\Phi^{-1}}(x_t, v, t)$ cannot be evaluated for the directional approximators. Instead, we directly use $\tilde{N}_{\Phi^{-1}}(x_t, \frac{v}{\|v\|}, t)$ in $\mathcal{RD}$ algorithm by the modifications mentioned in Remark 8.

Remark 8 *As mentioned above, it may be helpful to work with directional-approximator $\tilde{N}(x_t, v/\|v\|, t)$ for $\Phi^{-1}(x_t, v, t)$, which learn only the direction (and not the magnitude) of the vector $\Phi^{-1}(x_t, v, t) \approx \nabla_v \Phi^{-1}(x_t, v, t)v$ for small values of $\|v\|$. One may then work directly with the the directional approximator $\tilde{N}(x_t, v/\|v\|, t)$ in $\mathcal{RD}$ by simply replacing line 8 with the two statements*

(8a) $\hat{v}_-^k \leftarrow \tilde{N}_{\Phi^{-1}}(x_t^k, \frac{v^k}{\|v^k\|}, t)$

(8b) $\hat{v}_-^k \leftarrow (s \times \|v^k\|) \cdot \hat{v}_-^k$,

while keeping the rest of the algorithm unchanged.

The NExG algorithms used in the subsequent sections are based on directional-approximators $\tilde{N}_{\Phi^{-1}}$ and the corresponding modifications to $\mathcal{RD}$ as mentioned in Remark 8.

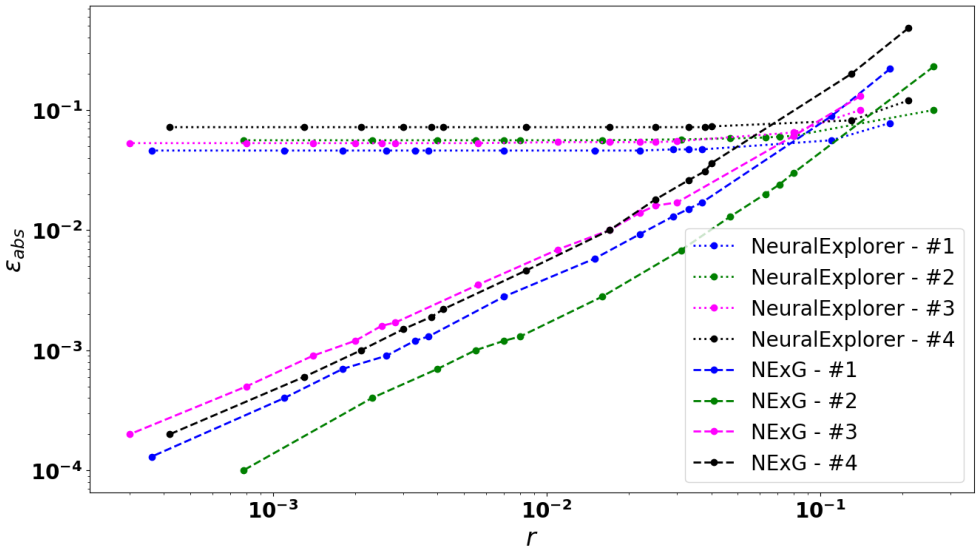


Figure 6.3: Empirical values of the additive error ε_{abs} in Definition 23 (assuming $\varepsilon_{\text{rel}} \approx 0$) for the approximators $N_{\Phi^{-1}}$ learned for NeuralExplorer and NExG as a function of the evaluation radius r. Note that we have the used the oracle-estimator $N_{\Phi^{-1}}$ given by (6.13) to estimate the additive error of NExG, since NExG only learns a directional approximator $\tilde{N}_{\Phi^{-1}}$.

6.3 Evaluation

We choose a standard benchmark suite of control systems with neural feedback functions (Dutta, Chen & Sankaranarayanan 2019, Jankovic, Fontaine & Kokotovic 1996, Johnson, Lopez, Musau, Tran, Botoeva, Leofante, Maleki, Sidrane, Fan & Huang 2020, Lopez, Musau, Tran & Johnson 2019) for evaluation. To be more specific, systems #10-#12 are adopted from (Jankovic, Fontaine & Kokotovic 1996), system #13 is adopted from (Dutta, Chen & Sankaranarayanan 2019) and the rest of the benchmarks are adopted from the ARCH suite (Johnson et al. 2020, Lopez et al. 2019). Considered benchmarks span 6 dimensional systems, controllers with 6-10 hidden layers and 100-300 neurons per layer (c.f. Table 6.1).

6.3.1 Network architecture and Training

For each benchmark, we generate a fixed number (40) of *anchor* trajectories using a given ODE solver. We generate additional 10 trajectories from the states randomly sampled in the small neighborhood ($\|v\| = 0.001$) of the initial states of each anchor trajectory. We chose step size (h) and steps count (T) as shown in Table 6.1, however, a user can pick any suitable values for these parameters and the number

of anchor trajectories. Our preliminary analysis shows that increasing the number of *anchor* trajectories from 40 to 50 slightly improves the MRE, however, this improvement comes at the expense of training time. This trade offs between the amount of data required, training time, distance between neighboring points, and accuracy of the approximation are subjected to future research. Nonetheless, the evaluations presented in subsequent sections underscores the promise of our approach even when the resources are constrained. The data used for training the neural network is collected as previously described. We use 90% of the data for training and 10% for testing.

Table 6.1: Training inverse sensitivity approximator $N_{\Phi^{-1}}$ in NExG. Each neural network feedback controller configuration is given as the *number of hidden layers* and the maximum of *neurons per layer*. *Dims* is the number of system variables and T is simulation time bound.

System				$N_{\Phi^{-1}}$ Training			
No.	Name	Dims	NN controller config	Max steps (T)	Training Time (min)	MSE	MRE %
#1	ARCH-1	2	6/50	200	8.0	0.018	16.0
#2	ARCH-2	2	7/100	300	11.0	0.038	15.0
#3	ARCH-3	2	5/50	350	14.0	0.014	10.2
#4	S. Pend.	2	2/25	250	9.0	0.021	15.0
#5	ARCH-4	3	7/100	250	10.0	0.007	6.0
#6	ARCH-5	3	7/100	250	10.0	0.009	13.0
#7	ARCH-6	3	6/100	250	10.0	0.007	5.6
#8	ARCH-7	3	2/300	250	11.0	0.01	9.5
#9	ARCH-8	4	5/100	250	12.0	0.005	8.0
#10	ARCH-9-I	4	3/100	250	12.0	0.005	4.2
#11	ARCH-9-II	4	3/20	250	11.0	0.005	7.3
#12	ARCH-9-III	4	3/20	250	11.0	0.005	4.5
#13	Unicycle	4	1/500	250	11.0	0.005	5.3
#14	D. Pend.-I	4	2/25	250	11.0	0.006	8.1
#15	D. Pend.-II	4	2/25	200	10.0	0.02	17.0
#16	I. Pend.	4	1/10	200	9.0	0.007	9.0
#17	ACC-3L	6	3/20	200	11.0	0.004	8.3
#18	ACC-5L	6	5/20	200	11.0	0.004	6.7
#19	ACC-7L	6	7/20	200	12.0	0.004	6.7
#20	ACC-10L	6	10/20	200	12.0	0.005	12.0

We use Python Multilayer Perceptron implemented in `Keras 2.3` (Chollet et al. 2015) library with Tensorflow as the backend. Every network has 3 layers with 512 neurons each and an output layer. The input layer's activation function is *Radial Basis Function* (RBF) with *Gaussian basis* (Vidnerová 2019). The other two layers have `ReLU` activation (except System #11 which has its feedback controller trained

with `Sigmoid` activation) and the output layer has *linear* activation function. The optimizer used is stochastic gradient descent. The network is trained using Levenberg-Marquardt backpropagation algorithm optimizing the mean absolute error loss function and the Nguyen-Widrow initialization. The training and evaluation are performed on a system running Ubuntu 18.04 with a 2.20GHz Intel Core i7-8750H CPU with 12 cores and 32 GB RAM. The network $N_{\Phi^{-1}}$ training time, *mean squared error* (MSE) and *mean relative error* (MRE) for learning Φ^{-1} are given in Table 6.1.

Although empirical, our choice for network architecture and evaluation metrics are somewhat motivated by NeuralExplorer (Goyal & Duggirala 2020*b*). But the network training time in (Goyal & Duggirala 2020*b*) and training error are notably high that may render the work unfavorable for many practical applications. After performing experiments on multiple activation functions, we chose a non linear multi variate radial basis function (RBF) with Gaussian basis as the input layer's activation function because its performance during evaluation was found to be consistent across benchmarks.

6.3.2 *ReachDestination* Evaluation

We analyze the performance of Algorithm 9 by picking, at every invocation of the algorithm, a random reference trajectory ξ_A, a time $t \in [0, T]$, and a target state z, reachable at time t in the domain of interest. We choose them randomly to not bias the evaluation of our search procedure to a specific sub-space. The performance metrics used to evaluate various runs are *number of course corrections* (k) and/or *minimum relative distance* (d_r). The threshold δ is fixed as 0.004.

1. **Comparison with NeuralExplorer (Goyal & Duggirala 2020*b*):** The neural network architectures used in this work are the same as those used in NeuralExplorer. For a given $N \in \mathbb{Z}_+$ number of anchor trajectories, NeuralExplorer creates all possible $C(N, 2)$ pairs of these trajectories for training as it attempts to learn the inverse sensitivity function for any $v \in \mathbb{R}^n$ in the domain of interest. Whereas NExG focuses on learning only the direction of the inverse sensitivity. So we only sample a few random points (say, y) in a small neighborhood of the initial state of each anchor trajectory and generate total $y \times N$ pairs. As a consequence, we achieve up to 60% reduction in the training time. Further, the state space exploration algorithm in NeuralExplorer predicts inverse sensitivity directly for w^k and course corrects at every step; whereas, NExG predicts only the direction of the inverse sensitivity vector needed to move in the direction w^k. Hence the NExG

Table 6.2: Performance evaluation w.r.t. NeuralExplorer. The common parameters values are $\delta = 0.004$ and $\mathcal{K} = 30$. We fix $s = 0.5$ and $p = 2$ for NExG. k is the number of simulations generated. d_a^k is the distance between $\xi_A^k(t)$ and the destination z, and $d_r\% = (d_a^k/d_{init}) \times 100$.

System	NeuralExplorer		NExG		System	NeuralExplorer		NExG	
	k	$d_r\%$	k	$d_r\%$		k	$d_r\%$	k	$d_r\%$
#1	21	14	5	1.8	#11	30	6.1	4	0.4
#2	27	9.6	6	1.3	#12	30	13.8	4	0.6
#3	10	6.4	5	2.7	#13	30	11.4	13	2.3
#4	18	5.5	5	1.4	#14	29	7.6	10	1.9
#5	30	13.3	7	2.7	#15	26	12.7	8	3.7
#6	24	11.2	5	3.9	#16	29	22.5	6	2.2
#7	28	12.5	5	2.1	#17	30	8.0	8	0.4
#8	23	5.5	7	1.7	#18	30	6.7	7	0.4
#9	30	12.3	12	3.5	#19	30	5.5	6	0.3
#10	29	4.3	10	1.0	#20	30	5.4	17	1.6

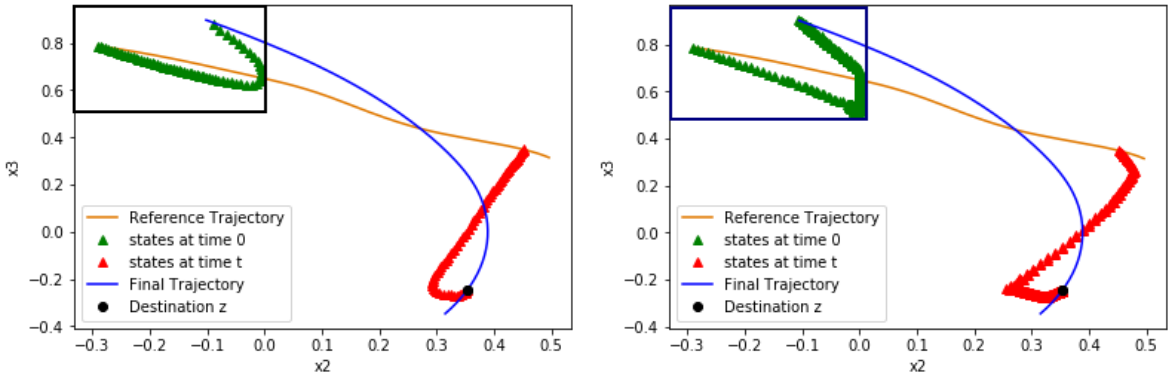
search is guided by additional parameters like the scaling factor s and the correction period p. We report in Table 6.2 the mean values of k and d_r computed over 250 runs of each technique for each system. The evaluation shows that NExG has a relative error of 1-4% (with considerably fewer iterations) as compared to the relative error of 5-15% for NeuralExplorer.

2. **Correction period (p) and scaling factor (s):** We fix the tuple (ξ_A, z, t) in each benchmark, run $\mathcal{RD}$ for $s \in \{0.01, 0.1\}$, $p \in \{2, 5, 10\}$. The evaluation results presented in Table 6.3 are to make some important observations and emphasize that the technique performs consistently across systems. For a fixed tuple (ξ_A, z, t), change in the number of trajectories (k) generated by $\mathcal{RD}$ is roughly inversely proportional to the change in the product $s \cdot p$. For example, k in the first row (i.e., $s = 0.01$) in System #1 shows that the number of trajectories reduces from ~ 500 to ~ 50 (10 fold reduction) when course correction is performed only once for every 10 steps instead of at every steps. This trend is observed in almost all systems for appropriate values of the product $s \cdot p$. It can also be observed that the number of trajectories remains roughly the same for different (s, p) pairs as long as the product $s \cdot p$ is same. For e.g., the value of k for pairs ($s = 0.01$, $p = 10$) and ($s = 0.1$, $p = 1$) is ~ 50 in System #1. These results are consistent with the theoretical bound (6.4) that decreases geometrically in k for a fixed value of $s \cdot p$.

3. **Satisfying initial conditions:** As the algorithm increments the initial state x_0^k in line 9, it may happen that next $\hat{x}_0^{k+1} \doteq (x_0^k + \hat{v}_-^k)$ is not in the initial set θ, thus violating the initial constraint or

Table 6.3: $\mathcal{RD}$ evaluation. d_{init} is the distance between the initial reference trajectory $\xi_A^0(t)$ and destination z, s is the scaling factor, and p is the correction period. For the reasonable values of the product $s \cdot p$, the results validate that k remains roughly the same for same $s \cdot p$.

System	d_{init}	s	Course corrections k		
			$p = 1$	$p = 5$	$p = 10$
#1	0.43	0.01	528	105	52
		0.1	52	10	3
#7	0.39	0.01	418	83	41
		0.1	41	7	3
#11	0.37	0.01	381	76	37
		0.1	37	7	3
#17	0.85	0.01	550	109	54
		0.1	54	10	3



(a) Original $\mathcal{RD}$ (b) Axis-aligned $\mathcal{RD}$

Figure 6.4: $\mathcal{RD}$ can be customized to provide different algorithm(s) for state space exploration with a constrained initial set. In the figure, the inner box represents the initial set. As shown, original $\mathcal{RD}$ tends to generate smoother trajectories because it moves in the direction of the target at each step.

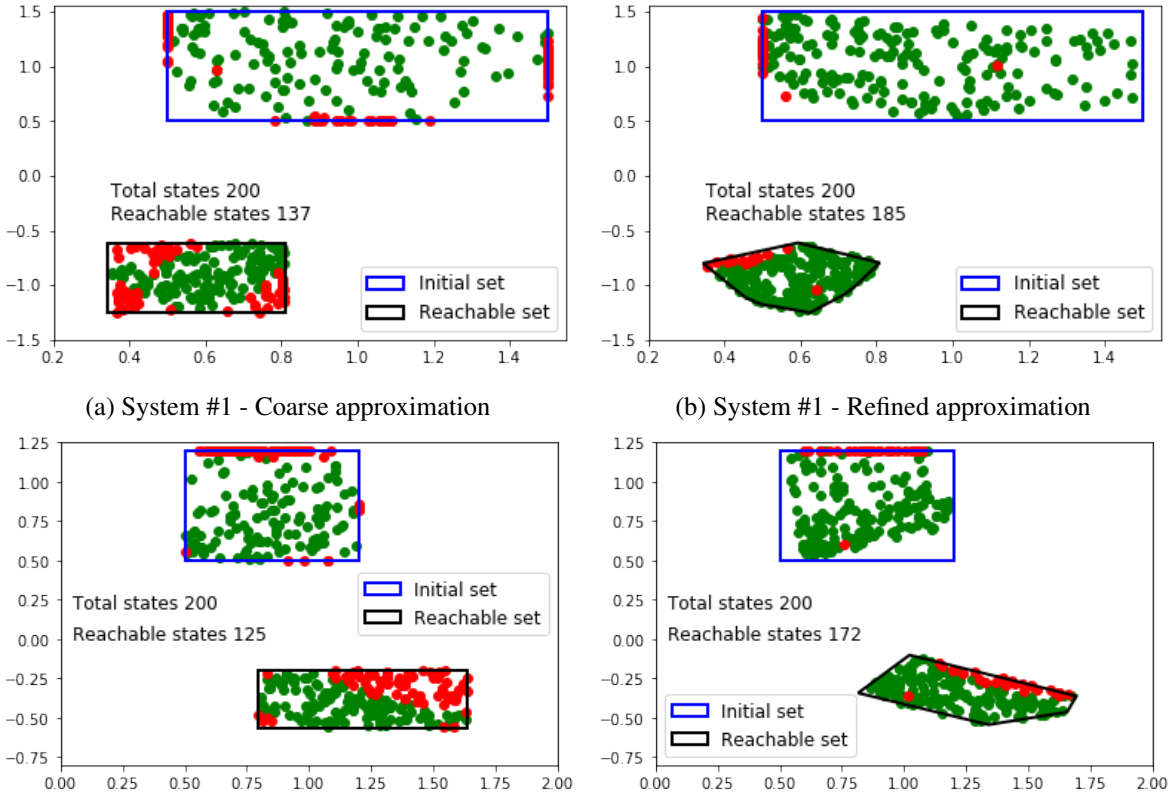
resulting into the unsatisfiability of the reachability problem definition. We address this problem by picking a element-wise projection of $\hat{x}_0^k$ in θ denoted as $\hat{x}_0^{k,\theta} \doteq proj_\theta(\hat{x}_0^k) \in \theta$, defined by $\hat{x}_0^{k,\theta} \doteq \arg\min_{x \in \theta} \|x - \hat{x}_0^k\|$. Consider System #10 with 2^{nd} component of its initial set hyper-rectangle, given as $\theta[2] \doteq [-0.5, 0.0]$. Both Figures 6.4a and 6.4b demonstrate how the course of exploration makes a detour around the initial set boundary in order to satisfy its constraints.

4. **Customizing the state space exploration algorithm:** Note that the inverse sensitivity approximator $N_{\Phi^{-1}}$ is agnostic to the exact state space exploration technique. While our implementation of $\mathcal{RD}$ uses this estimator to proceed in a straight line direction towards the destination (i.e. v^k has the same direction as $z - x_t^k$), the progress direction can also be customized. This allows

for designing custom state space exploration algorithm by prioritizing trajectories along different directions at different steps. For an n-dimensional system, at every step, one might be interested in picking a direction among the $2n$ unit vectors $\pm\hat{e}_j, j \in \{1, 2, \ldots n\}$ that are aligned with the orthonormal axes. For instance, one can choose the direction vector that is closest to $z - x_t$. The illustration of one such *axis-aligned* approach is given in Figure 6.4b. It emphasizes that instead of $\mathcal{RD}$, we can also use some other state space exploration algorithm that requires an inverse sensitivity approximator.

5. **Coverage analysis:** Given an initial set $\theta \subseteq \mathbb{D}$, we assess the coverage among the set of reachable states at time $t \in [0, T]$ by calculating the proportion of points in the reachable set that $\mathcal{RD}$ converges to, within a neighborhood of radius δ. To obtain a convenient representation of the reachable set for an n-dimensional system, we use a polygon with faces in the $2n$ template directions $\{\pm e_i : i = 1, 2, \ldots, n\}$. While we have used orthonormal vectors as template directions, different set of template directions can yield a less conservative approximation of the reachable set. The polygon in our experiment was obtained by starting from the destination state of random anchor trajectory $\xi_A(\cdot)$ at time t, and using a modification of $\mathcal{RD}$ to maximally perturbs the destination state in each of the template directions. This provides as many extremal points as the number of template directions, and can be used to construct the bounding polygon (e.g. see the black rectangle in Figure 6.5a), denoted by $\mathcal{Z}$, as an approximation of the reachable set. Next, to assess the coverage for $\mathcal{Z}$, we sampled 200 points from $\mathcal{Z}$ uniformly at random, and examined which were the ones that $\mathcal{RD}$ could converge within a $\delta = 4 \times 10^{-3}$ neighborhood at time t starting from the initial set θ. As shown in Figure 6.5a, 137 out of these 200 points were reached from $\mathcal{RD}$, with the color of the point (green or red) representing if $\mathcal{RD}$ was successful or not. For each of these points in $\mathcal{Z}$, we also plot the best initial point output by $\mathcal{RD}$. Most of these red initial points (that did not reach the destination) lie on the boundary of the initial set θ, suggesting that the trajectory that can possibly reach its destination might perhaps start from a state outside the given initial set.

   While the reachable set coverage - as a side effect - does provide an intuition about the coverage of the initial set, one can run another set of experiments to explicitly measure initial set coverage. A given initial set is partitioned into multiple subsets and a fixed number of states are sampled in



(a) System #1 - Coarse approximation

(b) System #1 - Refined approximation

(c) System #2 - Coarse approximation

(d) System #2 - Refined approximation

Figure 6.5: Measuring coverage of a set in Systems #1 and #2. For every red colored state in the destination set, $\mathcal{RD}$ could not find a trajectory that reaches within its δ-neighborhood.

each of these subsets. Now, simulations $\xi(x, \cdot)$ are generated from these sampled states x and, for a given time t, the points $\xi(x, t)$ in simulations constitute the set of destinations. Finally, $\mathcal{RD}$ is invoked for these set of destination points and their corresponding initial states as returned by the procedure yields a notion of coverage for the initial set as shown in Figure 6.6. Another notion of coverage in the given initial set is to sample states on its boundary and check how many boundary states are reachable from the centroid of the initial set.

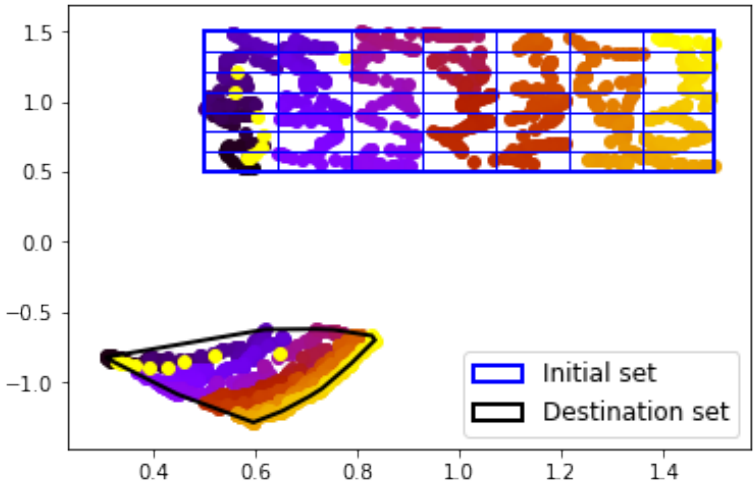


Figure 6.6: Measuring coverage of the initial set. States are sampled in the destination set and inverse sensitivity approximator is used to obtain their corresponding initial states.

6.4 Falsification of a Safety Specification

For a given system and corresponding safety specification in either Signal or Metric Temporal Logic (Maler, Nickovic & Pnueli 2008, Koymans 1990), the *falsification* is aimed at finding a system parameter or an input that violates the specification. Existing falsification schemes generate executions using some heuristics or stochastic global optimization and compute their robustness w.r.t. the safety specification denoted as a set of states. *Robustness* ($\rho \in \mathbb{R}$) is used as a measure to quantify how deep is the execution within the set or how far away it is from the set. Informally, it determines the degree to which an execution satisfies ($\rho > 0$) or violates ($\rho < 0$) a given safety specification. Our framework can currently handle a subset of MTL formulas.

$$\varphi ::= \top \mid p \mid \neg\varphi \mid \top \mathcal{U}_l \varphi$$

where p is an atomic proposition, l is a non-empty interval of $\mathbb{R}_{\geq 0}$, and φ is a well formed MTL formula. The temporal operator $\Diamond$ $(eventually)$ is defined as $\diamond_l \varphi := \top \mathcal{U}_l \varphi$. The reader can refer to (Nghiem et al. 2010) for robust semantics of MTL formulas.

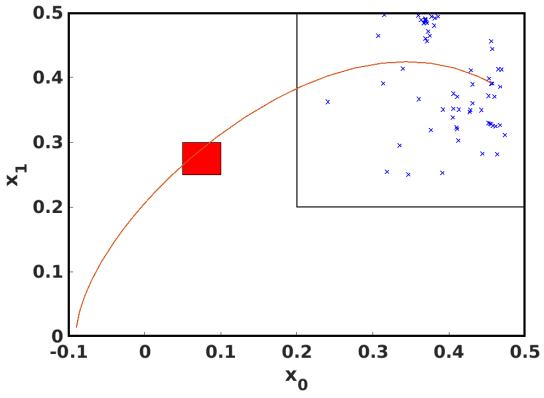
6.4.1 Our Falsification algorithm

We describe a simple $\mathcal{RD}$-based algorithm to obtain a falsifying trajectory to a given safety specification $\neg\Diamond_l U$, where $U \subseteq \mathbb{R}^n$ is the unsafe set. We generate an anchor trajectory ξ_A, sample a state $z \in U$, and choose $t = \arg\min_{t' \in l} \|\xi_A(t') - z\|$. We then invoke $\mathcal{RD}$ sub-routine for generating trajectories until we obtain a counterexample ($\rho^k < 0$) to the given safety specification or bound I is exhausted, where ρ^k is the robustness of trajectory ξ_A^k. To be precise, in the falsification run of $\mathcal{RD}$- (i) distance d_a^k is replaced by robustness ρ^k, (ii) constraint $d_a^k > \delta$ is replaced by $\rho^k > 0$, and (iii) an additional constraint $x_t^k \notin U$ is added to the main **while** loop condition. While it may be the case that z is not reachable at time t, both these parameters primarily act as anchors to guide the procedure in obtaining a falsifying execution.

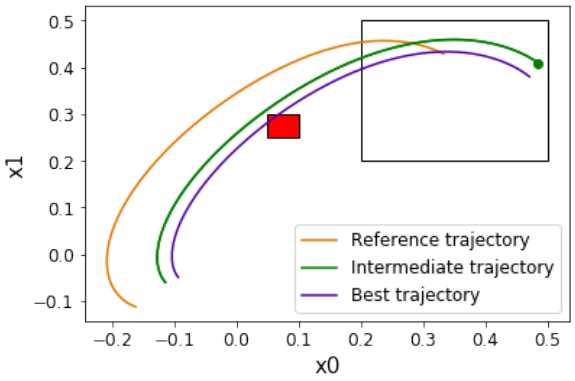
6.4.2 Evaluation of Falsification techniques

We evaluate our falsification algorithm against one of the widely used *falsification* platforms, S-TaLiRo (Sankaranarayanan & Fainekos 2012, Nghiem et al. 2010). Monte-Carlo sampling scheme in S-TaLiRo is sensitive to the "temperature" parameter β, where the adaptation of β is performed after every fixed number of iterations provided it is unable to find a counterexample by then. We keep $\beta = 50$ which is the default value, and we consider $p = 2, s = 0.5$ for our $\mathcal{RD}$-based falsification scheme. Although adaptation parameters and mechanisms in both approaches are different, an upper bound ($\mathcal{K}$) on the number of trajectories is crucial to both of them. We fix $\mathcal{K} = 100$ for systems #1-#16 and $\mathcal{K} = 150$ for systems #17-#30 in S-TaLiRo. We consider $\mathcal{K} = 50$ for NExG as we notice that, if it can, it usually finds a trajectory of interest in notably less number of iterations. The sampling time is fixed as 0.01.

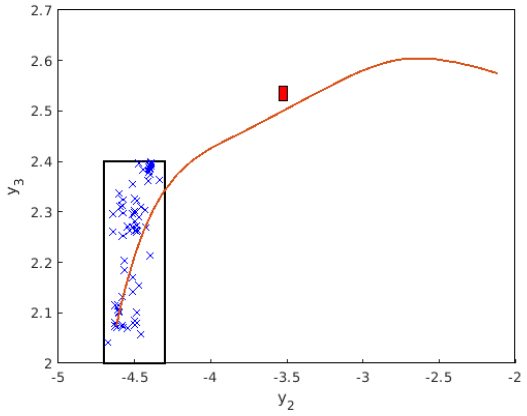
We exclude cases where the initial reference trajectory ξ_A is falsifying so as to minimize the bias induced by different distributions in different schemes. For a given pair of initial configuration θ and safety specification $\neg\Diamond_l U$ in each system, we report in Table 6.4 the *mean* of total trajectories (k) generated along with *mean* robustness (ρ) computed over 250 runs of respective techniques.



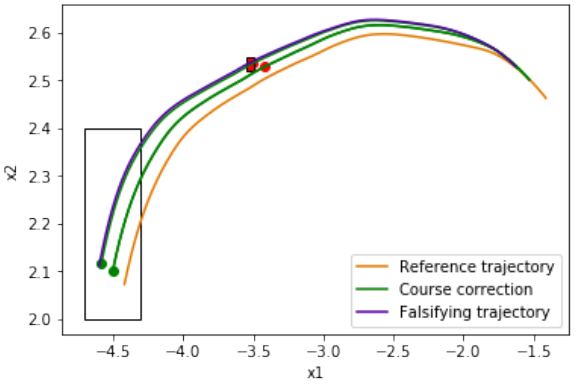
(a) #8 from S-TaLiRo



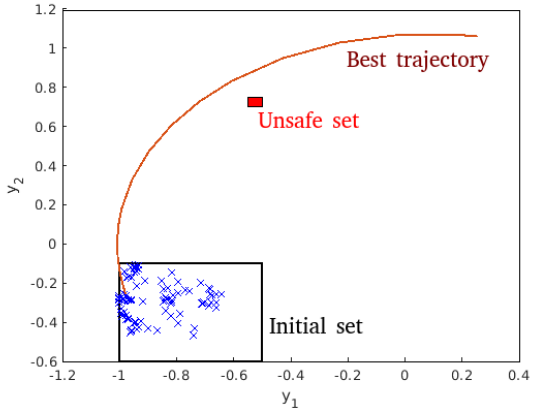
(b) #8 from NExG



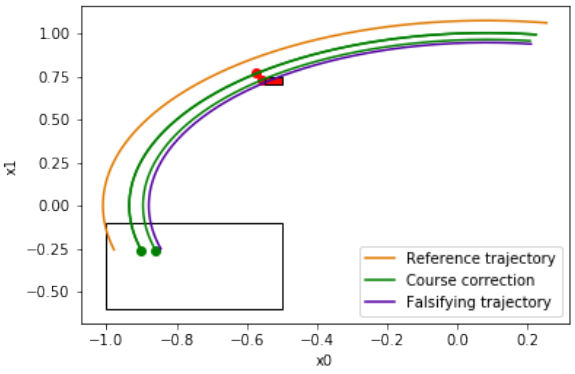
(c) #13 from S-TaLiRo



(d) #13 from NExG



(e) #11 from S-TaLiRo



(f) #11 from NExG

Figure 6.7: Falsification demonstrations. The red-colored box is the unsafe set and the inner while-colored box is the initial set. Figures show that NExG takes a very few trajectories to find a counterexample in a more directed manner and it can supplement other falsification tools.

The evaluations exhibit that our algorithm not only takes a very few trajectories to converge but also the obtained counterexample is often more robust. Unlike NExG, the performance of S-TaLiRo seems to deteriorate further with increase in the number of system dimensions and complexity. Figures 6.7a and 6.7b demonstrate this behavior. Even in scenarios where $\mathcal{RD}$ generates more than 10 trajectories, experiments indicate that it is able to reach the neighborhood around U within fewer iterations. This observation motivated us to attempt to integrate both frameworks. In the case of non-convergence in S-TaLiRo, its best execution can be used as the input *reference* trajectory for $\mathcal{RD}$. One such instance is shown in Figure 6.7e where S-TaLiRo is unable to find a falsifying trajectory within 100 iterations. We use its best sample as the reference ξ_A for $\mathcal{RD}$ and find 4^{th} trajectory to be a counterexample (Figure 6.7f). This exercise is performed for illustration purpose i.e., at present we manually port the best sample from S-TaLiRo to NExG. One of the future tasks is to automate this integration. Additionally, our approach - as a side effect - provides intuition about the course of exploration leading to the falsifying execution unlike scattered stochastically sampled states generated in S-TaLiRo.

Another important take away from this comparison is that if S-TaLiRo fails to find a counterexample for a given specification, the user is left with a sample of trajectories generated by S-TaLiRo and the execution that comes closest to falsifying the given safety specification. Instead, in our case, the user can still access the inverse sensitivity approximator and manually (or algorithmically) probe nearby trajectories and proceed to discover a falsifying trajectory. Finally, S-TaLiRo's implementation platform is `MATLAB` while our framework is implemented in `Python`. We do not report the wall-clock time taken by respective frameworks as performance differences are expected due to their different implementation platforms.

One may analogize inverse sensitivity approximator to an offline hash of results, which may make the comparison with S-TaLiRo (which performs Falsification in an online fashion) seem unfair. This analogy, however, motivates us to make some clarifications and highlight a few important differences as a rationale behind the comparison.

Note first that, in the training phase, we learn the inverse sensitivity based on trajectories starting from uniformly sampled points in the domain. In particular, instead of learning a system property specific to the task at hand, we are learning a generic system property that can be used in many applications.

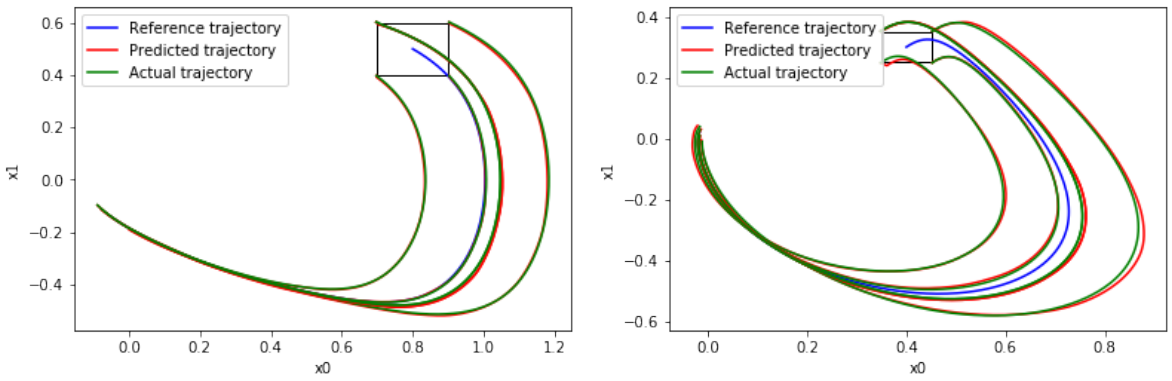
Secondly, our intention to perform this comparative analysis is not to compete with S-TaLiRo, but to emphasize the effectiveness of learning the sensitivity function in an offline manner to aid in state

space exploration. Standard gradient based techniques are not designed to use an offline component because they only rely on the current simulation to falsify the given specification based on robustness, and information from previous simulations is discarded at each step.

In our systematic state space exploration, simulation at each successive step is guaranteed (under some assumptions) to get closer to a desirable execution. We agree that we pre-process data to learn system properties, but this learned information enable us to deliver more promise by guaranteeing convergence as supported by our evaluations.

While we have primarily focused on neural-network feedback control systems, this approach can be applied to other nonlinear dynamical and hybrid systems. Our analysis with higher dimensional non-linear systems yields similar performance results. The falsification results in S-TaLiRo for *LaubLoomis* (7-dim), *Biological model I* (7-dim), and *Biological model II* (9-dim) are ($\rho = 0.003, k = 112$), ($\rho = 0.046, k = 149$), and ($\rho = 0.155, k = 150$) respectively; the results for these systems in NExG are ($\rho = 0.003, k = 21$), ($\rho = -0.0016, k = 15$), and ($\rho = 0.002, k = 29$).

6.5 Predicting Trajectories



(a) System #2 (b) System #5

Figure 6.8: Predicting system trajectories using sensitivity approximation for small perturbations. Taking the trajectory that starts from the centroid of the initial set as the reference, we predict the trajectories that start at the corners of the initial set.

As already discussed in previous chapter, predicting trajectories is another commonly used state space exploration technique. Since the numerical ODE solvers used for generating trajectories are sequential, accelerating the refinement is hard. Moreover, invoking the exact simulation engine may become computationally expensive for exploring a high-dimensional space. To alleviate these bottlenecks,

Table 6.4: Initial configuration and safety specification for falsification techniques

System	Initial configuration θ	Safety specification $\neg\Diamond_I U$
#1	[(0.5, 1.5)∧(0.5, 1.5)]	$\neg\Diamond_{[0.7,0.9]}$[(0.30, 0.35)∧(-1.1, -1.05)]
#2	[(0.8, 1.2)∧(0.9, 1.2)]	$\neg\Diamond_{[0.7,0.9]}$[(1.50, 1.55)∧(0.20, 0.25)]
#3	[(0.4, 1.2)∧(0.4, 1.2)]	$\neg\Diamond_{[0.8,1.0]}$[(0.3, 0.4)∧(0.0, 0.1)]
#4	[(1.5, 2.0)∧(1.0, 1.5)]	$\neg\Diamond_{[0.7,0.9]}$[(1.15, 1.2)∧(-0.95, -0.90)]
#5	[(0.2, 0.7)∧(0.2, 0.7)∧(0.2, 0.7)]	$\neg\Diamond_{[0.6,0.8]}$[(1.0, 1.05)∧(0.05, 0.1)∧(-1.15, -1.10)]
#6	[(0.1, 0.6)∧(0.1, 0.6)∧(0.1, 0.6)]	$\neg\Diamond_{[0.7,0.9]}$[(0.10, 0.15)∧(0.0, 0.05)∧(0.10, 0.15)]
#7	[(0.2, 0.5)∧(0.2, 0.5)∧(0.2, 0.5)]	$\neg\Diamond_{[1.0,1.2]}$[(0.4, 0.45)∧(-0.3, -0.25)∧(-0.45, -0.4)]
#8	[(0.2, 0.5)∧(0.2, 0.5)∧(0.2, 0.5)]	$\neg\Diamond_{[1.0,1.2]}$[(0.05, 0.1)∧(0.25, 0.3)∧(-0.35, -0.3)]
#9	[(0.1, 0.4)∧(0.1, 0.4) ∧(0.1, 0.4)∧(0.1, 0.4)]	$\neg\Diamond_{[0.6,0.8]}$[(-0.15, -0.10)∧(-0.80, -0.75) ∧(0.0, 0.05)∧(-0.60, -0.55)]
#10	[(0.5, 1.0)∧(-1, -0.5) ∧(-0.5, 0)∧(0.5, 1)]	$\neg\Diamond_{[0.8,1.0]}$[(-0.2, -0.15)∧(-1.05, -1.0) ∧(0.2, 0.25)∧(0.25, 0.3)]
#11	[(-1, -0.5)∧(-0.6, -0.1) ∧(0.2, 0.7)∧(-0.5, 0)]	$\neg\Diamond_{[1.0,1.2]}$[(-0.55, -0.5)∧(0.7, 0.75) ∧(0.65, 0.7)∧(0.25, 0.3)]
#12	[(-1, -0.5)∧(-0.6, -0.1) ∧(0.2, 0.7)∧(-0.5, 0)]	$\neg\Diamond_{[1.0,1.2]}$[(-0.55, -0.5)∧(0.25, 0.3) ∧(-0.05, 0.0)∧(-0.3, -0.25)]
#13	[(9.3, 9.7)∧(-4.7, -4.3) ∧(2, 2.4)∧(1.3, 1.7)]	$\neg\Diamond_{[0.8,1.0]}$[(8.4, 8.45)∧(-3.55, -3.5) ∧(2.5, 2.55)∧(2.2, 2.25)]
#14	[(1.0, 1.5)∧(1.0, 1.5) ∧(1, 1.5)∧(1, 1.5)]	$\neg\Diamond_{[1.0,1.2]}$[(1.55, 1.6)∧(0.25, 0.3) ∧(-0.65, -0.6)∧(-1.2, -1.15)]
#15	[(1.0, 1.4)∧(1.0, 1.4) ∧(1.0, 1.4)∧(1.0, 1.4)]	$\neg\Diamond_{[0.4,0.6]}$[(0.90, 0.95)∧(0.65, 0.7) ∧(-1.8, -1.75)∧(-1.20, -1.15)]
#16	[(0.0, 0.3)∧(0.0, 0.3) ∧(0.0, 0.3)∧(-0.3, 0.0)]	$\neg\Diamond_{[0.8,1.0]}$[(0.05, 0.1)∧(-0.05, 0.0) ∧(-0.05, 0.0)∧(0.0, 0.05)]
#17	[(90.0, 92.0)∧(32.0, 32.5)∧(0.0, 0.0) ∧(10.0, 11.0)∧(30.0, 30.5)∧(0.0, 0.0)]	$\neg\Diamond_{[0.7,0.9]}$[(113.5, 114.0)∧(31.3, 31.4)∧(-1.60, -1.55) ∧(32.0, 32.5)∧(29.5, 30.0)∧(-0.10, -0.05)]
#18	[(90.0, 92.0)∧(32.0, 32.5)∧(0.0, 0.0) ∧(10.0, 11.0)∧(30.0, 30.5)∧(0.0, 0.0)]	$\neg\Diamond_{[0.7,0.9]}$[(116.5, 117.0)∧(31.6, 31.7)∧(-1.65, -1.6) ∧(34.5, 35.0)∧(29.5, 30.0)∧(-0.45, -0.35)]
#19	[(90.0, 92.0)∧(32.0, 32.5)∧(0.0, 0.0) ∧(10.0, 11.0)∧(30.0, 30.5)∧(0.0, 0.0)]	$\neg\Diamond_{[0.8,1.0]}$[(120.0, 120.3)∧(31.1, 31.2)∧(-1.75, -1.7) ∧(37.0, 38.0)∧(30.5, 31.0)∧(0.1, 0.2)]
#20	[(90.0, 92.0)∧(32.0, 32.5)∧(0.0, 0.0) ∧(10.0, 11.0)∧(30.0, 30.5)∧(0.0, 0.0)]	$\neg\Diamond_{[0.6,0.8]}$[(112.8, 112.9)∧(31.7, 31.8)∧(-1.55, -1.5) ∧(31.0, 31.5)∧(30.1, 30.2)∧(0.0, 0.1)]

Table 6.5: Performance of falsification techniques. k is the number of simulations generated and ρ is the robustness. The parity of ρ determines whether the execution satisfies ($\rho > 0$) or falsifies ($\rho < 0$) a given safety specification, whereas its magnitude determines how robust is the execution.

System	S-TaLiRo		NExG		System	S-TaLiRo		NExG	
	k	ρ	k	ρ		k	ρ	k	ρ
#1	12	-0.01✓	3	-0.01✓	#11	95	0.031	3	-0.016✓
#2	24	-0.006✓	3	-0.005	#12	98	0.058	6	-0.009✓
#3	8	-0.02✓	4	-0.014	#13	81	0.009	4	-0.01✓
#4	11	-0.008✓	3	-0.008✓	#14	59	-0.002✓	6	-0.002✓
#5	39	0.008	17	-0.005✓	#15	55	-0.002	8	-0.007✓
#6	21	-0.005	3	-0.007✓	#16	47	-0.002	9	-0.003✓
#7	20	-0.005	3	-0.007✓	#17	150	0.10	15	-0.001✓
#8	36	0.008	9	0.001✓	#18	150	0.81	14	-0.002✓
#9	76	0.005✓	22	0.005✓	#19	149	0.11	9	-0.007✓
#10	88	0.01	5	-0.005✓	#20	145	0.14	34	0.015✓

neural network approximation of sensitivity (N_Φ) can be used generate (predict) trajectories in a parallel fashion.

The procedure is slightly different from what was discussed in Section 5.4 as NExG approximates the gradient of sensitivity. Given a reference trajectory $\xi(x_0, \cdot)$, assume that we are interested in predicting a trajectory $\xi(x_0', \cdot)$. For $w \doteq x_0' - x_0$ and $s \ll 1$, we use $N_\Phi(x_0, sw, t)$ to estimate the perturbation needed at $\xi(x_0, t), \forall t \in [1, T]$. This way we predict the trajectory $\xi(x_0 + sw, \cdot)$ (see Equation 5.1). We treat this trajectory as the next reference and repeat the procedure until we predict the trajectory $\xi(x_0', \cdot)$.

The choice of reference trajectory is crucial to the prediction. A distant initial reference trajectory would result into coarse approximation of the predicted trajectory due to the error compounded at each step. For our analysis, we attempt to predict the trajectories originating at the corners of the given initial set. Therefore, we choose the reference trajectory that starts from the centroid of the given initial set as centroid is equidistant to each corner. Figure 6.8 illustrates the result of predicting system trajectories for Systems #2 and #5. Actual trajectories in the figure are computed with a given numerical ODE solver.

6.6 Chapter Summary

This chapter has proposed a new state space exploration technique NExG that is an improvement over existing state space and falsification approaches. This work can also be used to generate near-miss trajectories. Instead of continuing to run the $\mathcal{RD}$ algorithm to search for execution that reaches the

target state, a control designer can terminate early and use a custom state space exploration using inverse sensitivity to generate *near miss* safety instances where the trajectory approaches the set of unsafe states within a threshold.

The illustration on estimating system trajectories has shown that such sensitivity approximation is considerably effective. The approach also gives the designer the freedom to choose only a subset of the initial states for only a specific time interval for prediction and refine the probability distribution for generating new states.

Acknowledgement. Miheer Dewaskar (Duke University, NC) as involved throughout the project, and in particular, helped in building theoretical framework for providing convergence guarantees.

CHAPTER 7: CONCLUSION

Traditional control design techniques have primarily focused on stability or convergence, however, with ever increasing interest in autonomous systems and their intricate behaviors, it is imperative to invest into their safety for them to become mainstream. At the same time, systems (software and hardware) as well as specifications are getting complex and more involved, thus making safety verification task extremely challenging. Many techniques have been proposed for safe CPS design and analysis. However, unlike stability analysis tools, most of these system analysis techniques may not yield much insight into the system behavior beyond proving or falsifying a property or giving a representative execution at best. There is a need for tools that can aid the designer in performing systematic state space exploration or validating the quality of existing analysis frameworks. Extension of a verification engine to validate safety violations along with other state space exploration techniques presented in this thesis can enable a control designer to gain additional information about an otherwise difficult to analyze system. Further, these behavioral validation mechanisms are flexible enough to supplement existing system engineering and analysis techniques. This, in turn, would result in to an enhanced certification process of complex cyber physical systems, a crucial step in expediting their mainstream adoption. Additionally, the state space exploration work can pave a way towards deploying neural networks for making this certification endeavor scalable.

This chapter briefly summarizes the results of this dissertation (Section 7.1) and directions for future work (Section 7.2).

7.1 Summary of Results

In this thesis, we have provided multiple ways to conduct behavioral validation of Cyber-physical systems. We have demonstrated how data can be used to learn useful system traits and methodically obtain desirable system behaviors. In the first part of this thesis, we have defined longest, deepest and robust counterexamples to a safety specification for a given linear hybrid systems. We have developed these counterexample generation algorithms in a linear hybrid system model checking tool, HyLAA.

The evaluations have shown that change in the depth direction or the size of the unsafe set can result into different counterexamples. The results have demonstrated that these counterexamples not only yield useful insights regarding system behaviors to a control designer but also helps her in comparing different controllers across iterations during control synthesis.

In the second part, we have introduced the notion of absolute longest counterexample and provide SMT and MILP-based frameworks to generate such counterexamples. The experiments demonstrate that the length of the longest counterexample can be different from an actual intersection duration of the reachable set with the unsafe set. The evaluations also illustrate that both frameworks may return different longest counterexamples of the same length. We have also presented an approach to find an unsafe execution corresponding to a pattern given by the user.

This work is followed by generating complete characterization of counterexamples to represent all modalities of a safety violation given as an STL specification. An algorithm is described to represent these characterizations as a binary decision diagram (BDD). As the size of the diagram may grow exponential in the number of overlaps between the reachable set and the unsafe set, a framework for dynamically finding isomorphic nodes has been proposed and shown to significantly reduce the size of the original decision diagram.

Finally, we have proposed two frameworks which use sensitivity (or its gradient) approximation for systematic state space exploration in non-linear dynamical systems and neural network feedback control systems. We have demonstrated with various experiments how these frameworks can be used for reachability, falsification as well as predicting system trajectories. We have also presented a theoretical study to provide convergence guarantees of the sensitivity gradient based algorithm. In addition to out-performing state of the art falsification techniques, these frameworks enable the control designer to develop custom algorithms for state space exploration and generate trajectories that navigate the state space along with additional constraints.

7.2 Directions for future work

There are multiple directions for future work based on the contributions of this dissertation.

Counterexamples guided control synthesis. While various counterexamples provides useful metrics to compare different controllers, the designer would ideally be interested in using these counterexamples

in synthesizing a safe controller similar to CEGIS techniques. As a preliminary step, we designed a mechanical method which exploits the longest contiguous and deepest counterexamples (more as heuristics) for synthesizing a safe stable controller in some cases. We give an overview of this approach in the Appendix. Although our empirical approach acts as a good first step and provides some guidance for safe controller synthesis using multiple counterexample candidates, it lacks insights on a possible theoretical connection between stability and the type of the safety violation. To reap the full benefit of our counterexample generation work, a more formal way of such counterexample driven safe controller synthesis that is generalizable to a class of systems should be explored. If synthesizing an optimal safe controller is challenging, the designer may be interested in exploiting these counterexamples in obtaining a sub-optimal (i.e., least-violating) controller (Girard & Eqtami 2021, Apaza-Perez & Girard 2022) for safety.

Another interesting direction is to extend the notion of various counterexamples to the systems with disturbances. Here, the aim could be to find the least (or highest) violating disturbance, based on a performance metric, during the iterative control synthesis, which can hopefully provide some useful information to the designer and likely expedite the design process.

Characterization of counterexamples. Our technique for the characterization of counterexamples currently supports safety specifications with one proposition. An extension to this work would be to characterize violations to a safety property specified as the conjunction of propositions. But the main challenge with such a specification is that its negation would introduce disjunction of propositions. Consequently, we may end up building as many decision diagrams as the number of propositions in the disjunction. Alternatively, we can construct a multivariate decision diagram (Brodley & Utgoff 1995) by introducing a new decision variable for every proposition. However, the size of these multiple decision diagrams or such multivariate decision tree would be exponential in the number of propositions as well as exponential in the number of overlaps between the reachable set and the unsafe set. For addressing this exponential state space blow up, one interesting future direction would be to find an approximation algorithm for BDD construction that captures a large fraction but not all characterizations of the safety violation. Another possible direction is to explore a more efficient graphical representation such as probabilistic decision trees for these characterizations.

Extensions of the NExG framework. This task can be broken into multiple sub-tasks. As our choice of network architecture, training data, scaling factor etc. is mostly empirical, one sub-task could

be to investigate other network architectures to accelerate network training, explore other parameters configurations, machine learning libraries or different activation functions to improve the performance of NExG. Another extension would be to enhance the presented falsification technique to the general class of STL specification, automate its integration with S-TaLiRo, and falsify large industrial scale designs with complex specifications.

We have briefly described a technique in NExG to visualize the coverage of the initial set or the unsafe set. But having some formal notion(s) of coverage could be useful in designing better state space exploration techniques in NExG framework and formally arguing about their performances. This notion of coverage can potentially be extended to the set coverage for a general performance specification to validate the performance and quality of an existing reachability analysis or falsification algorithm.

Many realistic modern day systems are modeled as feedback systems with environmental inputs. Therefore, extending our state space exploration framework to handle such generic systems would be important. While we have given some thought to such framework which might use Recurrent Neural Network (RNN), Long short-term memory (LSTM) or their variants, we believe such an extension requires some engineering for incorporating these machine learning frameworks into NExG. Finally, integrating our method into frameworks used for generating adversarial executions during control synthesis can greatly aid the design process.

Explore beyond sensitivity approximation. While we have leveraged sensitivity information for our state space exploration work, there might be other fundamental characteristics of a closed loop dynamical system. Thus, it may be worthwhile to explore either adapting scalable techniques from linear systems literature to closed loop- or open loop- feedback control systems or looking at some application specific behavioral properties such as superposition of trajectories, Jacobian matrices, discrepancy functions. One might also consider exploring existing frameworks used for learning state density distribution (Meng et al. 2021), reachability function (Sun & Mitra 2022), state classification (Phan et al. 2018) or PDE (Long et al. 2018) in designing more efficient systematic state space exploration schemes.

APPENDIX A: COUNTEREXAMPLE BASED CONTROL SYNTHESIS

In this chapter, we present an approach which uses longest and deepest counterexamples to a safety specification for finding a safe and stable control input. The technique is purely empirical, is developed for our own understanding, and is documented here for future references. It is a part of the future research to explore an underlying theoretical framework for establishing the relationship, if any, between safety and stability using such counterexamples.

A.1 Linear Quadratic Regulator

Linear Quadratic Regulator (LQR) is the problem of finding a control that stabilizes a time-invariant linear system to the origin. LQR has been extensively studied and researched upon by control theory experts. While we introduce the problem for helping the reader with the context, there are many texts available for someone interested in more details. One of the widely referred textbooks is *Linear Systems Theory* (Hespanha 2018).

Given a linear time-invariant system

$$\dot{x(t)} = \mathcal{A}x(t) + \mathcal{B}u(t), \tag{A.1}$$

LQR problem is aimed at finding the control input $u(t), t \in [0, \infty)$ which minimizes the following cost function (with $Q = Q' \succeq 0, R = R' \succ 0$):

$$J \triangleq \int_0^\infty [x(t)'Qx(t) + u(t)'Ru(t)]dt. \tag{A.2}$$

The matrix Q controls the energy of the system output and R controls the energy of the control signal. Since decreasing one energy requires increase in the other term, LQR seeks a controller that minimizes both energies by establishing a trade-off between them. The optimal input is generally a constant state feedback $u(t) = Kx(t)$, where $K \doteq -(R + B'PB)^{-1}B'PA$ and P satisfies a Algebraic Riccati Equation (ARE) given as $P \doteq Q + A'PA - A'PB(R + B'PB)^{-1}B'PA$.

input : Simulation equivalent reachable set computation routine: computeSimEquivReach, time bound: T, matrices: $\mathcal{A}, \mathcal{B}$, initial set: Θ, unsafe set: Ψ, LQR function: LQR, iterations bound: $\mathcal{K}$

output : Hybrid system: $\mathcal{H}$ such that computeSimEquivReach$(\mathcal{H}, \Theta) \cap \Psi = \emptyset$, iterations: k

$n \leftarrow$ get_sys_dims $(\mathcal{A})$; // n is the number of system variables

$m \leftarrow$ get_inp_dims $(\mathcal{B})$; // m is the number of inputs

$Q, R \leftarrow I_n, I_m$; // State and input weight matrices

$K \leftarrow \mathrm{LQR}(\mathcal{A}, \mathcal{B}, Q, R)$; // State feedback gains

$\mathcal{L}' \triangleq \langle \mathcal{A}', \mathbf{0} \rangle$ where $\mathcal{A}' \doteq \mathcal{A} - \mathcal{B} \times K$;

$Reach(\Theta) \leftarrow$ computeSimEquivReach $(\mathcal{L}', \Theta)$;

if $Reach(\Theta) \cap \Psi = \emptyset$ **then**

  return $\mathcal{L}'$;

end if

$lce \leftarrow \mathrm{computeLCE}(Reach(\Theta), \Psi)$; // Algorithm 2

$\zeta \leftarrow \mathrm{get\_time\_seqs}(lce, \mathrm{T})$;

$\mathcal{H} \leftarrow \mathrm{constructHybridAutomaton}(\mathcal{L}', \zeta)$;

$Reach(\Theta) \leftarrow$ computeSimEquivReach $(\mathcal{H}, \Theta)$;

$k \leftarrow 1$;

while $lce \neq \perp$ & $k \leq \mathcal{K}$ **do**

  $\bar{d} \leftarrow \mathbf{1}_n$; // n dimensional vector

  for *each orthonormal direction* $d_i \in \mathbb{R}^n$ **do**

    $\boldsymbol{d}^i_{max} \leftarrow \max d_i^T x$ given $x \in S$ where $S \in Reach(\Theta), S \cap \Psi \neq \emptyset$;

    $\boldsymbol{d}^i_{min} \leftarrow \min d_i^T x$ given $x \in S$ where $S \in Reach(\Theta), S \cap \Psi \neq \emptyset$;

    $\bar{d}[i] \leftarrow |\boldsymbol{d}^i_{max} - \boldsymbol{d}^i_{min}|$;

  end for

  $Q \leftarrow (\bar{d})^T Q$;

  $K \leftarrow \mathrm{LQR}(\mathcal{A}, \mathcal{B}, Q, R)$; // New state feedback gains

  for $l \in \mathcal{H}.L$ **do**

    $\mathcal{A}'_l \leftarrow \mathcal{A}'_l - \mathcal{B}'_l \times K$;

  end for

  $Reach(\Theta) \leftarrow$ computeSimEquivReach $(\mathcal{H}, \Theta)$;

  $k \leftarrow k + 1$; // increment iterations by 1

  $lce' \leftarrow \mathrm{computeLCE}(Reach(\Theta), \Psi)$; // Re-compute counterexample

  if $lce'.length > lce.length$ **then**

    break;

  end if

  $lce \leftarrow lce'$;

end while

if $lce \neq \perp$ & $k < \mathcal{K}$ **then**

  return $\perp$**;**

end if

else

  return $\mathcal{H}$**;**

end if

Algorithm 10: Control synthesis using a variety of counterexamples

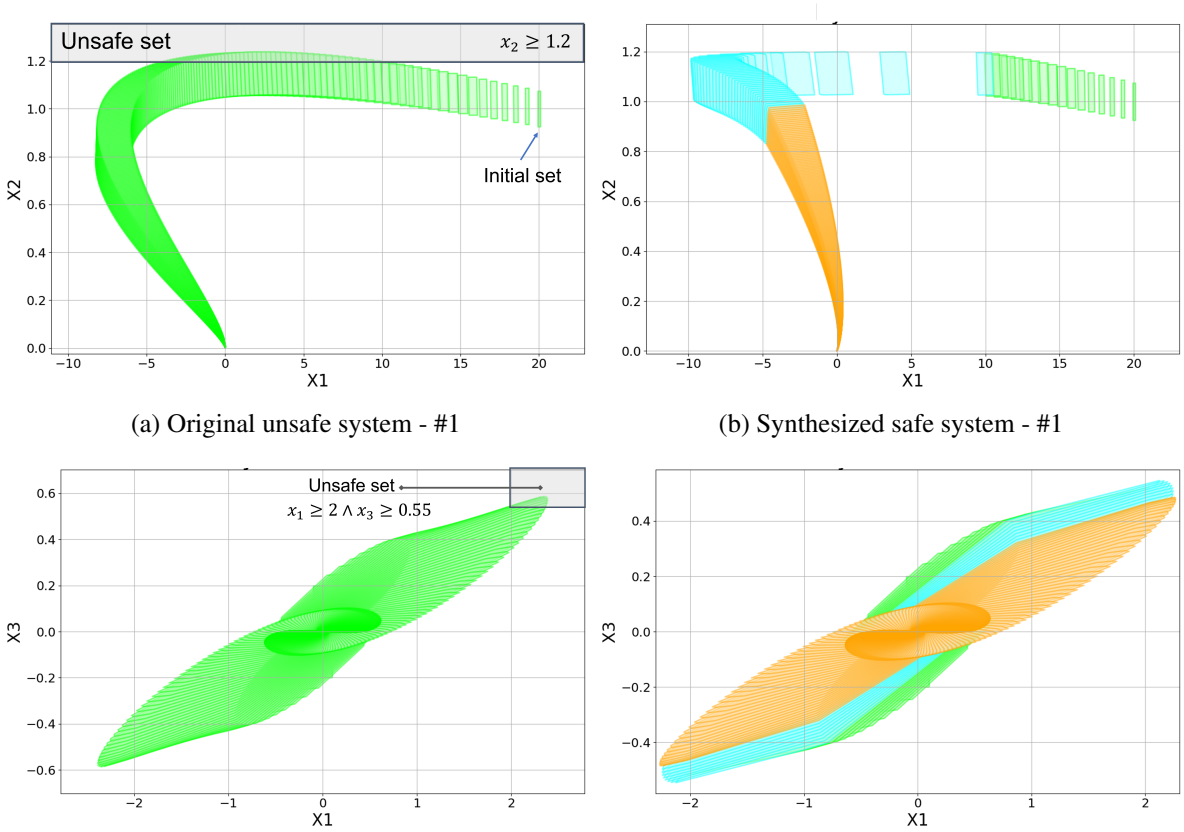
A.2 Control synthesis algorithm

The technique for synthesizing a safe control input denoted as ceBasedControlSynth (abbreviated as $\mathcal{CS}$) is presented in Algorithm 10. It takes as input the matrices $\mathcal{A}, \mathcal{B}$, time bound T, initial set Θ, unsafe set Ψ, iterations bound $\mathcal{K}$, and handles to the simulation equivalent reachable set computation routine and LQR function. If successful, the algorithm returns the synthesized linear hybrid system $\mathcal{H}$ such that its simulation equivalent reachable set is safe w.r.t. Ψ and the number of iterations k taken to obtain $\mathcal{H}$.

The algorithm initializes the matrices Q, R in line 3. It computes state feedback gains for the given dynamics matrices (line 4), uses these gains to construct a linear system (line 5) which stabilizes to the origin, and computes its simulation equivalent reachable set (line 6). If the overlap between the reachable set and the unsafe set is non-empty, it obtains a longest contiguous counterexample (lce) in line 10. Next, for the lce interval duration $[t_1, t_2]$, the routine "get_time_seqs" yields 3 intervals $[0, t_1 - 1], [t_1 - 1, t_2 + 1], [t_2 + 1, T - 1]$. These intervals are used to construct a hybrid automaton $\mathcal{H}$ (in line 12) having one location for each interval. The switching time between two successive intervals denotes the guard condition of the discrete transition between corresponding locations. For instance, the guard condition of the transition $L_1 \rightarrow L_2$ between locations L_1 and L_2 would be $G_{L_1 \rightarrow L_2} \doteq t \geq (t_1 - 1)$ and the guard of the transition $L_2 \rightarrow L_3$ would be $G_{L_2 \rightarrow L_3} \doteq t \geq (t_2 + 1)$.

The **while** loop (lines 15-34) iterates until the simulation equivalent reachable set $Reach(\Theta)$ has empty overlap with Ψ or the iterations bound $\mathcal{K}$ is exhausted. Every iteration in the while loop does the following. In lines 18-19, the deepest counterexamples in $2n$ orthonormal directions (for n dimensional system) are obtained; then, for each dimension, the difference in the depths of its corresponding counterexamples is stored in the vector $\bar{d}$ (line 20). The matrix Q is updated by computing its dot product with $\bar{d}$ in line 22. New feedback gains are obtained (line 23), the dynamics in each location of the hybrid system are updated (line 26), and simulation equivalent reachable set is re-computed in line 27). If the duration of the longest counterexample is longer than the duration of the previous one, the routine terminates (lines 30- 32); otherwise, next iteration of the **while** loop is carried out.

Evaluation and discussion. We have implemented Algorithm 10 in HyLAA and the results of performing this control synthesis technique in two systems are shown in Figure A.1. Unsafe set for



(a) Original unsafe system - #1

(b) Synthesized safe system - #1

(c) Original unsafe system - #2

(d) Synthesized safe system - #2

Figure A.1: Illustration of safe controller synthesis using counterexamples. Left hand side figures correspond to the reachable set computed for original stable unsafe system $\mathcal{L}'$. Right hand side figures are the reachable sets obtained for the hybrid system $\mathcal{H}$ returned by Algorithm 10. Different colors in the reachable set correspond to different locations in the hybrid system.

System #1 is $\Psi_1 \doteq x_2 \geq 1.2$ and for System #2 is $\Psi_2 \doteq x_1 \geq 2 \wedge x_3 \geq 0.55$. The number of iterations (k) taken for synthesizing safe control system are 2 and 7, respectively, for these systems.

These examples are presented only to illustrate how Algorithm 10 works in practice. While we could find a few such successful case studies, there were a couple of other benchmarks where the algorithm either performed worse (i.e., the duration of the longest counterexample got longer by using depths as the weights for the Q matrix) or did not terminate (i.e., the duration of the longest counterexample did not reduce after a point). As mentioned earlier, developing a theoretical framework for understanding the nuances of our counterexamples based control synthesis is a part of future research. Such theoretical study, additionally, may yield some intuition for designing a holistic approach generalizable to a larger set of systems.

APPENDIX B: OTHER WORKS

Some publications to which the author contributed fall beyond the scope of this dissertation and may as well not related to Cyber-physical systems. These contributions are summarized briefly here.

Concurrency groups: a new way to look at real-time multiprocessor lock nesting. (Nemitz, Amert, Goyal & Anderson 2019)

When designing a real-time multiprocessor locking protocol, the allowance of lock nesting creates complications that can inhibit parallelism. Such protocols are typically designed by focusing on the arbitration of re- source requests that should be prohibited from executing concurrently. This paper proposes "concurrency groups," a new concept that reflects an alternative point of view that focuses instead on requests that can be allowed to execute concurrently. A concurrency group is simply a group of lock requests, determined offline, that can safely execute together. This paper's main contribution is the CGLP, a new real-time multiprocessor locking protocol that supports lock nesting through the use of concurrency groups. The CGLP is able to reap run time parallelism benefits that have eluded prior protocols by investing effort offline in the construction of concurrency groups. A schedulability study is presented to quantify these benefits, as well as an approach to determining such groups using an Integer Linear Program (ILP) solver, which we show to be efficient in practice.

Safety and progress proofs for a reactive planner and controller for autonomous driving. (Karimi, Goyal & Duggirala 2021)

We perform a safety and progress analysis of a *map-less* path planner and path-tracking controller for an autonomous car racing on a circuit. Here, *map-less* means that the planner operates purely based on the current on-board sensor data to follow the racing track. In contrast, a *map-based* planner uses both sensor data and a map (either known apriori or built at run time) to localize the vehicle within the map and perhaps plan a racing line. Our planner chooses a path on the Voronoi diagram of the current perception and our controller uses *pure-pursuit* to track the path. A version of our planner and controller won the Generalized RAcing Intelligence Competition (GRAIC) which includes dynamic obstacles as well. Our safety and performance analysis has two parts. The first part gives sufficient conditions such that the oblivious plan is consistent with a map-based plan computed from the Voronoi diagram of the full

map of the racing track. The second part proves the safety and progress with respect to a hybrid automata model of the closed system (i.e. the map, planner, and controller), using reachable set computation. The map is needed for the proofs, but not needed by the motion planner. As examples, we prove the safety and progress of our planner for five circuits. Furthermore, the convergence of the computed reachable sets to the tracking path proves the stability of the pure-pursuit controller.

An Approach to Align Programs for Checking Equivalence. (Goyal, Azeem, Madhukar & Venkatesh 2021)

The problem of checking whether two programs are semantically equivalent or not has a diverse range of applications, and is consequently of substantial importance. Our current motivation comes from the need to automatically evaluate an assignment submission in a programming course, by comparing it with a reference implementation made available by the teacher. This task is typically done by generating test-cases, and then checking that the two programs behave similarly for all the tests. It is certainly desirable to not be limited to test-cases, and to ask the question of equivalence in general, i.e. whether the two programs would behave similarly for every input. There are several techniques that address this problem, chiefly by constructing a product program that makes it easier to derive useful invariants. A novel addition to these is a technique that uses alignment predicates to align traces of the two programs, in order to construct a program alignment automaton. Being guided by predicates is not just beneficial in dealing with syntactic dissimilarities, but also in staying relevant to the property. However, there are also drawbacks of a trace-based technique. Obtaining traces that cover all program behaviors is difficult, and any under-approximation may lead to an incomplete product program. Moreover, an indirect construction of this kind is unaware of the missing behaviors, and has no control over the aforesaid incompleteness. This paper, addressing these concerns, presents an algorithm to construct the program alignment automaton directly instead of relying on traces.

Artificial neural networks in tandem with molecular descriptors as predictive tools for continuous liposome manufacturing. (Sansare, Duran, Mohammadiarani, Goyal, Yenduri, Costa, Xu, O'Connor, Burgess & Chaudhuri 2021)

The current study utilized an artificial neural network (ANN) to generate computational models to achieve process optimization for a previously developed continuous liposome manufacturing system. The liposome formation was based on a continuous manufacturing system with a co-axial turbulent jet in a co-flow technology. The ethanol phase with lipids and aqueous phase resulted in liposomes of homogeneous sizes. The input features of the ANN included critical material attributes (CMAs) (e.g., hydrocarbon tail length, cholesterol percent, and buffer type) and critical process parameters (CPPs) (e.g., solvent temperature and flow rate), while the ANN outputs included critical quality attributes (CQAs) of liposomes (i.e., particle size and polydispersity index (PDI)). Two common ANN architectures, multiple-input-multiple-output (MIMO) models and multiple-input–single-output (MISO) models, were evaluated in this study, where the MISO outperformed MIMO with improved accuracy. Molecular descriptors, obtained from PaDEL-Descriptor software, were used to capture the physio-chemical properties of the lipids and used in training of the ANN. The combination of CMAs, CPPs, and molecular descriptors as inputs to the MISO ANN model reduced the training and testing mean relative error. Additionally, a graphic user interface (GUI) was successfully developed to assist the end-user in performing interactive simulated risk analysis and visualizing model predictions.

BIBLIOGRAPHY

Abbas, Houssam & Georgios E. Fainekos. 2011. Linear Hybrid System Falsification through Local Search. In *Automated Technology for Verification and Analysis, 9th International Symposium, ATVA 2011, Taipei, Taiwan, October 11-14, 2011. Proceedings*. pp. 503–510.

Adimoolam, Arvind & Indranil Saha. 2022. Using Intersection of Unions to Minimize Multi-Directional Linearization Error in Reachability Analysis. In *Proceedings of the 25th ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '22 New York, NY, USA: Association for Computing Machinery.

Adimoolam, Arvind, Thao Dang, Alexandre Donzé, James Kapinski & Xiaoqing Jin. 2017. Classification and Coverage-Based Falsification for Embedded Control Systems. In *Computer Aided Verification*, ed. Rupak Majumdar & Viktor Kunčak. Cham: Springer International Publishing pp. 483–503.

Akers. 1978. "Binary Decision Diagrams." *IEEE Transactions on Computers* C-27(6):509–516.

Allen Zhu, Zeyuan, Zhenyu Liao & Lorenzo Orecchia. 2014. "Using Optimization to Find Maximum Inscribed Balls and Minimum Enclosing Balls." *CoRR* abs/1412.1001.

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

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

Alur, R., T. Dang & F. Ivancic. 2006. "Counterexample-guided predicate abstraction of hybrid systems." *Theoretical Computer Science* 354(2):250–271.

Alur, Rajeev, Thao Dang & Franjo Ivančić. 2003. Progress on Reachability Analysis of Hybrid Systems Using Predicate Abstraction. In *Hybrid Systems: Computation and Control*, ed. Oded Maler & Amir Pnueli.

Annpureddy, Yashwanth, Che Liu, Georgios Fainekos & Sriram Sankaranarayanan. 2011. S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. Parosh Aziz Abdulla & K. Rustan M. Leino. Berlin: Springer pp. 254–257.

Apaza-Perez, W & Antoine Girard. 2022. "Synthesis of Input-to-State Safety and Attractivity Controllers using Nested Sequences of Abstractions.".

Argentim, Lucas M., Willian C. Rezende, Paulo E. Santos & Renato A. Aguiar. 2013. PID, LQR and LQR-PID on a quadcopter platform. In *International Conference on Informatics, Electronics and Vision*. pp. 1–6.

Bak, Stanley, Omar Ali Beg, Sergiy Bogomolov, Taylor T. Johnson, Luan Viet Nguyen & Christian Schilling. 2019. "Hybrid automata: from verification to implementation." *International Journal on Software Tools for Technology Transfer* .

Bak, Stanley & Parasara Sridhar Duggirala. 2017*a*. Hylaa: A tool for computing simulation-equivalent reachability for linear systems. In *Proceedings of the 20th International Conference on Hybrid Systems: Computation and Control*. ACM pp. 173–178.

Bak, Stanley & Parasara Sridhar Duggirala. 2017*b*. Rigorous simulation-based analysis of linear hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer pp. 555–572.

Bak, Stanley & Parasara Sridhar Duggirala. 2017*c*. Simulation-Equivalent Reachability of Large Linear Systems with Inputs. In *Computer Aided Verification*, ed. Rupak Majumdar & Viktor Kunčak. Cham: Springer International Publishing pp. 401–420.

Barbon, Gianluca, Vincent Leroy & Gwen Salaün. 2018. Counterexample Simplification for Liveness Property Violation. In *Software Engineering and Formal Methods*, ed. Einar Broch Johnsen & Ina Schaefer. Cham: Springer International Publishing pp. 173–188.

Barbon, Gianluca, Vincent Leroy & Gwen Salaün. 2019. Debugging of Behavioural Models with CLEAR. In *TACAS 2019 - 25th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Vol. 11427 of *Lecture Notes in Computer Science* Prague, Czech Republic: Springer pp. 386–392.

Beer, Ilan, Shoham Ben-David, Cindy Eisner & Yoav Rodeh. 1997. Efficient detection of vacuity in ACTL formulas. In *Computer Aided Verification*, ed. Orna Grumberg. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 279–290.

Beer, Ilan, Shoham Ben-David, Hana Chockler, Avigail Orni & Richard Trefler. 2009. Explaining Counterexamples Using Causality. In *Computer Aided Verification*, ed. Ahmed Bouajjani & Oded Maler. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 94–108.

Beg, Omar, Ali Davoudi & Taylor T. Johnson. 2017. Reachability Analysis of Transformer-Isolated DC-DC Converters. In *ARCH17. 4th International Workshop on Applied Verification of Continuous and Hybrid Systems, collocated with Cyber-Physical Systems Week (CPSWeek) on April 17, 2017 in Pittsburgh, PA, USA*. pp. 52–64.

Behle, Markus. 2007. Binary decision diagrams and integer programming PhD dissertation Max Planck Institute for Computer Science.

Berger, Guillaume O. & Sriram Sankaranarayanan. 2022. "Learning fixed-complexity polyhedral Lyapunov functions from counterexamples.".

Bogomolov, Sergiy, Goran Frehse, Amit Gurung, Dongxu Li, Georg Martius & Rajarshi Ray. 2019. Falsification of Hybrid Systems Using Symbolic Reachability and Trajectory Splicing. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19 NY, USA: ACM p. 1–10.

Bollig, Beate. 2014. On the Width of Ordered Binary Decision Diagrams. In *Combinatorial Optimization and Applications*, ed. Zhao Zhang, Lidong Wu, Wen Xu & Ding-Zhu Du. Cham: Springer International Publishing pp. 444–458.

Bollig, Beate & Ingo Wegener. 1996. "Improving the Variable Ordering of OBDDs Is NP-Complete." *IEEE Trans. Comput.* 45(9):993–1002.

Bradley, Aaron R. 2011. SAT-Based Model Checking without Unrolling. In *Verification, Model Checking, and Abstract Interpretation*, ed. Ranjit Jhala & David Schmidt. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 70–87.

Branicky, Michael S. 1998. "Multiple Lyapunov functions and other analysis tools for switched and hybrid systems." *IEEE Transactions on automatic control* 43(4):475–482.

Brodley, Carla E. & Paul E. Utgoff. 1995. "Multivariate Decision Trees." *Machine Learning* 19(1):45–77.

Bryant. 1986. "Graph-Based Algorithms for Boolean Function Manipulation." *IEEE Transactions on Computers* C-35(8):677–691.

Butler, Kenneth M., Don E. Ross, Rohit Kapur & M. Ray Mercer. 1991. Heuristics to Compute Variable Orderings for Efficient Manipulation of Ordered Binary Decision Diagrams. In *Proceedings of the 28th ACM/IEEE Design Automation Conference*. DAC '91 New York, NY, USA: Association for Computing Machinery p. 417–420.

Chen, Ricky T. Q., Yulia Rubanova, Jesse Bettencourt & David Duvenaud. 2018. "Neural Ordinary Differential Equations.".

Chen, Xin, Erika Ábrahám & Sriram Sankaranarayanan. 2013. Flow*: An Analyzer for Non-linear Hybrid Systems. In *Computer Aided Verification*, ed. Natasha Sharygina & Helmut Veith.

Chollet, François et al. 2015. "Keras." `https://github.com/fchollet/keras`.

Chung, P.-Y., I.M. Hajj & J.H. Patel. 1993. Efficient variable ordering heuristics for shared ROBDD. In *1993 IEEE International Symposium on Circuits and Systems*. Vol. 3 pp. 1690–1693.

Clarke, Edmund, Ansgar Fehnker, Zhi Han, Bruce Krogh, Olaf Stursberg & Michael Theobald. 2003. Verification of Hybrid Systems Based on Counterexample-Guided Abstraction Refinement. In *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. TACAS'03 Berlin, Heidelberg: Springer-Verlag p. 192–207.

Clarke, Edmund, Orna Grumberg, Somesh Jha, Yuan Lu & Helmut Veith. 2000. Counterexample-Guided Abstraction Refinement. In *Computer Aided Verification*, ed. E. Allen Emerson & Aravinda Prasad Sistla. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 154–169.

Dang, Thao, Alexandre Donze, Oded Maler & Noa Shalev. 2008. Sensitive state-space exploration. In *2008 47th IEEE Conference on Decision and Control*. pp. 4049–4054.

Dang, Thao & Oded Maler. 1998. Reachability analysis via face lifting. In *Hybrid Systems: Computation and Control*, ed. Thomas A. Henzinger & Shankar Sastry. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 96–109.

de Moura, Leonardo & Nikolaj Bjørner. 2008. Z3: An Efficient SMT Solver. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. C. R. Ramakrishnan & Jakob Rehof. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 337–340.

Deshmukh, Jyotirmoy V., Georgios E. Fainekos, James Kapinski, Sriram Sankaranarayanan, Aditya Zutshi & Xiaoqing Jin. 2015. Beyond single shooting: Iterative approaches to falsification. In *American Control Conference, ACC 2015, Chicago, IL, USA, July 1-3, 2015*. p. 4098.

Dierks, Henning, Sebastian Kupferschmid & Kim G. Larsen. 2007. Automatic Abstraction Refinement for Timed Automata. In *Formal Modeling and Analysis of Timed Systems*, ed. Jean-François Raskin & P. S. Thiagarajan. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 114–129.

Ding, Jerry & Claire J. Tomlin. 2010. Robust reach-avoid controller synthesis for switched nonlinear systems. In *49th IEEE Conference on Decision and Control (CDC)*. pp. 6481–6486.

Donzé, Alexandre. 2010. Breach, A Toolbox for Verification and Parameter Synthesis of Hybrid Systems. In *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010.* pp. 167–170.

Donzé, Alexandre & Oded Maler. 2007. Systematic simulation using sensitivity analysis. In *International Workshop on Hybrid Systems: Computation and Control*.

Donzé, Alexandre & Oded Maler. 2010. Robust Satisfaction of Temporal Logic over Real-Valued Signals. In *Formal Modeling and Analysis of Timed Systems - 8th International Conference, FORMATS 2010, Klosterneuburg, Austria, September 8-10, 2010. Proceedings*. pp. 92–106.

Duggirala, Parasara Sridhar & Mahesh Viswanathan. 2016. Parsimonious, Simulation Based Verification of Linear Systems. In *Computer Aided Verification*, ed. Swarat Chaudhuri & Azadeh Farzan. Cham: Springer International Publishing pp. 477–494.

Duggirala, Parasara Sridhar, Matthew Potok, Sayan Mitra & Mahesh Viswanathan. 2015. C2E2: A Tool for Verifying Annotated Hybrid Systems. In *HSCC*. New York, NY, USA: Association for Computing Machinery p. 307–308.

Duggirala, Parasara Sridhar & Sayan Mitra. 2011. Abstraction-Refinement for Stability. In *Proceedings of 2nd IEEE/ACM International Conference on Cyber-physical systems (ICCPS 2011)*. Chicago, IL: .

Duggirala, Parasara Sridhar, Sayan Mitra & Mahesh Viswanathan. 2013. Verification of Annotated Models from Executions. In *Proceedings of the 11th ACM International Conference on Embedded Software*. Montreal, Canada: IEEE Press.

Dutta, Souradeep, Xin Chen & Sriram Sankaranarayanan. 2019. Reachability Analysis for Neural Feedback Systems Using Regressive Polynomial Rule Inference. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. ACM p. 157–168.

Dutta, Souradeep, Xin Chen, Susmit Jha, Sriram Sankaranarayanan & Ashish Tiwari. 2019. Sherlock - A Tool for Verification of Neural Network Feedback Systems: Demo Abstract. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19 New York, NY, USA: Association for Computing Machinery p. 262–263.

Fainekos, Georgios E. & George J. Pappas. 2009. "Robustness of temporal logic specifications for continuous-time signals." *TCS* 410.

Fan, Chuchu & Sayan Mitra. 2015. Bounded verification with on-the-fly discrepancy computation. In *International Symposium on Automated Technology for Verification and Analysis*. pp. 446–463.

Fan, Chuchu, Umang Mathur, Sayan Mitra & Mahesh Viswanathan. 2018. Controller Synthesis Made Real: Reach-Avoid Specifications and Linear Dynamics. In *Computer Aided Verification*, ed. Hana Chockler & Georg Weissenbacher. Cham: Springer pp. 347–366.

Fey, Görschwin & Rolf Drechsler. 2003. Finding Good Counter-Examples to Aid Design Verification. In *Proceedings of the First ACM and IEEE International Conference on Formal Methods and Models for Co-Design*. MEMOCODE '03 USA: IEEE Computer Society p. 51.

Frehse, Goran. 2005. PHAVer: Algorithmic Verification of Hybrid Systems Past HyTech. In *Hybrid Systems: Computation and Control*, ed. Manfred Morari & Lothar Thiele. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 258–273.

Frehse, Goran, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Manish Goyal, Rodolfo Ripado, Thao Dang, Oded Maler, Colas Le Guernic & Antoine Girard. 2011. Safety Analysis of Hybrid Systems with SpaceEx. In *Computational Modeling and Analysis for Complex Systems (CMACS) Seminar*. CMACS.

Frehse, Goran, Bruce H. Krogh & Rob A. Rutenbar. 2006. Verifying Analog Oscillator Circuits Using Forward/Backward Abstraction Refinement. In *Proceedings of the Conference on Design, Automation and Test in Europe: Proceedings*. DATE '06 3001 Leuven, Belgium: European Design and Automation Association pp. 257–262.

Frehse, Goran, Colas Le Guernic, Alexandre Donzé, Scott Cotton, Rajarshi Ray, Olivier Lebeltel, Rodolfo Ripado, Antoine Girard, Thao Dang & Oded Maler. 2011. SpaceEx: Scalable Verification of Hybrid Systems. In *Proc. 23rd International Conference on Computer Aided Verification (CAV)*. LNCS Springer.

Ghosh, Shromona, Dorsa Sadigh, Pierluigi Nuzzo, Vasumathi Raman, Alexandre Donzé, Alberto L. Sangiovanni-Vincentelli, S. Shankar Sastry & Sanjit A. Seshia. 2016. Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. HSCC '16 New York, NY, USA: ACM p. 31–40.

Girard, Antoine & Alina Eqtami. 2021. "Least-violating symbolic controller synthesis for safety, reachability and attractivity specifications." *Automatica* 127:109543.

Goubault, Eric & Sylvie Putot. 2019. Inner and Outer Reachability for the Verification of Control Systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. New York, NY, USA: ACM p. 11–22.

Goyal, Manish. 2012. "Reachability Analysis of Hybrid Systemsan Experience Report." *International Journal of Modeling and Optimization* 2(6):681.

Goyal, Manish, David Bergman & Parasara Sridhar Duggirala. 2020. Generating Longest Counterexample: On the Cross-roads of Mixed Integer Linear Programming and SMT. In *American Control Conference*. IEEE pp. 1823–1829.

Goyal, Manish, Miheer Dewaskar & Parasara Sridhar Duggirala. 2022. "NExG: Provable and Guided State Space Exploration of Neural Network Control Systems using Sensitivity Approximation." *CoRR* abs/2207.03884.

Goyal, Manish, Muqsit Azeem, Kumar Madhukar & R. Venkatesh. 2021. "Direct Construction of Program Alignment Automata for Equivalence Checking." *CoRR* abs/2109.01864.

Goyal, Manish & Parasara Sridhar Duggirala. 2018. On Generating a Variety of Unsafe Counterexamples for Linear Dynamical Systems. In *Proc. 6th IFAC Conference on Analysis and Design of Hybrid Systems*. IFAC-PapersOnLine Elsevier.

Goyal, Manish & Parasara Sridhar Duggirala. 2019. Learning Robustness of Nonlinear Systems Using Neural Networks. In *Design and Analysis of Robust Systems, DARS 2019, New York, USA, July 13, 2019*.

Goyal, Manish & Parasara Sridhar Duggirala. 2020*a*. "Extracting counterexamples induced by safety violation in linear hybrid systems." *Automatica* 117:109005.

Goyal, Manish & Parasara Sridhar Duggirala. 2020*b*. NeuralExplorer: State Space Exploration of Closed Loop Control Systems Using Neural Networks. In *Automated Technology for Verification and Analysis*. Vol. 12302 of *Lecture Notes in Computer Science* Springer pp. 75–91.

Goyal, Manish & Parasara Sridhar Duggirala. 2020*c*. NeuralExplorer: State Space Exploration of Closed Loop Control Systems Using Neural Networks. In *L4DC*. Vol. 120 of *Proceedings of Machine Learning Research* PMLR p. 697.

Groce, Alex & Willem Visser. 2003. What Went Wrong: Explaining Counterexamples. In *Model Checking Software*, ed. Thomas Ball & Sriram K. Rajamani. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 121–136.

Grumberg, Orna, Shlomi Livne & Shaul Markovitch. 2003. "Learning to Order BDD Variables in Verification." *J. Artif. Int. Res.* 18(1):83–116.

Gühring, Ingo, Mones Raslan & Gitta Kutyniok. 2020. "Expressivity of deep neural networks." *arXiv preprint arXiv:2007.04759* .

Gurobi Optimization, LLC. 2018. "Gurobi Optimizer Reference Manual.".
URL: *http://www.gurobi.com*

Hespanha, João P. 2018. *Linear Systems Theory*. Princeton, New Jersey: Princeton Press. ISBN13: 9780691179575.

Hu, Alan John. 1995. Techniques for Efficient Formal Verification Using Binary Decision Diagrams PhD dissertation cs-tr-95-1561 Stanford University.

Hu, Haimin, Mahyar Fazlyab, Manfred Morari & George J. Pappas. 2020. Reach-SDP: Reachability Analysis of Closed-Loop Systems with Neural Network Controllers via Semidefinite Programming. In *CDC*. IEEE pp. 5929–5934.

Huang, Chao, Jiameng Fan, Wenchao Li, Xin Chen & Qi Zhu. 2019. "ReachNN: Reachability Analysis of Neural-Network Controlled Systems." *ACM Trans. Embed. Comput. Syst.* 18(5s).

Huang, Xiaowei, Marta Kwiatkowska, Sen Wang & Min Wu. 2017. Safety Verification of Deep Neural Networks. In *Computer Aided Verification*, ed. R. Majumdar & V. Kunčak. Cham: Springer pp. 3–29.

Huang, Zhenqi & Sayan Mitra. 2014. Proofs from Simulations and Modular Annotations. In *Proceedings of the 17th International Conference on Hybrid Systems: Computation and Control*. ACM p. 183–192.

Huang, Zhenqi, Yu Wang, Sayan Mitra, Geir E. Dullerud & Swarat Chaudhuri. 2015. Controller synthesis with inductive proofs for piecewise linear systems: An SMT-based algorithm. In *2015 54th IEEE Conference on Decision and Control (CDC)*. pp. 7434–7439.

Immler, Fabian, Matthias Althoff, Xin Chen, Chuchu Fan, Goran Frehse, Niklas Kochdumper, Yangge Li, Sayan Mitra, Mahendra Singh Tomar & Majid Zamani. 2018. ARCH-COMP18 Category Report: Continuous and Hybrid Systems with Nonlinear Dynamics. In *ARCH@ADHS*.

Ivanov, Radoslav, James Weimer, Rajeev Alur, George J. Pappas & Insup Lee. 2019. Verisig: Verifying Safety Properties of Hybrid Systems with Neural Network Controllers. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. HSCC '19 New York, NY, USA: Association for Computing Machinery p. 169–178.

Ivanov, Radoslav, Taylor Carpenter, James Weimer, Rajeev Alur, George Pappas & Insup Lee. 2021. Verisig 2.0: Verification of Neural Network Controllers Using Taylor Model Preconditioning. In *Computer Aided Verification*, ed. A. Silva & K. Rustan M. Leino. Cham: Springer International Publishing pp. 249–262.

Jankovic, M., D. Fontaine & P. V. Kokotovic. 1996. "TORA example: cascade- and passivity-based control designs." *IEEE Transactions on Control Systems Technology* 4(3):292–297.

Jha, Susmit & Sanjit A. Seshia. 2014. Are There Good Mistakes? A Theoretical Analysis of CEGIS. In *3rd Workshop on Synthesis (SYNT)*. Vol. 157 of *EPTCS* pp. 84–99.

Jin, HoonSang, Kavita Ravi & Fabio Somenzi. 2002. Fate and FreeWill in Error Traces. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. Joost-Pieter Katoen & Perdita Stevens. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 445–459.

Johnson, Taylor T, Diego Manzanas Lopez, Patrick Musau, Hoang-Dung Tran, Elena Botoeva, Francesco Leofante, Amir Maleki, Chelsea Sidrane, Jiameng Fan & Chao Huang. 2020. Artificial Intelligence and Neural Network Control Systems (AINNCS) for Continuous and Hybrid Systems Plants. In *7th International Workshop on Applied Verification of Continuous and Hybrid Systems*, ed. G. Frehse & M. Althoff. Vol. 74 Sydney, Australia: EasyChair pp. 107–139.

Kapinski, James, Jyotirmoy V. Deshmukh, Xiaoqing Jin, Hisahiro Ito & Ken Butts. 2016. "Simulation-Based Approaches for Verification of Embedded Control Systems: An Overview of Traditional and Advanced Modeling, Testing, and Verification Techniques." *IEEE Control Systems Magazine* 36(6):45–64.

Karimi, Abolfazl, Manish Goyal & Parasara Sridhar Duggirala. 2021. "Safety and progress proofs for a reactive planner and controller for autonomous driving." *CoRR* abs/2107.05815.

Khatri, S.P., A. Narayan, S.C. Krishnan, K.L. McMillan, R.K. Brayton & A. Sangi. 1996. Engineering change in a non-deterministic FSM setting. In *33rd Design Automation Conference Proceedings, 1996*. pp. 451–456.

Koymans, Ron. 1990. "Specifying real-time properties with metric temporal logic." *Real-time systems* 2(4):255–299.

Kupferman, Orna & Moshe Vardi. 2000. Vacuity Detection in Temporal Model Checking. In *International Journal on Software Tools for Technology Transfer (STTT)*. Vol. 4.

Levine, Sergey, Peter Pastor, Alex Krizhevsky, Julian Ibarz & Deirdre Quillen. 2018. "Learning hand-eye coordination for robotic grasping with deep learning and large-scale data collection." *The International Journal of Robotics Research* 37(4-5):421–436.

Lewis, F. L., A. Yesildirak & Suresh Jagannathan. 1998. *Neural Network Control of Robot Manipulators and Nonlinear Systems*. USA: CRC Press.

Lin, Hai & Panos J Antsaklis. 2009. "Stability and stabilizability of switched linear systems: a survey of recent results." *IEEE Transactions on Automatic control* 54(2):308–322.

Lindenbaum, Michael, Shaul Markovitch & Dmitry Rusakov. 1999. Selective Sampling for Nearest Neighbor Classifiers. In *Proceedings of the Sixteenth National Conference on Artificial Intelligence and the Eleventh Innovative Applications of Artificial Intelligence Conference Innovative Applications of Artificial Intelligence*. AAAI '99/IAAI '99 USA: American Association for Artificial Intelligence p. 366–371.

Long, Zichao, Yiping Lu, Xianzhong Ma & Bin Dong. 2018. PDE-Net: Learning PDEs from Data. In *Proceedings of the 35th International Conference on Machine Learning*, ed. Jennifer Dy & Andreas Krause. Vol. 80 Stockholm, Sweden: PMLR pp. 3208–3216.

Lopez, Diego Manzanas, Patrick Musau, Hoang-Dung Tran & Taylor T. Johnson. 2019. Verification of Closed-loop Systems with Neural Network Controllers. In *ARCH19. 6th International Workshop on Applied Verification of Continuous and Hybrid Systems*, ed. Goran Frehse & Matthias Althoff. Vol. 61 of *EPiC Series in Computing* Montreal, Canada: EasyChair pp. 201–210.

Maler, Oded, Dejan Nickovic & Amir Pnueli. 2008. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*. Springer pp. 475–505.

Matouek, Jirí & Bernd Gärtner. 2006. *Understanding and Using Linear Programming*. Springer Berlin, Heidelberg.

Meng, Yue, Dawei Sun, Zeng Qiu, Md Tawhid Bin Waez & Chuchu Fan. 2021. Learning Density Distribution of Reachable States for Autonomous Systems. In *5th Annual Conference on Robot Learning*.

Miller, W. Thomas, Richard S. Sutton & Paul J. Werbos. 1991. *Neural Networks for Control*. Cambridge, MA, USA: The MIT Press.

Moore, Kevin L. 2012. *Iterative learning control for deterministic systems*. London: Springer Science & Business Media.

Narendra, Kumpati S & Jeyendran Balakrishnan. 1994. "A common Lyapunov function for stable LTI systems with commuting A-matrices." *IEEE Transactions on automatic control* 39(12):2469–2471.

Nemitz, Catherine E., Tanya Amert, Manish Goyal & James H. Anderson. 2019. Concurrency Groups: A New Way to Look at Real-Time Multiprocessor Lock Nesting. In *Proceedings of the 27th International Conference on Real-Time Networks and Systems*. RTNS '19 New York, NY, USA: Association for Computing Machinery p. 187–197.

Nghiem, Truong, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta & George J. Pappas. 2010. 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)*. ACM.

Nguyen, Luan Viet & Taylor T Johnson. 2015. Benchmark: DC-to-DC Switched-Mode Power Converters (Buck Converters, Boost Converters, and Buck-Boost Converters). In *ARCH14-15. 1st and 2nd*

International Workshop on Applied veRification for Continuous and Hybrid Systems, ed. Goran Frehse & Matthias Althoff. Vol. 34 of *EPiC Series in Computing* EasyChair pp. 19–24.

Papachristodoulou, A., J. Anderson, G. Valmorbida, S. Prajna, P. Seiler & P. A. Parrilo. 2013. *SOSTOOLS: Sum of squares optimization toolbox for MATLAB*.
URL: *http://arxiv.org/abs/1310.4716*

Phan, Dung, Nicola Paoletti, Timothy Zhang, Radu Grosu, Scott A. Smolka & Scott D. Stoller. 2018. Neural State Classification for Hybrid Systems. In *Automated Technology for Verification and Analysis*, ed. Shuvendu K. Lahiri & Chao Wang. Cham: Springer International Publishing pp. 422–440.

Prabhakar, Pavithra, Parasara Sridhar Duggirala, Sayan Mitra & Mahesh Viswanathan. 2013. Hybrid automata-based cegar for rectangular hybrid systems. In *Verification, Model Checking, and Abstract Interpretation*. Springer pp. 48–67.

Raissi, Maziar, Paris Perdikaris & George Em Karniadakis. 2018. "Multistep Neural Networks for Data-driven Discovery of Nonlinear Dynamical Systems.".

Raman, Vasumathi, Alexandre Donzé, Dorsa Sadigh, Richard M Murray & Sanjit A Seshia. 2015. Reactive synthesis from signal temporal logic specifications. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. ACM pp. 239–248.

Ratschan, Stefan & Zhikun She. 2005. Safety Verification of Hybrid Systems by Constraint Propagation Based Abstraction Refinement. In *Hybrid Systems: Computation and Control*, ed. Manfred Morari & Lothar Thiele. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 573–589.

Ravanbakhsh, Hadi & Sriram Sankaranarayanan. 2019. "Learning control lyapunov functions from counterexamples and demonstrations." *Autonomous Robots* 43:275–307.

Rober, Nicholas, Michael Everett & Jonathan P. How. 2022. "Backward Reachability Analysis for Neural Feedback Loops.".

Roehm, Hendrik, Jens Oehlerking, Thomas Heinz & Matthias Althoff. 2016. STL Model Checking of Continuous and Hybrid Systems. In *Automated Technology for Verification and Analysis*, ed. Cyrille Artho, Axel Legay & Doron Peled. Cham: Springer International Publishing pp. 412–427.

Sadraddini, Sadra & Calin Belta. 2016. Feasibility envelopes for metric temporal logic specifications. In *2016 IEEE 55th Conference on Decision and Control (CDC)*. pp. 5732–5737.

Sankaranarayanan, Sriram & Georgios E. Fainekos. 2012. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *Hybrid Systems: Computation and Control (part of CPS Week 2012), HSCC'12, Beijing, China, April 17-19, 2012*. pp. 125–134.

Sansare, Sameera, Tibo Duran, Hossein Mohammadiarani, Manish Goyal, Gowtham Yenduri, Antonio Costa, Xiaoming Xu, Thomas O'Connor, Diane Burgess & Bodhisattwa Chaudhuri. 2021. "Artificial neural networks in tandem with molecular descriptors as predictive tools for continuous liposome manufacturing." *International Journal of Pharmaceutics* 603:120713.

Singh, Nikhil Kumar & Indranil Saha. 2020. "Specification-Guided Automated Debugging of CPS Models." *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems* 39(11):4142–4153.

Singh, Nikhil Kumar & Indranil Saha. 2021. "Specification Guided Automated Synthesis of Feedback Controllers." *ACM Trans. Embed. Comput. Syst.* 20(5s).

Sloth, Christoffer & Rafael Wisniewski. 2011. "Verification of Continuous Dynamical Systems by Timed Automata." *Form. Methods Syst. Des.* 39(1):47–82.

Solar-Lezama, Armando, Liviu Tancau, Rastislav Bodík, Sanjit A. Seshia & Vijay A. Saraswat. 2006. Combinatorial sketching for finite programs. In *Proceedings of the 12th International Conference on Architectural Support for Programming Languages and Operating Systems, ASPLOS, San Jose, CA, USA*. pp. 404–415.

Sotoudeh, Matthew & Aditya V. Thakur. 2021. SyReNN: A Tool for Analyzing Deep Neural Networks. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. Jan Friso Groote & Kim Guldstrand Larsen. Cham: Springer International Publishing pp. 281–302.

Sun, Dawei & Sayan Mitra. 2022. NeuReach: Learning Reachability Functions from Simulations. In *Tools and Algorithms for the Construction and Analysis of Systems*, ed. Fisman Dana & Rosu Grigore. Munich: Springer pp. 222–337.

Sun, Xiaowu, Haitham Khedr & Yasser Shoukry. 2019. Formal Verification of Neural Network Controlled Autonomous Systems. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. New York, USA: ACM p. 147–156.

Sun, Youcheng, Xiaowei Huang, Daniel Kroening, James Sharp, Matthew Hill & Rob Ashmore. 2019. "Structural Test Coverage Criteria for Deep Neural Networks." *ACM Trans. Embed. Comput. Syst.* 18(5s).

Sutton, Richard S & Andrew G Barto. 2018. *Reinforcement learning: An introduction*. MIT press.

Tanaka, Kazuo, Tsuyoshi Hori & Hua O Wang. 2003. "A multiple Lyapunov function approach to stabilization of fuzzy control systems." *IEEE Transactions on fuzzy systems* 11(4):582–589.

Tani, Seiichiro, Kiyoharu Hamaguchi & Shuzo Yajima. 1993. The complexity of the optimal variable ordering problems of shared binary decision diagrams. In *Algorithms and Computation*, ed. K. W. Ng, P. Raghavan, N. V. Balasubramanian & F. Y. L. Chin. Berlin, Heidelberg: Springer Berlin Heidelberg pp. 389–398.

Thati, Prasanna & Grigore Roşu. 2005. "Monitoring Algorithms for Metric Temporal Logic Specifications." *Electronic Notes in Theoretical Computer Science* 113:145–162. Proceedings of the Fourth Workshop on Runtime Verification (RV 2004).

Tiwari, Ashish. 2003. Approximate reachability for linear systems. In *International Workshop on Hybrid Systems: Computation and Control*. Springer pp. 514–525.

Tjeng, Vincent, Kai Y. Xiao & Russ Tedrake. 2019. Evaluating Robustness of Neural Networks with Mixed Integer Programming. In *International Conference on Learning Representations*. LA, USA: OpenReview.net.

Tran, Hoang-Dung, Feiyang Cai, Manzanas Lopez Diego, Patrick Musau, Taylor T. Johnson & Xenofon Koutsoukos. 2019. "Safety Verification of Cyber-Physical Systems with Reinforcement Learning Control." *ACM Transaction on Embedded Computing Systems* 18(5s).

Vidnerová, Petra. 2019. "RBF-Keras: an RBF Layer for Keras Library." `https://github.com/PetraVidnerova/rbf_keras/`.

Wegener, Ingo. 2000. *Branching Programs and Binary Decision Diagrams*. Society for Industrial and Applied Mathematics.

Xiang, Weiming, Hoang-Dung Tran, Xiaodong Yang & Taylor T. Johnson. 2021. "Reachable Set Estimation for Neural Network Control Systems: A Simulation-Guided Approach." *IEEE Transactions on Neural Networks and Learning Systems* 32(5):1821–1830.

Xie, Yulai, Jack Snoeyink & Jinhui Xu. 2006. Efficient Algorithm for Approximating Maximum Inscribed Sphere in High Dimensional Polytope. In *Proceedings of the Twenty-second Annual Symposium on Computational Geometry*. SCG '06 New York, NY, USA: ACM pp. 21–29.

Zeller, Andreas. 1999. Yesterday, My Program Worked. Today, It Does Not. Why? In *Proceedings of the 7th European Software Engineering Conference Held Jointly with the 7th ACM SIGSOFT International Symposium on Foundations of Software Engineering*. ESEC/FSE-7 Berlin, Heidelberg: Springer-Verlag p. 253–267.

Zhao, Hengjun, Xia Zeng, Taolue Chen & Zhiming Liu. 2020. Synthesizing Barrier Certificates Using Neural Networks. In *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*. New York, USA: Association for Computing Machinery pp. 1–11.

Zutshi, Aditya, Jyotirmoy V Deshmukh, Sriram Sankaranarayanan & James Kapinski. 2014. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*. ACM p. 5.

Zutshi, Aditya, Sriram Sankaranarayanan, Jyotirmoy V. Deshmukh, James Kapinski & Xiaoqing Jin. 2015. Falsification of Safety Properties for Closed Loop Control Systems. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. HSCC '15 New York, NY, USA: Association for Computing Machinery p. 299–300.