Title
Oracle-Guided Design and Analysis of Learning-Based Cyber-Physical Systems

Author
Ghosh, Shromona

Publication Date
2019

Peer reviewed|Thesis/dissertation

Oracle-Guided Design and Analysis of Learning-Based Cyber-Physical Systems

by

Shromona Ghosh

A dissertation submitted in partial satisfaction of the

requirements for the degree of

Doctor of Philosophy

in

Engineering - Electrical Engineering and Computer Science

in the

Graduate Division

of the

University of California, Berkeley

Committee in charge:

Professor Alberto Sangiovanni-Vincentelli, Co-chair
Professor Sanjit A. Seshia, Co-chair
Professor Claire J. Tomlin
Professor Francesco Borelli

Fall 2019

Oracle-Guided Design and Analysis of Learning-Based Cyber-Physical Systems

Abstract

Oracle-Guided Design and Analysis of Learning-Based Cyber-Physical Systems

by

Shromona Ghosh

Doctor of Philosophy in Engineering - Electrical Engineering and Computer Science

University of California, Berkeley

Professor Alberto Sangiovanni-Vincentelli, Co-chair

Professor Sanjit A. Seshia, Co-chair

We are in world where autonomous systems, such as self-driving cars, surgical robots, robotic manipulators are becoming a reality. Such systems are considered *safety-critical* since they interact with humans on a regular basis. Hence, before such systems can be integrated into our day to day life, we need to guarantee their safety. Recent success in machine learning (ML) and artificial intelligence (AI) has led to an increase in their use in real world robotic systems. For example, complex perception modules in self-driving cars and deep reinforcement learning controllers in robotic manipulators. Although powerful, they introduce an additional level of complexity when it comes to the formal analysis of autonomous systems. In this thesis, such systems are designated as Learning-Based Cyber-Physical Systems (LB-CPS).

In this thesis, we take inspiration from the Oracle-Guided Inductive Synthesis (OGIS) paradigm to develop frameworks which can aid in achieving formal guarantees in different stages of an autonomous system design and analysis pipeline. Furthermore, we show that to guarantee the safety of LB-CPS, the design (synthesis) and analysis (verification) must consider feedback from the other. We consider five important parts of the design and analysis process and show a strong coupling among them, namely (i) Robust Control Synthesis from High Level Safety Specifications; (ii) Diagnosis and Repair of Safety Requirements for Control Synthesis; (iii) Counter-example Guided Data Augmentation for training high-accuracy ML models; (iv) Simulation-Guided Falsification and Verification against Adversarial Environments; and (v) Bridging Model and Real-World Gap. Finally, we introduce a software toolkit VerifAI for the design and analysis of AI based systems, which was developed to provide a common formal platform to implement design and analysis frameworks for LB-CPS.

To ma and baba.

Contents

List of Figures

List of Tables

Acknowledgments

The last six and a half years would not have been possible without the support of a large number of people in my life. I owe each and everyone of them immense gratitude and love for being with me every step of the way.

First and foremost, I would like to thank my advisors Sanjit A. Seshia and Alberto Sangiovanni-Vincentelli. I cannot complete this list without Claire J. Tomlin, who has for all purposes been an unofficial advisor. I have been very fortunate to have had three pioneers as my advisors who have each helped shape a different side of this thesis. Alberto, who trusted me enough to take me in as a student when Robert Brayton (Bob) first suggested I work with him. He has given me an immense amount of independence and shown me a lot of trust when it came to my research. It is because of him that I always ask myself before doing anything, what is the bigger picture and how will anything I do achieve it. Sanjit, who has had such a strong influence in my PhD since my first year. Since my first semester, he has provided me unique insights and advise in all my projects. He was always willing to sit down and work out the rough edges in my research. He also pushed me to step outside my comfort zone and try different things while always helping me come back when I strayed too much. Claire has been an inspiration for me. Although I started working with her in my third year, I felt that was three years too late. With her I could discuss even the silliest of ideas I had. She always encouraged me to try everything before making a decision. It was comforting and extremely important for me to know that I had that unconditional support from her. I would also like to express my gratitude to Francesco Borrelli for being part of my qualifying exam and thesis committee. Without your guidance and push to find an application for my research, I would have never found my love for self-driving cars. I would also like to thank Anca Dragan for always being available to discuss new ideas and ventures for my work in the domain of human-robot interaction. Although we never got around to collaborating during my time at Berkeley, I hope to get a chance to work with her in the future.

I would also like to thank Bob for accepting me as a student and helping me transition to Alberto's group when I decided I wanted to work in a different area. Without you, I would not be here. I still remember the day Jaijeet Roychowdhury interviewed me in Bangalore for a Berkeley PhD, I am deeply indebted to you.

I would like to express a special thanks to George Pappas and Rajeev Alur, who have always kept in touch with my research and provided me very useful advise whenever they can. My one regret is that I have never had a chance to formally work with them.

A huge thank you to Shirley Salanio for being available at all times of the day to answer all my administrative questions and just chatting with me about life in general. Thanks for your patience and always going that extra step to sort out all my administrative messes. A special thanks to Jessica Gamble and Mary Stewart who have made day to day life in Berkeley seamless.

Throughout my PhD I have been very fortunate to intern with leading pioneers in industry. For my time at Microsoft Research (MSR), I owe my gratitude to Ashish Kapoor And Shaz Qadeer. Ashish been a constant source of motivation and advise throughout the last

couple of years. Shaz has been that person who pushed me to ask the difficult questions, with out which my research would always fall short. Finally, I'd also like to thank Gireeja Ranade when during her time at MSR was a mentor both professionally and personally. For my time is SRI, I owe my gratitude to Susmit Jha and Nataranjan Shankar. Susmit was really willing to get down to nitty-gritty details and had immense amount of patience to sit through the long math sessions with me. Shankar really bought perspective to my work. He'd always find a way to relate my work to other fields, which really pushed me to explore.

I have also been so fortunate to collaborate with great many people during my time at Berkeley. This thesis would not have been possible without Tommaso Dreossi, who has been my longest collaborator and am fortunate enough to call a close friend; Somil Bansal, a friend who grew to be an amazing collaborator; Marcell Vazquez-Chanlatte, who taught me how to be practical in my research; Dorsa Sadigh, who pushed me when I needed to and taught me the perfect work life balance. I would also like to thank all my other co-authors and collaborators, specially people who helped with the work of this dissertation: Vasumathi Raman, Alexandre Donze, Pierluigi Nuzzo, Shankar Sastry, Xiangyu Yue, Felix Berkenkamp, Daniel Fremont, Kurt Keutzer, Hadi Ravanbakhsh and Edward Kim.

I owe a huge thanks to Antonio Iannopollo, who I met on my first day at Berkeley and has been a constant in my PhD since that day, till the day I left Berkeley. I thank you for your friendship and your support in every step of the way. To listening to me vent, to offering useful advise and now helping me with administrative stuff, I cannot express how fortunate I am to have you in my life. My life in the DOP center would not be complete without Inigo Incer and Baihong Jin, two great friends I have shared numerous conversations with and who are willing to run around to help me even today. Finally, to Marten Lohstroh, Fabio Cremona, Eric Kim, Sylvia Herbert, Anayo Akametalu and Jaime Fernandez who have been very supportive and great friends over the last 6 years. Thank you to Branden Ghena and Joshua Adkins for being amazing co-GSIs in EECS 149: Introduction to Embedded Systems.

Everyone says having a good support system outside of work is very important, and I have had one of the best. Words are not enough to express my gratitude for Anurag Khandelwal, without whom my last six years would have been impossible to get by, who has supported me through every decision and every difficult time. I am thankful to have had you in my life. A huge thanks to Aishwarya Parasuram, Moitrayee Bhattacharyya, Tathagata Das, Nitesh Mor for being my constants the last 6 years, and being the ones I turn to at every step. To Lauren Iannopollo and Rusi-Ko Mchedlishvili for the amazing get togethers and for accepting me as one of your own. To James Devine and Alyssa Morrow for making my time at MSR memorable and amazing. To Sreeta Gorripaty, Nikunj Bajaj, Radhika Mittal, Saurabh Gupta, Shubham Tulsiani, Bharat Hariharan, Neeraja Yadwadkar and Vivek Chawda for your great friendships at Berkeley. To Madhumitha Sridhara, for believing in me and pushing me to apply for Berkeley, to motivating me to stay when I wanted to quit and live streaming my graduation just to watch me graduate; I would not be here without you. To Aditi Vutukuri, Priyanka Ganapathi, Sneha Abhyankar and Kritika Mehta; who will always go that extra mile, however far just to make sure I are okay, thank you for supporting me even at the worse of times.

Finally, I owe everything I am today to my constants throughout my life, my ma Ruby Ghosh and my baba Sankar Prasad Ghosh. For sacrificing so much for me and never letting me feel I missed out on anything. For taking care of all my whims and motivating me when I'm down. For being my number one fan and celebrating the happy moments and grieving the sad ones with me, for always being there. It is now my turn to take care of you.

I dedicate this thesis to each and everyone of you and to all the other people I have had the good fortune to meet along the way. To the ones who have brought sunshine to my life, you make me happy when the skies are gray. You have all impacted my life for the better.

Chapter 1

Introduction

1.1 Motivation

Today, we are in an era where autonomous systems are becoming a reality. Hence, much of the recent research in robotics and control theory has focused on developing complex autonomous systems such as robotic manipulators, autonomous vehicles and surgical robots. Since such systems are expected to interact and share the world with humans, we consider them to be *safety-critical*. Before such systems can be deployed into the real world it is important to design and analyze systems to satisfy safety objectives or guarantees.

In general, the design pipeline for Cyber-Physical Systems (CPS) has two highly interacting components (i) synthesis or design of the overall system; and (ii) verification or analysis of the system. Correct-by-construction (Figure 1.1) design has emerged as a new design paradigm for synthesis. In this framework, the high level system safety requirement is directly incorporated into the synthesis process, which guarantees that the resulting system satisfies the safety requirements. To be able to achieve this, the system designer must first be able to express (or design) the safety specification (requirement); and then develop synthesis algorithms which are capable of incorporating the specification in the design. To be able to achieve this, we often make many simplifying assumptions; such as simplified system models and environments. To counteract this, we follow synthesis with verification. The goal of verification is to mathematically prove that the designed system indeed satisfies the high level safety specification. The complexity of verification process is highly dependent on the synthesized module; which limits its usability to simple systems. Falsification/testing has risen as a more general and scale-able analysis technique, where the one searches for system behaviors which falsify the safety specification. Independent of the actual algorithms or frameworks used for the design and analysis, a common consensus in the design of CPS, is the tight coupling between synthesis and verification. It is important to go through multiple rounds of the each before we can consider the design a success.

Recent successes in machine learning (ML) and artificial intelligence (AI) have motivated an increased use of such techniques in the design of complex perception modules and

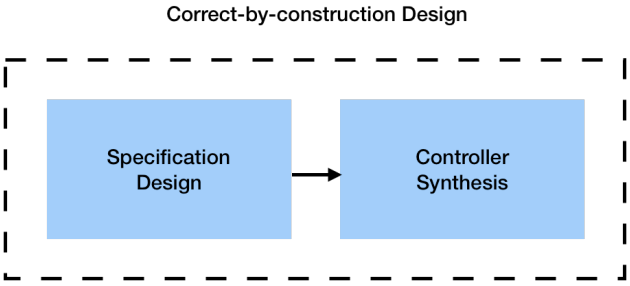


Figure 1.1. Correct-by-construction design

.

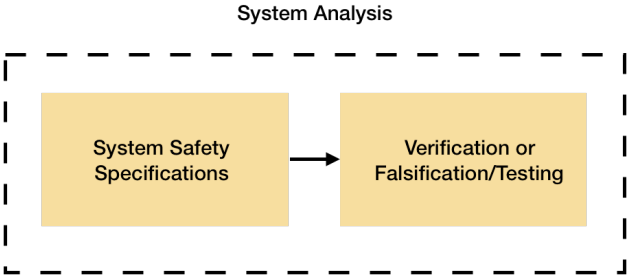


Figure 1.2. System Analysis and Verification

.

controllers for to achieve complex tasks for CPS systems.

In real world robotic systems, ML based techniques have shown to far outperform classical computer vision techniques for designing high fidelity perception modules. Models produced by machine learning algorithms, especially *deep neural networks* (NN), are being deployed in domains where trustworthiness is a big concern, creating the need for higher accuracy and assurance [130, 135]. However, learning high-accuracy models using deep learning is limited by the need for large amounts of data, and, even further, by the need of labor-intensive labeling. Hence, designing formal frameworks that can analyze and automatically generate data that can be used for re-training is of vital importance.

To achieve a rich set of maneuvers in complex robotic systems, reinforcement learning (RL) [149], optimal control (OC) [146] and model predictive control (MPC) [112] techniques have been developed. RL based techniques have shown to achieve a range of complex maneuvers like flying a quadrotor [83] to complex tasks on robots [96]. While these controllers can handle some degree of uncertainty [115, 125], and have been successful in synthesizing high fidelity controllers; they fail to provide any formal guarantee of safety and merely measure performance in expectation. On the other hand, model based control synthesis techniques like reachability [38, 109] and MPC [112, 127, 128] provide strong safety guarantees but often make simplifying assumptions on the system. Hence, there is a need to develop formal frameworks to provide similar safety guarantees for RL based controllers. Moreover, even while using a model for synthesis, it is not clear what safety specifications

we need to consider during the synthesis process.

With an increased use of learning in the design pipeline, we are presented with new challenges while designing synthesis and verification frameworks which now need to consider the added complexity introduced by learning. Today's systems are composed of complex ML modules which interact with the controller and the physical system. Monolithic synthesis algorithms that consider the synthesis of the entire system cannot simultaneously synthesize all the components. Even if one could define the correctness of the overall system, to ensure correctness of each sub-component, we must be able to define the correctness of each component. Mathematically capturing/defining the correctness or ML components is hard, for e.g., how can one define the correctness of vision system. In spite of the recent successes in designing high fidelity vision systems in the ML community, there correctness is not guaranteed. As a result, there is a large body of work which focuses on the (robustness) analysis of such systems [46]. During controller design, one must now synthesize controllers which have to be robust to the errors of the ML modules. In [135], the authors have done a comprehensive analysis of challenges introduced by ML components in formal verification. To start, one needs to capture the formal correctness of such systems. Then one must be able to mathematically define the the environment or the domain of inputs for them. Finally, one must design computational engines which are are to reason about the correctness of such system, this generally involves a search over the entire input space.

This has motivated us to study and develop frameworks that can be used to design and analyze high fidelity robotic systems composed of complex controllers and ML components interacting with the physical system.

1.2 Thesis Approach

In this thesis, we develop formal design and analysis frameworks that can be used in various parts of the design and analysis pipeline to improve the safety and hence, design high fidelity robotic systems with formal safety guarantees. In particular, we recognize five important parts of the pipeline (detailed in Section 1.3) and propose frameworks that can used to design them taking inspiration from control theory and formal verification. The proposed frameworks are inspired by the oracle-guided inductive synthesis (OGIS) framework introduced in [80]. An instance of OGIS $\mathcal{I}$ consists of a learner $\mathcal{L}$ and an oracle $\mathcal{O}$ (Figure 1.3). The learner attempts to learn or synthesize a concept (e.g. control policy, deep learning network, verification proof) by querying an oracle (e.g. verifier, optimization engine, simulation engine). The framework does not know the correct concept a-priori and tries to learn it by minimizing the number of queries made to the oracle. Hence, it makes it particularly attractive for design and analysis of robotic systems by modeling or learning the smallest concept necessary without knowledge of the overall system or its sub-components. Moreover, the OGIS framework helps us provide strong safety guarantees based on the learner and oracle we choose. In some chapters we use a specific instance of the OGIS framework, Counter-Example Guided Inductive Synthesis (CEGIS). In each chapter, we show how an instance of

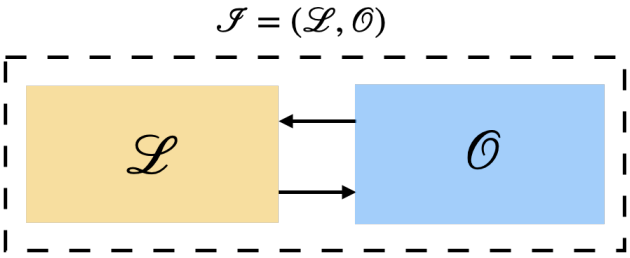


Figure 1.3. OGIS framework $\mathcal{I} = (\mathcal{L}, \mathcal{O})$

.

the OGIS framework helps us decouple a complex design problem to simpler building blocks which can be reduced to designing a simple learner (or oracle) and interaction between them, for e.g., for robust controller synthesis (Figure 3.2).

1.3 Thesis Contribution

In this thesis, we take five parts of the design and analysis of CPS systems pipeline and reformulate each as an instance of a OGIS framework and detail the learner and oracle design and the interaction between them.

For correct-by-construction design, we consider the drawbacks of the current framework (Figure 1.1 and propose an unified framework that introduces feedback between robust controller synthesis (Chapter 3) and specification design (Chapter 4). As outlined in Figure 1.1, correct-by-construction synthesis requires designing the specification and the synthesis algorithm. In general it is hard for a designer to design a specification for a system which is synthesizable without understanding the synthesis algorithm. Hence, there is a tight coupling with the specification and synthesis process. While Figure 1.1 shows a one way flow of information from specification design to synthesis, we propose that the specification design and the synthesis process are dependent on each other. One can refine (or repair a specification) based on the results of synthesis. In return, one has to utilize the updated specification and environment models for consecutive rounds of synthesis. In this thesis, we propose there needs to be a strong feedback between these two parts of the pipeline. In Chapter 3, we study the synthesis of robust control strategies from high level specifications [155]. Today the requirements for robotic systems rely not only on safety requirements but also aim to fulfill performance (liveness) requirements. Moreover, to account for modeling error or interaction with other agents, our synthesis process must be robust to environment disturbances. Rather than designing controllers and verifying after the fact that they satisfy high level requirements, there has been a paradigm shift towards correct-by-construction controller synthesis. This has been commonly observed in optimal control [109] for safety requirements. Recently, synthesis tools like TuLiP [159] and LtLMoP [73] have been developed to synthesize controllers from a high level temporal logic specification in non-adversarial settings. In this thesis, we look into the problem of synthesizing robust controllers in adversarial settings

from high-level temporal specifications. To achieve this, we draw inspiration from [128] and formulate the synthesis problem as a game. However, the solutions proposed in literature are infeasible to implement in practice. In this chapter, we reformulate the game in the OGIS framework, where the interaction between the two players is captured as an interaction between the learner and the oracle. We provide approximate but sound implementations for the oracle and learner, and use that to provide correctness guarantees for the overall synthesis procedure. In Chapter 4, we look at the problem of design (through diagnosis and repair) of specifications for controller synthesis [64]. To synthesize or verify controllers, one needs to first mathematically capture the requirements of the system. However, specification design is a hard problem even for the more experienced of designers. Recently, several controller synthesis methods have been proposed for expressive temporal logics and a variety of system dynamics. However, a major challenge to the adoption of these methods in practice is the difficulty of writing the requisite formal specifications before hand. Specifications that are poorly stated, incomplete, or inconsistent can produce synthesis problems that are unrealizable (no controller exists for the provided specification), intractable (synthesis is computationally too hard), or lead to solutions that fail to capture the designer's intent. In this chapter, we reformulate the diagnosis and repair of specifications into an interaction between the learner who repairs the specification and the oracle who diagnoses the specification. We prove that the proposed frameworks and algorithms minimally modify or repair specifications for infeasible controller synthesis problems.For the special case of specifications involving environment assumptions, we show that the oracle relies on synthesis engine which can compute robust controllers. As we showed in Chapter 3, we can build such a synthesis engine by instantiating an OGIS instance which solves a game. As a result, we develop a hierarchical OGIS framework to diagnose and repair environment assumptions by using the frameworks presented in Chapter 3. This further exemplifies the tight co-relation between the specification design and the synthesis algorithms.

While designing high fidelity autonomous systems, the overall safety of the system is highly dependent on interaction between the designed controller and other components in the system. In most real world autonomous systems, these components are ML or AI based perception or decision modules. Such modules are used in conjunction to controllers to percieve or sense the environment around, e.g., perception modules, and rely on rich models like neural network. Hence, one needs to ensure that we can build these modules to correct to guarantee the correctness or safety of the overall system. To train deep neural networks to provide high accuracy results, we need rich training sets with large amounts of data and labor-intensive labeling. If the dataset is not representative of the environments in which the model is expected to operate, then the trained model would perform very poorly. A key issue while choosing the training data set, is being able to decide of the diversity of the input. One cannot a-priori detect what inputs are not being represented in the training set. So one has to analyze (or test) the model to detect where the network is failing. In this thesis, we show that synthesizing rich data sets requires a model analysis (or verification) step in the loop and propose a framework that uses a tight coupling between model synthesis and falsification to design data sets for training. *Data augmentation* overcomes the lack of data

by inflating training sets with label-preserving transformations, i.e., transformations which do not alter the label. Traditional data augmentation schemes [51, 138, 29, 28, 91] involve geometric transformations which alter the geometry of the image (e.g., rotation, scaling, cropping or flipping); and photometric transformations which vary color channels. However, these schemes add data without taking into account what kind of features the model has already learned. To overcome this, in Chapter 5, we propose a counter-example guided data augmentation [46] that analyzes the network to find images where it performs poorly and augments the training set. This ensures that we now populate the training set with images which the network could not capture originally and provide high-level explanations of network failure. In this chapter, we formulate the overall training and data set design into an interaction between a learner which is responsible for training the ML module and an oracle which analyzes the the trained model to find counterexamples which are then added to the training data set. Hence, the learner here is a synthesizer and the oracle is falsifier. We show that by going over this loop multiple times, we are able to synthesize high accuracy image detection models using deep learning.

When one does not consider uncertainty in the controller synthesis process or relies on RL for synthesizing controllers, we generally have to perform a verification and analysis step after the synthesis. Even for model based control synthesis techniques, since we make simplifying assumptions during synthesis, we might have to verify that the synthesized control is robust to errors in the actual system. However, formal verification requires that the system, environment and specification be formally defined. For complex dynamical systems and ML based controllers, this is often hard to do. To this end, simulation guided falsification has been suggested to find failure cases of such controllers [50, 8, 39, 163, 37]. However, these techniques are far from providing verification guarantees. In Chapter 6, we look into simulation-based falsification for systems with learning based components (like controllers) which have some smoothness properties. In this chapter, we formulate simulation-guided falsification as in interaction between a learner which attempts to learn the cause of the failure using Gaussian Processes (GP) and an oracle which uses Bayesian Optimization (BO) to search for likely counterexamples to simulate on the system. While this scheme has shown to find counterexamples more quickly, we also study the assumptions and conditions of the system under which, we can provide probabilistic verification guarantees even when the underlying system is unknown.

In Chapters 3, 4, 5 and 6, we study design and analysis in a purely model or simulation world. Most of the work in verification and synthesis rely on simpler models or simulations of the real system. Even if we are able to formally synthesize or verify that these systems satisfy our safety requirements, we arefar from proving their correctness on the actual physical system. This mismatch between the model world and the real world occurs because we consider simplified models. Even if we have access to a high fidelity simulator of the actual system, we may still miss out certain environmental effects for e.g., wear and tear, sensor and actuator noise. Hence it is important to develop a framework which decides under what circumstances the analysis of the model holds for the real system. In Chapter 7, we study a specific problem of how we transfer model level verification guarantees to the actual system

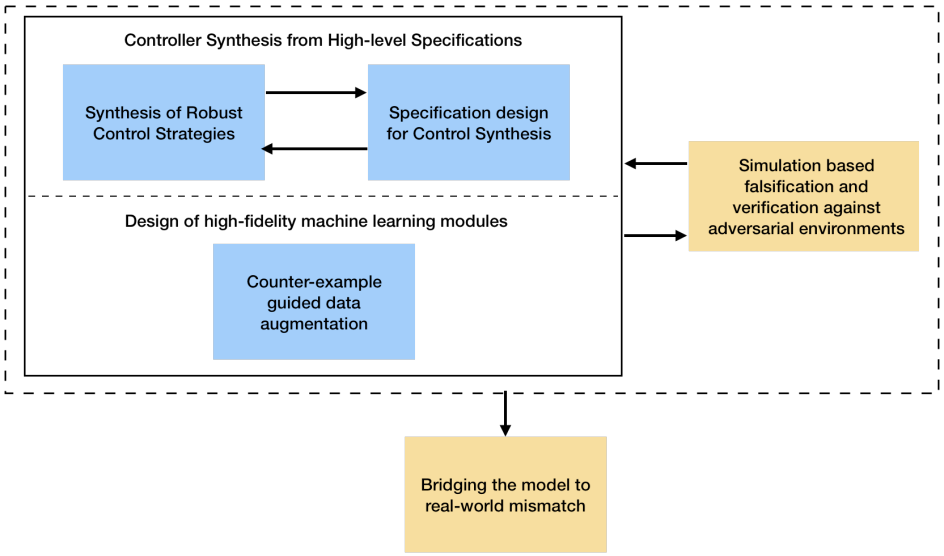


Figure 1.4. Thesis contribution in overall CPS design.

.

for reach-avoid control problems. To this end, we define a specification-centric simulation metric SPEC by taking inspiration from the simulation metric [5, 66, 9] that captures the mismatch between the model and the system. Unlike the simulation metric, SPEC considers the underlying safety specification, and hence, less conservative. Like the simulation metric, SPEC retains the necessary properties which can be used to synthesize robust controllers using the model which is guaranteed to be safe for the system. In this chapter, we formulate the synthesis of safe controllers for actual system as in interaction between a learner which synthesizes (proposes) controllers for the model using SPEC and an oracle which attempts to verify the controller on the actual system and computes SPEC. We further propose a sampling based technique to compute SPEC which can be formulated as an instance of OGIS. Hence, the overall framework becomes an instance of hierarchical OGIS.

The stages mentioned above are important parts of an autonomous system design pipeline. While each may tackle a different facet of the design process, to ensure the safety and correctness of the overall design, we need to be able to provide strong safety guarantees at each stage. Moreover, we need would like to expose the vulnerability of each process to the other, so it can be considered which designing the next step. This leads to a tight coupling (and feedback) among the design and analysis steps. For example, if we could analyze the ML modules to realize when the models fail; they can be used in the control synthesis process for designing controllers that are robust to their failure. Figure 1.4 shows how our contributions are placed in the overall design and analysis framework. The blue boxes represent design-centric frameworks, while the yellow boxes represent analysis-centric frameworks. There is a tight couple between the stage, suggesting design is tightly coupled with analysis and vice versa.

Finally, we wrap this thesis by presenting VerifAI [49] in Chapter 8, a toolkit for the

design for the analysis of AI based systems. The toolkit incorporates many of the algorithms and frameworks described in this thesis. The toolkit takes as input an overall system as a simulator, a description of environment configurations and system level specifications. By relying on simulators for system descriptions, VerifAI can be used with very general systems which may not have well defined models. By considering system level specifications, we overcome the need to define component level mathematical specifications for ML components which are often hard. Moreover, component level requirements do not capture the interaction among components, e.g controllers and perception module. By considering system level requirements, the ML components are analyzed the context in which they are used in the system, which gives us more realistic analysis data. We have incorporated the analysis techniques from Chapters 5 and 6. Finally, we have shown results using a range of robotic simulators like OpenAI [17] and Webots [140]. The toolkit is the first to analyze CPS with ML components with system level specifications.

1.4 Thesis Outline

This thesis includes and revises content from several of my previously published papers. I gratefully acknowledge and thank my advisors, Alberto Sangiovanni-Vincentelli and Sanjit A. Seshia, who have played an important role in shaping the contributions in all these papers. Chapter 3 is based on our paper [155] which is joint work Marcell Vazquez-Chanlatte I would also like to thank Vasumathi Raman for help in developing the theory CEGIS loop for robust control synthesis. Chapter 4 revises the material from [64]. which is joint work with Dorsa Sadigh. I thank Pierluigi Nuzzo and Vasumathi Raman for help with the proofs in the chapter. Chapter 5 revises our paper [48, 47] which is joint work with Tommaso Dreossi and Xiangyu Yue. In Chapter 6 we revise the material from our paper [65]. I thank Ashish Kapoor, Shaz Qadeer and Gireeja Ranade for their guidance and advise. I thank Felix Berkenkamp for explaining the theory of Gaussian Process and Bayesian Optimization and developing the proofs with me. In Chapter 7 we extend the work from out paper [63], which is joint work with Somil Bansal. I thank Claire Tomlin for her advise and unique insights into the problem. In Chapter 8 we present a new toolkit VerifAI from our paper [49] which is joint work with Tommaso Dreossi and Daniel J. Fremont. I would like to thank Hadi Ravanbaksh, Edward Kim and Marcell Vazquez-Chanlatte for their feedback and help in developing and integrating additional functionalities into the toolkit.

Chapter 2

Mathematical Preliminaries

In this chapter we summarize the key mathematical concepts used in this thesis.

2.1 Oracle-Guided Inductive Synthesis

Oracle-Guided Inductive Synthesis (OGIS) was introduced by Jha and Seshia ([80]) as a framework that captures a family of synthesizers that operate by iteratively querying an oracle. An instance of the OGIS framework $\mathcal{I} = (\mathcal{L}, \mathcal{O})$ is defined by the tuple consisting of a learner (or synthesizer) $\mathcal{L}$ and an oracle $\mathcal{O}$. The learner $\mathcal{L}$ attempts to infer or synthesize a ''concept" or an ''artifact" from a domain of possible artifacts which satisfies a high level specification φ by iteratively querying the oracle $\mathcal{O}$ on examples selected from a domain of examples $\mathbf{E}$ or candidate concepts. This domain is problem dependent and will explained more in detail in the individual chapters. In this chapter we summarize the key notations and definitions required to setup an instance of OGIS $\mathcal{I} = (\mathcal{L}, \mathcal{O})$. For more details refer to [80].

Definition 1 (Artifact Class)**.** *An artifact (concept) class $\mathbb{C}$ is the domain of artifacts from which the learner $\mathcal{L}$ searches for (synthesizes) an artifact using the queries exchanged with the oracle $\mathcal{O}$.*

The concept class may either be specified in the original synthesis problem or arise as a result of a *structure hypothesis* that restricts the space of candidate concepts. Formally, we can imagine each concept to be a set of examples. Hence, $\mathbb{C} \subseteq 2^{\mathbf{E}}$. Depending on the specific instance, the domain of examples $\mathbf{E}$ and the concept class $\mathbb{C}$ can be finite or infinite.

We define the specification or the requirement by φ. The format of φ depends on the synthesis problem. In this thesis we focus on specifications defined over finite horizon system trajectories.

OGIS comprises of two key components: an inductive learning engine (also sometimes referred to as a ''Learner") and an oracle (also referred to as a ''Teacher"). The interaction between the learner and the oracle is in the form of a dialogue comprising queries and

responses. The oracle is defined by the types of queries that it can answer, and the properties of its responses. Synthesis is thus an iterative process: at each step, the learner formulates and sends a query to the oracle, and the oracle sends its response. The oracle may be tasked with determining whether the learner has found a correct target concept. In this case, the oracle implicitly or explicitly maintains the specification φ and can report to the learner when it has terminated with a correct concept or artifact.

The oracle $\mathcal{O}$ is defined by the type of queries it can accept. Let Q be domain of queries (input to the $\mathcal{O}$ or conversely output of the learner $\mathcal{L}$), and R be corresponding set of responses (output of the $\mathcal{O}$ or conversely the input to the learner $\mathcal{L}$). A valid dialogue d of the $\mathcal{O}$ is a query-response pair (q, r) such that $q \in Q$ and $r \in R$.A sequence of valid dialogue sequence $\mathbf{D}^*$ is sequence of valid dialogue pairs.

Definition 2. *An oracle is a (potentially non-deterministic) mapping* $\mathcal{O} : \mathbf{D}^* \times Q \to R$. *A learner is (potentially non-deterministic) mapping* $\mathcal{L} : \mathbf{D}^* \to Q \times \mathbb{C}$.

The learner observes a valid dialogue sequence $\delta \in \mathbf{D}^*$ to propose a candidate context $\in \mathbb{C}$ and a corresponding query $q \in Q$ to the oracle. The $\mathcal{O}$ observes the dialogue sequence $\delta \in \mathbf{D}^*$ and the current query q to produce a response $r \in R$.

An OGIS procedure is defined by properties of the learner and the oracle. Relevant properties of the learner include (i) its *inductive bias* that restricts its search to a particular family of concepts and a search strategy over this space, and (ii) *resource constraints*, such as finite or infinite memory. Relevant properties of the oracle include the types of queries it supports and of the responses it generates. We now discuss some common queries we use throughout this thesis:

1. *Membership query* ($q_{mem}(x)$): The learner selects an example $x \in \mathbf{E}$ and queries the oracle if the example satisfies the specification or not.

2. *Simulation query* ($q_{sim}(x)$): The learner selects an example $x \in \mathbf{E}$ and queries the oracle for a simulation behavior.

3. *Counter-example query* ($q_{ce}(c)$): The learner proposes a candidate concept $c \in \mathbb{C}$ and asks the oracle for counter-examples, i.e., $e \in c$ such that e does not specify φ. If it can't find such an example, it returns $\perp$.

4. *Verification query* ($q_{ver}(c)$): The learner proposes a candidate concept $c \in \mathbb{C}$ and asks the oracle if this concept satisfies φ.

This set of queries is not exhaustive and we discuss individual queries in the different chapters.

Formally, the learner $\mathcal{L}$ has to synthesize or learn the concept $c \in \mathbb{C}$ such that all the examples making up c satisfies the specification φ by querying the oracle $\mathcal{O}$ at different examples $x \in \mathbf{E}$ or with candidate concepts $c \in \mathbb{C}$. In each iteration, the queries made by the learner depends on the dialogue sequence $\mathbf{D}^*$ observed so far.

2.1.1 Counter-example guided Inductive Synthesis

A special case of the OGIS framework is Counter-example guided Inductive Synthesis (CEGIS) introduced in [141]. An instance of CEGIS is defined similar to OGIS, $\mathcal{C} = (\mathcal{L}, \mathcal{O})$. In CEGIS, the $\mathcal{O}$ accepts only a subset of queries, counter-example queries and positive witness queries (where the learner asks the oracle for positive examples). In this thesis, we use CEGIS frameworks with only counter-example queries. Most of the frameworks presented in the thesis are instances of CEGIS.

2.2 Hybrid Dynamical Systems

In this work we consider continuous-time hybrid dynamical system,

$$\dot{x} = f(x, u, e) \tag{2.1}$$

where $x \in \mathcal{X} \subseteq (\mathbb{R}^{n_c} \times \{0,1\}^{n_l})$ represent the hybrid (continuous and logical) states, $u \in \mathcal{U} \subseteq (\mathbb{R}^{m_c} \times \{0,1\}^{m_l})$ are the hybrid control inputs and $e \in \mathcal{E} \subseteq (\mathbb{R}^{e_c} \times \{0,1\}^{e_l})$ are the hybrid external inputs, including disturbances and other adversarial inputs from the environment. For the purposes of this work, we assume that the state of the system is fully observable. Using a sampling period $\Delta > 0$, the continuous-time system in (2.1) lends itself to the discrete-time approximation,

$$x_{t+1} = f_d(x_t, u_t, e_t) \tag{2.2}$$

where $x_t \in \mathcal{X}$, $u_t \in \mathcal{U}$ and $e_t \in \mathcal{E}$.

Given an initial state $x_0 \in \mathcal{X}$, finite horizon H control sequence $\mathbf{u} = (u_0, \ldots, u_{H-1})$ and environment (disturbance) sequence $\mathbf{e} = (e_0, \ldots, e_{H-1})$, the finite horizon trajectory (or behavior) of the system $\mathcal{S}$ modeled by the dynamics in (2.2) is uniquely expressed as $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) = \{(x_0, u_0, e_0), \ldots, (x_{H-1}, u_{H-1}, e_{H-1}\}$. We denote $\xi_{\mathcal{S}}(t; x_0, \mathbf{u}, \mathbf{e})$ the trajectory of the system $\mathcal{S}$ at time t. Moreover, let Ξ denote the set of all finite horizon trajectories of the system.

We make a further simplifying assumption that $\mathcal{U} = [-1, 1]^{n_u}$ and $\mathcal{E} = [-1, 1]^{n_e}$ where n_u and n_e are the dimensions of $\mathcal{U}$ and $\mathcal{E}$ respectively. This is not a limiting assumption, as one could always scale the dynamics to modify the control and environment domain. Further, since $u \in \mathcal{U}$, we have the finite-horizon control sequence $\mathbf{u} \in \mathcal{U}^H$. For ease of notation, we simply say $\mathbf{u} \in \mathcal{U}$ to imply $u_0, \ldots u_{H-1} \in \mathcal{U}$. Similarly, we say $\mathbf{e} \in \mathcal{E}$ to imply $e_0, \ldots, e_{H-1} \in \mathcal{E}$.

2.3 Safety Specification

We specify the safety specification by φ. They are defined on finite-length trajectories $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})$ of the system that can be obtained by rolling out the dynamics in (2.2) over the horizon. Alternatively, one can imagine φ to be set of all finite-horizon trajectories of

the system that satisfy the system level-safety specification; $\varphi \subseteq \Xi$. For example, φ can be temporal behaviors of the system properties in, e.g. Signal Temporal Logic (STL) [104].We say the system behavior satisfies the specification φ, i.e., $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \models \varphi$ if and only if $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \in \varphi$.

We further assume, we have access to the quantitative semantics of φ represented by $\rho_\varphi : \Xi \to \mathbb{R}$, such that:

$$\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \models \varphi \leftrightarrow \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})) > 0$$
$$\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \not\models \varphi \leftrightarrow \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})) < 0$$

Since the trajectory $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})$ is deterministic given $\mathbf{u}$ and $\mathbf{e}$ and a fixed x_0, we will use $\rho_\varphi(\mathbf{u}, \mathbf{e})$ instead of $\rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}))$ and $\varphi(\mathbf{u}, \mathbf{e})$ to represent as $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \models \varphi$ in the rest of the chapter.

By evaluating ρ_φ on the system behavior in a given environment e, $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})$, we can comment on the satisfaction of the system behavior and hence, the safety of the corresponding environment e. Typically, $\rho_\varphi(\xi) = 0$ is considered to be an unknown behavior and hence, we cannot comment on the satisfaction of ξ. One would have to then evaluate the boolean satisfaction by checking if the $\xi \in \varphi$. In this work, we take a pessimistic approach and consider $\rho_\varphi(\xi) = 0$ to imply unsatisfactory behavior. This allows for behaviors that are atleast $\epsilon > 0$ robust, which is a valid assumption to make while evaluating the safety of the system.

We assume that ρ_φ is Lipschitz continuous in $\mathbf{u}$, i.e., there exists a constant L_{ρ_φ} such that for all $e \in \mathcal{E}$:

$$\forall \mathbf{u}, \mathbf{u}' \in \mathcal{U} \; |\rho_\varphi(\mathbf{u}, \mathbf{e}) - \rho_\varphi(\mathbf{u}', \mathbf{e})| \leq L_{\rho_\varphi} |\mathbf{u} - \mathbf{u}'| \tag{2.3}$$

We further require ρ_φ to be allow a total ordering in the Ξ, i.e., if $\rho_\varphi(\xi_1) > \rho_\varphi(\xi_2)$ then ξ_1 is said to be more *safe* compared to ξ_2, and hence is a more desirable behavior. This allows for an ordering among the control strategies $\mathbf{u}$. If ξ_1 (ξ_2) was generated in $\mathbf{u}_1$ ($\mathbf{u}_2$), then we can say $\mathbf{u}_2$ is a more "robust" compared to $\mathbf{u}_1$. Larger values offer a higher degree of satisfaction while lower values offer a lower degree of satisfaction.

Example 1. *Consider the discrete time system with two states* $x = [y, z]^T$, $f_d = \begin{matrix} y_t + u_t + e_t \\ z_t + u_t + e_t \end{matrix}$ *where* $u_t, e_t \in [-1, 1]$. *Consider the specification "For the next 30 seconds,* $y > 2$ *implies that* z *will be less than* 4 *within two seconds". Consider the following syntactically- generated quantitative semantics over the state, following the quantitative semantics for STL defined in [104].*

$$\rho_\varphi(\mathbf{u}, \mathbf{e}) = \max_{t \in \{0, \ldots, 30\}} \left(y_t - 2, \min_{t' \in \{t, \ldots, t+3\}} (4 - z_{t'}) \right)$$

Since u_t *and* e_t *are both bounded and* ρ_φ *is smooth, substituting the dynamics equations into* ρ_φ *to get a function over* u_t *and* e_t *yields a quantitative semantics satisfying* (2.3) *and* (2.3)

2.4 Temporal Logic

Temporal logic has risen as a popular language for capturing and defining mathematical properties on trajectories. Linear temporal logic (LTL) was first introduced in [123] to capture behaviors of sequential programs. Since then LTL has been extended with to capture properties over dense time in Metric Temporal Logic (MITL) in [90]. A more popular extension of MTL used to define properties over the behaviors of dynamical systems is Signal Temporal Logic (STL).

2.4.1 Signal Temporal Logic

Signal Temporal Logic (STL) was first introduced as an extension of *Metric Temporal Logic (MTL)* to reason about the behavior of real-valued dense-time signals [104]. STL has been largely applied to specify and monitor real-time properties of hybrid systems [42]. We use the robust, quantitative interpretation for the satisfaction of a temporal formula [41, 40], as further detailed below. In this setting our safety specification φ is represented as a STL formula evaluated on the system trajectory $\xi_{\mathcal{S}}$ at some time t. We say $(\xi_{\mathcal{S}}, t) \models \varphi$ when φ evaluates to true for $\xi_{\mathcal{S}}$ at time t. We instead write $\xi_{\mathcal{S}} \models \varphi$, if $\xi_{\mathcal{S}}$ satifies φ at time 0. The atomic predicates of STL are defined by inequalities of the form $\mu(\xi_{\mathcal{S}}(t)) > 0$, where μ is some function of the trajectory $\xi_{\mathcal{S}}$ at time t. Specifically, μ is used to denote both the function of $\xi_{\mathcal{S}}(t)$ and the predicate. Any STL formula φ consists of Boolean and temporal operations on such predicates. The syntax of STL formulae is defined recursively as follows:

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \mathbf{G}_{[a,b]}\psi \mid \mathbf{F}_{[a,b]}\psi \mid \varphi\, \mathbf{U}_{[a,b]}\psi, \tag{2.4}$$

where ψ and φ are STL formulae, $\mathbf{G}$ is the *globally* operator, $\mathbf{F}$ is the *finally* operator and $\mathbf{U}$ is the *until* operator. Intuitively, $\xi_{\mathcal{S}} \models \mathbf{G}_{[a,b]}\psi$ specifies that ψ must hold for the trajectory $\xi_{\mathcal{S}}$ at all times of the given interval, $t \in [a, b]$. Similarly $\xi_{\mathcal{S}} \models \mathbf{F}_{[a,b]}\psi$ specifies that ψ must hold at some time t' of the given interval. Finally, $\xi_{\mathcal{S}} \models \varphi\, \mathbf{U}_{[a,b]}\psi$ specifies that φ must hold starting from time 0 until a specific time $t \in [a, b]$ at which ψ becomes true. Formally, the satisfaction of a formula φ for a trajectory $\xi_{\mathcal{S}}$ at time t is defined as:

$$\begin{array}{lcl}
(\xi_{\mathcal{S}}, t) \models \mu & \Leftrightarrow & \mu(\xi_{\mathcal{S}}(t)) > 0 \\
(\xi_{\mathcal{S}}, t) \models \neg\mu & \Leftrightarrow & \neg((\xi_{\mathcal{S}}, t) \models \mu) \\
(\xi_{\mathcal{S}}, t) \models \varphi \wedge \psi & \Leftrightarrow & (\xi_{\mathcal{S}}, t) \models \varphi \wedge (\xi_{\mathcal{S}}, t) \models \psi \\
(\xi_{\mathcal{S}}, t) \models \mathbf{F}_{[a,b]}\varphi & \Leftrightarrow & \exists t' \in [t + a, t + b], (\xi_{\mathcal{S}}, t') \models \varphi \\
(\xi_{\mathcal{S}}, t) \models \mathbf{G}_{[a,b]}\varphi & \Leftrightarrow & \forall t' \in [t + a, t + b], (\xi_{\mathcal{S}}, t') \models \varphi \\
(\xi_{\mathcal{S}}, t) \models \varphi\ \mathbf{U}_{[a,b]}\ \psi & \Leftrightarrow & \exists t' \in [t + a, t + b] \text{ s.t. } (\xi_{\mathcal{S}}, t') \models \psi \\
 & & \wedge \forall t'' \in [t, t'], (\xi_{\mathcal{S}}, t'') \models \varphi.
\end{array} \tag{2.5}$$

Similar definitions as the ones in (2.4) and (2.5) can also be provided when the intervals of the temporal operators are open, such as $(a, b], [a, b)$, or (a, b), or unbounded, such as $[a, +\infty)$. The *bound* of an STL formula is defined as the maximum over the sums of all

nested upper bounds on the temporal operators of the STL formula. For instance, given $\psi = \mathbf{G}_{[0,20]}\mathbf{F}_{[1,6]}\varphi_1 \wedge \mathbf{F}_{[2,25]}\varphi_2$, the *bound* can be calculated as $\max(6+20, 25) = 26$. An STL formula φ is *bounded-time* if it contains no unbounded operators.

Robust Satisfaction A *quantitative* or *robust semantics* is defined for an STL formula φ by associating it with a real-valued function ρ_φ of the trajectory $\xi_\mathcal{S}$ and time t, which provides a "measure" of the margin by which φ is satisfied. Specifically, we require $(\xi_\mathcal{S}, t) \models \varphi$ if and only if $\rho_\varphi(\xi_\mathcal{S}, t) > 0$. The magnitude of $\rho_\varphi(\xi_\mathcal{S}, t)$ can then be interpreted as an estimate of the "distance" of $\xi_\mathcal{S}$ from the set of signals satisfying or violating φ.

We define the quantitative semantics as follows:

$$\begin{array}{lcl}
\rho_\mu(\xi_\mathcal{S}, t) & = & \mu(\xi_\mathcal{S}(t)) \\
\rho_{\neg\mu}(\xi_\mathcal{S}, t) & = & -\mu(\xi_\mathcal{S}(t)) \\
\rho_{\varphi\wedge\psi}(\xi_\mathcal{S}, t) & = & \min(\rho_\varphi(\xi_\mathcal{S}, t), \rho_\psi(\xi_\mathcal{S}, t)) \\
\rho_{\mathbf{G}_{[a,b]}\varphi}(\xi_\mathcal{S}, t) & = & \min_{t'\in[t+a,t+b]} \rho_\varphi(\xi_\mathcal{S}, t') \\
\rho_{\mathbf{F}_{[a,b]}\varphi}(\xi_\mathcal{S}, t) & = & \max_{t'\in[t+a,t+b]} \rho_\varphi(\xi_\mathcal{S}, t') \\
\rho_{\varphi\mathbf{U}_{[a,b]}\psi}(\xi_\mathcal{S}, t) & = & \max_{t'\in[t+a,t+b]}(\min(\rho_\psi(\xi_\mathcal{S}, t'), \\
 & & \qquad \min_{t''\in[t,t']} \rho_\varphi(\xi_\mathcal{S}, t'')).
\end{array} \tag{2.6}$$

Using the above definitions, the robustness value can be computed recursively for any STL formula. For brevity, in this chapter we use $\rho_\varphi(\xi_\mathcal{S}) = \rho_\varphi(\xi_\mathcal{S}, 0)$.

Chapter 3

Synthesizing Robust Control Strategies

3.1 Introduction

A key step in the design of Cyber-Physical Systems (CPS) or robotic systems, is the synthesis of robust control strategies from high level-safety specifications. One would like to synthesize controllers for safety-critical systems which can control such systems to satisfy high-level safety requirements. In this chapter, we address the problem of synthesizing robust controllers for linear systems from high-level temporal specifications. Specifically, we propose Correct-by-Construction controller synthesis algorithms based on the Counter-Example Guided Inductive Synthesis (CEGIS) paradigm.

Correct-by-construction controller synthesis from high-level formal specifications offers a promising means of raising the level of abstraction for implementation. In particular, *reactive synthesis* from temporal logic generates programs or controllers which maintain an ongoing interaction with their (possibly adversarial) environments. Reactive synthesis from linear temporal logic using automata-theoretic methods has been demonstrated for synthesizing high-level controllers for robotics. However, for embedded or robotic systems, reactive synthesis becomes much more challenging for several reasons. First, the specification languages go from discrete-time, propositional temporal logics to metric-time temporal logics over both continuous and discrete signals, so the previous automata-theoretic methods do not easily extend. Second, even for simple classes of dynamical systems and metric-time temporal logics, verification is itself undecidable, let alone synthesis. Third, the state of the art for solving games over infinite state spaces, as required for metric or quantitative temporal objectives, is far less developed than that for finite games.

To address these challenges, researchers have resorted to various simplifications. One simplification is to consider the control problem over a finite horizon instead of infinite horizons. This ensures verification is decidable for many interesting and practical systems and specification languages, like metric temporal logic (MTL) [90] and signal temporal logic (STL)

[104]. Reachability [109] based techniques offer a suite of well-developed and mature tools for handling finite horizon games, whose constraints amount to simple safety invariants such as obstacle avoidance. However, direct extensions to temporal constraints are not straightforward. Existing approaches model the environment as a bounded, non-deterministic disturbance [38], which is often unrealistic, leading to infeasibility in applications like autonomous driving for even very mundane cases. Instead, one may have a more complicated model of the environment that leverages either data-driven techniques or known aspects of the behavior of the other agents, such as their formal specifications.

In this chapter we build a synthesis engine which solves the reactive-synthesis problem by solving a series of finite-horizon robust control problems. This technique known as Receding Horizon Control (RHC) was solved using a *Counter-Example Guided Inductive Synthesis* (CEGIS) [141] framework in [128]. However, the practical implementation of the CEGIS framework that was provided was unsound. We first provide empirical results which show that naively implementing the CEGIS framework lead to scaling issues. To overcome the scalability issue and ensure soundness of our algorithm, we provide 2 variants of CEGIS which, (i) hybrid CEGIS-limits problem growth across iterations; and (ii) dominant CEGIS-builds dominant strategies using Satisfiability Modulo Theory (SMT) [13] as opposed to optimally robust strategies. Finally, we provide a theoretical analysis of our algorithms along with a worst-case convergence characterization. The results in this chapter are adapted from [155].

3.2 Preliminaries

We first introduce the flavor of reactive synthesis we consider in this chapter, Receding Horizon Control, and describe how the CEGIS framework can be instantiated to solve it effectively.

3.2.1 Receding Horizon Control and Counter-Example Guided Inductive Synthesis

A promising and scalable approach to synthesizing reactive strategies is to formulate reactive synthesis over a finite, receding horizon as a series of zero-sum games between the system and the environment, where the environment assumptions and system objectives are systematically encoded into rewards [128]. This form of controller, known as Receding Horizon Control (RHC), offers a (limited) form of reactivity. Each game is solved using a *Counter-Example Guided Inductive Synthesis* (CEGIS) [141, 6] scheme to search for a dominant strategy.

Model Predictive Control (MPC) or *Receding Horizon Control* (RHC), is a well studied hybrid system control method [58, 112]. In RHC, at each time step, the state of the system is observed and a finite horizon optimization problem is solved, given a set of constraints and cost function J. Given, the system dynamics f, we locally linearize f at each MPC step to solve the optimization problem, to generate a sequence optimal control $\mathbf{u}^*$. For example,

at time t=k, the dynamics f are linearized around the current system state and used to compute the optimal finite horizon control sequence $\mathbf{u}^*$. Only the first component of $\mathbf{u}^*$ is applied to the system and the new system state is computed at time $t = k+1$. The dynamics f are now linearized around the new state at $t = k + 1$ and a similar optimization problem is solved. While the global optimum of MPC is not guaranteed, the technique is frequently used and performs well in practice. This framework has seen success on (hybrid-)linear dynamical systems and specifications in a large fragment of Signal Temporal Logic, where the resulting optimization problem reduces to a Mixed Integer Linear Programming (MILP) Problem [127]. The cost function J, in this case was the robust semantics of STL [104].

In [128], the authors extended the framework in [127] to synthesize reactive controllers in the presence of disturbance from high level STL specifications. To solve the RHC at each time step, [128] proposed using a CEGIS like framework to synthesis a robust optimal control strategy at each step of the MPC. At each iteration of the MPC, they solved a zero-sum game between the controller (learner) and the environment (oracle). The learner solved the optimization problem to maximize the robustness to propose a candidate control strategy to the oracle. The oracle, solved a similar optimization problem to minimize the robustness. If the robustness is negative, the sequence of environment actions is a *counter-example* and is returned to the learner. The learner now proposes a new control strategy that is robust to all previously seen counter-examples. The process iterates until a dominant strategy is found.

Practical implementations of CEGIS suffers from two major shortcomings. First, CEGIS is an iterative paradigm that produces a series of candidate solutions with corresponding counterexamples. In each iteration of CEGIS, one must simultaneously solve the original optimization problem for all previously found counterexamples. The authors in [128] speculated that in the case of MILPs, CEGIS would scale poorly as the set of counterexamples grew; but this had not been empirically demonstrated. Nevertheless, this scaling issue was circumvented by considering only the most recent counterexample in each iteration of CEGIS. This heuristic leads to the second shortcoming, considering only the most recent counterexample may oscillate indefinitely between the same set of counterexamples.

Example 2. *Consider the situation shown in Fig 3.1. Depending on the ego car's behavior,*

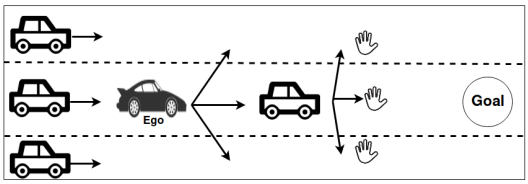


Figure 3.1. The ego car is attempting to reach the goal while avoiding collisions with the other cars.

the lead car may move to block the ego car, leading to a crash. If the ego car stops, it may be rear ended. Reasoning about the infeasibility of reaching the goal safely under an adversarial

environment requires access to at least three counterexamples. Maintaining only the last counterexample will cause the CEGIS to loop indefinitely. An ideal synthesis procedure would quickly reveal that the environment assumptions need to be strengthened (e.g. assume that given enough time the other cars would avoid the ego car or stop). Such discrete sub-problems arise naturally within more complicated (continuous space and time) driving scenarios.

To deploy a RHC on a complex CPS, the synthesis procedure associated with each planning step must be implemented in real time, and making CEGIS practical is an important first step. However, prior efforts to speed up CEGIS lead to incompleteness or non-termination. For example, [52] evaluated alternate techniques such as Monte Carlo and dual formulations for CEGIS to improve tractability. These techniques either provide only probabilistic completeness, or suffer from similar scaling-related issues.

CEGIS for RHC: Given a cost function over trajectories of system $J : \Xi \to \mathbb{R}$, and safety specification φ (as defined in Section 2.3) the game-theoretic formulation of the controller synthesis problem can be represented as a *minimax* optimization problem,

$$\begin{aligned} &\min_{\mathbf{u}\in\mathcal{U}} \max_{\mathbf{e}\in\mathcal{E}} J(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})) \\ &\text{subject to } \forall \mathbf{e} \in \mathcal{E} \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \models \varphi \end{aligned} \tag{3.1}$$

In this chapter, the cost function is defined to be the negative of the quantitative semantics of the safety specification i.e., $J = -\rho_\varphi$ and we replace the constraints by the quantitative semantics $\rho_\varphi(\mathbf{u}, \mathbf{e})$. Moreover, to avoid the $-$ sign in the objective, we change the control objective to max and the environment objective to min. An ϵ-robust control strategy is a $\mathbf{u}$ such that $\forall \mathbf{e} \in \mathcal{E}\ \rho_\varphi(\mathbf{u}, \mathbf{e}) > \epsilon$. Such solutions are often desirable in RHC as they heuristically offer more resilience to the uncertainty introduced by modeling errors (introduced by considering (2.2) as an approximation for (2.1)) and other finite horizon approximations.

To solve (3.1), [128] proposed a CEGIS style algorithm. As discussed in Section 2.1, the oracle in a CEGIS instance only accepts counter-example queries. The learner infers a robust controller in every iteration of RHC. Hence, the concept class for the learner in this case is domain of controls $\mathcal{U}$. The oracle here is a mapping from $\mathcal{U} \to \mathcal{E}$, i.e., given a candidate control sequence, the oracle provides counter-example environment sequence that falsifies the proposed control strategy.

The learner $\mathcal{L}_{\mathcal{C}_N}$ solves,

$$\begin{aligned} \mathbf{u}^* = &\operatorname*{argmax}_{\mathbf{u}\in\mathcal{U}} && \rho_\varphi(\mathbf{u}, \mathbf{e}) \\ &\text{subject to } \forall e \in \mathcal{E}_{CE} && \rho_\varphi(\mathbf{u}, \mathbf{e}) > 0 \end{aligned} \tag{3.2}$$

where instead of finding the optimal control strategy robust to all possible environment behaviors, the learner proposes an optimal control strategy $\mathbf{u}^*$ robust to a finite set of environments, $\mathcal{E}_{CE}$ called the counter-example set, to the oracle $\mathcal{O}_{\mathcal{C}_N}$. This limits the size of

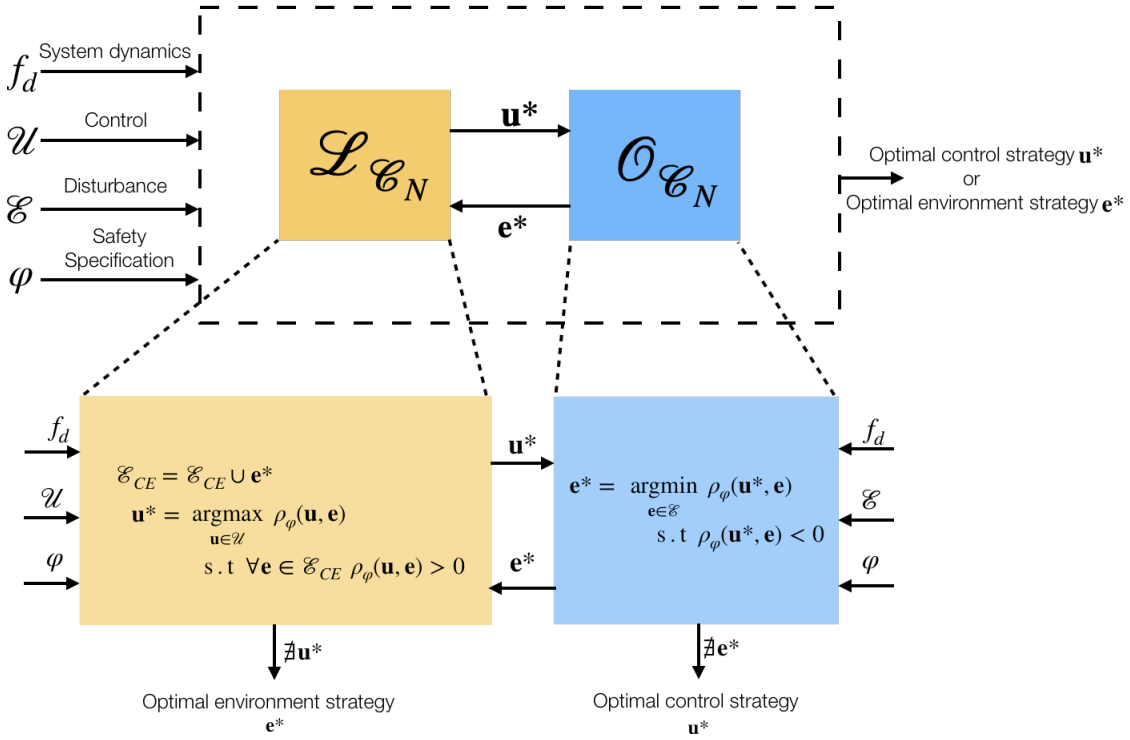


Figure 3.2. Naive CEGIS: $\mathcal{C}_N = (\mathcal{L}_{\mathcal{C}_N}, \mathcal{O}_{\mathcal{C}_N})$. The oracle $\mathcal{O}_{\mathcal{C}_N}(\mathcal{L}_{\mathcal{C}_N})$ is shown in blue (yellow). The oracle and learner together play a zero sum game. The size of the counter-example $\mathcal{E}_{CE}$ set grows every iteration. Hence, over time the optimization problem solved by the $\mathcal{L}_{\mathcal{C}_N}$ grows while that solved by $\mathcal{O}_{\mathcal{C}_N}$ remains the same.

the optimization problem the learner has to solve. The oracle $\mathcal{O}_{\mathcal{C}_N}$ solves,

$$\begin{aligned} \mathbf{e}^* = &\underset{\mathbf{e}\in\mathcal{E}}{\text{argmin}}\ \rho_\varphi(\mathbf{u},\mathbf{e}) \\ &\text{subject to } \rho_\varphi(\mathbf{u},\mathbf{e}) < 0 \end{aligned} \tag{3.3}$$

and searches for an optimal environment strategy $\mathbf{e}^*$ that can break the current proposed control sequence $\mathbf{u}^*$. Such a control strategy is called a counter-example and is returned to the learner $\mathcal{L}_{\mathcal{C}_N}$. The learner than adds it to the set of counter-examples $\mathcal{E}_{CE}$ and the process continues. The game ends when either the learner cannot find the optimal robust control strategy $\mathbf{u}^*$ or the oracle fails to find a counter-example $\mathbf{e}^*$ to the proposed candidate control strategy. This tends to scale poorly for the learner as the size of the counter-example set $\mathcal{E}_{CE}$ grows over the iterations. We refer to this implementation of CEGIS as *naive CEGIS* $\mathcal{C}_N = (\mathcal{L}_{\mathcal{C}_N}, \mathcal{O}_{\mathcal{C}_N})$ shown in Fig 3.2. This variant of CEGIS is guaranteed to converge asymptotically. However, this is not practical since, the optimization problem solved by $\mathcal{L}_{\mathcal{C}_N}$ grows indefinitely which may not be solvable by any existing optimization engine.

To avoid the scaling issues introduced by the growing counter-example set $\mathcal{E}_{CE}$ in MILPs, the authors in [128] proposed the learner considers only the last counterexample returned by the oracle. We refer to this variant of CEGIS as $\mathcal{C}_S = (\mathcal{L}_{\mathcal{C}_S}, \mathcal{O}_{\mathcal{C}_S})$ shown in Fig 3.3. While this

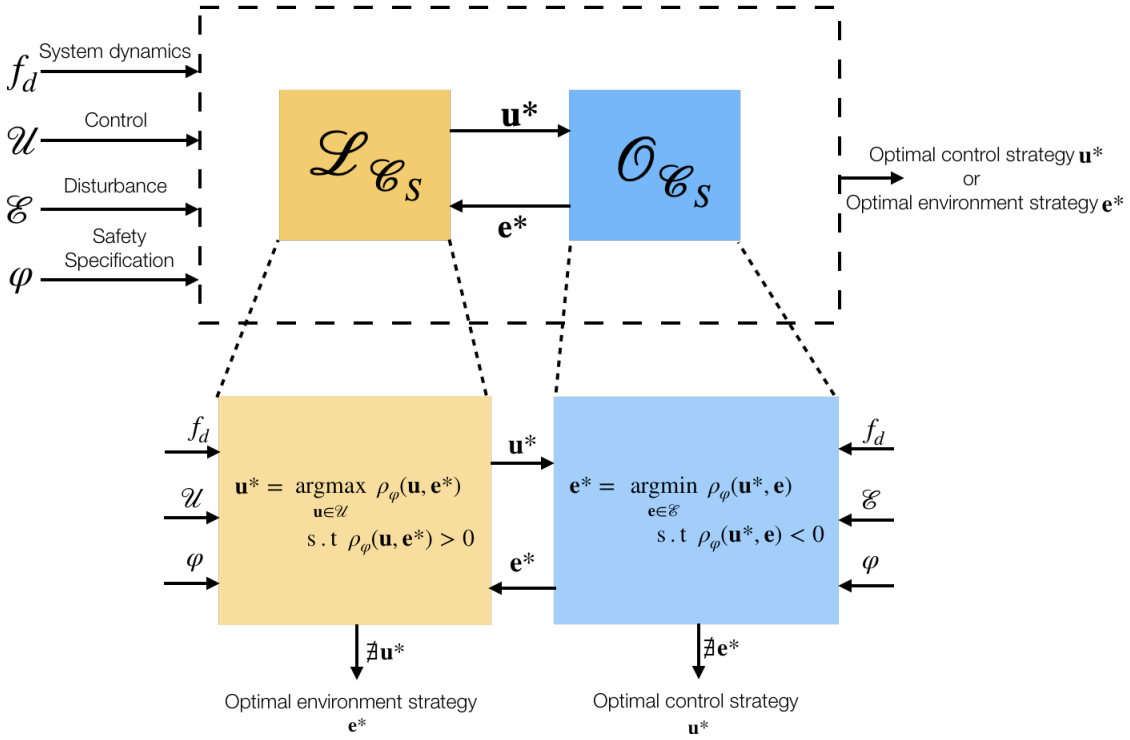


Figure 3.3. Single CE CEGIS: $\mathcal{C}_S = (\mathcal{L}_{\mathcal{C}_S}, \mathcal{O}_{\mathcal{C}_S})$. The oracle $\mathcal{O}_{\mathcal{C}_S}(\mathcal{L}_{\mathcal{C}_S})$ is shown in blue (yellow). The oracle and learner together play a zero sum game. The oracle $\mathcal{O}_{\mathcal{C}_S}$ is the same as $\mathcal{O}_{\mathcal{C}_N}$. However, the learner $\mathcal{L}_{\mathcal{C}_S}$ solves an optimization problem with the most recent counter-example $\mathbf{e}^*$ returned by the oracle $\mathcal{O}_{\mathcal{C}_S}$. This is the same optimization problem as in (3.2) where $\mathcal{E}_{CE} = \mathbf{e}^*$ is a singleton set, with the most recent counter-example. This ensures the size of the optimization problem solved by the learner $\mathcal{L}_{\mathcal{C}_S}$ does not grow over the iterations. Although this circumvents the scalability issue of $\mathcal{C}_N$, it suffers from oscillating indefinitely among the same set of counter-examples as shown in Example 2.

variant of CEGIS, overcomes the scalability issue in $\mathcal{C}_N$, it suffers from oscillating indefinitely among the same set of counter-examples as shown in Example 2. When $\mathcal{C}_S$ terminates we either get an optimal robust control strategy $\mathbf{u}^*$ or robust environment strategy $\mathbf{e}^*$. Unfortunately, $\mathcal{C}_S$ is not guaranteed to terminate because of the oscillating counter-example issue. Hence, this implementation of CEGIS is unsound, i.e., it is not guaranteed to find the robust control strategy if one exists.

The satisfiability version of the minimax optimization problem in (3.1) simply searches for a dominant control strategy $\mathbf{u}^* \in \mathcal{U}$ such that $\forall \mathbf{e} \in \mathcal{E}\ \rho_\varphi(\mathbf{u}^*, \mathbf{e}) > 0$. Let us refer to the CEGIS which solves this as $\mathcal{C}_B = (\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_B})$. The learner $\mathcal{L}_{\mathcal{C}_B}$ solves the following satisfiability problem $\mathbf{u}^* \in \mathcal{U}$ such that $\forall e \in \mathcal{E}_{CE}\ \rho_\varphi(\mathbf{u}^*, \mathbf{e}) > 0$ where $\mathcal{E}_{CE}$ is a set of counter-examples returned by the oracle. The oracle $\mathcal{O}_{\mathcal{C}_B}$ searches for $\mathbf{e}^* \in \mathcal{E}$ such that $\rho_\varphi(\mathbf{u}^*, \mathbf{e}^*) < 0$. This is pictorially shown in Fig 3.4. Here we can replace the learner and oracle by SMT solvers as opposed to optimization engines. While learner still suffers from scalability issues since $\mathcal{E}_{CE}$ grows in every iteration like in $\mathcal{C}_N$, it has been shown that SMT solver scale better than optimization engines. $\mathcal{C}_B$ also asymptotically converges to a dominant strategy if one

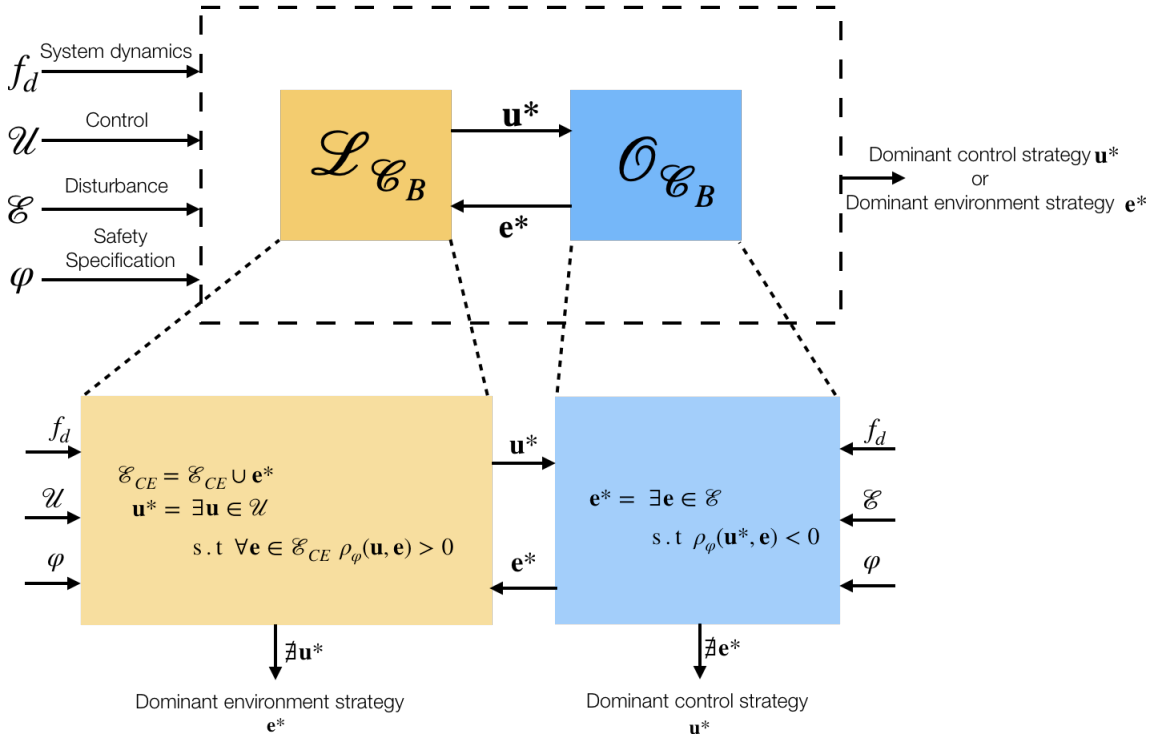


Figure 3.4. Satisfiability CEGIS: $\mathcal{C}_B = (\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_B})$. The oracle $\mathcal{O}_{\mathcal{C}_B}(\mathcal{L}_{\mathcal{C}_B})$ is shown in blue (yellow). The oracle and learner together play a zero sum game. Here unlike $\mathcal{C}_N$ and $\mathcal{C}_S$, they oracle and learner solve satisfiability problems as opposed to optimization problem. Hence, the strategy returned by either one of them is a dominant strategy and not an optimal one.

exists.

3.3 Problem Formulation

We address the problem of generating dominant control strategies $\mathbf{u}^*$ by solving the zero-sum game proposed in (3.1) using the CEGIS framework. The proposed framework should tackle both the scalability issue in $\mathcal{C}_N$ and the oscillatory behavior in $\mathcal{C}_S$.

3.4 Solution approach

Before we describe our CEGIS framework, it is fruitful to characterize some requirements for soundness and completeness of a CEGIS scheme.

Definition 3. *A (counter-example) oracle $\mathcal{O}_\mathcal{C}$ is sound if when given a candidate control strategy $\mathbf{u}^*$, it returns $\perp$ when $\mathbf{u}^*$ is a dominant (robust) control strategy; else returns $\mathbf{e}^*$ when $\mathbf{e}^*$ refutes $\mathbf{u}^*$, i.e., $\rho_\varphi(\mathbf{u}^*, \mathbf{e}^*) < 0$. The oracle $\mathcal{O}_\mathcal{C}$ is complete if when given a $\mathbf{u}^*$ such that there exists a counter-example $\mathbf{e}^*$, it returns $\mathbf{e}^*$.*

Definition 4. *A (candidate) learner $\mathcal{L}_\mathcal{C}$ is sound if when given a set of counter-examples $\mathcal{E}_{CE}$ returns $\perp$ when no dominant strategy $\mathbf{u}^*$ can simultaneously counter all counter-examples in $\mathcal{E}_{CE}$; else returns a dominant control strategy $\mathbf{u}^*$ such that $\forall e \in \mathcal{E}_{CE}\ \rho_\varphi(\mathbf{u}^*, e) > 0$. The learner $\mathcal{L}_\mathcal{C}$ is complete if it returns $\mathbf{u}^*$ such that $\forall e \in \mathcal{E}_{CE}\ \rho_\varphi(\mathbf{u}^*, e) > 0$ for a given counter-example set $\mathcal{E}_{CE}$ when such a $\mathbf{u}^*$ exists.*

The soundness of a CEGIS $\mathcal{C}$ framework follows immediately from the soundness of the oracle $\mathcal{O}_\mathcal{C}$ and the learner $\mathcal{L}_\mathcal{C}$.

Proposition 1 (Soundness of CEGIS)**.** *Let $\mathbf{u}^*$ be the result of a CEGIS framework$\mathcal{C}$ using a sound oracle $\mathcal{O}_\mathcal{C}$ and sound learner $\mathcal{L}_\mathcal{C}$ for specification φ. If $\mathbf{u}^* = \perp$ then there is no dominant control strategy and if $\mathbf{u}^* \neq \perp$, then $\mathbf{u}^*$ is a dominant strategy.*

Similarly, we observe that if either the oracle $\mathcal{O}_\mathcal{C}$ or the learner $\mathcal{L}_\mathcal{C}$ are not complete, then the CEGIS framework $\mathcal{C}$ is not complete.

From Definition 3, it is clear that $\mathcal{O}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_S}, \mathcal{O}_{\mathcal{C}_N}$ are all sound. From Definition 4, it is clear that $\mathcal{L}_{\mathcal{C}_B}, \mathcal{L}_{\mathcal{C}_N}$ are sound but $\mathcal{L}_{\mathcal{C}_S}$ is not sound. This is because the strategy returned by $\mathcal{L}_{\mathcal{C}_S}$ is guaranteed to refute only the last counter-example returned by $\mathcal{O}_{\mathcal{C}_S}$ and not all previous seen counter-examples. Hence, from Proposition 1, $\mathcal{C}_B$ and $\mathcal{C}_N$ is sound while $\mathcal{C}_S$ is not sound. This explains why $\mathcal{C}_S$ is not guaranteed to find the optimal dominant strategy and suffers from oscillatory behavior.

The completeness of the oracles $\mathcal{O}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_S}, \mathcal{O}_{\mathcal{C}_N}$ and learners $\mathcal{L}_{\mathcal{C}_B}, \mathcal{L}_{\mathcal{C}_S}, \mathcal{L}_{\mathcal{C}_N}$ depends on the completeness of the underlying optimization and SMT engines. In the remaining of this chapter, we assume that are complete.

This poses the question: *If the learner $\mathcal{L}_\mathcal{C}$ and oracle $\mathcal{O}_\mathcal{C}$ are complete, does that imply the CEGIS loop $\mathcal{C} = (\mathcal{L}_\mathcal{C}, \mathcal{O}_\mathcal{C})$ is complete?*

Example 3. *Consider the specification $\varphi = u + e \geq 0$ where $u, e \in [-1, 1]$. Clearly, $u = 1$ is a dominant strategy. Let us see what can happen is we use the CEGIS framework $\mathcal{C}_B = (\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_B})$. We know that $\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_B}$ are both sound and complete (Definition 4, 3). Suppose in the first iteration the counter-example set is $\mathcal{E}_{CE} = \{0\}$, $\mathcal{L}_{\mathcal{C}_B}$ proposes $\mathbf{u}^* = 0$. The $\mathcal{O}_{\mathcal{C}_B}$ can refute by providing $\mathbf{e}^* = -0.1$. The counter-example set grows to $\mathcal{E}_{CE} = \{0.0, -0.1\}$. $\mathcal{L}_{\mathcal{C}_B}$ can now propose $\mathbf{u}^* = 0.1$ and the $\mathcal{O}_{\mathcal{C}_B}$ now proposes $\mathbf{e}^* = -0.11$. Observe that this sequence of appending 1s can continue indefinitely. Thus $\mathcal{C}_B$ would never halt despite the existence of a dominant strategy.*

Example 3 proves that although $\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_B}$ is sound and complete, $\mathcal{C}_B$ is sound but not complete. However, if we were to use $\mathcal{L}_{\mathcal{C}_N}$ instead of $\mathcal{L}_{\mathcal{C}_B}$ it would maximize $\rho_\varphi = u + e$ and would propose $\mathbf{u}^* = 1.0$ in the first iteration. Alternately using $\mathcal{O}_{\mathcal{C}_N}$ instead of $\mathcal{O}_{\mathcal{C}_B}$ would minimize ρ_φ and propose $\mathbf{e}^* = -1.0$ and the CEGIS loop would terminate in a couple of iterations. This suggests forcing either the learner or the oracle to return "optimal" (max or min ρ_φ)) candidate or counter-example leads to fewer iterations before termination.

However, the optimality of the learner or oracle does not guarantee the overall CEGIS loop is complete. Consider a situation where $\forall \mathbf{u} \in \mathcal{U}, \mathbf{e} \in \mathcal{U} \rho_\varphi(\mathbf{u}, \mathbf{e}) = 0$, then given any $\mathbf{u}^*$

there are infinite counter-examples. Hence, finding the "optimal" counter-example does not help converge to a dominant strategy. At this time, we do not know of any general conditions on ρ_φ that guarantees completeness of the CEGIS loop.

3.4.1 CEGIS for dominant strategies $\mathcal{C}_D$

The first variant of CEGIS we propose combines an optimal counterexample oracle $\mathcal{O}_{\mathcal{C}_N}$ and the satisfiability learner $\mathcal{L}_{\mathcal{C}_B}$ which proposes dominant control strategies as opposed to optimal control strategies. We refer to this as $\mathcal{C}_D = (\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_N})$ (Fig 3.5).

As a consequence of using $\mathcal{O}_{\mathcal{C}_N}$ (making the counter-example optimal as opposed to the control strategies) is that we can now obtain an "anytime" algorithm by maintaining the $\mathbf{u}$ with the best "worst-case" seen so far. This control strategy is the closest to a dominant strategy that has been found so far. This is relevant for time bounded computations where optimal strategies, while preferable, may not be necessary e.g., when φ is a soft constraint as opposed to a hard requirement. Additionally, while we do not get completeness, we do get a weak termination guarantee. *Suppose we are willing to accept a solution* $\mathbf{u}^*$ *that allows* $\rho_\varphi(\mathbf{u}^*, \cdot) \in [\delta, 0)$ *for some* $\delta < 0$*, then* $\mathcal{C}_D$ *will terminate*

Theorem 1 ($\delta-$termination)**.** *If* $\mathcal{U}$ *is bounded, and oracle returns optimal counter-examples and* $\rho_\varphi(\cdot,\cdot)$ *satisfies* (2.3) *and* (2.3)*, then either the CEGIS loop terminates or for any* $\delta < 0$ *there is an iteration* $n \in \mathbb{N}$ *such that for* $\forall \mathbf{e} \in \mathcal{E}$ $\rho_\varphi(\mathbf{u}_n, \mathbf{e}) \geq \delta$*.*

Lemma 1. *Let* $\rho_\varphi(\cdot,\cdot)$ *satisfy* (2.3) *and* (2.3) *with Lipschitz constant* L_{ρ_φ}*. If* $\rho_\varphi(\mathbf{u}, \mathbf{e}) = \delta \neq 0$*, then for all* $\mathbf{u}'$ *in the open ball of radius* $|\delta / L_{\rho_\varphi}|$ *centered at* $\mathbf{u}$ *we have* $\varphi(\mathbf{u}, \mathbf{e}) = \varphi(\mathbf{u}', \mathbf{e})$*.*

Proof: [Lemma 1] Via Lipschitz continuity (2.3), one must perturb $\mathbf{u}$ by atleast $|\delta / L_{\rho_\varphi}|$ to make $\rho_\varphi(\mathbf{u}', \mathbf{e})$ to 0. By (2.3) the sign of $\rho_\varphi(\mathbf{u}', \mathbf{e})$ determines $\varphi(\mathbf{u}', \mathbf{e})$. Thus, for all $\mathbf{u}'$ in the open ball of radius $|\delta / L_{\rho_\varphi}|$ centered at $\mathbf{u}$, $\varphi(\mathbf{u}, \mathbf{e}) = \varphi(\mathbf{u}', \mathbf{e})$. □

Proof: [Theorem 1] Pick an arbitrary $\delta < 0$. Assume for contradiction that the CEGIS loop does not terminate, and for each iteration i, $\rho_\varphi(\mathbf{u}_i, \mathbf{e}_i) < \delta$. Via Lemma 1, the ball of radius $|\delta / L_{\rho_\varphi}|$ of inputs around $\mathbf{u}_i$ is refuted by $\mathbf{e}_i$. Thus, at each iteration, including $\mathbf{e}_i$ in $\mathcal{E}_{CE}$ refutes at least this ball around $\mathbf{u}_i$. Observe that $\mathcal{U}$ is bounded. Thus, there exists an iteration $k \in \mathbb{N}$ where the learner must return $\perp$ since all of the input space has been refuted. This terminates the loop, leading to a contradiction. Therefore, either the CEGIS loops terminates or $\exists n \in \mathbb{N}$ such that $\rho_\varphi(\mathbf{u}_n, \mathbf{e}) \geq \delta$. Further, since the oracle is optimal, $\rho_\varphi(\mathbf{u}_n, \mathbf{e}) \geq \delta \implies \forall \mathbf{e} \in \mathcal{E},\ \rho_\varphi(\mathbf{u}_n, \mathbf{e}) \geq \delta$. □

Since, Theorem 1 holds for CEGIS frameworks where we use an oracle which provide optimal counter-examples, we have $\delta-$termination holds for both $\mathcal{C}_N, \mathcal{C}_D$. The proof of Theorem 1 suggests a simple worst-case complexity of the CEGIS schemes.

Theorem 2. *Consider CEGIS variants where oracle returns optimal counter-examples,* $\rho_\varphi(\cdot,\cdot)$ *satisfies* (2.3) *and* (2.3) *with Lipschitz constant* L_{ρ_φ} *and* $\mathcal{U} = [-1, 1]^{n_u}, \mathcal{E} = [-1, 1]^{n_e}$*.*

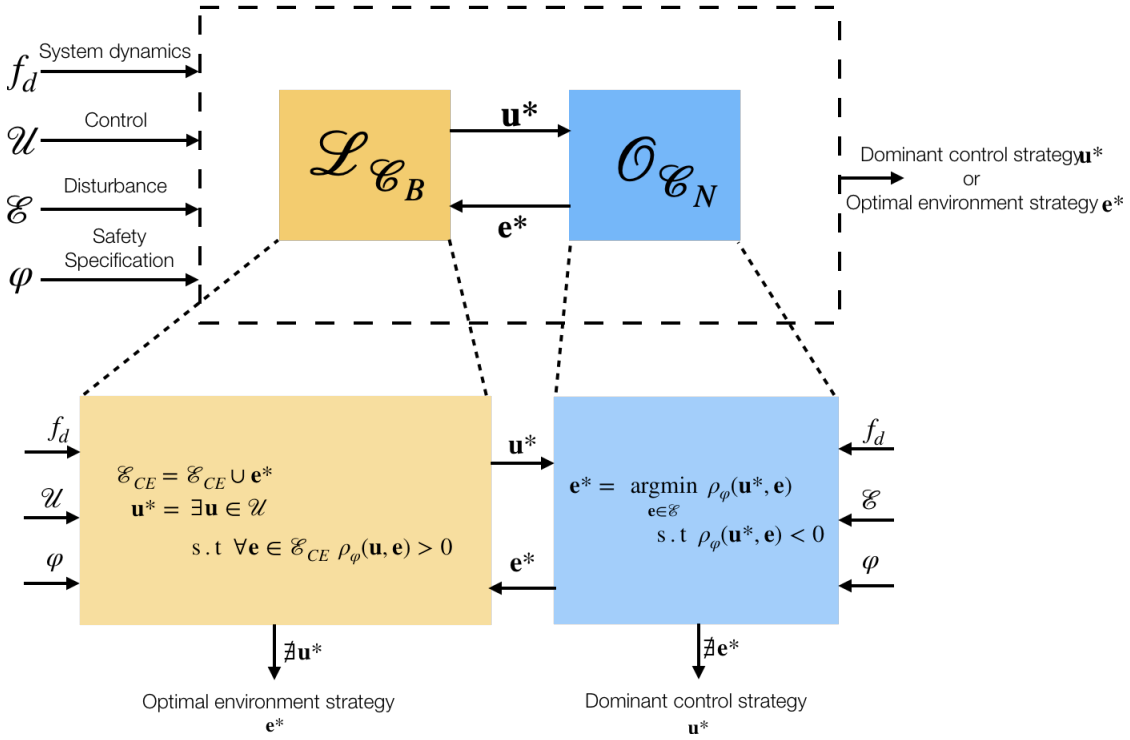


Figure 3.5. Dominant strategy, optimal counter-example CEGIS: $\mathcal{C}_D = (\mathcal{L}_{\mathcal{C}_B}, \mathcal{O}_{\mathcal{C}_N})$. The oracle $\mathcal{O}_{\mathcal{C}_N}(\mathcal{L}_{\mathcal{C}_B})$ is shown in blue (yellow). The oracle $\mathcal{O}_{\mathcal{C}_N}$) solves an optimization problem to propose optimal counter-examples. The learner $\mathcal{L}_{\mathcal{C}_B}$ solves a satiafiability problem and proposes dominant control strategies as opposed to optimal control strategies.

If each iteration of CEGIS takes at most T steps, then the worst case running time for $\delta-$termination is,

$$O(T \cdot (L_{\rho_\varphi}/\delta)^{n_u})$$

Proof: [Theorem 2 The result follows directly from the fact that that any two refuted ballscan share at most half their volume (since they cannot intersect the other's center) and refuted balls have volume proportional to $(L_{\rho_\varphi}/\delta)^{n_u}$. □ Theorem 1 and 2 hold for both $\mathcal{C}_N$ and $\mathcal{C}_D$. The oracle in both $\mathcal{O}_{\mathcal{C}_N}$ is a MILP optimization problem. The key difference being the learner in $\mathcal{C}_D$ is $\mathcal{L}_{\mathcal{C}_B}$ which solves a satisfiability query (as opposed to solving a growing MILP problem in $\mathcal{L}_{\mathcal{C}_N}$). For $\mathcal{L}_{\mathcal{C}_B}$ we use SMT engine with Real Linear Arithmetic, which empirically scale better with growing counter-example set $\mathcal{E}_{CE}$ compared to MILP solvers.

3.4.2 CEGIS with refuted input square $\mathcal{C}_H$

Example 4. *Consider a continuous variant of the familiar zero-sum game: Rock, Paper, Scissors (RPS). Two players, the system (u) and the environment (e) simultaneously choose either (R)ock, (P)aper or (S)cissors. Let us represent the state of the system by x, the*

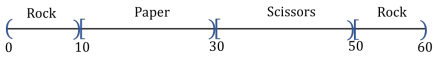


Figure 3.6. Continuous RPS.

environment by x', the system move by u and the environment move by e. The dynamics are given by $x = 60u$, $x' = 60e$.

Fig 3.6 depicts the embedding of Rock, Paper, and Scissors onto $\mathbb{R}$. If the state of either player is in $(0, 10) \cup [50, 60)$, it is considered to be playing R; similarly, $[10, 30)$ is P and $[30, 50)$ is S. Letting $A = (S, P, R)$, the system's winning condition is:

$$\varphi := \bigwedge_{i=0}^{2} \Big((x' \in A[i \bmod 3]) \implies (x \notin A[(i+1) \bmod 3]) \Big) \tag{3.4}$$

which encodes: Rock beats Scissors, Scissors beats Paper, and Paper beats Rock, respectively.

First observe that neither the system or the environment has a dominant strategy, and that for CEGIS to terminate the system must have a counterexample from each of the R, P and S regions in order to refute the entire input space.

If one were to use $\mathcal{C}_S$ to solve Example 4 then the loop would never terminate.

Let N denote the number of constraints required to encode a single counterexample in (3.2). The original motivation [128] for using $\mathcal{C}_S$ instead of $\mathcal{C}_N$ was that the size of (3.2) grew linearly with a rate at least as large as N. In this section, we explore a slight modification to (3.2) that grows linearly with a rate near the dimension of $\mathcal{U}$, which in general is much smaller than N. We denote this learner as $\mathcal{L}_{\mathcal{C}_H}$ and the resulting CEGIS scheme $\mathcal{C}_H = (\mathcal{L}_{\mathcal{C}_H}, \mathcal{O}_{\mathcal{C}_N})$.

One can think of the process of simultaneously handling the previously seen counterexamples as *implicitly* removing all parts of the input space that each counterexample refuted. We can also *explicitly* compute sets of refuted strategies ($\subset \mathcal{U}$) and modify the input space, $\mathcal{U}$ to exclude subsets containing the refuted candidates. Notice, however, that such sets are non-unique and potentially non-convex, so explicitly finding the maximal refuted set is not straightforward. Instead, inspired by Lemma 1, we choose to find the largest closed ball centered around $\mathbf{u}$ which is refuted by $\mathbf{e}$. More precisely, we seek to find the radius $r \in \mathbb{R}$ that solves:

$$\text{argmax}_{r \in \mathbb{R}} \{ r \in \mathbb{R}_{>0} \mid \forall \mathbf{u} \in B_r(\mathbf{u}) \; . \; \neg\varphi(\mathbf{u}, \mathbf{e}) \} \tag{3.5}$$

where $B_r(\mathbf{u})$ is the closed ball of radius r around $\mathbf{u}$. Denote the set of refuted balls by the ith round of CEGIS as $\mathcal{B}_i$. Then the learner $\mathcal{L}_{\mathcal{C}_H}$ solves:

$$\text{argmax}_{\mathbf{u} \in \mathcal{U}_i} \left\{ \min_{\mathbf{e} \in \mathcal{E}_k} (\rho_\varphi(\mathbf{u}, \mathbf{e})) \right\} \tag{3.6}$$

where $\mathcal{U}_i = \mathcal{U} \setminus \bigcup \mathcal{B}_i$ and $\mathcal{E}_k k$ is the k most recent counter-examples.

Remark 1. *To approximately solve* (3.5)*, first notice that given a candidate radius, one can use* $\mathcal{L}_{\mathcal{C}_B}$ *to query if a candidate control exists within that radius. If there does, one must decrease the radius size. If not, one may increase the radius size. As the input space is bounded, one can then simply perform a binary search over the radii, and over approximate the optimal radius by an arbitrarily small margin. In our experiments, these series of queries are handled efficiently by an SMT engine.*

Encoding a refuted ball needs roughly $2n_u$ constraints (where $\dim(\mathcal{U}) = n_u$). Further, if refers to each input at least once, $2n_u$ will be much smaller than N. Thus, this addresses the primary concern in developing $\mathcal{C}_S$. Given a counter-example, we can now choose to allot either N or $2n_u$ constraints. Let $i \in \mathbb{N}$ refer to the current iteration. In our implementation, we simultaneously keep the $k \in \mathbb{N}$ most recent counter-examples. The remaining $i - k$ candidate counterexample pairs are encoded as refuted balls. This trades-off encoding size for potentially more iterations.

Remark 2. *Due to the approximate nature of binary search on the real line, one must either err on the side of over- or under-approximation. We choose to err on the side over-approximation, potentially throwing out dominant strategies. A simple corollary of Lemma 1 is that if* $\mathbf{e}$ *is an optimal response to* $\mathbf{u}$*,* $\rho_\varphi(\mathbf{u}, \mathbf{e}) = a$*, and* r *is radius found by binary search, then all strategies,* $\mathbf{u}'$*, between* r *and the true radius* r^* *have* $\rho_\varphi(\mathbf{u}', \mathbf{e}) \leq a + (r - r^*)/L_{\rho_\varphi}$*. Thus, if* $(r - r^*)/L_{\rho_\varphi}$ *is at most* ϵ*, one will only miss strategies that are not* ϵ *robust. Thus, when computing* r*, if one uses an* $\epsilon/(2L_{\rho_\varphi})$ *tolerance and additionally adds* $\epsilon/(2L_{\rho_\varphi})$ *to the result, then one always over approximates* r^* *by less than* ϵ*. We refer to this approximation as "bloating by epsilon".*

Observe that this provides a tunable completeness guarantee for $\mathcal{C}_H$:

Theorem 3 (ϵ-Completeness)**.** *Given a bounded input space,* $\mathcal{U}$*, and* ρ_φ *satisfying* (2.3) *and* (2.3)*, if there exists an* ϵ*-robust solution, then* $\mathcal{C}_H$ *with* ϵ*-bloating will find a dominant strategy.*

Proof: [Theorem 3]At any iteration, i, of the CEGIS loop, let $\mathbf{u}_i$ and $\mathbf{e}_i$ be the candidate strategy and counter-example returned by $\mathcal{L}_{\mathcal{C}_H}$ and $\mathcal{O}_{\mathcal{C}_N}$, respectively. Due to ϵ-bloating, the explicit refuted ball in each round has non-zero radius. As in the proof for Theoremm 1, since $\mathcal{U}$ is bounded, $\mathcal{C}_H$ must terminate. Further, by construction, ϵ-bloating only omits strategies that are not ϵ-robust. Therefore, if there exists an ϵ-robust solution, then the final input space is non-empty. Thus, by exhaustive search, if there exists an ϵ-robust solution, then $\mathcal{C}_H$ with ϵ-bloating will find a dominant strategy. □

$\mathcal{C}_H$ is shown in Fig 3.7. Since the oracle returns optimal counter-examples, Theorems 2 and 1 hold for $\mathcal{C}_H$. The learner $\mathcal{L}_{\mathcal{C}_H}$ solves a sequence of SMT queries and a MILP optimization to propose an $\epsilon-$robust strategy.

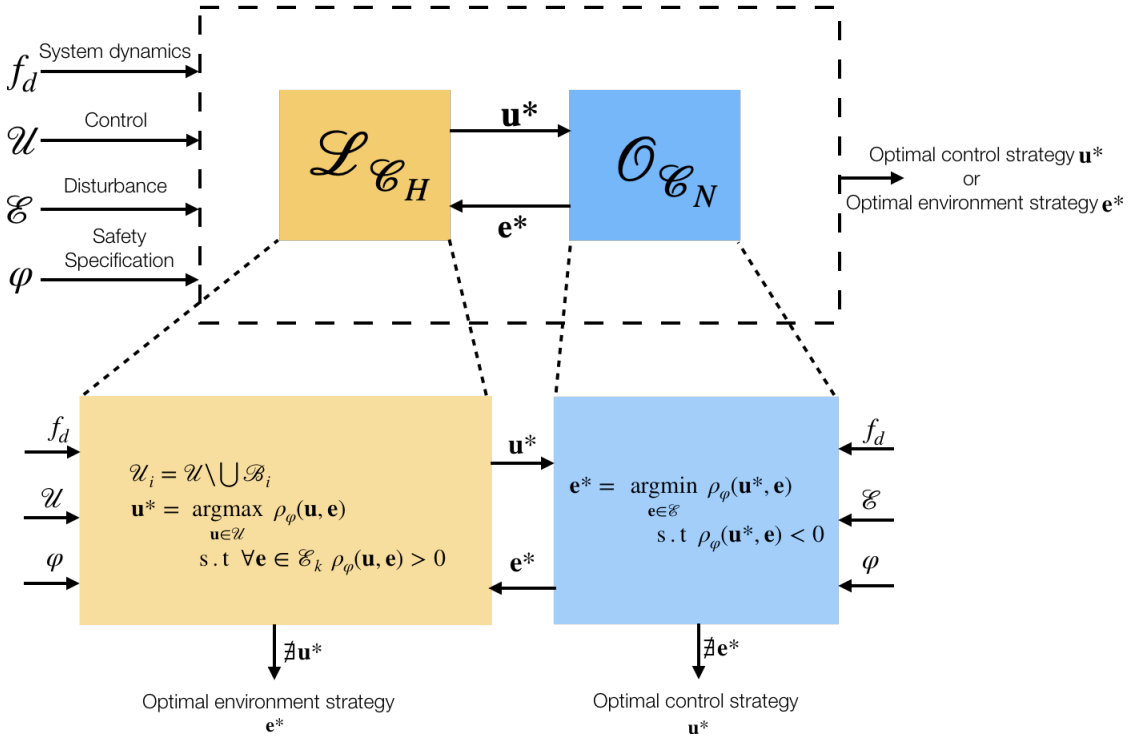


Figure 3.7. Dominant strategy, optimal counter-example CEGIS: $\mathcal{C}_D = (\mathcal{L}_{\mathcal{C}_H}, \mathcal{O}_{\mathcal{C}_N})$. The oracle $\mathcal{O}_{\mathcal{C}_N}(\mathcal{L}_{\mathcal{C}_H})$ is shown in blue (yellow). The oracle $\mathcal{O}_{\mathcal{C}_N}$) solves an optimization problem to propose optimal counter-examples. The learner $\mathcal{L}_{\mathcal{C}_H}$ solves a sequence of SMT queries to compute the refuted balls, and then solves a MILP problem to find an $\epsilon-$robust control strategy.

3.5 Evaluation

We summarize the CEGIS variants studied in this chapter in Table 3.1. We benchmark the performance of the CEGIS variants $\mathcal{C}_H, \mathcal{C}_D, \mathcal{C}_N$ and $\mathcal{C}_S$ across five experiments, each designed to vary the difficulty in a specific manner.

- In *Experiment 1*, we extend the RPS game (Example 4) to study the effects of subdividing the input space into more regions (increased N) while keeping the number of counterexamples fixed.
- *Experiment 2* extends Example 4 with additional moves. The number of counterexamples grows with the number of moves.
- In *Experiment 3* we study the overhead of introducing refuted rectangles.
- In *Experiment 4 and 5* we introduce linear dynamics to Example 4.

Our implementations of $\mathcal{C}_N$, $\mathcal{C}_S$, $\mathcal{C}_H$ and $\mathcal{C}_D$ are available as a python toolbox Magnum-STL [153]. Our tool encodes the system and the specifications as a MILP using the encoding

Symbol	Optimal	Memory	"Terminates"	New
$\mathcal{C}_N$	Yes	∞	Yes	No
$\mathcal{C}_S$	Yes	1	No	No
$\mathcal{C}_H$	Yes	$k \in \mathbb{N}$	Yes	Yes
$\mathcal{C}_D$	No	∞	Yes	Yes

Table 3.1. Summarizes the algorithms studied in this paper. "Optimal" means maximal with respect to a reward function capturing satisfaction of the high-level specification. "Memory" refers to the number of counterexamples the candidate oracle takes into account before proposing a candidate strategy. "Terminates" denotes whether the algorithm terminates.

in [127, 128]. It uses two backend solvers, GLPK [103] for MILP (interfaced through optlang [79]) and Z3 [33] for SMT (interfaced through pysmt [59]).

Further, as $\mathcal{C}_S$ cannot terminate in all but one experiment, for the purposes of comparison, we have omitted it. We consider two instances of $\mathcal{C}_H$ that remember the most recent $\mathcal{C}_H^{k=1}$ and the two most recent counterexamples $\mathcal{C}_H^{k=2}$, respectively.

3.5.1 Experiment 1

In this experiment we modify the embedding of R, P and S onto the real line (Fig 3.6) to that shown in Fig 3.8 where the domain is broken down into repeated $R-P-S$ segments.

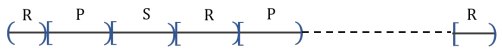


Figure 3.8. Generalized continuous rps.

Notice that the required number of counterexamples remains three. However, for the $\mathcal{C}_H$, the number of iterations required increases as they must explicitly sample more $(\mathbf{u}, \mathbf{e})$ pairs. Our results are shown in Table 3.2.

We see that as predicted in [128], $\mathcal{C}_N$ scales poorly as the number of regions increase. This is due to the fact that the encoding size, N, increases as the number of regions increased. $\mathcal{C}_H^{k=2}$ scales only slightly worse than $\mathcal{C}_H^{k=1}$ since, the MILP has to keep track of at least two counterexamples and is at least twice as big. $\mathcal{C}_D$ performs the best, where the time taken remains more or less invariant to the number of regions.

# Regions	$\mathcal{C}_D$	$\mathcal{C}_N$	$\mathcal{C}_H^{k=2}$	$\mathcal{C}_H^{k=1}$
4	0.263	1.34	1.04	0.819
7	0.435	6.36	6.34	3.58
10	0.594	31	12.2	6.54

Table 3.2. Experiment 1 run times in seconds.

3.5.2 Experiment 2

In this experiment, we generalize our RPS example to that shown in Fig 3.9: where the

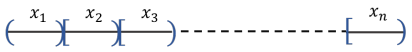


Figure 3.9. Continuous RPS with n counterexamples

domain is broken into $n \geq 1$ possible plays, $x_1, \cdots, x_n$ such that x_1 beats x_2, x_2 beats x_3, ..., x_n beats x_1.

In every round, the system wins if the following is satisfied:

$$\varphi_i = x_t' \in x_j \rightarrow x_t \notin x_i \tag{3.7}$$

where $j = i + 1 \pmod{n}$.

Here, both the number of counterexamples as well as the number of $(\mathbf{u}, \mathbf{e})$ pairs generated by CEGIS iterations increase. Our results are shown in Table 3.3. We observe that the results follow the same trend as *Exp 1*.

# Regions	$\mathcal{C}_D$	$\mathcal{C}_N$	$\mathcal{C}_H^{k=2}$	$\mathcal{C}_H^{k=1}$
1	0.039	0.061	0.062	0.061
2	0.155	0.394	0.392	0.360
3	0.350	1.68	1.14	0.676
4	0.604	6.01	2.26	1.34
5	0.951	180	3.81	1.98

Table 3.3. Experiment 2 run times in seconds.

3.5.3 Experiment 3

In this experiment, we consider a simple linear system, $x_{i+1} = x_i + \frac{5}{n}(u_i + w_i)$, and a high level specification, $\varphi = \bigvee_{i=0}^{H} x > 5$. Additionally, we have, $\mathcal{U} = \mathcal{E} = [-1, 1]$ and the initial

state $x_0 = 0$. It is clear that the environment has a dominant strategy $w_i = -1$, and our conjecture is that we need only a few iterations for $\mathcal{C}_H$ to converge. Particularly, we study the overhead of introducing refuted rectangles. Our results are shown in Fig 3.10.

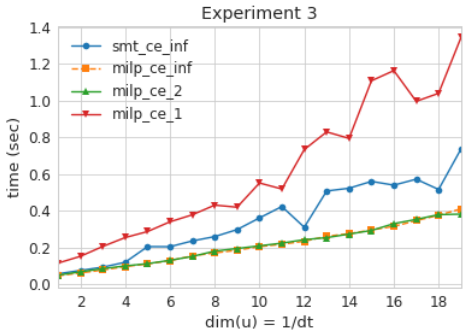


Figure 3.10. The x-axis shows the dimension of the input space, while the y-axis shows the time taken for the CEGIS loop to converge. *milp_ce_1* refers to $\mathcal{C}_H^{k=1}$, *milp_ce_2* refers to $\mathcal{C}_H^{k=2}$, *milp_ce_inf* refers to $\mathcal{C}_N$, and *smt_ce_inf* refers to $\mathcal{C}_D$.

All engines perform fairly well (compare with the times in *Experiments 1 and 2*). As the original specification has a small encoding which is comparable to encoding the refuted rectangles, i.e., $N \approx 2n_u$, in this particular case, it appears better to encode the counterexample than the refuted input space.

3.5.4 Experiments 4 and 5

Experiments 4 and 5 are modifications to *Experiment 1*. In *Experiment 4*, we introduce linear dynamics to the RPS,

$$x_{i+1} = x_i + 30u_i/n \qquad x'_{i+1} = x'_i + 30w_i/n \tag{3.8}$$

where n is the number of steps taken and the initial state is ($state_0 = x'_0 = 20$). The specifications on the system remain the same. As before, there is no dominant strategy for the system, and the number of counterexamples remains three. Our results are shown in Table 3.4. $\mathcal{C}_H^{k=1}$ performs reasonably well initially, but times out eventually (for discretization time > 3). $\mathcal{C}_D$ outperforms all the other methods and remains more or less invariant to changes to discretization time. The timeouts for $\mathcal{C}_N$ and $\mathcal{C}_H^{k=2}$ happen after two counterexamples are found. The encoding then seems to become too large for GLPK to handle. For $k = 1$ the initial increase is due in part to the size of the MILP encoding, but also because more iterations are required since the counterexample in each iteration refutes less of the total input space.

In *Experiment 5*, we introduce a small gap between R and P in *Experiment 4*. We denote this to be the '*Spock*'. We update the specification φ with an additional specification,

$$\varphi_{dom} = (x'_t \in \{R, P, S\}) \rightarrow (x \in Spock)$$

$\dim(\mathcal{U})$	$\mathcal{C}_D$	$\mathcal{C}_N$	$\mathcal{C}_H^{k=2}$	$\mathcal{C}_H^{k=1}$
1	0.35	18.01	1.52	0.88
2	0.53	timeout	timeout	5.11
3	1.00	timeout	timeout	6.38

Table 3.4. Experiment 4 run times in seconds.

Choosing *Spock* yields a small positive ρ_φ against all disturbances and is, thus, a dominant strategy. Nevertheless, $\mathbf{u} \in \mathit{Spock}$ has ρ_φ so small that $\mathcal{O}_{\mathcal{C}_N}$ does not propose *Spock* unless R, P and S are refuted. $\mathcal{C}_N$ and $\mathcal{C}_D$ would take at least three iterations to refute them.

$\dim(\mathcal{U})$	$\mathcal{C}_D$	$\mathcal{C}_N$	$\mathcal{C}_H^{k=2}$	$\mathcal{C}_H^{k=1}$
1	0.206	1.23	1.25	0.773
2	0.321	3.57	3.61	7.28
3	0.662	28.31	33.40	39.00

Table 3.5. Experiment 5 run times in seconds.

Introducing the *Spock* region significantly improves performance. $\mathcal{C}_N$ and $\mathcal{C}_H^{k=2}$ no longer time out. We suspect that this is due to the dominant strategy aiding in the branch and bound heuristics the MILP solver uses. In practice, such a region will often exist. For example, this could correspond to a near miss angle in the example in Fig 3.1. Again, $\mathcal{C}_D$ outperforms the others. Not shown are additional experiments increasing the time resolution by a factor of 12, at which point the optimality of $\mathcal{O}_{\mathcal{C}_N}$ leads to exponential blow up similar to the other variants.

3.6 Conclusion

In this chapter, we addressed the problem of synthesizing robust control strategies for continuous two-player games using the CEGIS framework. We studied the shortcomings of the existing practical implementations of the CEGIS algorithm, and introduced two algorithms based on SMT which overcome these. Finally, we conclude the chapter with theoretical analysis and empirical results which shows the scalability of our framework.

Chapter 4

Specification Synthesis for Controller Synthesis

4.1 Introduction

In Chapter 3 we studied how we can instantiate the CEGIS framework to synthesize robust control strategies that satisfy high-level safety specifications. A key challenge that arises during the design of safety-critical robotic systems, is quantifying or defining what one means by safety. Often a designer starts with a specification that they build from prior knowledge, and then refine it during subsequent cycles of the design process.

Techniques for automatic synthesis of controllers for safety-critical robotic systems from high-level specification languages promise to raise the level of abstraction for the designer while ensuring correctness of the resulting controller. In particular, several controller synthesis methods have been proposed for expressive temporal logics and a variety of system dynamics. However, a major challenge to the adoption of these methods in practice is the difficulty of writing or specifying the requisite formal specifications. Specifications that are poorly stated, incomplete, or inconsistent can produce synthesis problems that are unrealizable (no controller exists for the provided specification), intractable (synthesis is computationally too hard), or lead to solutions that fail to capture the designer's intent. In this chapter, we present an algorithmic approach to reduce the specification burden for controller synthesis from temporal logic specifications, focusing on the case where the original specification is unrealizable.

4.1.1 Diagnosis and Repair for Synthesis from Temporal Logic

Logical specifications can be provided in multiple ways. One approach is to provide *monolithic* specifications, combining within a single formula constraints on the environment with desired properties of the system under control. In many cases, a system specification can be conveniently provided as a contract, to distinguish the responsibilities of the system under control (guarantees) from the assumptions on the external, possibly adversarial environ-

ment [118, 116]. In such a scenario, an unrealizable specification can be made realizable by either *"weakening" the guarantees* or *"tightening" the assumptions*. In fact, when a specification is unrealizable, it could be either because the environment assumptions are too weak, or the requirements are too strong, or a combination of both. Finding the "problem" with the specification manually can be a tedious and time-consuming process, nullifying the benefits of automatic synthesis. Further, in the *reactive* setting, when the environment is adversarial, finding the right assumptions a priori can be difficult. Thus, given an unrealizable logical specification, there is a need for tools that localize the cause of unrealizability to (hopefully small) parts of the formula, and provide suggestions for repairing the formula in an "optimal" manner.

The problem of diagnosing and repairing formal requirements has received its share of attention in the formal methods community. The authors in [53] perform diagnosis on faulty executions of systems with specifications expressed in Linear Temporal Logic (LTL) and Metric Temporal Logic (MTL) [53]. They identify the cause of unsatisfiability of these properties in the form of prime implicants, which are conjunctions of literals, and map the failure of a specification to the failure of these prime implicants. Similar syntax-tree based definitions of unsatisfiable cores for LTL were presented in [133]. In the context of synthesis from LTL specifications, [126] addresses the problem of categorizing the causes of unrealizability, and how to detect them in high-level robot control specifications. The use of counter-strategies to debug unrealizable cores in a set of specifications or derive new environment assumptions for synthesis has also been explored [97, 89, 4, 98]. When a LTL specification in unrealizable, controller synthesis reduces to finding the path through an automaton composed of the system and the specification, that maximizes some reward function ([152, 93]). For automatons, repair reduces to finding the specification automaton closest to the original automaton ([86], [122]). However, these approaches suffer from computational blow up as the number of states increase. In [24] the authors use a sampling based technique to find a discrete state approximation of continuous system and define LTL properties of the system. The authors allow for controller synthesis for non adversarial environments by maximizing a cost function which maximizes some reward function. We provide corrections instead of finding the next best control in an adversarial environment.

In [88] and [81] the authors learn STL specifications from data collected from black box systems which best describe system. We consider white box systems where the dynamics and well known and provide corrections for controller synthesis.

Our approach, based on exploiting information already available from off-the-shelf optimization solvers, is similar to the one adopted by in [117] to extract unsatisfiable cores for Satisfiability Modulo Theories (SMT) solving.

In this chapter, we address the problem of diagnosing and repairing specifications formalized in *Signal Temporal Logic (STL)* [104], a specification language that is well-suited for hybrid systems. Our work is conducted in the setting of automated synthesis from STL using optimization methods in a Model Predictive Control (MPC) framework [127, 128]. In this approach to synthesis, both the system dynamics and the STL requirements encoded as mixed integer constraints on variables modeling the dynamics of the system and

its environment. Controller synthesis is then formulated as an optimization problem to be solved subject to these constraints [127]. In the reactive setting, this approach proceeds by iteratively solving a combination of optimization problems using a *Counterexample-Guided Inductive Synthesis* (CEGIS) scheme [128]. In this context, an unrealizable STL specification leads to an infeasible optimization problem. We leverage the ability of existing Mixed Integer Linear Programming (MILP) solvers to localize the cause of infeasibility to so-called *Irreducibly Inconsistent Systems* (IIS). Our algorithms use the IIS to localize the cause of unrealizability to the relevant parts of the STL specification. Additionally, we give a method for generating a *minimal set of repairs* to the STL specification such that, after applying those repairs, the resulting specification is realizable. The set of repairs is drawn from a suitably defined space that ensures that we rule out vacuous and other unreasonable adjustments to the specification. Specifically, in this paper, we focus on the numerical parameters in a formula, since their specification is often the most tedious and error-prone part. Our algorithms are sound and complete, i.e., they provide a correct diagnosis, and always terminate with a reasonable specification that is realizable using the chosen synthesis method, when such a repair exists in the space of possible repairs.

The problem of infeasibility in constrained predictive control schemes has also been widely addressed in the literature, e.g., by adopting robust MPC approaches, soft constraints, and penalty functions [85, 134, 15]. Rather than tackling general infeasibility issues in MPC, our focus is on providing tools to help debug the controller specification at design time. However, the deployment of robust or soft-constrained MPC approaches can also benefit from our techniques. Our use of MILP does not restrict our method to linear dynamical systems; indeed, we can handle constrained linear and piecewise affine systems, Mixed Logical Dynamical (MLD) systems [14], and certain differentially flat systems. The results in this chapter are adapted from [64].

4.2 Preliminaries

We consider discrete time hybrid dynamical systems defined by (2.2) in Section 2.2. Refer to Section 2.2 for the definition of system trajectories $\xi_{\mathcal{S}}(t; x_0, \mathbf{u}, \mathbf{e})$. In this chapter, we define the safety specification in STL defined in 2.4.1

4.2.1 Model Predictive Control

We have already covered *Model Predictive Control* (MPC) and *Receding Horizon Control* (RHC) in Section 3.2.1. In this chapter, we use STL to express temporal constraints on the environment and system runs for MPC. We then translate an STL specification into a set of mixed integer linear constraints, as further detailed below [127, 128]. Given a STL formula φ to be satisfied over a finite horizon H, the associated optimization problem has

the form:

$$\begin{aligned} \underset{\mathbf{u}}{\text{maximize}} \quad & \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})) \\ \text{subject to} \quad & \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})) > 0 \end{aligned} \tag{4.1}$$

which extracts a finite horizon control strategy $\mathbf{u}$ that maximizes the satisfaction of the specification φ, $\rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}))$ over the finite-horizon trajectory $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})$, while satisfying the STL formula φ at time step 0. In a closed-loop setting, we compute a fresh $\mathbf{u}$ at every time step $i \in \mathbb{N}$, replacing x_0 with x_i in (4.1) [127, 128].

While (4.1) applies to systems without disturbance inputs, a more general formulation can be provided to account for an uncontrolled disturbance input $\mathbf{e}$ that can act, in general, adversarially. To provide this formulation, we assume that the specification is given in the form of an STL *assume-guarantee (A/G) contract* [118, 116] $C = (V, \varphi_e, \varphi \equiv \varphi_e \rightarrow \varphi_s)$, where V is the set of variables, φ_e captures the assumptions (admitted behaviors) over the (uncontrolled) environment inputs $\mathbf{e}$, and φ_s describes the guarantees (promised behaviors) over all the system variables. A game-theoretic formulation of the controller synthesis problem [128] can then be represented as a *minimax* optimization problem:

$$\begin{aligned} \underset{\mathbf{u}}{\text{maximize}} \quad & \underset{\mathbf{e} \in \mathcal{E}_e}{\text{minimize}} \quad \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})) \\ \text{subject to} \quad & \forall \mathbf{e} \in \mathcal{E}_e \quad \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})) > 0, \end{aligned} \tag{4.2}$$

where we aim to find a strategy $\mathbf{u}$ that maximizes the the worst case satisfaction of $\rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}))$ over the finite horizon trajectory, under the assumption that the disturbance signal $\mathbf{e}$ acts adversarially. We use $\mathcal{E}_e$ in (4.2) to denote the set of disturbances that satisfy the environment specification φ_e, i.e., $\mathcal{E}_e = \{\mathbf{e} \in \mathcal{E} | \mathbf{e} \models \varphi_e\} \subseteq \mathcal{E}$.

4.2.2 Mixed Integer Linear Program Formulation

To solve the control problems in (4.1) and (4.2) the STL formula φ can be translated into a set of mixed integer constraints, thus reducing the optimization problem to a *Mixed Integer Program* (MIP), as long as the system dynamics can also be translated into mixed integer constraints. Specifically, in this paper, we consider control problems that can be encoded as *Mixed Integer Linear Programs* (MILP). These problems encompass, for instance, Mixed Logical Dynamic (MLD) systems [14] with STL specifications that only include piecewise linear or affine predicates.

The MILP constraints are constructed recursively on the structure of the STL specification as in [127, 128], and express the robust satisfaction value of the specification. A first set of variables and constraints capture the robust satisfaction of the atomic predicates of the formula. To generate the remaining constraints, we traverse the parse tree of φ from the leaves (associated with the atomic predicates) to the root node (corresponding to the robustness satisfaction value of the overall formula ρ_φ), adding variables and constraints that obey the quantitative semantics in (2.6).

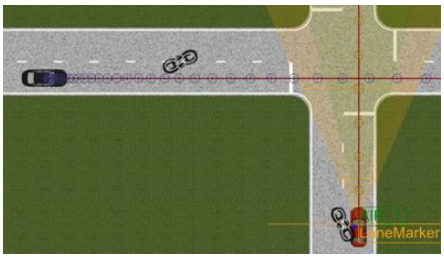


Figure 4.1. Vehicles crossing an intersection. The red car is the *ego* vehicle, while the black car is part of the environment.

Recall from Section 2.4.1 that the robustness value of subformulae with temporal and Boolean operators is expressed as the *min* or *max* of the robustness values of the operands over time. We discuss here the encoding of the *min* operator as an example. To encode $p = \min(\rho_{\varphi_1}, \ldots, \rho_{\varphi_n})$, we introduce Boolean variables z^{φ_i} for $i \in \{1, \ldots, n\}$ and MILP constraints:

$$\begin{gathered} p \leq \rho_{\varphi_i}, \sum_{i=1\ldots n} z^{\varphi_i} \geq 1 \\ \rho_{\varphi_i} - (1 - z^{\varphi_i})M \leq p \leq \rho_{\varphi_i} + (1 - z^{\varphi_i})M \end{gathered} \tag{4.3}$$

where M is a constant selected to be much larger than $|\rho_{\varphi_i}|$ for all i, and $i \in \{1, \ldots, n\}$. The above constraints ensure that $z^{\varphi_i} = 1$ and $p = \rho_{\varphi_i}$ only if ρ_{φ_i} is the minimum over all i. For *max*, we replace $\leq$ by $\geq$ in the first constraint of (4.3).

We solve the resulting MILP with an off-the-shelf solver. If the receding horizon scheme is feasible, then the controller synthesis problem is *realizable*, i.e., the algorithm returns a controller that satisfies the specification and optimizes the objective. However, if the MILP is infeasible, the synthesis problem is *unrealizable*. In this case, the failure to synthesize a controller may well be attributed to just a portion of the STL specification. In the rest of the chapter we discuss how infeasibility of the MILP constraints can be used to infer the "cause" of failure and, consequently, diagnose and repair the original STL specification.

4.3 Running Example

To illustrate our approach, we introduce a running example from the autonomous driving domain. As shown in Fig. 4.1, we consider a scenario in which two moving vehicles approach an intersection. The red car, labeled the *ego* vehicle, is the vehicle under control, while the black car is part of the external environment and may behave, in general, adversarially. The state of the system includes the position and velocity of each vehicle, the control input is the acceleration of the *ego* vehicle, and the environment input is the acceleration of the other vehicle, i.e.,

$$\begin{aligned} x_t &= [x_t^{\text{ego}}, y_t^{\text{ego}}, v_t^{\text{ego}}, x_t^{\text{adv}}, y_t^{\text{adv}}, v_t^{\text{adv}}] \\ u_t &= a_t^{\text{ego}} \quad e_t = a_t^{\text{adv}}. \end{aligned} \tag{4.4}$$

We also assume the dynamics of the system is given by a simple double integrator for each vehicle:

$$\begin{bmatrix} \dot{x^{\text{ego}}} \\ \dot{y^{\text{ego}}} \\ \dot{v^{\text{ego}}} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 1 \\ 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x^{\text{ego}} \\ y^{\text{ego}} \\ v^{\text{ego}} \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ 1 \end{bmatrix} u. \tag{4.5}$$

The ego vehicle is thus constrained to move along the vertical axis. A similar equation holds for the adversary vehicle, which is constrained to move along the horizontal axis. The dynamics can be discretized with an appropriate time step Δt. We assume the ego vehicle is initialized at the coordinates $(0, -1)$ and the adversary vehicle is initialized at $(-1, 0)$. We further assume all the units in this example follow the metric system. We would like to design a controller for the ego vehicle to satisfy an STL specification under some assumptions on the external environment, and provide diagnosis and feedback if the specification is infeasible. We discuss the following three scenarios.

Example 5 (Collision Avoidance)**.** *The specification is to avoid a collision between the ego and the adversary vehicle. We assume the adversary vehicle's acceleration is fixed at all times, i.e.,* $e_t = a_t^{\text{adv}} = 2$*, while the initial velocities are* $v_0^{\text{adv}} = 0$ *and* $v_0^{\text{ego}} = 0$*. We encode our requirements using the formula* $\varphi := \varphi_1 \wedge \varphi_2$*, where* φ_1 *and* φ_2 *are defined as follows:*

$$\begin{aligned} \varphi_1 &= \mathbf{G}_{[0,\infty)} \neg \big((-0.5 \leq y_t^{\text{ego}} \leq 0.5) \wedge (-0.5 \leq x_t^{\text{adv}} \leq 0.5) \big), \\ \varphi_{@} &= \mathbf{G}_{[0,\infty)} \big(1.5 \leq a_t^{\text{ego}} \leq 2.5 \big). \end{aligned} \tag{4.6}$$

We prescribe bounds on the system acceleration, and state that both cars should never be confined together within a box of width 1 *around the intersection* $(0, 0)$ *to avoid a collision.*

Example 6 (Non-adversarial Race)**.** *We discuss a race scenario, in which the* ego *vehicle must increase its velocity to exceed* 0.5 *whenever the adversary's initial velocity exceeds* 0.5*. We then formalize our requirement as a contract* $(\psi_e, \psi_e \to \psi_s)$*, where* ψ_e *are the assumptions made on the environment and* ψ_s *are the guarantees of the system provided the environment satisfies the assumptions. Specifically:*

$$\begin{aligned} \psi_e &= (v_0^{\text{adv}} \geq 0.5), \\ \psi_s &= \mathbf{G}_{[0,\infty)}(-1 \leq a_t^{\text{ego}} \leq 1) \wedge \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \geq 0.5). \end{aligned} \tag{4.7}$$

The initial velocities are $v_0^{\text{adv}} = 0.55$ *and* $v_0^{\text{ego}} = 0$*, while the environment vehicle's acceleration is* $e_t = a_t^{\text{adv}} = 1$ *at all times. We also require the acceleration to be bounded by* 1*.*

Example 7 (Adversarial Race)**.** *We discuss another race scenario, in which the environment vehicle acceleration* a_t^{adv} *is no longer fixed, but can vary up to a maximum value of* 2*. Initially,* $v_0^{\text{adv}} = 0$ *and* $v_0^{\text{ego}} = 0$ *hold. Under these assumptions, we would like to guarantee that the velocity of the* ego *vehicle exceeds* 0.5 *if the speed of the adversary vehicle exceeds* 0.5*, while*

maintaining an acceleration in the $[-1, 1]$ *range. Altogether, we capture the requirements above via a contract* $(\phi_w, \phi_w \rightarrow \phi_s)$*, where:*

$$\begin{aligned} \phi_w &= \mathbf{G}_{[0,\infty)}\big(0 \leq a_t^{\text{adv}} \leq 2\big), \\ \phi_s &= \mathbf{G}_{[0,\infty)}\big((v_t^{\text{adv}} > 0.5) \rightarrow (v_t^{\text{ego}} > 0.5)\big) \wedge \big(|a_t^{\text{ego}}| \leq 1\big). \end{aligned} \tag{4.8}$$

4.4 Problem Formulation

In this section, we define the problems of specification diagnosis and repair in the context of controller synthesis from STL. We assume that the discrete-time system dynamics f_d, the initial state x_0, the STL specification φ, and a cost function ρ_φ are given. The *controller synthesis* problem, denoted $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$, is to solve (4.1) (when φ is a monolithic specification of the desired system behavior) or (4.2) (when φ represents a contract between the system and the environment).

If synthesis fails, the *diagnosis* problem is, intuitively, to return an explanation in the form of a subset of the original problem constraints that are already infeasible when taken alone. The *repair* problem is to return a "minimal" set of changes to the specification that would render the resulting controller synthesis problem feasible. To diagnose and repair an STL formula, we focus on its sets of atomic predicates and time intervals of the temporal operators. We then start by providing a definition of the *support* of its atomic predicates, i.e., the set of times at which the value of a predicate affects satisfiability of the formula, and define the set of allowed repairs.

Definition 5 (Support)**.** *The* support *of a predicate* μ *in an STL formula* φ *is the set of times* t *such that* $\mu(\xi_{\mathcal{S}}(t))$ *appears in* φ*.*

For example, given $\varphi = \mathbf{G}_{[6,10]}(x_t > 0.2)$, the support of predicate $\mu = (x_t > 0.2)$ is the time interval $[6, 10]$. We can compute the support of each predicate in φ by traversing the parse tree of the formula from the root node to the leaves, which are associated with the atomic predicates. The support of the root of the formula is $\{0\}$ by definition. While parsing φ, new nodes are created and associated with the Boolean and temporal operators in the formula. Let κ and δ be the subsets of nodes associated with the Boolean and bounded temporal operators, respectively, where $\delta = \delta_{\mathbf{G}} \cup \delta_{\mathbf{F}} \cup \delta_{\mathbf{U}}$. The support of the predicates can then be computed by recursively applying the following rule for each node i in the parse tree:

$$\sigma_i = \begin{cases} \sigma_r \text{ if } r \in \kappa \\ \sigma_r + I_r \text{ if } r \in \delta_{\mathbf{G}} \cup \delta_{\mathbf{F}} \\ [\sigma_r^{lb}, \sigma_r^{ub} + I_r^{ub}] \text{ if } r \in \delta_{\mathbf{U}}, \end{cases} \tag{4.9}$$

where r is the parent of i, σ_r is the support of r, and I_i is the interval associated with i when i corresponds to a temporal operator. We denote as $I_1 + I_2$ the Minkowski sum of the sets I_1 and I_2, and as I^{lb} and I^{ub}, respectively, the lower and upped bounds of interval I.

Definition 6 (Allowed Repairs)**.** *Let Φ denote the set of all possible STL formulae. A* repair action *is a relation $\gamma : \Phi \to \Phi$ consisting of the union of the following:*

- *A* predicate repair *returns the original formula after modifying one of its atomic predicates μ to μ^*. We denote this sort of repair by $\varphi[\mu \mapsto \mu^*] \in \gamma(\varphi)$;*
- *A* time interval repair *returns the original formula after replacing the interval of a temporal operator. This is denoted $\varphi[\Delta_{[a,b]} \mapsto \Delta_{[a^*,b^*]}] \in \gamma(\varphi)$ where $\Delta \in \{\mathbf{G}, \mathbf{F}, \mathbf{U}\}$.*

=

Repair actions can be composed to get a *sequence of repairs* $\Gamma = \gamma_n(\gamma_{n-1}(\dots(\gamma_1(\varphi))\dots))$. Given an STL formula φ, we denote as $\texttt{REPAIR}(\varphi)$ the set of all possible formulae obtained through compositions of allowed repair actions on φ. Moreover, given a set of atomic predicates $\mathcal{D}$ and a set of time intervals $\mathcal{T}$, we use $\texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi) \subseteq \texttt{REPAIR}(\varphi)$ to denote the set of repair actions that act only on predicates in $\mathcal{D}$ or time intervals in $\mathcal{T}$. We are now ready to provide the formulation of the problems addressed in the paper, both in terms of diagnosis and repair of a *monolithic* specification φ (*general diagnosis and repair*) and an A/G contract $(\varphi_e, \varphi_e \to \varphi_s)$ (*contract diagnosis and repair*).

Problem 1 (General Diagnosis and Repair)**.** *Given a controller synthesis problem $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$ such that (4.1) is infeasible, find:*

- *A set of atomic predicates $\mathcal{D} = \{\mu_1, \dots, \mu_d\}$ or time intervals $\mathcal{T} = \{\tau_1, \dots, \tau_d\}$ of the original formula φ,*
- *$\varphi' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$,*

such that $\mathcal{P}' = (f_d, x_0, \varphi', \rho_{\varphi'})$ is feasible, and the following minimality conditions hold:

- (predicate minimality) *if φ' is obtained by predicate repair*[1]*, $s_i = \mu_i^* - \mu_i$ for $i \in \{1, \dots, d\}$, $s_\mathcal{D} = (s_1, \dots, s_d)$, and $||\cdot||$ is a norm on $\mathbb{R}^d$, then*

$$\nexists\, (\mathcal{D}', s_{\mathcal{D}'}) \quad \text{s.t.} \quad ||s_{\mathcal{D}'}|| \leq ||s_\mathcal{D}|| \tag{4.10}$$

  and $\mathcal{P}'' = (f_d, x_0, \varphi'', \rho_{\varphi''})$ is feasible, with $\varphi'' \in \texttt{REPAIR}_{\mathcal{D}'}(\varphi)$.

- (time interval minimality) *if φ' is obtained by time interval repair, $\mathcal{T}^* = \{\tau_1^*, \dots, \tau_l^*\}$ are the non-empty repaired intervals, and $||\tau||$ is the length of interval τ:*

$$\nexists\, \mathcal{T}' = \{\tau_1', \dots, \tau_l'\}, \text{ s.t. } \exists i \in \{1, \dots, l\}, ||\tau_i^*|| \leq ||\tau_i'|| \tag{4.11}$$

  and $\mathcal{P}'' = (f_d, x_0, \varphi'', \rho_{\varphi''})$ is feasible, with $\varphi'' \in \texttt{REPAIR}_{\mathcal{T}'}(\varphi)$.

[1]For technical reasons, our minimality conditions are predicated on a single type of repair being applied to obtain φ'.

Problem 2 (Contract Diagnosis and Repair)**.** *Given a controller synthesis problem* $\mathcal{P} = (f_d, x_0, \varphi \equiv \varphi_e \rightarrow \varphi_s, \rho_\varphi)$ *such that* (4.2) *is infeasible, find:*

- *Sets of atomic predicates* $\mathcal{D}_e = \{\mu_1^e, \ldots, \mu_d^e\}$, $\mathcal{D}_s = \{\mu_1^s, \ldots, \mu_{\bar{d}}^s\}$ *or sets of time intervals* $\mathcal{T}_e = \{\tau_1^e, \ldots, \tau_l^e\}$,$\mathcal{T}_s = \{\tau_1^s, \ldots, \tau_{\bar{l}}^s\}$, *respectively, of the original formulas* φ_e *and* φ_s,
- $\varphi_e' \in \texttt{REPAIR}_{\mathcal{T}_e,\mathcal{D}_e}(\varphi_e)$, $\varphi_s' \in \texttt{REPAIR}_{\mathcal{T}_s,\mathcal{D}_s}(\varphi_s)$,

such that $\mathcal{P}' = (f_d, x_0, \varphi', \rho_{\varphi'})$ *is feasible,* $\mathcal{D} = \mathcal{D}_e \cup \mathcal{D}_s$, $\mathcal{T} = \mathcal{T}_e \cup \mathcal{T}_s$, *and* $\varphi' \equiv \varphi_e' \rightarrow \varphi_s'$ *satisfies the minimality conditions of Problem* (1)*.*

In the following sections, we discuss our solution to the above problems.

4.5 Solution Approach

We extend the OGIS ([80]) framework to solve Problem 1 in Section 4.6.

The OGIS loop for Problem 1 is represented as $\mathcal{I}_{\varphi_M} = (\mathcal{L}_{\mathcal{I}_{\varphi_M}}, \mathcal{O}_{\mathcal{I}_{\varphi_M}})$. In the first iteration the learner solves (4.1) to synthesize a control sequence $\mathbf{u}$. If the learner is un-successful, the oracle *diagnoses* the specification and extracts a *Irreducibly Inconsistent System* (IIS) which explains the cause of infeasibility of (4.1). The oracle extracts a set of diagnosed atomic predicates $\mathcal{D}'$ and associated constraints I' in the optimization problem from the IIS and returns it to the learner. The learner then modifies the optimization problem in (4.1) by introducing ''slack" variables to the constraints returned by the oracle I'. If the modified optimization problem is feasible, then the learner returns a proposed repair $\varphi' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$ a solution to Problem 1 and a control strategy $\mathbf{u}$. If not, the learner sends the updated optimization problem $\mathcal{M}$ to the the oracle, which then diagnoses the modified optimization problem to propose a new IIS. This loop continues until the learner finds a repaired specification and $\mathbf{u}$. Moreover, the solution returned by the learner is a minimal repair $\varphi' \in \texttt{REPAIR}_{\mathcal{T},\mathcal{D}}(\varphi)$ to the original specification φ.

We next shows that with minor modifications to $\mathcal{I}_{\varphi_M}$, we can solve Problem 2 in non-adversarial settings.

To solve Problem 2 in adversarial settings, we propose a hierarchical CEGIS ([141]) framework represented as $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}})$. We detail this in Section 4.7.

4.6 Monolithic specifications

The overall OGIS framework $\mathcal{I}_{\varphi_M} = (\mathcal{L}_{\mathcal{I}_{\varphi_M}}, \mathcal{O}_{\mathcal{I}_{\varphi_M}}))$ for Problem 1 is shown in Figure 4.2. The OGIS loop iteratively diagnoses inconsistencies in the specification and provides constructive feedback to the designer.

The concept class for the learner $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$ here is the domain of all specifications. The oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$ here is deterministic mapping from MILP problems to infeasibility core.

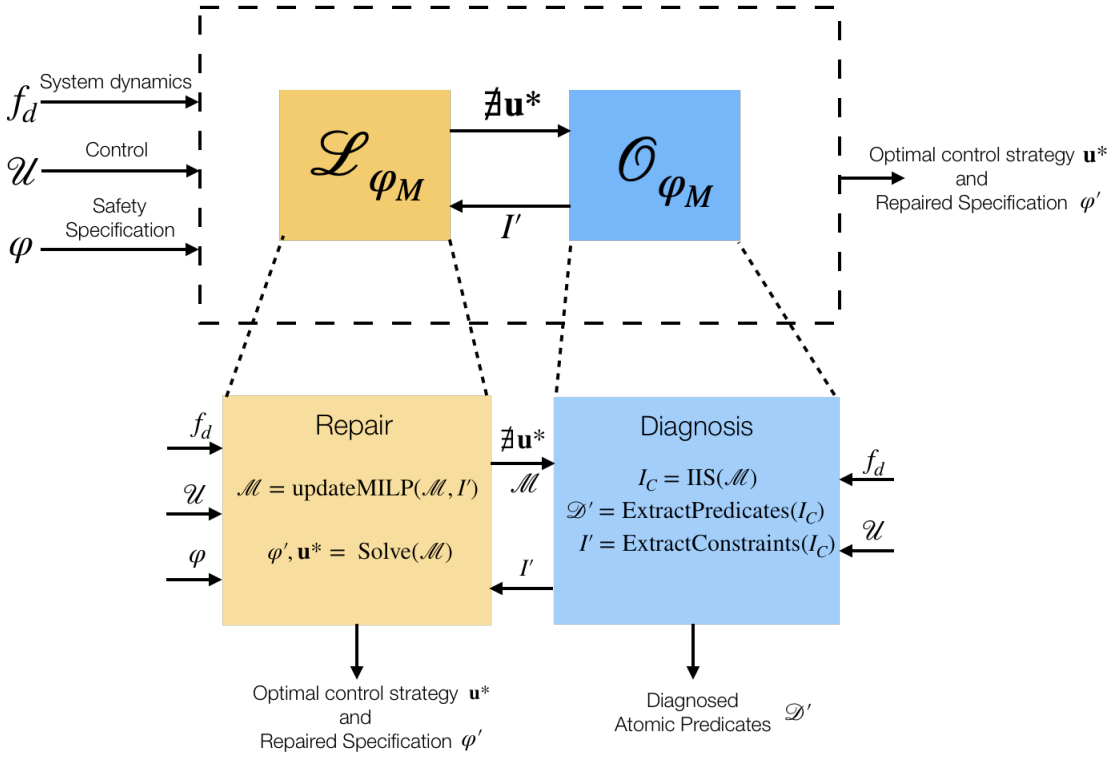


Figure 4.2. OGIS for Problem 1 $\mathcal{I}_{\varphi_M} = (\mathcal{L}_{\mathcal{I}_{\varphi_M}}, \mathcal{O}_{\mathcal{I}_{\varphi_M}})$. The oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$ ($\mathcal{L}_{\mathcal{I}_{\varphi_M}}$) is shown in blue (yellow). The oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$ diagnoses the infeasible control synthesis ($\nexists \mathbf{u}$) problem being solved by the learner. It first extracts the Irreducibly Inconsistent System (IIS) from the optimization problem $\mathcal{M}$ ($I_C = \text{IIS}(\mathcal{M})$). It then extracts the set of diagnosed atomic predicates $\mathcal{D}' = \text{ExtractPredicates}(I_C)$ and associated constraints $I' = \text{ExtractConstraints}(I_C)$ and returns it to the learner. The learner updates the MILP by introducing ''slack" variables to the constraints pertaining to the to the diagnosed predicates I'. It then solves the modified the updated optimization problem. If this is feasible, the learner returns the updated specification φ' and synthesized control strategy $\mathbf{u}$. If not, it sends the updated specification to the oracle and loop continues. The oracle returns a set of diagnosed predicates $\mathcal{D}'$ to the user at every iteration. By introducing the slack variables to the atomic predicates, we are guaranteed that the OGIS loop will terminate with a repaired specification φ'.

The overall OGIS is summarized in Algorithm 1. The Diagnosis procedure is implemented by the oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$ and the Repair and Solve procedures are implemented by the learner $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$.

Given a problem $\mathcal{P}$, defined as in Section 4.4, GenMILP reformulates (4.1) in terms of the following MILP:

$$\begin{aligned} \underset{\mathbf{u}}{\text{maximize}} \quad & \rho_\varphi(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})) \\ \text{subject to} \quad & f_i^d \leq 0 \quad i \in \{1, \ldots, m_d\} \\ & f_k^\varphi \leq 0 \quad k \in \{1, \ldots, m_s\}, \end{aligned} \tag{4.12}$$

where f^d and f^φ are mixed integer linear constraint functions over the states, outputs, and

Algorithm 1 OGIS $\mathcal{I}_{\varphi_M}$ for Diagnosis and Repair

procedure DIAGNOSEREPAIR($\mathcal{P}$)
  $(\rho_\varphi, C) \leftarrow$ GenMILP($\mathcal{P}$), $repaired \leftarrow 0$
  $\mathbf{u} \leftarrow$ Solve(ρ_φ, C) ▷ In $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$
  if $\mathbf{u} = \emptyset$ **then**
    $\mathcal{D} \leftarrow \emptyset$, $S \leftarrow \emptyset$, $I \leftarrow \emptyset$, $\mathcal{M} \leftarrow (0, C)$
    while $repaired = 0$ **do**
      $(\mathcal{D}', S', I') \leftarrow$ Diagnosis($\mathcal{M}$, $\mathcal{P}$) ▷ In $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$
      $\mathcal{D} \leftarrow \mathcal{D} \cup \mathcal{D}'$, $S \leftarrow S \cup S'$, $I \leftarrow I \cup I'$ ▷ In $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$
      $options \leftarrow$ UserInput($\mathcal{D}'$)
      $\lambda \leftarrow$ ModifyConstraints($I', options$)
      $(repaired, \mathcal{M}, \varphi') \leftarrow$ Repair($\mathcal{M}, I', \lambda, S, \varphi$) ▷ In $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$
    $\mathbf{u} \leftarrow$ Solve($\rho_{\varphi'}$, $\mathcal{M}.C$) ▷ In $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$
  return $\mathbf{u}, \mathcal{D}, repaired, \varphi'$

Algorithm 2 Diagnosis (Oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$)

procedure Diagnosis($\mathcal{M}$, $\mathcal{P}$)
  $I_C \leftarrow$ IIS($\mathcal{M}$)
  $(\mathcal{D}, S) \leftarrow$ ExtractPredicates(I_C,$\mathcal{P}$)
  $I' \leftarrow$ ExtractConstraints($\mathcal{M}$,$\mathcal{D}$)
  return $\mathcal{D}$, S, I'

inputs of the finite horizon trajectory $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u})$ associated, respectively, with the system dynamics and the STL specification φ. We let (ρ_φ, C) represent the MILP correcponding to the original specification φ, where ρ_φ is the robustness objective, and C is the set of constraints. If problem (4.12) is infeasible, we iterate between diagnosis and repair phases until the repaired feasible specification φ' is obtained. We let $\mathcal{D}$ and I denote, respectively, the set of predicates returned by the diagnosis procedure, and the constraints corresponding to those predicates.

Optionally, we support an interactive repair mechanism, where the designer provides a set of *options* that prioritize which predicates to modify (UserInput procedure) and get converted into a set of weights λ (ModifyConstraints routine). The designer can then leverage this weighted-cost variant of the problem to distinguish between "hard" constraints, i.e., constraints that should never be violated in the controller synthesis problem, and "soft" constraints, i.e., constraints whose violation or perturbation is admitted within a predefined margin. In the following, we detail the implementation of the Diagnosis and Repair routines.

4.6.1 Diagnosis

Our diagnosis procedure is implemented as the oracle $\mathcal{O}_{\mathcal{I}_{\varphi_M}}$ and summarized in Algorithm 2. Diagnosis receives as inputs the controller synthesis problem $\mathcal{P}$ and an associated MILP formulation $\mathcal{M}$. $\mathcal{M}$ can either be the *feasibility problem* corresponding to the original problem (4.12), or a relaxation of it. This feasibility problem has the same constraints as (4.12) (possibly relaxed) but constant cost. Formally, we provide the following definition of relaxed constraint and relaxed optimization problem.

Definition 7 (Relaxed Problem)**.** *We say that a constraint $f' \leq 0$ is a* relaxed version *of $f \leq 0$ if $f' = (f - s)$ for some slack variable $s \in \mathbb{R}^+$. In this case, we also say that $f \leq 0$ "is relaxed to" $f' \leq 0$. An optimization problem O' is a* relaxation *of another optimization problem O if it is obtained from O by relaxing at least one of its constraints.*

When $\mathcal{M}$ is infeasible, we rely on the capability of state-of-the-art MILP solvers to provide an *Irreducibly Inconsistent System* (IIS) [72, 26] of constraints I_C, defined as follows.

Definition 8 (Irreducibly Inconsistent System)**.** *Given a feasibility problem $\mathcal{M}$ with constraint set C, an* Irreducibly Inconsistent System *I_C is a subset of constraints $I_C \subseteq C$ such that: (i) the optimization problem $(0, I_C)$ is infeasible; (ii) $\forall\ c \in I_C$, problem $(0, I_C \setminus \{c\})$ is feasible.*

In other words, an IIS is an infeasible subset of constraints that becomes feasible if any single constraint is removed. For each constraint in I_C, ExtractPredicates traces back the STL predicate(s) originating it, which will be used to construct the set $\mathcal{D} = \{\mu_1, \ldots, \mu_d\}$ of STL atomic predicates in Problem 1, and the corresponding set of support intervals $S = \{\sigma_1, \ldots, \sigma_d\}$ (adequately truncated to the current horizon H) as obtained from the STL syntax tree. $\mathcal{D}$ will be used to produce a relaxed version of $\mathcal{M}$ as further detailed in Section 4.6.2. For this purpose, the procedure also returns the subset I of all the constraints in $\mathcal{M}$ that are associated with the predicates in $\mathcal{D}$. In our application, we process all the constraints in the IIS together.

4.6.2 Repair

The diagnosis procedure isolates a set of STL atomic predicates that jointly produce a reason of infeasibility for the synthesis problem. For repair, we are instead interested in how to modify the original formula to make the problem feasible. The repair procedure is implemented as part of the learner $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$ and summarized in Algorithm 3. We formulate relaxations of the feasibility problem $\mathcal{M}$ associated with problem (4.12) by using *slack variables*.

Algorithm 3 Repair (Learner $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$)

procedure Repair($\mathcal{M}$, I, λ, S, φ)
  $\mathcal{M}.\rho_{\varphi'} \leftarrow \mathcal{M}.\rho_{\varphi} + \lambda^{\top} s_I$
  for c in I **do**
    if $\lambda(c) > 0$ **then**
      $\mathcal{M}.C(c) \leftarrow \mathcal{M}.C(c) + s_c$
  ($repaired$, $\mathbf{s}^*$) $\leftarrow$ Solve($\mathcal{M}.\rho_{\varphi'}$, $\mathcal{M}.C$)
  if $repaired = 1$ **then**
    $\varphi' \leftarrow$ ExtractFeedback($\mathbf{s}^*$,S,φ)
  return $repaired$, $\mathcal{M}$, φ'

Let f_i, $i \in \{1, \ldots, m\}$ denote both of the categories of constraints f_{dyn} and f_{φ} in the feasibility problem $\mathcal{M}$. We reformulate $\mathcal{M}$ into the following *slack feasibility problem*:

$$\begin{aligned} \underset{\mathbf{s}\in\mathbb{R}^{|I|}}{\text{minimize}} \quad & ||\mathbf{s}|| \\ \text{subject to} \quad & f_i - s_i \leq 0 i \in \{1, \ldots, |I|\} \\ & f_i \leq 0 i \in \{|I|+1, \ldots, m\} \\ & s_i \geq 0 i \in \{1, \ldots, |I|\}, \end{aligned} \tag{4.13}$$

where $\mathbf{s} = s_1 ... s_{|I|}$ is a vector of slack variables corresponding to the subset of optimization constraints I, as obtained after the latest call of Diagnosis. Not all the constraints in the original optimization problem (4.12) can be modified. For instance, the designer will not be able to arbitrarily modify constraints that can directly affect the dynamics of the system, i.e., constraints encoded in f^{dyn}. Solving problem (4.13) is equivalent to looking for a set of slacks that make the original control problem feasible while minimizing a suitable norm $||\cdot||$ of the slack vector. In most of our application examples, we choose the l_1-norm, which tends to provide sparser solutions for $\mathbf{s}$, i.e., nonzero slacks for a smaller number of constraints. However, other norms can also be used, including weighted norms based on the set of weights λ. If problem (4.13) is feasible, ExtractFeedback uses the solution $\mathbf{s}^*$ to repair the original infeasible specification φ. Otherwise, the infeasible problem is subjected to another round of diagnosis to retrieve further constraints to relax. In what follows, we provide details on the implementation of ExtractFeedback.

Based on the encoding discussed in Section 4.2.2, the constraints in $\mathcal{M}$ capture, in a recursive fashion, the robust satisfaction of the STL specification as a function of its subformulae, from φ itself to the atomic predicates μ_i. To guarantee satisfaction of a Boolean operation in φ at time t, we must be able to perturb, in general, all the constraints associated with its operands, i.e., the children nodes of the corresponding Boolean operator in the parse tree of φ, at time t. Similarly, to guarantee satisfaction of a temporal construct at time t, we must be able to perturb the constraints associated with the operands of the corresponding operator at all times in their support. By recursively applying this line of reasoning, we

can then conclude that, to guarantee satisfaction of φ, it is sufficient to introduce slacks to all the constraints associated with all the diagnosed predicates in $\mathcal{D}$ over their entire support. For each $\mu_i \in \mathcal{D}$, $i \in \{1, \ldots, d\}$, let $\sigma_i = [\sigma_i^{lb}, \sigma_i^{ub}]$ be its support interval. The set of slack variables $\{s_1, \ldots, s_{|I|}\}$ in (4.13) can then be seen as the set of variables $s_{\mu_i,t}$ used to relax the constraints corresponding to each diagnosed predicate $\mu_i \in \mathcal{D}$ at time t, for all $t \in \{\max\{0, \sigma_i^{lb}\}, \ldots, \min\{H-1, \sigma_i^{ub}\}\}$ and $i \in \{1, \ldots, d\}$.

If a minimum norm solution $\mathbf{s}^*$ is found for (4.13), then the slack variables $\mathbf{s}^*$ can be mapped to a set of *predicate repairs* $s_{\mathcal{D}}$, as defined in Problem 1, as follows. The slack vector $\mathbf{s}^*$ in Algorithm 3 consists of the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t, $i \in \{1, \ldots, d\}$. We set

$$\forall\ i \in \{1, \ldots, d\}\ s_i = \mu_i^* - \mu_i = \max_{t \in \{\sigma_{i,l}, \cdots, \sigma_{i,u}\}} s^*_{\mu_i,t}, \tag{4.14}$$

where H is the time horizon for (4.12), $s_{\mathcal{D}} = \{s_1, \ldots, s_d\}$, $\sigma_{i,l} = \max\{0, \sigma_i^{lb}\}$, and $\sigma_{i,u} = \min\{H-1, \sigma_i^{ub}\}$.

To find a set of *time-interval repairs*, we proceed, instead, as follows:

1. The slack vector $\mathbf{s}^*$ in Algorithm 3 consists of the set of slack variables $\{s^*_{\mu_i,t}\}$, where $s^*_{\mu_i,t}$ is the variable added to the optimization constraint associated with an atomic predicate $\mu_i \in \mathcal{D}$ at time t. For each $\mu_i \in \mathcal{D}$, with support interval σ_i, we search for the largest time interval $\sigma_i' \subseteq \sigma_i$ such that the slack variables $s^*_{\mu_i,t}$ for $t \in \sigma_i'$ are 0. If $\mu_i \notin \mathcal{D}$, then we set $\sigma_i' = \sigma_i$.

2. We convert every temporal operator in φ into a combination of $\mathbf{G}$ (timed or untimed) and untimed $\mathbf{U}$ by using the following transformations:

$$\mathbf{F}_{[a,b]}\psi = \neg\mathbf{G}_{[a,b]}\neg\psi,$$

$$\psi_1 \mathbf{U}_{[a,b]} \psi_2 = \mathbf{G}_{[0,a]}(\psi_1 \mathbf{U}\ \psi_2) \wedge \mathbf{F}_{[a,b]}\psi_2,$$

   where $\mathbf{U}$ is the untimed (unbounded) *until* operator. Let $\hat{\varphi}$ be the new formula obtained from φ after applying these transformations[2].

3. The nodes of the parse tree of $\hat{\varphi}$ can then be partitioned into three subsets, ν, κ, and δ, respectively associated with the *atomic predicates*, *Boolean operators*, and *temporal operators* ($\mathbf{G}, \mathbf{U}$) in $\hat{\varphi}$. We traverse this parse tree from the leaves (atomic predicates) to the root and recursively define for each node i a new support interval σ_i^* as follows:

$$\sigma_i^* = \begin{cases} \sigma_i' & \text{if } i \in \nu \\ \bigcap\limits_{j \in C(i)} \sigma_j^* & \text{if } i \in \kappa \cup \delta_{\mathbf{U}} \\ \sigma^*_{C(i)} & \text{if } i \in \delta_{\mathbf{G}} \end{cases} \tag{4.15}$$

[2]While the second transformation introduces a new interval $[0, a]$, its parameters are directly linked to the ones of the original interval $[a, b]$ (now inherited by the $\mathbf{F}$ operator) and will be accordingly processed by the repair routine.

where $C(i)$ denotes the set of children of node i, while $\delta_{\mathbf{G}}$ and $\delta_{\mathbf{U}}$ are, respectively, the subsets of nodes associated with the **G** and **U** operators. We observe that the set of children for a **G** operator node is a singleton. Therefore, with some abuse of notation, we also use $C(i)$ in (4.15) to denote a single node in the parse tree.

4. We define the interval repair $\hat{\tau}_j$ for each (timed) temporal operator node j in the parse tree of $\hat{\varphi}$ as $\hat{\tau}_j = \sigma_j^*$. If $\hat{\tau}_j$ is empty for some j, no time-interval repair is possible. Otherwise, we map back the set of intervals $\{\hat{\tau}_j\}$ into a set of interval repairs $\mathcal{T}^*$ for the original formula φ according to the transformations in step 2 and return $\mathcal{T}^*$. We do not define a interval repair $\hat{\tau}$ for a boolean operators, since these operators are stateless. An empty support for a temporal operator implies there are no interval repair for temporal operators. An empty support for boolean operators, propagates an empty support to its parent and so on.

We provide an example of predicate repair below, while time interval repair is exemplified in Section 4.7.1.

Example 8 (Collision Avoidance)**.** *We diagnose the specifications introduced in Example 5. To formulate the synthesis problem, we assume a horizon $H = 10$ and a discretization step $\Delta t = 0.2$. The system is found infeasible at the first MPC run, and* Diagnosis *detects the infeasibility of $\varphi_1 \wedge \varphi_2$ at time $t = 6$. Intuitively, given the allowed range of accelerations for the ego vehicle, both cars end up entering the forbidden box at the same time. Algorithm 1 chooses to repair φ_1 by adding slacks to all of its predicates, such that $\varphi_1' = (-0.5 - s_{l1} \leq y_t^{\text{ego}} \leq 0.5 + s_{u1}) \wedge (-0.5 - s_{l2} \leq x_t^{\text{adv}} \leq 0.5 + s_{u2})$. Table 4.1 shows the optimal slack values at each t, while s_{u1} and s_{l2} are set to zero at all t. We can then conclude that the specification replacing φ_1 with φ_1'*

$$\varphi_1' = \mathbf{G}_{[0,\infty)} \neg \big((-0.24 \leq y_t^{\text{ego}} \leq 0.5) \wedge (-0.5 \leq x_t^{\text{adv}} \leq 0.43) \big) \tag{4.16}$$

is feasible, i.e., the cars will not collide, but the original requirement was overly demanding.

Alternatively, the user can choose to run the repair procedure on φ_2 and change its predicate as $(1.5 - s_l \leq a_t^{\text{ego}} \leq 2.5 + s_u)$. In this case, we decide to stick with the original requirement on collision avoidance, and tune, instead, the control "effort" to satisfy it. Under the assumption of constant acceleration (and bounds), the slacks will be the same at all t. We then obtain $[s_l, s_u] = [0.82, 0]$, which ultimately turns into $\varphi_2' = \mathbf{G}_{[0,\infty)} \big(0.68 \leq a_t^{mathrmego} \leq 2.5 \big)$. The ego *vehicle should then slow down to prevent entering the forbidden box at the same time as the other car. This latter solution is, however, suboptimal with respect to the l_1-norm selected in this example when both repairs are allowed.*

Our algorithm offers the following guarantees, for which a proof is reported below.

Theorem 4 (Soundness)**.** *Given a controller synthesis problem $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$, such that (4.1) is infeasible at time t, let $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ be the repaired formula returned from Algorithm 1 without human intervention, for a given set of predicates $\mathcal{D}$ or time interval*

time	0	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_{l1}	0	0	0	0	0	-0.26	0	0	0	0
s_{u2}	0	0	0	0	0	0	-0.07	0	0	0

Table 4.1. Slack values over a single horizon, for $\Delta t = 0.2$ and $H = 10$.

$\mathcal{T}$. Then, $\mathcal{P}' = (f_d, x_0, \varphi', \rho_{\varphi'})$ is feasible at time t and (φ', $\mathcal{D}$, $\mathcal{T}$) satisfy the minimality conditions in Problem 1.

Proof: [Proof (Theorem 4)] Suppose $\mathcal{M}$ is the MILP encoding of $\mathcal{P}$ as defined in (4.12), φ' is the repaired formula, and $\mathcal{D}$ the set of diagnosed predicates, as returned by Algorithm 1. We start by discussing the case of predicate repair.

We let $\mathcal{M}'$ be the MILP encoding of $\mathcal{P}'$ and $\mathcal{D}^* \subseteq \mathcal{D}$ be the set of predicates that are fixed to provide φ', i.e., such that $s = (\mu^* - \mu) \neq 0$, with $\mu \in \mathcal{D}$. Algorithm 1 modifies $\mathcal{M}$ by introducing a slack variable $s_{\mu,t}$ into each constraint associated with an atomic predicate μ in $\mathcal{D}$ at time t. Such a transformation leads to a feasible MILP $\mathcal{M}''$ and an optimal slack set $\{s^*_{\mu,t} | \mu \in \mathcal{D}, t \in \{0, \ldots, H-1\}\}$. We now observe that $\mathcal{M}'$ and $\mathcal{M}''$ are both relaxations of $\mathcal{M}$. In fact, we can view $\mathcal{M}'$ as a version of $\mathcal{M}$ in which only the constraints associated with the atomic predicates in $\mathcal{D}^*$ are relaxed. Therefore, each constraint having a nonzero slack variable in $\mathcal{M}''$ is also relaxed in $\mathcal{M}'$. Moreover, by (4.14), the relaxed constraints in $\mathcal{M}'$ are offset by the largest slack value over the horizon H. Then, because $\mathcal{M}''$ is feasible, $\mathcal{M}'$, and subsequently $\mathcal{P}'$, are feasible.

We now prove that (φ',$\mathcal{D}$) satisfy the predicate minimality condition of Problem 1. Let $\tilde{\varphi}$ be any formula obtained from φ after repairing a set of predicates $\tilde{\mathcal{D}}$ such that the resulting problem $\tilde{\mathcal{P}}$ is feasible. We recall that, by Definition 8, at least one predicate in $\mathcal{D}$ generates a conflicting constraint and must be repaired for $\mathcal{M}$ to become feasible. Then, $\tilde{\mathcal{D}} \cap \mathcal{D} \neq \emptyset$ holds. Furthermore, since Algorithm 1 iterates by diagnosing and relaxing constraints until feasibility is achieved, $\mathcal{D}$ contains all the predicates that can be responsible for the infeasibility of φ. In other words, Algorithm 1 finds all the IISs in the original optimization problem and allows relaxing any constraint in the union of the IISs. Therefore, repairing any predicate outside of $\mathcal{D}$ is redundant: a predicate repair set that only relaxes the constraints associated with predicates in $\bar{\mathcal{D}} = \tilde{\mathcal{D}} \cap \mathcal{D}$, by the same amount as in $\tilde{\varphi}$, and sets to zero the slack variables associated with predicates in $\mathcal{D} \setminus \bar{\mathcal{D}}$ is also effective and exhibits a smaller slack norm. Let $s_{\bar{\mathcal{D}}}$ be such a repair set and $\bar{\varphi}$ the corresponding repaired formula. $s_{\bar{\mathcal{D}}}$ and $s_{\mathcal{D}}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (4.13), we are guaranteed that $s_{\mathcal{D}}$ has minimum norm; then, $||s_{\mathcal{D}}|| \leq ||s_{\bar{\mathcal{D}}}||$ will hold for any such formulas $\bar{\varphi}$, and hence $\tilde{\varphi}$.

We now consider the MILP formulation $\mathcal{M}'$ associated with $\mathcal{P}'$ and φ' in the case of time-interval repairs. For each atomic predicate $\mu_i \in \mathcal{D}$, for $i \in \{1, \ldots, |\mathcal{D}|\}$, $\mathcal{M}'$ includes only the associated constraints evaluated over time intervals σ'_i for which the slack variables $\{s_{\mu_i,t}\}$ are zero. Such a subset of constraints is trivially feasible. All the other constraints

enforcing the satisfaction of Boolean and temporal combinations of the atomic predicates in φ' cannot cause infeasibility with these atomic predicate constraints, or the associated slack variables $\{s_{\mu_i,t}\}$ would be non-zero. So, $\mathcal{M}'$ is feasible.

To show that (φ',$\mathcal{T}$) satisfy the minimality condition in Problem 1, we observe that, by the transformations in step 2 of the time-interval repair procedure, φ is logically equivalent to a formula $\hat{\varphi}$ which only contains *untimed* **U** and *timed* **G** operators. Moreover, $\hat{\varphi}$ and φ have the same interval parameters. Therefore, if the proposed repair set is minimal for $\hat{\varphi}$, this will also be the case for φ. We now observe that Algorithm 1 selects, for each atomic predicate $\mu_i \in \mathcal{D}$ the largest interval σ_i' such that the associated constraints are feasible, i.e., their slack variables are zero after norm minimization[3]. Because feasible intervals for Boolean combinations of atomic predicates are obtained by intersecting these maximal intervals, and then propagated to the temporal operators, the length of the intervals of each **G** operator in $\hat{\varphi}$, hence of the temporal operators in φ, will also be maximal. □

Theorem 5 (Completeness)**.** *Assume the controller synthesis problem $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$ results in* (4.1) *being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ or time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq$ `REPAIR`$_{\mathcal{D},\mathcal{T}}(\varphi)$ for which $\forall\ \phi \in \Phi$, $\mathcal{P}' = (f_d, x_0, \phi, \rho_\phi)$ is feasible at time t and (ϕ, $\mathcal{D}$, $\mathcal{T}$) are minimal in the sense of Problem 1, then Algorithm 1 returns a repaired formula φ' in Φ.*

Proof: [Proof (Theorem 5)] We first observe that Algorithm 1 always terminates with a feasible solution φ' since the set of MILP constraints to diagnose and repair is finite. We first consider the case of predicate repairs. Let $\mathcal{D}$ be the set of predicates modified to obtain $\phi \in \Phi$ and $\mathcal{D}'$ the set of diagnosed predicates returned by Algorithm 1. Then, by Definition 8 and the iterative approach of Algorithm 1, we are guaranteed that $\mathcal{D}'$ includes all the predicates responsible for inconsistencies, as also argued in the proof of Theorem 4. Therefore, we conclude $\mathcal{D} \subseteq \mathcal{D}'$. $s_{\mathcal{D}}$ and $s_{\mathcal{D}'}$ can then be seen as two repair sets on the same predicate set. However, by the solution of Problem (4.13), we are guaranteed that $s_{\mathcal{D}'}$ has minimum norm; then, $||s_{\mathcal{D}'}|| \leq ||s_{\mathcal{D}}||$ will hold, hence $\varphi' \in \Phi$.

We now consider the case of time-interval repair. If a formula $\phi \in \Phi$ repairs a set of intervals $\mathcal{T} = \{\tau_1, \ldots, \tau_l\}$, then there exists a set of constraints associated with atomic predicates in φ which are consistent in $\mathcal{M}$, the MILP encoding associated with ϕ, and make the overall problem feasible. Then, the relaxed MILP encoding $\mathcal{M}'$ associated with φ after slack norm minimization will also include a set of predicate constraints admitting zero slacks over the same set of time intervals as in $\mathcal{M}$, as determined by $\mathcal{T}$. Since these constraints are enough to make the entire problem $\mathcal{M}$ feasible, this will also be the case for $\mathcal{M}'$. Therefore, our procedure for time-interval repair terminates and produces a set of non-empty intervals $\mathcal{T}' = \{\tau_1', \ldots, \tau_l'\}$. Finally, because Algorithm 1 finds the longest intervals for which the slack variables associated with each atomic predicate are zero, we are also guaranteed that

[3]Because we are not directly maximizing the sparsity of the slack vector, time-interval minimality is to be interpreted with respect to slack norm minimization. Directly maximizing the number of zero slacks is also possible but computationally more intensive.

$||\tau_i'|| \geq ||\tau_i||$ for all $i \in \{1, \ldots, l\}$, as also argued in the proof of Theorem 4. We can then conclude that $\varphi' \in \Phi$ holds. □

In the worst case, Algorithm 1 solves a number of MILP problem instances equal to the number of atomic predicates in the STL formula. While the complexity of solving a MILP is NP-hard, the actual runtime depends on the size of the MILP, which is $O(H \cdot |\varphi|)$, where H is the length of the horizon and $|\varphi|$ is the number of predicates and operators in the STL specification.

Remark 3. *We depend on the predicate repairs to compute the time interval repair. In this work, out repairs are such that they are either purely predicate repairs or time interval repairs. We would like to extend this to compute a pareto front of repairs (repairs which have both predicate and time interval repairs).*

4.7 Contract specifications

In this section, we consider specifications provided in the form of a contract $(\varphi_e, \varphi_e \rightarrow \varphi_s)$, where φ_e is an STL formula expressing the assumptions, i.e., the set of behaviors assumed from the environment, while φ_s captures the guarantees, i.e., the behaviors promised by the system in the context of the environment. To repair contracts, we can capture tradeoffs between assumptions and guarantees in terms of minimization of a weighted norm of slacks. We describe below our results for both non-adversarial and adversarial environments.

We first describe the OGIS framework for the non-adversarial environments and show that it can be solved with a small modification to $\mathcal{I}_{\varphi_M}$. We then describe the overall CEGIS framework $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}}))$ for Problem 2 in adversarial setting.

4.7.1 Non-Adversarial Environment

For a contract, we make a distinction between controlled inputs u_t and uncontrolled (environment) inputs e_t of the dynamical system. In this section we assume that the environment signal $\mathbf{e}$ can be predicted over a finite horizon and set to a known value for which the controller must be synthesized. With $\varphi \equiv \varphi_e \rightarrow \varphi_s$, (4.2) reduces to:

$$\begin{aligned} \underset{\mathbf{u}}{\text{maximize}} \quad & \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e}) \\ \text{subject to} \quad & \rho_{\varphi}(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}, \mathbf{e})) > 0. \end{aligned} \tag{4.17}$$

Because of the similarity of Problem (4.17) and Problem (4.1), we can then diagnose and repair a contract using the methodology illustrated in Section 4.6. However, to reflect the different structure of the specification, i.e., its partition into assumption and guarantees, we adopt a weighted sum of the slack variables in Algorithm 1, allocating different weights to predicates in the assumption and guarantee formulae. Hence, by modifying the learner $\mathcal{L}_{\mathcal{I}_{\varphi_M}}$ to now solve for the weighted sum of the slack variables we can reuse $\mathcal{I}_{\varphi_M}$ to solve Problem 2 in non-adversarial settings. We can then provide the same guarantees as in Theorems 4

time	0.2	0.4	0.6	0.8	1	1.2	1.4	1.6	1.8
s_s	0.31	0.11	0	0	0	0	0	0	0

Table 4.2. Slack variables used in Example 6 and 9.

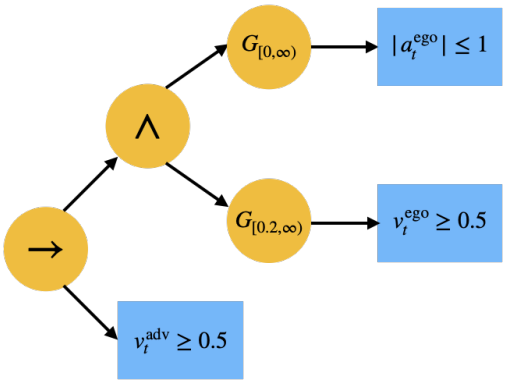


Figure 4.3. Parse tree of $\psi \equiv \psi_e \rightarrow \psi_s$ used in Example 6 and 9.

and 5, where $\varphi \equiv \varphi_e \rightarrow \varphi_s$ and the minimality conditions are stated with respect to the weighted norm.

Example 9 (Non-adversarial Race)**.** *We consider Example 6 with the same discretization step* $\Delta t = 0.2$ *and horizon* $H = 10$ *as in Example 5. The MPC scheme results infeasible at time* 1*. In fact, we observe that* ψ_e *is true as* $v_0^{\text{adv}} \geq 0.5$*. Since* $v_1^{\text{ego}} = 0.2$*, the predicate* $\psi_{s2} = \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \geq 0.5)$ *in* ψ_s *is found to be failing. As in Section 4.6.2, we can modify the conflicting predicates in the specification by using slack variables as follows:* $v_t^{\text{adv}} + s_e(t) \geq 0.5$ *(assumptions) and* $v_t^{\text{ego}} + s_s(t) \geq 0.5$ *(guarantees). However, we also assign a set of weights to the assumption* (λ_e) *and guarantee* (λ_s) *predicates, our objective being* $\lambda_e|s_e| + \lambda_s|s_s|$*. By setting* $\lambda_s > \lambda_e$*, we encourage modifications in the assumption predicate, thus obtaining* $s_e = 0.06$ *at time* 0 *and zero otherwise, and* $s_s = 0$ *at all times. We can then set* $\psi_e' = (v_0^{\text{adv}} \geq 0.56)$*, which falsifies* ψ_e' *so that* $\psi_e' \rightarrow \psi_s$ *is satisfied. Alternatively, by setting* $\lambda_s < \lambda_e$*, we obtain the slack values in Table 4.2, which lead to the following predicate repair:* $\psi_{s2}' = \mathbf{G}_{[0.2,\infty)}(v_t^{\text{ego}} \geq 0.2)$*.*

We can also modify the time interval of the temporal operator associated with ψ_{s2} *to repair the overall specification. To do so, Algorithm 1 uses the parse tree of* $\psi_e \rightarrow \psi_s$ *in Figure 4.3. For any of the leaf node predicates* μ_i*,* $i \in \{1, 2, 3\}$*, we get a support* $\sigma_i = [0, 9]$*, which is only limited by the finite horizon* H*. Then, based on the slack values in Table 4.2, we can conclude* $\sigma_1' = \sigma_2' = [0, 9]$ *(the optimal slack values for these predicates are always zero), while* $\sigma_3' = [3, 9]$*. For the given syntax tree, we also have* $\sigma_1^* = \sigma_1'$*,* $\sigma_2^* = \sigma_2'$*, and* $\sigma_3^* = \sigma_3'$ *for the temporal operator nodes that are parent nodes of* μ_1*,* μ_2*, and* μ_3*, respectively. Since none of the above intervals is empty, a time interval repair is indeed possible by modifying the time interval of the parent node of* μ_3*, thus achieving* $\tau_3^* = \sigma_3^*$*. This leads to the following*

Algorithm 4 CEGIS $\mathcal{C}_{\varphi_C}$ for Diagnosis and Repair in an adversarial setting

procedure DiagnoseRepairAdversarial($\mathcal{P}$)
  $(\rho_\varphi, C) \leftarrow$ GenMILP($\mathcal{P}$)
  $(\mathbf{u}, \mathbf{e}, sat) \leftarrow$ CheckSAT(ρ_φ, C)
  if *sat* **then**
    $\mathcal{E}^*_{CE} \leftarrow$ SolveCEGIS$(\mathbf{u}, \mathcal{P})$
    $\mathcal{E}_{CE} \leftarrow \mathcal{E}^*_{CE}$
    while $\mathcal{E}_{CE} \neq \emptyset$ **do**
      $\mathcal{P}_w \leftarrow$ RepairAdversarial$(\mathcal{E}_{CE}, \mathcal{P})$ ▷ In $\mathcal{L}_{\mathcal{C}_{\varphi_C}}$
      $\mathcal{E}_{CE} \leftarrow$ SolveCEGIS$(\mathbf{u}, \mathcal{P}_w)$ ▷ In $\mathcal{O}_{\mathcal{C}_{\varphi_C}}$
    $\mathcal{E}_{CE} \leftarrow \mathcal{E}^*_{CE}$, $\mathcal{P}_\psi \leftarrow \mathcal{P}$
    while $\mathcal{E}_{CE} \neq \emptyset$ **do**
      $\mathcal{P}_\psi \leftarrow$ DiagnoseRepair$(\mathcal{P}_\psi)$
      $\mathcal{E}_{CE} \leftarrow$ SolveCEGIS$(\mathbf{u}, \mathcal{P}_\psi)$
    $\mathcal{P}' \leftarrow$ FindMin$(\mathcal{P}_w, \mathcal{P}_\psi)$
  return $\mathbf{u}$, $\mathcal{P}'$

proposed sub-formula $\psi'_{s2} = \mathbf{G}_{[0.6,\infty)}(v_t^{\text{ego}} \geq 0.5)$. *In this example, repairing the specification over the first horizon is enough to guarantee controller realizability in the future. We can then keep the upper bound of the* $\mathbf{G}$ *operator to infinity..*

4.7.2 Adversarial Environment

When the environment can behave adversarially, the control synthesis problem assumes the structure in (4.2). Specifically, in this section, we allow e_t to lie in an interval $[e_{\min}, e_{\max}]$ at all times; this corresponds to the STL formula $\varphi_w = \mathbf{G}_{[0,\infty)}(e_{\min} \leq e_t \leq e_{\max})$. We decompose a specification φ of the form $\varphi_w \wedge \varphi_e \rightarrow \varphi_s$, representing the contract, as $\varphi \equiv \varphi_w \rightarrow \psi$, where $\psi \equiv (\varphi_e \rightarrow \varphi_s)$. To solve the repair problem for specifications with environment assumptions φ_w, we propose a CEGIS framework $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}})$. The concept class for the learner $\mathcal{L}_{\mathcal{C}_{\varphi_C}}$ is the set of all environment assumptions, i.e, φ_w. The oracle accepts as input a proposed environment assumption φ_w, the oracle returns a counter-example set $\mathcal{E}_{CE}$ such that $\forall e \in \mathcal{E}_{CE},\ e \models \varphi_w$ and the control synthesis problem cannot find a robust control. To solve the robust control problem within the oracle, the oracle itself is implements as an instance of a CEGIs framework similar to Chapter 3. Our diagnosis and repair method is summarized in Algorithm 4.

We first check the satisfiability of the control synthesis problem by examining whether

there exists a pair of $\mathbf{u}$ and $\mathbf{e}$ for which problem (4.2) is feasible (CheckSAT routine):

$$\begin{aligned} \underset{\mathbf{u},\mathbf{e}}{\text{maximize}} \quad & \rho_\varphi(\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}, \mathbf{e})) \\ \text{subject to} \quad & \rho_\varphi(\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}, \mathbf{e})) > 0 \\ & \mathbf{e} \models \varphi_w \wedge \varphi_e. \end{aligned} \tag{4.18}$$

If problem (4.18) is unsatisfiable, we can use the techniques introduced in Section 4.6 and 4.7.1 to diagnose and repair the infeasibility. Therefore, in the following, we assume that (4.18) is satisfiable, hence there exist $\mathbf{u}_0$ and $\mathbf{e}_0$ that solve (4.18) where $\mathbf{u}_i(\mathbf{e}_i)$ is the control (disturbance) sequence in iteration i in the CEGIS.

If problem (4.18) is satisfiable, we use the CEGIS framework $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}})$. The learner $\mathcal{L}_{\mathcal{C}_{\varphi_C}}$ *repairs* φ_w and proposes a candidate modification to the oracle. The oracle $\mathcal{O}_{\mathcal{C}_{\varphi_C}}$ accepts an input the candidate specification φ_w and checks the realizability of the synthesis process with the proposed φ_w and either it finds a dominant control strategy $\mathbf{u}$ or proposes a list of counterexamples $\mathcal{E}_{CE}$.

To check realizability, we can use any of the CEGIS loops $(\mathcal{C}_B, \mathcal{C}_S, \mathcal{C}_D, \mathcal{C}_H)$ presented in Chapter 3. Hence, the oracle $\mathcal{O}_{\mathcal{C}_{\varphi_C}}$ (SolveCEGIS routine) is a CEGIS loop. In this chapter, we build on top of CEGIS loop proposed in [128], with minor modifications. The overall CEGIS framework $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}})$ is, hence, a *hierarchical* CEGIS framework, where we have a CEGIS loop within another.

We first describe the design of the oracle $\mathcal{O}_{\mathcal{C}_{\varphi_C}}$ i.e., inner CEGIS loop. We refer to this as $\mathcal{C}_e = (\mathcal{L}_{\mathcal{C}_e}, \mathcal{O}_{\mathcal{C}_e})$. In iteration i, the learner $\mathcal{L}_{\mathcal{C}_e}$ solves for a dominant strategy $\mathbf{u}_i$ solving (4.17). The oracle $\mathcal{O}_{\mathcal{C}_e}$ proposes a counter-example to $\mathbf{u}_i$, $\mathbf{e}_i$ by solving (4.18) By first fixing the control trajectory to $\mathbf{u}_0$, we find the worst case disturbance trajectory $\mathbf{e}_1$ that minimizes the robustness value of φ by solving the following problem:

$$\begin{aligned} \underset{\mathbf{e}}{\text{minimize}} \quad & \rho_\varphi(\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}, \mathbf{e})) \\ \text{subject to} \quad & \mathbf{e} \models \varphi_e \wedge \varphi_w \end{aligned} \tag{4.19}$$

with $\mathbf{u} = \mathbf{u}_0$. If the robustness value pertaining to $\mathbf{e}_1$ is greater than zero, then we can terminate the CEGIS loop and $\mathbf{u}_0$ is a dominant control strategy. The optimal $\mathbf{e}_1$ from (4.19) will falsify the specification if the resulting robustness value is below zero[4].

If this is the case, we look for a $\mathbf{u}_1$ which solves (4.17) with the additional restriction of $\mathbf{e} \in \mathcal{E}_{CE} = \{\mathbf{e}_1\}$. If this step is feasible, we once again attempt to find a worst-case disturbance sequence $\mathbf{e}_2$ that solves (4.19) with $\mathbf{u} = \mathbf{u}_1$: this is the counterexample-guided inductive step. At each iteration i of this CEGIS loop, the set of candidate disturbance sequences $\mathcal{E}_{CE}$ expands to include $\mathbf{e}_i$. If the loop terminates at iteration i with a successful $\mathbf{u}_i$ (one for which the worst case disturbance $\mathbf{e}_i$ in (4.19) has positive robustness), we conclude that the formula φ is realizable.

[4]A tolerance $\rho_{\min}$ can be selected to accommodate approximation errors, i.e., $\rho_\varphi(\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}, \mathbf{e})) < \rho_{\min}$.

The CEGIS loop may not terminate if the set $\mathcal{E}_{CE}$ is infinite. We, therefore, run it for a maximum number of iterations. If SolveCEGIS fails to find a control sequence prior to the timeout, then (4.17) is infeasible for the current $\mathcal{E}_{CE}$, i.e., there is no control input that can satisfy φ for all disturbances in $\mathcal{E}_{CE}$. We conclude that the specification is not realizable (or, equivalently, the contract is inconsistent).

While this infeasibility can be repaired by modifying ψ based on the techniques in Section 4.6 and 4.7.1, an alternative solution is to repair φ_w by minimally pruning the bounds on e_t. The learner $\mathcal{L}_{\mathcal{C}_{\varphi_C}}$ implements this repair in the RepairAdversarial routine.

To do so, given a small tolerance $\epsilon \in \mathbb{R}^+$, we find

$$\begin{aligned} e_u &= \max_{\substack{\mathbf{e}_i \in \mathcal{E}_{CE} \\ t \in \{0,\dots,H-1\}}} e_{i,t} \\ e_l &= \min_{\substack{\mathbf{e}_i \in \mathcal{E}_{CE} \\ t \in \{0,\dots,H-1\}}} e_{i,t} \end{aligned} \tag{4.20}$$

and define $s_u = e_{\max} - e_u$ and $s_l = e_l - e_{\min}$. We then use s_u and s_l to update the range for e_t in φ_w to a maximal interval $[e'_{\min}, e'_{\max}] \subseteq [e_{\min}, e_{\max}]$ and such that at least one $\mathbf{e}_i \in \mathcal{E}_{CE}$ is excluded. Specifically, if $s_u \leq s_l$, we set $[e'_{\min}, e'_{\max}] = [e_{\min}, e_u - \epsilon]$; otherwise we set $[e'_{\min}, e'_{\max}] = [e_l + \epsilon, e_{\max}]$. The smaller the value of ϵ, the larger the resulting interval. Finally, we use the updated formula φ'_w to run SolveCEGIS again until a realizable control sequence $\mathbf{u}$ is found. For improved efficiency, the linear search proposed above to find the updated bounds $e'_{\min}$ and $e'_{\max}$ can be replaced by a binary search. Moreover, in Algorithm 4, assuming a predicate repair procedure, FindMin provides the solution with minimum slack norm between the ones repairing ψ and φ_w.

The overall CEGIS framework $\mathcal{C}_{\varphi_C}$ is shown in Figure 4.4

Example 10 (Adversarial Race)**.** *We consider the specification in Example 7. For the same horizon as in the previous examples, after solving the satisfiability problem, for the fixed* $\mathbf{u}_0$*, the CEGIS loop returns* $a_t^{\text{adv}} = 2$ *for all* $t \in \{0, \dots, H-1\}$ *as the single element in* $\mathcal{E}_{CE}$ *for which no controller sequence can be found. We then choose to tighten the environment assumptions to make the controller realizable, by shrinking the bounds on* a_t^{adv} *by using Algorithm 4 with* $\epsilon = 0.01$*. After a few iterations, we finally obtain* $e'_{\min} = 0$ *and* $e'_{\max} = 1.24$*, and therefore* $\phi'_w = \mathbf{G}_{[0,\infty)}\big(0 \leq a_t^{\text{adv}} \leq 1.24\big)$.

To account for the error introduced by ϵ, given $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$, we say that $(\varphi', \mathcal{D}, \mathcal{T})$ are ϵ-minimal if the magnitudes of the predicate repairs (predicate slacks) or time-interval repairs differ by at most ϵ from a minimal repair in the sense of Problem 2. Assuming that SolveCEGIS terminates before reaching the maximum number of iterations[5], the following theorems state the properties of Algorithm 4.

Theorem 6 (Soundness)**.** *Given a controller synthesis problem* $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$*, such that* (4.2) *is infeasible at time* t*, let* $\varphi' \in \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ *be the repaired formula returned from*

[5]Under failing assumptions, Algorithm 4 terminates with UNKNOWN.

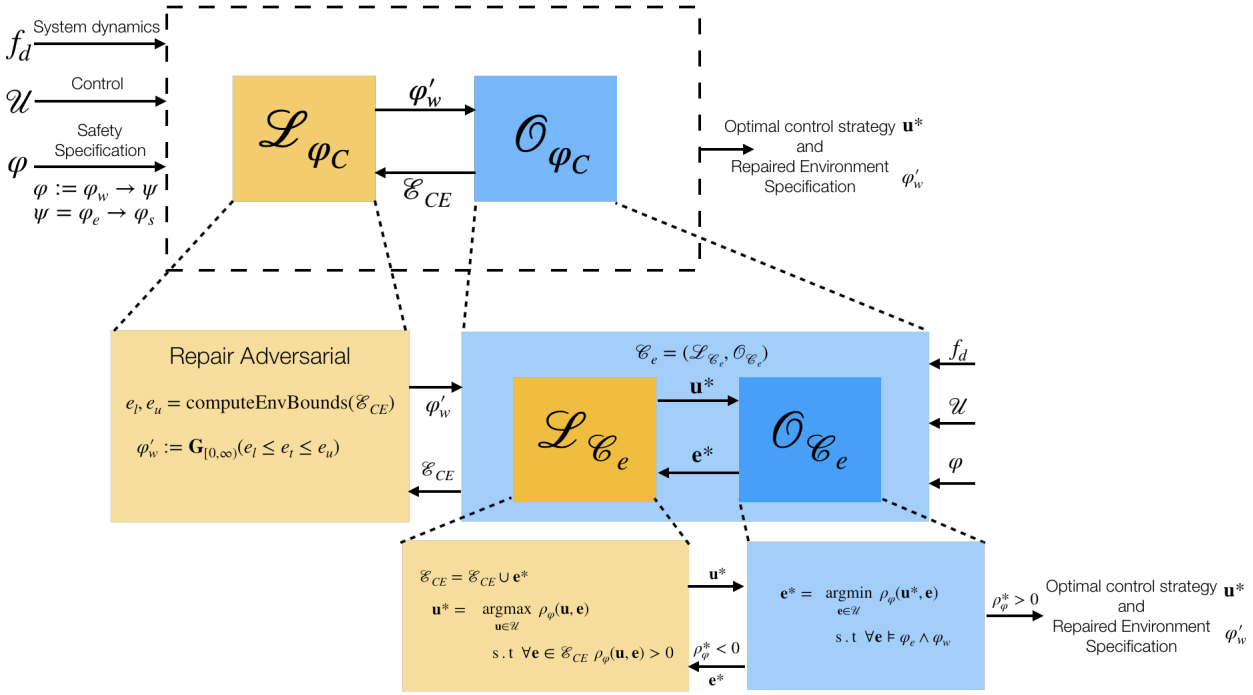


Figure 4.4. Hierarchical CEGIS for Problem 2 $\mathcal{C}_{\varphi_C} = (\mathcal{L}_{\mathcal{C}_{\varphi_C}}, \mathcal{O}_{\mathcal{C}_{\varphi_C}})$. The oracle $\mathcal{O}_{\mathcal{C}_{\varphi_C}}(\mathcal{L}_{\mathcal{C}_{\varphi_C}})$ is shown in blue (yellow). The oracle $\mathcal{O}_{\mathcal{C}_{\varphi_C}}$ implements a CEGIS framework $\mathcal{C}_E = (\mathcal{L}_{\mathcal{C}_e}, \mathcal{O}_{\mathcal{C}_e})$. The learner $\mathcal{L}_{\mathcal{C}_{\varphi_C}}$ repairs the adversarial environment assumption φ_w. The loop terminates when the φ_w becomes empty (trivially true) or $\mathcal{O}_{\mathcal{C}_e}$ does not find a counter-example to the candidate control strategy $\mathbf{u}^*$ proposed by $\mathcal{L}_{\mathcal{C}_e}$.

Algorithm 4 for a given set of predicates $\mathcal{D}$ or time interval $\mathcal{T}$. Then, $\mathcal{P}' = (f_d, x_0, \varphi', \rho_{\varphi'})$ is feasible at time t and $(\varphi', \mathcal{D}, \mathcal{T})$ is ϵ-minimal.

Proof: [Proof (Theorem 6)] We recall that $\varphi \equiv \varphi_w \rightarrow \psi$. Moreover, Algorithm 4 provides the solution with minimum slack norm between the ones repairing ψ and φ_w in the case of predicate repair. Then, when $\psi = \varphi_e \rightarrow \varphi_s$ is modified using Algorithm 1, soundness is guaranteed by Theorem 4 and the termination of the CEGIS loop. On the other hand, assume Algorithm 4 modifies the atomic predicates in φ_w. Then, the RepairArdversarial routine and (4.20), together with the termination of the CEGIS loop, assure that φ_w is also repaired in such a way that the controller is realizable, and ϵ-optimal (i.e., the length of the bounding box around e_t differs from the maximal interval length by at most ϵ), which concludes our proof. □

Theorem 7 (Completeness)**.** *Assume the controller synthesis problem $\mathcal{P} = (f_d, x_0, \varphi, \rho_\varphi)$ results in (4.2) being infeasible at time t. If there exist a set of predicates $\mathcal{D}$ and time-intervals $\mathcal{T}$ such that there exists $\Phi \subseteq \texttt{REPAIR}_{\mathcal{D},\mathcal{T}}(\varphi)$ for which $\forall\ \phi \in \Phi$, $\mathcal{P}' = (f_d, x_0\phi, \rho_\phi)$ is feasible at time t and $(\phi, \mathcal{D}, \mathcal{T})$ is ϵ-minimal, then Algorithm 4 returns a repaired formula φ' in Φ.*

Proof: [Proof (Theorem 7)] As discussed in the proof of Theorem 6, if Algorithm 4 modifies $\psi = \varphi_e \rightarrow \varphi_s$ using Algorithm 1, completeness is guaranteed by Theorem 5 and the termination of the CEGIS loop. On the other hand, let us assume there exists a minimum norm repair for the atomic predicates of φ_w, which returns a maximal interval $[e'_{\min}, e'_{\max}] \subseteq [e_{\min}, e_{\max}]$. Then, given the termination of the CEGIS loop, by repeatedly applying (4.20) and RepairAdversarial, we produce a predicate repair such that the corresponding interval $[e''_{\min}, e''_{\max}]$ makes the control synthesis realizable and is maximal within an error bounded by ϵ (i.e., its length differs by at most ϵ from the one of the maximal interval $[e'_{\min}, e'_{\max}]$). Hence, $\varphi' \in \Phi$ holds. □

4.8 Evaluation

We developed the toolbox DiaRY (Diagnosis and Repair for sYnthesis)[6] implementing our algorithms. DiaRY uses Yalmip [102] to formulate the optimization problems and Gurobi [72] to solve them. It interfaces to different synthesis tools, e.g., BluSTL[7] and CrSPrSTL[8]. Here, we summarize some of the results of DiaRY for diagnosis and repair.

4.8.1 Autonomous Driving

We consider the problem of synthesizing a controller for an autonomous vehicle in a city driving scenario. We analyze the following two tasks: (i) changing lanes on a busy road; (ii) performing an unprotected left turn at a signalized intersection. We use a simple point-mass model for the vehicles on the road. For each vehicle, we define the state as $\mathbf{x} = [x \;\; y \;\; \theta \;\; v]^\top$, where x and y denote the coordinates, and θ and v represent the direction and speed, respectively. Let $u = [u_1 \;\; u_2]^\top$ be the control input for each vehicle, where u_1 is the steering input and u_2 is the acceleration. Then, the vehicle's state evolves according to the following dynamics:

$$\begin{aligned} \dot{x} &= v \cos\theta \\ \dot{y} &= v \sin\theta \\ \dot{\theta} &= v \cdot u_1/m \\ \dot{v} &= u_2, \end{aligned} \tag{4.21}$$

where m is the vehicle mass. To determine the control strategy, we linearize the overall system dynamics around the initial state at each run of the MPC, which is completed in less than 2 s on a 2.3-GHz Intel Core i7 processor with 16-GB memory. We further impose the following constraints on the *ego* vehicle (i.e., the vehicle under control): (i) a minimum distance must be established between the *ego* vehicle and other cars on the road to avoid

[6] https://github.com/shromonag/DiaRY
[7] https://github.com/BluSTL/BluSTL
[8] https://github.com/dsadigh/CrSPrSTL

collisions; (ii) the *ego* vehicle must obey the traffic lights; (iii) the *ego* vehicle must stay within its road boundaries.

4.8.1.1 Lane Change

We consider a lane change scenario on a busy road as shown in Figure 4.5a. The *ego* vehicle is in red. *Car 1* is at the back of the left lane, *Car 2* is in the front of the left lane, while *Car 3* is on the right lane. The states of the vehicles are initialized as follows: $x_0^{\text{Car 1}} = [-0.2 \;\; -1.5 \;\; \frac{\pi}{2} \;\; 0.5]^\top$, $x_0^{\text{Car 2}} = [-0.2 \;\; 1.5 \;\; \frac{\pi}{2} \;\; 0.5]^\top$, $x_0^{\text{Car 3}} = [0.2 \;\; 1.5 \;\; \frac{\pi}{2} \;\; 0]^\top$, and $x_0^{\text{ego}} = [0.2 \;\; -0.7 \;\; \frac{\pi}{2} \;\; 0]^\top$. The control inputs for *ego* and *Car 3* are initialized at $[0 \;\; 0]^\top$; the ones for *Car 1* and *Car 2* are set to $u_0^{\text{Car 1}} = [0 \;\; 1]^\top$ and $u_0^{\text{Car 2}} = [0 \;\; -0.25]^\top$. The objective of *ego* is to safely change lane, while satisfying the following requirements:

$$\begin{aligned} \varphi_{\text{str}} &= \mathbf{G}_{[0,\infty)}(|u_1| \leq 2) \text{ Steering Bounds} \\ \varphi_{\text{acc}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 1) \text{ Acceleration Bounds} \\ \varphi_{\text{vel}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1) \text{ Velocity Bounds} \end{aligned} \tag{4.22}$$

The solid blue line in Figure 4.5 is the trajectory of *ego* as obtained from our MPC scheme, while the dotted green line is the future trajectory pre-computed for a given horizon at a given time. MPC becomes infeasible at time $t = 1.2$ s when the no-collision requirement is violated, and a possible collision is detected between the *ego* vehicle and *Car 1* before the lane change is completed (Figure 4.5a). Our solver takes 2 s, out of which 1.4 s are needed to generate all the IISs, consisting of 39 constraints. To make the system feasible, the proposed repair increases both the acceleration bounds and the velocity bounds on the *ego* vehicle as follows:

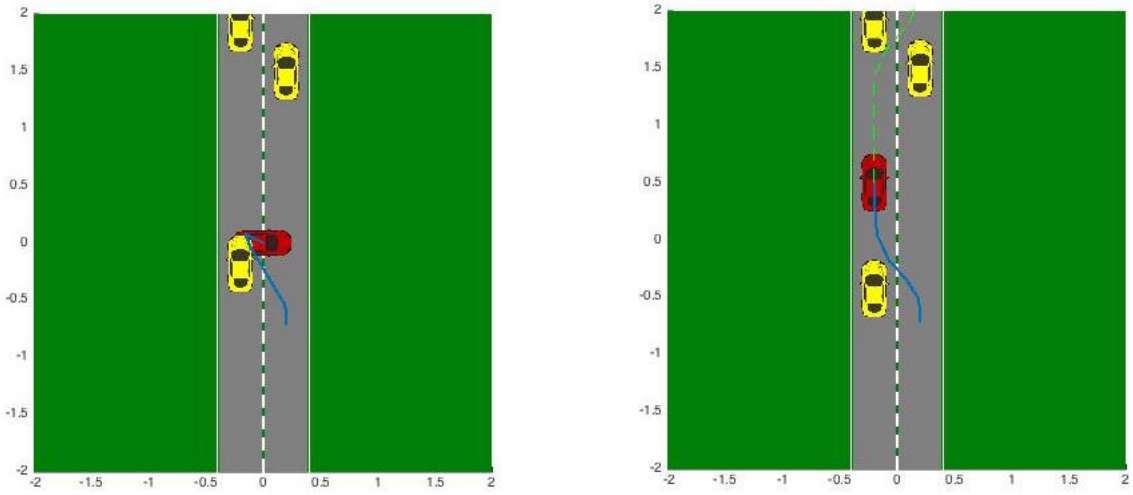
$$\begin{aligned} \varphi_{\text{acc}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 3.5) \\ \varphi_{\text{vel}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}(|v| \leq 1.54). \end{aligned} \tag{4.23}$$

When replacing the initial requirements φ_{acc} and φ_{vel} with the modified ones, the revised MPC scheme allows the vehicle to travel faster and safely complete a lane change maneuver, without risks of collision, as shown in Figure 4.5b.

4.8.1.2 Unprotected Left Turn

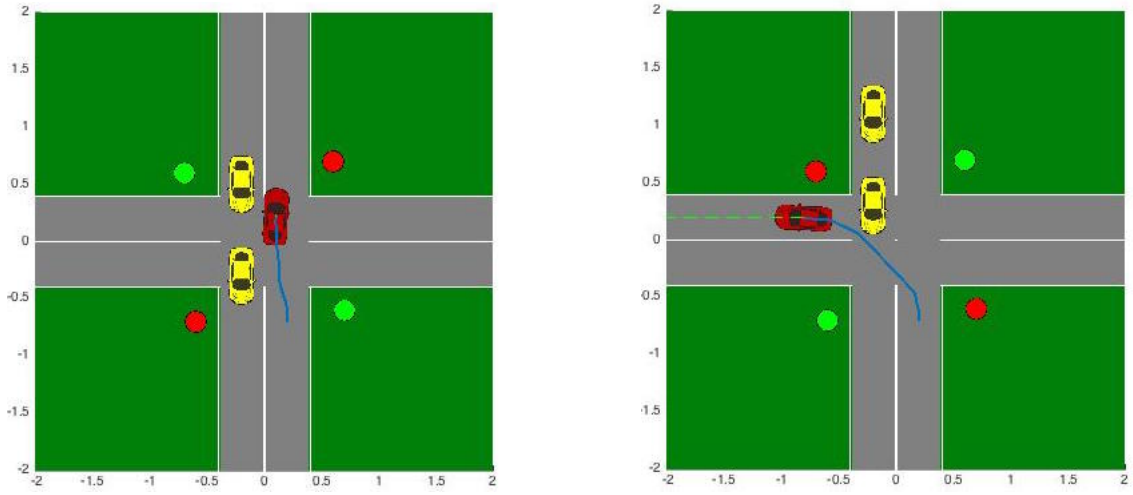
In the second scenario, we would like the ego vehicle to perform an unprotected left turn at a signalized intersection, where the ego vehicle has a green light and is supposed to yield to oncoming traffic, represented by the yellow cars crossing the intersection in Figure 4.6. The environment vehicles are initialized at the states $x_0^{\text{Car 1}} = [-0.2 \;\; 0.7 \;\; -\frac{\pi}{2} \;\; 0.5]^\top$ and $x_0^{\text{Car2}} = [-0.2 \;\; 1.5 \;\; -\frac{\pi}{2} \;\; 0.5]^\top$, while the ego vehicle is initialized at $x_0^{\text{ego}} = [0.2 \;\; -0.7 \;\; \frac{\pi}{2} \;\; 0]^\top$. The control input for each vehicle is initialized at $[0 \;\; 0]^\top$. Moreover, we use the same bounds as in (4.22).

The MPC scheme becomes infeasible at $t = 2.1$ s. The solver takes 5 s, out of which 2.2 s are used to generate the IISs, including 56 constraints. As shown in Fig. 4.6a, the ego



(a) (b)

Figure 4.5. Changing lane is infeasible at $t = 1.2$ s in (a) and is repaired in (b).



(a) (b)

Figure 4.6. Left turn becomes infeasible at time $t = 2.1$ s in (a) and is repaired in (b).

vehicle yields in the middle of intersection for the oncoming traffic to pass. However, the traffic signal turns red in the meanwhile, and there is no feasible control input for the ego vehicle without breaking the traffic light rules. Since we do not allow modifications to the traffic light rules, the original specification is repaired again by increasing the bounds on acceleration and velocity, thus obtaining:

$$\begin{aligned}\varphi_{\text{acc}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}\big(|u_2| \leq 11.903\big)\\ \varphi_{\text{vel}}^{\text{new}} &= \mathbf{G}_{[0,\infty)}\big(|v| \leq 2.42\big).\end{aligned} \tag{4.24}$$

As shown by the trajectory in Figure 4.6b, under the assumptions and initial conditions of our scenario, higher allowed velocity and acceleration make the ego vehicle turn before the oncoming cars get close or cross the intersection.

4.8.2 Quadrotor Control

We assume a quadrotor dynamical model including a 12-dimensional state, based on the model reported in [77]. The state variables express the 6 degrees of freedom for the quadrotor, i.e., position, x, y, z, and rotation angles, ϕ, θ, ψ, together with their first derivatives, $\dot{x}, \dot{y}, \dot{z}, p, q, r$. Variables ϕ, θ, ψ denote, respectively, the roll, pitch, and yaw angles. The system has a 4-dimensional input, $[u_1 \;\; u_2 \;\; u_3 \;\; u_4]^\top$, where u_1, u_2, and u_3 are the roll, pitch, and yaw control inputs, and u_4 is the thrust applied to the quadrotor. To control the system, we locally linearize the following nonlinear dynamics at every time step:

$$\begin{aligned}f_1 &= \begin{bmatrix}\dot{x} & \dot{y} & \dot{z}\end{bmatrix}^\top\\ f_2 &= \begin{bmatrix}0 & 0 & g\end{bmatrix}^\top - (1/m)R_1(\dot{x}, \dot{y}, \dot{z}) \begin{bmatrix}0 & 0 & 0 & u_4\end{bmatrix}^\top\\ f_3 &= R_2(\dot{x}, \dot{y}, \dot{z}) \begin{bmatrix}p & q & r\end{bmatrix}^\top\\ f_4 &= I^{-1} \begin{bmatrix}u_1 & u_2 & u_3\end{bmatrix}^\top - R_3(p, q, r)I \begin{bmatrix}p & q & r\end{bmatrix}^\top\end{aligned} \tag{4.25}$$

where R_1, R_2, and R_3 are rotation matrices relating the body frame and the inertial frame, and I is the inertial matrix.

Our goal is to synthesize a strategy for the quadrotor to travel from a starting position $[x_0, y_0, z_0] = [0, 0, -0.4]$ with zero roll, pitch, and yaw angles, i.e., $[\phi_0, \theta_0, \psi_0] = [0, 0, 0]$, to a destination $[x_d, y_d, z_d] = [1, 1, -0.1]$, still with zero roll, pitch, and yaw. All the other elements in the state vector and the control input are initialized at zero. We define the following constraints on the quadrotor:

$$\begin{aligned}\varphi_{\text{h}} &= \mathbf{G}_{[0,\infty)}(-1.1 \leq z \leq 0)\text{Height of Flight Bounds}\\ \varphi_{\text{roll}} &= \mathbf{G}_{[0,\infty)}(|u_1| \leq 0.3)\text{Roll Bounds}\\ \varphi_{\text{pitch}} &= \mathbf{G}_{[0,\infty)}(|u_2| \leq 0.3)\text{Pitch Bounds}\\ \varphi_{\text{thr}} &= \mathbf{G}_{[0,\infty)}(0 \leq u_4 \leq 6.5)\text{Thrust Bounds.}\end{aligned} \tag{4.26}$$

Our MPC scheme becomes infeasible at time $t = 0.675$ s because φ_{h} and φ_{roll} are both violated. Given the bounds on the control inputs, the trajectory is not guaranteed to lie in the desired region. This is visualized in Fig. 4.7 (a) and (c), respectively showing a two-dimensional and three-dimensional projection of the quadrotor trajectory. The solid blue line shows the path computed by the MPC framework and taken by the quadrotor, aiming at traveling from the origin, marked by a black square, to the target, marked by a red square. Because of the bounds on the control inputs, the quadrotor touches the boundary of the allowed region (along the z axis) at $t = 0.675$ s.

The solver takes less than 0.1 s to generate the IIS, including 32 constraints, out of which 11 constraints are associated with the predicates in φ_{h} and φ_{roll}. Our algorithm adds a slack of 1.58 to the upper bound on z in φ_{h}. We then modify φ_{h} to:

$$\varphi_{\text{h}}^{\text{new}} = \mathbf{G}_{[0,\infty)}(-1.1 \leq z \leq 1.58), \tag{4.27}$$

thus allowing the quadrotor to violate the upper bound on the vertical position during the maneuver. The new specification makes the problem realizable, and the resulting trajectory is shown in blue in Figure 4.7 (b) and (d). We can view the above slack as the estimated margin from the boundary needed for the quadrotor to complete the maneuver based on the linearized model of the dynamics. As apparent from Figure 4.7b, such a margin is much larger than the space actually used by the quadrotor to complete its maneuver. A better estimate can be achieved by using a finer time interval for linearizing the dynamics and executing the controller. While the solution above provides the minimum slack norm for the given linearization, it is still possible to notify DiaRY that φ_{h} must be regarded as a hard constraint, e.g., used to mark a rigid obstacle. In this case, DiaRY tries to relax the bounds on the control inputs to achieve feasibility.

4.8.3 Aircraft Electric Power System

Figure 4.8 shows a simplified architecture for the primary power distribution system in a passenger aircraft [118].

Two power sources, the left and right generators G_0 and G_1, deliver power to a set of high-voltage AC and DC buses (B_0, B_1, DB_0, and DB_1) and their loads. AC power from the generators is converted to DC power by rectifier units (R_1 and R_2). A bus power control unit (controller) monitors the availability of power sources and configures a set of electromechanical switches, denoted as contactors ($C_0, \ldots, C_4$), such that essential buses remain powered even in the presence of failures, while satisfying a set of safety, reliability, and real-time performance requirements [118]. Specifically, we assume that only the right DC bus DB_1 is essential, and use our algorithms to check the feasibility of a controller that accommodates a failure in the right generator G_1, by rerouting power from the left generator to the right DC bus in a time interval which is less than or equal to $t_{\max} = 100$ ms. In addition, the controller must satisfy the following set of requirements, all captured by an STL contract.

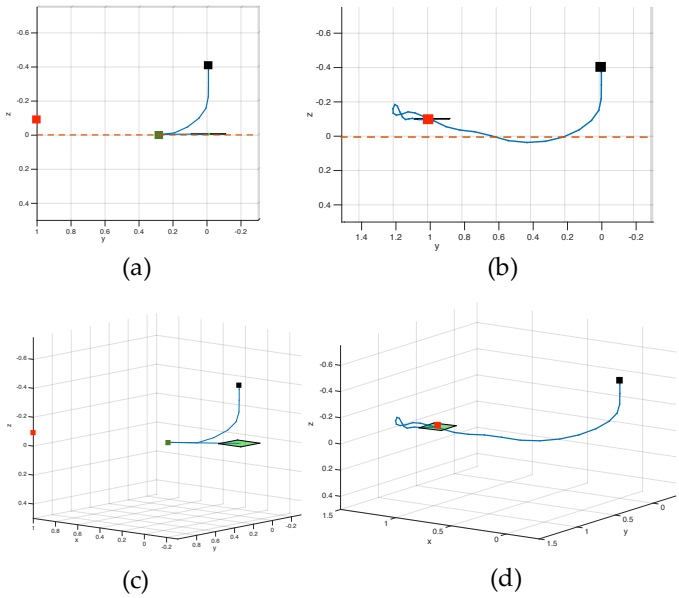


Figure 4.7. Diagnosis and repair of infeasibilities in quadrotor control. The top and bottom figures show, respectively, a 2D and 3D projection of the trajectory (blue line) of the quadrotor, represented as a green rectangle. The black square marks the initial position while the red square marks the goal. As shown in Fig. (a) and (c), the original specification becomes infeasible at time $t = 0.675$, which is marked by a green square along the trajectory, when the quadrotor hits the boundary represented by the dotted red line. After updating the specification, controller synthesis becomes feasible, as shown in Fig. (b) and (d), where the quadrotor reaches the final position, at the cost of passing through the red dotted line.

Assumptions. *When a contactor receives an open (close) signal, it shall become open (closed) in 80 ms or less.* Let $\Delta t = 20$ ms be the time discretization step, $\tilde{c}_i$, $i \in \{0, \ldots, 4\}$, be a set of Boolean variables describing the controller signal (where 1 stands for "closed" and 0 for "open"), c_i, $i \in \{0, \ldots, 4\}$, be a set of Boolean variables denoting the state (open/closed) of the contactors. We can capture the system assumptions via a conjunction of formulae of the form: $\mathbf{G}_{[0,\infty)}(\tilde{c}_i \rightarrow \mathbf{F}_{[0,4]} c_i)$, providing a model for the discrete-time binary-valued contactor states. The actual delay of each contactor can then be modeled using an integer (environment) variable k_i for which we require: $\mathbf{G}_{[0,\infty)}(0 \leq k_i \leq 4)$.

Guarantees. *If a generator becomes unavailable (fails), the controller shall disconnect it from the power network in 20 ms or less.* Let g_0 and g_1 be Boolean environment variables representing the state of the generators, where 1 stands for "available" and 0 for "failure." We encode the above guarantees as $\mathbf{G}_{[0,\infty)}(g_i \rightarrow \mathbf{F}_{[0,1]} \tilde{c}_i)$. *A DC bus shall never be disconnected from an AC generator for 100 ms or more,* i.e., $\mathbf{G}_{[0,\infty)}(\neg b_i \rightarrow \mathbf{F}_{[0,5]} b_i)$, where b_i, $i \in \{0, \ldots, 3\}$, is a set of Boolean variables denoting the status of a bus, where 1 stands for "powered" and 0 for "unpowered." Additional guarantees, which can also be expressed as STL formulae, include: (i) If both AC generators are available, the left AC generator shall power the left AC bus, and the right AC generator shall power the right AC bus. C_3 and C_4 shall be closed. (ii) If one generator becomes unavailable, all buses shall be connected to

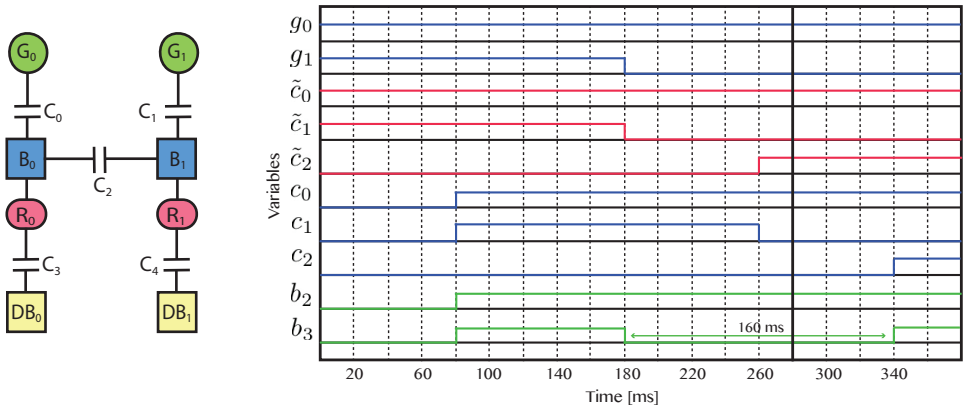


Figure 4.8. Simplified model of an aircraft electric power system (left) and counterexample trajectory (right). The blue, green and red lines represent environment, state, and controller variables, respectively, for a 380-ms run.

the other generator. (iii) Two generators must never be directly connected.

We apply the diagnosis and repair procedure in Section 4.7.2 to investigate whether there exists a control strategy that can satisfy the specification above over all possible values of contactor delays. As shown in Figure 4.8, the controller is unrealizable; a trace of contactor delays equal to 4 at all times provides a counterexample, which leaves DB_1 unpowered for 160 ms, exceeding the maximum allowed delay of 100 ms. In fact, the controller cannot close C_2 until C_1 is tested as being open, to ensure that G_1 is safely isolated from G_2. To guarantee realizability, Algorithm 4 suggests to either modify our assumptions to $\mathbf{G}_{[0,\infty)}(0 \leq k_i \leq 2)$ for $i \in \{0, \ldots, 4\}$ or relax the guarantee on DB_1 to $G_{[0,\infty)}(\neg b_3 \rightarrow \mathbf{F}_{[0,8]} b_3)$. The overall execution time was 326 s, which includes formulating and executing three CEGIS loops, requiring a total of 6 optimization problems.

4.9 Conclusion

In this chapter, we described how the OGIS framework can be extended to diagnose and repair high-level temporal specifications for synthesis. We presented an OGIS framework $\mathcal{I}_{\varphi_M}$ which relies on Irreducibly Inconsistent System (IIS) to diagnose (and repair) monolithic specification in the absence of environmental agents. We then present a hierarchical CEGIS framework $\mathcal{C}_{\varphi_C}$ which relies on the CEGIS frameworks introduced in Chapter 3 to diagnose (and repair) safety specifications in adversarial environments. Finally, we conclude the chapter with examples from autonomous driving and aircraft electric power system.

Chapter 5

Counter-Example Guided Data Augmentation

5.1 Introduction

In Chapter 3 we address the controller design and analysis pipeline for robotic systems. In this chapter, we study the problem of designing and analyzing perception modules. Perception modules are becoming increasingly important in robotic systems to perceive or sense the world around us. Hence, it is important to be able to formally analyze the correctness of such models. In this chapter, we present a novel CEGIS framework for augmenting data sets for machine learning systems based on *counterexamples* (misclassified images).

Models produced by machine learning algorithms, especially *deep neural networks*, are being deployed in domains where trustworthiness is a big concern, creating the need for higher accuracy and assurance [130, 135]. However, learning high-accuracy models using deep learning is limited by the need for large amounts of data, and, even further, by the need of labor-intensive labeling.

5.1.1 Data Augmentation

Data augmentation overcomes the lack of data by inflating training sets with label-preserving transformations, i.e., transformations which do not alter the label. Traditional data augmentation schemes [51, 138, 29, 28, 91] involve geometric transformations which alter the geometry of the image (e.g., rotation, scaling, cropping or flipping); and photometric transformations which vary color channels. The efficacy of these techniques have been demonstrated recently (see, e.g., [162, 158]). Traditional augmentation schemes, like the aforementioned methods, add data to the training set hoping to improve the model accuracy without taking into account what kind of features the model has already learned. More recently, a sophisticated data augmentation technique has been proposed [99, 105] which uses Generative Adversarial Networks [70], a particular kind of neural network able to generate synthetic data, to inflate training sets. There are also augmentation techniques, such as hard negative

mining [137], that inflate the training set with targeted negative examples with the aim of reducing false positives.

In this chapter, we propose a new augmentation scheme, *counterexample-guided data augmentation*. The main idea is to augment the training set only with new misclassified examples rather than modified images coming from the original training set. The proposed augmentation scheme consists of the following steps: 1) Generate synthetic images that are misclassified by the model, i.e., the counterexamples; 2) Add the counterexamples to the training set; 3) Train the model on the augmented dataset. These steps can be repeated until the desired accuracy is reached. Note that our augmentation scheme depends on the ability to generate misclassified images. For this reason, we developed an *image generator* [48] that cooperates with a *sampler* to produce images that are given as input to the model. The images are generated in a manner such that the ground truth labels can be automatically added. The incorrectly classified images constitute the augmentation set that is added to the training set. In addition to the pictures, the image generator provides information on the misclassified images, such as the disposition of the elements, brightness, contrast, etc. This information can be used to find features that frequently recur in counterexamples. We collect information about the counterexamples in a data structure we term as the "*error table*". Error tables are extremely useful to provide *explanations* about counterexamples and find recurring patterns that can lead an image to be misclassified. The error table analysis can also be used to generate images which are likely to be counterexamples, and thus, efficiently build augmentation sets.

In summary, the main contributions of this chapter are:

- A *counterexample-guided data augmentation* approach where only misclassified examples are iteratively added to training sets;
- A synthetic *image generator* that renders realistic counterexamples;
- *Error tables* that store information about counterexamples and whose analysis provides explanations and facilitates the generation of counterexample images.

We conducted experiments on Convolutional Neural Networks (CNNs) for object detection by analyzing different counterexample data augmentation sampling schemes and compared the proposed methods with classic data augmentation. Our experiments show the benefits of using a counterexample-driven approach against a traditional one. The improvement comes from the fact that a counterexample augmentation set contains information that the model had not been able to learn from the training set, a fact that was not considered by classic augmentation schemes. In our experiments, we use synthetic data sets generated by our image generator. This ensures that all treated data comes from the same distribution.

The results in this chapter are adapted from [48, 47].

5.2 Preliminaries

This section provides the notation used throughout this chapter.

Let $\mathbf{a}$ be a vector, a_i be its i-th element with index starting at $i = 1$, $a_{i:j}$ be the range of elements of $\mathbf{a}$ from i to j; and $\mathbb{A}$ be a set. $\mathbb{X}$ is a set of training examples, $\mathbf{x}^{(i)}$ is the i-th example from a dataset and $\mathbf{y}^{(i)}$ is the associated label. $f : \mathbb{A} \to \mathbb{B}$ is a model (or function) f with domain $\mathbb{A}$ and range $\mathbb{B}$. $\hat{\mathbf{y}} = f(\mathbf{x})$ is the prediction of the model f for input $\mathbf{x}$. In the object detection context, $\hat{\mathbf{y}}$ encodes bounding boxes, scores, and categories predicted by f for the image $\mathbf{x}$. $f_{\mathbb{X}}$ is the model f trained on $\mathbb{X}$.

Let B_1 and B_2 be bounding boxes encoded by $\hat{\mathbf{y}}$. The *Intersection over Union* (IoU) is defined as $IoU(A_{B_1}, A_{B_2}) = A_{B_1} \cap A_{B_2} / A_{B_1} \cup A_{B_2}$, where A_{B_i} is the area of B_i, with $i \in \{1, 2\}$. We consider $B_{\hat{\mathbf{y}}}$ to be a *detection* for $B_{\mathbf{y}}$ if $IoU(B_{\hat{\mathbf{y}}}, B_{\mathbf{y}}) > 0.5$. *True positives* t_p is the number of correct detections; *false positives* f_p is the number of predicted boxes that do not match any ground truth box; *false negatives* f_n is the number of ground truth boxes that are not detected.

Precision and *recall* are defined as $p(\hat{\mathbf{y}}, \mathbf{y}) = t_p/(t_p + f_p)$ and $r(\hat{\mathbf{y}}, \mathbf{y}) = t_p/(t_p + f_n)$. In this work, we consider an input $\mathbf{x}$ to be *misclassified* if $p(\hat{\mathbf{y}}, \mathbf{y})$ or $r(\hat{\mathbf{y}}, \mathbf{y})$ is less than 0.75. Let $\mathbb{T} = \{(\mathbf{x}^{(1)}, \mathbf{y}^{(1)}), \ldots, (\mathbf{x}^{(m)}, \mathbf{y}^{(m)})\}$ be a test set with m examples. The *average precision* and *recall* of f are defined as $\bar{p}_f(\mathbb{T}) = \frac{1}{m} \sum_{i=1}^{m} p(f(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$ $\bar{r}_f(\mathbb{T}) = \frac{1}{m} \sum_{i=1}^{m} r(f(\mathbf{x}^{(i)}), \mathbf{y}^{(i)})$. We use average precision and recall to measure the accuracy of a model, succinctly represented as $acc_f(\mathbb{T}) = (\bar{p}_f(\mathbb{T}), \bar{r}_f(\mathbb{T}))$.

5.3 Solution Approach

Our overall goal is to design a technique for improving model accuracy using data augmentation. We formalize the data augmentation scheme as an instance of the CEGIS framework $\mathcal{C}_{da} = (\mathcal{L}_{\mathcal{C}_{da}}, \mathcal{O}_{\mathcal{C}_{da}})$ as shown in Figure 5.1. The concept class for the learner is the set of all learning models corresponding to a fixed structure. The counter-example queries handled by oracle provides not only a set of counter-examples (defined as an augmentation set) but also an error table with detailed information about the counter-examples. The training of the model f using the learning algorithm $\mathcal{A}$ is done by the learner $\mathcal{L}_{\mathcal{C}_{da}}$. The learner sends the model to the oracle $\mathcal{O}_{\mathcal{C}_{da}}$. Figure 5.1 summarizes the proposed counterexample-guided augmentation scheme (CEGDA). The oracle takes as input the model f, the image generator γ, and a modification space, $\mathbb{M}$, the space of possible configurations of our image generator. The space $\mathbb{M}$ is constructed based on domain knowledge to be a space of "semantic modifications;" i.e., each modification must have a meaning in the application domain in which the machine learning model is being used. This allows us to perform more meaningful data augmentation than simply through adversarial data generation performed by perturbing an input vector (e.g., adversarially selecting and modifying a small number of pixel values in an image).

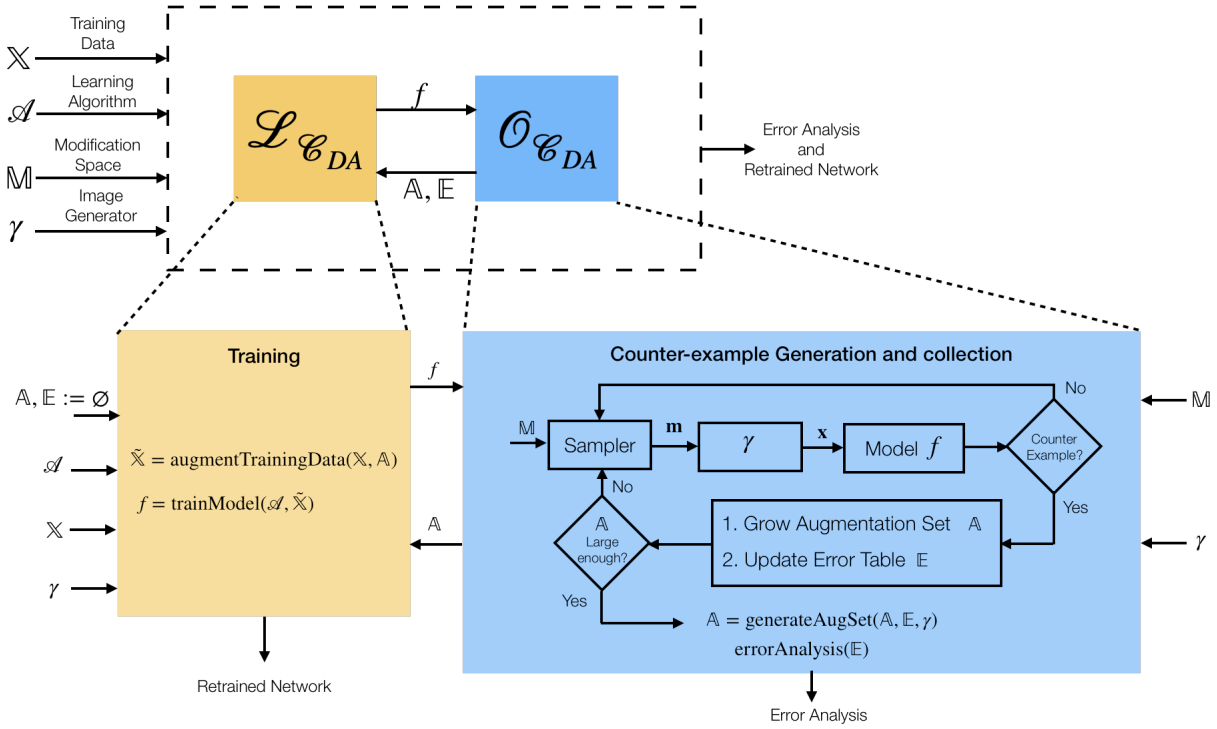


Figure 5.1. Counter-example guided data augmentation (CEGDA): $\mathcal{C}_{da} = (\mathcal{L}_{\mathcal{C}_{da}}, \mathcal{O}_{\mathcal{C}_{da}})$. The oracle $\mathcal{O}_{\mathcal{C}_{da}}(\mathcal{L}_{\mathcal{C}_{da}})$ is shown in blue (yellow). The oracle $\mathcal{O}_{\mathcal{C}_{da}}$ generates an *augmentation set* $\mathbb{A}$ and an *error table* $\mathbb{E}$. It then extracts features from the error table $\mathbb{E}$ to explain the cause of failure of the trained model f, which is used to generate $\mathbb{A}$. Moreover, it analyzes the $\mathbb{E}$ to provide feedback to the user. The learner $\mathcal{L}_{\mathcal{C}_{da}}$ then augments the training data $\mathbb{X}$ to generate $\tilde{\mathbb{X}}$ which is used to train the model f.

In each loop, the sampler selects a modification, $\mathbf{m}$, from $\mathbb{M}$. The sample is determined by a sampling method that can be biased by a precomputed *error table* $\mathbb{E}$, a data structure that stores information about images that are misclassified by the model. The sampled modification is rendered into a picture $\mathbf{x}$ by the image generator. The image $\mathbf{x}$ is given as input to the model f that returns the prediction $\hat{\mathbf{y}}$. We then check whether $\mathbf{x}$ is a counterexample, i.e., the prediction $\hat{\mathbf{y}}$ is wrong. If so, we add $\mathbf{x}$ to our augmentation set $\mathbb{A}$ and we store $\mathbf{x}$'s information (such as $\mathbf{m}$, $\hat{\mathbf{y}}$) in the error table that will be used by the sampler at the next iteration. The loop is repeated until the augmentation set $\mathbb{A}$ is large enough (or $\mathbb{M}$ has been sufficiently covered).

This scheme returns an *augmentation set*, that will be used to retrain the treated model, along with an *error table*, whose analysis identifies common features among counterexamples and aids the sampler to select candidate counterexamples.

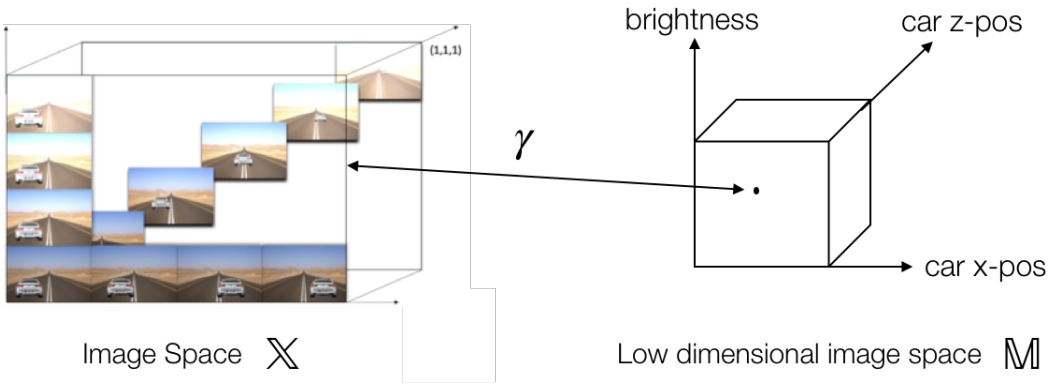


Figure 5.2. The low dimension (3D) modification space $\mathbb{M}$ on the right can be projected to the high dimensional feature (image) space $\mathbb{X}$ on the left through the image generator function γ.

5.4 Design of Image Generator γ

In this section, we discuss the design of the image generator (image renderer). At the core of our counterexample augmentation scheme is an image generator (similar to the one defined in [44, 48]) that renders realistic synthetic images of road scenarios. Since counterexamples are generated by the synthetic data generator, we have full knowledge of the ground truth labels for the generated data. In our case, for instance, when the image generator places a car in a specific position, we know exactly its location and size, hence the ground truth bounding box is accordingly determined. In this section, we describe the details of our image generator.

5.4.1 Modification Space

The image generator implements a *generation function* $\gamma : \mathbb{M} \to \mathbb{X}$ that maps every modification $\mathbf{m} \in \mathbb{M}$ to a feature $\gamma(\mathbf{m}) \in \mathbb{X}$. Intuitively, a modification describes the configuration of an image.

Example 11 (Visualization of 3D Modification Space)**.** *Consider the set of all* 1242×375 *RGB picture (KITTI image resolution [61]). Since we are interested in the automotive context, we consider the subset* $\mathbb{X}$ *of pictures of cars defined by a low-dimensional* $\mathbb{M}$*. Specifically, let us consider a 3D modification space which characterizes the car* x *(lateral) and* z *(away) displacement on the road and the image brightness. The generation function* $\gamma : [0,1]^3 \to \mathbb{X}$*. For instance,* $\gamma(0,0,0)$ *places the car on the left close to the observer with high contrast,* $\gamma(1,0,0)$ *shifts the car to the right, or* $\gamma(1,1,1)$ *sees the car on the right, far from the observer, with low contrast. The spectrum of images generated by* γ *is shown in Figure 5.2. When moving along the* x*-axis in the* $\mathbb{M}$*, the car shifts horizontally; a change on the* y*-axis affects the position of the car along the road; the* z*-axis affects the brightness of the overall image. To make the images realistic, we need to appropriately re-size the car based on its*

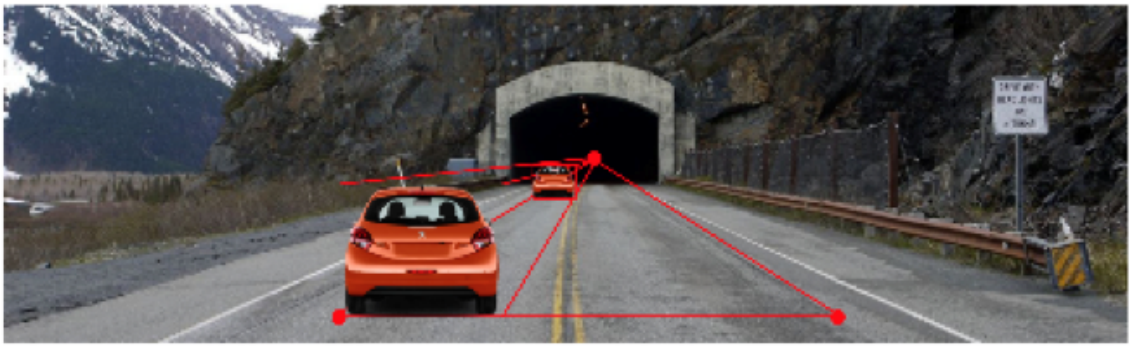


Figure 5.3. Car re-sizing and displacement using vanishing point and lines.

position on the road . In this example, we chose the extreme positions of the car (i.e., maximum and minimum x and y position of the car) on the sidelines of the road and the image vanishing point. Both the sidelines and the vanishing point can be automatically detected ([7, 87]). The vanishing point is useful to determine the vanishing lines necessary to re-size and place the car when altering its position in the y modification dimension. For instance, the car is placed and shrunk towards the vanishing point as the y coordinate of its modification element gets close to 1 (see Figure 5.3).

A generator can be used to abstract and compactly represent a subset of a high-dimensional image space.

In this chapter, the image generator is based on a 14D modification space whose dimensions determine a road background; number of cars (one, two or three) and their x and z positions on the road; brightness, sharpness, contrast, and color of the picture. Figure 5.4 depicts some images rendered by our image generator. (More details are available in [48]).

We can define a metric over the modification space to measure the diversity of different pictures. Intuitively, the distance between two configurations is large if the concretized images are visually diverse and, conversely, it is small if the concretized images are similar.

Let $\mathbf{m}^{(1)}, \mathbf{m}^{(2)} \in \mathbb{M}$ be modifications. The metric distance between the modifications is defined as:

$$d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = \sum_{i=1}^{4} \mathbf{1}_{m_i^{(1)} \neq m_i^{(2)}} + \|m_{5:14}^{(1)} - m_{5:14}^{(2)}\| \tag{5.1}$$

where $\mathbf{1}_{condition}$ is 1 if the condition is true, 0 otherwise, and $\|\cdot\|$ is the L^2 norm. The distance counts the differences between background and car models and adds the Euclidean distance of the points corresponding to x and z positions, brightness, sharpness, contrast, and color of the images.

Figure 5.4 depicts three images with their modifications $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$, and $\mathbf{m}^{(3)}$. For brevity, captions report only the dimensions that differ among the images, that are background, car models and x, z positions. The distances between the modifications are $d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$, $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$, $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$. Note how similar images, like Fig. 5.4 (a) and (b)

(a) $\mathbf{m}^{(1)} = (53, 25, 2, 0.11, 0.98, \ldots, 0.50, 0.41, \ldots)$

(b) $\mathbf{m}^{(2)} = (53, 25, 2, 0.11, 0.98, \ldots, 0.20, 0.80, \ldots)$

(c) $\mathbf{m}^{(3)} = (13, 25, 7, 0.11, 0.98, \ldots, 0.50, 0.41, \ldots)$

Figure 5.4. Distance over modification space used to measure visual diversity of concretized images. $d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$, $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$, $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$.

(same backgrounds and car models, slightly different car positions), have smaller distance ($d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) = 0.48$) than diverse images, like Fig. (a) and (c); or (b) and (c) (different backgrounds, car models, and vehicle positions), whose distances are $d(\mathbf{m}^{(1)}, \mathbf{m}^{(3)}) = 2.0$ and $d(\mathbf{m}^{(2)}, \mathbf{m}^{(3)}) = 2.48$.

Later on, we use this metric to generate sets whose elements ensure a certain amount of diversity (see Section 5.7.1).

5.4.2 Picture Concretization

Once a modification is fixed, our picture generator renders the corresponding image. The concretization is done by superimposing basic images (such as road background and vehicles) and adjusting image parameters (such as brightness, color, or contrast) accordingly to the values specified by the modification. Our image generator comes with a database of backgrounds and car models used as basic images. Our database consists of 35 road scenarios

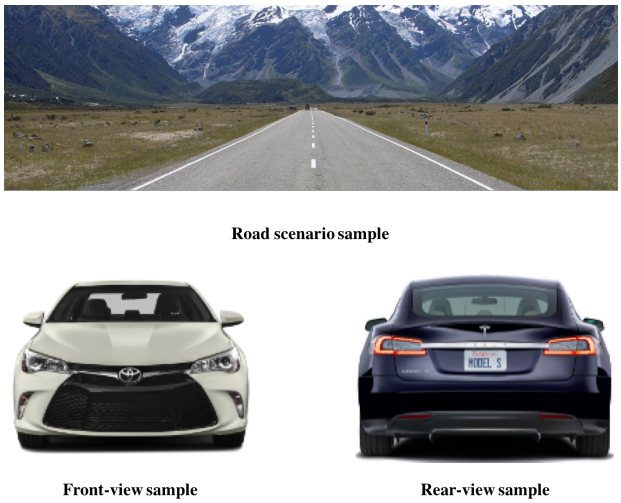


Figure 5.5. Sample basic images. The image at the top shows a road scenario sample, and the images at the bottom show car model samples.

(e.g., desert, forest, or freeway scenes) and 36 car models (e.g., economy, family, or sports vehicles, from both front and rear views). These basic images are superimposed in order to render a realistic images. Examples of basic pictures are shown in Figure 5.5. The database can be easily extended or replaced by the user.

5.4.3 Annotation Tool

In order to render realistic images, the picture generator must place cars on the road and scale them accordingly. For instance, a car must appear big when close to the observer but small when moved towards the vanishing point. To facilitate the conversion of a modification point describing x and z position into a proper superimposition of the car image on a road, we equipped the image generator with an annotation tool that can be used to specify the sampling area on a road and the scaling factor of a vehicle. For a particular road, the user draws a trapezoid designating the area where the image generator is allowed to place a car. The user also specifies the scale of the car image on the trapezoid bases, i.e., at the closest and furthest points from the observer (see Figure 5.6). When sampling a point at an intermediate position, i.e., inside the trapezoid, the tool interpolates the provided car scales and determines the scaling at the given point. Moreover, the image generator superimposes different vehicles respecting the perspective of the image. For instance, a car close to the observer will partially hide a car on the horizon. The image generator also performs several checks to ensure that the rendered cars are visible.

Figure 5.6. Annotation trapezoid. User adjusts the four corners that represent the valid sampling subspace of x and z. The size of the car scales according to how close it is to the vanishing point.

5.4.4 Image Generators as Simulation Engines

While the images in this section are generated by superimposing one image above another, the overall image may itself not be realistic. For e.g., when there are multiple cars on the road it is hard to generate images where they do not overlap since the position of one car depends on the position of the others. Moreover, it is impossible to define a modification space $\mathbb{M}$ with all potential backgrounds and cars. To overcome this we propose two modifications to our image generator:

- Using Scenic [56] to define constrained domains. Scenic use a probabilistic programming language to define distribution over scenes. It can handle geometric constraints or relations between objects (e.g., do not intersect, maintain some minimum distance, or angle). More details can be found in Chapter 8.
- Defining the modification space $\mathbb{M}$ to capture scenes in simulation engines like Webots [140], Carla [43] or game engines like GTA-V. This allows for generation of scenes which are more realistic and can capture a larger variation within the $\mathbb{M}$. More details can be found in Chapter8

5.5 Sampling Methods

In this section, we discuss the sampler in $\mathcal{O}_{\mathcal{C}_{da}}$ The goal of the sampler is to provide a good coverage of the modification space and identify samples whose concretizations lead to counterexamples.

We now briefly describe some sampling methods (similar to those defined in [48, 44]) that we integrated into our framework:

5.5.1 Non-active Sampling

These sampling techniques rely only on the modification space $\mathbb{M}$. They sample the space without using any prior information of the behavior of the previously sampled modifications.

5.5.1.1 Uniform Random Sampling

Uniform random sampling ensures an equal probability of sampling any possible point from $\mathbb{M}$, which guarantees a good mix of generated images for both training and testing. Although a simple and effective technique for both training as well as testing, it may not provide a good coverage of the modification space; Moreover, during testing it does not guarantee that the samples lead to misclassifying concretizations reducing the effectiveness of the sampling procedure. Random sampling can be applied to both discrete and continuous (or a mix) modification space $\mathbb{M}$. We also consider a variant of random sampling with distance constraints in our experiments, i.e., we impose that two samples $\mathbf{m}^{(1)}, \mathbf{m}^{(2)}$ are atleast some distance apart $d(\mathbf{m}^{(1)}, \mathbf{m}^{(2)}) \geq \epsilon$. This is implemented with rejection sampling, i.e., we reject any sample which is not atleast ϵ away from all previously sampled points.

5.5.1.2 Low-Discrepancy Sampling

A *low-discrepancy* (or quasi-random) sequence is a sequence of n-tuples that fills a nD space more uniformly than uncorrelated random points. Low-discrepancy sequences are useful to cover boxes by reducing gaps and clustering of points which ensures uniform coverage of the sample space.

Let $U = [0, 1]^n$ be a n-D box, $J \subseteq U$ be a sub-box and $X \in U$ be a set of m points. The *discrepancy*, $D(J, X)$, of J is the difference between the ratio of points in J compared to U and the ratio of volume J compared to U:

$$D(J, X) = |\#(J)/m - vol(J)| \tag{5.2}$$

where $\#(J)$ is the number of points of X in J and $vol(J)$ is the volume of J. The star discrepancy, $D^*(X)$, is the worst case distribution of X:

$$D^*(X) = \max_J D(J, X) \tag{5.3}$$

Low-discrepancy sequences generate sets of points that minimize the star-discrepancy. Some examples of low-discrepancy sequences are the Van der Corput, Halton [74], or Sobol [139] sequences. In our experiments, we use the Halton [114] sequence. These sampling methods ensure an optimal coverage of the modification space and allows us to identify clusters of misclassified pictures as well as isolated corner cases which can be missed by random sampling. There are two main advantages in having optimal coverage: first, we increase the chances of quickly discovering counterexamples, and second, the set of counterexamples will have high diversity; implying the concretized images will look different and thus the model will learn diverse new features.

The modification space $\mathbb{M}$ for Halton sampling is purely continuous (can be captured by a n-D box).

5.5.2 Active Sampling

These sampling techniques utilize the behavior of the previously sampled modifications to direct the sampling of future samples. This allows for faster convergence to counter-examples as opposed to searching the space more evenly. The active samplers searching for counter-examples can be represented by an instance of $\mathcal{I}_f$ described in Chapter 6. Hence, the overall CEGIS framework $\mathcal{C}_{da}$ is a hieraechical OGIS framework where the sampler is represented as $\mathcal{I}_f$.

5.5.3 Cross-Entropy Sampling

The *cross-entropy* [32] method was developed as a general Monte Carlo approach to combinatorial optimization and importance sampling. It is a iterative sampling technique, where we sample from a a given probability distribution, and update the distribution by minimizing the cross-entropy.

Intuitively, given a probability distribution P defined over the modification space $\mathbb{M}$, a set of counter-examples and its associated distribution Q, and a discounting factor α, we update the distribution $P = \alpha \cdot P + (1 - \alpha) \cdot Q$. The effect of the sampled counter-examples on the distribution P depends on the values of α. For low values of α, the distribution is affected by the distribution of the counter-examples Q and tends to "forget" P more easily. In this chapter, we assume that P is a discrete probability distribution. For continuous spaces, we achieve this by breaking the domains into "buckets" and treat all points in a given buckets as equivalent. This allows us to treat both discrete and continuous domains in $\mathbb{M}$ in the same framework.

One drawback of this technique is that it does not capture relationships between the dimensions of $\mathbb{M}$, it treats each of them independently. However,cross-entropy is especially well suited for discrete domains which are categorical i.e., no ordering in the domain.

The $\mathcal{I}_f$ framework for falsification (finding counter-examples) using cross entropy can be built similarly as detailed in Chapter 6. The key difference lying in the design of the learner $\mathcal{L}_f$. The $\mathcal{L}_f$ for cross-extropy implements the distribution update.

5.5.4 Bayesian Optimization

For $\mathbb{M}$ which are continuous in all dimensions (can be captured by n-D box), we can use Bayesian Optimization (BO) for sampling. Refer to Section 6.2.3 for details. BO can capture relations among the different dimensions in $\mathbb{M}$ in the co-variance matrix. We do not handle discrete domain in BO since they mostly appear to categorical in our setting. Hence, there may not always be a relationship across the domain. We do not results with BO since our 14-D modification space is a mix of continuous and discrete domains.

The $\mathcal{I}_f$ framework for falsification (finding counter-examples) using BO is detailed in Chapter 6.

Car model	Background ID	Environment	Brightness	x	z
Toyota	12	Tunnel	0.9	0.2	0.9
BMW	27	Forest	1.1	0.4	0.7
Toyota	11	Forest	1.2	0.4	0.8

Table 5.1. Example of error table proving information about counterexamples. First rows describes Fig. 5.6. Implicit unordered features: car model, environment; explicit ordered features: brightness, x, z car coordinates; explicit unordered feature: background ID.

5.6 Error Tables

Every iteration of our augmentation scheme produces a counterexample that contains information pointing to a limitation of the learned model. It would be desirable to extract patterns that relate counterexamples, and use this information to efficiently generate new counterexamples. For this reason, we define *error tables* that are data structures whose columns are formed by important features across the generated images. The error table analysis is useful for:

1. Providing *explanations* about counterexamples, and
2. Generating *feedback* to sample new counterexamples.

In the first case, by finding common patterns across counterexamples, we provide feedback to the user like "*The model does not detect white cars driving away from us in forest roads*"; in the second case, we can bias the sampler towards modifications that are more likely to lead to counterexamples.

5.6.1 Error Table Features

We first provide the details of the kinds of features supported by our error tables. We categorize features along two dimensions:

- *Explicit* vs. *implicit* features: Explicit features are sampled from the modification space (e.g., x, z position, brightness, contrast, etc.) whereas implicit features are user-provided aspects of the generated image (e.g., car model, background scene, etc.).
- *Ordered* vs. *unordered* features: some features have a domain with a well-defined total ordering (e.g., sharpness) whereas others do not have a notion of ordering and are purely categorical (e.g., car model, identifier of background scene, etc.).

The set of implicit and explicit features are mutually exclusive. In general, implicit features are more descriptive and characterize the generated images. These are useful for providing feedback to explain the vulnerabilities of the classifier. While implicit features are unordered, explicit features can be ordered or unordered. Rows of error tables are the realizations of the features for misclassification.

Table 5.1 is an illustrative error table $\mathbb{E}$. The table includes car model and environment scene (implicit unordered features), brightness, x, z car coordinates (explicit ordered features), and background ID (explicit unordered feature). The first row of Table 5.1 actually refers to Fig. 5.6. The actual error tables generated by our framework are larger than Table 5.1. They include, for instance, our 14D modification space (see Section 5.4.1) and features like number of vehicles, vehicle orientations, dominant color in the background, etc.

The error table $\mathbb{E}$ is populated in the oracle $\mathcal{O}_{\mathcal{C}_{da}}$ in every iteration whenever a counter-example is generated. Finally, we would like to analyze it to provide feedback and utilize this feedback to sample new images.

5.6.2 Feature Analysis

The analysis of the error table $\mathbb{E}$ is done in the oracle $\mathcal{O}_{\mathcal{C}_{da}}$ (in errorAnalysis($\mathbb{E}$)). A naive analysis technique is to treat all the features equally, and search for the most commonly occurring element in each column of the error table. However, this assumes no correlation between the features, which is often not the case. Instead, we develop separate analysis techniques for ordered and unordered features. In the following we discuss how we can best capture correlations between the two sets:

- *Ordered features*: Since these features are ordered, a meaningful analysis technique would be to find the direction in the feature space where most of the falsifying samples occur. This is very similar to model order reduction using Principal Component Analysis (PCA). Specifically, we are interested in the first principal component, which is the singular vector corresponding to the largest singular value in the Singular Value Decomposition (SVD) of the matrix consisting of all the samples of the ordered features. We can use the singular vector to find how sensitive the model is with respect to the ordered features. If the value corresponding to a feature is small in the vector, it implies that the model is not robust to changes in that feature, i.e., changes in that feature would affect the misclassification. Or alternatively, features corresponding to larger values in the singular vector, act as "don't cares", i.e., by fixing all other features, the model misclassifies the image regardless the value of this feature. Another meaningful analysis technique is to use clustering such as k-means [78] to find clusters of counter-examples.;

- *Unordered features*: Since these features are unordered, their value holds little importance. The most meaningful information we can gather from them is the subsets of features which occurs together most often. To correctly capture this, we must explore all possible subsets, which is a combinatorial problem. This proves to be problematic when the space of unordered features is large. One way to overcome this is by limiting the size of the maximum subset to explore.

We conducted an experiment on a set of 500 counterexamples. The ordered features included x and z positions of each car; along with the brightness, contrast, sharpness, and

color of the overall image. The explicit features include the ordered features along with the discrete set of all possible cars and backgrounds. The implicit features include details like color of the cars, color of the background, orientation of the cars, etc. The PCA on the explicit ordered features revealed high values corresponding to the x position of the first car (0.74), brightness (0.45) and contrast (0.44). We can conclude that the model is not robust to changes in these ordered features. Specifically, if we were to maintain the other features to values that appear the most in the error table, then by varying these features, we should be able to get counterexample images with high probability. Now suppose, among the ordered features, if the value corresponding to brightness is high in the principal component, then we can conclude that the CNN is not robust to brightness, and we must retrain the CNN by varying the brightness. For the unordered features, the combination of forest road with one white car with its rear towards the camera and the other cars facing the camera, appeared 13 times. This provides an explanation of recurrent elements in counterexamples, specifically "*The model does not detect white cars driving away from us in forest roads*"".

5.6.3 Sampling Using Feedback

In the oracle $\mathcal{O}_{\mathcal{C}_{da}}$, we can use the error table $\mathbb{E}$ to guide what samples can be used for augmentation (in generateAugSet($\mathbb{A}, \mathbb{E}, \gamma$)). One could either, sample from the augmentation set $\mathbb{A}$ or regenerate new sample using the image generator γ and analysis of the error table $\mathbb{E}$. In this section, we discuss how our analysis and feedback from the $\mathbb{E}$ can be used to guide the sampling for subsequent training.

Note that we can only sample from the explicit features:

- *Feedback from Ordered Features*: The ordered features, which is a subset of the explicit features, already tell us which features need to vary more during the sampling process. For example, in the example of Sec. 5.6.2, our sampler must prioritize sampling different x positions for the first car, then brightness, and finally contrast among the other ordered features;

- *Feedback from Unordered Features*: Let $\mathbb{S}_{uf} = \mathbb{S}_{ef} \cup \mathbb{S}_{if}$ be the subset of most occurring unordered features returned by the analysis, where $\mathbb{S}_{ef}$ and $\mathbb{S}_{if}$ are the mutually exclusive sets of explicit and implicit features, respectively. The information of $\mathbb{S}_{ef}$ can be directly incorporated into the sampler. The information provided by $\mathbb{S}_{if}$ require some reasoning since implicit features are not directly sampled. However, they are associated with particular elements of the image (e.g., background or vehicle). We can use the image generator library and error table to recognize which elements in the library the components of $\mathbb{S}_{if}$ correspond to, and set the feature values accordingly. For instance, in the example of Sec. 5.6.2, the analysis of the unordered explicit features revealed that the combination of bridge road with a Tesla, Mercedes, and Mazda was most often misclassified. We used this information to generate more images with this particular combination by varying brightness and contrast.

	$\mathbb{T}_R$	$\mathbb{T}_H$	$\mathbb{T}_C$	$\mathbb{T}_D$	$\mathbb{T}_M$
$f_{\mathbb{X}}$	0.6169 0.7429	0.6279 0.7556	0.3723 0.4871	0.7430 0.8373	0.6409 0.7632
$f_{\mathbb{X}_S}$	0.6912 0.8080	0.6817 0.7987	0.3917 0.5116	0.7824 0.8768	0.6994 0.8138
$f_{\mathbb{X}_R}$	0.7634 0.8667	0.7515 0.8673	0.5890 0.7242	0.8484 0.9745	0.7704 0.8818
$f_{\mathbb{X}_H}$	0.7918 0.8673	0.7842 0.8727	0.5640 0.6693	0.8654 0.9598	0.7980 0.8828
$f_{\mathbb{X}_C}$	0.7778 0.7804	0.7632 0.7722	0.6140 0.7013	0.8673 0.8540	0.7843 0.7874
$f_{\mathbb{X}_D}$	0.7516 0.8642	0.7563 0.8724	0.6057 0.7198	0.8678 0.9612	0.7670 0.8815

Table 5.2. Comparison of augmentation techniques. Precisions (top) and recalls (bottom) are reported. $\mathbb{T}_T$ set generated with sampling method T; $f_{\mathbb{X}_T}$ model f trained on $\mathbb{X}$ augmented with technique $T \in \{S, R, H, C, D, M\}$; S: standard, R: uniform random, H: low-discrepancy Halton, C: cross-entropy, D: uniform random with distance constraint, M: mix of all methods.

Sec 5.7.5 shows how this technique leads to a larger fraction of counterexamples that can be used for retraining. The augmentation set $\mathbb{A}$ is then returned to the learner $\mathcal{L}_{\mathcal{C}_{da}}$ to re-train the model.

5.7 Evaluation

In this section, we show how the proposed techniques can be used to augment training sets and improve the accuracy of the considered models. We will experiment with different sampling methods, compare counterexample guided augmentation against classic augmentation, iterate over several augmentation cycles, and finally show how error tables are useful tools for analyzing models. The implementation of the proposed framework and the reported experiments are available at `https://github.com/dreossi/analyzeNN`.

In all the experiments we analyzed squeezeDet [160], a CNN real-time object detector for autonomous driving. All models were trained for 65 epochs.

The original training and test sets $\mathbb{X}$ and $\mathbb{T}$ contain 1500 and 750 pictures, respectively, randomly generated by our image generator. The initial accuracy $acc_{f_{\mathbb{X}}}(\mathbb{T}) = (0.9847, 0.9843)$ is relatively high (see Table 5.4). However, we will be able to generate sets of counterexamples as large as $\mathbb{T}$ on which the accuracy of $f_{\mathbb{X}}$ drops down. The highlighted entries in the tables show the best performances. Reported values are the averages across five different experiments.

5.7.1 Augmentation Methods Comparison

As the first experiment, we run the counterexample augmentation scheme using different sampling techniques (see Section 5.5). Specifically, we consider uniform random sampling, low-discrepancy Halton sequence, cross-entropy sampling, and uniform random sampling with a diversity constraint on the sampled points. For the latter, we adopt the distance defined in Section 5.4.1 and we require that the modifications of the counterexamples must be at least distant by 0.5 from each other.

For every sampling method, we generate 1500 counterexamples, half of which are injected into the original training set $\mathbb{X}$ and half are used as test sets. Let R, H, C, D denote uniform random, Halton, cross-entropy, and diversity (i.e., random with distance constraint) sampling methods. Let $T \in \{R, H, C, D\}$ be a sampling technique. $\mathbb{X}_T$ is the augmentation of $\mathbb{X}$, and $\mathbb{T}_T$ is a test set, both generated using T. For completeness, we also defined the test set $\mathbb{T}_M$ containing an equal mix of counterexamples generated by all the R, H, C, D sampling methods.

Table 5.2 reports the accuracy of the models trained with various augmentation sets evaluated on test sets of counterexamples generated with different sampling techniques. The first row reports the accuracy of the model $f_{\mathbb{X}}$ trained on the original training set $\mathbb{X}$. Note that, despite the high accuracy of the model on the original test set ($acc_{f_{\mathbb{X}}}(\mathbb{T}) = (0.9847, 0.9843)$), we were able to generate several test sets from the same distribution of $\mathbb{X}$ and $\mathbb{T}$ on which the model poorly performs.

The first augmentation that we consider is the standard one, i.e., we alter the images of $\mathbb{X}$ using `imgaug`[1], a Python library for images augmentation. We augmented 50% of the images in $\mathbb{X}$ by randomly cropping $10 - 20\%$ on each side, flipping horizontally with probability 60%, and applying Gaussian blur with $\sigma \in [0.0, 3.0]$. Standard augmentation improves the accuracies on every test set. The average precision and recall improvements on the various test sets are 4.91% and 4.46%, respectively (see Row 1 Table 5.2).

Next, we augment the original training set $\mathbb{X}$ with our counterexample-guided schemes (uniform random, low-discrepancy Halton, cross-entropy, and random with distance constraint) and test the retrained models on the various test sets. The average precision and recall improvements for uniform random are 14.43% and 14.56%, for low-discrepancy Halton 16.05% and 14.57%, for cross-entropy 16.11% and 6.18%, and for random with distance constraint 14.95% and 14.26%. First, notice the improvement in the accuracy of the original model using counterexample-guided augmentation methods is larger compared to the classic augmentation method. Second, among the proposed techniques, cross-entropy has the highest improvement in precision but low-discrepancy tends to perform better than the other methods in average for both precision and recall. This is due to the fact that low-discrepancy sequences ensure more diversity on the samples than the other techniques, resulting in different pictures from which the model can learn new features or reinforce the weak ones.

The generation of a counterexample for the original model $f_{\mathbb{X}}$ takes in average for uniform random sampling 30s, for Halton 92s, and for uniform random sampling with constraints 55s.

[1]imgaug: `https://github.com/aleju/imgaug`

f_X	f_{X_r}	f_{X_d}
0.6957	0.8392	**0.8678**
0.7982	0.9371	**0.9612**

Table 5.3. Random vs Distance augmentation.

This shows the trade-off between time and gain in model accuracy. The maximization of the diversity of the augmentation set (and the subsequent accuracy increase) requires more iterations.

5.7.2 Random vs Low-discrepancy Sampling

In the case study we analyze how a different sampling method affects the augmentation. Let M_r and M_H be sets two sets of 1500 misclassified images generated using uniform random and Halton low-discrepancy sampling methods, respectively (see Section 5.5). We randomly split M_r and M_H in halves. We use two halves as test set T while the other two to augment the original training set. Let X_r and X_H be X augmented with M_r and M_H halves, respectively.

5.7.3 Random vs Diversity Augmentation

In this experiment we study the effects of training sets augmented by images that satisfy a certain diversity condition. Intuitively, we want to verify whether a set augmented with diverse images differs from one augmented by pictures randomly generated.

In this experiment we considered the model f_X trained on the original data set X. Next, we defined two new training sets $X_r = X \cup M_r$ and $X_d = X \cup M_d$ where M_r is a set of 250 misclassifying images randomly generated and M_d is a set of 250 misclassifying images such that for every $\mathbf{x}^{(i)}, \mathbf{x}^{(j)} \in M_d$, $d\mathbf{m}^{(i)}\mathbf{m}^{(j)} \geq 0.5$, where $\mathbf{x}^{(i)} = \gamma(\mathbf{m}^{(i)})$ and $\mathbf{x}^{(j)} = \gamma(\mathbf{m}^{(j)})$. In other words, the modification of an image in M_d is far at least by 0.5 from every configuration of images in M_d. By doing so, we make sure that the images in M_d are visually different.

We trained models trained on X_r and X_d and tested them against the test set $M_T^{[0]}$. The obtained results are shown in Table 5.3. The tables shows the averages of average precision and recall over five different experiments.

Notice how both the augmentation techniques improve the accuracy of the original model. However, ensuring diversity over the augmentation set leads to a more accurate model.

5.7.4 Augmentation Loop

For this experiment, we consider only the uniform random sampling method and we incrementally augment the training set over several augmentation loops, i.e., over several iteration of $\mathcal{C}_{da}$. In this section, we consider the augmentation set $\mathbb{A}$ to be a subset of the augmentation set returned by the oracle $\mathcal{O}_{\mathcal{C}_{da}}$ when it used random sampling. Here we do not use any feedback extracted from the error table to guide the sampling. The goal of this experiments

is to understand whether the model overfits the counterexamples and see if it is possible to reach a saturation point, i.e., a model for which we are not able to generate counterexamples. We are also interested in investigating the relationship between the quantity of injected counterexamples and the accuracy of the model.

Consider the i-th augmentation cycle. For every augmentation round, we generate the set of counterexamples by considering the model $f_{\mathbb{X}_r^{[i]}}$ with highest average precision and recall. Given $\mathbb{X}^{[i]}$, our analysis tool generates a set $\mathbb{C}^{[i]}$ of counterexamples. We split $\mathbb{C}^{[i]}$ in halves $\mathbb{C}_{\mathbb{X}}^{[i]}$ and $\mathbb{C}_{\mathbb{T}}^{[i]}$. We use $\mathbb{C}_{\mathbb{X}}^{[i]}$ to augment the original training set $\mathbb{X}^{[i]}$ and $\mathbb{C}_{\mathbb{T}}^{[i]}$ as a test set. Specifically, the augmented training set $\mathbb{X}_{r'}^{[i+1]} = \mathbb{X}_r^{[i]} \cup \mathbb{C}_{\mathbb{X}}^{[i]}$ is obtained by adding misclassified images of $\mathbb{C}_{\mathbb{X}}^{[i]}$ to $\mathbb{X}^{[i]}$. r, r' are the ratios of misclassified images to original training examples. For instance, $|\mathbb{X}_{0.08}| = |\mathbb{X}| + 0.08 * |\mathbb{X}|$, where $|\mathbb{X}|$ is the cardinality of $\mathbb{X}$. We consider the ratios $0.08, 0.17, 0.35, 0.50$. We evaluate every model against every test set.

Table 5.4 shows the accuracies for three augmentation cycles. For each model, the table shows the average precision and recall with respect to the original test set $\mathbb{T}$ and the tests sets of misclassified images. The generation of the first loop took around 6 hours, the second 14 hours, the third 26 hours. We stopped the fourth cycle after more than 50 hours. This shows how *it is increasingly hard to generate counterexamples for models trained over several augmentations*. This growing computational hardness of counterexample generation with the number of cycles is an informal, empirical measure of increasing assurance in the machine learning model.

Notice that for every cycle, our augmentation improves the accuracy of the model with respect to the test set. Even more interesting is the fact that the model accuracy on the original test set does not decrease, but actually improves over time (at least for the chosen augmentation ratios).

5.7.5 Error Table-Guided Sampling

In this last experimental evaluation, we use error tables to analyze the counterexamples generated for $f_{\mathbb{X}}$ with uniform random sampling, i.e., we generate new images using γ based on the feedback from $\mathbb{E}$ in the learner $\mathcal{L}_{\mathcal{C}_{da}}$ (in generateAugSet$(\mathbb{A}, \mathbb{E}, \gamma)$). We analyzed both the ordered and unordered features (see Section 5.6.2). The PCA analysis of ordered features revealed the following relevant values: sharpness 0.28, contrast 0.33, brightness 0.44, and x position 0.77. This tells us that the model is more sensitive to image alterations rather than to the disposition of its elements. The occurrence counting of unordered features revealed that the top three most occurring car models in misclassifications are white Porsche, yellow Corvette, and light green Fiat. It is interesting to note that all these models have uncommon designs if compared to popular cars. The top three most recurring background scenes are a narrow bridge in a forest, an indoor parking lot, and a downtown city environment. All these scenarios are characterized by a high density of details that lead to false positives. Using the gathered information, we narrowed the sampler space to the subsets of the modification space identified by the error table analysis. The counterexample generator was able to produce

	$\mathbb{T}$	$\mathbb{C}_{\mathbb{T}}^{[1]}$	$\mathbb{C}_{\mathbb{T}}^{[2]}$	$\mathbb{C}_{\mathbb{T}}^{[3]}$
$f_{\mathbb{X}}$	0.9847 0.9843	0.6957 0.7982		
$f_{\mathbb{X}_{0.08}^{[1]}}$	0.9842 0.9861	0.7630 0.8714		
$f_{\mathbb{X}_{0.17}^{[1]}}$	0.9882 0.9905	0.8197 0.9218	0.5922 0.8405	
$f_{\mathbb{X}_{0.35}^{[1]}}$	0.9843 0.9906	0.8229 0.9110		
$f_{\mathbb{X}_{0.50}^{[1]}}$	0.9872 0.9912	0.7998 0.9149		
$f_{\mathbb{X}_{0.08}^{[2]}}$	0.9947 0.9955	0.7286 0.8691	0.7159 0.8612	
$f_{\mathbb{X}_{0.17}^{[2]}}$	0.9947 0.9954	0.7424 0.8422	0.7288 0.8628	
$f_{\mathbb{X}_{0.35}^{[2]}}$	0.9926 0.9958	0.7732 0.8720	0.7570 0.8762	
$f_{\mathbb{X}_{0.50}^{[2]}}$	0.9900 0.9942	0.8645 0.9339	0.8223 0.9187	0.5308 0.7017
$f_{\mathbb{X}_{0.08}^{[3]}}$	0.9889 0.9972	0.7105 0.8571	0.7727 0.8987	0.7580 0.8860
$f_{\mathbb{X}_{0.17}^{[3]}}$	0.9965 0.9970	0.8377 0.9116	0.8098 0.9036	0.8791 0.9473
$f_{\mathbb{X}_{0.35}^{[3]}}$	0.9829 0.9937	0.7274 0.8060	0.8092 0.8816	0.7701 0.8480
$f_{\mathbb{X}_{0.50}^{[3]}}$	0.9905 0.9955	0.7513 0.8813	0.7891 0.9573	0.7902 0.9169

Table 5.4. Augmentation loop. For the best (highlighted) model, a test set $\mathbb{C}_{\mathbb{T}}^{[i]}$ and augmented training set $\mathbb{X}_r^{[i+1]}$ are generated. r is the ratio of counterexamples to the original training set.

329 misclassification with 10k iterations, against the 103 of pure uniform random sampling, 287 of Halton, and 96 of uniform random with distance constraint.

Finally, we retrained f on the training set $\mathbb{X}_E$ that includes 250 images generated using the error table analysis. The obtained accuracies are,

$$\begin{aligned} acc_{f_{\mathbb{X}_E}}(\mathbb{T}_R) &= (0.7490, 0.8664) \\ acc_{f_{\mathbb{X}_E}}(\mathbb{T}_H) &= (0.7582, 0.8751) \\ acc_{f_{\mathbb{X}_E}}(\mathbb{T}_D) &= (0.8402, 0.9438) \\ acc_{f_{\mathbb{X}_E}}(\mathbb{T}_M) &= (0.7659, 0.8811) \end{aligned}$$

Note how error table-guided sampling reaches levels of accuracy comparable to other counter-example guided augmentation schemes (see Table 5.2) but with a third of augmenting images.

5.8 Conclusion

In this chapter, we proposed a novel CEGIS framework to design data sets for perception modules. We considered the specific example of a perception module on a self-driving car which detects cars on the road. We first presented an image generator which could synthesize images by sampling from a low dimensional feature space. We then presented out CEGDA framework which generates synthetic counter-example images which are populated and analyzed in an error table. Finally, we showed that augmenting the counter-examples and retraining, we are able to increase the accuracy of the model.

Chapter 6

Simulation Guided Falsification and Verification

6.1 Introduction

In recent years, research in control theory and robotics has focused on developing efficient controllers for robots that operate in the real world. Controller synthesis techniques such as reinforcement learning, optimal control, and model predictive control have been used to synthesize complex policies. However, if there is a large amount of uncertainty about the real world environment that the system interacts with, the robustness of the synthesized controller becomes critical. This is particularly true in *safety-critical* systems, where the actions of an autonomous agent may affect human lives.

We have addressed techniques for correct-by construction controller synthesis and specification repair in Chapters 3 and 4. In this chapter we study the problem of provably verifying the properties of controllers in simulation against uncertainties that are introduced in the design process.

Historically, designing robust controllers has been considered in control theory [131, 144]. A common issue with these techniques is that, although they consider uncertainty, they rely on simple linear models of the underlying system. This means that resulting controllers are often either overly conservative or violate safety constraints if they fail to capture nonlinear effects.

For nonlinear models with complex dynamics, reinforcement learning has been successful for synthesizing high fidelity controllers. Recently, algorithms based on reinforcement learning that can handle uncertainty have been proposed [83, 115, 125], where the performance is measured in expectation. A fundamental issue with learned controllers is that it is difficult to provide formal guarantees for safety in the presence of uncertainty. For example, a controller for an autonomous vehicle must consider human driver behaviors, pedestrian behaviors, traffic lights, uncertainty due to sensors, etc. Without formally verifying that these controllers are indeed safe, deploying them on the road could lead to loss of property

or human lives.

6.1.1 Verification and Falsification

A *verification* engine attempts to prove that when a system $\mathcal{S}$ composed with an environment $\mathcal{E}$ exhibits behaviors which satisfy a specification (or property) φ [30]. Mathematically, we could like to prove

$$\mathcal{S} \| \mathcal{E} \models \varphi$$

As a result, we either have a formal mathematical proof or we have counterexamples, i.e., $e \in \mathcal{E}$ where the $\mathcal{S}$ behavior does not satisfy φ. For systems with complex non-linear dynamics, reachability algorithms based on level set methods have been used to approximate backward reachable sets for safety verification [109, 107, 106]. A more detailed review of reachability analysis can be found in [11]. Recently tools such as C2E2 [50] and Flow* [25] have been developed to verify continuous and hybrid models with non-linear dynamics and discrete transitions against bounded time invariant properties using approximation. However, these methods suffer from three major drawbacks: (1) the curse of dimensionality of the state space, which limits them to low-dimensional systems; (2) the complexity of $\mathcal{S}$ and $\mathcal{E}$ and (2) *a priori* knowledge of the system dynamics.

A dual, and often simpler, problem is *falsification*, which tests the system within a set of environment conditions for adversarial examples. A falsification engine over comes the drawbacks of modeling the system, by relying on black-box simulations of the system. Testing black-box systems in simulators is a well studied problem in the formal methods community [39, 8, 163]. Recently, [48] have focused on testing of closed-loop safety critical systems with neural networks by finding "meaningful" perturbations. The heart of research in black-box testing focuses on developing smarter search techniques which efficiently samples the uncertainty space. Indeed, in recent years, several sequential search algorithms based on heuristics such as Simulated Annealing [8], Tabu search [37], and CMA-ES [75] have been suggested. Although these algorithms sample the uncertainty space efficiently, they do not utilize any of the information gathered during previous simulations.

A recent active learning approach based on Bayesian Optimization (BO) [111] an optimization method that aims to find the global optimum of an *a priori* unknown function based on noisy evaluations. Typically, BO algorithms are based on Gaussian Process (GP [129]) models of the underlying function and certain algorithms provably converge close to the global optimum [142]. In the testing setting, BO has been used to actively find counter examples by treating the search problem as a minimization problem in [36] over adversarial control signals. However, the authors do not consider the structure of the problem and thereby violate the smoothness assumptions made by the GP model. As a result, their methods are slow to converge or may fail to find counterexamples. In [113], the authors generate test scenarios for a black box autonomous system that demonstrate critical transitions in its performance modes by employing adaptive sampling over GP regression models.

The results in this chapter are adapted from [65].

6.2 Preliminaries

6.2.1 Mathematical Preliminaries

Let us represent the physical plant or the system by $\mathcal{S}$ and the controller by π. The closed-loop system $\mathcal{S}_\pi$ is composed of the physical plant and the associated controller. We assume that we have access to a simulation of the system that includes the control strategy π, i.e., the closed-loop system. For brevity, $\mathcal{S}$ represents the closed loop system in the remaining of this chapter. Since, the simulator simulates the system, we will use $\mathcal{S}$ to represent the simulator (or the system). The system $\mathcal{S}$ operates in a given environment $e \in \mathcal{E}$. We assume that the environment $\mathcal{E}$ models all sources of uncertainty. For example, they can represent environment effects such as weather, non-deterministic components such as other agents interacting with the system, or uncertain parameters of the physical system, e.g., friction, un-modeled sub components, initial states. The simulator simulates the the closed-loop system in a given environment to produce a finite-horizon trajectory. Let us denote by $\xi_\mathcal{S}(t; e)$ the trajectory of the system $\mathcal{S}$ at time t in environment e. Moreover, let Ξ denote the set of all finite horizon trajectories of the system. Formally, the system (or the simulator) can be represented as a function $\mathcal{S} : \mathcal{E} \to \Xi$ producing the trajectory $\xi_\mathcal{S}(\cdot; e)$.

We specify the safety specification by φ. They are defined on finite-length trajectories $\xi_\mathcal{S}(\cdot; e)$ of the system that can be obtained by simulating the system for a given set of environment parameters $e \in \mathcal{E}$. Alternatively, one can imagine φ to be set of all finite-horizon trajectories of the system that satisfy the system level-safety specification; $\varphi \subseteq \Xi$. For example, φ can encode state or input constraints that have to be satisfied over time. We say the system behavior in a given environment $e \in \mathcal{E}$ satisfies the specification φ, i.e., $\xi_\mathcal{S}(\cdot; e) \models \varphi$ if and only if $\xi_\mathcal{S}(\cdot; e) \in \varphi$.

Mathematically, we can represent the specification as a function $\varphi : \Xi \to \mathbb{B}$ where $\mathbb{B}$ is domain of booleans. φ evaluates the system behavior (trajectory) $\xi_\mathcal{S}$ to True (False) if the trajectory $\models$ ($\not\models$)φ. the The specification is a combination of individual predicates $\mu > 0$ which are functions of trajectories combined using a grammar of logical operations,

$$\varphi := \mu | \neg\mu | \varphi \wedge \psi | \varphi \vee \psi$$

The operation $\neg, \wedge, \vee$ represent negation, conjunction (and) and disjunction (or) respectively. These basic operations can be combined to define complex boolean formula such as implication $\to$ and if-and-only-if $\leftrightarrow$ using the rules,

$$\varphi \to \psi := \neg\varphi \vee \psi, \text{ and } \varphi \leftrightarrow \psi := (\neg\varphi \wedge \neg\psi) \vee (\varphi \wedge \psi). \tag{6.1}$$

While, the definition of φ closely resembles temporal logic [123, 104], the key difference being the predicates in our setting are defined over trajectories and are hence path properties as opposed to state properties in the temporal logic setting.

We further assume that the predicates allows for *quantitative semantics* which are continuous and smooth functions of the trajectories $\xi \in \Xi$, $\rho_\mu : \Xi \to \mathbb{R}$ such that,

$$\xi_{\mathcal{S}}(\cdot; e) \models \mu \leftrightarrow \rho_\mu(\xi_{\mathcal{S}}(\cdot; e)) > 0$$
$$\xi_{\mathcal{S}}(\cdot; e) \not\models \mu \leftrightarrow \rho_\mu(\xi_{\mathcal{S}}(\cdot; e)) < 0$$

The quantitative semantics of the predicates μ can be used to define the quantitative semantics of the safety spec φ by following the rules,

$$\begin{aligned} \rho_{\mu>0}(\xi) &:= \mu(\xi) \\ \rho_{\neg\mu}(\xi) &:= -\mu(\xi) \\ \rho_{\varphi\wedge\psi}(\xi) &:= \min(\rho_\varphi(\xi), \rho_\psi(\xi)) \\ \rho_{\varphi\vee\psi}(\xi) &:= \max(\rho_\varphi(\xi), \rho_\psi(\xi)) \end{aligned} \tag{6.2}$$

By evaluating ρ_φ on the system behavior in a given environment e, $\xi_{\mathcal{S}}(\cdot; e)$, we can comment on the satisfaction of the system behavior and hence, the safety of the corresponding environment e. Typically, $\rho_\varphi(\xi) = 0$ is considered to be an unknown behavior and hence, we cannot comment on the satisfaction of ξ. One would have to then evaluate the boolean satisfaction by checking if the $\xi \in \varphi$. In this work, we take a pessimistic approach and consider $\rho_\varphi(\xi) = 0$ to imply unsatisfactory behavior. This allows for behaviors that are atleast $\epsilon > 0$ robust, which is a valid assumption to make while evaluating the safety of the system (and the controller π). We further require ρ_φ to be allow a total ordering in the Ξ, i.e., if $\rho_\varphi(\xi_1) > \rho_\varphi(\xi_2)$ then ξ_1 is said to be more *safe* compared to ξ_2, and hence is a more desirable behavior. This allows for an ordering among the environment $\mathcal{E}$. If ξ_1 (ξ_2) was generated in e_1 (e_2), then we can say e_2 is a more "dangerous" environment compared to e_1. We can further confirm that a logic statement φ holds true for all trajectories generated by simulator in all environment $\mathcal{E}$, by confirming that the $\rho_\varphi(\xi_{\mathcal{S}}(\cdot; e))$ takes positive values for all $e \in \mathcal{E}$.

For brevity, we show the dependence of φ on the environment and use $\rho_\varphi(e)$ instead of $\rho_\varphi(\xi_{\mathcal{S}}(\cdot; e))$.

6.2.2 Gaussian Process

For general black-box systems, the dependence of the specification $\rho_\varphi(\cdot)$ on the parameters $e \in \mathcal{E}$ is unknown *a priori*. We use a GP to approximate each predicate $\rho_\mu(\cdot)$ in the domain $\mathcal{E}$. We detail the modeling of $\rho_\varphi(\cdot)$ in Section 6.5. The following introduction about GPs is based on [129].

GPs are non-parametric regression method from machine learning, where the goal is to find an approximation of the nonlinear function $\rho_\mu : \mathcal{E} \to \mathbb{R}$ from an environment $e \in \mathcal{E}$ to the function value ρ_μ. This is done by considering the function values $\rho_\mu(e)$ to be random variables, such that any finite number of them have a joint Gaussian distribution.

The Bayesian, non-parametric regression is based on a prior mean function and the kernel function $k(e, e')$, which defines the covariance between the function values $\rho_\mu(e), \rho_\mu(e')$ at two

points $e, e' \in \mathcal{E}$. We set the prior mean to zero, since we do not have any knowledge about the system. The choice of kernel function is problem-dependent and encodes assumptions about the unknown function.

We can obtain the posterior distribution of a function value $\rho_\mu(e)$ at an arbitrary state $e \in \mathcal{E}$ by conditioning the GP distribution of ρ_μ on a set of n past measurements, $\mathbf{y}_n = (\hat{\rho}_\mu(e_1), \ldots, \hat{\rho}_\mu(e_n))$ at environment scenarios $W_n = \{e_1, \ldots, e_n\}$, where $\hat{\rho}_\mu(e) = \rho_\mu(e) + \omega$ and $\omega \sim \mathcal{N}(0, \sigma^2)$ is Gaussian noise. The posterior over $\rho_\mu(e)$ is a GP distribution again, with mean $m_n(e)$, covariance $k_n(e, e')$, and variance $\sigma_n(e)$:

$$\begin{aligned} m_n(e) &= \mathbf{k}_n(e)(\mathbf{K}_n + \mathbf{I}_n\sigma^2)^{-1}\mathbf{y}_n, \\ k_n(e, e') &= k(e, e') - \mathbf{k}_n(e)(\mathbf{K}_n + \mathbf{I}_n\sigma^2)^{-1}\mathbf{k}_n^T(e'), \\ \sigma_n^2(e) &= k_n(e, e'), \end{aligned} \tag{6.3}$$

where the vector $\mathbf{k}_n(e) = [k(e, e_1), \ldots, k(e, e_n)]$ contains the covariances between the new environment, e, and the environment scenarios in W_n, the kernel matrix $\mathbf{K}_n \in \mathbb{R}^{n \times n}$ has entries $[\mathbf{K}_n](i, j) = k(e_i, e_j)$, with $i, j \in \{1, \ldots, n\}$, and $\mathbf{I}_n \in \mathbb{R}^{n \times n}$ is the identity matrix.

6.2.3 Bayesian Optimization

In the following we use BO in order to find the minimum of the unknown function ρ_φ, which we construct using the GP models on ρ_μ in Section 6.5. BO uses a GP model to query parameters that are informative about the minimum of the function. In particular, the GP-LCB algorithm from [142] uses the GP prediction and associated uncertainty in (6.3) to trade off exploration and exploitation by, at iteration n, selecting an environment according to

$$e_n = \operatorname*{argmin}_{e \in \mathcal{E}} m_{n-1}(e) - \beta_n^{1/2}\sigma_{n-1}(e), \tag{6.4}$$

where β_n determines the confidence interval. We provide an appropriate choice for β_n in Theorem 8.

At each iteration, (6.4) selects parameters for which the lower confidence bound of the GP is minimal. Repeatedly evaluating the true function ρ_φ at samples given by (6.4) improves the GP model and decreases uncertainty at candidate locations for the minimum, such that the global minimum is found eventually [142].

6.3 Problem Formulation

We address the problem of testing complex black-box closed-loop systems, which are composed of the physical plant $\mathcal{S}$ and the controller π, in simulation.

The goal is to test whether the closed loop-system remains safe for all possible sources of uncertainty in the environment $\mathcal{E}$.

We would like to test whether there exists an adversarial example $e \in \mathcal{E}$ for which the specification is violated, i.e., $\rho_\varphi(e) < 0$.

Typically, adversarial examples are found by randomly sampling the environment and simulating the behaviors. However, this approach does not provide any guarantees and does not allow us to conclude that no adversarial example exist if none are found in our samples. Moreover, since high-fidelity simulations can often be very expensive, we want to minimize the number of simulations that we have to carry out in order to find a counterexample.

We propose an active learning framework for testing, where we utilize the results from previous simulation runs to make more informed decisions about which environment to simulate next. In particular, we pose the search problem for a counterexample as an optimization problem,

$$\underset{e \in \mathcal{E}}{\operatorname{argmin}} \ \rho_\varphi(e), \tag{6.5}$$

where we want to minimize the number of queries e until a counterexample is found or we can verify that no counterexample exists. The main challenge is that the functional dependence $\rho_\varphi(\cdot)$ between parameters in $\mathcal{E}$ and the specification is unknown *a priori*, since we treat the simulator as a black-box. Solving this problem is difficult in general, but we can exploit regularity properties of $\rho_\varphi(e)$. In particular, in the following we use GP to model the specification and use the model to pick parameters that are likely to be counterexamples.

6.4 Solution Approach

We would like to test the controller safety under uncertainty that arises from stochastic environments and errors in modeling. In this chapter, we design a falsification engine for learned controllers in simulation. We formulate our solution as an instance of Oracle-Guided Inductive Synthesis (OGIS) [80]. Specifically, we design $\mathcal{I}_f = (\mathcal{L}_f, \mathcal{O}_f)$ into the design of the learner $\mathcal{L}_f$ and the oracle $\mathcal{O}_f$. We then show that under mild assumptions made about the system under analysis, our falsification engine can provide probabilistic verification guarantees. The overall OGIS framework $\mathcal{I}_f = (\mathcal{L}_f, \mathcal{O}_f)$ is presented in Fig 6.1 We provide a short description of the oracle $\mathcal{O}_f$ and the the learner $\mathcal{L}_f$ below.

Learner Design: The learner $\mathcal{L}_f$ models the unknown quantitative satisfaction landscape ρ_φ as a parse tree $\mathcal{T}_\varphi$ whose leaves (the predicates ρ_μ) are modeled as GPs. The learner $\mathcal{L}_f$ uses the trajectory returned by the oracle $\mathcal{O}_f$, $\xi_\mathcal{S}(\cdot; e)$ to update the GP models. It returns the updated parse tree with the updates models to the oracle. The concept class is the set of all function realizations of $\rho_\varphi(\cdot)$.

Oracle Design: In this setting, the oracle $\mathcal{O}_f$ is a composition of two oracles detailed below. **Simulation oracle** is a high-fidelity black-box simulator of the closed-loop system $\mathcal{S}$. It takes as input an environment configuration, and simulates the closed-loop system $\mathcal{S}$ to produce a finite horizon trajectory $\xi_\mathcal{S}(\cdot; e)$ which is sent to the learner. Hence, our framework is compatible with any software-in-the loop simulators like MATLAB or Simulink, robotics simulators like OpenAI Gym [17], Webots [140], CARLA [43]. The query made to

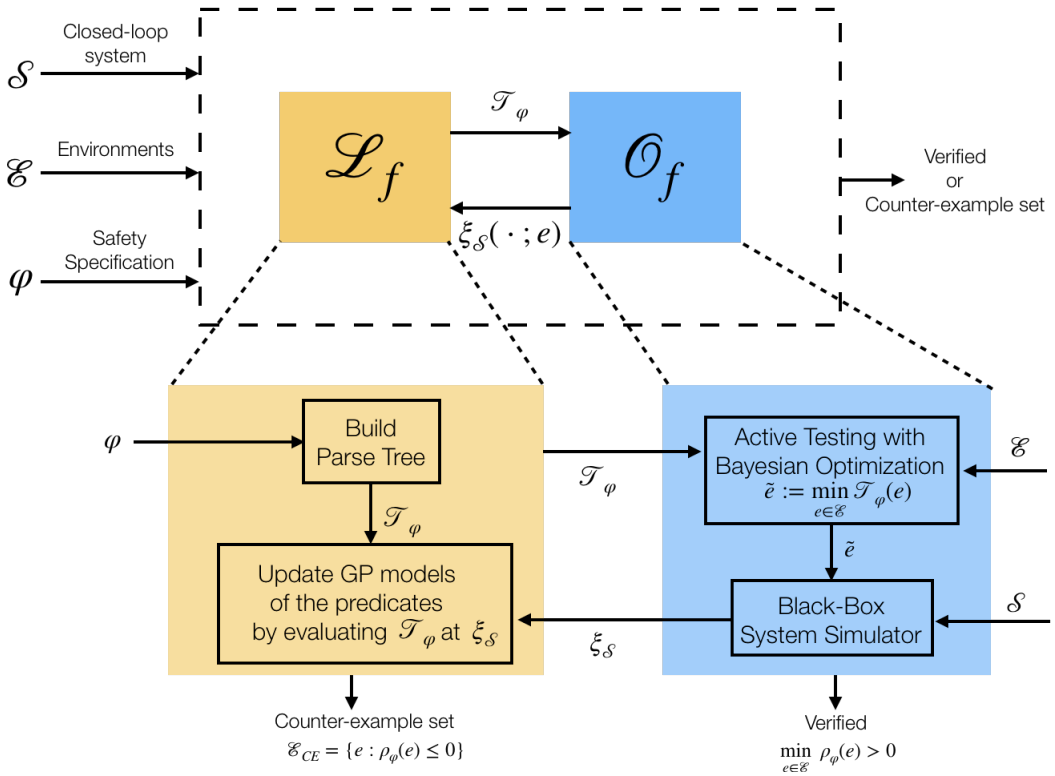


Figure 6.1. OGIS for Falsification $\mathcal{I}_f = (\mathcal{L}_f, \mathcal{O}_f)$. The oracle $\mathcal{O}_f$ (learner $\mathcal{L}_f$) is shown in blue (yellow). The oracle $\mathcal{O}_f$ is a composition of two individual oracles, (i) the first is an active-learning framework to propose candidate environment e for simulation; and (ii) a black-box simulation engine which generates finite-horizon trajectories $\xi_{\mathcal{S}}(\cdot; e)$ of the closed-loop system $\mathcal{S}$ in a given environment configuration e. The learner $\mathcal{L}_f$ first builds a parse tree corresponding to the safety specification $\mathcal{T}_\varphi$ whose leaf nodes are GPs modeling the predicates. At every iteration, the learner uses the system trajectory returned by the oracle to update the GP models of the predicates. It then sends the updated tree with the updated GP models $\mathcal{T}_\varphi$ to the oracle.

this oracle is similar to the simulation query q_{sim} described in Section 2.1. **Optimization Oracle** actively searches for adversarial environments under which the controller could have to operate that lead failure modes in simulation. It takes as input a parse tree $\mathcal{T}_\varphi$ and outputs an environment $e \in Env$ for simulation. We optimize the learner to minimize the number of queries (simulations) it requests the oracle $\mathcal{O}_f$. We design our oracle to use a global optimization BO. At each iteration, it uses the updated parse tree proposed by the learner to estimate the true minimum of the function ρ_φ and proposes it to the simulation oracle. The details are presented in Sec 6.5. The oracle here is an optimization oracle. Given a parse tree from the learner, this oracles responds by with an environment which it conjectures minimizes the optimization problem.

The framework terminates when the optimization oracle learns the unknown function $\rho_\varphi(e)$ with high confidence to find the true minimum $e^* = \text{argmin}_{e \in \mathcal{E}} \rho_\varphi(e)$. If $\rho_\varphi(e^*) > 0$, then the closed-loop system has been verified, i.e., $\forall e \in Env\ \rho_\varphi(e) > 0$. Or else, the

learner $\mathcal{L}_f$ returns a set of counter-examples (adversarial examples) $\mathcal{E}' \subseteq \mathcal{E}$ such that $\forall e \in \mathcal{E}_{CE}\ \rho_\varphi(e) \leq 0$. The counter-example set $\mathcal{E}_{CE}$ also contains the "worst-case" counterexample e^* where $\rho_\varphi(e^*) = \min_{e\in\mathcal{E}} \rho_\varphi(e)$.

6.5 Active-Learning for Falsification

In this section, we show how to model specifications ρ_φ in (6.5) using GPs without violating smoothness assumptions and use this to find adversarial counterexamples.

In order to use BO to optimize (6.5), we need to construct reliable confidence intervals on ρ_φ. However, if we were to model ρ_φ as a GP with commonly-used kernels, it would need it to be a smooth function of e. Even though the predicates, ρ_μ, are typically smooth functions of the trajectories, and hence smooth in e, conjunction and disjunction (min and max) in (6.2) are non-smooth operators that render ρ_φ to become non-smooth as well. Instead, we exploit the structure of the specification φ and decompose φ into a parse tree, where the leaf nodes correspond to the quantitative semantics function of the predicates.

Definition 9 (Quantitative Parse Tree $\mathcal{T}_\varphi$)**.** *Given a specification formula φ, the corresponding parse tree, $\mathcal{T}_\varphi$, has leaf nodes that correspond to function predicates ρ_{mu}, while other nodes are* max *(disjunctions) and* min *(conjunctions).*

A parse tree is an equivalent graphical representation of φ. Evaluating the parse tree for a given trajectory ξ gives us quantitative satisfaction $\rho_\varphi(\xi)$. For example, consider the specification

$$\varphi := (\mu_1 \vee \mu_2) \rightarrow (\mu_3 \vee \mu_4) = (\neg\mu_1 \wedge \neg\mu_2) \vee (\mu_3 \vee \mu_4), \tag{6.6}$$

where the second equality follows from De-Morgan's law. We can obtain an equivalent function $\varphi(e)$ with (6.2),

$$\begin{aligned}\rho_\varphi(e) = \max\big(&\min(-\rho_{\mu_1}(e),\ -\rho_{\mu_2}(e)),\\ &\max(\rho_{\mu_3}(e),\ \rho_{\mu_4}(e))\big).\end{aligned} \tag{6.7}$$

The parse tree, $\mathcal{T}_\varphi$, for φ in (6.7) is shown in Fig 6.2. We can use the parse tree to decompose any complex specification into min and max functions of the individual predicates; that is, $\rho\varphi(e) = \mathcal{T}_\varphi(\rho_{\mu_1}(e), \ldots, \rho_{\mu_q}(e))$.

We now model each predicate $\rho_{\mu_i}(e)$ in the parse tree $\mathcal{T}_\varphi$ of φ with a GP and combine them with the parse tree to obtain confidence intervals on the overall specification $\rho_\varphi(e)$ for BO. This is done in the learner $\mathcal{L}_f$. GP-LCB as expressed in (6.4) can be used to search for the minimum for a single GP. A key insight to extending (6.4) across multiple GPs, is that the minimum of (6.5) is, with high probability, lower bounded by the lower-confidence interval of one of the GPs used to model the predicates of φ. This is because, the max and min operators do not change the value of the predicates, but only make a choice between them. As a consequence, we can model the smooth parts of φ, i.e., the predicates, using GPs and then consider the non-smoothness through the parse tree.

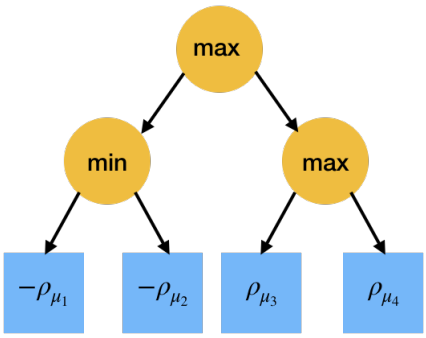


Figure 6.2. Equivalent parse tree $\mathcal{T}_\varphi$ for ρ_φ in (6.6) to the function (6.7). We replace the predicates ρ_{μ_i} with their corresponding pessimistic GP predictions to obtain a lower bound on $\rho_\varphi(e)$.

For each predicate ρ_{μ_i} in the parse tree $\mathcal{T}_\varphi$ of φ, we construct a lower confidence bound $l_i = m^i_{n-1}(e) - \beta_n^{1/2}\sigma^i_{n-1}(e)$, where m^i, σ^i are the mean and standard deviation of the GP corresponding to ρ_{μ_i}. From this, we can construct a lower-confidence interval on φ as $\mathcal{T}_\varphi(l_1(e), \ldots, l_q(e))$, where we replace the ith leaf node μ_i of the parse tree with the pessimistic prediction l_i of the corresponding GP. Similar to (6.4), the corresponding acquisition function for BO uses this lower bound to select the next evaluation point,

$$e_n = \underset{e \in \mathcal{E}}{\operatorname{argmin}}\ \mathcal{T}_\varphi(l_1(e), \ldots, l_q(e)). \tag{6.8}$$

Intuitively, the next environment selected to simulate is the one that minimizes the worst-case predictions on φ. Effectively, we propagate the confidence intervals associated with the GP for each predicates through the parse tree $\mathcal{T}_\varphi$ in order to obtain predictions about φ directly. Note, that (6.8) does not return an environment sample that minimizes the satisfaction of all the predicates, it only minimizes the lower bound on φ. This is done in the oracle $\mathcal{O}_f$.

Algorithm (5) describes our active testing procedure. The algorithm proceeds by first computing the parse tree $\mathcal{T}_\varphi$ from the specification, φ. At each iteration n of BO, we select new environment parameters e_n according to (6.8). We then simulate the system with parameters e_n and evaluate each predicate ρ_{μ_i} on the simulated trajectories. Lastly, we update each GP with the corresponding measurement of ρ_{μ_i}. The algorithm either returns a counterexample that minimizes (6.5); or when $\mathcal{T}_\varphi(l_1(e), \ldots, l_q(e))$ is greater then zero, and we can conclude that the system has been verified.

6.5.1 Theoretical Results

We can transfer theoretical convergence results for GP-LCB [142] to the setting of Alg 5. To do this, we need to make structural assumptions about the predicates. In particular, we assume that they have bounded norm in the Reproducing Kernel Hilbert Space (RKHS, [143]) that corresponds to the GP's kernel. These are well-behaved functions of the form $\rho_{\mu_i}(e) = \sum_{j=0} \alpha_j k_i(e, e_j)$ with representer points e_j and weights α that decay sufficiently quickly.

Algorithm 5 Active Testing with Bayesian Optimization

procedure ActiveTesting($\varphi, \mathcal{E}, \beta, GPs$)
  Build parse tree $\mathcal{T}_\varphi$ based on specification φ ▷ In $\mathcal{L}_f$
  for $n = 0, \ldots$ **do** ▷ Until budget or convergence
    $l_i(e) = \rho_{\mu_i}(e) - \beta_n^{1/2}\sigma_i(e)$, $i = 1, \ldots, q$ ▷ In $\mathcal{O}_f$
    $e_n = \text{argmin}_{e\in\mathcal{E}}\, \mathcal{T}_\varphi(l_1(e), \ldots, l_q(e))$ ▷ In $\mathcal{O}_f$
    Update each GP model of the predicates with measurements $(e_n,\ \rho_{\mu_i}(e_n))$. ▷ In $\mathcal{L}_f$
  return $\min_i\ \rho_\varphi(e_i)$, the worst result.

We leverage theoretical results from [27] and [16] that allow us to build reliable confidence intervals using the GP models. We have the following result.

Theorem 8. *Assume that each predicate ρ_{μ_i} has RKHS norm bounded by B_i and that the measurement noise is σ-sub-Gaussian. Select $\delta \in (0, 1)$, e_n according to* (6.8)*, and let $\beta_n^{1/2} = \sum_i B_i + 4\sigma\sqrt{1 + \ln(1/\delta) + \sum_i I(\mathbf{y}_{n-1}^i; \rho_{\mu_i})}$. If $\mathcal{T}_\varphi(l_1(e_n), \ldots, l_q(e_n)) > 0$, then with probability at least $1 - \delta$ we have that $\min_{e\in\mathcal{E}} \rho_\varphi(e) > 0$ and the system has been verified against all environments in $\mathcal{E}$.*

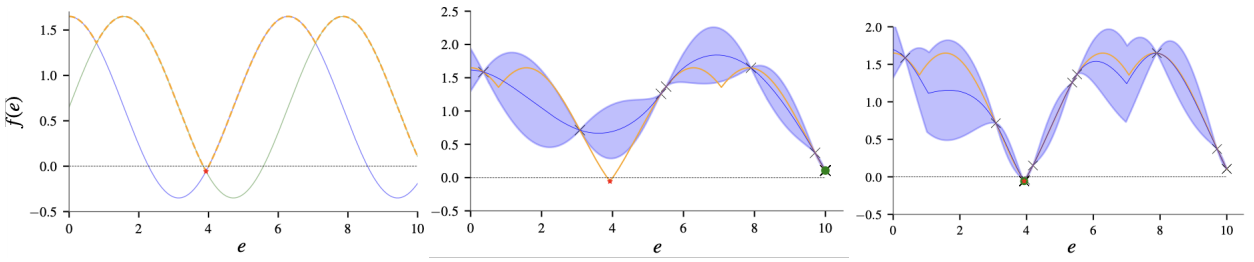
Here $I(\mathbf{y}_{n-1}^i; \rho_{\mu_i})$ is the mutual information between $\mathbf{y}_{n-1}^i$, the $n-1$ noisy measurements of ρ_{μ_i}, and the GP prior of ρ_{μ_i}. This function was shown to be sublinear in n for many commonly-used kernels in [142], see Section 6.8 for more details. Theorem 8 states that we can verify the system against adversarial examples with high probability, by checking whether the worst-case lower-confidence bound is greater than zero. We provide additional theoretical results about the existence of a finite n such that the system can be verified up to ϵ accuracy in Section 6.8.

6.6 Evaluation

In this section, we evaluate our method on several challenging test cases. A Python implementation of our framework and the following experiments can be found at
`https://github.com/shromonag/adversarial_testing.git`

In order to use Alg 5, we have to solve the optimization problem (6.8). In practice, different optimization techniques have been proposed to find the global minimum of the function. One popular algorithm is DIRECT [54], a gradient-free optimization method. An alternative is to use gradient-based methods together with random-restarts. Particularly, we sample a large number of potential environment scenarios at random from $\mathcal{E}$, and run seperate optimization routines to minimize (6.8) from these.

Another challenge is that the dimensionality of the optimization problem can often be very large. However, methods that allow for more efficient computation do exist. These methods reduce the effective size of the input space and thereby make the optimization



(a) True functions.

(b) Modeling the non-smooth ρ_φ directly.

(c) Modeling and combining smooth predicates ρ_μ.

Figure 6.3. The dashed orange line in Fig 6.3a represents the true, non-smooth optimization function in (6.9) while the green and blue line represent $\sin(e)$ and $\cos(e)$ respectively. Modeling this function directly as a GP leads to model errors Fig Fig. 6.3b, where the 95% confidence interval of the GP (blue shaded) with mean estimate (in blue line) does not capture the true function $\rho_\varphi(e)$ in orange. In fact, the minimum (red star) is not contained within the shaded region, causing the optimization to diverge. BO converges to the green dot, where $\rho_\varphi(e) > 0$ which is not a counterexample. Instead, modeling the two predicates individually and combining them with the parse tree, leads to the model in Fig Fig. 6.3c. Here, the true function is completely captured in the confidence interval. As a consequence, BO converges to the global minimum (the red star and green dot converge).

problem more tractable. One possibility is to use random embedding to reduce the input dimension as done in Random Embedding Bayesian Optimization (REMBO [156]). We can then model the GP in this smaller input dimension and carry out BO in the lower dimension input space.

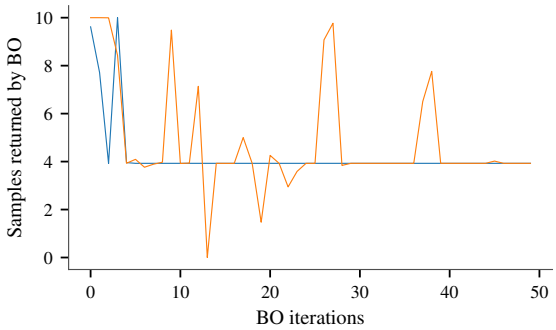
6.6.1 Modeling Smooth vs Non-Smooth Functions

In the following, we show the effectiveness of modeling smooth functions by GPs and considering the non-smooth operations in the BO search as opposed to modeling the non-smooth function by a single GP.

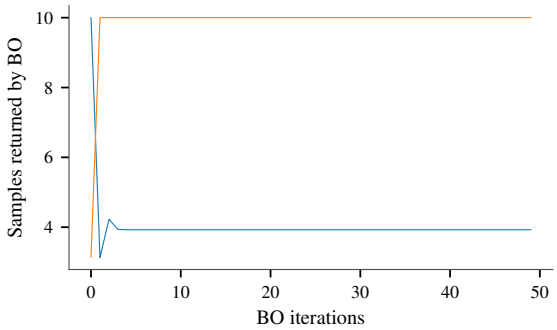
Consider the following, illustrative optimization problem,

$$e^* = \underset{e \in (0,10)}{\operatorname{argmin}} \max(\sin(e) + 0.65, \cos(e) + 0.65) \tag{6.9}$$

We consider two modeling scenarios, one where we model $\max(\sin(e), \cos(e))$ as a single GP, and another where we model $\sin(w)$ by one GP and $\cos(e)$ by another. We initialize the GP models for $\sin(e)$, $\cos(e)$ and $\max(\sin(e), \cos(e))$ with 5 samples chosen in random. We then use BO to find e^*. We were able to model smooth functions like $\sin(e)$ and $\cos(e)$ with GPs, even with fewer samples. At each iteration of BO, we computed the next sample by solving for the $e \in (0, 10)$ which minimized the maximum across the two GPs. This quickly stabilizes to the true w^* Fig (6.3c). When we model $\max(\sin(e), \cos(e))$ using a GP, in Fig 6.3b, the initial 5 samples were not able to model it well. In fact, the original function in orange is not contained within the uncertainty bounds of the GP. Hence, in each iteration of BO, where we chose $e \in (0, 10)$ which minimized this function, we were never



(a) Modeling as separate GPs take around 5 iterations to stabilize to e^* (in blue), while modeling as a single GP takes around 45 iteration to stabilize to e^* (in orange)



(b) Modeling as separate GPs take around 5 iterations to stabilize to e^* (in blue), while modeling as a single GP does not stabilize to e^* (in orange)

Figure 6.4. The orange and blue lines in Fig 6.4a and Fig Fig. 6.4b show the evolution of samples returned over the BO iterations when (6.9) is modeled as a single GP and multiple GPs respectively for two different initialization. We see the that when modeling as a single GP, it takes longer to stabilize to e^* and in some cases (Fig 6.4b) does not stabilize to e^*.

able to converge e^*. It is not surprising to see that, given these models, BO does not always converge when we model non-smooth functions such as in (6.9).

To support our claim, we repeat this experiment 15 times with different initial samples. In each experiment we run BO for 50 iterations. When modeling $\sin(e)$ and $\cos(e)$ as separate GPs, BO stabilized to e^* in about 5 iterations in all 15 experiments. However, when modeling $\max(\sin(e), \cos(e))$ as a single GP, it takes over 35 iterations to converge and in 5 out of the 15 cases, it did not converge to e^*. We show these two different behaviors in Fig 6.4.

6.6.2 Collision Avoidance with High Dimensional Uncertainty

Consider an autonomous car that travels on a straight road with a obstacle at x_{obs}. We require that the car can come to a stop before colliding with an obstacle. The car has two states; location, x, and velocity, v; and one control input acceleration; a. The dynamics of the car is given by,

$$\dot{x} = v, \quad \dot{v} = a. \tag{6.10}$$

Our safety specification for collision avoidance is given by, $\varphi = \min(x_{obs} - x(t))$, i.e., the minimum distance between the position of the car and the obstacle over a horizon of length 100. We assume that the car does not know where the obstacle is *a priori*, but receives locations of the obstacle through a sensor at each time instant, $x_s(t)$. The controller is a simple linear state feedback control, K, such that at time t, $a(t) = K \cdot \big[x(t) - x_s(t),\, v(t)\big]^{\mathrm{T}}$.

We assume that the car initially starts at location $x_{init} = 0$, with a velocity $v_{init} = 3\,\mathrm{m/s}$. Let the obstacle be at $x_{obs} = 5$, which is not known by the car. Instead, it receives sensor readings for the location of the obstacle such that $x_s = [4.5, 5.5]$. If φ is negative, then

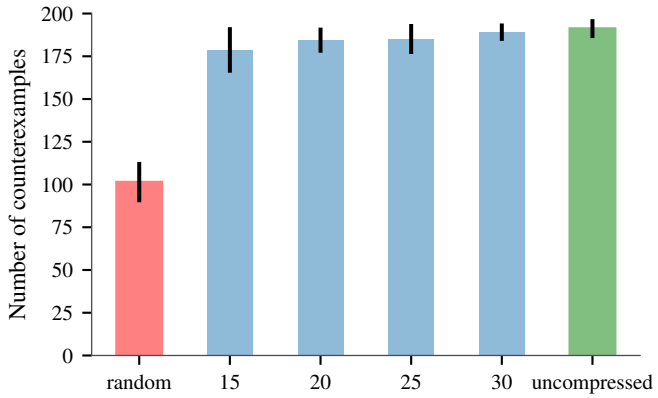


Figure 6.5. The red, blue and green bars shows the average number of counterexamples found using random sampling; applying BO on the reduced input space and original input space respectively for the example in Sec 6.6.2. The black lines show the standard deviation across the experiments.

$x(t) > x_{obs}$ for some t which signifies collision. Moreover, we constrain the acceleration to lie in $a \in [-3, 3]$.

The domain of our uncertainty is $\mathcal{E} = [4.5, 5.5]^{100}$, i.e., the sensor readings x_s over the horizon $H = 100$. We compare across three experimental setups, first, we model the GP in the original space of $\mathcal{E}$ i.e., with 100 inputs; second, we model the GP in a lower dimension input space as described in the preamble of this section; and third, we randomly sample inputs and test them. We run BO for 250 iterations on the GPs, and consider 250 random samples for the random testing. We repeat this experiment 10 times and show our results in Fig Fig. 6.5. The green and blue bar in Fig 6.5 show the average number of counterexamples returned running BO on the GP defined over the original input space and in the low dimension input space. In general, active testing in the high-dimensional input space gives the best results, which deteriorates with an increase in compression of the input space. Random testing, shown in red performs the worst. This is not surprising as, (1) 250 samples is not sufficient to cover an input space of 100 dimensions uniformly; and (2) the samples are all independent of each other. Moreover, in the uncompressed input case, the specification evaluated at the worst counterexample, $\rho_\varphi(e^*)$, has a mean and standard deviation of -0.0138 and 0.004 as compared to -0.0067 and 0.0011 for random sampling.

6.6.3 OpenAI Gym Environments

We interfaced our tool with environments from OpenAI gym [17] to test controllers from Open AI baselines [119]. For brevity, we refer the details of the environments to [120]. In both case studies, we introduce uncertainty around the parameters the controller has been trained for. The rationale behind this is that the parameters in a simulator are an estimate of the true values. This ensures that counterexamples found, can indeed occur in the real system.

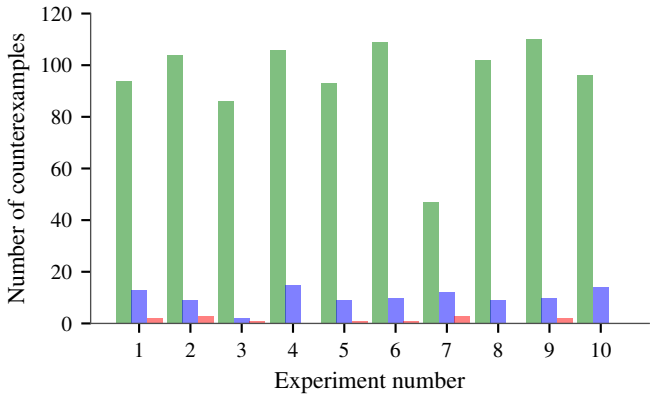


Figure 6.6. The green, blue and red bars show the number of counter examples generated when modeling $\rho_{\mu_1}, \rho_{\mu_2}$ as separate GPs; modeling ρ_φ as a single GP and random testing respectively for the reacher example (Sec 6.6.3.1). Our modeling paradigm, finds more counterexamples compared to the other two methods.

6.6.3.1 Reacher

In the reacher environment, we have a 2D robot trying to reach a target. For this environment we have six sources of uncertainty: two for the goal position, $(x_{goal}, y_{goal}) \in [-0.2, 0.2]^2$, two for state perturbations $(\delta_x, \delta_y) \in [-0.1, 0.1]^2$ and two for velocity perturbations $(\delta_{vx}, \delta_{vy}) \in [-0.005, 0.005]^2$. The state of the reacher is tuple with the current location, $\mathbf{x} = (x, y)$, velocity $\mathbf{v} = (v_x, v_y)$, and rotation, θ. A trajectory of the system, $\xi_{\mathcal{S}}$, is a sequence of states over time, i.e., $\xi_{\mathcal{S}} = (\mathbf{x}(t), \mathbf{v}(t), \theta(t)), t = 0, 1, 2, \ldots$. Our uncertainty space is, $\mathcal{E} = [-0.2, 0.2]^2 \times [-0.1, 0.1]^2 \times [-0.005, 0.005]^2$. Given an instance of $e \in \mathcal{E}$, the trajectory, $\xi_{\mathcal{S}}$, of the system is uniquely defined.

We trained a controller using the Proximal Policy Optimization (PPO) [132] implementation available at Open AI baselines. We determine a trajectory to be safe if either the reacher reaches the goal, or if it does not rotate unnecessarily. This can be captured as $\varphi = \mu_1 \vee \mu_2$, where, $\mu_1(w)$ is the minimum distance between the trajectory and the goal position, and μ_2 is total rotation accumulated over the trajectory; and its continuous variant, $\rho_\varphi = \max(\rho_{\mu_1}, \rho_{\mu_2})$.

Using our modeling approach, we model this using two GPs, one for ρ_{μ_1} and another for ρ_{μ_2}. We compare this to modeling ρ_φ as a single GP and random sampling. We run 200 BO iterations and consider 200 random samples for random testing. We repeat this experiment 10 times. In Fig Fig. 6.6, we plot the number of counterexamples found by each of the three methods over 10 runs of the experiment. Modeling the predicates by separate GPs and applying BO across them (shown in green) consistently performs better than applying BO on a single GP modeling ρ_φ (shown in blue) and random testing (shown in red). We see the that random testing performs very poorly, in some cases (experiment runs $4, 8, 10$) finds no counterexamples.

By modeling the predicates separately, the specification evaluated at the worst coun-

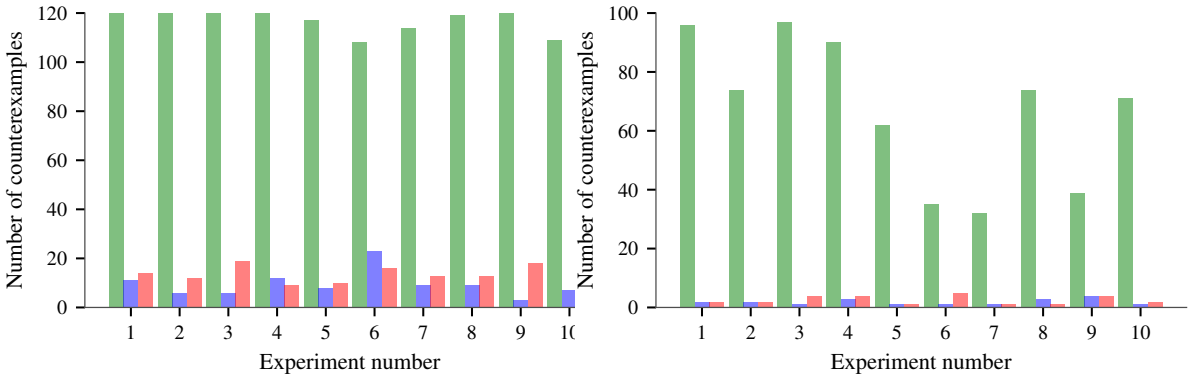
terexample, $\rho_\varphi(e^*)$, has a mean and standard deviation of -0.1283 and 0.0006 as compared to -0.1212 and 0.0042 when considering a single GP. This suggests, that using our modeling paradigm BO converges (since the standard deviation is small) to a more falsifying counterexample (since the mean is smaller).

6.6.3.2 Mountain Car Environment

The mountain car environment in OpenAI gym, is a car on a one-dimensional track, positioned between two mountains. The goal is to drive the car up the mountain on the right. The environment comes with one source of uncertainty, the initial state $x_{init} \in [-0.6, -0.4]$. We introduced four other sources of uncertainty, for the initial velocity, $v_{init} \in [-0.025, 0.025]$; goal location, $x_{goal} \in [0.4, 0.6]$; maximum speed, $v_{max} \in [0.55, 0.75]$ and maximum power magnitude, $p_{max} \in [0.0005, 0.0025]$. The state of the mountain car is a tuple with the current location, x, and velocity, v. A trajectory of the system, $\xi_{\mathcal{S}}$, is a sequence of states over time, i.e., $\xi_{\mathcal{S}} = (x(t), v(t)), t = 0, 1, 2, \ldots$. Our uncertainty space is given by, $\mathcal{E} = [-0.6, -0.4] \times [-0.025, 0.025] \times [0.4, 0.6] \times [0.55, 0.75] \times [0.0005, 0.0025]$. Given an instance of $e \in \mathcal{E}$, the trajectory, $\xi_{\mathcal{S}}$, of the system is uniquely defined.

We trained two controllers one using PPO and another using an actor critic method (DDPG) for continuous Deep Q-learning [100]. We determine a trajectory to be safe, if it reaches the goal quickly or if does not deviate too much from its initial location and always maintains its velocity in some bound. Our safety specification can be written as $\varphi = \mu_1 \vee (\mu_2 \wedge \mu_3)$, where, $\mu_1(w)$ is time taken to reach the goal, μ_2 is the deviation from the initial location and μ_3 is the deviation from the velocity bound; and its continuous variant of $\rho_\varphi = \max(\rho_{\mu_1}, \min(\rho_{\mu_2}, \rho_{\mu_3}))$. We model ρ_φ, by modeling each predicate, ρ_μ, by a GP. We compare this to modeling ρ_φ with a single GP and random sampling. We run 200 BO iterations for the GPs and consider 200 random samples for random testing. We repeat this experiment 10 times. We show our results in Fig 6.7, where we plot the number of counterexamples found by each of the three methods over 10 runs of the experiment for each controller. Fig 6.7 demonstrates the strength of our approach. The number of counterexamples found by our method (in green bar) is much higher compared to random sampling (in red) and modeling ρ_φ as a single GP (in blue). In Fig 6.7a the blue bars are smaller than even the ones in red, suggesting random sampling performs better than applying BO on the GP modeling ρ_φ. The is because the GP is not able to model ρ_φ, and is so far away from the true model, that the sample returned by the BO is worse than if were to sample randomly.

This is further highlighted by the value of the specification at worst counterexample, $\rho_\varphi(e^*)$. The mean and standard deviation for $\rho_\varphi(e^*)$ over the 10 experiment runs is -0.5435 and 0.028 for our method, -0.3902 and 0.0621 when $\rho_\varphi i$ is modeled as a single GP; and -0.04379 and 0.0596 for random sampling. A similar but less drastic result holds in the case of the controller trained with DDPG.



(a) Controller trained with PPO (b) Controller trained with DDPG

Figure 6.7. The green, blue and red bars show the number of counter examples generated when modeling $\rho_{\mu_1}, \rho_{\mu_2}, \rho_{\mu_3}$ as separate GPs, modeling ρ_{φ} as a GP and random testing respectively for the mountain car example (Sec 6.6.3.2). While our modeling paradigm, finds orders of magnitude more counterexample compared to the other two methods, we notice that modeling ρ_{φ} as a single GP performs much worse than random sampling for the controller trained with PPO Fig 6.7a and comparable for the controller trained with DDPG Fig 6.7b.

6.7 Conclusion

In this chapter we addressed the problem of verifying controllers against uncertainties introduced in the design process. These uncertainties include (but not limited to) parts of the system or environment not modeled. To achieve this, we developed an active sampling based simulation based falsification technique based of Gaussian Process and Bayesian Optimization which can can provide strong verification guarantees. The effectiveness of our approach is shown by empirical evaluation of OpenAI gym environments like *cartpole* and *mountain car* with controllers trained using deep reinforcement learning.

6.8 Proofs

In this section, we prove the convergence of our algorithm Alg 5 under specified regularity assumptions on the underlying predicates. Consider the specification

$$\rho_{\varphi}(e) = \mathcal{T}_{\varphi}(\rho_{\mu_1}(e), \ldots, \rho\mu_q(e)), \tag{6.11}$$

where q represents the number of predicates. Let the domain of the predicate indices be represented by, $\mathcal{I} = \{1, \ldots, q\}$. The convergence proofs for classical Bayesian optimization in [142, 27] proceed by building reliable confidence intervals for the underlying function and then showing, that these confidence intervals concentrate quickly enough at the location of the optimum under the proposed evaluation strategy. For ease of exposition, we assume that measurements of each predicate ρ_{μ_i} are corrupted by the same measurement noise.

To leverage these proofs, we need to account for the fact that our GP model is composed of several individual predicates and that we obtain one measurement for each predicates at every iteration of the algorithm.

We start by defining a composite function $f : \mathcal{E} \times \mathcal{I} \to \mathbb{R}$, which returns the function values for the individual predicates indexed by i.

$$f(e, i) = \rho_{\mu_i}(e) \tag{6.12}$$

The function $f(\cdot, \cdot)$ is a single output function, which can be modeled with a single GP with a scalar output over the extended input space, $\mathcal{E} \times \mathcal{I}$. For example, if we assume that the predicates are independent of each other, the kernel function for f would look like,

$$k((e, i), (e', i')) = \begin{cases} k_i(e, e') \text{ if } i = i' \\ 0 \text{ otherwise} \end{cases}, \tag{6.13}$$

where k_i is the kernel function corresponding to the GP for the $i-$th predicate, μ_i. It is straightforward to include correlations between functions in this formulation too.

This reformulation allows us to build reliable confidence intervals on the underlying predicates, given regularity assumptions. In particular, we make the assumption that the function f has bounded norm in the Reproducing Kernel Hilbert Space (RKHS, [143]) corresponding to the same kernel $k(\cdot, \cdot)$ that is used for the GP on f.

Remark 4. *Note, that this model is more general then the case where we assume that each predicate, ρ_{μ_i}, individually has bounded RKHS norm B_i. In this case, the function, $f(e, i)$ has RKHS norm with respect to the kernel in* (6.13) *bounded by $B = \sum_i^q B_i$.*

Lemma 2. *Assume that f has RKHS norm bounded by B and the measurements are corrupted by σ-sub-Gaussian noise. If $\beta_{n \cdot q}^{1/2} = B + 4\sigma\sqrt{I(y_{q \cdot (n-1)}; f) + 1 + \ln 1/\delta}$, then the following holds for all environment scenarios, $e \in \mathcal{E}$, predicate indices, $i \in \mathcal{I}$, and iterations $n \geq 1$ jointly with probability at least $1 - \delta$,*

$$|f(e, i) - m_{q \cdot (n-1)}^i(e, i)| \leq \beta_{q \cdot n}^{1/2} \sigma_{q \cdot (n-1)}^i(e, i) \tag{6.14}$$

Proof: This follows directly from [16], which extends the results from [27] and Lemma 5.1 from [142] to the case of multiple measurements. □

The scaling factor for the confidence intervals, $\beta_{n \cdot q}$, depends on the mutual information $I(\mathbf{y}_{q \cdot (n-1)}; f)$ between the GP model of f and the q measurements of the individual predicates that we have obtained for each time step so far. It can easily be computed as

$$\begin{aligned} I(\mathbf{y}_{q \cdot (n-1)}; f) &= \log(1 + \frac{1}{\sigma^2} \mathbf{K}_{q \cdot (n-1)}), \\ &= \sum_{j=1}^{n-1} \sum_{i=1}^{q} \log(1 + \sigma_{j \cdot q}^2(e_j, i)/\sigma^2), \end{aligned} \tag{6.15}$$

where $\mathbf{K}_{q\cdot(n-1)}$ is the kernel matrix of the single GP over the extended parameter space and the inner sum in the second equation indicates the fact that we obtain q measurements at every iteration.

Based on these individual confidence intervals on ρ_μ, we can construct confidence intervals on ρ_φ. In particular, let

$$\begin{aligned} l_i(e) &= m_{q\cdot(n-1)}(e,i) - \beta_{q\cdot n}^{1/2}\sigma_{q\cdot(n-1)}(e,i) \\ u_i(e) &= m_{q\cdot(n-1)}(e,i) + \beta_{q\cdot n}^{1/2}\sigma_{q\cdot(n-1)}(e,i) \end{aligned} \tag{6.16}$$

be the lower and upper confidence intervals on each predicate. From this, we construct reliable confidence intervals on $\rho_\varphi(e)$ as follows:

Lemma 3. *Under the assumptions of Lemma 2. Let $\mathcal{T}_\varphi$ be the parse tree corresponding to ρ_φ. Then the following holds for all environment scenarios, $e \in \mathcal{E}$ and iterations $n \geq 1$ jointly with probability at least $1-\delta$,*

$$\mathcal{T}_\varphi(l_1(e),\ldots,l_q(e)) \leq \rho_\varphi(e) \leq \mathcal{T}_\varphi(u_1(e),\ldots,u_q(e)) \tag{6.17}$$

Proof: This is a direct consequence of Lemma 2 and the properties of the min and max operators. □

We are now able to prove the main theorem as a direct consequence of Lemma 3.

Theorem 9. *Assume that each predicate ρ_{μ_i} has RKHS norm bounded by B_i and that the measurement noise is σ-sub-Gaussian. Select $\delta \in (0,1)$, e_n according to (6.8), and let $\beta_n^{1/2} = \sum_i B_i + 4\sigma\sqrt{1+\ln(1/\delta)+\sum_i I(\mathbf{y}_{n-1}^i;\rho_{\mu_i})}$. If $\mathcal{T}_\varphi(l_1(e_n),\ldots,l_q(e_n)) > 0$, then with probability at least $1-\delta$ we have that $\min_{e\in\mathcal{E}}\rho_\varphi(e) > 0$ and the system has been verified against all environments in $\mathcal{E}$.*

Proof: For independent variables the mutual information decomposes additively and following Note 4 this is a direct consequence of Lemma 3, since $\mathcal{T}_\varphi(l_1(e),\ldots,l_q(e)) \leq \rho_\varphi(e)$ holds for all $e \in \mathcal{E}$ with probability at least $1-\delta$. □

6.8.1 Convergence proof

In the following, we prove a stronger result about convergence of our algorithm.

The key quantity in the behavior of the algorithm is the mutual information (6.15). Importantly, it was shown in [16] that it can be upper bounded by the worst-case mutual information, the information capacity, which in turn was shown to be sublinear by [142]. In particular, let $\mathbf{f}_\mathbb{W}$ denote the noisy measurements obtained when evaluating the function f at

points in $\mathbb{W}$. The mutual information obtained by the algorithm can be bounded according to

$$\begin{aligned} I(\mathbf{f}_{\mathcal{E}_n \times \mathcal{I}}; f) &\leq \max_{\bar{\mathcal{E}} \subset \mathcal{E}, |\bar{\mathcal{E}}| \leq n} I(\mathbf{f}_{\bar{\mathcal{E}} \times \mathcal{I}}; f); \\ &\leq \max_{\mathcal{D} \subset \mathcal{E} \times \mathcal{I}, |\mathcal{D}| \leq n \cdot q} I(\mathbf{f}_{\mathcal{D}}; f); \\ &= \gamma_{q \cdot n}, \end{aligned} \tag{6.18}$$

where γ_n is the worst-case mutual information that we can obtain from n measurements,

$$\gamma_n = \max_{\mathcal{D} \subset \mathcal{E} \times \mathcal{I}, |\mathcal{D}| = n} I(\mathbf{f}_{\mathcal{D}}; f). \tag{6.19}$$

This quantity was shown to be sublinear in n for many commonly-used kernels in [142].

A key quantity to show convergence of the algorithm is the instantaneous regret,

$$r_n = \min_{e \in \mathcal{E}} \rho_\varphi(e) - \rho_{spec}(e_n), \tag{6.20}$$

the difference between the unknown true minimizer of ρ_φ and the environment parameters e_n that Alg 5 selects at iteration n. If the instantaneous regret is equal to zero, the algorithm has converged.

In the following, we will show that the cumulative regret, $R_n = \sum_{i=1}^n r_i$ is sublinear in n, which implies convergence of Alg 5.

We start by bounding the regret in terms of the confidence intervals on ρ_{μ_i}.

Lemma 4. *Fix $n \geq 1$, if $|f(e,i) - m_{q\cdot(n-1)}(e,i)| \leq \beta_{q\cdot n}^{1/2} \sigma_{q\cdot(n-1)}(e,i)$ for all $e, i \in \mathcal{E} \times \mathcal{I}$, then the regret is bounded by $r_n \leq 2\beta_{q\cdot n}^{1/2} \max_i \sigma_{q\cdot(n-1)}(e,i)$.*

Proof: The proof is analogous to [142, Lemma 5.2]. The maximum standard deviation follows from the properties of the max and min operators in the parse tree $\mathcal{T}_\varphi$. In particular, let $a_1, b_1, a_2, b_2 \in \mathbb{R}$ with $a_1 - b_1 < a_2 - b_2$. Then for all $c_1 \in [-b_1, b_1]$ and $c_2 \in [-b_2, b_2]$ we have that

$$a_1 - b_1 \leq \min(a_1 + c_1, a_2 + c_2) \leq a_1 + b_1. \tag{6.21}$$

The max operator is analogous. Thus, since the parse tree $\mathcal{T}_\varphi$ is composed only of min and max nodes, the regret is bounded by the maximum error over all predicates. The result follows. □

Lemma 5. *Pick $\delta \in (0,1)$ and $\beta_{q\cdot n}$ as shown in Lemma 2, then the following holds with probability at least $1 - \delta$,*

$$\sum_{i=1}^n r_n^2 \leq \beta_{q\cdot n} C_1 q \mathbf{I}(\mathbf{f}_{e_n \times \mathcal{I}}; f) \leq \beta_{q\cdot n} C_1 \gamma_{q\cdot n} \tag{6.22}$$

where r_n is the regret between the true minimizing environment scenario, e^* and the current sample, e_n; and $C_1 = 8/\log 1 + \sigma^{-2}$

Proof: The first inequality follows similar to [142, Lemma 5.4] and the proofs in [16]. In particular, as in [16],

$$r_n^2 \leq 4\beta_{q\cdot n}^2 \max_{i\in\mathcal{I}} \sigma_{q\cdot(n-1)}^2(e_n, i)$$

The second inequality follows from (6.18). □

Lemma 6. *Under the assumptions of Lemma 3, let $\delta \in (0,1)$ and choose e_n according to (6.8). Then, the cumulative regret R_N over N iterations of Alg 5 is bounded with high probability,*

$$Pr\left\{R_n \leq \sqrt{C_1 \beta_N N \gamma_{q\cdot N}} \quad \forall N \geq 1\right\} \geq 1-\delta \tag{6.23}$$

where $C_1 = \frac{8}{\log 1+\sigma^{-2}}$.

Proof: Since, $R_N = \sum_{i=1}^N r_i$, from Cauchy-Schwartz inequality we have, $R_N^2 \leq N \sum_{i=1}^N r_i^2$. The rest follows from Lemma 5. □

We introduce some notation, let

$$\hat{e}_n = \text{argmin}_{e\in\{e_1,\ldots,e_n\}} \rho_\varphi(e) \tag{6.24}$$

be the minimizing environment scenario sampled by BO in n iterations and let

$$e^* = \underset{e\in\mathcal{E}}{\text{argmin}}\, \rho_\varphi(e) \tag{6.25}$$

be the unknown, optimal parameter.

Corollary 1. *For any $\delta \in (0,1)$ and $\epsilon \in \mathbb{R}^+$, there exits a n^*,*

$$\frac{n^*}{\beta_{q\cdot n^*}\gamma_{q\cdot n^*}} = \frac{C_1}{\epsilon^2} \tag{6.26}$$

such that $\forall n \geq n^*$, $\rho_\varphi(e^*) - \rho_\varphi(\hat{e}_n) \leq \epsilon$ holds with probability at least $1-\delta$.

Proof: The cumulative reward over n iterations, $R_n = \sum_{i=1}^n \rho_\varphi(e^*) - \rho_\varphi(e_i)$ where e_i is the i-th BO sample. Defining $\hat{e}_n$ as in (6.24) we have,

$$\begin{aligned} R_n &= \sum_{i=1}^n \rho_\varphi(e^*) - \rho_\varphi(e_i) \\ &\geq \sum_{i=1}^n \rho_\varphi(e^*) - \rho_\varphi(\hat{e}_n) \\ &= n(\rho_\varphi(\hat{e}_n) - \rho_\varphi(e^*)) \end{aligned} \tag{6.27}$$

Combining this result with Lemma 6, we have with probability greater than $1-\delta$ that

$$\begin{aligned}\rho_\varphi(e^*) - \rho_\varphi(\hat{e}_n) &\leq \frac{R_n}{n} \\ &\leq \sqrt{\frac{C_1 \beta_{q \cdot n} \gamma_{q \cdot n}}{n}}\end{aligned} \tag{6.28}$$

To find, n^*, we bound the RHS by ϵ,

$$\sqrt{\frac{C_1 \beta_{q \cdot n^*} \gamma_{q \cdot n^*}}{n^*}} \leq \epsilon \Rightarrow \frac{n^*}{\beta_{q \cdot n^*} \gamma_{q \cdot n^*}} \geq \frac{C_1}{\epsilon^2} \tag{6.29}$$

For $n > n^*$, the minimum $\rho_\varphi(\hat{e}_n) \leq \rho_\varphi(\hat{e}_{n^*}) \implies \rho_\varphi(\hat{e}_n) - \rho_\varphi(e^*) \leq \epsilon$. □

We are now ready to prove our main convergence theorem.

Theorem 10. *Under the assumptions of Lemma 3, choose $\delta \in (0,1)$, $\epsilon \in \mathbb{R}^+$ and define n^* using Corollary 1. If $n \geq n^*$ and $\rho_\varphi(\hat{e}_n) > \epsilon$, then, with probability greater than $1-\delta$, the following statements hold jointly*

- $\rho_\varphi(e^*) > 0$
- *The closed loop system satisfies φ, i.e., the control can safely control the system in all environment scenarios, $\mathcal{E}$*
- *The system has been verified against all environments, $\mathcal{E}$*

Proof: This holds from Lemma 6 and Corollary 1. From Corollary 1, we have $\forall n \geq n^*$, $\Pr(\rho_\varphi(e^*) - \rho_\varphi(\hat{e}_n) \leq \epsilon) > 1-\delta$. If $\exists n \geq n^*$, such that $\rho_\varphi(\hat{e}_n) > \epsilon$, then we have $\Pr(\rho_\varphi(e^*) > 0)1-\delta$, i.e., the minimum value ρ_φ can achieve on the closed loop system is greater than 0. φ is hence, satisfied by our system in all $e \in \mathcal{E}$. □

Chapter 7

Specification-centric Simulation Metric

7.1 Introduction

In Chapter 6, we addressed the problem of analyzing and verifying synthesized controllers against model or environment uncertainties. However, even if one were to verify these controllers in simulation, differences in behavior might occur due to mismatch with the real world. In this chapter, we consider a subset of real world control problems with reach-avoid objectives and propose a framework where controllers verified with a model can be adapted to the real world.

Recent research in robotics and control theory has focused on developing complex autonomous systems, such as robotic manipulators, autonomous vehicles, and surgical robots. Since many of these systems are safety-critical, it is important to design provably-safe controllers while determining environments in which safety can be guaranteed. In this work, we focus on reach-avoid objectives, where the goal is to design a controller to reach a target set of states (referred to as reach set) while avoiding unsafe states (avoid set). Reach-avoid problems are common for autonomous vehicles in the real world; for example, a drone flying in an indoor setting. Here the reach set could be a desired goal position and the avoid set could be the set of the obstacles. In such a setting, it is important to determine the environments in which the drone can safely navigate, as well as the corresponding safe controllers.

Typically, a mathematical model of the system, such as a physics-based first principles model, is used for synthesizing a safe controller in different environments (e.g., [150, 147]). However, when the system dynamics are unknown, synthesizing such a controller becomes challenging. In such cases, it is a common practice to identify a model for the system. This model represents an abstraction of the system behavior. Recently, there has been an increased interest in using machine learning (ML) based tools, such as neural networks and Gaussian processes, for learning abstractions *directly* from the data collected on the system [12, 10, 95]. One of the many verification challenges for ML-based systems [135] is

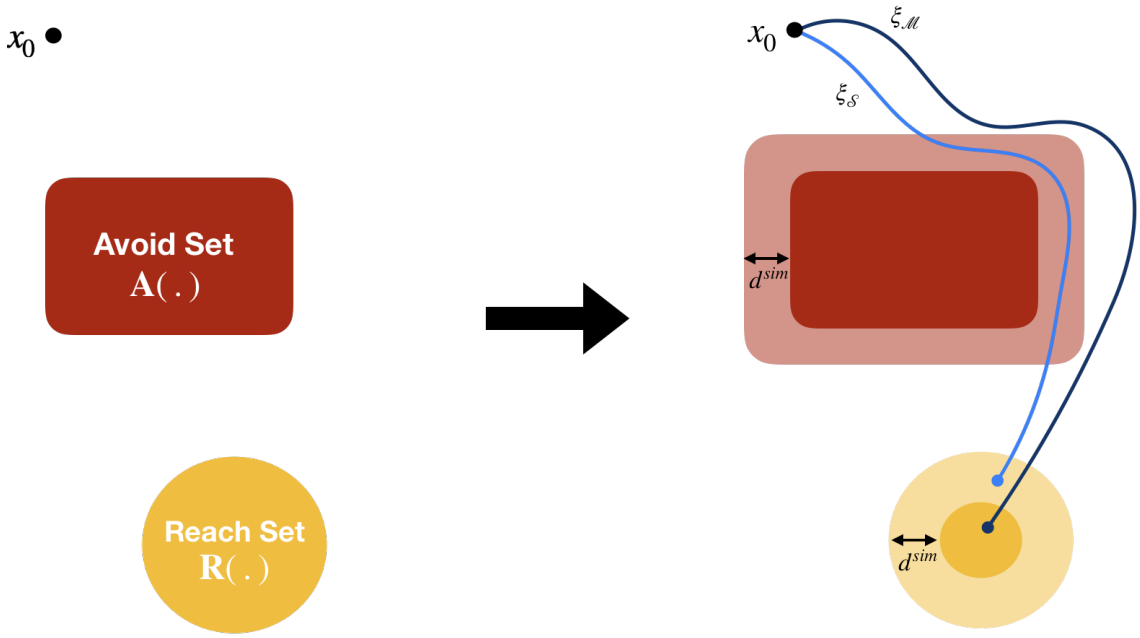


Figure 7.1. The avoid set is expanded and the reach set is contracted with the simulation metric d^{sim}. If the abstraction trajectory ($\xi_{\mathcal{M}}$) stays clear of the expanded avoid set and reaches the contracted reach set, the system trajectory ($\xi_{\mathcal{S}}$) also stays clear of the original avoid set and reaches the original reach set.

that such abstractions cannot be directly used for verification, since it is not clear *a priori* how representative the abstraction is of the actual system. Hence, to use the abstraction to provide guarantees for the system, we need to first quantify the differences between it and the system.

7.1.1 Quantifying system and model mismatches

One approach is to use model identification techniques that provide bounds on the mismatch between the dynamics of the system and its abstraction both in time and frequency domains (see [62, 76, 101] and references therein). This bound is then used to design a provably stabilizing controller for the system. These approaches have largely been limited to linear abstractions and systems, and the focus has been on designing asymptotically stabilizing controllers.

Another way to quantify the difference between a general non-linear system and its abstraction relies on the notion of a (approximate) *simulation metric* [5, 66, 9]. Such a metric measures the maximal distance between the system and the abstraction output trajectories over all finite horizon control sequences. Standard simulation metrics (referred to as SSM here on) have been used for a variety of purposes such as safety verification [67], abstraction design for discrete [94], nonlinear [124], switched [68] systems, piecewise deterministic and labelled Markov processes [35, 145], and stochastic hybrid systems [2, 57, 82, 18], model checking [9, 84], and model reduction [34, 121]. Once computed, the SSM is used to *expand*

the unsafe set (or avoid set) in [2]. For reach-avoid scenarios, we additionally use it to *contract* the reach set as shown in Figure 7.1. If we can synthesize a safe controller that ensures the abstraction trajectory avoids the expanded avoid set and reaches the contracted reach set, then the system trajectory is guaranteed to avoid and reach the original avoid set and reach set respectively. This follows from the property that SSM captures the worst case distance between the trajectories of the system and the abstraction. Consequently, the set of safe environments for the system can be obtained by finding the set of environments for which we can design a safe controller for the abstraction with the modified specification.

Even though powerful in its approach, SSM computes the maximal distance between the system and the abstraction trajectories across all possible controllers. We show in this paper that this is unnecessary and might lead to a conservative bound on the quality of the abstraction for the purposes of controller synthesis. In particular, the larger the distance between the system and the abstraction, the larger the expansion (contraction) of the avoid (reach) set. In many cases, this results in unrealizability wherein there does not exist a safe controller for the abstraction for the modified specification.

In this chapter, we propose SPEC, SPEcification-Centric simulation metric, that overcomes these limitations. SPEC achieves this by computing the distance across

1. only those controllers that can be synthesized by a particular control scheme and that are safe for the abstraction (in the context of the original reach-avoid specification) — these are the only potential safe controllers for the system;

2. only those abstraction and system trajectories for which the system violates the reach-avoid specification, and

3. only between the abstraction trajectory and the reach and the avoid sets.

If the reach-avoid specification is changed using SPEC in a similar fashion as that for SSM, it is guaranteed that if a controller is safe for the abstract model, it remains safe for the system. SPEC can be significantly less conservative than SSM, and can be used to design safe controllers for the system for a broader range of reach-avoid specifications. In fact, we show that, among all uniform distance bounds (i.e., a single distance bound is used to modify the specification in all environments), SPEC provides the largest set of environments such that a safe controller for the abstraction is also safe for the system.

Note that a similar metric has been used earlier [64] to find tight environment assumptions for temporal logic specifications. However, it applies in much more restricted settings since it relies on having simple linear representations of the abstraction which can be expressed as a linear optimization problem.

In general, it is challenging to compute both SSM and SPEC when the dynamics of the system are not available. Several approaches have been proposed in the literature for computing SSM [1, 66, 82]; however, restrictive assumptions on the dynamics of the systems are often required to compute it. More recently, a randomized approach has been proposed to compute SSM [2, 57] for finite-horizon properties that relies on "scenario optimization",

which was first introduced for solving robust convex programs via randomization [21] and then extended to semi-infinite chance-constrained optimization problems [22]. Scenario optimization is a sampling-based method to solve semi-infinite optimization problems, and has been used for system and control design [20, 23]. In this work, we propose a scenario optimization-based computational method for SPEC that has general applicability and is not restricted to a specific class of systems. Indeed, the only assumption is that the system is available as an oracle, with known state and control spaces, which we can simulate to determine the corresponding output trajectory. Given that the distance metric is obtained via randomization and, hence, is a random quantity, we provide probabilistic guarantees on the performance of SPEC. However, this confidence is a design parameter and can be chosen as close to 1 as desired (within a simulation budget). To summarize, this chapter's main contributions are:

- SPEC, a new simulation metric that is less conservative than SSM, and provides the largest set of environments such that a safe controller for the abstraction is also safe for the system;
- a method to compute SPEC that is not restricted to a specific class of systems, and
- a demonstration of the proposed approach on numerical examples and simulations of real-world autonomous systems, such as a quadrotor and an autonomous car.

The results in this chapter are adapted from [63].

7.2 Preliminaries

Let $\mathcal{S}$ be an unknown, discrete-time, potentially non-linear, dynamical system with state space $\mathbb{R}^{n_x}$ and control space $\mathbb{R}^{n_u}$. Let $\mathcal{M}$ be an abstraction of $\mathcal{S}$ with the same state and control spaces as $\mathcal{S}$, whose dynamics are known. We also assume that the bounds between the dynamics of $\mathcal{M}$ and $\mathcal{S}$ are not available beforehand (i.e., we cannot *a priori* quantify how different the two are). $\xi_{\mathcal{S}}(t; x_0, \mathbf{u})$ denotes the trajectory of $\mathcal{S}$ at time t starting from the initial state x_0 and applying the controller $\mathbf{u}$. $\xi_{\mathcal{M}}$ is similarly defined. For ease of notation, we drop $\mathbf{u}$ and x_0 from the trajectory arguments wherever convenient.

We define by $\mathcal{E} := \mathcal{X}_0 \times \mathcal{A} \times \mathcal{R}$ the set of all reach-avoid scenarios (also referred to as *environment scenarios* here on), for which we want to synthesize a controller for $\mathcal{S}$. A reach-avoid scenario $e \in \mathcal{E}$ is a three-tuple, $(x_0, A(\cdot), R(\cdot))$, where $x_0 \in \mathcal{X}_0 \subset \mathbb{R}^{n_x}$ is the initial state of $\mathcal{S}$. $A(\cdot) \in \mathcal{A}, A(\cdot) \subset \mathbb{R}^{n_x}$ and $R(\cdot) \in \mathcal{R}, R(\cdot) \subset \mathbb{R}^{n_x}$ are (potentially time varying) sequences of avoid and reach sets respectively. We leave $\mathcal{A}$ and $\mathcal{R}$ abstract except where necessary. If the sets are not time varying, we can replace $R(\cdot)$ (respectively $A(\cdot)$) by the stationary R (respectively A). Similarly, if there is no avoid or reach set at a particular time, we can represent $A(t) = \emptyset$ and $R(t) = \mathbb{R}^{n_x}$.

For each $e \in \mathcal{E}$, we define a reach-avoid specification, $\varphi(e)$

$$\varphi(e) := \{\xi(\cdot) : \forall t \in \mathcal{T}_H \; \xi(t) \notin A(t) \wedge \xi(t) \in R(t)\}, \tag{7.1}$$

where $\mathcal{T}_H$ denote the time-horizon $\{0, 1, \ldots, H\}$. We say $\xi(\cdot)$ satisfies the specification $\varphi(e)$, denoted $\xi(\cdot) \models \varphi(e)$, if $\xi(\cdot) \in \varphi(e)$.

The reader might observe that our use of $R(\cdot)$ in (7.1) differs somewhat from the intuitive notion of a reach set (depicted, e.g., in Figure 7.1). Specifically, (7.1) defines the reach-avoid specification such that the output trajectory must remain within $R(t)$ at all times t, while the usual notion involves *eventually* reaching a desired set of states. Note, however, that for the purposes of defining $\varphi(e)$, these notions are equivalent if $R(t)$ in (7.1) represents the backwards reachable tube corresponding to the desired reach set: if a state is reachable eventually, then the trajectory stays within the backwards reachable tube at all time points. We henceforth use the $R(t)$ in the latter sense since it simplifies the mathematics in the paper.

Finally, we define $\mathcal{U}_\Pi(e) \subset \mathcal{U}$ to be the space of all permissible controllers for e, and $\mathcal{U}$ to be the space of all finite horizon control sequences over $\mathcal{T}_H$. For example, if we restrict ourselves to linear feedback controllers, $\mathcal{U}_\Pi$ represents the set of all linear feedback controllers that are defined over the time horizon $\mathcal{T}_H$.

7.3 Running Example

We now introduce a very simple example that we will use to illustrate our approach, a 2 state linear system in which the system and the abstraction differ only in one parameter. Although simple, this example illustrates several facets of SPEC. We present more realistic case studies in Section 7.8.

Consider a system $\mathcal{S}$ whose dynamics are given as

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0 \end{bmatrix} u(t). \tag{7.2}$$

We are interested in designing a controller for $\mathcal{S}$ to regulate it from the initial state $x(0) := x_0 = [0, 0]$ to a desired state $x^* = [x_1^*, 0]$ over a time-horizon of 20 steps, i.e, $H = 20$. In particular, we have

$$\mathcal{X}_0 = \{[0, 0]\}, \quad \mathcal{A} = \emptyset, \quad \mathcal{R} = \bigcup_{-4 \le x_1^* \le 4} R(\cdot; x^*),$$

where

$$\begin{aligned} R(t; x^*) =& \mathbb{R}^2, t \in \{0, 1, \ldots, H-1\}, \\ R(H; x^*) =& \{x : \|x - x^*\|_2 < \gamma\}. \end{aligned}$$

We use $\gamma = 0.5$ in our simulations. Thus, each $e \in \mathcal{E}$ consists of a final state x^* (equivalently, a reach set $R(H; x^*)$) to which we want the system to regulate, starting from the origin. Consequently, the system trajectory satisfies the reach-avoid specification in this case if $\xi_{\mathcal{S}}(H; x_0, \mathbf{u}) \in R(H; x^*)$.

For the purpose of this example, we assume that the system dynamics in (7.2) are unknown; only the dynamics of its abstraction $\mathcal{M}$ are known and given as

$$x(t+1) = \begin{bmatrix} x_1(t+1) \\ x_2(t+1) \end{bmatrix} = \begin{bmatrix} 2 & 0 \\ 0 & 0.1 \end{bmatrix} \begin{bmatrix} x_1(t) \\ x_2(t) \end{bmatrix} + \begin{bmatrix} 1 \\ 0.1 \end{bmatrix} u(t). \tag{7.3}$$

In this example, we use the class of linear feedback controllers as $\mathcal{U}_\Pi(e)$, although other control schemes can very well be used. In particular, for any given environmental scenario e, the space of controllers $\mathcal{U}_\Pi(e)$ is given by

$$\mathcal{U}_\Pi(e) = \{LQR(q, x^*) : 0.1 \leq q \leq 100\},$$

where $LQR(q, x^*)$ is a Linear Quadratic Regulator (LQR) designed for the abstraction dynamics in (7.3) to regulate the abstraction trajectory to x^* [1], with the state penalty matrix $Q = qI$ and the control penalty coefficient $R = 1$. Here, $I \in \mathbb{R}^{2\times 2}$ is an identity matrix. Thus, for different values of q we get different controllers, which affect the various characteristics of the resultant trajectory, such as overshoot, undershoot, and final state. $LQR(q)$ for any given q can be obtained by solving the discrete-time Riccati equation [92]. Our goal thus is to use the dynamics in (7.3) to find the set of final states to which $\mathcal{S}$ can be regulated and the corresponding regulator in $\mathcal{U}_\Pi(e)$.

7.4 Problem Formulation

Given the set of reach-avoid scenarios $\mathcal{E}$, the controller scheme $\mathcal{U}_\Pi$, and the abstraction $\mathcal{M}$, our goal is two-fold:

1. to find the environment scenarios for which it is possible to design a controller such that $\xi_{\mathcal{S}}(\cdot)$ satisfies the corresponding reach-avoid specification $\varphi(e)$,
2. to find a corresponding safe controller for each scenario in (1).

Mathematically, we are interested in computing the set $\mathcal{E}_{\mathcal{S}}$

$$\mathcal{E}_{\mathcal{S}} = \{e \in \mathcal{E} \ : \exists \mathbf{u} \in \mathcal{U}_\Pi(e) \ , \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}, \tag{7.4}$$

and the corresponding set of safe controllers $\mathcal{U}_{\mathcal{S}}(e)$ for each $e \in \mathcal{E}_{\mathcal{S}}$

$$\mathcal{U}_{\mathcal{S}}(e) = \{\mathbf{u} \in \mathcal{U}_\Pi(e) \ : \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}. \tag{7.5}$$

[1] That is, we penalize the trajectory deviation to the desired state x^* in the LQR cost function.

When a dynamics model of $\mathcal{S}$ is known, several methods have been studied in literature to compute the sets $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$ for reach-avoid problems [151, 110, 150]. However, since a dynamics model of $\mathcal{S}$ is unknown, the computation of these sets is challenging in general. To overcome this problem, one generally relies on the abstraction $\mathcal{M}$. We make the following assumptions on $\mathcal{S}$ and $\mathcal{M}$:

Assumption 1. *$\mathcal{S}$ is available as an oracle that can be simulated, i.e., we can run an execution (or experiment) on $\mathcal{S}$ and obtain the corresponding system trajectory $\xi_{\mathcal{S}}(\cdot)$.*

Assumption 2. *For any $e \in \mathcal{E}$, we can determine if there exist a controller such that $\xi_{\mathcal{M}} \models \varphi(e)$ and can compute such a controller.*

Assumption 1 states that even though we do not know the dynamics of $\mathcal{S}$, we can run an execution of $\mathcal{S}$. Assumption 2 states that it is possible to verify whether $\mathcal{M}$ satisfies a given specification $\varphi(e)$ or not. Although it is not a straightforward problem, since the dynamics of $\mathcal{M}$ are known, several existing methods can be used for obtaining a safe controller for $\mathcal{M}$.

Under these assumptions, we show that we can convert a verification problem on $\mathcal{S}$ to a verification problem on $\mathcal{M}$. In particular, we compute a distance bound, SPEC, between $\mathcal{S}$ and $\mathcal{M}$ which along with $\mathcal{M}$ allows us to compute a conservative approximation of $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$.

7.5 Solution Approach

We formulate the controller synthesis process for the system $\mathcal{S}$ in the OGIS framework $\mathcal{I}_{SIM} = (\mathcal{L}_{SIM}, \mathcal{O}_{SIM})$. The concept class for the learner is the set of all control policies. The learner $\mathcal{L}_{SIM}$ synthesizes a control policy for the system $\mathcal{S}$ by using the model $\mathcal{M}$ as shown in Figure 7.1. The synthesized controller is sent to the oracle $\mathcal{O}_{SIM}$. The oracle is composed of two oracles. **Verification oracle** attempts to verify the synthesized controller can achieve the reach-avoid task on the system. This oracle handles only verification queries of the type q_{ver} as described in Section 2.1. If the verifier in $\mathcal{O}_{SIM}$ can verify that the synthesized controller is safe for the system, the OGIS framework terminates and outputs the controller. If not, then we compute the simulation metric SPECwhich quantifies the mismatch between the model and the system in the **computation oracle** and return it to the learner. The computation oracle handles computation queries which computes a high confidence estimate of SPEC. The learner then uses the SPECto modify the control objective. If the bound is overly conservative, the model (and hence the system) is safe in smaller set of environment allowing for a smaller set of safe controllers. Moreover, the bound is independent of the control synthesis process (and hence $\mathcal{L}_{SIM}$). In the example presented in Section 7.3, the $\mathcal{L}_{SIM}$ synthesizes LQR controllers in $\mathcal{U}_{\Pi}(e) = \{LQR(q, x^*) : 0.1 \leq q \leq 100\}$.

In Section 7.6 we show that the computation of $SPEC$ in the oracle $\mathcal{O}_{SIM}$ can be implemented in an OGIS framework $\mathcal{I}_{SPEC} = (\mathcal{L}_{SPEC}, \mathcal{O}_{SPEC})$ where the $\mathcal{O}_{SPEC}$ is a black-box simulator of the system $\mathcal{S}$. The design of the learner $\mathcal{L}_{SPEC}$ depends the algorithm used for estimating SPEC.

7.5.1 Computing approximate safe sets using $\mathcal{M}$ and simulation metric

Computing sets $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}$ exactly can be challenging since the dynamics of $\mathcal{S}$ are unknown *a priori*. Generally, we use the abstraction $\mathcal{M}$ as a replacement for $\mathcal{S}$ to synthesize and analyze safe controllers for $\mathcal{S}$. However, to provide guarantees on $\mathcal{S}$ using $\mathcal{M}$, we would need to quantify how different the two are.

We quantify this difference through a distance bound, d, between $\mathcal{S}$ and $\mathcal{M}$. d is used to modify the specification $\varphi(e)$ to a more stringent specification $\varphi(e;d)$ such that if $\xi_{\mathcal{M}}(\cdot) \models \varphi(e;d)$ then $\xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. Thus, the set of safe controllers for $\mathcal{M}$ for $\varphi(e;d)$ can be used as an approximation for $\mathcal{U}_{\mathcal{S}}(e)$. In particular, if we define the sets $\mathcal{U}_{\varphi(e;d)}$ and $\mathcal{E}_{\varphi}(d)$ as

$$\begin{aligned} \mathcal{U}_{\varphi(e;d)} &:= \{\mathbf{u} \in \mathcal{U}_{\Pi}(e) : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e;d)\} \\ \mathcal{E}_{\varphi}(d) &:= \{e \in \mathcal{E} : \mathcal{U}_{\varphi(e,d)} \neq \emptyset\}, \end{aligned} \tag{7.6}$$

then $\mathcal{U}_{\varphi(e;d)}$ and $\mathcal{E}_{\varphi}(d)$ can be used as an approximation of $\mathcal{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\mathcal{S}}$ respectively. Consequently, a verification problem on $\mathcal{S}$ can be converted into a verification problem on $\mathcal{M}$ using the modified specification.

One such distance bound d is given by the simulation metric, SSM, between $\mathcal{M}$ and $\mathcal{S}$ defined as

$$d^{sim} = \max_{e \in \mathcal{E}} \max_{\mathbf{u} \in \mathcal{U}_{\Pi}(e)} \|\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) - \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u})\|_{\infty} \tag{7.7}$$

Here, the ∞-norm is the maximum distance between the trajectories across all timesteps. Typically SSM is computed over the space of all finite horizon controls $\mathcal{U}$ instead of $\mathcal{U}_{\Pi}(e)$ [67]. Since we are interested in a given control scheme, we restrict this computation to $\mathcal{U}_{\Pi}(e)$. In general, d^{sim} is difficult to compute, because it requires searching over (the potentially infinite) space of controllers and environments. An approximate technique to compute d^{sim} was presented for systems whose dynamics were unknown with probabilistic guarantees in [2].

However, if d^{sim} can be computed then it can be used to modify a specification $\varphi(e)$ to $\varphi(e; d^{sim})$ as follows: "expand" the avoid set $A(\cdot)$ to get the augmented avoid set $A(\cdot; d^{sim}) = A(\cdot) \oplus d^{sim}$, and "contract" the reach set $R(\cdot)$ to obtain a conservative reach set $R(\cdot; d^{sim}) = R(\cdot) \ominus d^{sim}$ (see Figure 7.1). Here, $\oplus$ ($\ominus$) is the Minkowski sum(difference)[2]. Consequently, $\varphi(e; d^{sim})$ is the set of trajectories which avoid $A(\cdot; d^{sim})$ and are always contained in $R(\cdot; d^{sim})$,

$$\varphi(e; d^{sim}) := \{\xi(\cdot) : \xi(t) \notin A(t; d^{sim}), \xi(t) \in R(t; d^{sim}) \forall t \in \mathcal{T}_H\}. \tag{7.8}$$

Then it can be shown that any controller that satisfies the specification $\varphi(e; d^{sim})$ for $\mathcal{M}$ also ensures that $\mathcal{S}$ satisfies the specification $\varphi(e)$.

Proposition 2. *For any $e \in \mathcal{E}$ and controller $\mathbf{u} \in \mathcal{U}_{\Pi}(e)$, we have $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{sim})$ implies $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$.*

[2]The Minkowski sum of a set K and a scalar d is the set of all points that are the sum of any point in K and $B(d)$, where $B(d)$ is a disc of radius d around the origin.

Proof. Let us consider for a given environment $e \in \mathcal{E}$ and control $\mathbf{u} \in \mathcal{U}_\Pi(e)$, $\xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{sim})$. We would like to prove that $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$.

From (7.7), we have

$$\|\xi_\mathcal{S}(t) - \xi_\mathcal{M}(t)\| \leq d^{sim} \; \forall t \in \mathcal{T}_H. \tag{7.9}$$

From the definition of specification in (7.8), we have $\xi_\mathcal{M}(\cdot) \models \varphi(e; d^{sim})$ if and only if $\xi_\mathcal{M}(\cdot) \in \varphi(e; d^{sim})$. Therefore, $\xi_\mathcal{M}(t) \notin A(t) \oplus d^{sim}$ and $\xi_\mathcal{M}(t) \in R(t) \ominus d^{sim} \; \forall t \in \mathcal{T}_H$. Since $\xi_\mathcal{M}(t) \notin A(t) \oplus d^{sim}$,

$$\|\xi_\mathcal{M}(t) - a\| > d^{sim} \; , \forall t \in \mathcal{T}_H \; , \forall a \in A(t). \tag{7.10}$$

Combining (7.9) and (7.10) implies that

$$\|\xi_\mathcal{S}(t) - a\| > 0 \; , \forall t \in \mathcal{T}_H \; , \forall a \in A(t). \tag{7.11}$$

Equation (7.11) implies that $\xi_\mathcal{S}(t) \notin A(t)$ for any $t \in \mathcal{T}_H$. Similarly, it can be shown that

$$\|\xi_\mathcal{S}(t) - r\| > 0 \; , \forall t \in \mathcal{T}_H \; , \forall r \in R(t)^c,$$

where $R(t)^c$ denotes the complement of the set $R(t)$. Therefore, $\xi_\mathcal{S}(t) \in R(t) \; \forall t \in \mathcal{T}_H$.

Since $\xi_\mathcal{S}(t) \notin A(t)$ and $\xi_\mathcal{S}(t) \in R(t)$ for all $t \in \mathcal{T}_H$, we have $\xi_\mathcal{S}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$. □

Proposition 2 implies that $\mathcal{E}_\varphi(d^{sim})$ and $\mathcal{U}_{\varphi(e;d^{sim})}$ can be used as approximations of $\mathcal{E}_\mathcal{S}$ and $\mathcal{U}_\mathcal{S}(e)$ respectively. However, the distance bound in (7.7) does not take into account the reach-avoid specification (environment) for which a controller needs to be synthesized. Thus, d^{sim} can be quite conservative. As a result, the modified specification can be so stringent that the set of environments $\mathcal{E}_\varphi(d^{sim})$ for which we can synthesize a provably safe controller for the abstraction (and hence for the system) itself will be very small, resulting in a very conservative approximation of $\mathcal{E}_\mathcal{S}$.

7.5.2 Specification-Centric Simulation Metric (SPEC)

To overcome these limitations, we propose SPEC,

$$d^{SPEC} = \max_{e \in \mathcal{E}} \max_{u \in \mathcal{U}_{\varphi(e)}} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)), \tag{7.12}$$

where

$$\begin{aligned} d(\xi_\mathcal{S}(\cdot), \xi_\mathcal{M}(\cdot)) = \min_{t \in \mathcal{T}_H} (\min\{&h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), A(t)\right), \\ &- h\left(\xi_\mathcal{M}(t; x_0, \mathbf{u}), R(t)\right)\}) \mathbb{1}_{(\xi_\mathcal{S}(\cdot) \not\models \varphi(e))} \end{aligned} \tag{7.13}$$

Here $\mathcal{U}_{\varphi(e)} := \{\mathbf{u} \in \mathcal{U}_\Pi(e) : \xi_\mathcal{M}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\}$ is the set of all controls such that $\mathcal{M}$ satisfies the specification $\varphi(e)$. $\mathbb{1}_l$ represents the indicator function which is 1 if l is true and 0 otherwise, $h(x, K)$ is the signed distance function defined as

$$h(x, K) := \begin{cases} \inf_{k \in K} \|x - k\|, & \text{if } x \notin K \\ -\inf_{k \in K^C} \|x - k\|, & \text{otherwise.} \end{cases}$$

If for any $e \in \mathcal{E}$, $\mathcal{U}_{\varphi(e)}$ is empty, we define the distance function $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot))$ to be zero. Similarly, if there is no $A(\cdot)$ or $R(\cdot)$ at a particular t, the corresponding signed distance function is defined to be ∞. There are four major differences between (7.7) and (7.12):

1. To compute the d^{SPEC} we only consider the feasible set of controllers that can be synthesized by the control policy, $\mathcal{U}_{\varphi(e)} \subseteq \mathcal{U}_{\Pi}(e)$, as all other controllers do not help us in synthesizing a safe controller for $\mathcal{S}$ (as they are not even safe for $\mathcal{M}$).

2. To compute the distance between $\mathcal{S}$ and $\mathcal{M}$, we only consider those trajectories where $\mathcal{S}$ violates the specification. This is because a non-zero distance between the trajectories of $\mathcal{S}$ and $\mathcal{M}$, where the $\xi_{\mathcal{S}} \models \varphi(e)$ does not give us any additional information in synthesizing a safe controller.

3. Within a falsifying $\xi_{\mathcal{S}}$, we compute the minimum distance of the abstraction trajectory from the avoid and reach sets rather than the system trajectory, as that is sufficient to obtain a margin to discard behaviors that are safe for the abstraction but unsafe for the system.

4. Finally, a minimum over time of this distance is sufficient to discard an unsafe trajectory, as the trajectory will be unsafe if it is unsafe at any t.

These considerations ensure that d^{SPEC} is far less conservative compared to d^{sim} and allows us to synthesize a safe controller for the system for a wider set of environments. We first prove that d^{SPEC} can be used to compute an approximation of $\mathcal{E}_{\mathcal{S}}$.

Proposition 3. *If* $\mathcal{U}_{\varphi(e;d^{SPEC})} \subseteq \mathcal{U}_{\varphi(e)}$, *then* $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{SPEC})$ *implies* $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$ $\forall e \in \mathcal{E}, \mathbf{u} \in \mathcal{U}_{\Pi}(e)$.

Proof. We prove the desired result by contradiction. Suppose there exists an environment $e \in \mathcal{E}$ and a controller $\mathbf{u} \in \mathcal{U}_{\varphi(e,d^{SPEC})}$ such that $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{SPEC})$ but $\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \not\models \varphi(e)$.

Since $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{SPEC})$, we have that,

$$\forall t \in \mathcal{T}_H \; \xi_{\mathcal{M}}(t; x_0, \mathbf{u}) \notin A(t; d^{SPEC}) = A(t) \oplus d^{SPEC} \tag{7.14}$$

$$\forall t \in \mathcal{T}_H \; \xi_{\mathcal{M}}(t; x_0, \mathbf{u}) \in R(t; d^{SPEC}) = R(t) \ominus d^{SPEC} \tag{7.15}$$

Since d^{SPEC} is the solution to (7.12), and $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$ (as $\mathcal{U}_{\varphi(e;d^{SPEC})} \subseteq \mathcal{U}_{\varphi(e)}$) is such that $\xi_{\mathcal{M}}(\cdot) \not\models \varphi(e)$, (7.12) and (7.13) imply that,

$$\min_{t \in \mathcal{T}_H} \left(\min\{ h\left(\xi_{\mathcal{M}}(t), A(t)\right), -h\left(\xi_{\mathcal{M}}(t), R(t)\right) \} \right) \leq d^{SPEC}. \tag{7.16}$$

Therefore, $\exists t' \in \mathcal{T}_H$ such that

$$\min\{ h\left(\xi_{\mathcal{M}}(t'), A(t')\right), -h\left(\xi_{\mathcal{M}}(t'), R(t')\right) \} \leq d^{SPEC}, \tag{7.17}$$

which implies that either

1. $h\left(\xi_{\mathcal{M}}(t'), A(t')\right) \leq d^{SPEC}$, or
2. $h\left(\xi_{\mathcal{M}}(t'), R(t')\right) \geq -d^{SPEC}$

If $h\left(\xi_{\mathcal{M}}(t'), A(t')\right) \leq d^{SPEC}$, $\exists a \in A(t')$ such that

$$\|\xi_{\mathcal{M}}(t') - a\| \leq d^{SPEC}.$$

Therefore, $\xi_{\mathcal{M}}(t') \in A(t') \oplus d^{SPEC}$, which contradicts (7.14). Similarly, if $h\left(\xi_{\mathcal{M}}(t'), R(t')\right) \geq -d^{SPEC}$, $\exists r \in R(t')^c$ such that

$$\|\xi_{\mathcal{M}}(t') - r\| \leq d^{SPEC},$$

which implies that $\xi_{\mathcal{M}}(t') \notin R(t') \ominus d^{SPEC}$, which contradicts (7.15).

When $\mathcal{U}_{\varphi(e,d^{SPEC})} \not\subseteq \mathcal{U}_{\varphi(e)}$, for any controller $\mathbf{u} \in \mathcal{U}_{\varphi(e,d^{SPEC})} \backslash \mathcal{U}_{\varphi(e)}$ such that $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; d^{SPEC})$, we can no longer comment on the behavior of the corresponding system trajectory. This is because while computing d^{SPEC}, these controllers were not taken into account. □ Thus, if we define $\mathcal{U}_{\varphi(e;d^{SPEC})}$ and $\mathcal{E}_{\varphi}(d^{SPEC})$ as in (7.6) then they can be used as approximations of $\mathcal{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\mathcal{S}}$ respectively. Proposition 3 requires that the set of controllers that satisfy the modified specification, $\mathcal{U}_{\varphi(e;d^{SPEC})}$, is a subset of the set of the controllers that satisfy the actual specification, $\mathcal{U}_{\varphi(e)}$. When $\mathcal{U}_{\Pi}(e) = \mathcal{U}$, this condition is trivially satisfied as the modified specification is more stringent than the actual specification. Other control schemes, such as the set of linear feedback controllers and feasibility-based optimization schemes also satisfy this condition. In fact, in such cases, the proposed metric, d^{SPEC}, quantifies the tightest (largest) approximation of $\mathcal{E}_{\mathcal{S}}$, i.e., $\nexists d < d^{SPEC}$, such that $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\mathcal{S}}$.

Theorem 11. *Let $\mathcal{U}_{\Pi}$ be such that $\mathcal{U}_{\varphi(e;d_1)} \subseteq \mathcal{U}_{\varphi(e;d_2)}$ whenever $d_1 > d_2$. Let $d \in \mathbb{R}^+$ be any distance bound such that*

$$\forall e \in \mathcal{E}\ , \forall \mathbf{u} \in \mathcal{U}_{\Pi}(e)\ , \xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e). \tag{7.18}$$

Then $\forall e \in \mathcal{E}$,$\mathcal{U}_{\varphi(e;d)} \subseteq \mathcal{U}_{\varphi(e;d^{SPEC})} \subseteq \mathcal{U}_{\mathcal{S}}(e)$. Moreover, $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\varphi}(d^{SPEC}) \subseteq \mathcal{E}_{\mathcal{S}}$. Hence, $\mathcal{E}_{\varphi}(d^{SPEC})$ and $\mathcal{U}_{\varphi(e;d^{SPEC})}$ quantify the tightest (largest) approximations of $\mathcal{E}_{\mathcal{S}}$ and $\mathcal{U}_{\mathcal{S}}(e)$ respectively among all uniform distance bounds d.

The proof of the theorem can be found below after the statement of Corollary 2. Theorem 11 states that d^{SPEC} is the smallest among all (uniform) distance bounds between $\mathcal{M}$ and $\mathcal{S}$, such that a safe controller synthesized on $\mathcal{M}$ is also safe for $\mathcal{S}$. Even though this is a stricter condition than we need for defining $\mathcal{E}_{\mathcal{S}}$, where we care about the existence of at least one safe controller for $\mathcal{S}$, it allows us to use *any* safe controller for $\mathcal{M}$ as a safe controller for $\mathcal{S}$. Formally, $d^{SPEC} \leq d$, for all $d \in \mathbb{R}^+$ such that $\forall e \in \mathcal{E}_{\varphi}(d)\ , \forall \mathbf{u} \in \mathcal{U}_{\varphi(e;d)}\ , \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$.

Intuitively, to compute (7.12), we collect all $\xi_{\mathcal{M}}(\cdot), \xi_{\mathcal{S}}(\cdot)$ pairs (across all $e \in \mathcal{E}$ and $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$) where $\xi_{\mathcal{M}}(\cdot) \models \varphi(e)$ and $\xi_{\mathcal{S}}(\cdot) \not\models \varphi(e)$. We then evaluate (7.13) for each pair and take the maximum to compute d^{SPEC}. By expanding (contracting) every $A(\cdot) \in \mathcal{A}$

($R(\cdot) \in \mathcal{R}$) uniformly by d^{SPEC}, we ensure that none of the $\xi_{\mathcal{M}}(\cdot)$ collected above is feasible once the specification is modified, and hence, $\xi_{\mathcal{S}}(\cdot)$ will never falsify $\varphi(e)$. To ensure this, we prove that d^{SPEC} is the minimum distance by which the avoid sets should be augmented (or the reach sets should be contracted). Thus, d^{SPEC} can also be interpreted as the minimum d by which the specification should be modified to ensure that $\mathcal{U}_{\varphi(e;d)} \subseteq \mathcal{U}_{\mathcal{S}}(e)$ for all $e \in \mathcal{E}$.

Corollary 2. *Let $d \in [0, d^{SPEC}]$ satisfies (7.18), then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d)$ implies $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d^{SPEC})$.*

Proof of Theorem 11 and Corollary 2. Consider any $d > d^{SPEC}$. From the statement of Theorem 11, we have that $\mathcal{U}_{\varphi(e;d)} \subseteq \mathcal{U}_{\varphi(e;d^{SPEC})}$. Hence, $\mathcal{E}_{\varphi}(d) \subseteq \mathcal{E}_{\varphi}(d^{SPEC})$ follows from the definition of $\mathcal{E}_{\varphi}(d)$ in (7.6). $\mathcal{U}_{\varphi(e;d^{SPEC})} \subseteq \mathcal{U}_{\mathcal{S}}(e)$ and $\mathcal{E}_{\varphi}(d^{SPEC}) \subseteq \mathcal{E}_{\mathcal{S}}$ is already ensured by Proposition 3, and hence Theorem 1 follows.

We now prove that for all $0 < d < d^{SPEC}$, $\exists\ e \in \mathcal{E}$ such that (7.18) does not hold, and hence the result of Theorem 1 trivially holds. We prove the result by contradiction. Suppose $0 < d < d^{SPEC}$ be such that (7.18) holds. Let $(e^*, \mathbf{u}^*)$ be the environment, controller pair where $d(\xi_{\mathcal{M}}(\cdot; x_0^*, \mathbf{u}^*), \xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*)) = d^{SPEC}$. Equation (7.12) and (7.13) thus imply that

$$\min_{t \in \mathcal{T}_H} \left(\min\{h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), A^*(t)\right), -h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), R^*(t)\right)\}\right) = d^{SPEC}, \tag{7.19}$$

and $\xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Equation (7.19) implies that

$$\forall t \in \mathcal{T}_H,\ h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), A^*(t)\right) \geq d^{SPEC} \tag{7.20}$$

$$\forall t \in \mathcal{T}_H,\ h\left(\xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*), R^*(t)\right) \leq -d^{SPEC}. \tag{7.21}$$

Equations (7.20) and (7.21) imply that

$$\forall t \in \mathcal{T}_H,\ \xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*) \notin A^*(t) \oplus d, \xi_{\mathcal{M}}(t; x_0^*, \mathbf{u}^*) \in R^*(t) \ominus d. \tag{7.22}$$

Consequently, we have $\xi_{\mathcal{M}}(\cdot; x_0^*, \mathbf{u}^*) \models \varphi(e^*; d)$. This contradicts (7.18) since $\xi_{\mathcal{S}}(\cdot; x_0^*, \mathbf{u}^*) \not\models \varphi(e^*)$. Therefore, for all $0 < d < d^{SPEC}$, $\exists\ e \in \mathcal{E}\ , \mathbf{u} \in \mathcal{U}_{\Pi}(e)\ , \xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \not\Rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$.

To prove the corollary, we first prove that if $d_1 > d_2$, then $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$ implies $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)\ , \forall e \in \mathcal{E}$. Since $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_1)$, we have

$$\forall t \in \mathcal{T}_H, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_1, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_1$$

Since $d_1 > d_2$, the above equation implies that

$$\forall t \in \mathcal{T}_H, \xi_{\mathcal{M}}(t) \notin A(t) \oplus d_2, \xi_{\mathcal{M}}(t) \in R(t) \ominus d_2.$$

Therefore, $\xi_{\mathcal{M}}(\cdot) \models \varphi(e; d_2)$. The corollary now follows from noting that for all $0 < d < d^{SPEC}$, $\exists\ e \in \mathcal{E}\ , \mathbf{u} \in \mathcal{U}_{\Pi}(e)\ , \xi_{\mathcal{M}}(\cdot) \models \varphi(e; d) \not\Rightarrow \xi_{\mathcal{S}}(\cdot) \models \varphi(e)$. □ We conclude this section by discussing the relative advantages and limitations of SPEC and SSM, and a few remarks.

Comparing SPEC and SSM SSM is specification-independent (and hence environment-independent); and hence can be reused across different tasks and environments. This is ensured by computing the distance between trajectories across all input control sequences; however, the very same aspect can make SSM overly-conservative. Making SPEC specification-dependent trades in generalizability for a less conservative measure. Although environment-dependent, the set of safe environments obtained using SPEC is larger compared to SSM. This is an important trade-off to make for any distance metric–the utility of a distance metric could be somewhat limited if it is too conservative.

The computational complexities for computing SPEC and SSM are the same since they both can be computed using Algorithm 6. To compute SSM we sample from a domain of all finite horizon controls. To compute SPEC we additionally need to be able to define and sample from the set of environment scenarios, but we believe that some representation of the environment scenarios is important for practical applications.

Remark 5. *The proposed framework can also be used in the scenarios where there is a deterministic controller for each environment. In such cases, $\mathcal{U}_\Pi(e)$ (and $\mathcal{U}_{\varphi(e)}$) is a singleton set for every environment e (see Section 7.8.2 for an example). However, from a control theory perspective, it might be useful to have a set of safe controllers that have different transient behaviors, that the system designer can choose from without recomputing the distance metric.*

7.6 Distance Metric Computation

In this section we describe the construction and design of $\mathcal{O}_{SIM}$ to compute a high confidence estimate of d^{SPEC}. Since a dynamics model of $\mathcal{S}$ is not available, the computation of the distance bound d^{SPEC} is generally difficult. Interestingly, this computational issue can be resolved using a randomized approach, such as scenario optimization [20]. Scenario optimization has been used for a variety of purposes [23, 22], such as robust control, model reduction, as well as for the computation of SSM [2].

Computing d^{SPEC} by scenario optimization (in $\mathcal{O}_{SIM}$) is summarized in Algorithm 6. We start by (randomly) extracting N realizations of the environment e_i, $i = 1, 2, \ldots, N$ (Line 2). Each realization e_i consists of an initial state x_0^i, and a sequence of reach and avoid sets, $A^i(t)$ and $R^i(t)$. For each e_i, we extract a controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ (Line 5). If such a controller does not exist, we denote $\mathbf{u}_i$ to be a null controller $\mathbf{u}_\phi$. $\mathbf{u}_i$ (if not $= \mathbf{u}_\phi$) is then applied to both the system as well as the abstraction to obtain the corresponding trajectories $\xi^i_{\mathcal{S}}(\cdot; x_0^i, \mathbf{u}_i)$ and $\xi^i_{\mathcal{M}}(\cdot; x_0^i, \mathbf{u}_i)$ (Line 6). We next compute the distance between these two trajectories, d_i, using (7.13) (Line 7). If $\mathbf{u}_i = \mathbf{u}_\phi$, no satisfying controller exists for $\mathcal{M}$, and hence $d(\xi_{\mathcal{S}}(\cdot; x_0^i, \mathbf{u}_n), \xi_{\mathcal{M}}(\cdot; x_0^i, \mathbf{u}_n))$ is trivially 0. The maximum across all these distances, $\hat{d}_\epsilon$, is then used as an estimate for d^{SPEC} (Line 10). Algorithm 6 can be formalized in the OGIS framework as well, $\mathcal{O}_{SIM} := \mathcal{I}_{SPEC} = (\mathcal{L}_{SPEC}, \mathcal{O}_{SPEC})$. The $\mathcal{O}_{SPEC}$ is a black-box physics simulator which takes as input an $env_i \in \mathcal{E}$ and control sequence $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ and simulates the unknown system $\mathcal{S}$ to produce a finite-horizon trajectory $\xi^i_{\mathcal{S}}(\cdot; x_i, \mathbf{u}_i)$. At each

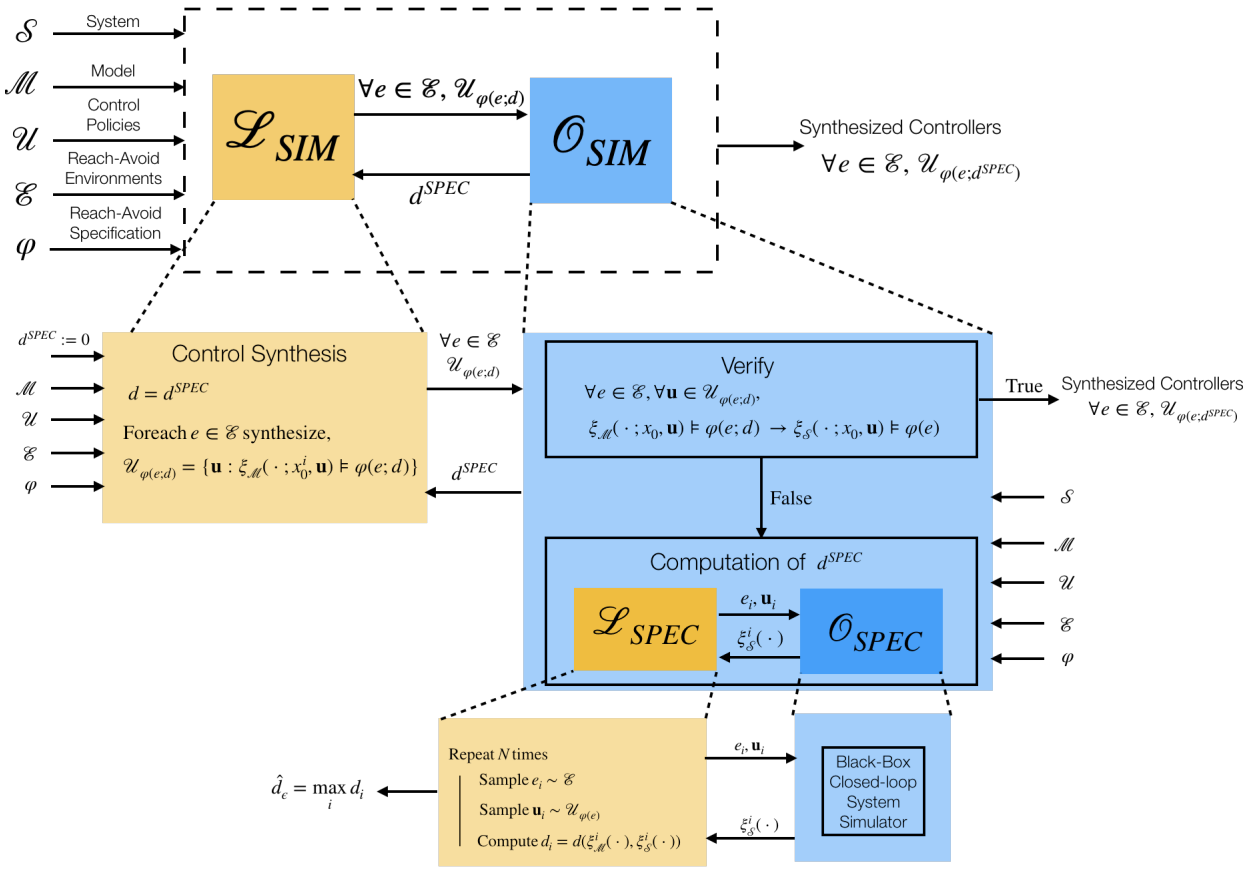


Figure 7.2. Hierarchical OGIS to extract safe environments and controllers for $\mathcal{S}$ with unknown dynamics: $\mathcal{I}_{SIM} = (\mathcal{L}_{SIM}, \mathcal{O}_{SIM})$. The oracle $\mathcal{O}_{SIM}(\mathcal{L}_{SIM})$ is shown in blue (yellow). The learner $\mathcal{L}_{SIM}$ synthesizes controllers for $\mathcal{S}$ using the model $\mathcal{M}$. To do so it modifies the specification for the model $\varphi(e; d^{SPEC})$. The oracle $\mathcal{O}_{SIM}$ first verifies if the synthesized controller is safe for the system. If yes, it terminates and outputs the synthesized controller. If not, it computes a high confidence estimate $\hat{d}_\epsilon$ of SPECd^{SPEC}. To compute $\hat{d}_\epsilon$, we use Scenario Optimization which is formalized as a OGIS framework $\mathcal{I}_{SPEC} = (\mathcal{L}_{SPEC}, \mathcal{O}_{SPEC})$ where $\mathcal{O}_{SPEC}$ is a black-box physics simulator and $\mathcal{L}_{SPEC}$ implements scenario optimization.

iteration, the learner $\mathcal{L}_{SPEC}$ randomly samples $e_i \in \mathcal{E}$ and extract a controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ and send it to the oracle. When the oracle returns the system trajectory $\xi^i_{\mathcal{S}}(\cdot; \mathbf{u}_i, e_i)$, the learner computes d_i. The OGIS framework terminates after N iterations and returns $\hat{d}_\epsilon$ to the top level learner $\mathcal{L}_{SIM}$. The overall framework is shown in Figure 7.2.

Although simple in its approach, scenario optimization provides provable approximation guarantees. In Algorithm 6, we have to sample both an $e \in \mathcal{E}$ and a corresponding controller $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$. We define a joint sample space

$$\mathcal{D} = \{(e \times \mathcal{U}_{\varphi(e)}) : e \in \mathcal{E}, \mathcal{U}_{\varphi(e)} \neq \emptyset\} \cup \{(e, \mathbf{u}_\phi) : e \in \mathcal{E}, \mathcal{U}_{\varphi(e)} = \emptyset\} \tag{7.23}$$

$\mathcal{D}$ contains all feasible $(e, \mathbf{u})$ pairs for $\mathcal{M}$. We create a dummy sample $(e, \mathbf{u}_\phi)$ for all e where a satisfying controller does not exist for the abstraction. We next define a probability

distribution on $\mathcal{D}$, $p(e, \mathbf{u}) = p(e) \cdot p(\mathbf{u} \mid e)$ where $p(e)$ is probability of sampling $e \in \mathcal{E}$ and $p(\mathbf{u} \mid e)$ is the probability of sampling $\mathbf{u} \in \mathcal{U}_{\varphi(e)}$ given e. This distribution is key to capture the sequential nature of sampling $\mathbf{u}$ only after sampling e. For $e \in \mathcal{E}$ where $\mathcal{U}_{\varphi(e)} = \emptyset$, $p(\mathbf{u}_\phi \mid e) = 1$ since $\mathcal{D}$ has only a single entry for e, i.e, $(e, \mathbf{u}_\phi)$. In Algorithm 6, in Line 2, we sample $e_i \sim p(e)$. In Line 5, we sample $\mathbf{u}_i \sim p(\mathbf{u} \mid e_i)$.

Proposition 4. *Let $\mathcal{D}$ be the joint sample space as defined in* (7.23)*, with the probability distribution $p_{\mathcal{D}} = p(e, \mathbf{u})$. Select a 'violation parameter' $\epsilon \in (0, 1)$ and a 'confidence parameter' $\beta \in (0, 1)$. Pick N such that*

$$N \geq \frac{2}{\epsilon}\left(\ln \frac{1}{\beta} + 1\right), \tag{7.24}$$

then, with probability at least $1 - \beta$, the solution $\hat{d}_\epsilon$ to Algorithm 6 satisfies the following conditions:

1. $\mathbb{P}((e, \mathbf{u}) \in \mathcal{D} : d(\xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}), \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u})) > \hat{d}_\epsilon) \leq \epsilon$
2. $\mathbb{P}\left((e, \mathbf{u}) \in \mathcal{D} : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\right) > 1 - \epsilon$ *provided* $\mathcal{U}_{\varphi(e, \hat{d}_\epsilon)} \subseteq \mathcal{U}_{\varphi(e)}$.

Proof. Statement (1) of Proposition 4 follows directly from the guarantees provided by Scenario Optimization (Theorem 1 in [23]). To use the result in [23], we need to prove: (a) computing d^{SPEC} can be converted into a standard Scenario Optimization problem and (b) Algorithm 6 samples i.i.d from $\mathcal{D}$ with probability $p_{\mathcal{D}}$.

(7.12) can be re-written as $d^{SPEC} = \max_{(e,\mathbf{u}) \in \mathcal{D}} d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot))$ which can be formalized as the following optimization problem,

$$\begin{aligned} &\min\ g \\ &\text{s.t. } \forall (e, \mathbf{u}) \in \mathcal{D}\ , d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) \leq g \end{aligned}$$

This is semi-infinite optimization problem where the constraints are convex (in fact, linear) in the optimization variable g for any given $(e, \mathbf{u})$. Statement (1) now follows from Theorem 1 in [23] by replacing $c = 1$, γ by g, Δ by $\mathcal{D}$, and f by $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) - g$. Theorem 1 in [23], however, requires that i.i.d samples are chosen from the distribution $p_{\mathcal{D}}$. This can be proved by noticing that, in Algorithm 6, we first sample $e_i \sim p(e)$ (in Line 2), and then sample $\mathbf{u}_i \sim p(\mathbf{u} \mid e)$ (in Line 4.) Hence, every $(e_i, \mathbf{u}_i)$ is sampled from $p_{\mathcal{D}} = p(e) \cdot p(\mathbf{u} \mid e)$. Since each $i = 1, \ldots, N$ is sampled randomly and independent of each other, the $(e_i, \mathbf{u}_i)$ pairs are indeed sampled i.i.d from $p_{\mathcal{D}}$.

Algorithm 6 returns an estimate $\hat{d}_\epsilon$ for d^{SPEC}. We have already established that $\hat{d}_\epsilon$ satisfies the probabilistic guarantees provided by scenario optimization (Statement (1)). From Proposition 3, we have $\forall (e, \mathbf{u}) \in \mathcal{D}$ where $d(\xi_{\mathcal{S}}(\cdot), \xi_{\mathcal{M}}(\cdot)) \leq \hat{d}_\epsilon$, $\xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)$, provided $\mathcal{U}_{\varphi(e; \hat{d}_\epsilon)} \subseteq \mathcal{U}_{\varphi(e)}$. Therefore,

$$\mathbb{P}\left((e, \mathbf{u}) \in \mathcal{D} : \xi_{\mathcal{M}}(\cdot; x_0, \mathbf{u}) \models \varphi(e; \hat{d}_\epsilon) \rightarrow \xi_{\mathcal{S}}(\cdot; x_0, \mathbf{u}) \models \varphi(e)\right) > 1 - \epsilon.$$

□

Intuitively, Proposition 4 states that $\hat{d}_\epsilon$ is a high confidence estimate of d^{SPEC}, if a large enough N is chosen. If we discard the confidence parameter β for a moment, this proposition states that the size of the violation set (the set of $(e, \mathbf{u}) \in \mathcal{D}$ where the corresponding distance is greater than $\hat{d}_\epsilon$) is smaller than or equal to the prescribed ϵ value. As ϵ tends to zero, $\hat{d}_\epsilon$ approaches the desired optimal solution d^{SPEC}. In turn, the simulation effort grows unbounded since N is inversely proportional to ϵ.

As for the confidence parameter β, one should note that $\hat{d}_\epsilon$ is a random quantity that depends on the randomly extracted $(e, \mathbf{u})$ pairs. It may happen that the extracted samples are not representative enough, in which case the size of the violation set will be larger than ϵ. Parameter β controls the probability that this happens; and the final result holds with probability $1 - \beta$. Since N in (7.24) depends logarithmically on $1/\beta$; β can be pushed down to small values such as 10^{-16}, to make $1 - \beta$ so close to 1 to lose any practical importance.

Finally, once we have a high confidence estimate of d^{SPEC}, we can use it with Proposition 3 to provide guarantees on the safety of a controller for the system, provided that it is safe for the abstraction. (Statement (2) in Proposition 4)

Note that the controller $\mathbf{u}_i$ is extracted randomly from the set $\mathcal{U}_{\varphi(e_i)}$ (Line 5). Obtaining $\mathcal{U}_{\varphi(e_i)}$ and randomly sampling from it can be challenging in itself depending on the control scheme, Π, and the specification, $\varphi(e_i)$. However, one way to randomly extract $\mathbf{u}_i$ is using rejection sampling, i.e., we randomly sample controllers from the set $\mathcal{U}_\Pi$ until we find a controller that satisfies the specification for the model. Since the controller performance is evaluated only on the model during this process, it is often cheap and does not put the system at risk. Nevertheless, choosing a good control scheme makes this process more efficient, as the number of samples rejected before a feasible controller is found will be fewer (see Section 7.7 for further discussion on this). Rejection sampling, however, poses a problem when $\mathcal{U}_{\varphi(e_i)} = \emptyset$ and there is no way of knowing that beforehand. In such cases, one can impose a limit on the number of rejected samples to make sure the algorithm terminates. This problem can also be overcome easily when there is a single safe controller for each environment, i.e., $\mathcal{U}_{\varphi(e_i)}$ is a singleton set (see Remark 1).

Remark 6. *Even though we have presented scenario optimization to estimate d^{SPEC}, alternative derivative free optimization approaches such as Bayesian optimization, simulated annealing, evolutionary algorithms, and covariance matrix adaptation can be used as well. However, for many of these algorithms, it might be challenging to provide formal guarantees on the quality of the resultant estimate of the distance bound.*

Algorithm 6 samples N environment scenarios and corresponding controllers prior to running any executions on $\mathcal{M}$ and $\mathcal{S}$. Imagine at iteration i, we have $d_i > 0$; and if at iteration $(i + 1)$, $d_{i+1} < d_i$, then the $(i + 1)$th sample is not informative for approximating d^{SPEC}. A simple way to overcome this issue would be to consider only $\mathcal{U}_{\varphi(e_i;d_i)}$ as the set of feasible controllers at the $(i+1)$th iteration; i.e., consider controllers where $\xi_{\mathcal{M}}(\cdot) \models \varphi(e_i; d_i)$. This variant of Algorithm 1 would reduce the number of executions of the system; and ensure

Algorithm 6 Scenario optimization for estimating SPEC (OGIS $\mathcal{I}_{SPEC}$

1: **procedure** SCENARIO OPTIMIZATION($\mathcal{S}, \mathcal{M}, \mathcal{E}, \mathcal{U}_\Pi, \mathcal{E}$)
2: set $\hat{d}_\epsilon = 0$
3: extract N realizations of the environment $e_i, i = 1, 2, \ldots, N$
4: **for** $i = 0 : N-1$ **do**
5: **if** $\mathcal{U}_{\varphi(e_i)} \neq \emptyset$ **then**
6: extract a realization of a feasible controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$
7: run the controller $\mathbf{u}_i$ on $\mathcal{S}$ and $\mathcal{M}$, and obtain $\xi^i_{\mathcal{S}}(\cdot)$ and $\xi^i_{\mathcal{M}}(\cdot)$
8: compute $d_i = d(\xi^i_{\mathcal{S}}(\cdot), \xi^i_{\mathcal{M}}(\cdot))$
9: **else**
10: $\mathbf{u}_i = \mathbf{u}_\phi$ and $d_i = 0$
11: set $\hat{d}_\epsilon = \max_{i \in \{1,2,\ldots,N\}} d_i$
12: return $\hat{d}_\epsilon$

Algorithm 7 Iterative estimating SPEC with modified specification (OGIS $\mathcal{I}_{SPEC}$

1: **procedure** ITERATIVE ESTIMATION($\mathcal{S}, \mathcal{M}, \mathcal{E}, \mathcal{U}_\Pi, \mathcal{E}$)
2: set $\tilde{d}_\epsilon = 0$
3: extract N realizations of the environment $e_i, i = 1, 2, \ldots, N$
4: **for** $i = 0 : N-1$ **do**
5: **if** $\mathcal{U}_{\varphi(e_i;\tilde{d}_\epsilon)} \neq \emptyset$ **then**
6: extract a realization of a feasible controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i;\tilde{d}_\epsilon)}$
7: run the controller $\mathbf{u}_i$ on $\mathcal{S}$ and $\mathcal{M}$, and obtain $\xi^i_{\mathcal{S}}(\cdot)$ and $\xi^i_{\mathcal{M}}(\cdot)$
8: compute $d_i = d(\xi^i_{\mathcal{S}}(\cdot), \xi^i_{\mathcal{M}}(\cdot))$
9: **else**
10: $\mathbf{u}_i = \mathbf{u}_\phi$ and $d_i = 0$
11: set $\tilde{d}_\epsilon = \max(\tilde{d}_\epsilon, d_i)$
12: return $\hat{d}_\epsilon$

that each execution is informative for estimating d^{SPEC}. To implement this scheme, we would maintain a running max $d^{SPEC}_{(i)}$ which contains the maximum of d_i seen till now. In iteration $(i+1)$, instead of sampling from $\mathcal{U}_{\varphi(e)}$ in Line 5, we sample from $\mathcal{U}_{\varphi(e,d^{SPEC}_{(i)})}$. Further, before the end of loop, in Line 7, we update $d^{SPEC}_{(i+1)} = \max(d^{SPEC}_{(i)}, d_{i+1})$. This also ensures that the sequence $d_0, d_1, \ldots, d_N$ is a non-decreasing sequence, which gives us. This modified algorithm is shown in Algorithm 7.

In Algorithm 7. We initialize $\tilde{d}_\epsilon = 0$ (Line 1). We next (randomly) extract N realizations of the environment e_i, $i = 1, 2, \ldots, N$ (Line 2). Each realization e_i consists of a starting state x^i_0, a set of reach and avoid states over time t, $A^i(t)$ and $R^i(t)$, and an environment parameter p^i. For each sampled e_i, we randomly sample a controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i;\tilde{d}_\epsilon)}$ if $\mathcal{U}_{\varphi(e_i;\tilde{d}_\epsilon)} \neq \emptyset$; or else we set it to $\mathbf{u}_\phi$ (Line 4). If $\mathbf{u}_i \neq \mathbf{u}_\phi$, we execute the controller on $\mathcal{M}$ and $\mathcal{S}$ and record the

trajectories $\xi_{\mathcal{M}}(\cdot)$ and $\xi_{\mathcal{S}}(\cdot)$ (Line 5). We compute the distance d_i in Line 6 and update $\tilde{d}_\epsilon$ in Line 7.

Our OGIS framework $\mathcal{I}_{SIM}$ extends easily to compute the d^{SPEC} using Algorithm 7 by simply replacing $\mathcal{L}_{SPEC}$ in $\mathcal{O}_{SIM}$ with Algorithm 7.

Our OGIs framework $\mathcal{I}_{SIM}$ can also be extended to compute the d^{sim} by replacing the $\mathcal{L}_{SPEC}$ by $\mathcal{L}_{SSM}$ which computes the distance in (7.7) instead of (7.12) in Algorithm 6 (or Algorithm 7).

7.7 Running Example: Distance Computation

We now apply the proposed algorithm to compute d^{SPEC} for the setting described in Section 7.3. $\mathcal{U}_{\varphi(e)}$ in this case is given as

$$\mathcal{U}_{\varphi(e)} = \{\mathbf{u} \in \mathcal{U}_\Pi(e) : \|\xi_{\mathcal{M}}(H; x_0, \mathbf{u}) - x^*\|_2 < \gamma\},$$

where $\mathcal{U}_\Pi(e)$ is the set of LQR controllers (see Section 7.3). To illustrate the importance of the choice of distance metric, we compute two different distance metrics between $\mathcal{S}$ and $\mathcal{M}$: d^{sim} in (7.7) and d^{SPEC} in (7.12). To compute d^{SPEC}, we use Algorithm 6. To compute d^{sim}, we modify Algorithm 6 to sample a random controller from $\mathcal{U}_\Pi(e)$ in Line 5 and compute d_i using (7.7) in Line 7. Here, the $\mathcal{L}_{SIM}$ synthesizes LQR controllers in $\mathcal{U}_\Pi(e) = \{LQR(q, x^*) : 0.1 \leq q \leq 100\}$ while $\mathcal{O}_{SIM}$ implements scenario optimization in Algorithm 6.

According to the scenario approach with $\epsilon = 0.01$ and $\beta = 10^{-6}$, we extract $N = 2964$ different reach-avoid scenarios (i.e., N different final states to reach). For each $e_i, i \in \{1, 2, \ldots, 2964\}$, we obtain a feasible LQR controller $\mathbf{u}_i \in \mathcal{U}_{\varphi(e_i)}$ using rejection sampling. In particular, we randomly sample a penalty parameter q, solve the corresponding Riccati equation to obtain $LQR(q)$, and apply it on $\mathcal{M}$. If the corresponding $\xi_{\mathcal{M}}(\cdot)$ satisfies $\varphi(e_i)$, we use $\mathbf{u}_i$ as our feasible controller sample; otherwise, we sample a new q and repeat the procedure until a feasible controller is found. This procedure tends to be really fast and requires simulating only $\mathcal{M}$. A feasible controller was found within 3 samples of q for all e_i in this case. For d^{sim}, we randomly sample a penalty parameter q and use $LQR(q)$ as the controller.

The obtained distance metrics are $d^{sim} = 0.43, d^{SPEC} = 0$. Since $d^{sim} < \gamma$, it can be used to synthesize a safe controller for $\mathcal{S}$; however, we can synthesize controller only for those reach-avoid scenarios where $\mathcal{M}$ satisfies a much stringent specification: $\xi_{\mathcal{M}}$ must reach within a ball of radius 0.07 around the target state. Consequently, the set $\mathcal{E}_\varphi(d^{sim})$ is likely to be very small. In contrast, $d^{SPEC} = 0$; thus, Proposition 4 ensures that *any* controller designed on $\mathcal{M}$ that satisfies $\varphi(e)$ is guaranteed to satisfy it for $\mathcal{S}$ as well. In particular, the dynamics of $\mathcal{S}$ and $\mathcal{M}$ are same for the state x_1, and state x_2 is uncontrollable for $\mathcal{S}$ and remain 0 at all times. Thus, any controller that reaches within a ball of radius γ around a desired state x_1^* for $\mathcal{M}$, if applied on $\mathcal{S}$, also ensures that the system state reaches within the same ball. Even though this relationship between $\mathcal{S}$ and $\mathcal{M}$ is unknown, d^{SPEC} is able to capture it only through simulations of $\mathcal{S}$. This example also illustrates that d^{SPEC}

significantly reduces the conservativeness in SSM, and does not unnecessarily contract the set of safe environments.

7.8 Evaluation

We now demonstrate how SPEC can be used to obtain the safe set of environments and controllers for an autonomous quadrotor and an autonomous car. In Section 4.8.2, we demonstrate how SPEC provides much larger safe sets compared to SSM. In Section 7.8.2, we demonstrate how SPEC not only captures the differences between the dynamics of $\mathcal{S}$ and $\mathcal{M}$, but also other aspects of the system, in particular the sensor error, that might affect the satisfiability of a specification.

7.8.1 Safe Altitude Control for Quadrotor

Our first example illustrates how the proposed distance metric behaves when the only difference between the system and the abstraction is the value of one parameter. However, unlike the running example, the system and the abstraction dynamics are non-linear. Moreover, we illustrate how SPEC can be used in the cases where all safe controllers for $\mathcal{M}$ may not be safe for $\mathcal{S}$.

We use the reach-avoid setting described in [55], where the authors are interested in controlling the altitude of a quadrotor in an indoor setting while ensuring that it does not go too close to the ceiling or the floor, which are obstacles in our experiments.

A dynamic model of quadrotor vertical flight can be written as:

$$\begin{aligned} z(t+1) =& z(t) + \Delta v_z(t) \\ v_z(t+1) =& v_z(t) + \Delta(ku(t) + g), \end{aligned} \tag{7.25}$$

where z is the vehicle's altitude, v_z is its vertical velocity and u is the commanded average thrust. The gravitational acceleration is $g = -9.8m/s^2$ and the discretization step Δ is 0.01. The control input $u(t)$ is bounded to $[0, 1]$. We are interested in designing a controller for $\mathcal{S}$ that ensures safety over a horizon of 100 timesteps. In particular, we have $\mathcal{X}_0 = \{(z, v_z) : 0.5 \leq z \leq 2.5 \wedge -3 \leq v_z \leq 4\}$, $\mathcal{A} = \{A(\cdot)\}$, and $\mathcal{R} = \mathbb{R}^2$. The avoid set at any time t is given as $A(t) = \{(z, v_z) \in \mathbb{R}^2 : 0.5m \leq z(t) \leq 2.5m\}$. We again assume that the dynamics in (7.25) are unknown. Consider an abstraction of $\mathcal{S}$ with same dynamics as (7.25) except that the value of parameter k in the abstraction dynamics, $k_{\mathcal{M}}$, is different.

The space of controllers $\mathcal{U}_\Pi(e)$ is given by all possible control sequences over the time horizon (i.e., $\mathcal{U}_\Pi(e) = \mathcal{U}$.) For computing $\mathcal{U}_{\varphi(e)}$, we use the Level Set Toolbox [108, 106] that gives us both the set of initial states from which there exist a controller that will keep the $\xi_{\mathcal{M}}(\cdot)$ outside the avoid set at all times (also called the reachable set), as well as the corresponding least restrictive controller. In particular, we can apply any control when the abstraction trajectory is inside the reachable set and the safety-preserving control (given by the toolbox) when the trajectory is near the boundary of the reachable set. For computation

of the distance bounds, we sample a random controller sequence according to this safety-preserving control law. If any initial state lies outside the reachable set, then it is also guaranteed that $\mathcal{U}_{\varphi(e)} = \emptyset$ so we do not need to do any rejection sampling in this case.

When $k_{\mathcal{M}} < k$, $\mathcal{M}$ has strictly less control authority compared to $\mathcal{S}$. Thus, any controller that satisfies the specification for $\mathcal{M}$ will also satisfy the specification for $\mathcal{S}$, so $\mathcal{E}_{\varphi}(0)$ itself is an under approximation of $\mathcal{E}_{\mathcal{S}}$. SPEC is again able to capture this behavior. Indeed, we computed an estimate for the distance bound using Algorithm 6 and the obtained numbers are $d^{sim} = 0.30$ and $d^{SPEC} = 0$. Note that not only is d^{sim} conservative, it may not be particularly useful in synthesizing a safe controller for $\mathcal{S}$. d^{sim} computed using Algorithm 6 ensures that a safe controller designed on $\mathcal{M}$ for $\varphi(e; d^{sim})$ is also safe for $\mathcal{S}$ with high probability, only when this controller is *randomly* selected from the set $\mathcal{U}_{\Pi}$. However, a random controller selected from $\mathcal{U}_{\Pi}$ is unlikely to satisfy $\varphi(e; d^{sim})$ for $\mathcal{M}$ itself, and thus nothing can be said about $\mathcal{S}$ either. Thus, it is hard to *actually* compute an approximation of $\mathcal{E}_{\mathcal{S}}$. In contrast, d^{SPEC} samples a controller from the set $\mathcal{U}_{\varphi(e)}$ in Algorithm 6. Therefore, to synthesize a controller, we *randomly* select a controller from the set $\mathcal{U}_{\varphi(e;d^{SPEC})}$, which is guaranteed to be safe on both $\mathcal{M}$ and $\mathcal{S}$ with high probability. Therefore, it might be better to compare d^{SPEC} to d^{a}, which is defined similar to d^{sim}, except the inner maximum in (7.7) is computed over $\mathcal{U}_{\varphi(e)}$ instead. d^{a} in this case turns out to be 0.5.

Note that if we could instead compute the distance metrics exactly, $d^{a} \leq d^{sim}$, since $\mathcal{U}_{\varphi(e)} \subset \mathcal{U}_{\Pi}$. However, random sampling based estimate of d^{a} can be greater than that of d^{sim} if the controllers corresponding to a large distance between the $\xi_{\mathcal{S}}(\cdot)$ and $\xi_{\mathcal{M}}(\cdot)$ are sparse in $\mathcal{U}_{\Pi}$ compared to that in $\mathcal{U}_{\varphi(e)}$.

For illustration purposes, we also compute the reachable set $\mathcal{E}_{\varphi}(d^{SPEC})$, by augmenting the avoid set by d^{SPEC} and recomputing the reachable sets using the Level Set Toolbox. As shown in Figure 7.3, $\mathcal{E}_{\varphi}(0)$ (the area withing the blue contour) is indeed contained within $\mathcal{E}_{\mathcal{S}}$ (the area within the red contour). Here, $\mathcal{E}_{\mathcal{S}}$ has been computed using the system dynamics. Even though $\mathcal{E}_{\varphi}(d^{a})$ (the area within the magenta contour) is also contained in $\mathcal{E}_{\mathcal{S}}$, it is significantly smaller in size compared to $\mathcal{E}_{\varphi}(d^{SPEC})$.

When $k_{\mathcal{M}} > k$, $\mathcal{S}$ has strictly less control authority compared to $\mathcal{M}$. Consequently, there might exist some environments for which it is possible to synthesize a safe controller for $\mathcal{M}$, but the same controller when deployed on $\mathcal{S}$ might lead to an unsafe behavior. We again compute the distance bounds using Algorithm 6 and the obtained numbers are $d^{sim} = 0.30, d^{a} = 0.49, d^{SPEC} = 0.1$. The corresponding reachable sets are shown in Figure 7.4. Even though we start with an overly optimistic abstraction, both d^{a} and d^{SPEC} are able to compute an under approximation of $\mathcal{E}_{\mathcal{S}}$; however, the set estimated by d^{a} is, once again, overly conservative. Since the dynamics of $\mathcal{S}$ and $\mathcal{M}$ are known in this case we were able to compute the exact value of d^{sim} and d^{SPEC} and recompute the safe reach sets using the Level Set toolbox. The corresponding reachable sets are shown in Figure 7.5.

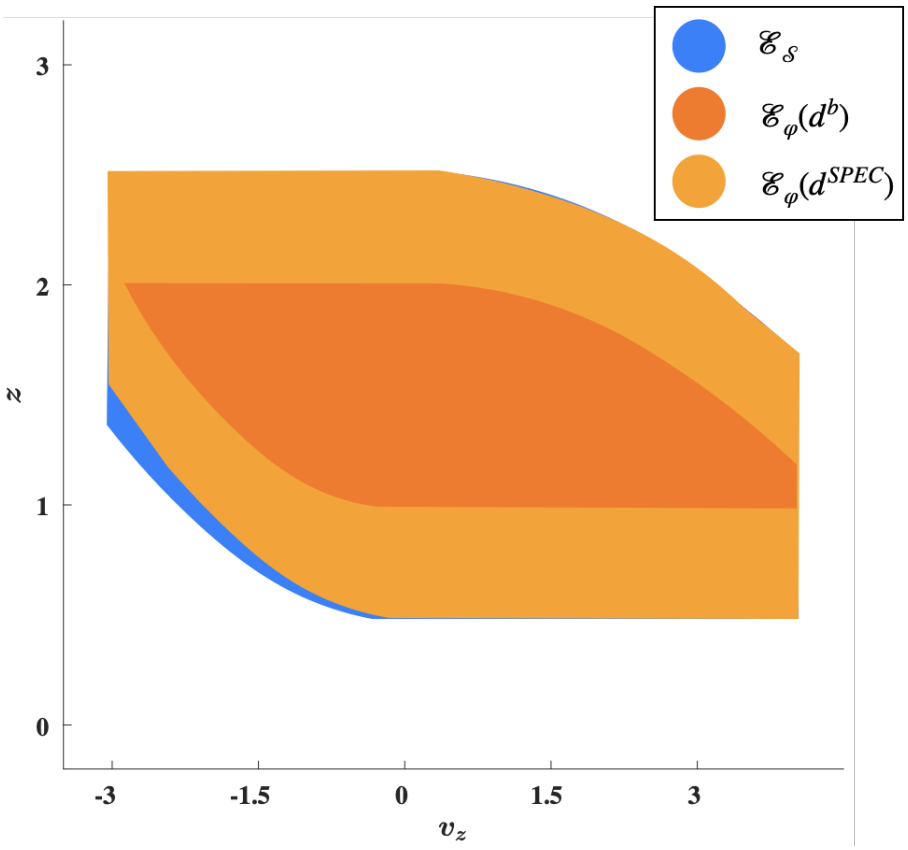
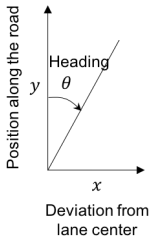


Figure 7.3. Different reachable sets when the quadrotor abstraction is conservative. The distance metric d^{SPEC} only considers the distance between trajectories that violates the specification on the system and satisfies it on the abstraction, leading to a less conservative estimate of the distance, and a better approximation of $\mathcal{E}_{\mathcal{S}}$.

7.8.2 Webots: Lane Keeping

We now show the application of the proposed metric for designing a safe lane keeping controller for an autonomous car.

In this example, we use the **Webots** simulator [140]. The car model within the simulator is our $\mathcal{S}$. For the abstraction $\mathcal{M}$ we consider the bicycle model,



$$\begin{aligned} \dot{x} &= v \cdot \sin\theta \\ \dot{y} &= v \cdot \cos\theta \\ \dot{v} &= a \\ \dot{\theta} &= \frac{v}{l} \tan\omega \end{aligned} \tag{7.26}$$

where $[x, y, v, \theta]$ is the state, representing perpendicular deviation from the center of the lane, position along the road, speed, and heading respectively. The maximum speed is limited to

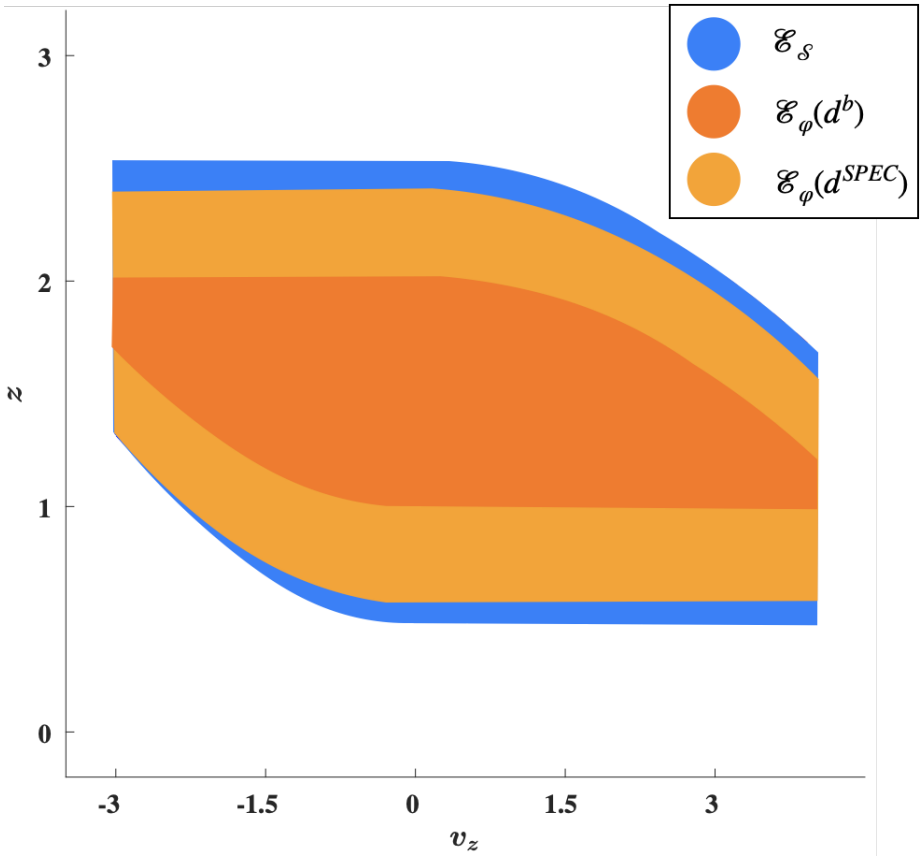


Figure 7.4. Different reachable sets when the quadrotor abstraction is overly optimistic. The distance metric d^{SPEC} achieves a far less conservative under-approximation of $\mathcal{E}_{\mathcal{S}}$ compared to the other distance metrics.

$v_{max} = 10$ km/hr. We have two inputs, (1) a discrete acceleration control $a = \{-\bar{a}, 0, \bar{a}\}$; and (2) a continuous steering control $\omega \in [-\pi/4, \pi/4] rad/s$. For our experiments, we use $H = 200$, which translates to about 6 seconds of simulated trajectory. The dynamics of $\mathcal{S}$ are typically much more complex than $\mathcal{M}$ and include the physical effects like friction and slip on the road.

In this case, $\mathcal{X}_0 = \{(x_0, \theta_0) : \|x\| \leq 0.2m \wedge \|\theta\| \leq \pi/4 rad\}$; the initial y_0 and v_0 is set to *zero*. $R(t) = \{[x(t), y(t), v(t), \theta(t)] \in \mathbb{R}^4 : \|x(t)\| \leq 0.5m\} \forall t \in \mathcal{T}_H$. The reach set corresponds to keeping the car within the $0.5m$ of the center of the lane. For keeping the car in the lane, the car is equipped with two sensors, a camera (to capture the lane ahead) and compass (to measure the heading of the car). There is an on board perception module, which first captures the image of the road ahead; and processes it to detect the lane edges and provide an estimate of the deviation of the car from the center of the lane.

There is another car (referred to as the environment car hereon) driving in the front of $\mathcal{S}$, which might obstruct the lane and cause the perception module to incorrectly detect the lane center. For each $e \in \mathcal{E}$, the set of possible initial states of the environment car is given

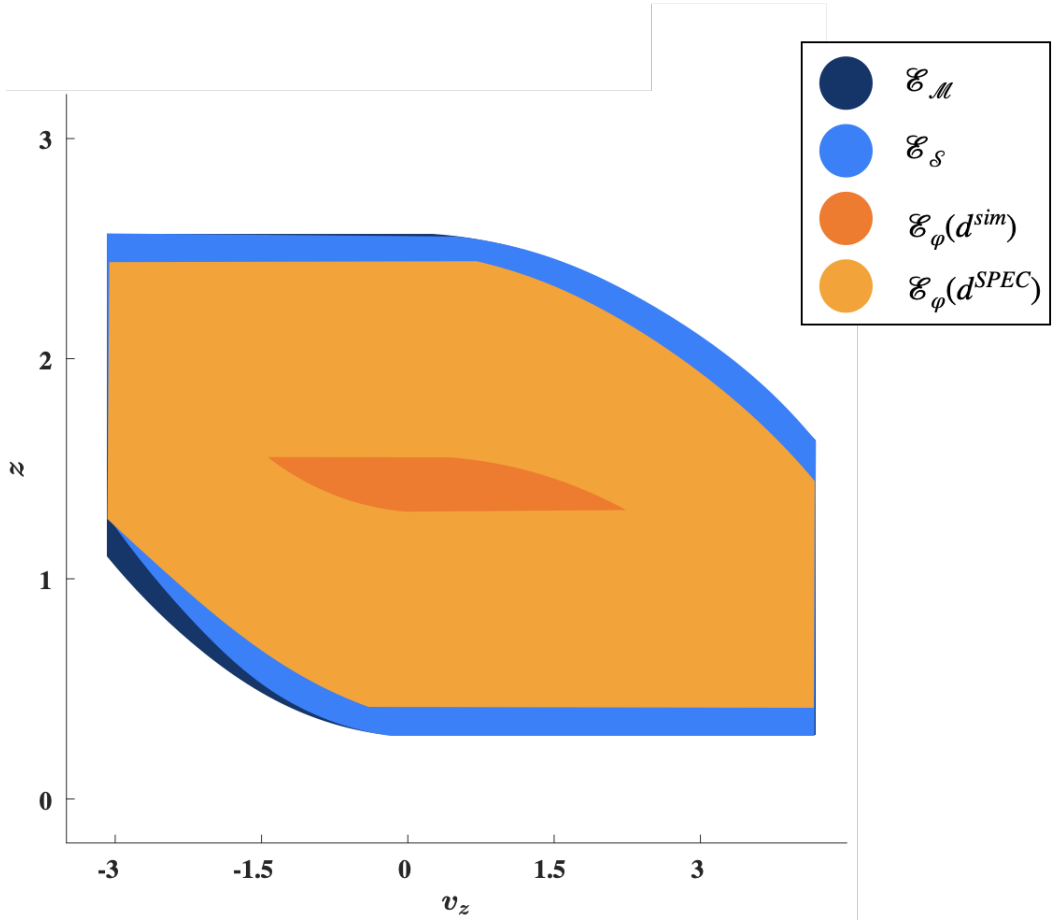


Figure 7.5. Different reachable sets when the quadrotor abstraction is overly optimistic. Here we compute the exact reach sets corresponding to $\mathcal{E}_{\mathcal{M}}, \mathcal{E}_{\mathcal{S}}, \mathcal{E}_{\varphi}(d^{SPEC})$ and $\mathcal{E}_{\varphi}(d^{sim})$. The distance metric d^{SPEC} achieves a far less conservative under-approximation of $\mathcal{E}_{\mathcal{S}}$ compared to d^{sim}.

by $\mathcal{P} = \{(x_e, y_e) : \|x_e - x_0\| \leq 2.0m \wedge 6.25m \leq y_e - y_0 \leq 8m\}$. We set the initial speed v_e and heading θ_e of the environment car to $v_{\max}$ and 0 respectively. We want to make sure that $\mathcal{S}$ remains within the lane despite all possible initial positions of the environment car. For this purpose, we compute the worst-case d^{SPEC} across all $p \in \mathcal{P}$.

If the environment car or its shadow covers the lane edges (see Figure 7.7 for some possible scenarios), then the lane detection fails. Technically speaking, if such a scenario occurs, then $\mathcal{S}$ should slow down and come to stop until the image processing starts detecting the lane again. Consequently, our control scheme $\mathcal{U}_{\Pi}$, is a hybrid controller shown in Figure 7.6, where in each mode the controller is given by an LQR controller (with a fixed Q and R matrix) corresponding to the (linearized) dynamics in that mode. In this example, our controller is a deterministic controller since the Q and R matrices are fixed, and hence $|\mathcal{U}_{\Pi}| = 1$. In Figure 7.6, in mode (1), the lane is detected and $v(t) < v_{max}$. When the $v(t) = v_{max}$ we transition to mode (2) given the lane is still detected. When the lane is no longer detected,

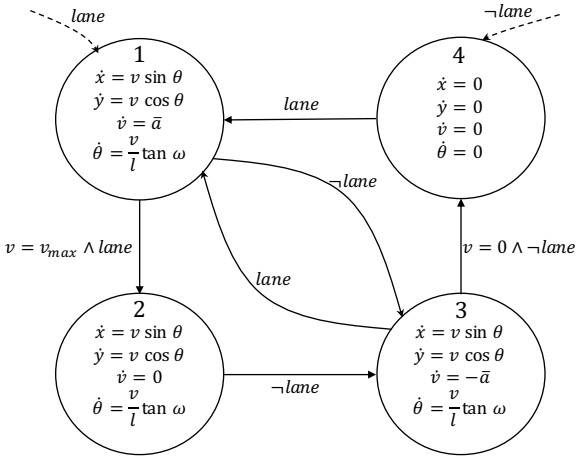
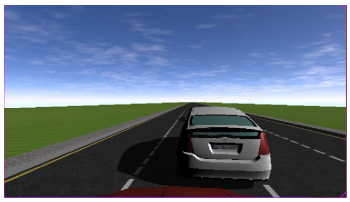


Figure 7.6. Hybrid controller for lane keeping. *lane* means a lane is detected by the perception system. The dashed line represents the transitions taken on initialization based on the value of *lane*. To closely follow the center of the lane, we synthesize a LQR controller in each mode.

(a) Environment car covers left lane.



(b) Shadow of environment car covers left lane.

(c) Lane detected correctly.

Figure 7.7. The lane detection fails for (a) and (b) and $\mathcal{S}$ car tries to slow down. When lane is correctly detected (c), the LQR controller tries to follow the lane

we transition to mode (3) if $v(t) > 0$, or mode (4) if $v(t) = 0$. In modes (3) and (4), the car slows down until the lane is detected again.

By setting $\epsilon = 0.01$ and $\beta = 1e-6$ we get $N \geq 2964$. We used Algorithm 6, to sample N different initial states of the $\mathcal{S}$, $(x_0, \theta_0) \in \mathcal{X}_0$; and environment car in the simulator, $p \in \mathcal{P}$. Since the controller is deterministic, the set of feasible controllers is a singleton set, and hence we do not need to sample a feasible controller (Line 5 in Algorithm 6). Among these environment scenarios, the controller on $\mathcal{M}$ is also able to safely control $\mathcal{S}$ for 2519 scenarios. $\hat{d}_\epsilon$ is determined entirely by the remaining 445 controller, and computed to be 0.34m. We show the application of the the computed $\hat{d}_\epsilon$ for a sample environment scenario in Figure 7.8. The green lines represent the original reach set. The yellow shaded region represents the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory (shown in blue) is contained in the yellow region and hence satisfies the more constrained specification. As a result, even though the system's trajectory (shown in dotted red) leaves the yellow region, it is contained within the original reach set at all times.

We also analyze these 445 environmental scenarios that contribute to $\hat{d}_\epsilon$, and notice that

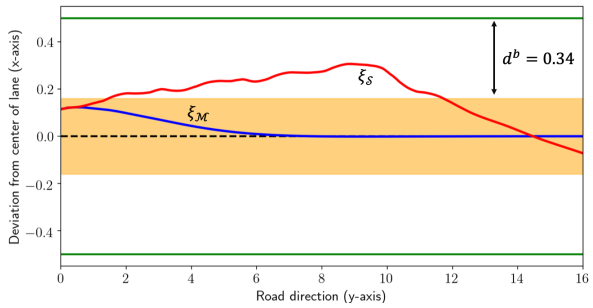


Figure 7.8. The green lines represent the boundaries of the original reach set. The yellow region is the contracted reach set for the model computed using $\hat{d}_\epsilon$. The model's trajectory shown in blue is entirely contained within the yellow region. Consequently, the system's trajectory (shown in dotted red) leaves the yellow region but is contained within the original reach set at all times.

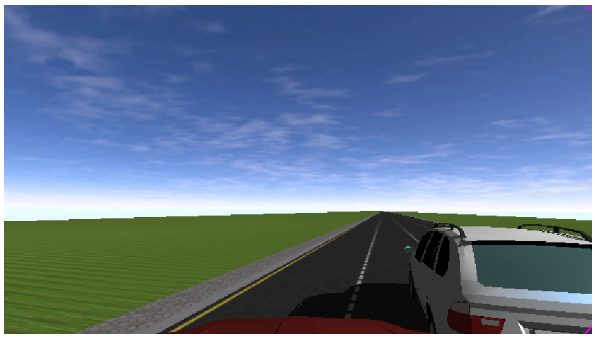


Figure 7.9. An example of the environment scenario that contributes to the distance between the model and the system. The environment samples used for computing SPEC can be used to identify the reasons behind the violation of the safety specification by the system.

the fault lies within the perception module. In Figure 7.9, we show one such scenario. In this case, $\theta_0 = -\pi/4$. Because of the left rotation of the car, the rightmost lane appears smaller and farther due to the perspective distortion. Furthermore, the presence of the environment car completely cover the rightmost lane in the image. The image processing module now detects the leftmost lane as the center lane and the center lane as the rightmost lane. Consequently, the module returns an inaccurate estimation of the center of the lane, causing $\mathcal{S}$ to go outside the center lane. This example illustrates that the samples in Algorithm 6 that contributed to $\hat{d}_\epsilon$ could also be used to analyze the reasons behind the violation of the safety specification by $\mathcal{S}$.

7.9 Practicality of SPEC

Computing $SPEC$ with real world experiments lead to two main issues (a) *sensor noise* while recording the system trajectory $\xi_{\mathcal{S}}$; and (b) *unsafe behaviors* rising during the experiments. We discuss how one can overcome these issues to compute $SPEC$ in the real world.

Sensor Noise: In the computation of $SPEC$ we only care about the behavior of the system ($\xi_{\mathcal{S}} \not\models \varphi(e)$), but the exact trajectory is not required for the computation. Since, we only need to observe the system behavior instead of being able to record it exactly, sensor noise would not affect our computation.

Unsafe Behaviors: Based on the definition of $SPEC$, we need only those behavior of the system which are unsafe i.e., $\xi_{\mathcal{S}} \not\models \varphi(e)$. This means, we would like to run real world experiments which are potentially unsafe for the system, which is not practical. To overcome this, we can either (a) create "fake" obstacles by designating a region as unsafe as opposed to placing an actual obstacle, (b) using data available from other experiments; and (c) use a human to take over the system in unsafe situations.

7.10 Conclusion

In this chapter, we proposed a framework to adapt controllers verified on models to the real world. Specifically, we developed a novel metric SPEC, which is the tightest lower bound required to adapt controllers to the real world. We conclude the chapter by showing how SPEC can be used to safely control quadrotor and self-driving car in simulation.

Chapter 8

VerifAI: A toolkit for Design and Analysis of Artificial Intelligence-Based Systems

8.1 Introduction

In this chapter, we present VerifAI, a software toolkit for the formal design and analysis of systems that include artificial intelligence (AI) and machine learning (ML) components. VerifAI particularly addresses challenges with applying formal methods to ML components, such as perception systems or decision maker (or controllers) based on deep neural networks, as well as systems containing them, and to model and analyze system behavior in the presence of environment uncertainty. We describe the initial version of VerifAI, which centers on simulation-based verification and synthesis, guided by formal models and specifications. VerifAI encompasses the techniques presented in this thesis, and creates an unifying toolkit where one can easily implement an OGIS or CEGIS framework to implement the different steps in the design pipeline for robotic systems.

The increasing use of artificial intelligence (AI) and machine learning (ML) in systems, including safety-critical systems, has brought with it a pressing need for formal methods and tools for their design and verification. As discussed in Section 6.1.1, for verification we need to define the system $\mathcal{S}$, environment $\mathcal{E}$ and the specification φ mathematically. However, AI/ML-based systems, such as autonomous vehicles, have certain characteristics that make the application of formal methods very challenging. We mention three key challenges here; see [135] for an in-depth discussion. First, with an increasing dependence of AI/ML components in safety-critical applications have raised the need to formally define safety of such components in the context of the overall system. For example, one of the key uses of AI/ML are for *perception*, the use of computational systems to mimic human perceptual tasks such as object recognition and classification, conversing in natural language, etc. For such perception components, writing a *formal specification* is extremely difficult, if not impossible.

Additionally, the signals processed by such components can be very high-dimensional, such as streams of images or LiDAR data. Second, *machine learning* being a dominant paradigm in AI, formal tools must be compatible with the data-driven design flow for ML and also be able to handle the *complex, high-dimensional structures* in ML components such as deep neural networks. Third, AI/ML-based systems operate can operate in very *complex environments*, with considerable uncertainty even about how many (which) agents are in the environment (both human and robotic), let alone about their intentions and behaviors. As an example, consider the difficulty in modeling urban traffic environments in which an autonomous car must operate. Indeed, AI/ML is often introduced into these systems precisely to deal with such complexity and uncertainty! From a formal methods perspective, this makes it very hard to create realistic environment models with respect to which one can perform verification or synthesis.

In this chapter, we introduce the VerifAI toolkit, our initial attempt to address the three core challenges — specification, learning, and environments — that are outlined above. VerifAI takes the following approach:

- *Specification* A ML component maps a concrete feature space (e.g. pixels, environment states) to an output such as a classification, prediction, or state estimate for perception components; or control decisions for controllers and decision-making. To deal with the lack of specification for such components, VerifAI analyzes them in the context of a closed-loop system using a system-level specification. Moreover, to scale to complex high-dimensional feature spaces, VerifAI operates on an *abstract feature space* (or *semantic feature space*) [45] that describes semantic aspects of the environment being perceived, not the raw features such as pixels.

- *Learning:* VerifAI aims to not only analyze the behavior of ML components but also use formal methods for their (re-)design. To this end, it provides features to (i) design the data set for training and testing as described in Chapter 5, (ii) analyze counterexamples using *Error tables* to gain insight into mistakes by the ML model as described in Chapter 5, as well as (iii) synthesize parameters, including hyperparameters for training algorithms and ML model parameters.

- *Environment Modeling:* Since it can be difficult, if not impossible, to exhaustively model the environments of AI-based systems, VerifAI aims to provide ways to capture a designer's assumptions about the environment, including distribution assumptions made by ML components, and to describe the abstract feature space in an intuitive, declarative manner. To this end, VerifAI provides users with Scenic [56], a probabilistic programming language for modeling environments. Scenic, combined with a renderer or simulator for generating sensor data, can produce semantically-consistent input for perception components. Details can be found in [56]. VerifAI also allows users to define the environment directly using the *abstract feature space*.

VerifAI is currently focused on AI-based cyber-physical systems (CPS), although its basic ideas can also be applied to other AI-based systems. As a pragmatic choice, we fo-

cus on simulation-based verification, where the simulator is treated as a black-box, so as to be broadly applicable to the range of simulators used in industry. This allows us to overcome the need to define the system mathematically. Our work is complementary to the work on industrial-grade simulators for AI/ML-based CPS. In particular, VerifAI enhances such simulators by providing formal methods for modeling (via the Scenic language), analysis (via temporal logic falsification), and parameter synthesis (via property-directed hyper/model-parameter synthesis). The input to VerifAI is a "closed-loop" CPS model, comprising a composition of the AI-based CPS system under verification with an environment model, and a property on the closed-loop model. The AI-based CPS typically comprises a perception component (not necessarily based on ML), a planner/controller (may or may not be ML based), and the plant (i.e., the system under control).

Given these, VerifAI offers the following use cases: (1) temporal-logic falsification based on algorithms in Chapter 6; (2) model-based fuzz testing; (3) counterexample-guided data augmentation based on algorithms in Chapter 5; (4) counterexample (error table) analysis based on Chapter 5; (5) hyper-parameter synthesis, and (6) model parameter synthesis. VerifAI is that it is the first tool to offer this suite of use cases in an integrated fashion, unified by a common representation of an abstract feature space, with an accompanying modeling language and search algorithms over this feature space, all provided in a modular implementation. The algorithms and formalisms in VerifAI are presented in papers published by the authors in other venues (e.g., [44, 46, 47, 45, 136, 56, 65]). While simulation-based verification or falsification of CPS models is well studied and several tools exist (e.g. [8, 50, 39]); VerifAI was the first to extend these techniques to CPS models with ML components [44, 46]. Work on verification of ML components, especially neural networks (e.g., [157, 60]), is complementary to the system-level analysis performed by VerifAI. Fuzz testing based on formal models is common in software engineering (e.g. [69]) but VerifAI extends these techniques to the CPS context. Similarly, property-directed parameter synthesis has also been studied in the formal methods/CPS community, but our work is the first to apply these ideas to the synthesis of hyper-parameters for ML training and ML model parameters. Finally, augmenting training/test data sets [47], implemented in VerifAI, is the first use of formal techniques for this purpose.

We first show the overall structure of VerifAI and then discuss each component. We then show VerifAI can be used across a series of application of verification and falsification in AI/ML models.

The results shown in this chapter is adapted from [49].

8.2 VerifAI Structure

Currently, VerifAI is focused on simulation-based design and analysis of AI components for perception and control (potentially those using ML) in the context of a closed-loop cyber-physical system. The overall structure of VerifAI is shown in Figure 8.1

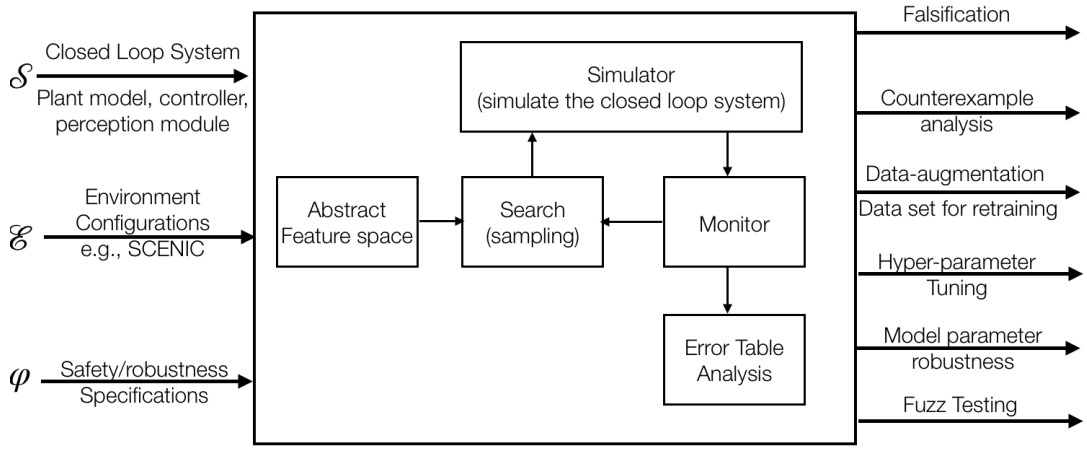


Figure 8.1. VERIFAI tool overview.

8.2.1 Inputs and Outputs

Inputs: Using VERIFAI requires setting up a simulator for the domain of interest. As we explain in Section 8.3, we have experimented with multiple robotics simulators and provide an easy interface to connect a new simulator. Till date VERIFAI has been interfaced with a number of robotics simulator like Webots [140], OpenAI Gym [17], CARLA [43] and XPlane [161]. The user then constructs the inputs to VERIFAI, including (i) a closed-loop model of the system which can be simulated, including code for one or more controllers and perception components, and a dynamical model of the system being controlled; (ii) a probabilistic model of the environment, specifying constraints on the workspace, the locations of agents and objects, and the dynamical behavior of agents, and (iii) a property over the composition of the system and its environment. VERIFAI can also take as input a model of the AI/ML component instead of the entire closed-loop system (Refer Section refsec:verifai:cegda where the system is the perception module under test). VERIFAI is implemented in Python for interoperability with ML/AI libraries and simulators across platforms. The code for the controller and perception component can be arbitrary executable code, invoked by the simulator. The environment model typically comprises a definition in the simulator of the different types of agents, plus a description of their initial conditions and other parameters using either the SCENIC probabilistic programming language [56] or directly defining the abstract feature space. Finally, the property to be checked can be expressed using Metric Temporal Logic (MTL) [3, 154], objective functions, or arbitrary code monitoring the property.

Outputs: The output of VERIFAI depends on the feature being invoked. For falsification, VERIFAI returns one or more *counterexamples*, simulation traces violating the property [44, 65]. Error analysis involves collecting counterexamples generated by the falsifier into a table $\mathbb{E}$, on which we perform analysis to identify features that are correlated with property fail-

ures (Refer to Section 5.6). Data augmentation uses falsification and error table analysis to generate additional data for training and testing an ML component [47] (Refer to Chapter 5). The property-driven synthesis of model parameters or hyper-parameters generates as output a parameter evaluation that satisfies the specified property. Finally, for fuzz testing, VERIFAI produces traces sampled from the distribution of behaviors induced by the probabilistic environment model [56].

8.2.2 Tool Components

We now discuss the the main modules of VERIFAI.

Abstract Feature Space The abstract feature space (referred to as $\mathcal{E}$) is a compact representation of the possible configurations of the simulation. Abstract features can represent parameters of the environment, controllers, or of ML components. For example, when analyzing a visual perception system for an autonomous car, an abstract feature space could consist of the initial poses and types of all vehicles on the road. Note that this abstract space, compared to the concrete feature space (of pixels or environment states) used as input to the controller or perception component, is better suited to the analysis of the overall closed-loop system (e.g. finding conditions under which the car might crash). The abstract feature space is similar tot he low dimensional modification space $\mathbb{M}$ described in the Chapter 5. In fact, for the data augmentation case study the abstract feature space corresponds to the 14-D modification space described in Section 5.4.1 in Chapter 5. VERIFAI provides two ways to construct abstract feature spaces. They can be constructed hierarchically, combining basic domains such as hyperboxes and finite sets into structures and arrays. For example, we could define a space for a car as a structure combining a 2D box for position with a 1D box for heading, and then create an array of these to get a space for several cars. Alternatively, VERIFAI allows a feature space to be defined using a program in the SCENIC language [56]. SCENIC provides convenient syntax for describing constrained geometric configurations and agent parameters, and, as a probabilistic programming language, allows placing a distribution over the feature space which can be conditioned by declarative constraints.

Search: Sampling the Feature Space Once the abstract feature space is defined, the next step is to search that space to find simulations that violate the property or produce other interesting behaviors. Currently, VERIFAI uses a suite of sampling methods for this purpose including exhaustive grid search, passive and active search techniques. In the future we expect to also integrate directed or exhaustive search methods including those from the adversarial machine learning literature (e.g., see [46, 45]). Passive samplers, which do not use any feedback from the simulation, include exhaustive grid search, uniform random sampling, simulated annealing, and Halton sequences [74] (quasi-random deterministic sequences with low-discrepancy guarantees we found effective for falsification [48, 44]). Distributions defined using SCENIC are also passive in this sense. Active samplers, whose selection of samples is

informed by feedback from previous simulations, include cross-entropy sampling [32] and Bayesian optimization [111]. The former selects samples and updates the prior distribution by minimizing cross-entropy; the latter updates the prior from the posterior over a user-provided objective function, e.g. the satisfaction level of a specification or the loss of an analyzed model. Details of the sampling techniques can be found in Chapter 6 and Section 5.5 in Chapter 5.

Specification Monitor: Trajectories generated by the simulator for a specific environment $e \in \mathcal{E}$, $\xi_{\mathcal{S}})\cdot; e)$ are evaluated by the monitor, which produces a score for a given property or objective function. VERIFAI supports monitoring MTL properties using the `py-metric-temporal-logic` [154] package, including both the Boolean and quantitative semantics of MTL. As mentioned above, the user can also specify a custom monitor as a Python function. The result of the monitor is used as feedback by the active search procedures to direct the sampling (search) towards falsifying scenarios. The result is also stored in the error table $\mathbb{E}$ which is used for further analysis and feedback to the user.

Error Table analysis: Counterexamples are stored in a data structure called the error table $\mathbb{E}$, whose rows are counterexamples and columns are abstract features (Refer to Section 5.6 in Chapter 5). The error table can be used offline to debug (explain) the generated counterexamples or online to drive the sampler towards particular areas of the abstract feature space. VERIFAI provides different techniques for error table analysis depending on the end use (e.g., counter-example analysis or data set augmentation), including principal component analysis (PCA) and clustering using $k-$means for ordered feature domains and subsets of the most recurrent values for unordered domains (see Section 5.6 in Chapter 5 for further details).

The communication between VERIFAI and the simulator is implemented in a client-server fashion using IPv4 sockets, where VERIFAI sends environment configurations $ein\mathcal{E}$ to the simulator which then returns trajectories (traces) $\xi_{\mathcal{S}}(\cdot; e)$. This architecture allows easy interfacing to a simulator and even with multiple simulators at the same time.

8.3 Features and Evaluation

This section illustrates the main features of VERIFAI through case studies in Figure 8.1 demonstrating its various use cases and simulator interfaces. VERIFAI and the following experiments can be found at `https://github.com/BerkeleyLearnVerify/aerifai.git`

8.3.1 Falsification

VERIFAI offers a convenient way to debug systems through systematic testing. Given a model and a specification, the tool can use active sampling to automatically search for inputs driving the model towards a violation of the specification. Specifically, in this experiment,

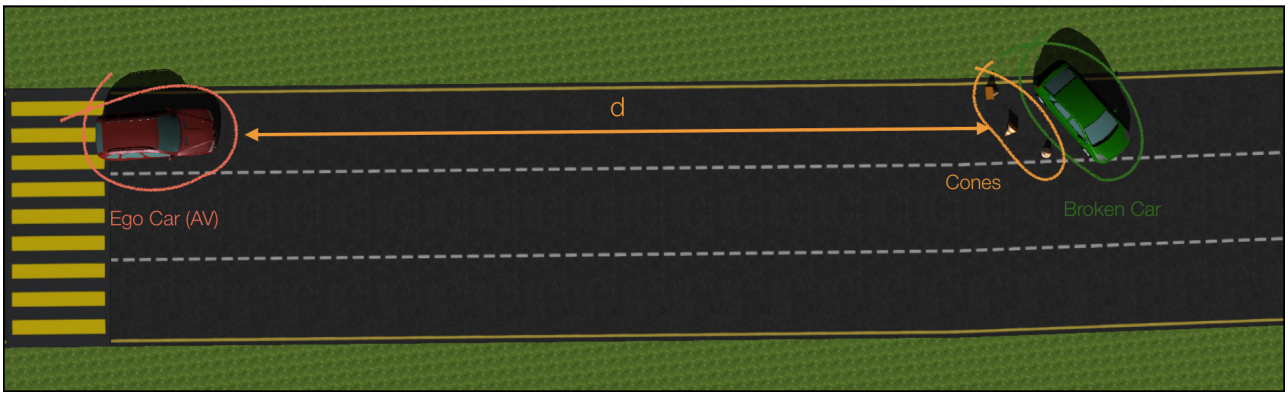


Figure 8.2. The red car and green car are the AV car and broken car respectively. The distance d captures the distance between the AV car and the cones. The AV car has to safely maneuver around the broken or disabled car.

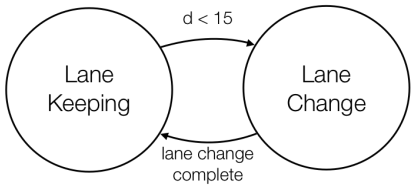


Figure 8.3. Hybrid controller for Av to safely maneuver around the broken car.

we consider an autonomous car (AV) simulated with the robotics simulator Webots [140]. For the experiments reported here, we used Webots 2018 which is commercial software. We use the cross-entropy sampler since it can handle both continuous and categorical features in abstract feature space.

We falsify the controller of an AV which is responsible for safely maneuvering around a disabled car and traffic cones which are blocking the road shown in Figure 8.2.

Control Design: We implemented a hybrid controller (Figure 8.3) which relies on perception modules for state estimation. Initially, the car is the *Lane Detection* mode where the controller relies on standard computer vision (non-ML) techniques to detect the center of the lane. It then uses a simple feedback controller to follow the lane. At the same time, a neural network (based on squeezeDet [160]) estimates the distance to the cones d. When the distance drops below 15 meters, the controller changes mode to *Lane Change* where it relies on pre-designed maneuver to change lane. After successfully changing lane, it switches back to *Lane Keeping* mode.

Safety Specification: The correctness of the AV is characterized by an MTL formula requiring the vehicle to maintain a minimum distance from the traffic cones and avoid overshoot while changing lanes.

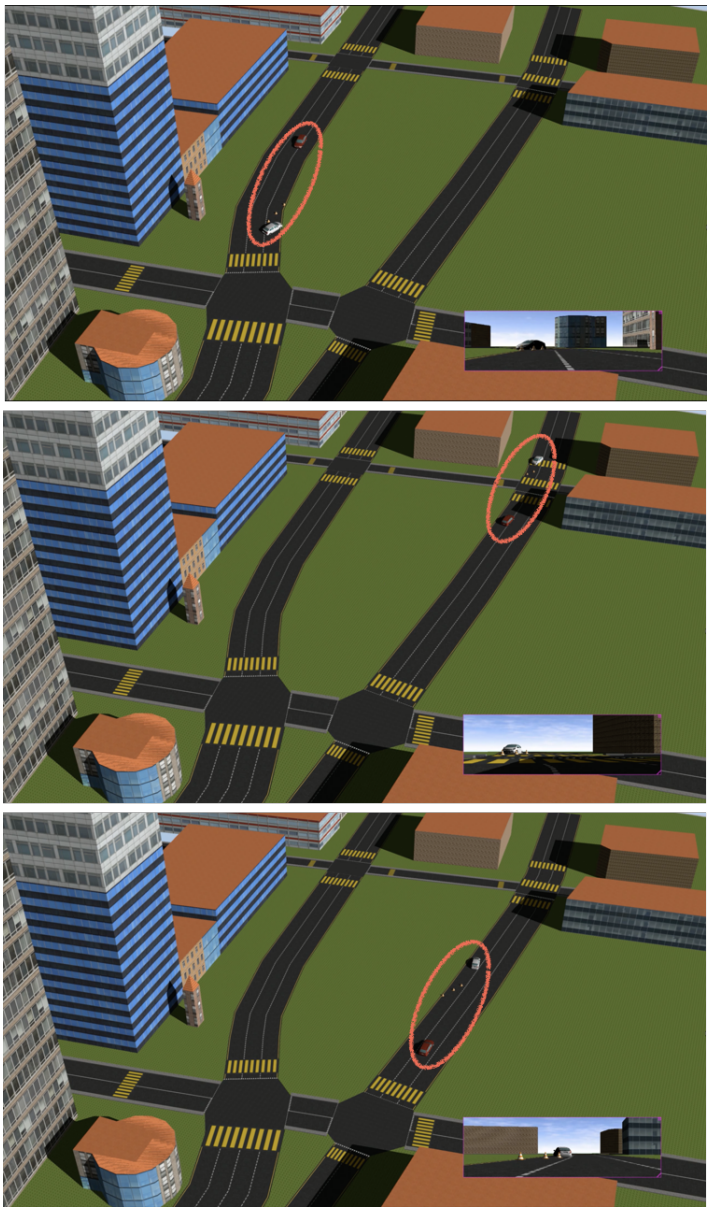


Figure 8.4. Scenes generated by SCENIC. The orange oval marks the placement of the AV car, broken car and cones.

We first use SCENIC to generate initial scene. We provide SCENIC a map of the city and generate scenes similar to that shown in 8.2. Figure 8.4 shows three scenes which were generated automatically by SCENIC in VERIFAI.

We picked the third scene generted by SCENIC in Figure 8.4 to be the initial scene for the falsifier. The task of the falsifier is to find small perturbations of the initial scene which cause the vehicle to violate this specification. We allowed perturbations of the initial positions and orientations of all objects (AV car, broken car and cones), the color of the broken car, and

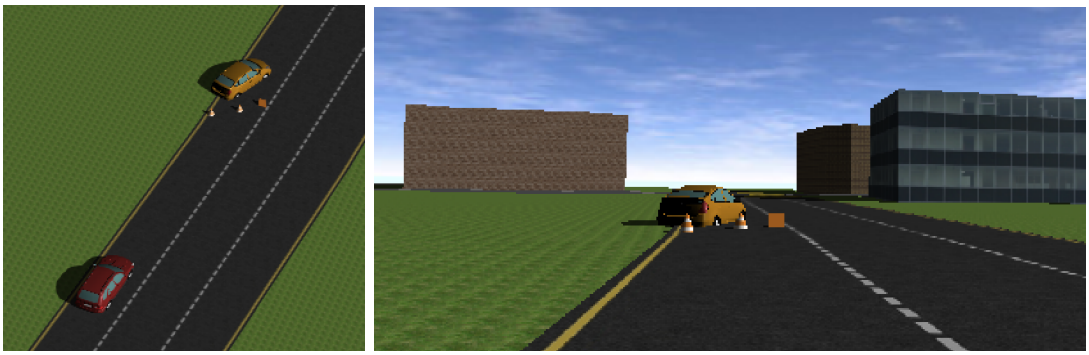


Figure 8.5. A falsifying scene automatically discovered by VERIFAI. The neural network misclassifies the traffic cones because of the orange vehicle in the background, leading to a crash. Left: bird's-eye view. Right: dash-cam view, as processed by the neural network.

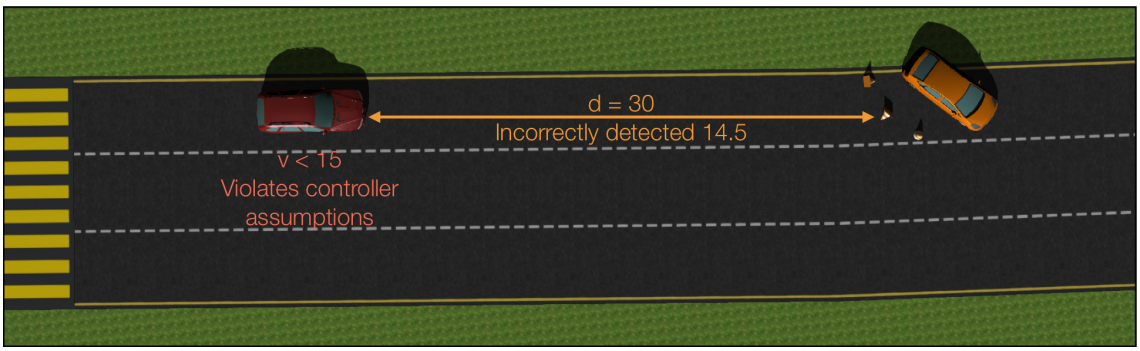


Figure 8.6. The NN incorrectly detects the orange car as an orange cone. Hence the distance d is incorrectly estimated to be $14.5m$ even when the distance is $30m$.

the cruising speed and reaction time (delay between switching from *Lane Keeping* and *Lane Change* mode in the hybrid controller in Figure 8.3) of the AV car.The task of the falsifier is to find small perturbations of the initial scene (generated by SCENIC) which cause the vehicle to violate this specification. Our experiments showed that the cross-entropy sampler driven by the robustness of the MTL specification can efficiently discover scenes that confuse the controller and yield faulty behavior. The counter-examples are stored in the error table $\mathbb{E}$. On analyzing the error table, we observe the main cause of failure was a combination of low cruising-speed and large reaction time. This implies, lane change at a lower cruising speed after a larger delay causes the AV car to crash into the cones.

We also observed a particularly interesting failure case shown in Figure 8.5.

Figure 8.6 provides the explanation. The neural network detected the orange car instead of the traffic cones, causing the lane change to be initiated too early. The pre-designed lane change maneuver assumes that the speed of the AV car is atleast $15m/s$. However, in every simulation the AV car starts from rest, and the early detection leads the lane change maneuver to happen at a lower speed. As a result, the controller performed only an incomplete lane change, leading to a crash.

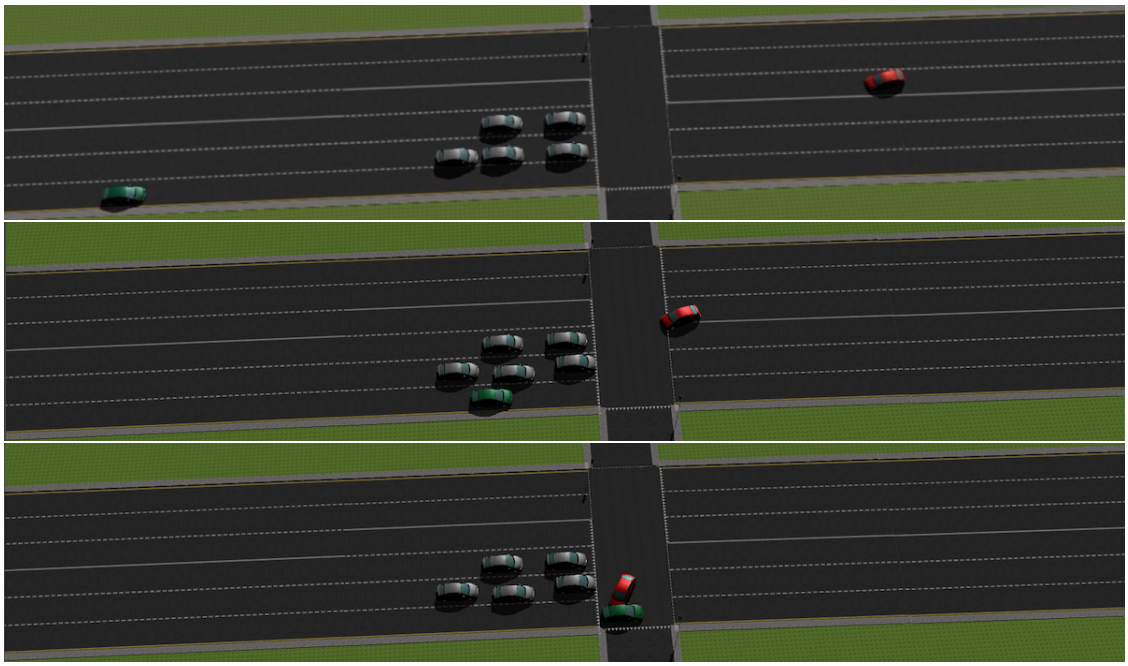


Figure 8.7. Accident scenario breakup: 1) initial scene sampled from the program; 2) the red car begins its turn, unable to see the green car; 3) the resulting collision.

As a result, we were able to go back and update the controller. Moreover, we used the dash-cam images to re-train the network as proposed in Chapter 5.

8.3.2 Fuzz-Testing

VerifAI can also perform model-based fuzz testing, exploring random variations of a scenario guided by formal constraints. In this experiment, we used VerifAI to simulate variations on an actual accident involving an AV [19]. The AV, proceeding straight through an intersection, was hit by a human turning left. Neither car was able to see the other because of two lanes of stopped traffic. We write a Scenic program to reproduce the accident in Webots which allows variations in the initial positions of the cars. We then ran simulations from random initial conditions sampled from the program, with the turning car using a controller trying to follow the ideal left-turn trajectory computed from OpenStreetMap data using the Intelligent Intersections Toolbox [71]. Figure 8.7 shows a accident scenario sampled by Scenic. The car going straight used a controller which either maintained a constant velocity or began emergency breaking in response to a message from a simulated "smart intersection" warning about the turning car. By sampling variations on the initial conditions, we could determine how much advance notice is necessary for such a system to robustly avoid an accident.

Figure 8.8. This image generated by our renderer was misclassified by the NN. The network reported detecting only one car when there were two.

8.3.3 Data Augmentation and Error Table analysis

Data augmentation is the process of supplementing training sets with the goal of improving the performance of ML models. Typically, datasets are augmented with transformed versions of preexisting training examples. In Chapter 5, we showed that augmentation with counterexamples is also an effective method for model improvement.

VerifAI implements a counterexample-guided augmentation scheme $\mathcal{C}_{da}$ details in Chapter 5, where a oracle $\mathcal{O}_{\mathcal{C}_{da}}$ generates misclassified data points that are then used to augment the original training set. As described in Chapter 5, the user can choose among different sampling methods, with passive samplers suited to generating diverse sets of data points while active samplers can efficiently generate similar counterexamples. In addition to the counterexamples themselves, VerifAI also returns an error table aggregating information on the misclassifications that can be used to drive the retraining process. Figure 8.8 shows the rendering of a misclassified sample generated by our falsifier.

For our experiments, we implemented the renderer described in Section 5.4 of Chapter 5 that generates images of road scenarios and tested the quality of our augmentation scheme on the squeezeDet convolutional neural network [160], trained for classification. We adopted three techniques to select augmentation images: 1) randomly sampling from the error table, 2) selecting the top k-closest (similar) samples from the error table, and 3) using PCA analysis to generate new samples. For details on the renderer and the results of counterexample-driven augmentation, see Chapter 5. We show that incorporating the generated counterexamples during re-training improves the accuracy of the network.

8.3.4 Model Robustness and Hyper-parameter Tuning

In this experiment, we demonstrate how VerifAI can be used to tune test parameters and hyper-parameters of AI systems. For the following case studies, we use OpenAI Gym [17], a framework for experimenting with reinforcement learning algorithms.

First, we consider the problem of testing the robustness of a learned controller for a

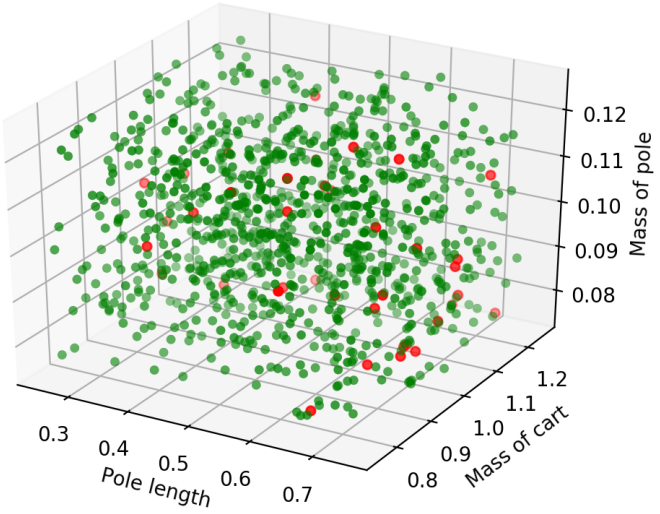


Figure 8.9. The green dots represent model parameters for which the cart-pole controller behaved correctly, while the red dots indicate specification violations. Out of 1000 randomly-sampled model parameters, the controller failed to satisfy the specification 38 times.

cart-pole, i.e., a cart that balances an inverted pendulum. We trained a neural network to control the cart-pole using Proximal Policy Optimization algorithms [132] with 100k training episodes. We then used VerifAI to test the robustness of the learned controller against the model parameters, varying the initial lateral position and rotation of the cart as well as the mass and length of the pole. Even for apparently robust controllers, VerifAI was able to discover configurations for which the cart-pole failed to self-balance. Fig. 8.9 shows 1000 iterations of the falsifier, where sampling was guided by the reward function used by OpenAI to train the controller. This function provides a negative reward if the cart moves more than 2.4 m or if at any time the angle maintained by the pole is greater than 12 degrees. For testing, we slightly modified these thresholds.

Finally, we used VerifAI to study the effects of hyper-parameters when training a neural network controller for a mountain car. In this case, the controller must learn to exploit momentum in order to climb a steep hill. Here, rather than searching for counterexamples, we look for a set of hyper-parameters under which the network *correctly* learns to control the car. Specifically, we explored the effects of using different training algorithms (from a discrete set of choices) and the size of the training set. We used the VerifAI falsifier to search the hyper-parameter space, guided again by the reward function provided by OpenAI Gym (here the distance from the goal position), but negated so that falsification implied finding a controller which successfully climbs the hill. In this way VerifAI built a table of effective hyper-parameters. PCA analysis then revealed which hyper-parameters the training process is most sensitive or robust to.

8.4 Conclusion

In this chapter we presented VerifAI, a software toolkit for the formal design and analysis of robotic systems that include AI and ML components. VerifAI is an open source toolkit, where one can implement and integrate any OGIS based framework for design and analysis of robotic systems. At the time of this thesis, Chapters 5 and 6 have been integrated into the toolkit.

Chapter 9

Conclusion and Future Work

Autonomous systems such as autonomous vehicles, medical robots, robotic manipulators are becoming a reality. For building such systems, there is an increasing dependence on artificial intelligence (AI) and machine learning (ML) since they are capable of inducing richer set of behaviors. However, since such systems are expected to operate near and interact with humans on a daily basis, guaranteeing their safety is a top priority. While classical control synthesis and formal verification techniques can provide strong safety guarantees, their applicability is quite limited for such systems in the presence of complex (potentially ML) sub-components.

In this thesis, we studied the key issues arising from ML components in the synthesis and verification pipeline ([135]) and proposed frameworks that can overcome them. We took inspiration from the oracle-guided inductive synthesis (OGIS) paradigm introduced in [80], to reformulate each step as an instance of the OGIS framework which consists of a learner which attempts to learn or synthesize concept by querying an oracle. This decoupling, allows us to design simpler solution approaches, while enjoying the guarantees provided by this framework.

A strong message that we want to convey in this thesis is that the stages of the design pipeline are highly coupled. Synthesis (design) and verification (analysis) have to have a strong feedback among them to ensure the safety of the overall system.

We utilized this framework to study five parts of the design and analysis pipeline. In Chapter 3, we developed a sound framework that can synthesize robust control strategies from high level specifications. In Chapter 4, we used the results of Chapter 3 to diagnose and repair high level safety specifications. This showed us that specification design is tightly coupled with the synthesis process. In Chapter 5, we developed a counter-example guided data-augmentation algorithm which further exemplifies the need of analysis (testing) in data set design, and hence model synthesis. In Chapter 6, we took a closer look into simulation-guided falsification. We studied the assumptions under which the proposed framework can provide us verification certificates. In Chapter 7, we studied how one can transfer model level verification guarantees to the real system, and bridge the gap. To this end, we developed a new specification-centric simulation metric to synthesize controllers for the real system based

on the model, and proved that this was the least conservative of all such metrics. Finally, in Chapter 8, we presented VerifAI a python toolkit that incorporates much of the work presented in this thesis to design and analyze AI-based systems.

The work presented in this thesis just scratches the surface of design and analysis pipeline for autonomous systems. There is still a long way to go truly design a verifiably safe autonomous system. Moving forward, we see the work in this thesis going in multiple directions.

The key challenge in adopting the algorithms proposed in this thesis to real systems is dealing with larger input or state or environment dimensions. This is because optimization and sampling based algorithms perform poorly with increasing dimensions. One way of overcoming this, is to decompose the larger problem to smaller sub problems. Once we design them independently, we could then compose them together to design the larger system. Another way of solving this is to design our algorithms to run in a distributed manner. Both the directions are promising and would be interesting next steps.

We would like to extend VerifAI to handle richer set of specifications [46, 136], interface to other state of the art robotic simulators; and develop novel search algorithms that can effectively handle higher dimension environments. We would also like to incorporate more error analysis techniques to provide richer feedback.

While we have that counter-example guided data augmentation technique works for models trained on synthetic data, we would like to explore how we can extend these results to real images. Domain adaptation [31] and domain randomization [148] seem to be possible solutions to this.

Finally, we have shown that our algorithms work in the model or simulation world. We would like to try our algorithms on the real world examples as well. We propose to incorporate the framework presented in Chapter 7 into the frameworks proposed in Chapters 3, 4 and 5 to test their real world applicability.

Bibliography

[1] Alessandro Abate. "A contractivity approach for probabilistic bisimulations of diffusion processes". In: *Proceedings of the 48th IEEE Conference on Decision and Control*. 2009.

[2] Alessandro Abate and Maria Prandini. "Approximate abstractions of stochastic systems: A randomized method". In: *Conference on Decision and Control and European Control Conference*. 2011.

[3] Rajeev Alur and Thomas A. Henzinger. "Logics and models of real time: A survey". In: *Real-Time: Theory in Practice*. 1992.

[4] Rajeev Alur, Salar Moarref, and Ufuk Topcu. "Counter-strategy guided refinement of GR(1) temporal logic specifications". In: *Formal Methods in Computer-Aided Design*. 2013.

[5] Rajeev Alur et al. "Discrete abstractions of hybrid systems". In: *Proceedings of the IEEE*. 2000.

[6] Rajeev Alur et al. "Syntax-Guided Synthesis". In: *Proceedings of the IEEE International Conference on Formal Methods in Computer-Aided Design (FMCAD)*. 2013.

[7] Mohamed Aly. "Real time detection of lane markers in urban streets". In: *IEEE Intelligent Vehicles Symposium*. 2008.

[8] Yashwanth Annpureddy et al. "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems". In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2011.

[9] Christel Baier and Joost-Pieter Katoen. *Principles of model checking*. MIT press, 2008.

[10] Somil Bansal et al. "Goal-Driven Dynamics Learning via Bayesian Optimization". In: *Conference on Decision and Control*. 2017.

[11] Somil Bansal et al. "Hamilton-Jacobi reachability: A brief overview and recent advances". In: *IEEE 56th Annual Conference on Decision and Control (CDC)*. 2017.

[12] Somil Bansal et al. "Learning quadrotor dynamics using neural network for flight control". In: *Conference on Decision and Control*. 2016.

[13] Clark Barrett and Cesare Tinelli. "Satisfiability modulo theories". In: *Handbook of Model Checking*. Springer, 2018.

[14] Alberto Bemporad and Manfred Morari. "Control of systems integrating logic, dynamics, and constraints". In: 1999.

[15] Alberto Bemporad and Manfred Morari. "Robust model predictive control: A survey". In: *Robustness in identification and control*. Springer, 1999.

[16] Felix Berkenkamp, Andreas Krause, and Angela P. Schoellig. *Bayesian optimization with safety constraints: safe and automatic parameter tuning in robotics*. 2016. eprint: `arXiv:1602.04450`.

[17] Greg Brockman et al. *OpenAI Gym*. 2016. eprint: `arXiv:1606.01540`.

[18] Manuela L. Bujorianu, John Lygeros, and Marius C. Bujorianu. "Bisimulation for general stochastic hybrid systems". In: *International Workshop on Hybrid Systems: Computation and Control*. 2005.

[19] Mike Butler. "Uber's Tempe accident raises questions of self-driving safety". In: *East Valley Tribune*. 2017.

[20] Giuseppe C. Calafiore and Marco C. Campi. "The scenario approach to robust control design". In: *IEEE Transactions on Automatic Control*. 2006.

[21] Giuseppe C. Calafiore and Marco C. Campi. "Uncertain convex programs: randomized solutions and confidence levels". In: *Mathematical Programming*. 2005.

[22] Marco C. Campi and Simone Garatti. "A sampling-and-discarding approach to chance-constrained optimization: feasibility and optimality". In: *Journal of Optimization Theory and Applications*. 2011.

[23] Marco C. Campi, Simone Garatti, and Maria Prandini. "The scenario approach for systems and control design". In: *Annual Reviews in Control*. 2009.

[24] Pratik Chaudhari, Tichakorn Wongpiromsarny, and Emilio Frazzoli. "Incremental minimum-violation control synthesis for robots interacting with external agents". In: *American Control Conference (ACC)*. 2014.

[25] Xin Chen, Sriram Sankaranarayanan, and Erika Abrahám. "Flow* 1.2: More Effective to Play with Hybrid Systems". In:

[26] John W Chinneck and Erik W Dravnieks. "Locating minimal infeasible constraint sets in linear programs". In: INFORMS, 1991.

[27] Sayak Ray Chowdhury and Aditya Gopalan. "On kernelized multi-armed bandits". In: *ICML*. 2017.

[28] Dan Ciregan, Ueli Meier, and Jürgen Schmidhuber. "Multi-column deep neural networks for image classification". In: *IEEE conference on Computer vision and pattern recognition (CVPR)*. 2012.

[29] Dan C Cireşan et al. "High-performance neural networks for visual object classification". In: *arXiv:1102.0183*. 2011.

[30] Edmund M. Clarke Jr., Orna Grumberg, and Doron A. Peled. *Model Checking*. MIT Press, 1999.

[31] Koby Crammer, Michael Kearns, and Jennifer Wortman. "Learning from multiple sources". In: *Journal of Machine Learning Research*. 2008.

[32] Pieter-Tjerk De Boer et al. "A tutorial on the cross-entropy method". In: *Annals of operations research*. 2005.

[33] Leonardo De Moura and Nikolaj Bjørner. "Z3: An efficient SMT solver". In: Springer, 2008.

[34] Thomas Dean and Robert Givan. "Model minimization in Markov decision processes". In: *AAAI/IAAI*. 1997.

[35] Josée Desharnais, Abbas Edalat, and Prakash Panangaden. "Bisimulation for labelled Markov processes". In: *Information and Computation*. 2002.

[36] Jyotirmoy V. Deshmukh et al. "Testing Cyber-Physical Systems through Bayesian Optimization". In: 2017.

[37] Jyotirmoy Deshmukh et al. "Stochastic local search for falsification of hybrid systems". In: *International Symposium on Automated Technology for Verification and Analysis*. 2015.

[38] Jerry Ding et al. "Reachability-based synthesis of feedback policies for motion planning under bounded disturbances". In: *IEEE International Conference on Robotics and Automation*. 2011.

[39] Alexandre Donzé. "Breach, a toolbox for verification and parameter synthesis of hybrid systems". In: *International Conference on Computer Aided Verification*. 2010.

[40] Alexandre Donzé, Thomas Ferrère, and Oded Maler. "Efficient robust monitoring for STL". In: *Computer Aided Verification*. 2013.

[41] Alexandre Donzé and Oded Maler. "Robust Satisfaction of Temporal Logic over Real-Valued Signals". In: *International Conference on Formal Modeling and Analysis of Timed Systems*. 2010.

[42] Alexandre Donzé et al. "On temporal logic and signal processing". In: *Automated Technology for Verification and Analysis*. 2012.

[43] Alexey Dosovitskiy et al. "CARLA: An Open Urban Driving Simulator". In: *Proceedings of the 1st Annual Conference on Robot Learning*. 2017.

[44] Tommaso Dreossi, Alexandre Donze, and Sanjit A. Seshia. "Compositional Falsification of Cyber-Physical Systems with Machine Learning Components". In: *Proceedings of the NASA Formal Methods Conference (NFM)*. 2017.

[45] Tommaso Dreossi, Somesh Jha, and Sanjit A. Seshia. "Semantic Adversarial Deep Learning". In: *30th International Conference on Computer Aided Verification (CAV)*. 2018.

[46] Tommaso Dreossi et al. "A Formalization of Robustness for Deep Neural Networks". In: *AAAI Spring Symposium on Verification of Neural Networks*. 2019.

[47] Tommaso Dreossi et al. "Counterexample-Guided Data Augmentation". In: *27th International Joint Conference on Artificial Intelligence (IJCAI)*. 2018.

[48] Tommaso Dreossi et al. "Systematic Testing of Convolutional Neural Networks for Autonomous Driving". In: *ICML Workshop on Reliable Machine Learning in the Wild (RMLW)*. 2017.

[49] Tommaso Dreossi et al. "VerifAI: A Toolkit for the Formal Design and Analysis of Artificial Intelligence-Based Systems". In: *Computer Aided Verification*. Springer International Publishing, 2019.

[50] Parasara Sridhar Duggirala et al. "C2E2: A Verification Tool for Stateflow Models". In: *Tools and Algorithms for the Construction and Analysis of Systems*. 2015.

[51] David A van Dyk and Xiao-Li Meng. "The Art of Data Augmentation". In: *Journal of Computational and Graphical Statistics*. 2001.

[52] Samira S Farahani, Vasumathi Raman, and Richard M Murray. "Robust model predictive control for signal temporal logic synthesis". In: 2015.

[53] Thomas Ferrère, Oded Maler, and Dejan Nickovic. "Trace Diagnostics Using Temporal Implicants". In: *Proc. Int. Symp. Automated Technology for Verification and Analysis*. 2015.

[54] Daniel E Finkel. "DIRECT optimization algorithm user guide". In: 2003.

[55] Jaime F. Fisac et al. "A general safety framework for learning-based control in uncertain robotic systems". In: *arXiv:1705.01292*. 2017.

[56] Daniel J Fremont et al. "Scenic: a language for scenario specification and scene generation". In: *Proceedings of the 40th ACM SIGPLAN Conference on Programming Language Design and Implementation*. 2019.

[57] Simone Garatti and Maria Prandini. "A simulation-based approach to the approximation of stochastic hybrid systems". In: *Analysis and design of hybrid systems*. 2012.

[58] Carlos E Garcia, David M Prett, and Manfred Morari. "Model predictive control: theory and practice—a survey". In: Elsevier, 1989.

[59] Marco Gario and Andrea Micheli. "PySMT: A solver-agnostic library for fast prototyping of SMT-based algorithms". In: *13th International Workshop on Satisfiability Modulo Theories*. 2015.

[60] Timon Gehr et al. "AI2: Safety and Robustness Certification of Neural Networks with Abstract Interpretation". In: *Security and Privacy (SP), 2018 IEEE Symposium on*. 2018.

[61] Andreas Geiger et al. "Vision meets robotics: The KITTI dataset". In: *The International Journal of Robotics Research*. 2013.

[62] Michel Gevers et al. "Model Validation for Control and Controller Validation in a Prediction Error Identification framework-Part I: Theory". In: *Automatica*. 2003.

[63] Shromona Ghosh et al. "A new simulation metric to determine safe environments and controllers for systems with unknown dynamics". In: *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*. 2019.

[64] Shromona Ghosh et al. "Diagnosis and Repair for Synthesis from Signal Temporal Logic Specifications". In: *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*. 2016.

[65] Shromona Ghosh et al. "Verifying Controllers against Adversarial Examples with Bayesian Optimization". In: *2018 IEEE International Conference on Robotics and Automation (ICRA)*. 2018.

[66] Antoine Girard and George J. Pappas. "Approximate bisimulation relations for constrained linear systems". In: *Automatica*. 2007.

[67] Antoine Girard and George J. Pappas. "Approximate Bisimulation: A bridge between computer science and control theory". In: *European Journal of Control*. 2011.

[68] Antoine Girard, Giordano Pola, and Paulo Tabuada. "Approximately bisimilar symbolic models for incrementally stable switched systems". In: *IEEE Transactions on Automatic Control*. 2010.

[69] Patrice Godefroid, Adam Kiezun, and Michael Y Levin. "Grammar-Based Whitebox Fuzzing". In: *ACM SIGPLAN Notices*. ACM. 2008.

[70] Ian Goodfellow et al. "Generative adversarial nets". In: *Advances in neural information processing systems*. 2014.

[71] Offer Grembek et al. *Making intersections safer with I2V communication*. 2019.

[72] *Gurobi Optimizer*. `http://www.gurobi.com/`.

[73] Hadas Kress-Gazit, Gerogios E. Fainekos, and George J. Pappas. "Temporal Logic based Reactive Mission and Motion Planning". In: *IEEE Transactions on Robotics*. 2009.

[74] John H Halton. "On the efficiency of certain quasi-random sequences of points in evaluating multi-dimensional integrals". In: *Numerische Mathematik*. 1960.

[75] Nikolaus Hansen. "The CMA evolution strategy: A tutorial". In: 2016.

[76] Hakkan Hjalmarsson and Lennart Ljung. "Estimating model variance in the case of undermodeling". In: *IEEE Transactions on Automatic Control*. 1992.

[77] Haomiao Huang et al. "Aerodynamics and control of autonomous quadrotor helicopters in aggressive maneuvering". In: *IEEE international conference on robotics and automation (ICRA)*. 2009.

[78] Anil K Jain, Richard C Dubes, et al. *Algorithms for clustering data*. Prentice hall Englewood Cliffs, NJ, 1988.

[79] Kristian Jensen, Joao Cardoso, and Nikolaus Sonnenschein. "Optlang: An algebraic modeling language for mathematical optimization". In: 2016.

[80] Susmit Jha and Sanjit A. Seshia. "A theory of formal synthesis via inductive learning". In: *Acta Informatica*. Springer, 2017.

[81] Xiaoqing Jin et al. "Mining requirements from closed-loop control models". In: 2015.

[82] A. Agung Julius and George J. Pappas. "Approximations of stochastic hybrid systems". In: *IEEE Transactions on Automatic Control*. 2009.

[83] Gregory Kahn et al. "Uncertainty-Aware Reinforcement Learning for Collision Avoidance". In: vol. abs/1702.01182. 2017.

[84] Joost-Pieter Katoen et al. "Bisimulation minimisation mostly speeds up probabilistic model checking". In: *International Conference on tools and algorithms for the construction and analysis of systems*. 2007.

[85] Eric C Kerrigan and Jan M Maciejowski. "Soft constraints and exact penalty functions in model predictive control". In: *Control Conference, Cambridge*. 2000.

[86] Kangjin Kim, Georgios Fainekos, and Sriram Sankaranarayanan. "On the minimal revision problem of specification automata". In: 2015.

[87] Hui Kong, Jean-Yves Audibert, and Jean Ponce. "Vanishing point detection for road detection". In: *IEEE Conference on Computer Vision and Pattern Recognition*. 2009.

[88] Zhaodan Kong et al. "Temporal logic inference for classification and prediction from data". In: *Proceedings of the 17th international conference on Hybrid systems: computation and control*. 2014.

[89] Robert Könighofer, Georg Hofferek, and Roderick Bloem. "Debugging formal specifications: a practical approach using model-based diagnosis and counterstrategies". In: 2013.

[90] Ron Koymans. "Specifying real-time properties with metric temporal logic". In: 1990.

[91] Alex Krizhevsky, Ilya Sutskever, and Geoffrey E Hinton. "Imagenet classification with deep convolutional neural networks". In: *Advances in neural information processing systems*. 2012.

[92] H. Kwakernaak and R. Sivan. *Linear optimal control systems*. 1972.

[93] Morteza Lahijanian et al. "This Time the Robot Settles for a Cost: A Quantitative Approach to Temporal Logic Planning with Partial Satisfaction". In: *The Twenty-Ninth AAAI Conference (AAAI-15)*. 2015.

[94] Kim G. Larsen and Arne Skou. "Bisimulation through probabilistic testing". In: *Information and computation*. 1991.

[95] Ian Lenz, Ross A. Knepper, and Ashutosh Saxena. "DeepMPC: Learning Deep Latent Features for Model Predictive Control". In: *Robotics: Science and Systems*. 2015.

[96] Sergey Levine et al. "End-to-end training of deep visuomotor policies". In: *The Journal of Machine Learning Research*. 2016.

[97] Wenchao Li, Lili Dworkin, and Sanjit A. Seshia. "Mining Assumptions for Synthesis". In: *ACM/IEEE Int. Conf. Formal Methods and Models for Codesign*. 2011.

[98] Wenchao Li et al. "Synthesis for Human-in-the-Loop Control Systems". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*. 2014.

[99] Xiaodan Liang et al. "Recurrent Topic-Transition GAN for Visual Paragraph Generation". In: *arXiv:1703.07022*. 2017.

[100] Timothy P. Lillicrap et al. *Continuous control with deep reinforcement learning*. 2015. eprint: `arXiv/1509.02971`.

[101] Lennart Ljung. *System identification: theory for the user*. 1987.

[102] J. Löfberg. "YALMIP: A Toolbox for Modeling and Optimization in MATLAB". In: *Proceedings of the CACSD Conference*. 2004. URL: `http://users.isy.liu.se/johanl/yalmip`.

[103] Andrew O. Makhorin. *GLPK*. `https://www.gnu.org/software/glpk/`. 2018.

[104] Oded Maler and Dejan Nickovic. "Monitoring temporal properties of continuous signals". In: *FORMATS/FTRTFT*. Springer. 2004.

[105] Marco Marchesi. "Megapixel Size Image Creation using Generative Adversarial Networks". In: *arXiv:1706.00082*. 2017.

[106] Ian M. Mitchell. "A Toolbox of Level Set Methods". In: 2004.

[107] Ian M. Mitchell. "Application of level set ethods to control and reachability problems in continuous and hybrid systems". In: *Ph.D. Dissertation, Stanford*. 2002.

[108] Ian M. Mitchell. "The flexible, extensible and efficient toolbox of level set methods". In: *Journal of Scientific Computing*. 2008.

[109] Ian M. Mitchell and Claire J Tomlin. "Level set methods for computation in hybrid systems". In: *International Workshop on Hybrid Systems: Computation and Control*. Springer. 2000.

[110] Stefan Mitsch, Khalil Ghorbal, and André Platzer. "On Provably Safe Obstacle Avoidance for Autonomous Robotic Ground Vehicles". In: *Robotics: Science and Systems*. 2013.

[111] Jonas Mockus. *Bayesian approach to global optimization: theory and applications*. Springer Science & Business Media, 2012.

[112] Manfred Morari and Jay H Lee. "Model predictive control: past, present and future". In: Elsevier, 1999.

[113] Galen E Mullins, Paul G Stankiewicz, and Satyandra K Gupta. "Automated generation of diverse and challenging scenarios for test and evaluation of autonomous vehicles". In: *2017 IEEE International Conference on Robotics and Automation (ICRA)*. 2017.

[114] Harald Niederreiter. "Low-discrepancy and low- sequences". In: *Journal of number theory*. 1988.

[115] Yael Niv et al. "Evolution of reinforcement learning in uncertain environments: A simple explanation for complex foraging behaviors". In: Sage Publications, 2002.

[116] Pierluigi Nuzzo et al. "A Platform-Based Design Methodology with Contracts and Related Tools for the Design of Cyber-Physical Systems". In: 2015.

[117] Pierluigi Nuzzo et al. "CalCS: SMT Solving for Non-linear Convex Constraints". In: *IEEE Int. Conf. Formal Methods in Computer-Aided Design*. 2010.

[118] Piurluigi Nuzzo et al. "A Contract-Based Methodology for Aircraft Electric Power System Design". In: 2014.

[119] *Open AI Baselines*. `https://github.com/openai/baselines`.

[120] *Open AI Gym Environments*. `https://gym.openai.com/envs`.

[121] Alessandro V. Papadopoulos and Maria Prandini. "Model reduction of switched affine systems". In: *Automatica*. 2016.

[122] Francisco Penedo Álvarez, Cristian Ioan Vasile, and Calin Belta. "Language-Guided Sampling-based Planning using Temporal Relaxation". In: *Workshop on the Algorithmic Foundations of Robotics (WAFR)*. 2016.

[123] Amir Pnueli. "The Temporal Logic of Programs". In: *Proceedings of the 18th Annual Symposium on Foundations of Computer Science*. 1977.

[124] Giordano Pola, Antoine Girard, and Paulo Tabuada. "Approximately bisimilar symbolic models for nonlinear control systems". In: *Automatica*. 2008.

[125] Pascal Poupart and Nikos Vlassis. "Model-based Bayesian reinforcement learning in partially observable domains". In: *Proc Int. Symp. on Artificial Intelligence and Mathematics,* 2008.

[126] Vasumathi Raman and Hadas Kress-Gazit. "Explaining Impossible High-Level Robot Behaviors". In: 2013.

[127] Vasumathi Raman et al. "Model predictive control with signal temporal logic specifications". In: *53rd IEEE Conference on Decision and Control*. IEEE.

[128] Vasumathi Raman et al. "Reactive Synthesis from Signal Temporal Logic Specifications". In: *18th International Conference on Hybrid Systems: Computation and Control*. 2015.

[129] CE. Rasmussen and CKI. Williams. *Gaussian Processes for Machine Learning*. MIT Press.

[130] Stuart Russell et al. "Letter to the Editor: Research Priorities for Robust and Beneficial Artificial Intelligence: An Open Letter". In: *AI Magazine*. 2015.

[131] Shankar Sastry and Marc Bodson. *Adaptive Control: Stability, Convergence, and Robustness*. Prentice-Hall, Inc., 1989.

[132] John Schulman et al. *Proximal Policy Optimization Algorithms*. 2017. eprint: `arXiv:1707.06347`.

[133] Viktor Schuppan. "Towards a Notion of Unsatisfiable Cores for LTL". In: *Fundamentals of Software Engineering*. 2009.

[134] Pierre OM Scokaert and James B Rawlings. "Feasibility issues in linear model predictive control". In: 1999.

[135] Sanjit A. Seshia, Dorsa Sadigh, and S. Shankar Sastry. "Towards Verified Artificial Intelligence". In: *arXiv:1606.08514*. 2016.

[136] Sanjit A. Seshia et al. "Formal Specification for Deep Neural Networks". In: *16th International Symposium on Automated Technology for Verification and Analysis (ATVA)*. Springer. 2018.

[137] Abhinav Shrivastava, Abhinav Gupta, and Ross Girshick. "Training region-based object detectors with online hard example mining". In: *IEEE conference on Computer vision and pattern recognition (CVPR)*. 2016.

[138] Patrice Y Simard, David Steinkraus, John C Platt, et al. "Best practices for convolutional neural networks applied to visual document analysis." In: *ICDAR*. 2003.

[139] Ilya M Sobol. "Uniformly distributed sequences with an additional uniform property". In: *USSR Computational Mathematics and Mathematical Physics*. 1976.

[140] Commercial Mobile Robot Simulation Software. *Webots*. Ed. by Cyberbotics Ltd. `http://www.cyberbotics.com`. 1998.

[141] Armando Solar-Lezama et al. "Combinatorial sketching for finite programs". In: 2006.

[142] Niranjan Srinivas et al. "Gaussian process optimization in the bandit setting: no regret and experimental design". In: 2012.

[143] Ingo Steinwart and Andreas Christmann. *Support vector machines*. Springer Science & Business Media, 2008.

[144] Robert F. Stengel. *Stochastic Optimal Control: Theory and Application*. John Wiley & Sons, Inc., 1986.

[145] Stefan Strubbe and Arjan Van Der Schaft. "Bisimulation for communicating piecewise deterministic Markov processes (CPDPs)". In: *International Workshop on Hybrid Systems: Computation and Control*. 2005.

[146] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[147] Paulo Tabuada. *Verification and Control of Hybrid Systems: A Symbolic Approach*. Springer Science & Business Media, 2009.

[148] Josh Tobin et al. "Domain randomization for transferring deep neural networks from simulation to the real world". In: *IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*. 2017.

[149] Emanuel Todorov. "Optimal control theory". In: *Bayesian brain: probabilistic approaches to neural coding*. MIT Press Cambridge (Massachusetts).

[150] Claire J. Tomlin, John Lygeros, and Shankar Sastry. "A game theoretic approach to controller design for hybrid systems". In: *Proceedings of the IEEE*. 2000.

[151] Claire J. Tomlin, George J. Pappas, and Shankar Sastry. "Conflict resolution for air traffic management: A study in multiagent hybrid systems". In: *IEEE Transactions on automatic control*. 1998.

[152] Jana Tumova et al. "Minimum-violation LTL planning with conflicting specifications". In: *American Control Conference, ACC*. 2013.

[153] Marcell Vazquez-Chanlatte. *MagnumSTL*. `https://github.com/mvcisback/magnumSTL.git`. 2018.

[154] Marcell Vazquez-Chanlatte. *mvcisback/py-metric-temporal-logic: v0.1.1*. 2019. URL: `https://doi.org/10.5281/zenodo.2548862`.

[155] Marcell Vazquez-Chanlatte et al. "Generating dominant strategies for continuous two-player zero-sum games". In: *IFAC Conference on Design and Analysis of Hybrid Systems (ADHS)*. 2018.

[156] Ziyu Wang et al. "Bayesian Optimization in High Dimensions via Random Embeddings." In: *IJCAI*. 2013.

[157] Matthew Wicker, Xiaowei Huang, and Marta Kwiatkowska. "Feature-Guided Black-Box Safety Testing of Deep Neural Networks". In: *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*. Springer. 2018.

[158] Sebastien C Wong et al. "Understanding data augmentation for classification: when to warp?" In: *IEEE International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. 2016.

[159] Tichakorn Wongpiromsarn et al. "TuLiP: A Software Toolbox for Receding Horizon Temporal Logic Planning". In: *International Workshop on Hybrid Systems: Computation and Control*. 2011.

[160] Bichen Wu et al. "SqueezeDet: Unified, Small, Low Power Fully Convolutional Neural Networks for Real-Time Object Detection for Autonomous Driving". In: *arXiv:1612.01051*. 2016.

[161] *X-Plane 11 Flight Simulator*. `https://www.x-plane.com`.

[162] Yan Xu et al. "Improved relation classification by deep recurrent neural networks with data augmentation". In: *arXiv:1601.03651*. 2016.

[163] Aditya Zutshi et al. "Falsification of safety properties for closed loop control systems". In: *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*. 2015.