Formal Requirements-Driven Analysis of Cyber Physical Systems

by

Bardh Hoxha

A Dissertation Presented in Partial Fulfillment
of the Requirements for the Degree
Doctor of Philosophy

Approved May 2017 by the
Graduate Supervisory Committee:

Georgios Fainekos, Chair
Hessam Sarjoughian
Ross Maciejewski
Heni Ben Amor

ARIZONA STATE UNIVERSITY

August 2017

ABSTRACT

Testing and Verification of Cyber-Physical Systems (CPS) is a challenging problem. The challenge arises as a result of the complex interactions between the components of these systems: the digital control, and the physical environment. Furthermore, the software complexity that governs the high-level control logic in these systems is increasing day by day. As a result, in recent years, both the academic community and the industry have been heavily invested in developing tools and methodologies for the development of safety-critical systems. One scalable approach in testing and verification of these systems is through guided system simulation using stochastic optimization techniques. The goal of the stochastic optimizer is to find system behavior that does not meet the intended specifications.

In this dissertation, three methods that facilitate the testing and verification process for CPS are presented:

1. A graphical formalism and tool which enables the elicitation of formal requirements. To evaluate the performance of the tool, a usability study is conducted.

2. A parameter mining method to infer, analyze, and visually represent falsifying ranges for parametrized system specifications.

3. A notion of conformance between a CPS model and implementation along with a testing framework.

The methods are evaluated over high-fidelity case studies from the industry.

. . . to my loving parents . . .

Naim & Nazife

ACKNOWLEDGMENTS

I enjoyed six sunny years in Tempe, Arizona. As part of the Cyber-Physical Systems lab at ASU, I had the opportunity to work with wonderful people from both the academic community and the industry. My experiences throughout this time include training at NASA, close collaboration with researchers from Toyota and Bosch, as well as work with the Air Force Research Labs, and Wright Brothers Institute. These experiences have been invaluable and have all in some shape or form contributed to the development of the work presented in this thesis.

First and foremost, I would like to express my deepest gratitude to my advisor, Georgios Fainekos, for the continuous support and supervision. He has guided me through the research process ranging from what classes to take all the way to writing papers. He was always available to meet and discuss ideas. His drive, hard work, and commitment to research and discovery has been a continuous source of inspiration.

I would like to thank the committee members Hessam, Ross, and Heni for their insights and comments. A special thanks goes to Charles Colbourn, for inviting me to be part of his legendary teaching team in theoretical computer science, and for showing me how to get the best out of people through respect and honesty.

I would also like to thank my friends at the CPS lab. Kangjin, for the stimulating discussions, and for inspiring me to stay healthy and participate in many running events. Houssam and Erkan for our close collaboration. Special thanks go to my colaborators Adel and Shakiba, for providing the model instrumentation for the hybrid nonlinear system in Section 5.3. Joe, for the great socio-economic discussions at lunch time. Ramtin, Shashank, Hengyi, Parth, Shih-Kai and Yashwanth for the work and fun times we shared together. I would like to thank the team from the Toyota Technical Center. Ken Butts, Jyotirmoy Deshmukh, James Kapinski, Xiaoqing Jin for the friendship and close collaboration throughout the years.

I moved to the US before I reached my twenties. I would not have been able to complete my studies without the love and support from my guardians away from home, Fekrije and Bujar Konjusha, Njomza and Faton Kaja, and Hyrie and Sadet Berisha who in ways they will never understand, made me a better person. I owe a lot to them for the wonderful years and for sharing their homes with me. I would also like to thank my friends Mentor, Alvaro, Valdrin and Osama. Our Thursday nights and trips throughout the west coast will always be remembered.

I would like to thank my uncles Gëzim Konjusha and Murat Cërvadiku, my aunts Kimete Dida, Laura Hajdini, Valbona Salihu, Zymryte Avdiu and their families for the endless support throughout the years. I would like to pay tribute to the memory of my grandmother Mirishahe Konjusha, for her unconditional love, and to the memory of my uncle Sabri Hoxha, for nurturing me and countless generations in the art of mathematics, for unveiling the beauty within mathematical equations.

I would like to thank my family, my betters. My siblings Hana, Ylli and Zana and their significant others. My loving parents Naim and Nazife. With your hard work and determination, in a time of terrible economical hardship for the region, you were able to support and encourage my pursuit for knowledge and betterment. I have received so much more than my share of good fortune in my life, and a large part of that comes from you.

Finally, I would like to thank Adelina, my partner. Thank you for being the person who lights my soul on fire every day. She is my treasure who has given me what I value most – her love.

My dissertation work was supported in part by the National Science Foundation under Grants CNS-1017074, CNS 1116136, CNS 1319560, CNS 1350420, IIP 1454143, CNS 1446730, IIP-1361926, by the NSF I/UCRC Center for Embedded Systems, and Air Force Research Labs Summer of Innovation 2017 program.

TABLE OF CONTENTS

CHAPTER Page

CHAPTER Page

CHAPTER Page

LIST OF TABLES

LIST OF FIGURES

Figure Page

Figure Page

PREFACE

This dissertation is the result of my six years work at Arizona State University. It is a collection of three closely related topics that support testing and verification of Cyber-Physical Systems.

Chapter 1 provides a general introduction to testing and verification methods for Cyber-Physical Systems. It also presents an overview of the three topics presented in this dissertation, along with an explicit list of papers published for each subject.

Chapter 2 presents the mathematical notation and preliminaries utilized in the rest of the thesis. It also introduces the case studies that will be used throughout.

Chapter 3 presents a graphical formalism and tool for the elicitation of formal specifications. A usability study is conducted to evaluate the performance of the tool.

Chapter 4 presents the topic of parameter mining framework for parametric temporal logic specifications. There the single-parameter mining framework is extended to multiple parameters and posed as an optimization problem.

Chapter 5 presents practical algorithms for measure conformance between a CPS model and implementation. Also, novel algorithms are presented for the exploration and coverage of the system input search space. Finally, a testing framework for finding non-conformant behaviors is presented.

Reading guide: The reader should read Chapter 1 and 2 before reading the other chapters. The other chapters are not dependent on each other.

Bardh Hoxha

Tempe, Summer 2017

Chapter 1

INTRODUCTION

1.1 Testing and Verification of CPS

In recent years, a number of accidents [109, 87, 107] and recalls [127, 33] have shown that there is a need for better testing, verification, and validation of Cyber-Physical Systems (CPS). Prime examples of such systems are aircraft, cars and medical devices. In literature, CPS are also referred to as Hybrid Systems [12]. In this thesis, we use both terms interchangeably. These systems are characterized by both continuous and discrete dynamics, with numerous subsystems interacting with each other in complex ways. This complexity makes both the design and verification problem a very challenging one. In addition, trends indicate that software complexity in CPS is going to increase in the future [118, 63, 33].

Ideally, a system developer constructs the model with a precise mathematical formulation. A suitable mathematical approach to modeling CPS is through *hybrid automata* [13, 76]. In brief, hybrid automata are extensions of finite state machines with the ability to model both discrete and continuous behavior. The discrete behavior is modeled by the finite state machine while the continuous behavior is modeled through algebraic and differential equations over a finite set of continuous variables. Several tools from the academic community such as SHIFT [50] and PTOLEMY [64] enable modeling of such systems. Also, there are several commercial tools such as Simulink/Stateflow, LabView, and Modelica that are widely used by the industry.

Verification methods [45, 78] have been very successful in finding bugs in software systems. In [96], the authors have formally verified a commercial grade, general-

purpose operating system kernel. In [23], the authors utilize formal methods for verification of railway transportation systems. In [30], the authors present favorable experimental evaluation of software verification methods on the Martian Rover software.

Similar success stories would be desirable for testing and verification of CPS. Towards this goal, there has been a substantial level of research in this direction (see [142, 92] for an overview). The methodologies range from simulation-based verification to exhaustive verification methods. The former have been shown to be more scalable and applicable to real world designs, however, in general, they are less formal and exhaustive than the latter.

One approach, within simulating based methods, works by repeatedly testing the system until a system behavior that does not satisfy the specification is observed [2]. The resulting behavior is then presented to the system developer as a counterexample that falsifies the specification. The problem of finding such a counterexample is referred to as the falsification problem. In [2], the system is tested against formal requirements which are defined in Metric Temporal Logic (MTL) [102]. The authors utilize the concept of robustness of MTL specifications [68] to turn the falsification problem into an optimization problem. The notion of the robustness metric enables system developers to measure how far a system behavior is from failing to satisfy a requirement. By utilizing such an approach, the search for falsifying behavior is guided toward less robust system behavior.

Another simulation based approach to the falsification problem is presented in [151]. There, the authors present an iterative approach for refining the abstract state graph of hybrid systems and finding counterexamples that fails the specification. The method uses partial disconnected system simulations to find a counterxample over

the discrete abstraction. After, optimization methods are used to splice these system simulations together to form a complete system trajectory that falls in the unsafe set.

Exhaustive based methods, on the other hand, include methods such as reachability analysis and theorem proving. By reachability analysis, we refer to the process of finding the reachable states of the system. To verify that a given safety specification holds, one must enumerate all the reachable states and check that they satisfy the specification. For CPS, this problem is undecidable, i.e., there is no general algorithm that terminates and answers whether a CPS satisfies a formal specification. Therefore, to alleviate the problem, several abstraction and over-approximation techniques [20, 41, 46, 73, 71, 123, 27, 9, 70] have been developed to tackle the problem.

In theorem proving, verification problems are posed as mathematical problems through theorems, lemmas, etc [6, 120, 89, 122]. By following a semi-automated process, one can prove that the system satisfies the specification. Under user guidance, the system applies inference procedures such as induction, rewriting and simplification to finally generate a proof certificate.

The topic of testing embedded software and, in particular, embedded control software is a well studied problem that involves many subtopics well beyond the scope of this thesis. We refer the reader to specialized book chapters and textbooks for further information [44, 100]. Similarly, a lot of research has been invested on testing methods for Model Based Development (MBD) of embedded systems [142]. However, the temporal logic testing of embedded and hybrid systems has not received much attention [139, 119, 114, 150].

The academic community has proposed a number of tools and methodologies that enable testing and verification of models of systems to a varying degree of formality. A number of them are listed in the following tables. Simulation-based tools are listed in Table 1.1. Reachability analysis tools are listed in Table 1.2. Theorem provers

are listed in Table 1.3. In Fig. 1.1, we place each of these tools in a graph in terms of scalability and complexity (similar to [93, Fig. S1], where the authors compare verification techniques).

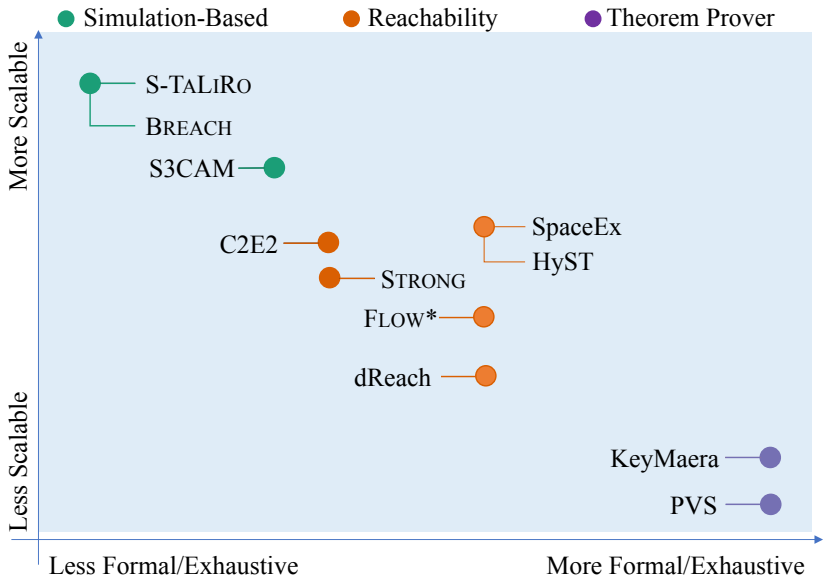


Figure 1.1: Testing and verification tools in terms of completeness and scalability.

1.1.1 V & V Terminology

We note that the terms verification, validation and testing are widely used in various research fields such as modeling, software engineering, formal methods, and control. Unfortunately, there is no consensus on a singular interpretation of these terms. To avoid confusion resulting from terminology differences, we will define the meaning of these terms in this thesis as follows.

- *Verification*: The process of proving that the system satisfies its requirements. Verification deals with the question of whether we are building the system correctly. Typically, it is desirable that this process is automated.

- *Validation*: The process of determining whether the model (and/or requirements) is

Table 1.1: Simulation-based tools for testing and verification of CPS.

S-TaLiRo	**Ref.:** [19, 82]
A Matlab toolbox for falsification, parameter mining and runtime monitoring of MTL/STL specifications. It also enables conformance testing of CPS. It can analyze arbitrary Simulink models or user-defined functions that model the system.	
Breach	**Ref.:** [58]
Similar to S-TaLiRo, Breach is a Matlab toolbox for falsification, parameter mining and runtime monitoring of STL specifications. It can analyze arbitrary Simulink models.	
S3CAM	**Ref.:** [152, 151]
The tool combines segmented trajectories to find simulations that falsify a safety property. It requires full observability and explicit representation of the states.	

Table 1.2: Theorem provers for verification of CPS.

KeYmaera [121]: An automated and interactive theorem prover aimed specifically at hybrid system verification. It also supports differential dynamic logic for hybrid programs.
PVS [6]: A verification tool which is intgrated with an automated and interactive theorem prover. PVS is compatible with powerful decision procedures and a symbolic model checker.

an accurate representation of the real word in terms of its intended use. Validation deals with the question of whether we are building the correct system. This is typically a manual process.

Table 1.3: Reachability tools for verification of CPS.

SpaceEx [70]: Enables reachability analysis over hybrid systems with piecwise affine, non-deterministic dynamics. The tool utilizes support functions and template polyhedra to represent the convex continuous sets. To find the reachable sets, the tool uses variable time-step flowpipe computation.
Strong [48]: A Matlab toolbox for bounded-time reachability of hybrid systems with linear dynamics. The tool simulates a number of trajectories and computes regions around their initial states, from which, any trajectory generated would follow the same sequence of locations and does not enter the unsafe set.
Flow* [37]: Enables bounded-time reachability analysis for non-linear hybrid automata. It uses Taylor model flowpipes to generate an over-approximation of the reachable states.
HyST [25]: The tool enables the translation of the hybrid system description from the input format utilized in SpaceEx to other verification tools which have the same hybrid system semantics. Furthermore, the tool conducts various system transformations such as order reduction and hybridization to facilitate the rachability analysis.
dReach [98]: A bounded reachability analysis tool for a wide range of non-linear hybrid systems. The tool encodes the rechability problem as δ-decision procedures for the SMT solver dReal [72].
C2E2 [62]: Enables the bounded-time reachability analysis of hybrid systems modeled as continuous or hybrid Simulink/Stateflow models. The tool uses a user-defined discrepancy function to compute over-approximations of the reach sets.

- *Testing*: The process of generating test cases (scenarios) and determining whether the test cases satisfy system requirements.

Different from verification, testing does not generaly prove that the system **satisfies** its requirements. However, testing may be used to prove that a system **does not satisfy** its requirements. Once a test case that does not satisfy the requirements is found, that implies that the system as a whole does not satisfy the requirements.

For a more elaborate discussion on the definitions, history and utilization of these terms, see [115, 116, 88].

1.1.2 Model-Based Design

A recent shift in system development, aimed to alleviate some of the testing and verification challenges, is the Model Based Design (MBD) paradigm. One of the benefits of MBD is that a significant amount of testing and verification of the system can be conducted at various stages of model development. In contrast with the traditional approach, where most of the testing and verification is conducted on the prototype of the system. The research goals presented in this dissertation aim to support a typical V process for MBD (see Fig. 1.2). In this process, the development begins with the formalization of informal requirements. Then, models and implementations are iteratively developed and tested with the goal that the final product conforms to the predetermined set of formal specifications.

1.2 Research Topics

In this dissertation, we cover three topics that support the MBD process. In the rest of this chapter, an overview of these topics is presented along with a summary of contributions and publications.

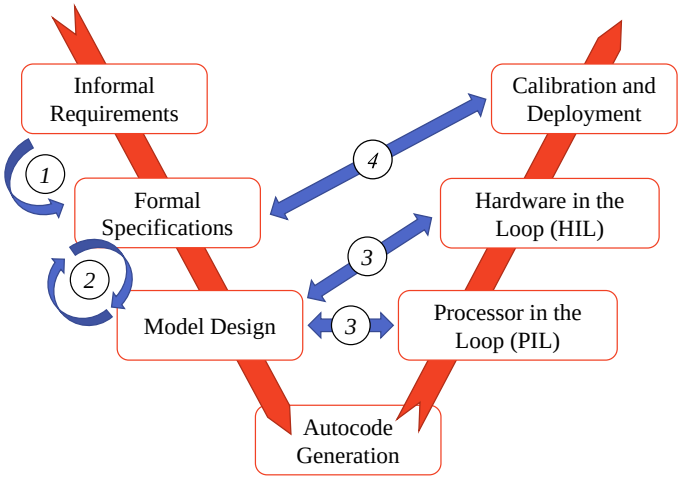


Figure 1.2: Typical V Process in MBD. (1) Elicitation of formal requirements; (2) Testing and verification of formal specification in model design; (3) Verifying conformance of the implementation to the model; (4) Verifying that the end product satisfies the functional requirements.

1.2.1 Specification Elicitation

The elicitation of formal specifications (cycle 1 in Fig. 1.2) is a challenging problem in itself. In general, requirements are expressed in natural language. The inherent ambiguity of natural language may lead to misunderstandings between development teams. These misunderstandings may result in increased costs and delays in development. Additionally, if the misunderstandings are not detected, then a product that does not meet the intended specifications may be developed. Ideally, specifications should be defined in a mathematical language, using formal logics. The use of formal logics not only removes ambiguity but also allows system developers to utilize the vast set of the methods discussed in Section 5.1 to conduct system testing and verification. It has been shown, that utilizing formal specifications can lead to improved testing and verification for software systems [104, 77]. It is possible to get

similar benefits in applying semi-formal methods for CPS as preliminary results in [66] show. Despite the benefits, one may conjecture that the primary reason for doing so is because the development of specifications through a formal logic requires a level of mathematical training that many users may not have [143]. Furthermore, even for expert users, writing formal specifications is an error prone task [79, 54]. As a result, the industry has been less willing to utilize formal specifications in their processes. The question that arises is: is it possible to develop an accessible graphical formalism that would enable users to accurately elicit formal specifications for CPS with little to no mathematical training in formal languages?

1.2.2 Multiple Parameter Specification Mining

To support cycles 2 and 4 in Fig. 1.2, it is often necessary to explore and determine system properties. The practitioner, with partial understanding of system specifications, would like to investigate the possible ranges for which the system specification is satisfied/falsified. For example, for an automotive system, a system specification could be as follows: "Always, the engine speed should be less than λ_1 and vehicle speed should be less than λ_2". An additional example could be as follows: "It is not the case that, eventually, within t_1 seconds, the vehicle speed is greater than 100 and always, engine speed is less than λ_1".

A suitable mathematical formalism for representing these specifications is Parametric Metric Temporal Logic (PMTL) [21]. PMTL allows for the formalization of specifications which are defined over both state and time, where either state or time may be parametrized as in the aforementioned natural language specifications. Once a PMTL specification, with unknown state and timing parameters is defined, it is necessary to find the range of parameter values such that the system is not satisfied.

The problem is challenging due to several factors. The systems under considera-

tion are complex CPS. In general, determining whether a CPS satisfies a specification with certain parameter values is undecidable. Therefore, a possible approach to exploring the parameter space is through system simulations. However, generating system simulations can be computationally expensive. Thus, a solution to this problem should use efficient methods for exploring system behaviors. A question that arises is, what guarantees can we provide on parameter ranges that satisfy/falsify specifications? Also, how do we visualize and illustrate the relationship between state and timing parameters in MTL specifications?

1.2.3 Practical Algorithms for Conformance

Throughout the MBD development process, it is often desirable to develop several models and implementations of varying fidelity (see cycle 3 in Fig. 1.2). Models of different fidelity levels can enable mathematical analysis of the model, control synthesis, faster simulation, etc. To ensure that the intended system behavior is preserved in the model refinement process, it is necessary to define a rigorous notion of conformance between different models and between models and their implementations. In our framework, we consider that the specifications are defined in MTL. Therefore, this conformance metric must encapsulate both state and timing behaviors of the model and implementation. How do we explore model and implementation behaviors to establish this notion of conformance? Would it be possible to guarantee that the implementation satisfies the specifications that were considered to hold on the model? Could we establish a conformance relation for cases where the practitioner does not have full state knowledge on the system and does not have complete control over system initial conditions? An example of this would be testing a system model and a hardware-in-the-loop implementation. Usually, for the former, you have full control over inputs and initial conditions which might not be the case for the latter.

1.3 Summary of Contributions

In the following, I summarize the contributions and publications related to the aforementioned topics.

Chapter 3 - Specification Elicitation:

- ***B. Hoxha**, H. Bach, H. Abbas, A. Dokhanchi, Y. Kobayashi and G. Fainekos, **Towards Formal Specification Visualization for Testing and Monitoring of Cyber-Physical Systems**, in Proc. of Int. Workshop on Design and Implementation of Formal Tools and Systems, 2014* [82].

  In this work, I proposed a graphical formalism for the elicitation of formal specifications specifically geared towards CPS. The formalism enables the visualization of a wide array of MTL specifications. It is designed for use with systems and signals and enables both event and time-based specifications.

- ***B. Hoxha**, N. Mavridis and G. Fainekos, **VISPEC : A graphical tool for elicitation of MTL requirements**, in Proc. of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems, 2015* [26].

  In this paper, I expanded and improved upon the graphical formalism presented in [82]. I developed a tool based on the graphical formalism. To evaluate the usefulness of the tool, I conducted a usability study. Through the usability study, it was shown that the tool could be utilized by both expert and non-expert users to define formal specifications accurately. The tool can be downloaded at `https://sites.google.com/a/asu.edu/s-taliro/vispec`.

- *A. Dokhanchi, **B. Hoxha**, and G. Fainekos, **Metric Interval Temporal Logic Specification Elicitation and Debugging**, in Proc. of the ACM-IEEE In-*

ternational Conference on Formal Methods and Models for System Design, 2015 [54].

In this work, the elicitation framework was extended by adding a debugging algorithm that enables the detection of various issues in specifications. The debugging algorithm detects validity, redundancy and vacuity issues in formal specifications. The ability to automatically debug improves the specification elicitation process and, ultimately, the testing and verification process. In this work, I developed the specification elicitation formalism and tool. I conducted an online study to evaluate the ability of Formal Methods experts to write accurate specifications in MTL. Additionally, the experimental results were based on the usability study I conducted in [26].

Chapter 4 - Parameter Mining:

- *H. Yang,* ***B. Hoxha*** *and G. Fainekos,* ***Querying parametric temporal logic properties on embedded systems****, in Testing Software and Systems, 2012* [147].

  In this work, the notion of system robustness is utilized to explore and determine system properties. In more detail, given a parameterized MTL specification [21], where there is a single unknown state or timing parameter, we find the range of values for the parameter such that the system is not satisfied. In this work, I contributed by implementing the parameter estimation algorithms in the S-TaLiRo toolbox.

- ***B. Hoxha*** *and G. Fainekos,* ***Pareto Front Exploration for Parametric Temporal Logic Specifications of Cyber-Physical Systems****, in the Workshop on Monitoring and Testing of Cyber-Physical Systems, 2016* [84].

- ***B. Hoxha****, A. Dokhanchi and G. Fainekos,* ***Mining parametric temporal***

***logic properties in model-based design for cyber-physical systems**, in the International Journal on Software Tools for Technology Transfer, 2017* [83].

In this paper, the parameter mining problem presented in [147] is extended and generalized to specifications with multiple parameters. In this work, I extended the theory of parameter mining for multiple parameters and improved on the efficiency of the stochastic optimization algorithms. Furthermore, I developed two algorithms that enable the exploration of the Pareto front generated by the robustness landscape of the problem. Finally, I conducted experiments using the method on an industrial-size case study of a high-fidelity engine model and performed an extensive analysis of the related works in the subject area.

Chapter 5 - Conformance Testing:

- *H. Abbas, **B. Hoxha**, G. Fainekos, J. Deshmukh, J. Kapinski and K. Ueda, **Conformance Testing as Falsification for Cyber-Physical Systems**, in arXiv Systems and Control, 2014* [3, 4].

  In this paper, a notion of conformance between models in different stages of development and their implementation is presented. In addition, a framework for testing non-conformance using automated methods is introduced. In this work, I researched various metrics to establish the conformance relation between systems. I conducted experimental tests with academic and industrial level models.

Other contributions and publications:

I have been one of the main contributors to a widely used, highly cited, testing and verification toolbox S-TaLiRo [126, 82]. S-TaLiRo has been applied to numerous challenging applications from the automotive and medical device industries. In [82], I wrote an updated overview of the toolbox and its features. The toolbox was

nominated as a technological breakthrough by the industry [131, 132]. The completed works presented in this proposal have been incorporated in S-TaLiRo.

Other contributions include:

- ***B. Hoxha**, H. Abbas, and G. Fainekos, **Benchmarks for Temporal Logic Requirements for Automotive Systems**, in the Workshop on Applied Verification for Continuous and Hybrid Systems, 2014* [80].

- ***B. Hoxha**, H. Abbas and G. Fainekos, **Using S-TaLiRo on industrial size automotive models**, in the Workshop on Applied Verification for Continuous and Hybrid Systems, 2014* [81].

  In the first paper, two Matlab/Simulink models of automotive systems are proposed as benchmark problems for hybrid system verification. Both models can be simulated quickly, making them ideal for testing-based verification methods that require a significant number of system output trajectories. The paper also defines a number of MTL specifications that must be satisfied by the models. In this work, I presented the benchmarks and the specifications.

  In the second paper, I demonstrated various testing and simulation-based methods using S-TaLiRo on an industrial size, high-fidelity engine model.

- *H. Abbas, **B. Hoxha**, G. Fainekos and K. Ueda, **Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems**, in the Proc. of the 2014 IEEE 4th Annual International Conference on Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014* [5].

  In this work, a testing and verification framework for stochastic CPS is presented. The goal of the testing framework is to detect system operating conditions that cause the system to exhibit the worst expected specification robustness with finite-time guarantees. The resulting expected robustness minimization problem is solved

using Markov chain Monte Carlo algorithms. This allows the development of finite-time guarantees, which quantify the quality of the solution after a finite number of simulations. In this work, I developed the Expected Robustness Guided Monte Carlo (ERGMC) algorithm and presented a design and verification development process for stochastic CPS. Finally, I conducted a case study on the performance of the framework using a high-fidelity engine model. The paper was a finalist for the best student paper award.

- *A. Dokhanchi,* ***B. Hoxha*** *and G.Fainekos,* ***On-Line Monitoring for Temporal Logic Robustness****, in Proc. of Runtime Verification, volume 8734 of LNCS, Springer, 2014* [53].

  In this work, a dynamic programming algorithm for online monitoring of MTL specifications was presented. In this work, I conducted a case study to establish an acceptable overhead of the monitoring algorithm for certain classes of practical specifications.

- *A. Dokhanchi,* ***B. Hoxha****, C.E. Tuncali and G.Fainekos,* ***An Efficient Algorithm for Monitoring Practical TPTL Specifications****, in Proc. of Runtime Verification, volume 8734 of LNCS, Springer, 2014* [56].

  Here, a dynamic programming algorithm for the monitoring of a fragment of Timed Propositional Temporal Logic (TPTL) specifications is presented. For this fragment, an efficient, polynomial-time algorithm for off-line monitoring of finite traces is presented.

Chapter 2

SYSTEMS, SIGNALS AND SPECIFICATIONS

2.1 Systems and Signals

In this section, we review the notations and definitions from [83, 2]. We treat CPS as an input-output map. Specifically, we consider a system Σ as a mapping from a compact set of *initial operating conditions* X_0 and *input signals* $\mathbf{U} \subseteq U^N$ to *output signals* Y^N and *timing* (or *sampling*) functions $\mathfrak{T} \subseteq \mathbb{R}_+^N$. Here, U is a compact set of possible input values at each point in time (input space), Y is the set of output values (output space), $\mathbb{R}$ is the set of real numbers and $\mathbb{R}_+$ the set of positive reals. We fix $N \subseteq \mathbb{N}$, where $\mathbb{N}$ is the set of natural numbers, to be a finite set of indexes for the finite representation of a system behavior.

We impose three assumptions/restrictions on the systems that we consider:

Assumption 2.1.1 *The input signals (if any) must be parameterizable using a finite number of parameters. That is, there exists a function* $\mathfrak{U}$ *such that for any* $u \in \mathbf{U}$*, there exist two parameter vectors* $\vec{\lambda} = [\lambda_1 \ \ldots \ \lambda_m]^\intercal \in \Lambda$*, where* Λ *is a compact set, and* $\vec{t} = [t_1 \ \ldots \ t_m]^\intercal \in \mathbb{R}_+^m$ *such that* m *is typically much smaller than the maximum number of indices in* N *and for all* $i \in N$*,* $u(i) = \mathfrak{U}(\vec{\lambda}, \vec{t})(i)$*.*

Assumption 2.1.2 *The output space* Y *must be equipped with a generalized metric* $\mathbf{d}$ *which contains a subspace* Z *equipped with a metric* d *[2].*

Assumption 2.1.3 *For a specific initial condition* x_0 *and input signal* u*, there must exist a unique output signal* $\mathbf{y}$ *defined over the time domain* R*. That is, the system* Σ *is deterministic.*

Further details on the necessity and implications of the aforementioned assumptions can be found in [2]. Assumption 2.1.3 can also be relaxed as shown in [5].

Under Assumption 2.1.3, a system Σ can be viewed as a function $\Delta_\Sigma : X_0 \times \mathbf{U} \to Y^N \times \mathfrak{T}$ which takes as an input an initial condition $x_0 \in X_0$ and an input signal $u \in \mathbf{U}$ and it produces as output a signal $\mathbf{y} : N \to Y$ (also referred to as *trajectory*) and a timing function $\tau : N \to \mathbb{R}_+$. The only restriction on the timing function τ is that it must be a monotonic function, i.e., $\tau(i) < \tau(j)$ for $i < j$. The pair $\mu = (\mathbf{y}, \tau)$ is usually referred to as a *timed state sequence*, which is a widely accepted model for reasoning about real-time systems [11].

A timed state sequence can represent a computer-simulated trajectory of a CPS or the sampling process that takes place when we digitally monitor physical systems. We remark that a timed state sequence can represent both the internal state of the software/hardware (usually through an abstraction) and the state of the physical system. The set of all timed state sequences of a system Σ will be denoted by $\mathcal{L}(\Sigma)$. That is,

$$\mathcal{L}(\Sigma) = \{(\mathbf{y}, \tau) \mid \exists x_0 \in X_0 \,.\, \exists u \in \mathbf{U} \,.\, (\mathbf{y}, \tau) = \Delta_\Sigma(x_0, u)\}.$$

2.2 Hybrid Systems

Hybrid systems can be modeled using hybrid automata [13]. In Fig. 2.1, we have an example of a hybrid nonlinear automaton. The system has two locations (or modes) S_1 and S_2. Depending on what location the system is in, the system evolves under different dynamics. The initial conditions of the system dictate whether the system is initialized at S_1 or S_2. In this example, only one transition from S_1 to S_2 is possible.

Generaly, hybrid automata may have a finite number of locations. The algebraic and differential equations in the locations dictate the continous evolution of the sys-

tem. Furthermore, the state variables may reset to a specific value once a transition from one location to the next occurs. For a formal definition of hybrid automata, we refer the reader to [76, 13].

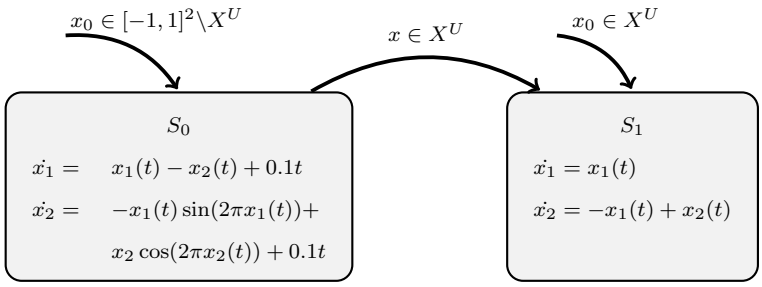


Figure 2.1: Hybrid nonlinear system with $X^U = [0.85, 0.95]^2$ and initial condition $x_0 \in [-1, 1] \times [-1, 1]$.

2.3 Metric Temporal Logic

Our high-level goal is to analyze, explore and infer properties that the system Σ satisfies. We do so by observing the system response (output signals) to particular input signals and initial conditions. In particular, we assume that the system developer can formalize the system properties in Metric Temporal Logic (MTL) [102].

MTL enables the formalization of complex requirements with respect to both state and time. In addition to propositional logic operators such as conjunction ($\wedge$), disjunction ($\vee$) and negation ($\neg$), MTL supports temporal operators such as next (X), until ($\mathcal{U}$), release ($\mathcal{R}$), always ($\Box$) and eventually ($\Diamond$). Among others, MTL can be utilized to express specifications such as:

- Safety ($\Box\phi$) : ϕ should always hold from this moment on.
- Liveness ($\Diamond\phi$): ϕ should hold at some point in the future (or now).

- Coverage ($\Diamond\phi_1 \wedge \Diamond\phi_2 \ ... \wedge \Diamond\phi_n$): ϕ_1 through ϕ_n should hold at some point in the future (or now), not necessarily in order or at the same time.

- Stabilization ($\Diamond\Box\phi$): At some point in the future (or now), ϕ should always hold.

- Recurrence ($\Box\Diamond\phi$) : At every point time, ϕ should hold at some point in the future (or now).

Definition 2.3.1 (Syntax of MTL) *Let $\overline{\mathbb{R}}$ be the set of truth degree constants, AP be the set of atomic propositions and $\mathcal{I}$ be a non-empty non-singular interval of $\overline{\mathbb{R}}_{\geq 0}$. The set MTL of all well-formed formulas (wff) is inductively defined using the following rules:*

- *Terms: True ($\top$), false ($\bot$), all constants $r \in \overline{\mathbb{R}}$ and atomic propositions p, $\neg p$ for $p \in AP$ are terms.*

- *Formulas: if ϕ_1 and ϕ_2 are terms or formulas, then $\phi_1 \vee \phi_2$, $\phi_1 \wedge \phi_2$, $\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2$ and $\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$ are formulas.*

The atomic propositions in our case label subsets of the output space Y. In other words, each atomic proposition is a shorthand for an arithmetic expression of the form $p \equiv g(y) \leq c$, where $g : Y \to \mathbb{R}$ and $c \in \mathbb{R}$. We define an observation map $\mathcal{O} : AP \to 2^Y$ such that for each $p \in AP$ the corresponding set is $\mathcal{O}(p) = \{y \mid g(y) \leq c\} \subseteq Y$.

In the above definition, $\mathcal{U}_{\mathcal{I}}$ is the timed *until* operator and $\mathcal{R}_{\mathcal{I}}$ the timed *release* operator. The subscript $\mathcal{I}$ imposes timing constraints on the temporal operators. The interval $\mathcal{I}$ can be open, half-open or closed, bounded or unbounded, but it must be non-empty ($\mathcal{I} \neq \emptyset$) (and, practically speaking, non-singular ($\mathcal{I} \neq \{t\}$)). In the case

where $\mathcal{I} = [0, +\infty)$, we remove the subscript $\mathcal{I}$ from the temporal operators, i.e., we just write $\mathcal{U}$ and $\mathcal{R}$. Also, we can define the *eventually* ($\Diamond_{\mathcal{I}}\phi \equiv \top\,\mathcal{U}_{\mathcal{I}}\phi$) and *always* ($\Box_{\mathcal{I}}\phi \equiv \bot\mathcal{R}_{\mathcal{I}}\phi$) temporal operators. Furthermore, for discrete signals, the *next* (X) ($X\phi = \top\,\mathcal{U}_{[1,1]}\phi$) operator may be utilized. Note that here the subscript [1,1] denotes the next sample. The formal robust semantics of MTL formulas are presented in Section 4.3.

Another popular formalism for the definition of formal requirements is Signal Temporal Logic (STL) [110]. Since MTL formulas are interpreted over behaviors of the CPS, the results provided in this thesis can be directly applied over STL formulas as well.

2.4 Parametric Metric Temporal Logic

In many cases, it is important to be able to describe an MTL specification with unknown parameters and then, mine the falsifying parameters values. In [21], Asarin et al. introduced Parametric Signal Temporal Logic (PSTL) and presented two algorithms for computing approximations for parameters over a given signal. Here, we review and extend some of the results in [21] while adapting them in the notation and formalism that we use in this thesis.

Definition 2.4.1 (Syntax of Parametric MTL) *Let $\vec{\theta} = [\theta_1 \dots \theta_n]$ be a vector of parameters. The set of all well-formed Parametric MTL (PMTL) formulas is the set of all well-formed MTL formulas where for all i, θ_i either appears in an arithmetic expression, i.e., $p[\theta_i] \equiv g(y) \leq \theta_i$, or in the timing constraint of a temporal operator, i.e., $\mathcal{I}[\theta_i]$.*

We will denote a PMTL formula ϕ with parameters $\vec{\theta}$ by $\phi[\vec{\theta}]$. Given a vector of parameters $\vec{\theta} \in \Theta$, then the formula $\phi[\vec{\theta}]$ is an MTL formula. There is an implicit mapping from the vector of parameters $\vec{\theta}$ to the corresponding arithmetic expressions

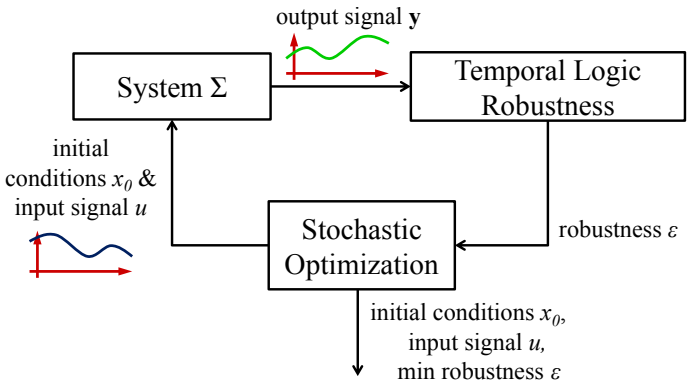


Figure 2.2: The S-TaLiRo automatic test case generation framework.

and temporal operators in the MTL formula. Once a parameter valuation is defined, a PMTL formula is transformed into a MTL formula and the robust semantics as defined in Section 4.3 apply.

2.5 Automated Test Case Generation

The work presented in Chapters 4 and 5 builds over the S-TaLiRo [19, 82] automatic test case generation framework (see Fig. 2.2). There, given a model and an MTL specification, the sampler produces a point x_0 from the set of initial conditions and an input signal u. The initial conditions and input signal are passed to the system simulator which returns an execution trace (output trajectory and timing function). The trace is then analyzed by the robustness analyzer which returns a robustness value. The robustness value computed is used by the stochastic sampler to decide on the next initial conditions and inputs. The process terminates once a falsifying trajecty is found or a maximum number of tests is reached. The algorithm will return the least robust system behavior with the corresponding input signal and initial conditions.

The input signals in this framework are parameterized using m number of control

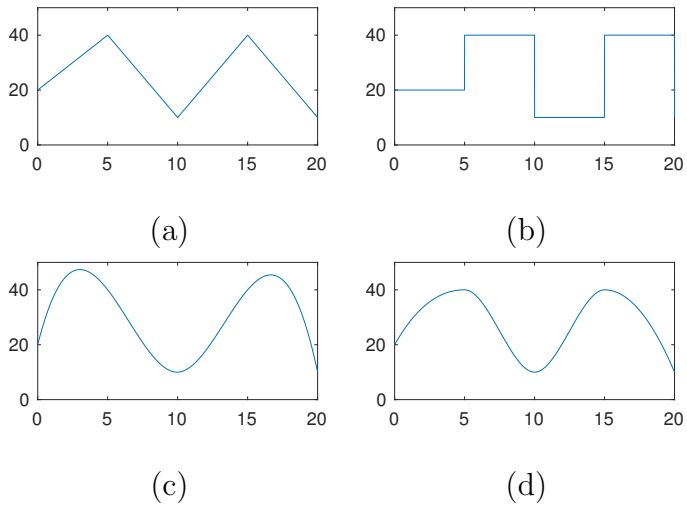
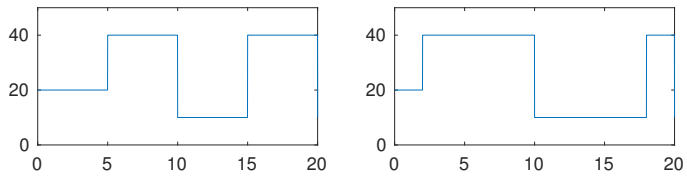


Figure 2.3: Signal generation with state control points $\vec{\lambda} = [20, 40, 10, 40, 10]$ and equidistant timing control points $\vec{t} = [0, 5, 10, 15, 20]$ with various interpolation functions. (a) Linear, (b) Piecewise constant, (c) Spline, (d) Piecewise cubic interpolation.

points. The control points state vector $\vec{\lambda}$ and the timing vector $\vec{t}$, in conjunction with an interpolation function $\mathfrak{U}$, define the input signal u. Namely, for time i, $u(i) = \mathfrak{U}(\vec{\lambda}, \vec{t})(i)$. The practitioner may choose different interpolation functions depending on the system and application. Example functions, as shown in Fig. 2.3, include linear, piecewise constant, splines, piecewise cubic interpolation etc. If timing control points are not included, the state control points will be distributed equidistantly with respect to time with a chosen interpolation function. Otherwise, the timing of the state control points is defined by the timing control points. The timing option is illustrated in Fig. 2.4. Choosing the appropriate number of control points and interpolation functions is application dependent. Timing should be included in the search space whenever the system should be tested under conditions where the input variation could be high in a very short period of time. By including timing between control points in the search space, one may be able to produce behaviors such as jerking behavior for the gas and brake throttle of an automotive vehicle. Note that



(a) (b)

Figure 2.4: Signal generation with state control points $\vec{\lambda} = [20, 40, 10, 40, 10]$ and piecewise constant interpolation. (a) With no timing control points, (b) With timing control points $\vec{t} = [0, 2, 10, 18, 20]$.

in this framework, for systems with multiple inputs, each input can have a different number of control points and interpolation function. This enables the practitioner to define a wide array of input signals.

2.6 Running Examples

2.6.1 Automotive Transmission (AT)

We consider a slightly modified version of the Automatic Transmission model provided by Mathworks as a Simulink demo [1] (Fig. 2.5). The only input u to the system is the throttle schedule, while the brake schedule is set simply to 0 for the duration of the simulation which is $T = 30\,\text{sec}$. The physical system has two continuous-time state variables x which are also its outputs $\mathbf{y}$: the speed of the engine ω (RPM) and the speed of the vehicle v, i.e., the output space is $Y = \mathbb{R}^2$ with $\mathbf{y}(i) = [\omega(i)\ v(i)]^T$ for all $i \in [0, 30]$. Initially, the vehicle is at rest at time 0, i.e., $X_0 = \{[0\ 0]^T\}$ and $x_0 = \mathbf{y}(0) = [0\ 0]^T$. Therefore, the output trajectories depend only on the input signal u which models the throttle, i.e., $(\mathbf{y}, \tau) = \Delta_\Sigma(u)$. The throttle at each point in time can take any value between 0 (fully closed) to 100 (fully open).

[1] Available at: `http://www.mathworks.com/help/simulink/examples/modeling-an-automatic-transmission-controller.html`

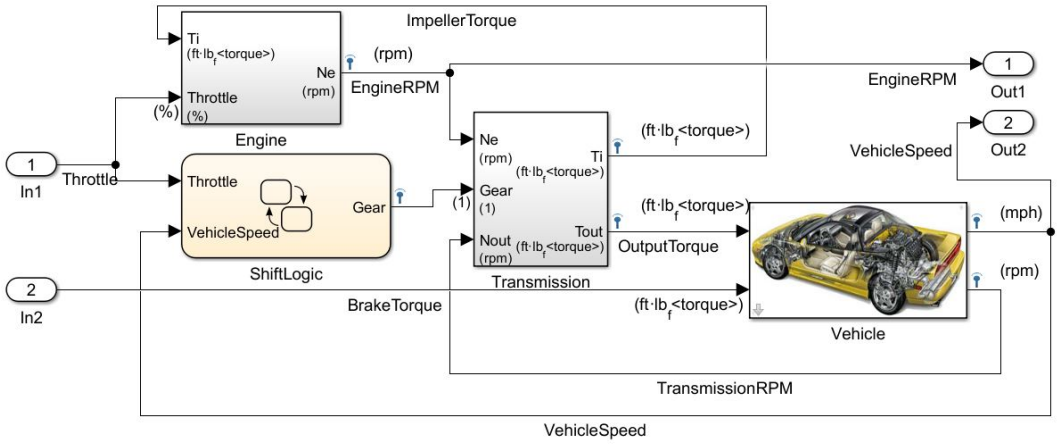


Figure 2.5: Automotive Transmission Model.

Namely, $u(i) \in U = [0, 100]$ for each $i \in N$. The model also contains a Stateflow chart with two concurrently executing Finite State Machines (FSM) with 4 and 3 states, respectively. The FSM models the logic that controls the switching between the gears in the transmission system. We remark that the system is deterministic, i.e., under the same input signal u, we will observe the same output signal $\mathbf{y}$. In [2], the authors demonstrated how to falsify requirements like: "The vehicle speed v is always under 120km/h or the engine speed ω is always below 4500 RPM". In MTL, the requirement is formalized as $\phi = \Box((v \leq 120) \wedge (\omega \leq 4500))$. A falsifying system trajectory appears in Fig. 2.6 (Left). More specifications over this model are presented in Table 2.1. For a more detailed presentation of this model see [80].

2.6.2 High-Fidelity Automotive Engine (HAE)

We utilize an industrial size high-fidelity engine model. The model is part of the SimuQuest Enginuity [135] Matlab/Simulink tool package. The Enginuity tool package includes a library of modules for engine component blocks. It also includes pre-assembled models for standard engine configurations, see Fig. 2.7. In this work,

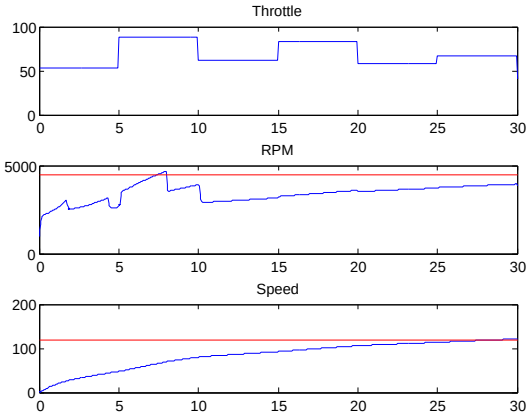
Table 2.1: Various specifications for the AT and HAE models [80].

	Natural Language	**MTL**
ψ_1	There should be no transition from gear two to gear one and back to gear two in less than 2.5 sec.	$\Box((g_2 \wedge Xg_1) \rightarrow \Box_{(0,2.5]}\neg g_2)$
ψ_2	After shifting into gear one, there should be no shift from gear one to any other gear within 2.5 sec.	$\Box((\neg g_1 \wedge Xg_1) \rightarrow \Box_{(0,2.5]}g_1)$
ψ_3	When shifting into any gear, there should be no shift from that gear to any other gear within 2.5sec.	$\wedge_{i=1}^{4}\Box((\neg g_i \wedge Xg_i) \rightarrow \Box_{(0,2.5]}g_i)$
ψ_4	If engine speed is always less than $\bar{\omega}$, then vehicle speed can not exceed $\bar{v}$ in less than T sec.	$\neg(\Diamond_{[0,T]}(v > \bar{v}) \wedge \Box(\omega < \bar{\omega}))$ or $\Box(\omega < \bar{\omega}) \rightarrow \Diamond_{[0,T]}(v > \bar{v})$
ψ_5	Within T sec the vehicle speed is above $\bar{v}$ and from that point on the engine speed is always less than $\bar{\omega}$.	$\Diamond_{[0,T]}((v \geq \bar{v}) \wedge \Box(\omega < \bar{\omega}))$
ψ_6	A gear increase from first to fourth in under 10secs, ending in an RPM above $\bar{\omega}$ within 2 seconds of that, should result in a vehicle speed above $\bar{v}$.	$((g_1 \ \mathcal{U} \ g_2 \ \mathcal{U} \ g_3 \ \mathcal{U} \ g_4) \wedge \Diamond_{[0,10]}(g_4 \wedge \Diamond_{[0,2]}(\omega \geq \bar{\omega}))) \rightarrow \Diamond_{[0,10]}(g_4 \wedge X(g_4 \, \mathcal{U}_{[0,1]} \, (v \geq \bar{v})))$

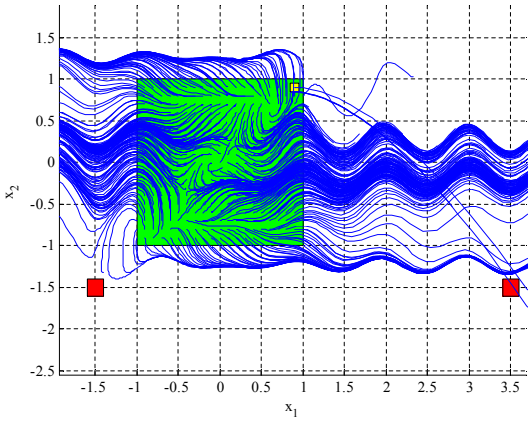
ω: Engine rotation speed, v: vehicle velocity, g_i : gear i.

Recommended values: $\bar{\omega}$: 4500, 5000, 5200, 5500 RPM; $\bar{v}$: 120, 160, 170, 200 mph; T : 4, 8, 10, 20 sec

$\Box$: Always, $\Diamond$: Eventually, $\mathcal{U}$: Until, X: Next



(a) Example 2.6.1 (AT)



(b) Example 2.6.3 (HS)

Figure 2.6: **Left (a):** Example 2.6.1 (AT): Throttle: A piecewise constant input signal u parameterized with $\Lambda = [0, 100]^6$ and $\vec{t} = [0, 5, 10, 15, 20, 25]$. RPM, Speed: The corresponding output signals that falsify the specification "The vehicle speed v is always under 120mph or the engine speed ω is always below 4500 RPM." **Right (b)**: Example 2.6.3 (HS): Simulated trajectories of the hybrid system containing a trajectory that falsifies the specification "A trajectory should never pass set $[-1.6, -1.4]^2$ or set $[3.4, 3.6] \times [-1.6, -1.4]$". The green square indicates the set of possible initial conditions and the red squares indicate the bad regions which the system should not enter. The yellow region indicates the set of initial conditions where the mode or location on the hybrid system changes.

we will use the Port Fuel Injected (PFI) spark ignition, 4 cylinder inline engine configuration. It models the effects of combustion from first physics principles on a cylinder-by-cylinder basis, while also including regression models for particularly complex physical phenomena. Simulink reports that this is a 56 state model. The model includes a tire-model, brake system model and a drive train model (including final drive, torque converter and transmission). The model is based on a zero-dimensional modeling approach so that the model components can all be expressed in terms of ordinary differential equations. The inputs to the system are the throttle and brake schedules, and the road grade, which represents the incline of the road. The outputs are the vehicle and engine speed, the current gear and a timer that indicates the time spent on a gear. Several specifications over this model are presented in Table 2.1. An interesting query about the system is the following. Does a transition exist from gear two to gear one and back to gear two in less than τ seconds? This is formalized with the following PMTL specification:

$$\phi = \Box((gear_2 \wedge X gear_1) \rightarrow \Box_{(0,\tau]} \neg gear_2)$$

2.6.3 Hybrid Nonlinear System (HS)

As the third running example, we consider the hybrid time-varying nonlinear system presented in Fig. 2.1 as a hybrid automaton [10]. The output of the system is the state of the system, i.e. $\mathbf{y}(t) = x(t)$. Interesting requirements on this system would be "A trajectory of the system should never pass through the sets $[-1.6, -1.4]^2$ or $[3.4, 3.6] \times [-1.6, -1.4]$". A falsifying system trajectory appears in Fig. 2.6 (Right).

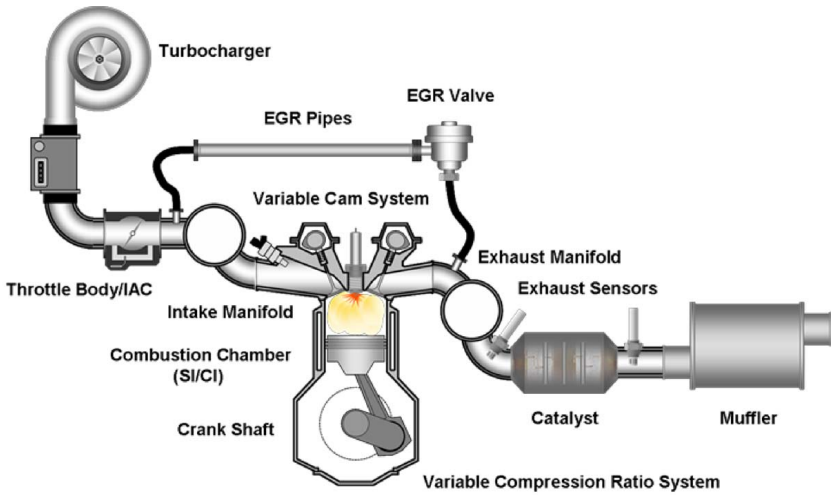


Figure 2.7: SimuQuest Enginuity high-fidelity engine model components. Used with permission, ©SimuQuest [135].

Chapter 3

ELICITATION OF FORMAL SPECIFICATIONS

3.1 Introduction

As robots become commercially available, their correct operation is of paramount importance. Especially for safety critical systems, safety must be guaranteed. For example, consider the verification works for autonomous vehicles [146] and medical robots [111, 101]. An integral component or this process is the elicitation of formal requirements.

Safety requirements are usually expressed in natural language, which is inherently ambiguous, in general. When it is used for defining system specifications, this ambiguity may lead to misunderstandings between development teams that may result in increased costs and delays in development. If the misunderstandings are not detected early, then a product that does not meet the intended specifications will be developed.

Ideally, specifications should be defined in a mathematical language, using formal logics. This not only removes ambiguity, but also allows system developers to utilize a vast set of methods [92] that have been developed by the academic community for testing and verification of systems. The academic community has also developed automatic tools such as S-TaLiRo [19, 82], Fapas [148], SpaceEx [70], CheckMate [134], Flow [37], Breach [58], C2E2 [61], KeYmaera [121] and Strong [48] that enable developers to conduct system testing and verification.

Even though it has been shown, that utilizing formal specifications can lead to improved testing and verification [66], the industry still utilizes natural language as the preferred approach in defining specifications. One may conjecture that the most

important reason for doing so is because the development of specifications through a formal logic requires a level of mathematical training that many users may not have [143]. Furthermore, even for expert users, writing formal specifications is an error prone task [79]. As a result, the industry has been less willing to utilize formal specifications in their processes.

In this chapter, we present a graphical formalism that enables non-expert users to develop formal specifications for control systems. The formalism enables the visualization of a large fragment of MTL. The main challenge in the development of the formalism lies in finding the right balance between expressive power and ease-of-use. It is designed for use with systems and signals and enables both event and time based specifications. This is the first time that a visual formal language representation is developed for specifications for Cyber-Physical Systems (CPS). Here by CPS we define any system that has discontinuous nonlinear dynamics and complex safety critical requirements (see Section 2.2). Prime examples are medical robotics and autonomous vehicles. A specification visualization tool has been developed based on the graphical formalism presented in this work. To evaluate the usefulness of the tool in terms of usability and ease-of-use, we have conducted a usability study.

In this chapter:

- A graphical formalism that enables the development of formal specifications is presented.
- A visual specification tool based on the graphical formalism is introduced.
- A usability study is conducted to evaluate the tool.
- Applications of the tool to real-world robots are presented.

3.2 Visual Specification Tool

The Visual Specification Tool (ViSpec) [1] enables the development of formal specifications for CPS. Users can develop requirements in a graphical formalism which is then translated to Metric Temporal Logic (MTL) [102] (see Section 2.3).

The topic of capturing requirements through graphical formalisms has been studied in the past [137, 8, 103, 22, 149]. However, to the best of the author's knowledge, the work presented here is the first attempt to do so specifically aimed for the development of specifications for CPS. The initial idea for the graphical formalism was first presented in [82] while the tool was still in the early stages of development. In [26], an updated version of the tool is presented along with a usability study. The improvements over the previous version include: a more streamlined interface; an updated representation of signals in the interface; and an updated template definition process.

For CPS specifications, it is often needed to account for both timing and event sequence occurrences. Both of these are necessary for reasoning over systems and signals. Consider the specification $\square_{[0,5]}((speed > 100) \rightarrow \square_{[0,5]}(rpm > 4000))$. It states that whenever within the first 5 seconds, the *vehicle speed* goes over 100, then from that moment on, the *engine speed (rpm)*, for the next 5 seconds, should always be over 4000. Here both the sequence and timing of the events are of critical importance.

To ensure that the tool can be utilized by non-expert users, the following goals for the tool are defined: 1) The user interface is intuitive to use, i.e, it does no have a high learning curve; 2) The visual representation of the requirements is visually distinct and unambiguous; 3) There is a one-to-one mapping from the visual representation of the requirement and the corresponding requirement in MTL. The set of specifications

[1] Available at `https://sites.google.com/a/asu.edu/s-taliro/vispec`

that can be generated from this graphical formalism is a proper subset of the set of MTL specifications.

Table 3.1: Classes of specifications expressible with the graphical formalism.

Specification Class	Explanation
Safety	Specifications of the form $\Box\phi$ used to define specifications where ϕ should always be true.
Reachability	Specifications of the form $\Diamond\phi$ used to define specifications where ϕ should become true at least once in the future (from now).
Stabilization	Specifications of the form $\Diamond\Box\phi$ used to define specifications that, at least once, ϕ should be true and from that point on, stay true.
Recurrence	Specifications of the form $\Box\Diamond\phi$ used to define specifications that, it is always the case, that at some point in the future, ϕ is true.
Implication	Specifications of the form $\phi \rightarrow \psi$ requires the ψ should hold when ϕ is true.
Reactive Request-Response	Specifications of the form $N(\phi \rightarrow M\psi)$, where N and M are temporal operators, used to define an implication response between two specifications where the timing of M is relative to timing of N.
Conjunction	Specifications of the form $\phi \wedge \psi$ used to define the conjunction of two sub-specifications
Non-strict Sequencing	Specifications of the form $N(\phi \wedge M\psi)$, where N and M are temporal operators, used to define a conjunction between two specifications where the timing of M is relative to timing of N.

Throughout the development process of the formalism, it was noticed that the more expressive the formalism, the more challenging to use it became. Therefore, we focused on several widely used classes of specifications which are described in Table 3.1. Examples of the classes of specifications are presented in the rest of this section.

To make the tool easier to use, we placed several constraints on the types of

signals used. Specifically, the signals and requirements are one-dimensional. This enables clear and structured visualization on a two-dimensional user interface.

In Fig. 3.1, the user interface of the tool is presented along with its most critical components. The user interface is composed of a menu, horizontal timeline, rectangular blocks called templates, and a zoom scroll. While the passage of time is represented horizontally, the sequence of events is presented vertically. The formulas are generated from templates as well as the connections between them. To enable a compositional representation of the templates we use the same template for reactive requirements as in the always case.

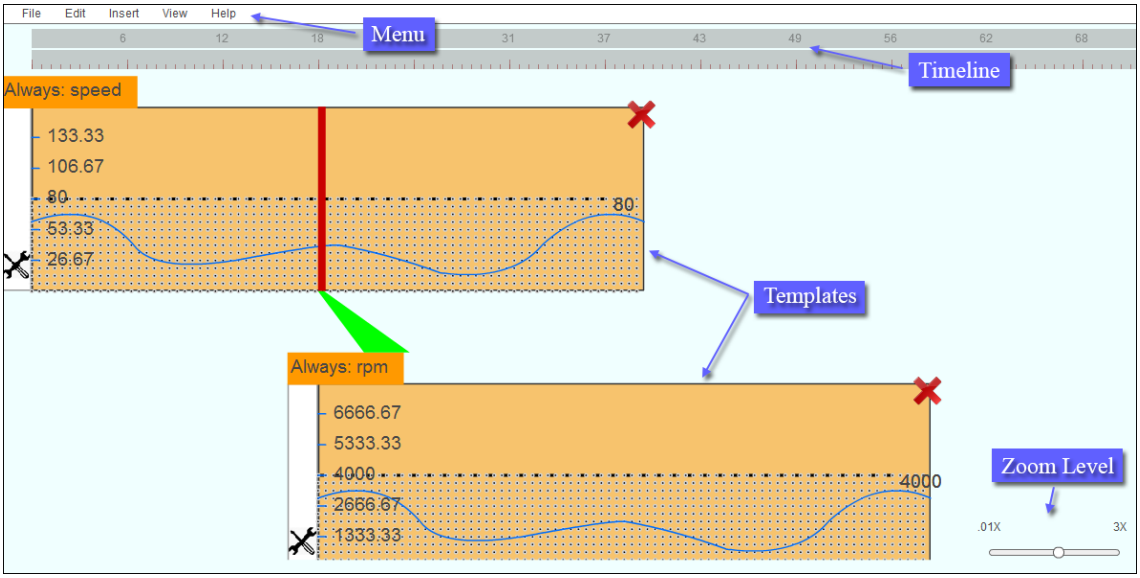


Figure 3.1: Overview of the graphical user interface of the ViSpec tool. The example shown represents the MTL specification $\phi = \square_{[0,40]}((speed < 80) \rightarrow \square_{[0,40]}(rpm < 4000))$.

The main building blocks of the formalism are templates. These are used for defining temporal logic operators, their timing intervals, and the expected signal shape. The user starts with an empty template and a setup assistant presents the user with a sequence of dialog boxes that aid in the development of the template.

The process is context dependent where each option selection leads to a potentially different set of options for the next step.

The first step in the template definition process is to define the temporal operator. Among the choices (and their corresponding MTL symbols) are: *Always* ($\Box$), *At Least Once* ($\Diamond$), *Eventually Always* ($\Diamond\Box$), *Repeatedly Often and Finally* ($\Box\Diamond$), and *now*. The options available enable users to define a wide range of specifications. The following sections will present examples of a subset of formulas that can be generated using this graphical formalism.

After the temporal operator is selected, the user sets the timing bounds for it. Many users might have difficulty defining timing bounds, especially for specifications with temporal operators such as *Eventually Always* ($\Diamond\Box$) and *Repeatedly Often and Finally* ($\Box\Diamond$). To illustrate the process, the tool provides a fill-in-the-blanks sentence format to the user. For example, if the operator *Eventually Always* is selected, the user will have to complete the following sentence with the timing bounds: "Eventually, between ___ and ___ seconds, the signal will become true, and from that point on, will stay true in the next ___ to ___ seconds". The set of timing intervals are visualized with color shaded regions in the template.

The next step in the process is in defining whether the predicate will evaluate to true when the signal is above or below a set threshold. For example, for the *Always* ($\Box$) operator, a signal is selected that is either always above or below a specified threshold. Once either option is selected, various signals that fit the requirement are automatically generated and presented visually. Instead of drawing the signal, the user will select from one of the generated options. Consider the following example:

Example 3.2.1 *A specification from the fragment of MTL formulas called Safety MTL specifications is presented. Specifically, the specification* $\phi_1 = \Box_{[0,36]}(rpm < 4000)$. *The formula states that in the next 36 seconds, engine speed should always be*

less than 4000. The corresponding graphical formalism for this formula is presented in Fig. 3.2. Note that, in regard to the specification, the signal can be of any shape as long as it is always below the 4000 threshold.

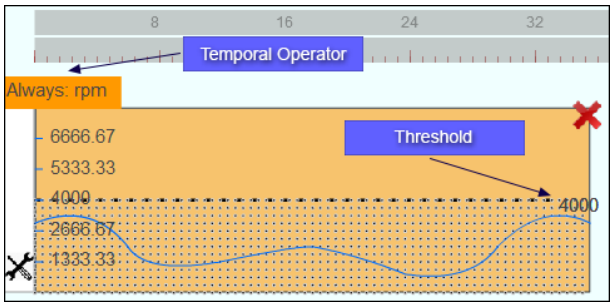


Figure 3.2: Example 3.2.1: A graphical representation for the *Safety* MTL specification $\phi_1 = \Box_{[0,36]}(rpm < 4000)$.

Consider the following example for the *At Least Once* ($\Diamond$) temporal operator:

Example 3.2.2 *A specification from the fragment of MTL formulas called Reachability MTL specifications is presented. Specifically, the specification* $\phi_2 = \Diamond_{[0,39]}(speed > 100)$. *The formula states that eventually, within the next 39 seconds, the vehicle speed will go over 100. The corresponding graphical formalism for this formula is presented in Fig. 3.3. Again, in regard to the specification, the signal can be of any shape as long as at one point, within the timing bounds of the temporal operator, it is above the 100 threshold.*

For the *Eventually Always* ($\Diamond\Box$) operator, at least once in the timing interval of the eventually operator, the signal should go above the threshold and stay there for the entire timing interval of the always operator. Two types of shading will indicate the timing bounds of the MTL operators.

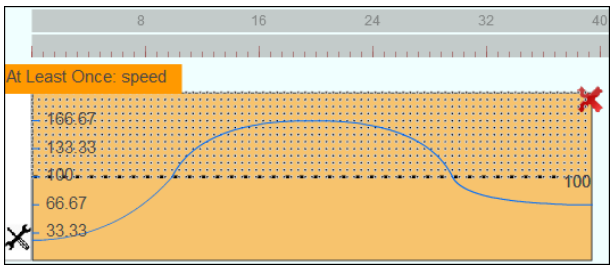


Figure 3.3: Example 3.2.2: The graphical formalism for the *Reachability* MTL specification $\phi_2 = \Diamond_{[0,39]}(speed > 100)$.

Example 3.2.3 *Consider the specification* $\phi_3 = \Diamond_{[0,30]}\Box_{[0,10]}(speed > 100)$. *The formula states that at some point in the first 30 seconds, the vehicle speed will go over 100 and stay above for 10 seconds. The corresponding graphical formalism for this formula is presented in Fig. 3.4.*

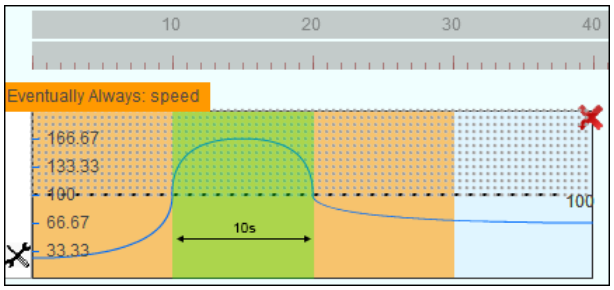


Figure 3.4: Example 3.2.3: The graphical formalism for the MTL specification $\phi_3 = \Diamond_{[0,30]}\Box_{[0,10]}(speed > 100)$.

For the *Repeatedly Often and Finally* ($\Box\Diamond$) operator, an oscillating signal is presented where two types of shading indicate the timing intervals for each MTL operator. Consider the following example:

Example 3.2.4 *The specification* $\phi_4 = \Box_{[0,30]}\Diamond_{[0,10]}(speed > 100)$ *is presented. The formula states that at every timestep of the simulation in the first 30 seconds, the speed*

will go over 100 within the next 10 seconds. The corresponding graphical formalism for this formula is presented in Fig. 3.5. No matter how far to the left or right the green shaded region is moved, contained within the orange region, there is always a point where the signal is above the threshold. Recall that the displayed signal is automatically generated so that it satisfies the options previously selected.

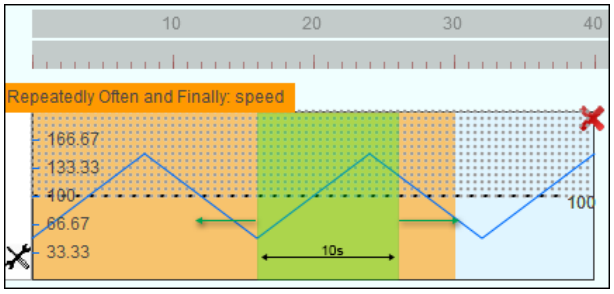


Figure 3.5: Example 3.2.4: A graphical representation for the MTL specification $\phi_4 = \Box_{[0,30]}\Diamond_{[0,10]}(speed > 100)$.

The next important concept in this graphical formalism is the relationship between templates.

First, the sequence relationship between two templates is presented. Assume that the first template is already created. If another template is added below it, then an order in the execution of the events is defined. The second template is only considered if the first template is evaluated to true. Formally, there is an implication relationship from the first template to the second. Consider the following example:

Example 3.2.5 *The specification* $\phi_5 = (\Diamond_{[0,40]}(speed > 100)) \rightarrow (\Diamond_{[0,30]}(rpm > 3000))$ *is presented. The formula states that if, within 40 seconds, the vehicle speed is above 100 then within 30 seconds from time 0, the engine speed should be over 3000. The corresponding graphical formalism for this formula is presented in Fig. 3.6.*

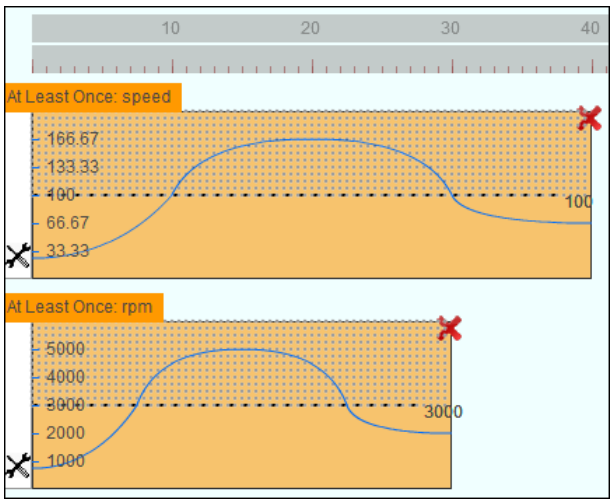


Figure 3.6: Example 3.2.5: A graphical representation for the MTL specification $\phi_5 = (\Diamond_{[0,40]}(speed > 100)) \rightarrow (\Diamond_{[0,30]}(rpm > 3000))$.

A second type of relationship enables the user to establish conjunction between two events. To achieve this, templates can be grouped. This is indicated by a bold black box. Doing so requires that both templates evaluate to true. Consider the following example:

Example 3.2.6 *Specification* $\phi_6 = (\Box_{[0,40]}(speed < 100)) \wedge (\Box_{[0,40]}(rpm < 4000))$. *The formula states that, within 40 seconds, the vehicle speed should be less than 100 and the engine speed should be under 4000. The corresponding graphical formalism for this formula is presented in Fig. 3.7.*

The third type of template relationship enables the user to establish relative timing between two templates. Consider the following example:

Example 3.2.7 *Specification* $\phi_7 = \Box_{[0,40]}((speed < 80) \rightarrow \Box_{[0,40]}(rpm < 4000))$. *Here, the nested specification* $\Box_{[0,40]}(rpm < 4000)$ *is evaluated every time* $(speed < 80)$ *is true. This formula is represented in the formalism with nested templates, otherwise*

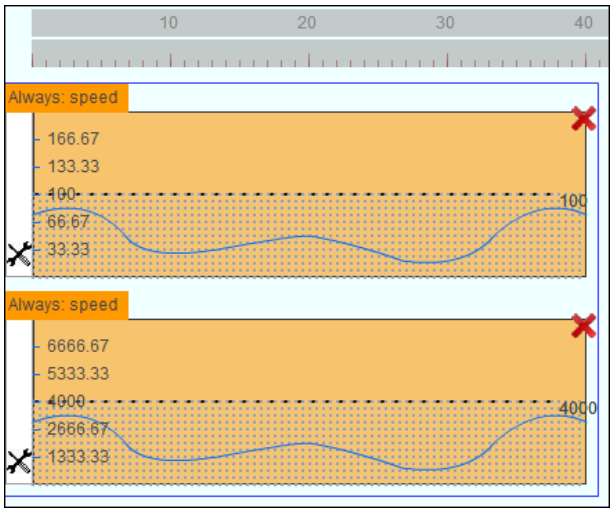


Figure 3.7: Example 3.2.6: A graphical representation for the MTL specification $\phi_6 = (\square_{[0,40]}(speed < 100)) \wedge (\square_{[0,40]}(rpm < 4000))$.

referred to as parent and child templates. The second template is tabbed and connected to the first template using a green indicator. In the GUI, such a nested template is initiated by clicking on the signal of the parent template. The corresponding graphical formalism is presented in Fig. 3.8.

The variety of templates and the connections between them allow users to express a wide variety of specifications.

3.3 Graphical Formalism

The specification development process in ViSpec is divided in two sub processes. First, given a user input in the ViSpec tool, it is translated to a tree structure where the nodes contain template information such as temporal operators, their corresponding timing parameters, group and the value threshold for the predicates. Secondly, the generated tree structure is traversed by a recursive algorithm to generate the MTL

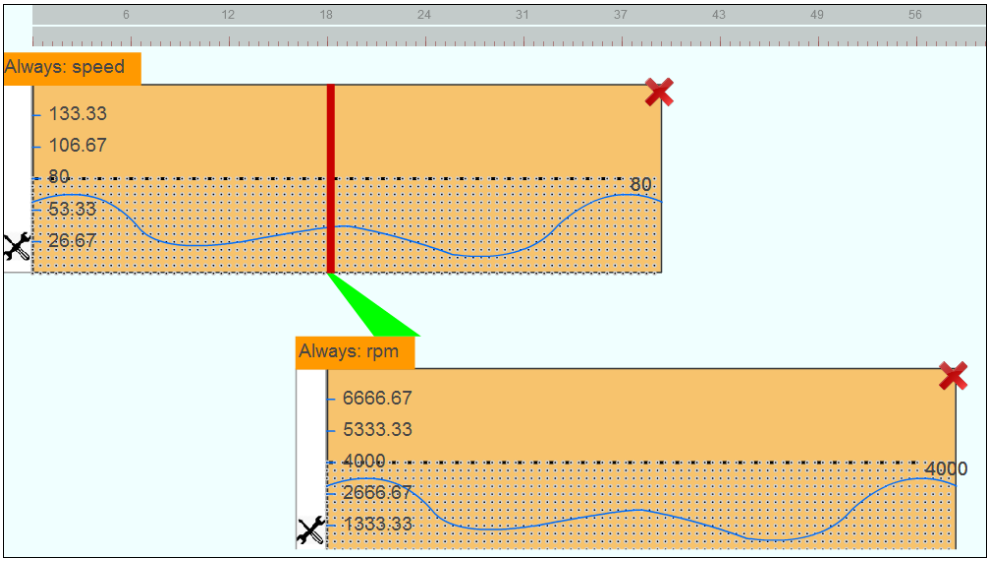


Figure 3.8: Example 3.2.7: A graphical representation for the MTL specification $\phi_7 = \Box_{[0,40]}((speed < 80) \rightarrow \Box_{[0,40]}(rpm < 4000))$.

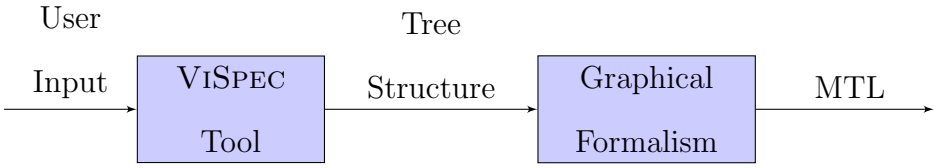


Figure 3.9: The specification development process using ViSpec

formula. There is a bijection between the visual representation of a specification and the MTL formula. An overview of the process is provided in Fig. 3.9.

An example of the tree structure for MTL formula $\phi = \Box(a \wedge \Diamond b) \rightarrow (\Box c \wedge \Diamond(d \rightarrow (a \wedge \Box b)))$ is shown in Fig. 3.10. The recursive algorithm for traversing the tree structure and generating the MTL formula is presented in Alg. 1. Note that the function addParenConn{A,B,C,D} add the parenthesis and connectives between predicates.

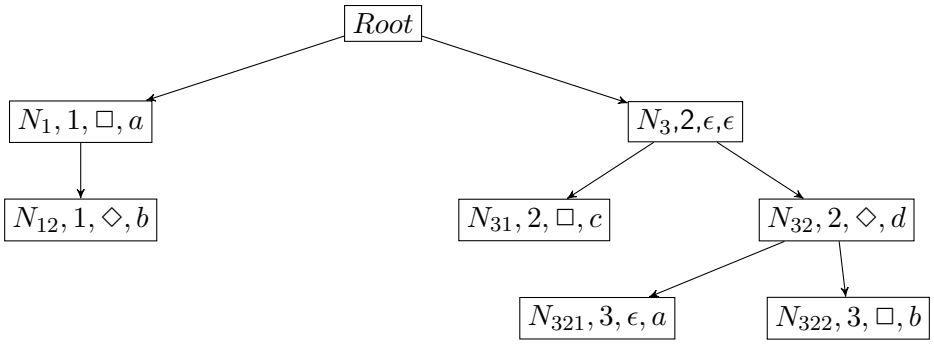


Figure 3.10: The corresponding tree structure for specification $\phi = \Box(a \wedge \Diamond b) \rightarrow (\Box c \wedge \Diamond(d \rightarrow (a \wedge \Box b)))$ where a, b, c and d are predicates. Each node is composed of a node name, group number, temporal operator, and predicate. The symbol ϵ indicates empty parameters.

3.4 Debugging Specifications

Through the guided process of developing templates, we avoid various syntactic issues with the generated MTL specifications. However, we cannot guarantee that the generated specification correctly captures the intention of the system engineer. To gain confidence on the generated specification, in [54], we present a debugging framework that operates over the ViSpec tool. The debugging framework is shown in Fig. 3.11. Once the user develops a specification in ViSpec, it is converted to an MITL specification and passed on to the debugger. The debugger checks for three types of errors:

- Validity: whether the specification is unsatisfiable or a tautology.

- Redundancy: whether the formula has redundant conjuncts.

- Vacuity: whether any of the sub formulas do not affect the satisfiability of the specification.

Algorithm 1 WriteMTL - Algorithm for generating the MTL formula given a tree structure of the graphical formalism

Input: Tree Structure $T = \langle V, E \rangle$ where $v \in V$ and $v = \langle G, Op, S \rangle$ where G is the group, Op is the temporal operator and S is the predicate string; formula ϕ.
Output: ϕ

```
1: function writeMTL(T, φ)
2:     C ← T.getChildren().
3:     sC ← size(C)
4:     for node i in C do
5:         φ ← conc(φ, i.Op)
6:         if i.isParent then
7:             if not(i.S.isEmpty) then
8:                 subC ← t.getChildren(i)
9:                 φ ← addParenConnA(φ, subC)
10:                φ ← writeMTL(i.subtree, φ)
11:                φ ← addParenConnB(φ, subC)
12:            else
13:                φ ← conc(φ, '(')
14:                φ ← writeMTL(i.subtree, φ)
15:                φ ← addParenConnC(sC, φ)
16:            end if
17:        else
18:            φ ← conc(φ, i.S)
19:            φ ← addParenConnD(φ, sC)
20:        end if
21:    end for
22: end function
```

As shown in Fig. 3.12, if an issue is detected, the formula is returned to the user for revision. Another tool that enables the validation of the elicited requirements is STLInspector [125]. The tool computes a set of representative signals which can be compared to several mutated variants. The practitioner, through visual inspection, can gain confidence on the correctness of the specification.

3.5 Usability Study

3.5.1 Hypotheses

The aim of the study is to evaluate whether ViSpec enables users to develop formal specifications. Two groups were considered:

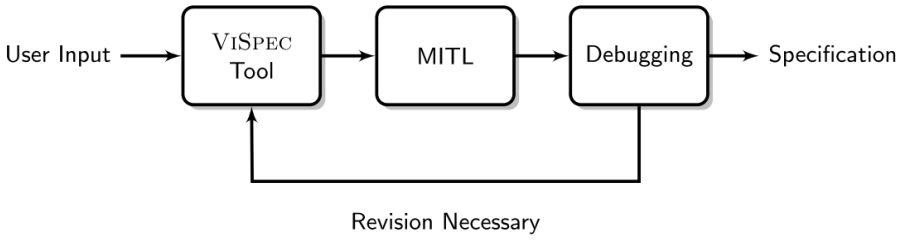


Figure 3.11: The debugging process in ViSpec.

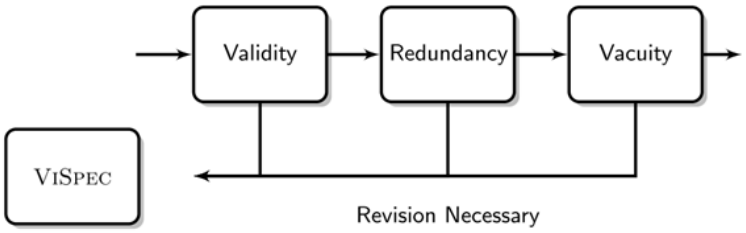


Figure 3.12: Three debugging steps with ViSpec.

1. Non-expert users: These are users who declared that they have no experience in working with requirements.

2. Expert users: These are users who declared that they have experience working with system requirements. Note that they do not necessarily have experience in writing requirements using formal logics.

Some of the interesting questions we wanted to investigate, which are also presented as hypotheses in Table 3.2, are:

- Whether the graphical formalism enables non-experts and experts to formalize requirements accurately.

- How well the expert cohort performs in comparison to the non-expert cohort.

- How user friendly and easy-to-use ViSpec is.

Table 3.2: Hypotheses and test results with level of significance $\alpha = 0.05$. User groups are defined in Section 3.5.2.

Alternative Hypothesis		Reject **Null** Hypothesis
$1a$	Non-expert users are able to define formal requirements accurately using formal logics such as MTL.	†
$1b$	Non-expert users are able to define formal requirements accurately using the Visual Specification Tool.	Yes
$2a$	Expert users from the industry are able to define formal requirements accurately using formal logics such as MTL.	⋆
$2b$	Expert users from the industry are able to define formal requirements accurately using the Visual Specification Tool.	Yes
3_{alt}	The mean grade per user for expert users is greater the mean grade per user for non-expert users.	Yes
Tx_{alt}	The mean grade per task x for industry users is greater than to the mean grade per task x for non-expert users.	Partially

†: We assume that we can reject Hypothesis $1a$ based on our informal experience.

⋆: Preliminary results on Hypothesis $2a$ presented in Section 3.7.

Writing formal requirements is a challenging task that requires a significant amount of training. Therefore, it is safe to assume that we can reject Hypothesis $1a$ as supported by our informal experience. Hypothesis $2a$ will be tested in a future work. In addition, we analyze user interaction and behavior to measure the ease-of-use of the tool. Note that in Table 3.2, for each hypothesis, we show whether the **null** hypothesis is rejected.

3.5.2 Demographics

The non-expert cohort was comprised of twenty subjects from the student community of Arizona State University. Most of the subjects are from an engineering

background with little to no experience working with requirements. The student demographics are presented in Table 3.3.

The expert subject cohort was comprised of ten subjects from the industry in the Phoenix area. The subjects have experience working with specifications and come from an engineering background.

Table 3.3: Hypothesis 1*b* Subject Demographics

Freshman	2	Computer Science	5	Male	12
Sophomore	2	Software Engineering	3	Female	8
Junior	5	Electrical Engineering	3		
Senior	5	Mechanical Engineering	6		
Masters	4	Engineering, other	3		
PhD	2				

3.5.3 Experimental Design

Each subject received a task list to complete. The task list contained ten tasks related to automotive system specifications. Each task asked the subject to formalize a natural language specification through ViSpec and generate an MTL formula. The list of tasks is presented in Table 3.4.

The tasks become more complex throughout the session. The higher the number of the task, the more steps necessary to complete the task successfully.

Each session was at most 45 minutes long. Subjects received a one minute and thirty second tutorial on using ViSpec to develop specifications. The computer screen was recorded and actions were logged for each session. The subjects also completed a demographic and post-completion questionnaire.

Table 3.4: Task list with automotive system specifications presented in natural language.

Task	Natural Language Specification
1. Safety	In the first 40 seconds, vehicle speed should always be less than 160.
2. Reachability	In the first 30 seconds, vehicle speed should go over 120.
3. Stabilization	At some point in time in the first 30 seconds, vehicle speed will go over 100 and stay above for 20 seconds.
4. Recurrence	At every point in time in the first 40 seconds, vehicle speed will go over 100 in the next 10 seconds.
5. Recurrence	It is not the case that, for up to 40 seconds, the vehicle speed will go over 100 in every 10 second period.
6. Implication	If, within 40 seconds, vehicle speed is above 100 then within 30 seconds from time 0, engine speed should be over 3000.
7. Request Response	If, at some point in time in the first 40 seconds, vehicle speed goes over 80 then from that point on, for the next 30 seconds, engine speed should be over 4000.
8. Conjunction	In the first 40 seconds, vehicle speed should be less than 100 and engine speed should be under 4000.
9. Non-strict sequencing	At some point in time in the first 40 seconds, vehicle speed should go over 80 and then from that point on, for the next 30 seconds, engine speed should be over 4000.
10. Long sequence	If, at some point in time in the first 40 seconds, vehicle speed goes over 80 then from that point on, if within the next 20 seconds the engine speed goes over 4000, then, for the next 30 seconds, the vehicle speed should be over 100.

3.5.4 Metrics

Two metrics are used for performance evaluation:

Task completion: this is a binary measure, which indicates whether users were able to finish the task within the set time.

Measure of Accuracy: a value from one to five which is used to quantify the accuracy of subject generated formulas. The formulas are graded by formal specification experts which were given the following two suggested criteria: a) How accurate the meaning of the natural language specification is captured, and b) Whether the inaccuracies in the user submitted formula can be easily debugged and corrected in the testing and verification process. Furthermore, in order to decrease subjectivity, the following instructions were given to the expert graders in order to anchor the meanings of the five different grades of the scale used: A grade of one indicates that the generated formula is totally inaccurate. A grade of two indicates that the formula is mostly inaccurate. A grade of three indicates an inaccurate formula which can be easily debugged and corrected to the proper formal logic specification by formal specification experts and, thus, this is the minimum acceptable satisfactory result. A grade of four indicates that the formula is inaccurate but can be debugged and improved by automated specification debugging tools. A grade of five indicates that the generated formula is completely accurate. The group of expert graders consisted of experts in formal methods and logic.

3.6 Results

Average grade per task

For both cohorts, the task performance is presented in Fig. 3.13. It can be observed that overall, the mean grade per task for both cohorts is high. Consider the mean

grade per task as a random variable $\bar{X}$. Specifically, $\bar{X} : \Omega \rightarrow \mathbb{R}$, where $\Omega \in \{y : 1 \leq y \leq 5\}$. In Figure 3.14, we present the survival function $S_{\bar{X}}(x) = 1 - F_{\bar{X}}(x) = 1 - P(\bar{X} \leq x) = P(\bar{X} > x)$ based on sample data. Note that x is the threshold of mean grade accuracy.

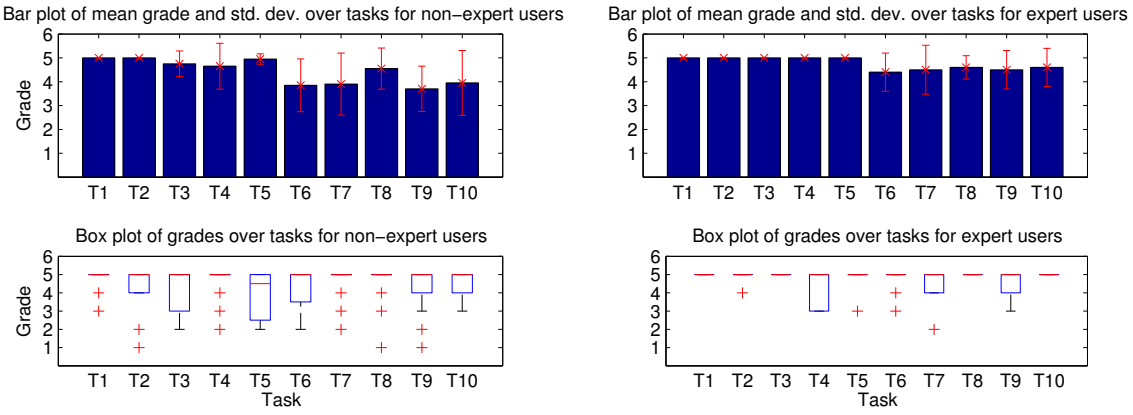


Figure 3.13: Subject accuracy grades over tasks for both the expert and non-expert cohorts.

Hypothesis Testing

In the following analysis, we will utilize t-tests [133, Chapter 6] to conduct hypothesis testing. The t-test is appropriate in our case since we will compare the population means of only two groups. In order to utilize such a method, we need to ensure that the population from which the sample is drawn is normally distributed. To test for normality of the underlying distribution of the data, we utilize the Kolmogorov-Smirnov test, the Chi-square g.o.f test, and the Anderson-Darling test. For more information on these tests, the reader is referred to [97, Chapter 16].

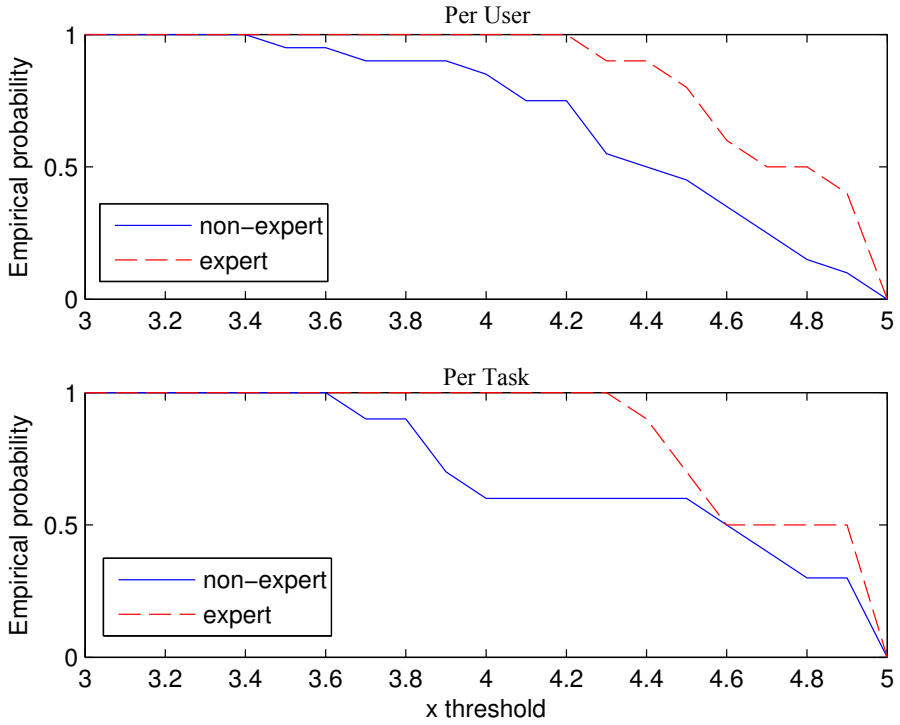


Figure 3.14: **Top**: The empirical probability that the mean grade per user is greater than threshold x for the non-expert and expert subjects, i.e., $P(\bar{Y} > x)$.

Bottom: The empirical probability that the mean grade per task is greater than threshold x for the non-expert and expert subjects, i.e., $P(\bar{X} > x)$.

Hypothesis $1b$

To test Hypothesis $1b$, we need to establish what is an acceptable threshold for accuracy in order to test the hypothesis. As discussed in the metrics section, we claim that a mean grade higher than three is an acceptable threshold for non-expert users. Therefore, the null hypothesis for hypothesis $1b$ is: the mean grade per user is less than or equal to three for non-experts.

Let us define the average grade per user as a random variable $\bar{Y}$. Specifically, $\bar{Y} : \Omega \rightarrow \mathbb{R}$, where $\Omega \in \{y : 1 \leq y \leq 5\}$. The sample data from 20 subjects has a mean grade of 4.43 and standard deviation of 0.41. We test for normality with

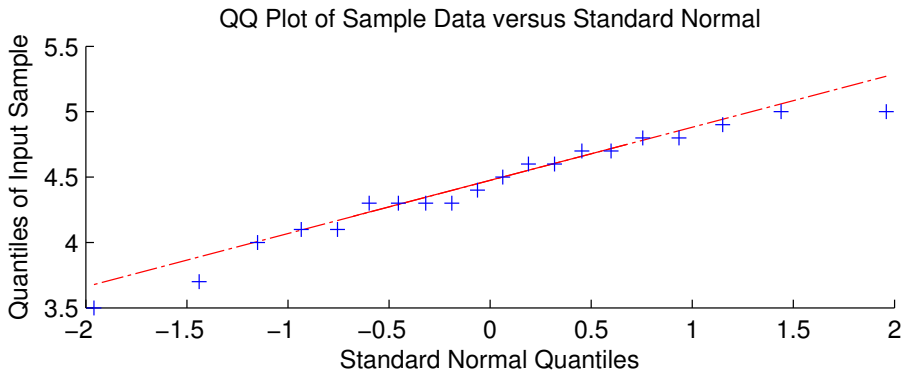


Figure 3.15: Q-Q plot for the non-expert sample data with respect to the normal distribution.

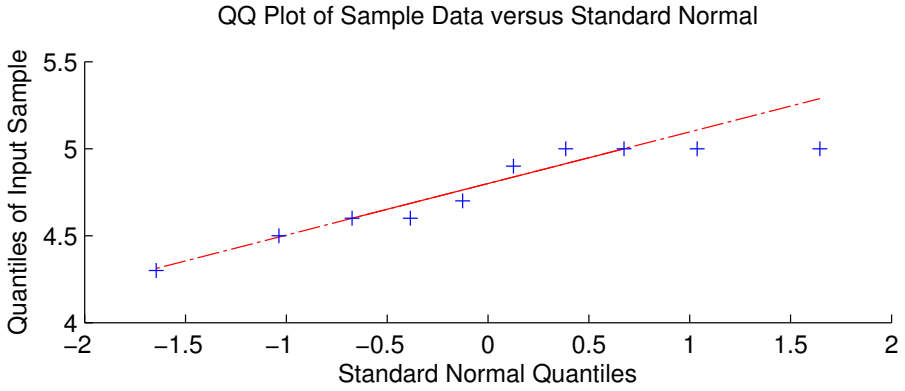


Figure 3.16: Q-Q plot for the expert sample data with respect to the normal distribution.

the Kolmogorov-Smirnov test, the Chi-square g.o.f test, and the Anderson-Darling test and all three fail to reject the null hypothesis that the data follows the normal distribution. Therefore, we cannot conclude that the data does not follow the normal distribution. In Fig. 3.15, we present the Q-Q plot between the non-expert sample data and the normal distribution. If we assume that the data constitute a random sample from a normal distribution, i.e. $\bar{Y} \sim \mathcal{N}$, we can use the t-statistic to test the hypothesis. We reject the null hypothesis with a p-value ≈ 0.

Hypothesis $2b$

Similarly, we test Hypothesis $2b$ for the expert cohort. The null hypothesis for hypothesis $2b$ is: the mean grade per user is less than or equal to three for expert users. We test for normality as in the previous case and all three test fail to reject the null hypothesis that the data follows the normal distribution.

Consider the average grade per user as a random variable $\bar{Z}$. Specifically, $\bar{Z} : \Omega \to \mathbb{R}$, where $\Omega \in \{y : 1 \leq y \leq 5\}$. The sample data from 10 subjects has a mean grade of 4.76 and standard deviation of 0.26. In Fig. 3.16, we present the Q-Q plot between the expert sample data and the normal distribution. If we assume that the data constitute a random sample from a normal distribution, i.e. $\bar{Z} \sim \mathcal{N}$ we can use the t-statistic to test the hypothesis. We reject the null hypothesis with a p-value ≈ 0.

Hypothesis 3_{alt}

To test Hypothesis 3_{alt}, we conduct a two sample t-test. The p-value returned from the test is 0.0024 for a significance level of 0.01, we reject the null hypothesis. Therefore we claim that the mean grade per user for expert users is greater than the mean grade per user for non-experts.

Hypothesis Tx

Next, we compare the mean grade of both cohorts in regards to each task. A two sample t-test is conducted for each task. The results for the tests are presented in Table 3.5. Task 9 is the most difficult task when it comes to the number of errors generated, and this is the only task where there is a clear difference in performance between the expert and non-expert cohorts. For hypotheses Tx_4, Tx_6, Tx_7, and Tx_{10} the p-values are between 0.05 and 0.10 and therefore need more investigation.

Table 3.5: Hypothesis testing of Tx_{null} with $\alpha = 0.05$.

x	Reject Tx_{null}	p-value	Conclusion
4	No	0.065	potentially true with more investigation
5	No	0.165	false
6	No	0.074	potentially true with more investigation
7	No	0.100	potentially true with more investigation
8	No	0.424	false
9	Yes	0.016	true
10	No	0.063	potentially true with more investigation

We observe that the only null hypothesis rejected is for task nine indicating that the mean grade for expert users is greater than the mean grade for non-expert users. The subject accuracy grades over tasks for is shown in Fig. 3.13.

Ease-of-use analysis

One indicator for the ease-of-use of the application is the total time spent per task. As can be observed in Fig. 3.17, the mean time spent per task on average is at most 167 seconds. For easier identification of points of difficulty, we divided each task into subtasks. It was observed that there is no correlation between the length of time spent in a subtask and correctness. This potentially indicates, as also verified by correlation testing between times and grades, that the subjects were unaware of mistakes in the process. From these and other observations, such as misclicks, and subject feedback, we have developed a set of refinements on the tool to improve the user experience. A partial list of improvements is presented in Table 3.6.

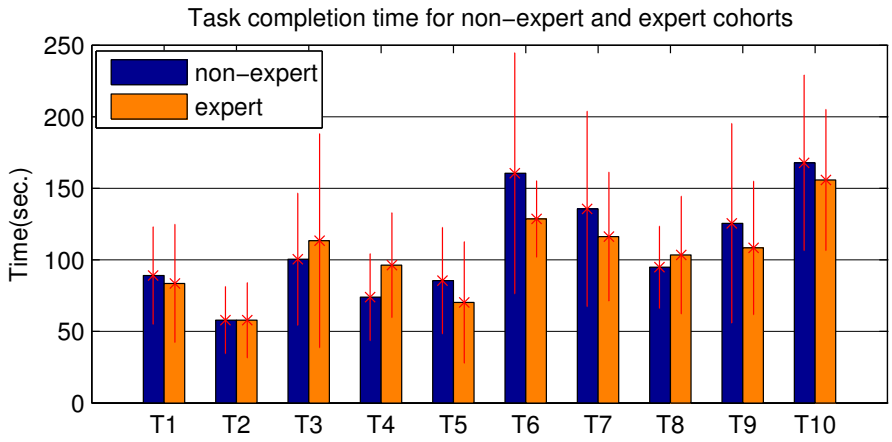


Figure 3.17: Task completion time for non-expert and expert cohorts.

Table 3.6: ViSpec improvements.

#	Improve...	Prime Indicators
1.	the process of creating child templates	misclicks, user feedback
2.	the tutorial by placing more emphasis on the difference between implication and conjunction between templates	task accuracy grade
3.	the visual representation of grouped templates	task accuracy grade, user feedback

3.7 Preliminary Results on Hypothesis 2a

As an ongoing process, to test Hypothesis 2a, we have been collecting online data through a survey [2] from the members of the academic community who have experience working with formal methods (self-characterization).

Twelve subjects participated in the survey. The subjects were asked to translate the same natural language requirements listed in Table 3.4 to MTL. From the responses submitted, five MTL specifications were incorrect. The incorrect specifications include vacuity, validity and redundancy errors [55]. For example, for the natural language specification "At some time in the first 30 seconds, the vehicle speed (v) will go over 100 and stay above 100 for 20 seconds" an expert in formal methods provided the MTL specification $\phi = \Diamond_{[0,30]}((v > 100) \rightarrow \Box_{[0,20]}(v > 100))$. The specification is not correct. In fact, in [55], it is proven that it is a tautology. In other words, it is evaluated to true irrespective of system behavior.

Preliminary results indicate that even subjects with MTL knowledge can produce inaccurate specifications. However, more data is necessary to reach conclusive results on this hypothesis. Considering the fact that the formalism is utilized in the development of safety-critical systems, it is concerning that several errors such as the aforementioned one are submitted by knowledgeable users in the formal requirements area.

3.8 Applications

3.8.1 Robotic Surgery

In the last few decades, there has been a significant increase in the number of robotics systems, especially in the health care system. They have been successfully

[2] Survey may be accessed at: https://sites.google.com/a/asu.edu/s-taliro/mtl-online-survey

introduced in multiple areas such as rehabilitation, telesurgery, physical therapy, elderly care, and remote physician care. In the following, we will focus on autonomous robotic systems for surgery where of paramount importance is the safety of these systems [101]. Specifically, we will consider a model of a robotic serial link manipulator as presented in [111].

One of the main tasks in surgery is the puncturing action. The high precision and repeatability of the process, make robot systems ideal for this task. Also, the trauma induced around the region is much lower and therefore the recovery process for the patient is quicker. To complete the puncturing action, the robot has to move towards the puncturing location. Test the tissue for various indicators to calibrate for optimal puncture, bring the puncturing needle to a perpendicular position and, finally, puncture with correct force and angle. If the force or angle is miscalculated, it might pose unintended harm to the patient. Consider the specifications from [111] that should hold on a serial manipulator for puncturing:

1. From [111]: The force applied to the patient by the end effector is always less than a given threshold, except for the puncturing subtask. Formally, assuming that the operation time is 30 seconds, we have: $\phi_{s1} = \Box_{[0,30]}(\neg puncturing \rightarrow f \leq f_{max})$.

2. From [111]: The task is feasible, and the position of the needle once it stops is inside the target region R. Formally, assuming that the operation time is 40 seconds, we have: $\phi_{s2} = \Diamond_{[0,40]}(Stop \wedge needle \in R))$.

3. Also, other requirements can be expressed for such a system. For example, the end effector speed should not be less than v_{min} and should not be greater than v_{max}. Formally: $\phi_{s3} = \Box_{[0,40]}(v_{min} < v_{eff} < v_{max})$

4. The VISPEC tool is utilized to develop the specifications for the robotic manipulator. For ϕ_{s1}, the specification is presented in Fig. 3.18. We assume that $f_{max} = 10$. For ϕ_{s2}, the specification is presented in Fig. 3.20. We assume that $needle \in R \iff 5 < n_x < 10 \wedge 5 < n_y < 10$, where n_x, n_y are the x and y coordinates for the needle. For ϕ_{s3}, the specification is presented in Fig. 3.19. We assume that $v_{min} = 10$ and $v_{max} = 20$.

3.8.2 Quadcopter

In recent years, quadcopters and other unmanned aerial vehicles (UAVs) have become a major focus for research both in the academic community and industry. Among others, they are used in military operations, nuclear disaster assessment, firefighting and entertainment. The challenges faced in developing these devices and their control algorithms come from the flight dynamics and the highly dynamical environment that they operate in. Also, as the complexity of these devices increases, so do the performance and reliability requirements.

Consider the following specifications for a quadrotor:

1. The absolute value of the pitch and roll angle should always be bellow certain thresholds. Formally, assuming that the operation time is 40 seconds, we have: $\phi_{q1} = \Box_{[0,40]}(|\alpha| < \alpha_{max}) \wedge \Box_{[0,40]}(|\beta| < \beta_{max})$.

2. If distance to the target region is smaller than a certain threshold d, then for then next 20 seconds, the speed should not exceed v_{max}. Formally, assuming that the operation time is 40 seconds, we have: $\phi_{q1} = \Box_{[0,40]}(dist < d \rightarrow \Box_{[0,20]}(v < v_{max}))$.

3. The VISPEC tool is utilized to develop the specifications for the quadrotor. For ϕ_{q1}, the specification is presented in Fig. 3.21. We assume that $\alpha_{max} = 45\,\text{deg}$,

$\beta_{max} = 45\,\text{deg}$ and $\gamma_{max} = 60\,\text{deg}$. For ϕ_{s3}, the specification is presented in Fig. 3.19. We assume that $v_{min} = 10$ and $v_{max} = 20$. For ϕ_{q2}, the specification is presented in Fig. 3.22. We assume that $d = 5$ and $v_{max} = 10$.

3.9 Related works

In order to help address the formal specification challenge, various graphical formalisms have been studied in the past [137, 8, 103, 22, 149, 138]. The most relevant works appear in [22] and [149]. In [22], the authors extend Message Sequence Charts and UML 2.0 Interaction Sequence Diagrams to propose a scenario based formalism called Property Sequence Chart (PSC). The formalism is mainly developed for specifications on concurrent systems. In [149], PSC is extended to Timed PSC which enables the addition of timing constructs to specifications.

In terms of usability studies for formal requirements very few works exist. In [143], the authors study the ability of expert users to develop requirements in Z. A related usability study for requirement representation is presented in [108], where the authors present and evaluate a system for generating, troubleshooting and executing controllers for robots using natural language.

3.10 Conclusion and Future Work

As robots and other cyber-physical systems become more complex and ubiquitous, so does the need for better testing and verification. A set of formal methods that improve this process require some formal representation of system specifications. In this work, a graphical formalism and a tool that enables users to easily develop formal specifications are presented. The ViSpec tool enables users who have little to no mathematical training in formal logics to develop formal specifications, as was verified by a usability study that was conducted in order to evaluate the usefulness

of the tool and to get insights on potential improvements. The tool was utilized to formalize specifications for two robots.

Last but not least, we would like to investigate if the potential inaccuracies of the specifications that users generate with the tool can be attributed mainly to the inherent ambiguity of the natural language descriptions which were given, or if not, which other factors contribute and to what extent. Thus, in an improved usability study, we aim towards exploring alternative methods of generation of requirements from engineers for a system, that do not involve the administration of a natural language description by the experimenter. This would enable us to study to what extent inherent natural language ambiguity causes the observed less-than-perfect accuracy that is sometimes, even if rarely, exhibited.

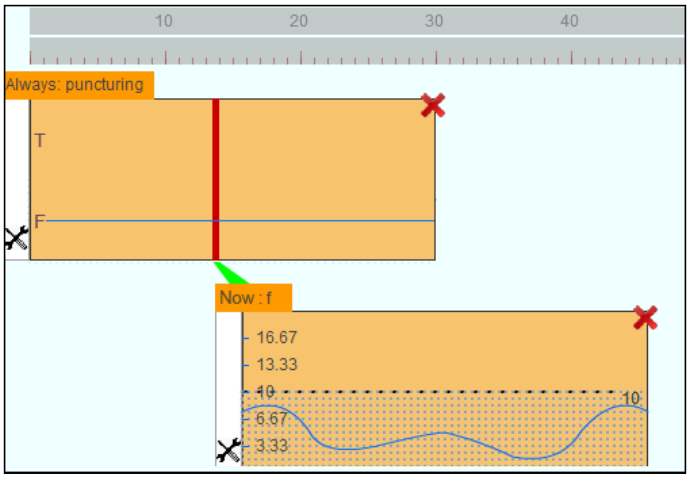


Figure 3.18: The graphical formalism for ϕ_{s1}.

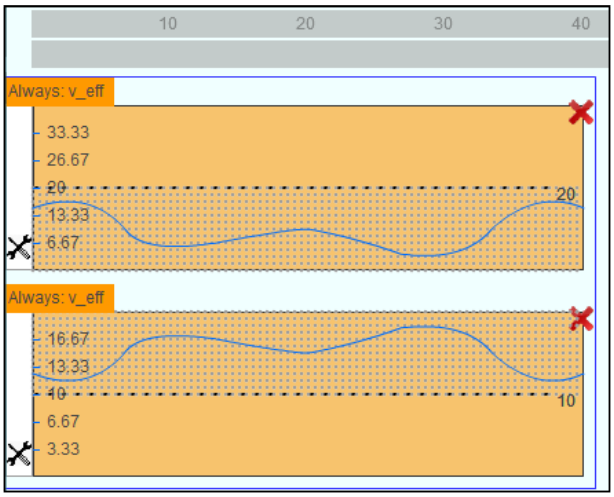


Figure 3.19: The graphical formalism for ϕ_{s3}.

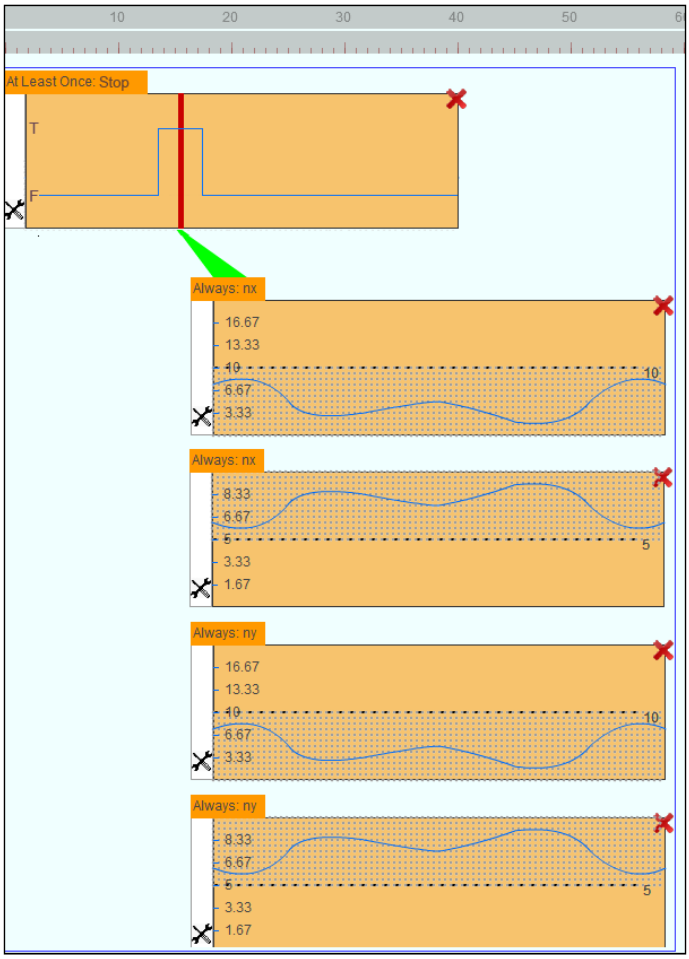


Figure 3.20: The graphical formalism for ϕ_{s2}.

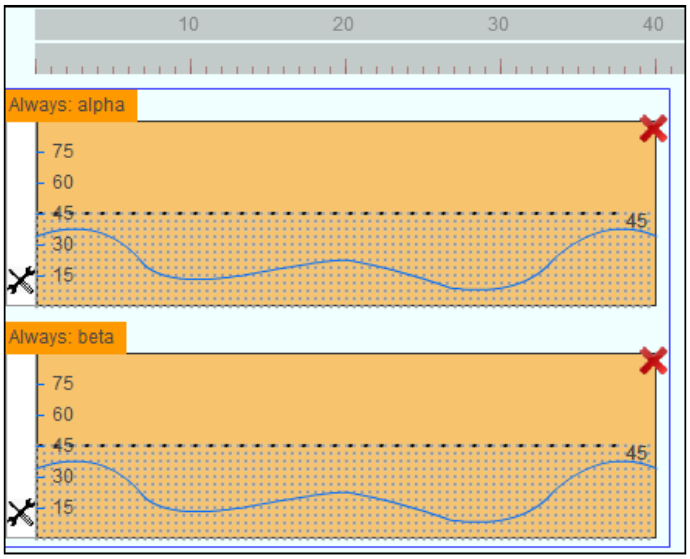


Figure 3.21: The graphical formalism for ϕ_{q1}.

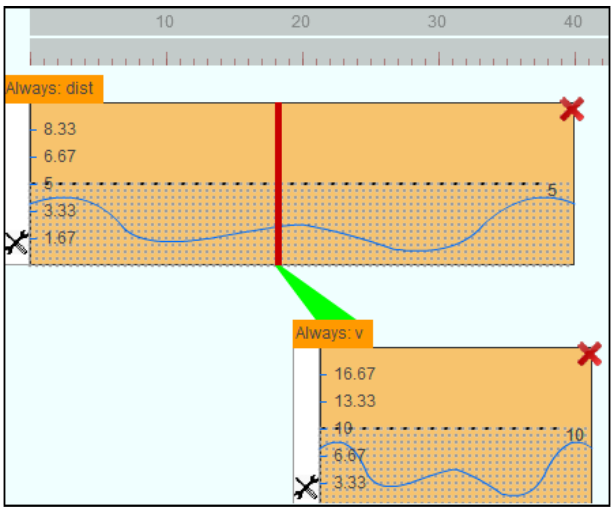


Figure 3.22: The graphical formalism for ϕ_{q2}.

Chapter 4

PARAMETER MINING OF PARAMETRIC MTL SPECIFICATIONS

4.1 Introduction

In this chapter, we consider the problem of multiple parameter mining of parameterized MTL specifications. Our high-level goal is to explore and infer properties that a system satisfies. We assume that the system designer has partial understanding about the properties that the system satisfies (or does not satisfy) and would like to be able to determine these properties precisely. In particular, we assume that the system developer can formalize system properties in MTL, where some parameters are unknown. Such parameters could be unknown threshold values for the continuous state variables of the system or some unknown real-time constraints.

In this work, we present a framework for multiple parameter mining and analysis of parametric MTL specifications. Such an exploration framework would be of great value to the practitioner. The benefits are twofold. One, it allows for the analysis and development of specifications. In many cases, system requirements are not well formalized by the initial system design stages. Two, it allows for the analysis and exploration of system behavior. If a specification can be falsified, then it is natural to inquire for the range of parameter values that cause falsification. That is, in many cases, the system design may not be modified, but the guarantees provided should be updated.

As the number of parameters in the specification increases so does the complexity of the resulting optimization problem. In the case of single parameter mining, the solution of the problem is a one-dimensional range. With multiple parameters, finding

a solution to the problem becomes more challenging since the optimization problem is converted to a multi-objective optimization problem where the goal is to determine the Pareto front [112]. To solve this problem, we present a method for effective one-sided exploration of the Pareto front and provide a visualization method for the analysis of parameters. The algorithms presented in this work are incorporated in the testing and verification toolbox S-TaLiRo [19, 126]. For an overview of the toolbox see [82]. Finally, we demonstrate our framework on a challenge problem from the industry on an industrial scale model and present experimental results on several benchmark problems.

Even though our examples and case study are from the automotive domain, our results can be applied to any application domain where Model Based Design (MBD) and temporal logic requirements are utilized, e.g., medical devices [130, 129, 90, 36].

In this chapter:

- The parameter mining problem for multiple parameters is presented.
- The solution to the multiple parameter mining problem is presented as a multi-criterion optimization problem.
- An efficient solution to the optimization problem is presented.
- Two algorithms for the exploration of the Pareto Front are presented.
- The methods presented in this work are demonstrated on an industrial size case study of a high-fidelity engine model.
- An extensive review on related works is presented.

4.2 Problem Formulation

Consider the AT running example from Section 2.6. In the development process for such systems, the practitioner might want to find answers to queries like "What is the shortest time that ω can exceed 3250 RPM" or "For how long can ω be below 4500 RPM". We can also answer queries about the relationships between parameters with regard to system falsification. For example, for the specification "Always the vehicle speed v and engine speed ω need to be less than parameters θ_1, θ_2, respectively". Also, we could ask "If I increase/decrease θ_1 by a specific amount, how much do I have to increase/decrease θ_2 so that the system Σ satisfies the specification?". Formally, the multiple parameter mining problem is defined as follows.

Problem 4.2.1 (MTL m-Parameter Mining) *Given an MTL formula $\phi[\vec{\theta}]$ with a vector of m unknown parameters $\vec{\theta} \in \Theta = [\underline{\vec{\theta}}, \overline{\vec{\theta}}]$ and a system Σ, find the set $\Psi = \{\vec{\theta^*} \in \Theta \mid \Sigma$ does not satisfy $\phi[\vec{\theta^*}]\}$.*

That is, the solution to Problem 4.2.1 is the set Ψ such that for any parameter $\vec{\theta^*}$ in Ψ the specification $\phi[\vec{\theta^*}]$ does not hold on system Σ. In the rest of the document, we refer to Ψ as the parameter falsification domain. An approximate solution for Problem 4.2.1 was presented in [147] for the case where θ is a scalar i.e. the formula has only one parameter. In [147], the solution to the problem returned a parameter with which the falsifying set can be inferred since the parameter range is one-dimensional. An extension for the multiple parameter mining problem was presented in [83]. In the multiple parameter setting, we have a set of possible solutions which we need to explore. That is, the solution to the multi-parameter mining problem is in the form of a Pareto front [112].

We note that the original observation that the falsification domain problem over *a single system output trace* has the structure of a Pareto front is made in [21]. In this

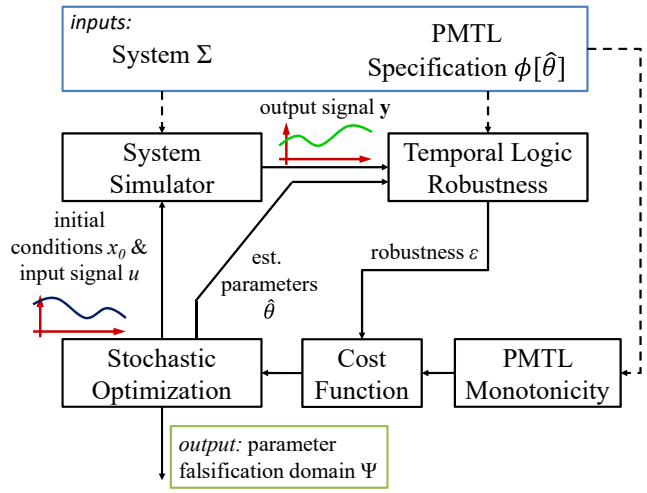


Figure 4.1: Overview of the solution to Problem 4.2.1, the PMTL parameter mining problem for CPS.

work, we observe that the falsification domain problem over *all system output traces* also has the structure of a Pareto front. Other methods for Pareto front computation have been studied in [106, 47]. However, the nature of the problem is significantly different in our case. Here, due to the undecidability of the problem [12], we can only guarantee that a parameter falsifies the specification. It is not the case that we can guarantee that a parameter value satisfies the specification. Therefore, the parameter falsification domain is generated strictly by utilizing a falsifying behavior.

Ideally, by solving Problem 4.2.1, we would also like to have the property that for any $\vec{\zeta} \in \Theta - \Psi$, $\phi[\vec{\zeta}]$ holds on Σ, i.e., $\Sigma \models \phi[\vec{\zeta}]$. However, even for a given $\vec{\zeta}$, the problem of algorithmically computing whether $\Sigma \models \phi[\vec{\zeta}]$ is undecidable for the classes of systems that we consider in this work [12]. We note that for some classes of systems, such as monotone dynamical systems, this problem is tractable [95].

An overview of our proposed solution to Problem 4.2.1 appears in Fig. 4.1. Given a model and a MTL specification with one or more parameters, the sampler produces

a point x_0 from the set of initial conditions, input signal u and vector of proposed (or mined) parameters $\vec{\theta}$ for the Parametric MTL specification. The initial conditions and input signal are passed to the system simulator which returns an execution trace (output trajectory and timing function). The trace, in conjunction with the proposed parameters, is then analyzed by the MTL robustness analyzer which returns a robustness value. The robustness value computed is used by the stochastic sampler to decide on next initial conditions, inputs, and estimated parameters to utilize. The process terminates once a maximum number of tests is reached or when no improvement on the proposed parameters has been made after a predefined number of iterations. As the number of parameters increases, so does the computational complexity of the problem. For formulas with more than one parameter, we present an efficient approach in Section 4.6 to explore the parameter falsification domain.

4.3 Robustness of Metric Temporal Logic Formulas

In this section, we introduce the notion of robustness of MTL formulas by reviewing and summarizing results and definitions from [83, 2]. MTL [102] enables reasoning over quantitative temporal properties of Boolean signals. To facilitate the proofs in the rest of this chapter, we present MTL in Negation Normal Form (NNF). We denote the extended real number line by $\overline{\mathbb{R}} = \mathbb{R} \cup \{\pm\infty\}$.

Definition 4.3.1 (Syntax of MTL in NNF) *The set of all well-formed MTL formulas (wff) is defined by* $\phi ::=$

$$\top \mid \bot \mid p \mid \neg p \mid \phi_1 \wedge \phi_2 \mid \phi_1 \vee \phi_2 \mid \phi_1 \mathcal{U}_{\mathcal{I}} \phi_2 \mid \phi_1 \mathcal{R}_{\mathcal{I}} \phi_2$$

where $\top$ *and* $\bot$ *are symbols, AP is the set of atomic propositions and* $p \in AP$*. Here,* $\mathcal{I}$ *is a non-empty, non-singular interval over* $\overline{\mathbb{R}}_{\geq 0}$*.*

In Boolean logic, the $\top$ and $\bot$ symbols are interpreted as true and false, but in multi-valued logics they are interpreted as the maximum and minimum of the possible logical values (see Def. 4.3.3).

Before proceeding to the actual definition of the robust semantics, we introduce some auxiliary notation. A metric space is a pair (X, d) such that the topology of the set X is induced by a metric d. Using a metric d, we can define the distance of a point $x \in X$ from a set $S \subseteq X$. Intuitively, this distance is the shortest distance from x to all the points in S. In a similar way, the depth of a point x in a set S is defined to be the shortest distance of x from the boundary of S. Both the notions of distance and depth play a fundamental role in the definition of the robustness degree. The metrics and distances utilized in this work are covered in more detail in [68, 2].

Definition 4.3.2 (Signed Distance) *Let $x \in X$ be a point, $S \subseteq X$ be a set and d be a metric on X. Then, we define the Signed Distance from x to S to be*

$$\mathbf{Dist}_d(x, S) := \begin{cases} -\min\{d(x, y) \mid y \in S\} & \text{if } x \notin S \\ \min\{d(x, y) \mid y \in X \backslash S\} & \text{if } x \in S \end{cases}$$

MTL formulas are interpreted over timed state sequences μ. In the past [67, 68], multi-valued semantics for MTL were proposed where the valuation function on the predicates takes values over the totally ordered set $\overline{\mathbb{R}}$ according to a metric d operating on the output space Y. In detail, we let the valuation function be the depth (or the distance) of the current point of the signal $\mathbf{y}(i)$ in the set $\mathcal{O}(p)$ labeled by the atomic proposition p. Intuitively, this distance represents how robust is the point $\mathbf{y}(i)$ within set $\mathcal{O}(p)$. We recall from Def. 2.3.1, $\mathcal{O}$ maps the atomic proposition p to a set. For example, for the specification $\Box p$, where $p \equiv (engine\_rpm \leq 4500)$ we have $\mathcal{O}(p) = (-\infty, 4500]$. This robustness concept is extended from points to trajectories by applying min and max operations over time. While positive robutsness values

indicate satisfaction, negative values indicate that the trajectory falsifies the MTL specification. This is referred to as the robustness estimate and is formally presented in Definition 4.3.3. The robustness estimate presents a bound on what perturbations a signal may tolerate without changing the Boolean truth value of the specification.

For the purposes of the following discussion, we use the notation $[\![\phi]\!]$ to denote the robustness estimate with which the timed state sequence μ satisfies the specification ϕ. Formally, the valuation function for a given formula ϕ is $[\![\phi]\!] : Y^N \times \mathfrak{T} \times N \to \overline{\mathbb{R}}$. In the definition below, we also use the following notation : for $Q \subseteq R$, the *preimage* of Q under τ is defined as : $\tau^{-1}(Q) := \{i \in N \mid \tau(i) \in Q\}$. Also, given an $\alpha \in \mathbb{R}$ and $\mathcal{I} = \langle l, u \rangle$, we define the timing interval shift operation as $\alpha + \mathcal{I} = \langle \alpha + l, \alpha + u \rangle$. Here, $\langle$ and $\rangle$ are used to denote brackets or parentheses for closed and open intervals.

Definition 4.3.3 (Robustness Estimate [68]) *Let $\mu = (\mathbf{y}, \tau) \in Y^{[0,T]}$, and $i, j, k \in N$, then the robustness estimate of any formula MTL formula is defined as:*

$$
\begin{aligned}
[\![\top]\!](\mu, i) &:= +\infty \\
[\![\bot]\!](\mu, i) &:= -\infty \\
[\![p]\!](\mu, i) &:= \mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p)) \\
[\![\neg p]\!](\mu, i) &:= -\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p)) \\
[\![\phi_1 \vee \phi_2]\!](\mu, i) &:= \max([\![\phi_1]\!](\mu, i), [\![\phi_2]\!](\mu, i)) \\
[\![\phi_1 \wedge \phi_2]\!](\mu, i) &:= \min([\![\phi_1]\!](\mu, i), [\![\phi_2]\!](\mu, i)) \\
[\![\phi_1 \mathcal{U}_{\mathcal{I}} \phi_2]\!](\mu, i) &:= \\
&\max_{j \in \tau^{-1}(\tau(i)+\mathcal{I})} \big(\min([\![\phi_2]\!](\mu, j), \min_{i \le k < j} [\![\phi_1]\!](\mu, k))\big) \\
[\![\phi_1 \mathcal{R}_{\mathcal{I}} \phi_2]\!](\mu, i) &:= \\
&\min_{j \in \tau^{-1}(\tau(i)+\mathcal{I})} \big(\max([\![\phi_2]\!](\mu, j), \max_{i \le k < j} [\![\phi_1]\!](\mu, k))\big)
\end{aligned}
$$

When $i = 0$, then we write $[\![\phi]\!](\mu)$.

Example 4.3.1 *As an example, consider the trajectory in Fig. 4.3 (left) and the specification* $\Diamond_{[0,30]}p$*, where* $p \equiv (\omega \geq 3500)$ *and* $\mathcal{O}(p) = [3500, +\infty)$*. Assume that the signal is sampled at every second, i.e.* $\tau(i+1) - \tau(i) = 1$ *such that* $\tau^{-1}([0, 30]) = \{0, 1, 2 \dots 30\}$*. Then, the robustness of the formula is:*

$$[\![\Diamond_{[0,30]}p]\!] = \max_{i \in \tau^{-1}([0,30])} \mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p)) = \max_{i \in \tau^{-1}([0,30])} \mathbf{y}(i) - 3500 = -113$$

which corresponds to the maximum of the distance between the trajectory and the set $\mathcal{O}(p)$*. In this case, the maximum is found at* $i = 30$ *and it is negative because the signal never exceeds the 3500 threshold. Further examples can be found in [68].*

The robustness of an MTL formula with respect to a timed state sequence can be computed using several existing algorithms [68, 66, 59]. If we consider the robustness estimate over systems, the resulting robustness landscape can be both nonlinear and non-convex. In Fig. 4.2, we present the robustness landscape for the two running examples, namely Examples 2.6.1 (AT) and 2.6.3 (HS), on two specifications.

We note that in the S-TaLiRo testing framework described in Section 2.5, MTL requirements can have both Boolean and physical constraints for their predicates. For example, consider the following temporal logic specification for an automotive system: $\phi = \Box(gear_1 \rightarrow (speed < 40))$. Here, the specification states that always, while in gear 1, the speed of the vehicle should be less than 40. We note that without additional information, the Boolean component ($gear_1$) will produce flat robustness semantics, i.e. the search space will be flat with respect to that predicate. To resolve this issue, the testing framework requires additional information. For example, in [2], the authors utilize a graph structure which describes the relationship between the locations of the system. In our example, that would be the gears of the vehicle and

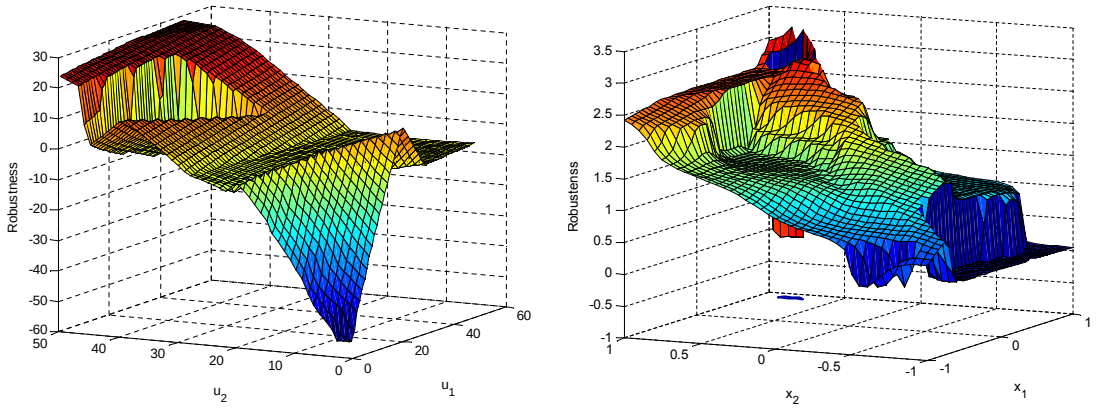


Figure 4.2: Robustness estimate landscape for two system specifications. **Left:** Example 2.6.1 (AT): $\phi_{AT} = \neg(\Diamond_{[0,30]}(v > 100) \wedge \Box(\omega \leq 4500)) \wedge \neg\Diamond_{[10,40]}\Box_{[0,5]}(60 < v \leq 80) \wedge \neg\Diamond_{[50,60]}\Box_{[0,3]}(v \leq 60)$. The input signal to the system is generated by linearly interpolating control points u_1, u_2 at time 0 and 60, respectively, for the throttle input u. That is, $u(t) = \frac{60-t}{60}u_1 + \frac{t}{60}u_2$.; **Right:** Example 2.6.3 (HS): $\phi_{HS} = \Box_{[0,2]}\neg a \wedge \Box_{[0,2]}\neg b$, where $\mathcal{O}(a) = [-1.6, -1.4]^2$ and $\mathcal{O}(b) = [3.4, 3.6] \times [-1.6, -1.4]$. Here x_1 and x_2 are initial conditions for the hybrid system.

the transition conditions between the gears. Using that information, the robustness semantics are modified so that the search problem is steered towards system locations of interest (in our case, that would gear 1). In [7], the authors take a different approach. There, the robustness semantics utilized are defined as integrals over the duration, and therefore both space and time are captured, but distance information is lost.

4.4 Monotonicity of Parametric Metric Temporal Logic Formulas

The syntax of Parametric MTL is formally presented in Section 2.4. Since the valuation function of an MTL formula is a composition of minimum and maximum operations quantified over time intervals, a formula $\phi[\theta]$, when θ is a scalar, is always

monotonic with respect to θ under certain conditions. Similarly, when $\vec{\theta}$ is a vector, then the valuation function is monotonic with respect to a priority function $f(\vec{\theta})$. In general, determining the monotonicity of PMTL formulas is undecidable [91]. The priority function will enable the system engineer to prioritize the optimization of some parameters over others by defining specific weights, or setting an optimization strategy such as optimizing the minimum, maximum, or norm of all parameters. The priority function will be defined in detail in the next section.

In the following, we present monotonicity results for single and multiple parameter PMTL formulas. We note that the monotonicity results apply to a subset of PMTL.

4.4.1 Single parameter PMTL formulas

The first example presented shows how monotonicity appears in the timing requirements of PMTL formulas.

Example 4.4.1 (AT) *Consider the PMTL formula $\phi[\theta] = \Box_{[0,\theta]} p$ where $p \equiv (\omega \leq 3250)$. Given a timed state sequence $\mu = (\mathbf{y}, \tau)$ with $\tau(0) = 0$, for $\theta_1 \leq \theta_2$, we have:*

$$[0, \theta_1] \subseteq [0, \theta_2] \implies \tau^{-1}([0, \theta_1]) \subseteq \tau^{-1}([0, \theta_2]).$$

Therefore, by Definitions (4.3.2) and (4.3.3) we have

$$\begin{aligned} [\![\phi[\theta_1]]\!](\mu) &= \min_{i \in \tau^{-1}([0,\theta_1])} (-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p))) \\ &\geq \min_{i \in \tau^{-1}([0,\theta_2])} (-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p))) = [\![\phi[\theta_2]]\!](\mu). \end{aligned}$$

That is, the function $[\![\phi[\theta]]\!](\mu)$ is non-increasing with respect to θ. Intuitively, this relationship holds since by extending the value of θ in $\phi[\theta]$, it becomes just as or more difficult to satisfy the specification. See Fig. 4.3 for an example using an output trajectory from the system in Example 2.6.1. $\triangle$

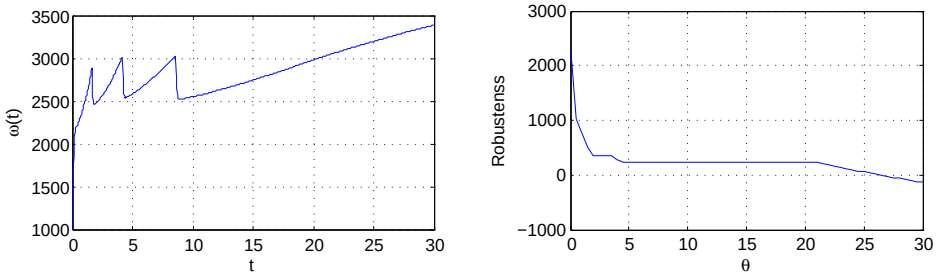


Figure 4.3: Example 4.4.1. Left: Engine speed $\omega(t)$ for constant throttle $u(t) = 50$. Right: The robustness estimate of the specification $\Box_{[0,\theta]}(\omega \leq 3250)$ with respect to θ.

The aforementioned example is formalized by the following monotonicity results.

Lemma 4.4.1 *Consider a PMTL formula $\phi[\theta]$ such that it contains one or more subformulas $\phi_1 Op_{\mathcal{I}[\theta]} \phi_2$ where $Op \in \{\mathcal{U}, \mathcal{R}\}$. Then, given a timed state sequence $\mu = (\mathbf{y}, \tau)$, for θ_1, $\theta_2 \in \overline{\mathbb{R}}_{\geq 0}$, such that $\theta_1 \leq \theta_2$, and for $i \in N$, we have:*

1. *if for all such subformulas, we have (i) $Op = \mathcal{U}$ and $\max \mathcal{I}(\theta) = \theta$ or (ii) $Op = \mathcal{R}$ and $\min \mathcal{I}(\theta) = \theta$, then $[\![\phi[\theta_1]]\!](\mu, i) \leq [\![\phi[\theta_2]]\!](\mu, i)$, i.e., the function $[\![\phi[\theta]]\!](\mu, i)$ is non-decreasing with respect to θ.*

2. *if for all such subformulas, we have (i) $Op = \mathcal{R}$ and $\max \mathcal{I}(\theta) = \theta$ or (ii) $Op = \mathcal{U}$ and $\min \mathcal{I}(\theta) = \theta$, then $[\![\phi[\theta_1]]\!](\mu, i) \geq [\![\phi[\theta_2]]\!](\mu, i)$, i.e., the function $[\![\phi[\theta]]\!](\mu, i)$ is non-increasing with respect to θ.*

A sketch of the proof is in Appendix A. Note that Lemma 4.4.1 allows for the repetition of a parameter in a PMTL formula. For example, consider the specification $\phi = \Box_{[\theta,5]} a \wedge \Diamond_{[0,\theta]} b \equiv \bot \mathcal{R}_{[\theta,5]} a \wedge \top \mathcal{U}_{[0,\theta]} b$. In this case, ϕ satisfies the conditions in Lemma 4.4.1. Thus, from Lemma 4.4.1 we know that for two values θ_1 and θ_2 where

$\theta_1 \leq \theta_2$:

$$[\![\Box_{[\theta_1,5]} a \wedge \Diamond_{[0,\theta_1]} b]\!](\mu, i) \leq [\![\Box_{[\theta_2,5]} a \wedge \Diamond_{[0,\theta_2]} b]\!](\mu, i)$$

In the following, we derive similar results for the case where the parameter appears in the numerical expression of the atomic proposition.

Lemma 4.4.2 *Consider a PMTL formula $\phi[\theta]$ with a single parameter variable θ such that it contains parametric atomic propositions $p_1[\theta]...p_n[\theta]$ in one or more sub-formulas. Then, given a timed state sequence $\mu = (\mathbf{y}, \tau)$, for all θ_1, $\theta_2 \in \overline{\mathbb{R}}_{\geq 0}$, such that $\theta_1 \leq \theta_2$, and for $i \in N$, we have:*

- *if $\forall j.p_j[\theta] \equiv g_j(x) \leq \theta$, then $[\![\phi[\theta_1]]\!](\mu, i) \leq [\![\phi[\theta_2]]\!](\mu, i)$, i.e., the function $[\![\phi[\theta]]\!](\mu, i)$ is non-decreasing with respect to θ, and*
- *if $\forall j.p_j[\theta] \equiv g_j(x) \geq \theta$, then $[\![\phi[\theta_1]]\!](\mu, i) \geq [\![\phi[\theta_2]]\!](\mu, i)$, i.e., the function $[\![\phi[\theta]]\!](\mu, i)$ is non-increasing with respect to θ.*

A sketch of the proof is in Appendix A.

4.4.2 Multiple parameter PMTL formulas

Next, we extend the result for multiple parameters.

Example 4.4.2 (AT) *Consider the PMTL formula $\phi[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]}\, q \wedge \Box p[\theta_2])$ where $\vec{\theta} = [\theta_1, \theta_2]^\intercal$, $p[\theta_2] \equiv (\omega \leq \theta_2)$ and $q \equiv (v \geq 100)$. Given a timed state sequence $\mu = (\mathbf{y}, \tau)$ with $\tau(0) = 0$, for two vectors of parameters $\vec{\theta}, \vec{\theta'} \in \mathbb{R}^2$ where $\vec{\theta} \preceq \vec{\theta'}$, for*

all i, we have:

$$\theta_2 \leq \theta_2' \implies \mathcal{O}(p[\theta_2]) \subseteq \mathcal{O}(p[\theta_2']) \implies$$

$$\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2])) \leq \mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2'])) \implies$$

$$-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2])) \geq -\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2'])) \tag{4.1}$$

$$\theta_1 \leq \theta_1' \implies [0, \theta_1] \subseteq [0, \theta_1'] \implies$$

$$\tau^{-1}([0, \theta_1]) \subseteq \tau^{-1}([0, \theta_1']) \tag{4.2}$$

Therefore, by (4.1) and (4.2) we obtain:

$$[\![\phi[\vec{\theta}]]\!](\mu) = \min_{i \in \tau^{-1}([0,\theta_1])} (-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2])))$$

$$\overset{(2)}{\geq} \min_{i \in \tau^{-1}([0,\theta_1])} (-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2'])))$$

$$\overset{(3)}{\geq} \min_{i \in \tau^{-1}([0,\theta_1'])} (-\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p[\theta_2']))) = [\![\phi[\vec{\theta'}]]\!](\mu)$$

That is, the function $[\![\phi[\vec{\theta}]]\!](\mu)$ is non-increasing for all $\vec{\theta}$ for which the relation $\preceq$ holds. ▲

Example 4.4.3 (AT) *Consider the PMTL formula $\phi[\vec{\theta}] = \Box(p[\theta_1] \wedge q[\theta_2])$ where $p[\theta_1] \equiv (v \leq \theta_1)$ and $q[\theta_2] \equiv (\omega \leq \theta_2)$. Given a timed state sequence $\mu = (\mathbf{y}, \tau)$ with $\tau(0) = 0$, for two vectors of parameters $\vec{\theta}, \vec{\theta'}$ where $\vec{\theta} \preceq \vec{\theta'}$, we have:*

$$\mathcal{O}(p[\theta_1]) \subseteq \mathcal{O}(p[\theta_1']) \implies$$

$$\mathbf{Dist}_d(\mathcal{O}(p[\theta_1])) \leq \mathbf{Dist}_d(\mathcal{O}(p[\theta_1'])) \implies$$

$$[\![p[\theta_1]]\!](\mu, i) \leq [\![p[\theta_1']]\!](\mu, i)$$

and

$$\mathcal{O}(q[\theta_2]) \subseteq \mathcal{O}(q[\theta_2']) \implies$$

$$\mathbf{Dist}_d(\mathcal{O}(p[\theta_2])) \leq \mathbf{Dist}_d(\mathcal{O}(p[\theta_2'])) \implies$$

$$[\![q[\theta_2]]\!](\mu, i) \leq [\![q[\theta_2']]\!](\mu, i)$$

Therefore, $[\![\phi[\vec{\theta}]]\!](\mu) \leq [\![\phi[\vec{\theta'}]]\!](\mu)$. *That is, the function* $[\![\phi[\vec{\theta}]]\!](\mu)$ *is non-decreasing for all* $\vec{\theta}$ *for which the relation* $\preceq$ *holds. Figure 4.4 presents the robustness landscape of two parameters over constant input.* ▲

Now we may state the main monotonicity theorem for multiple parameters. We remark that for convenience we define the parametric subformulas over all the possible parameters even though only some of them are used in each subformula.

Theorem 4.4.1 *Consider a PMTL formula* $\psi[\vec{\theta}]$, *where* $\vec{\theta}$ *is a vector of parameters, such that* $\psi[\vec{\theta}]$ *contains temporal subformulas* $\phi[\vec{\theta}] = \phi_1[\vec{\theta}] Op_{\mathcal{I}[\theta_s]} \phi_2[\vec{\theta}]$, $Op \in \{\mathcal{U}, \mathcal{R}\}$, *or propositional subformulas* $\phi[\vec{\theta}] = p[\vec{\theta}]$. *Then, given a timed state sequence* $\mu = (\mathbf{y}, \tau)$, *for* $\vec{\theta}$, $\vec{\theta'} \in \overline{\mathbb{R}}^n_{\geq 0}$, *such that* $\vec{\theta} \preceq \vec{\theta'}$, *where* $1 \leq j \leq n$, *and for* $i \in N$, *we have:*

- *if for all such subformulas (i)* $Op = \mathcal{U}$ *and* $\max \mathcal{I}(\theta_s) = \theta_s$ *or (ii)* $Op = \mathcal{R}$ *and* $\min \mathcal{I}(\theta_s) = \theta_s$ *or (iii)* $p[\vec{\theta}] \equiv g(x) \leq \vec{\theta}$, *then* $[\![\phi[\vec{\theta}]]\!](\mu, i) \leq [\![\phi[\vec{\theta'}]]\!](\mu, i)$, *i.e., function* $[\![\phi[\vec{\theta}]]\!](\mu, i)$ *is non-decreasing with respect to* $\vec{\theta}$,

- *if for all such subformulas (i)* $Op = \mathcal{R}$ *and* $\max \mathcal{I}(\theta_s) = \theta_s$ *or (ii)* $Op = \mathcal{U}$ *and* $\min \mathcal{I}(\theta_s) = \theta_s$ *or (iii)* $p[\vec{\theta}] \equiv g(x) \geq \vec{\theta}$, *then* $[\![\phi[\vec{\theta}]]\!](\mu, i) \geq [\![\phi[\vec{\theta'}]]\!](\mu, i)$, *i.e., function* $[\![\phi[\vec{\theta}]]\!](\mu, i)$ *is non-increasing with respect to* $\vec{\theta}$.

A sketch of the proof is in Appendix A.

In this section, we have presented several cases where we can syntactically determine the monotonicity of the PMTL formula with respect to its parameters. However, we remark that in general, determining the monotonicity of PMTL formulas is undecidable [91].

4.5 Temporal Logic Parameter Bound Computation

The notion of robustness of temporal logics will enable us to pose the parameter mining problem as an optimization problem. In order to solve the resulting optimiza-

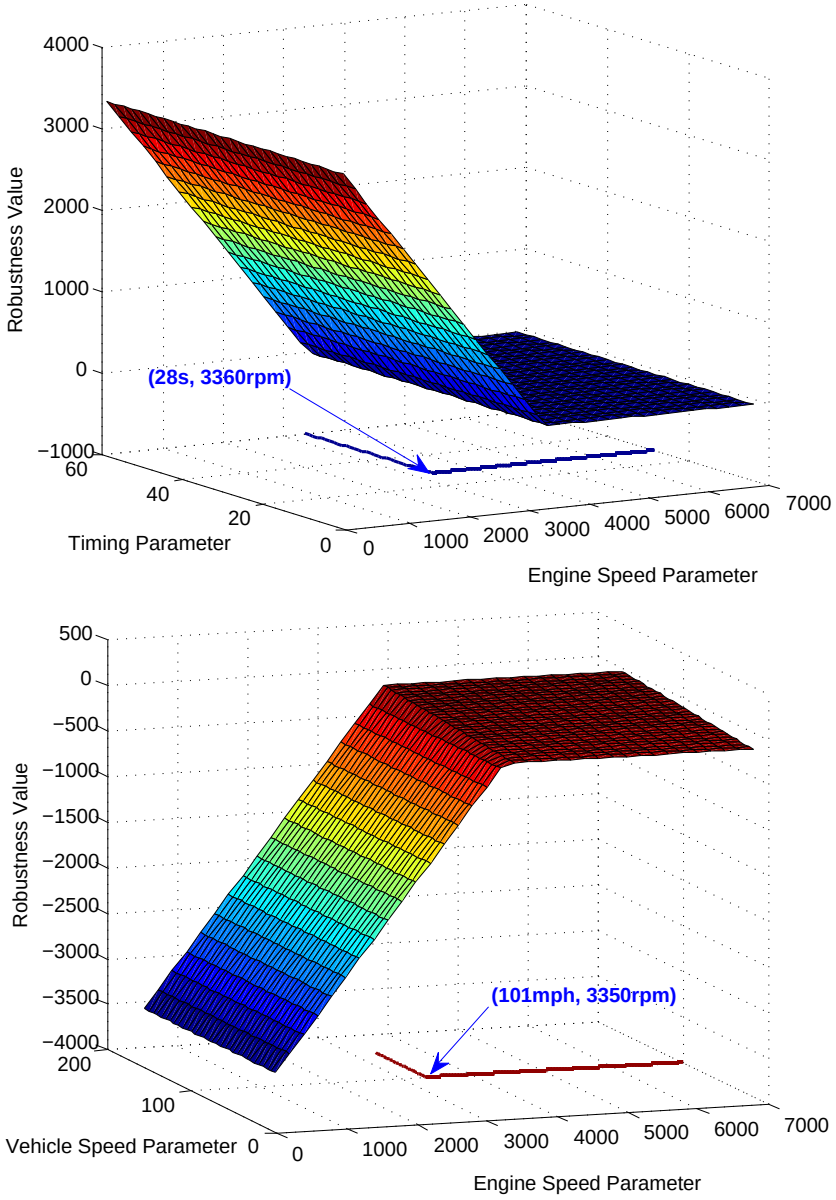


Figure 4.4: **Top:** Example 4.4.2: Robustness estimate landscape for varying parameters for engine and vehicle speed for constant throttle $u(t) = 50$. **Bottom**: Example 4.4.3: Robustness landscape for varying parameters for timing parameter and engine speed for constant throttle $u(t) = 50$. In both figures, the contour line shows the intersection of the robustness landscape with the zero level set.

tion problem, falsification methods and S-TaLiRo [126] can be utilized to estimate the solution for Problem 4.2.1.

As described in the previous section, the parametric robustness functions that we are considering are monotonic with respect to the search parameters. Therefore, if we are searching for a parameter vector over an interval $\Theta = [\underline{\vec{\theta}}, \overline{\vec{\theta}}]$, where Θ is a hypercube and $\underline{\vec{\theta}} = [\underline{\theta}_1, \underline{\theta}_2, \ldots, \underline{\theta}_n]^\intercal$ and $\overline{\vec{\theta}} = [\overline{\theta}_1, \overline{\theta}_2, \ldots, \overline{\theta}_n]^\intercal$, we are either trying to minimize or maximize a function f of $\vec{\theta}$ such that for all $\vec{\theta} \in \Theta^*$, we have $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$.

Example 4.5.1 (AT) *Let us consider again the automotive transmission example and the specification $\phi[\theta] = \Box_{[0,\theta]} p$ where $p \equiv (\omega \leq 4500)$. The specification robustness $[\![\phi[\theta]]\!](\Delta_\Sigma(u))$ as a function of θ and the input u appears in Fig. 4.5 for constant input signals. The creation of the graph required $100 \times 30 = 3,000$ simulations. The contour under the surface indicates the zero level set of the robustness surface, i.e., the θ and u values for which we get $[\![\phi[\theta]]\!](\Delta_\Sigma(u)) = 0$. From the graph, we can infer that $\theta^* \approx 2.8$ and that for any $\theta \in [2.8, 30]$, we have $[\![\phi[\theta]]\!](\Sigma) \leq 0$. The approximate value of θ^* is an estimate based on the granularity of the grid that we used to plot the surface.* ▲

In summary, in order to solve Problem 4.2.1, we would have to solve the following optimization problem:

$$\begin{aligned} \text{optimize} \quad & f(\vec{\theta}) \\ \text{subject to} \quad & \vec{\theta} \in \Theta \text{ and} \\ & [\![\phi[\vec{\theta}]]\!](\Sigma) = \min_{\mu \in \mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \leq 0 \end{aligned} \tag{4.3}$$

Where $f : \mathbb{R}^n \to \mathbb{R}$ is a either a non-increasing ($\geq$) or a non-decreasing ($\leq$) function. For two vector parameter values $\vec{\theta}$, $\vec{\theta}'$, if $\vec{\theta} \preceq \vec{\theta}'$ and $\vec{\theta} \geq 0$ then $f(\vec{\theta}) \bowtie f(\vec{\theta}')$, where $\bowtie \in \{\geq, \leq\}$ depending on the monotonicity.

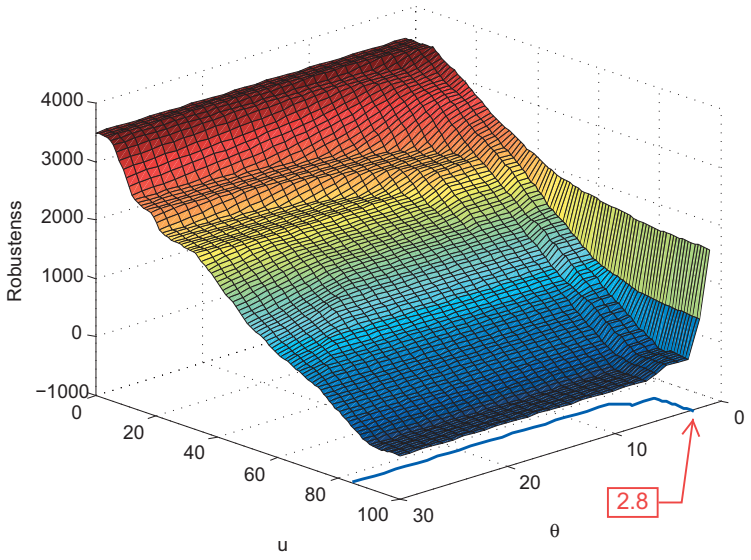


Figure 4.5: Example 4.5.1: Specification robustness estimate as a function of parameter θ and input u for specification $\phi[\theta] = \Box_{[0,\theta]}(\omega \leq 4500)$.

The function $[\![\phi[\vec{\theta}]]\!](\Sigma)$ can not be computed using reachability analysis algorithms nor is known in closed form for the systems we are considering. Therefore, we will have to compute an under-approximation of Θ^*. Our focus will be to formulate an optimization problem that can be solved using stochastic search methods. In particular, we will reformulate the optimization problem (4.3) into a new one where the constraints due to the specification are incorporated into the cost function:

$$\text{optimize}_{\vec{\theta}\in\Theta} \left(f(\vec{\theta}) + \begin{cases} \gamma \pm [\![\phi[\vec{\theta}]]\!](\Sigma) \\ \qquad \text{if } [\![\phi[\vec{\theta}]]\!](\Sigma) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) \tag{4.4}$$

where the sign ($\pm$) and the parameter γ depend on whether the problem is a maximization or a minimization problem. The parameter γ must be properly chosen so that the solution of problem (4.4) is in Θ if and only if $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$. Therefore, if

the problem in Eq. (4.3) is feasible, then the optimal points of Eq. (4.3) and Eq. (4.4) are the same.

4.5.1 Non-increasing Robustness Functions

In the case of non-increasing robustness functions $[\![\phi[\vec{\theta}]]\!](\Sigma)$ with respect to the search vector variable $\vec{\theta}$, the optimization problem is a minimization problem. Without loss of generality, let us consider the case for single parameter specifications. Assume that $[\![\phi[\overline{\theta}]]\!](\Sigma) \leq 0$. Since $\theta \leq \overline{\theta}$, we have $[\![\phi[\theta]]\!](\Sigma) \geq [\![\phi[\overline{\theta}]]\!](\Sigma)$, we need to find the minimum θ such that we still have $[\![\phi[\theta]]\!](\Sigma) \leq 0$. That θ value will be the desired θ^* since for all $\theta' \in [\theta^*, \overline{\theta}]$, we will have $[\![\phi[\theta']]\!](\Sigma) \leq 0$.

We will reformulate the problem of Eq. (4.4) so that we do not have to solve two separate optimization problems. From (4.4), we have:

$$\begin{aligned}
&\min_{\vec{\theta}\in\Theta} \left(f(\vec{\theta}) + \begin{cases} \gamma + \min_{\mu\in\mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } \min_{\mu\in\mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) = \\
&= \min_{\vec{\theta}\in\Theta} \left(f(\vec{\theta}) + \min_{\mu\in\mathcal{L}_\tau(\Sigma)} \begin{cases} \gamma + [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } [\![\phi[\vec{\theta}]]\!](\mu) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) = \\
&= \min_{\vec{\theta}\in\Theta} \min_{\mu\in\mathcal{L}_\tau(\Sigma)} \left(f(\vec{\theta}) + \begin{cases} \gamma + [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } [\![\phi[\vec{\theta}]]\!](\mu) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right)
\end{aligned} \tag{4.5}$$

The previous discussion is formalized as follows.

Proposition 4.5.1 *Let $\vec{\theta}^*$ be a set of parameters and μ^* be the system trajectory returned by an optimization algorithm that is applied to the problem in Eq. (4.5). If $[\![\phi[\vec{\theta}^*]]\!](\mu^*) \leq 0$, then for all $\vec{\theta} \succeq \vec{\theta}^*$, $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$.*

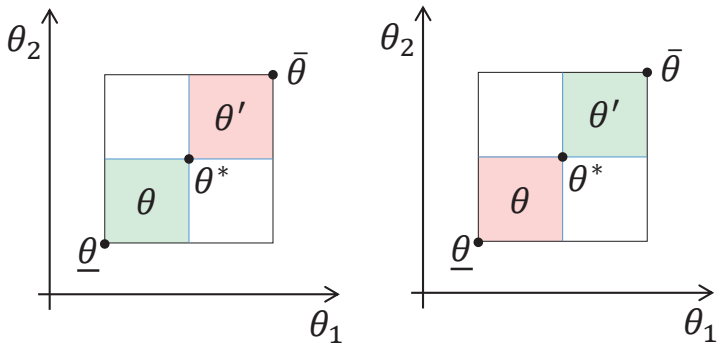


Figure 4.6: Illustration of the arrangement of parameters for non-increasing (**Left**) and non-decreasing (**Right**) robustness functions for a two parameter specification. The green (red) region represents parameter valuations for which we have a positive (negative) robustness value over all system behaviors.

Proposition 4.5.2 *If $f(\vec{\theta}) = \|\vec{\theta}\|$, and the robustness function is non-increasing, then $\gamma = \|\vec{\bar{\theta}}\|$ is a valid choice for parameter γ. Here, $\|\cdot\|$ denotes the euclidean norm.*

The proofs for Propositions 4.5.1 and 4.5.2 are in Appendix A.

Example 4.5.2 (AT) *Using Eq. (4.5) as a cost function, we can now compute a parameter for Example 4.5.1 using* S-TaLiRo *[19, 126]. In particular, using Simulated Annealing as a stochastic optimization function,* S-TaLiRo *returns $\theta^* \approx 2.45$ as optimal parameter for constant input $u(t) = 99.81$. The corresponding temporal logic robustness for the specification $\Box_{[0,2.45]}(\omega \leq 4500)$ is -0.0445. The number of tests performed for this example was 500 and, potentially, the accuracy of estimating θ^* can be improved if we increase the maximum number of tests. However, based on 100 tests the algorithm converges to a good solution within 200 tests.* ▲

Example 4.5.3 (AT) *Let us consider again the automotive transmission example and the specification $\phi[\theta] = \Box_{[0,\theta]}p$ where $p \equiv (\omega \leq 4500)$. Using Eq. (4.5) as a cost*

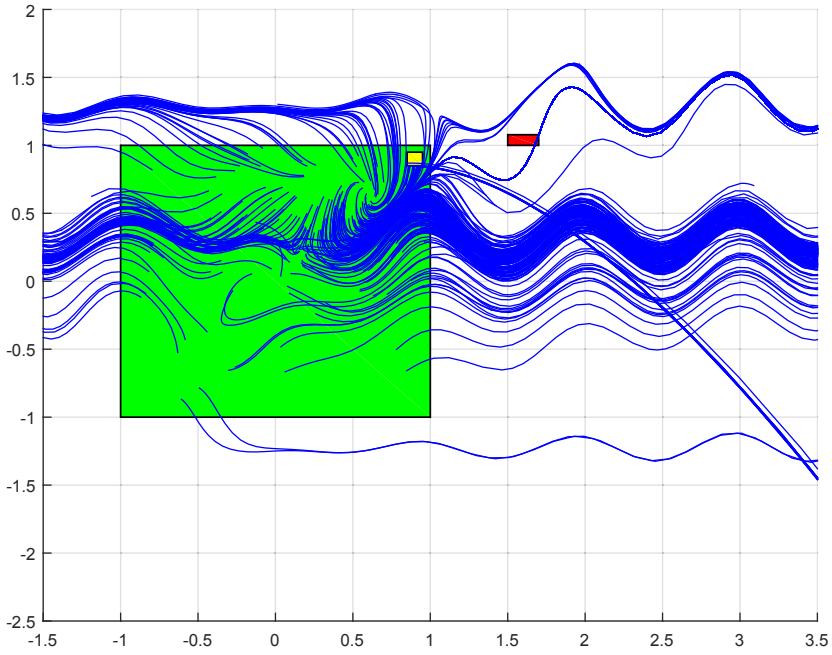


Figure 4.7: Example 2.6.3: Specification falsification for $\phi[\vec{\theta}] = \Box_{[0,\theta_1]}\neg a[\vec{\theta}]$ where $\mathcal{O}(a[\vec{\theta}]) = [1.5, \theta_2]\times[1, \theta_3]$ with mined parameters $\theta_1 = 3.417$, $\theta_2 = 1.7$, and $\theta_3 = 1.078$.

function, we can now compute a parameter using S-TaLiRo *[19, 126]. In particular, using Simulated Annealing as a stochastic optimization function,* S-TaLiRo *returns* $\theta^* \approx 2.45$ *as optimal parameter for constant input* $u(t) = 99.81$. *The corresponding temporal logic robustness for the specification* $\Box_{[0,2.45]}(\omega \leq 4500)$ *is* -0.0445. *The number of tests performed for this example was* 500 *and, potentially, the accuracy of estimating* θ^* *can be improved if we increase the maximum number of tests. However, based on 100 runs the algorithm converges to a good solution within* 200 *tests.* ▲

Example 4.5.4 (HS) *Let us consider the specification* $\phi[\vec{\theta}] = \Box_{[0,\theta_1]}\neg a$ *where* $\mathcal{O}(a) = [1.5, \theta_2] \times [1, \theta_3]$ *on our hybrid system running example from Section 2.6.3. Here, the bounds for the timing parameter are* $\theta_1 \in [0, 5]$ *and the bounds for the state parameters are* $\theta_2 \in [1.5, 2.1]$ *and* $\theta_3 \in [1.1, 1.6]$. *The ranges for the parameters are chosen based on prior knowledge and experience about the system. The parameter mining algorithm*

from S-TaLiRo *returns* $\theta_1^* = 3.417$, $\theta_2^* = 1.7$, *and* $\theta_3^* = 1.078$ *after running 1000 tests on the system. The generated trajectories by the parameter mining algorithm are presented in Fig. 4.7. The returned parameters guarantee that the system does not satisfy the specification for all parameters* $\vec{\theta}$ *where* $\vec{\theta}^* \preceq \vec{\theta}$. ▲

4.5.2 Non-decreasing Robustness Functions

The case of non-decreasing robustness functions is symmetric to the case of non-increasing robustness functions. In particular, the optimization problem is a maximization problem. We will reformulate the problem of Eq. (4.4) so that we do not have to solve two separate optimization problems. From (4.4), we have:

$$\begin{aligned}
&\max_{\vec{\theta}\in\Theta} \left(f(\vec{\theta}) + \begin{cases} \gamma - \max_{\mu\in\mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } \max_{\mu\in\mathcal{L}_\tau(\Sigma)} [\![\phi[\vec{\theta}]]\!](\mu) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) = \\
&= \max_{\vec{\theta}\in\Theta} \left(f(\vec{\theta}) + \max_{\mu\in\mathcal{L}_\tau(\Sigma)} \begin{cases} \gamma - [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } -[\![\phi[\vec{\theta}]]\!](\mu) \leq 0 \\ 0 \quad \text{otherwise} \end{cases} \right) = \\
&= \max_{\vec{\theta}\in\Theta} \max_{\mu\in\mathcal{L}_\tau(\Sigma)} \left(f(\vec{\theta}) + \begin{cases} \gamma - [\![\phi[\vec{\theta}]]\!](\mu) \\ \quad \text{if } [\![\phi[\vec{\theta}]]\!](\mu) \geq 0 \\ 0 \quad \text{otherwise} \end{cases} \right)
\end{aligned} \tag{4.6}$$

The previous discussion is formalized in the following result.

Proposition 4.5.3 *Let* $\vec{\theta}^*$ *be a set of parameters and* μ^* *be the system trajectory returned by an optimization algorithm that is applied to the problem in Eq. (4.6). If* $[\![\phi[\vec{\theta}^*]](\mu^*) \leq 0$, *then for all* $\vec{\theta} \preceq \vec{\theta}^*$, *we have* $[\![\phi[\theta]]\!](\Sigma) \leq 0$.

Proposition 4.5.4 *If* $f(\vec{\theta}) = \|\vec{\theta}\|$ *and the robustness function is non-decreasing, then* $\gamma = -\|\vec{\theta}\|$ *is a valid choice for parameter* γ.

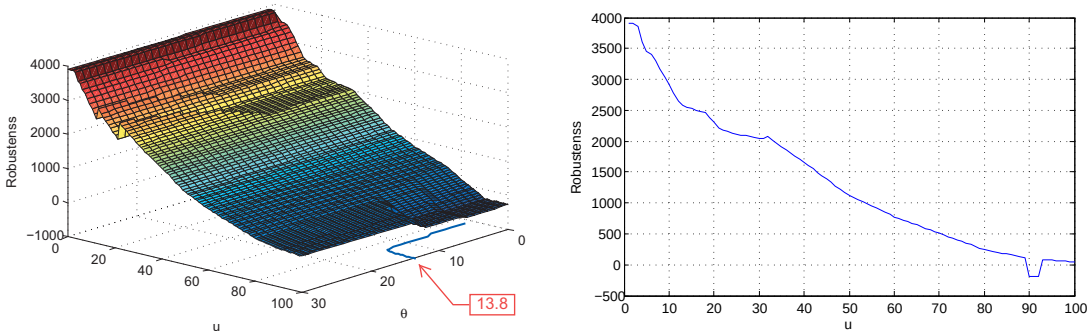


Figure 4.8: Example 4.5.5. **Left**: Specification robustness as a function of the parameter θ and the input u. **Right**: The robustness function $[\![\Box_{[12.59,30]}(\omega \leq 4500)]\!](\Delta_\Sigma(u))$.

The proofs for Propositions 4.5.3 and 4.5.4 are in Appendix A.

Example 4.5.5 (AT) *Let us consider the specification* $\phi[\theta] = \Box_{[\theta,30]}\ (\omega \leq 4500)$ *on our running example. The specification robustness* $[\![\phi[\theta]]\!](\Delta_\Sigma(u))$ *as a function of* θ *and the input* u *appears in Fig. 4.8 for constant input signals. The creation of the graph required* $100 \times 30 = 3,000$ *tests. The contour under the surface indicates the zero level set of the robustness surface, i.e., the* θ *and* u *values for which we get* $[\![\phi[\theta]]\!](\Delta_\Sigma(u)) = 0$. *We remark that the contour is actually an approximation of the zero level set computed by a linear interpolation using the neighboring points on the grid. From the graph, we could infer that* $\theta^* \approx 13.8$ *and that for any* $\theta \in [0, 13.8]$, *we would have* $[\![\phi[\theta]]\!](\Sigma) \leq 0$. *Again, the approximate value of* θ^* *is a rough estimate based on the granularity of the grid.*

Using Eq. (4.6) as a cost function, we can now compute a parameter for Example 4.5.5 using our toolbox S-TaLiRo *[19, 126].* S-TaLiRo *returns* $\theta^* \approx 12.59$ *as optimal parameter for constant input* $u(t) = 90.88$ *within 250 tests. The temporal logic robustness for the specification* $\Box_{[12.59,30]}(\omega \leq 4500)$ *with respect to the input* u *appears in Fig. 4.8 (Right).* ▲

4.6 Parameter Falsification Domain

We utilize the solution of Problem 4.2.1 and exploit the robustness landscape of a specific class of temporal logic formulas to present two algorithms to estimate $\Psi = \{\vec{\theta}^* \in \Theta \mid \Sigma \not\models \phi[\vec{\theta}^*]\}$ for Problem 4.2.1. In fact, we can reduce this problem to finding the set $\Theta^{bd} = \Psi \cap \{\vec{\theta}^* \in \Theta \mid [\![\phi[\vec{\theta}^*]]\!](\Sigma) = 0\}$ since the robustness landscape is monotonic. Here, Θ^{bd} represents the intersection of the robustness function with the zero level set. As a preprocessing step, the PMTL parameters are normalized in the range $[0, 1]$ to avoid bias during the optimization process. It is important to note, that due to the undecidable nature of the problem, we cannot determine satisfying parameter values. Therefore, we generate the parameter falsification domain by finding only falsifying parameter values.

4.6.1 RGDA Algorithm

The first method approximates Θ^{bd} by modifying the priority function f and thereby slightly shifting the minimum or maximum of the objective function in Eq. (4.5) or Eq. (4.6), respectively. The magnitude of the shift depends on the shape of the robustness landscape of the model and specification.

As shown in Algorithm 2, the set Ψ is explored iteratively. For use in the algorithm, we define a PMTL specification monotonicity function $\mathcal{M}$: PMTL $\rightarrow \{-1, 0, 1\}$ where

$$\mathcal{M}(\phi[\vec{\theta}]) = \begin{cases} 1 & \text{if } \phi[\vec{\theta}] \text{ is non-decreasing;} \\ -1 & \text{if } \phi[\vec{\theta}] \text{ is non-increasing;} \\ 0 & \text{otherwise.} \end{cases}$$

A monotonicity computation algorithm is presented in [21] and generalized in [91].

For every iteration of the algorithm, we draw a random vector ω with dimension

Algorithm 2 Robustness Guided Parameter Falsification Domain Algorithm RGDA(`opt`, Γ, Θ, ϕ, Σ, n, t)

Input: Stochastic optimization algorithm `opt`, search space Γ, parameter range Θ, specification ϕ, system Σ, number of iterations n and tests t
Output: Parameter falsification domain Ψ
Internal Variables: Parameter weights $\vec{\omega}$, parameters mined $\vec{\theta}^*$ and robustness value γ

1: $\langle \Psi, \vec{\omega}, \vec{\theta}^*, \gamma \rangle \leftarrow \langle \emptyset, \emptyset, \emptyset, \emptyset \rangle$
2: **for** $i = 0$ to n **do**
3: $\quad \vec{\omega} \leftarrow$ RandomVector($[0, 1]$, dimension(Θ))
4: $\quad [\vec{\theta}^*, \gamma] \leftarrow \texttt{opt}(\Gamma, \Theta, \phi, \Sigma, t, \vec{\omega}, \mathcal{M}(\phi[\vec{\theta}^*]))$ ▷ run parameter mining and robustness computation
5: $\quad$ **if** $(\gamma \leq 0)$ **then**
6: $\quad\quad$ **if** $(\mathcal{M}(\phi[\vec{\theta}^*]) = 1)$ **then**
7: $\quad\quad\quad \Psi \leftarrow \Psi \cup \{\vec{\theta} \in \Theta \mid \forall i \ (0 \leq \theta_i \leq \theta_i^*)\}$ ▷ expand the falsification domain Ψ
8: $\quad\quad$ **else if** $(\mathcal{M}(\phi[\vec{\theta}^*]) = -1)$ **then**
9: $\quad\quad\quad \Psi \leftarrow \Psi \cup \{\vec{\theta} \in \Theta \mid \forall i \ (\theta_i \geq \theta_i^* \geq 0)\}$
10: $\quad\quad$ **end if**
11: $\quad$ **end if**
12: **end for**
13: **return** Ψ

equal to the dimension of Θ. The random vector is used as parameter weights for the priority function $f(\vec{\theta})$. Namely, $f(\vec{\theta}) = \sum w_i \theta_i$. We run parameter mining, which returns an approximation for Eq. (4.4). In case $\phi[\vec{\theta}]$ is non-decreasing (or non-increasing), the optimization algorithm `opt` is a maximization (or minimization) algorithm. We utilize the values mined and the corresponding robustness value to expand Ψ and reduce the unknown parameter range for the next iteration. We present the iterative process in Fig. 4.9. In the example presented, for each parameter weight, 100 tests are conducted. The running time for 100 iterations was 53.63 minutes on a computer with Windows Server 2012 OS, Intel Xeon E5-2670v2 2.5GHz CPU, and 64GB of RAM.

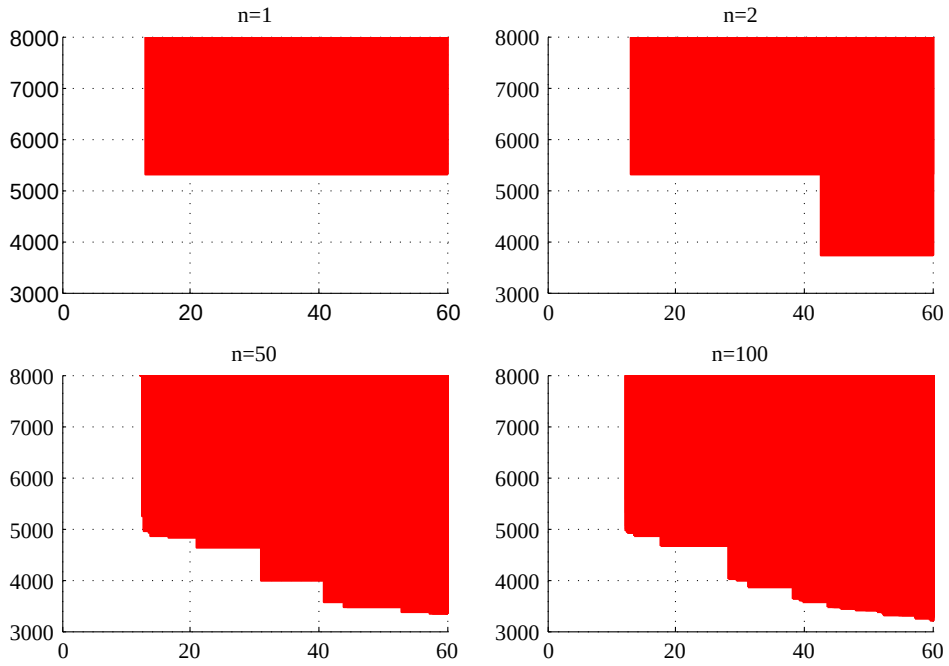


Figure 4.9: Illustration of the iterative process for Algorithm 2. Specification: $\phi[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]} q \wedge \Box p[\theta_2])$ where $p[\theta_2] \equiv (\omega \leq \theta_2)$ and $q \equiv (v \geq 100)$. Model: Automatic Transmission as described in Example 2.6.1. The red colored set represents set $\Psi = \{\vec{\theta} \in \Theta \mid \Sigma \not\models \phi[\vec{\theta}]\}$, i.e., the set of parameter values such that the system does not satisfy the specification. In each iteration of the algorithm, set Ψ gets expanded by the optimal falsifying parameter which is guided by the robustness landscape and the random weight in the priority function.

4.6.2 SDA Algorithm

The second algorithm presented explores the set Θ^{bd} by iteratively expanding the set of falsifying parameters, namely, the set Ψ. However, in this case, the search is finely structured and does not depend on randomized weights. For presentation purposes, let us consider the case for specifications with non-decreasing monotonicity. Given a normalized parameter range with dimension η, in each iteration of the

algorithm, we solve the optimization problem:

$$\begin{aligned} \text{maximize} \quad & c \\ \text{subject to} \quad & c * \vec{b} + \vec{p} \in \Theta \text{ and} \\ & \Sigma \not\models \phi[\, c * \vec{b} + \vec{p}\,] \end{aligned} \tag{4.7}$$

where $\vec{p}$ is the starting point of the optimization problem in each iteration and $\vec{b}$ is the bias vector which enables to prioritize specific parameters in the search. Namely, the choice of $\vec{b}$ directs the expansion of the parameter falsification domain along a specific direction. We refer to the solution of Eq. (4.7) in the i^{th} iteration of the algorithm as **marker**(i). Initially, for the first iteration, the value of $\vec{p}$ is set to $\vec{0}$ or $\vec{1}$ depending on the monotonicity of the specification. The returned **marker**(1) from Eq. (4.7) is then utilized to update Ψ, the set of parameters for which the system does not satisfy the specification. Next, we generate at most $2^{\eta} - 2$ initial position vectors induced by the returned **marker**(1).

Consider the example presented in Fig. 4.10 where we have **marker**$(1) = [136; 7268]$. That value is utilized to update Ψ and generate two new initial position vectors at $[0; 7268]$ and $[136; 0]$. In the next iteration of the algorithm, the search is initialized in one of the newly generated initial position vectors. Namely, the search starts in $[0; 7268]$ or $[136; 0]$ (see Fig. 4.10, Left). The initial position vector not utilized is stored in a list to be used in future iterations. In the second iteration, $[136; 0]$ is used as the initial position vector. We return the solution to Eq. (4.7) with **marker**(2) $= [143; 4425]$ which generates the initial position vectors $[143; 0]$ and $[136; 4425]$ (Fig. 4.10, Middle). Similarly, **marker**(3) is generated in Fig. 4.10 (Right). In this example, the directional vector $\vec{b}$, in each iteration, directs toward the bounds of the parameter range, namely $(160, 8000)$. The algorithm terminates when one of the following conditions is met: 1) The distance between **markers** is less than some value

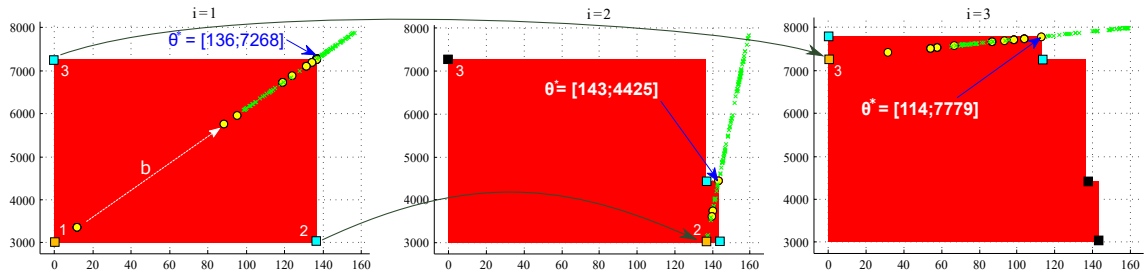


Figure 4.10: Illustration of the iterative process for Algorithm 3. Specification: $\phi[\vec{\theta}] = \Box(p[\theta_1] \wedge q[\theta_2])$ where $p[\theta_1] \equiv (v \leq \theta_1)$ and $q \equiv (\omega \leq \theta_2)$. Model: Automatic Transmission as described in Example 2.6.1. The parameter range for the specification is $\Theta = [0\ 160; 3000\ 8000]$. In each plot, the search is conducted in a specific direction $\vec{b}$. The plots from left to right represent three iterations of Algorithm 3. The yellow circles and green marks represent sample points of the search optimizer in the process of solving Eq. (4.7). Specifically, the yellow circles represent parameter values for which we have found system inputs and initial conditions that falsify the specification. The green marks represent parameter values for which falsification is not found. The largest yellow circle found by the stochastic optimizer is returned as the current marker. The orange squares represent the initial position of the search in the current iteration. The blue squares represent the initial positions generated by the current marker that will be considered in future iterations. The black squares represent initial positions that will be considered in future iterations. The red colored set represents set $\Psi = \{\vec{\theta} \in \Theta \mid \Sigma \not\models \phi[\vec{\theta}]\}$, i.e., the set of parameter values such that the system does not satisfy the specification.

ϵ, or 2) no new markers are generated from the current set of initial position vectors, or 3) a maximum number of iterations is exceeded. Experimentally, running the algorithm for 100 iterations took 51.32 minutes on a computer with Windows Server 2012 OS, Intel Xeon E5-2670v2 2.5GHz CPU, and 64GB of RAM.

4.7 Experiments and a Case Study

The algorithms and examples presented in this work are implemented and publicly available through the Matlab toolbox S-TaLiRo [19, 126].

The parametric MTL exploration of CPS is motivated by a challenge problem published by Ford in 2002 [40]. In particular, the report provided a simple–but still realistic–model of a powertrain system (both the physical system and the embedded control logic) and posed the question whether there are constant operating conditions that can cause a transition from gear two to gear one and then back to gear two. That behavior would imply that the gear transition from 1 to 2 was not necessary in the first place.

The system is modeled in Checkmate [134]. It has 6 continuous state variables and 2 Stateflow charts with 4 and 6 states, respectively. The Stateflow chart for the shift scheduler appears in Fig. 4.11. The system dynamics and switching conditions are linear. However, some switching conditions depend on the initial conditions of the system. The latter makes the application of standard system verification tools not a straightforward task.

In [66], we demonstrated that S-TaLiRo [19, 126] can successfully solve the challenge problem (see Fig. 4.11) by formalizing the requirement as an MTL specification $\phi_{e1}^{P} = \neg\Diamond(g_2 \wedge \Diamond(g_1 \wedge \Diamond g_2))$, where g_i is a proposition that is true when the system is in gear i. Stochastic search methods can be applied to solve the resulting optimization problem where the cost function is the robustness of the specification.

Algorithm 3 Structured Parameter Falsification Domain Algorithm SDA($\texttt{opt}$, Γ, Θ, ϕ, Σ, t, ϵ, $\vec{b}$, n)

Input: Stochastic optimization algorithm $\texttt{opt}$, search space Γ, parameter range Θ, specification ϕ, system Σ, number of tests t, minimum distance between markers ϵ, bias vector $\vec{b}$, maximum number of iterations n
Output: Parameter falsification domain Ψ
Internal Variables: List of initial positions $\mathcal{ML}$, termination condition $\mathcal{TC}$, initial positions generated in the current iteration $\mathcal{TL}$, iteration i

$\langle \Psi, \vec{p}, \mathcal{TC}, \mathcal{ML}, \mathcal{TL}, i \rangle \leftarrow \langle \emptyset, \emptyset, \bot, \{\}, \{\}, 0 \rangle$
if $(\mathcal{M}(\phi[\vec{\theta}]) = 1)$ **then**
    $\mathcal{ML}$.Add($\vec{0}$(dimension(Θ)))
else if $(\mathcal{M}(\phi[\vec{\theta}]) = -1)$ **then**
    $\mathcal{ML}$.Add($\vec{1}$(dimension(Θ)))
end if
while $\mathcal{TC} = \bot$ **do**
    $\mathcal{TL} \leftarrow \{\}$
    for $\vec{v}$ in $\mathcal{ML}$ **do**
        $i \leftarrow i + 1$
        $[\vec{\theta^*}, \gamma] \leftarrow \texttt{opt}(\Gamma, \Theta, \phi, \Sigma, t, \omega, \mathcal{M}(\phi[\vec{\theta}]), \vec{b}, \vec{v})$ ▷ run parameter mining starting at $\vec{v}$ and search along the directional vector $\vec{b}$
        if $(\gamma \leq 0)$ **then**
            $\mathcal{TL}$.Add(GenerateMarkers(θ^*, $\mathcal{M}(\phi[\vec{\theta}])$))
            if $(\mathcal{M}(\phi[\vec{\theta^*}]) = 1)$ **then**
                $\Psi \leftarrow \Psi \cup \{\vec{\theta} \in \Theta \mid \forall i \ (0 \leq \theta_i \leq \theta_i^*\}$
                $\Theta \leftarrow \Theta \setminus \Psi$
            else if $(\mathcal{M}(\phi[\vec{\theta^*}]) = -1)$ **then**
                $\Psi \leftarrow \Psi \cup \{\vec{\theta} \in \Theta \mid \forall i \ (\theta_i \geq \theta_i^* \geq 0)\}$
                $\Theta \leftarrow \Theta \setminus \Psi$
            end if
        end if
    end for
    $\mathcal{ML} \leftarrow \mathcal{TL}$
    if $\mathcal{ML}$.IsEmpty() or DistanceBetweenMarkers($\mathcal{ML}$) $< \epsilon$ or $i > n$ **then** $\mathcal{TC} \leftarrow \top$
    end if
end while
return Ψ

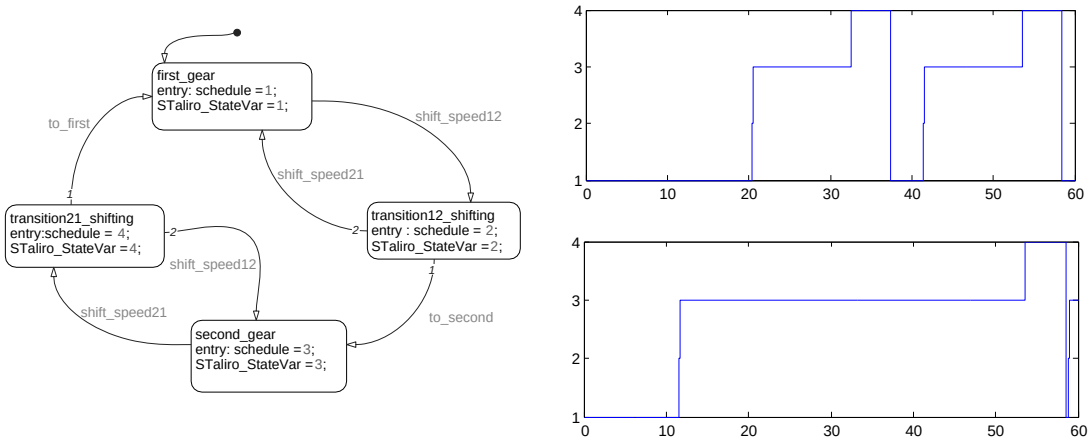


Figure 4.11: Left: The shift scheduler of the powertrain challenge problem. Right: Shift schedules. The numbers correspond to the variables in the states of the shift scheduler. Right Top: The shift schedule falsifying requirement ϕ^P_{e1}. Right Bottom: The shift schedule falsifying requirement $\phi^P_{e3}[0.4273]$.

Moreover, inspired by the success of S-TaLiRo on the challenge problem, we tried to ask a more complex question. Specifically, does a transition exist from gear two to gear one and back to gear two in less than 2.5 sec? An MTL specification that can capture this requirement is $\phi^P_{e2} = \Box((\neg g_1 \wedge X g_1) \rightarrow \Box_{(0,2.5]} \neg g_2)$. The natural question that arises is what would be the smallest time for which such a transition can occur? We can formulate a parametric MTL formula to query the model of the powertrain system: $\phi^P_{e3}[\theta] = \Box((\neg g_1 \wedge X g_1) \rightarrow \Box_{(0,\theta]} \neg g_2)$. We have extended S-TaLiRo to be able to handle parametric MTL specifications. The total simulation time of the model is set to 60 sec and the search interval is $\Theta = [0, 60]$. S-TaLiRo returned $\theta^* \approx 0.4273$ as the minimum parameter found (See Fig. 4.11) using about 300 tests of the system.

The challenge problem is extended to an industrial size high-fidelity engine model (HAE) from Section 2.6.2. In this work, we will use the Port Fuel Injected (PFI) spark ignition, 4 cylinder inline engine configuration. The model includes a tire-

model, brake system model, and a drive train model (including final drive, torque converter and transmission). The inputs to the system are the throttle and brake schedules, and the road grade, which represents the incline of the road. The outputs are the vehicle and engine speed, the current gear and a timer that indicates the time spent on a gear. We search for a particular input for the throttle schedule, brake schedule, and grade level. The inputs are parameterized using 12 search variables, where 7 are used to model the throttle schedule, 3 for the brake schedule, and 2 for the grade level. The search variables for each input are interpolated with the Piecewise Cubic Hermite Interpolating Polynomial (PCHIP) function provided as a Matlab function by Mathworks. The simulation time is 60s. We demonstrate the parameter mining method for two specifications:

$$\phi_1^S[\theta] = \Box_{[0,60]}((g2 \ \wedge \ Xg1) \rightarrow \Box_{[0,\theta]}((\tau \leq \theta) \rightarrow g1))$$

where τ is the time spent in a gear. The specification states that after shifting into gear one from gear two, there should be no shift from gear one to any other gear within θ seconds. Clearly, the property defined is equivalent to the property defined in the challenge problem in the sense that the set of trajectories that satisfy/falsify the property is the same. The reason for the change made is the improved performance of the hybrid distance metric [1] with the modified specification. The mined parameter for the specification returned is $1.29s$. Figure 4.12 presents a shift schedule for which a transition out of gear one occurs in 1.28 seconds.

$$\phi_2^S[\vec{\theta}] = \Box((v < \theta_1) \wedge (\omega < \theta_2))$$

where θ_1, θ_2 represent the vehicle and engine speed parameters, respectively. The specification states that the vehicle and engine speed is always less than θ_1 and θ_2, respectively. The mined parameters for the specification returned are 137.1 mph and 4870 rpm.

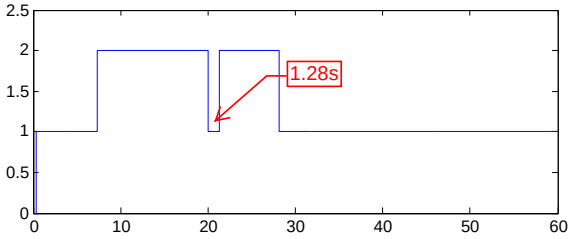


Figure 4.12: A shift schedule which falsifies the specification $\phi_1^S[\theta = 1.29] = \Box_{[0,60]}((g_2 \wedge Xg_1) \rightarrow \Box_{[0,1.29]}((\tau \leq 1.29) \rightarrow g_1))$ on the Simuquest high-fidelity engine model for specification.

In Table 4.1, we present experimental results for specifications on the Powertrain, Automotive Transmission, and Simuquest Enginuity high-fidelity engine models. A detailed description of the benchmark problems can be found in [2, 128] and the benchmarks can be downloaded with the S-TaLiRo version 1.6 [126], with the *Multi Parametric Toolbox (MPT)* version 3.0.

4.8 Related Work

In the following, we provide an overview of the related work in regard to several models of computation of increasing complexity.

4.8.1 Parameter Mining Over Finite State Machines

Parametric temporal logics were first defined over traces of finite state machines in [15]. The authors extend linear temporal logic to parametric temporal logic (PLTL), in which temporal operators can be sub-scripted, together with a direction $(\leq, >)$, by a variable ranging over natural numbers. There, the authors extend beyond the "yes/no" approach of traditional model checking to a framework where, for specifications such as $\phi = \Box(p \rightarrow \Diamond_{\leq x} q)$, they answer the questions "*for what values of* x

Table 4.1: Experimental results of Parameter Mining with S-TaLiRo. The parameters were mined by running 1000 tests.

Specification	S-TaLiRo		
	$f(\theta)$	Time (s)	Parameters Mined
$\phi_1^{AT}[\theta] = \neg\Diamond((v \geq 120) \wedge \Diamond_{[0,\theta]}(\omega \geq 4500))$	θ	135	$7.7s$
$\phi_2^{AT}[\theta] = \neg\Diamond((v \geq 120) \wedge \Diamond_{[0,\theta]}(v \geq 125))$	θ	138	$10.00s$
$\phi_3^{AT}[\theta] = \neg\Diamond((v \geq 120) \wedge \Diamond_{[0,\theta]}(\omega \geq 4500))$	θ	137	$7.57s$
$\phi_4^{AT}[\theta] = \neg\Diamond((v \geq 120) \wedge \Diamond_{[0,\theta]}(\omega \geq 4500))$	θ	132	$7.56s$
$\phi_5^{AT}[\vec{\theta}] = \Box((v \leq \theta_1) \wedge (\omega \leq \theta_2))$	$\|\vec{\theta}\|$	139	$\langle 138mph, 5981rpm\rangle$
	θ_1	137	$\langle 57mph, 6000rpm\rangle$
	θ_2	138	$\langle 180mph, 2910rpm\rangle$
	$max(\vec{\theta})$	138	$\langle 109mph, 6000rpm\rangle$
	$min(\vec{\theta})$	138	$\langle 154mph, 5300rpm\rangle$
$\phi_6^{AT}[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]}(v \geq 100) \wedge \Box(\omega \leq \theta_2))$	$\|\vec{\theta}\|$	144	$\langle 15.7s, 4820rpm\rangle$
	θ_1	142	$\langle 44.6s, 3598rpm\rangle$
	θ_2	138	$\langle 12.2s, 6000rpm\rangle$
	$max(\vec{\theta})$	140	$\langle 37.3s, 3742rpm\rangle$
	$min(\vec{\theta})$	142	$\langle 12.3s, 5677rpm\rangle$
$\phi_7^{AT}[\vec{\theta}] = \Box((v \leq \theta_1) \wedge (\omega \leq \theta_2)) \wedge \Diamond_{[0,\theta_3]}(v \geq 150) \wedge \Diamond_{[0,\theta_4]}(\omega \geq 4500)$	$\|\vec{\theta}\|$	145	$\langle 198mph, 4932rpm, 59.5s, 55s\rangle$
	$max(\vec{\theta})$	143	$\langle 129mph, 6000rpm, 48.9s, 28.3s\rangle$
	$min(\vec{\theta})$	142	$\langle 190mph, 5575rpm, 55.1s, 54.8s\rangle$
$\phi_8^{AT}[\vec{\theta}] = \Box((v \leq \theta_1) \wedge (\omega \leq \theta_2)) \wedge \Diamond_{[0,\theta_3]}(v \geq 150) \wedge \Diamond_{[0,\theta_4]}(\omega \geq 4500) \wedge \Box_{[\theta_5,60]}(v \geq 170) \wedge \Box_{[\theta_6,60]}(\omega \geq 4750)$	$\|\vec{\theta}\|$	146	$\langle 159mph, 5700rpm, 48.3s, 36.2s, 54.2s, 53.9s\rangle$
	$max(\vec{\theta})$	145	$\langle 85.9mph, 6000rpm, 3.8s, 38.8s, 44.5s, 51.5s\rangle$
	$min(\vec{\theta})$	143	$\langle 191mph, 4958rpm, 43s, 55.3s, 42s, 47.1s\rangle$
$\phi_{e3}^{P}[\theta] = \Box((\neg g_1 \wedge Xg_1) \rightarrow \Box_{(0,\theta]}\neg g_2)$	θ	2600	$0.1s$
$\phi_1^{S}[\theta] = \Box_{[0,60]}((g_2 \wedge Xg_1) \rightarrow \Box_{[0,\theta]}((t \leq \theta) \rightarrow g_1)$	θ	21803	$1.29s$

Legend: $\mathbf{f}(\tilde{\theta})$: the priority function used, $\phi_{\mathbf{i}}^{\mathbf{AT}}$: Specifications tested on the Automotive Transmission Model, $\phi^{\mathbf{P}}$: Specification tested on the Powertrain Model, $\phi^{\mathbf{S}}$: Specification tested on the Simuquest Enginuity high-fidelity Engine Model. The gray colored rows are first presented in [147] and are included for completeness.

Table 4.2: Experimental comparison of the parameter mining method ($\mathcal{A}$) presented in this thesis and the parameter synthesis method ($\mathcal{B}$) presented in [91].

Specification	Method	Parameters Mined		Time (s)	#Sim	#Rob
$\phi_2^S[\vec{\theta}] = \Box((v \leq \theta_1) \wedge (\omega \leq \theta_2))$	$\mathcal{A}$	137.1 mph	4870 rpm	20170	1000	1000
	$\mathcal{B}$	149.8 mph	4883 rpm	50017	2386	5130
$\phi_5^{AT}[\vec{\theta}] = \Box((v \leq \theta_1) \wedge (\omega \leq \theta_2))$	$\mathcal{A}$	100.2 mph	5987.6 rpm	106	1000	1000
	$\mathcal{B}$	137.5 mph	6000 rpm	253	2176	11485
$\phi_6^{AT}[\vec{\theta}] = \neg(\Diamond_{[0,\theta_1]}(v \geq 100) \wedge \Box(\omega \leq \theta_2)$	$\mathcal{A}$	21 s	3580 rpm	110	1000	1000
	$\mathcal{B}$	59.06 s	3296 rpm	397	3443	9718

Legend: **#Sim.**: the number of system simulations, **#Rob**: the number of robustness computations.

does the formula hold for the system being modeled?". It is important to note that timing here is of discrete nature (i.e., steps or transitions), e.g. $\Diamond_{\leq x} q$ states that "*in at most x steps q occurs*".

In [15], the authors provide several decidability results. Namely, given a system model K, represented by a Kripke structure, and a PLTL formula $\phi(\theta_1, ..., \theta_n)$ with one or more parameters, they provide algorithms for several problems. In the following, we emphasize two problems/results presented in [15].

a) Is there a parameter valuation α for which the system satisfies the specification, i.e., $K \models \phi(\alpha)$ (*emptiness*)? To answer this question, the authors exploit monotonicity results for PLTL specifications. In the following, we will consider the case for the always operator. For $\Box_{\leq y}$ specifications, it suffices to check whether the specification is satisfied at parameter value 0. If it does not hold, then that means that there are no satisfying valuations. As a result, the formula reduces to checking whether the model satisfies an LTL formula. For LTL model checking, we transform the LTL formula ϕ into a Büchi Automaton. This new automaton might have a state space

size which is exponential to the length of ϕ. Then, we check whether the composition of the Büchi Automaton and the finite machine has any accepting words. In fact, an upper bound on the time complexity is $\mathcal{O}(|K| \times 2^{|\phi|})$ [24].

b) The authors explain that parameter mining is also a decidable problem. Namely, for formulas with one or more parameters, they find a satisfying parameter valuation which maximizes the maximum (or maximizes the minimum) parameter value. Due to the the discrete nature of time imposed by the finite state machine, if given a parameter bound $[0, \theta]$, this problem can be solved by utilizing a binary search algorithm. For every value of the parameter, the formula can be flattened (ex. $\Box_{[0,2]}\phi = \phi \wedge X\phi \wedge XX\phi$) to an LTL formula and the previously discussed LTL-Büchi approach can be utilized to solve the problem. Note that this might increase the length of the formula significantly. The computational complexity of this approach depends on several factors. For each parameter, the binary search algorithm conducts $\log \theta$ checks, each costing $\mathcal{O}(|K| \times 2^{|\phi|})$.

Another related problem is model exploration for finite state machines. The problem was initially introduced by Chan in [32] under the term Temporal Logic Queries. The goal of model exploration is to help the designer achieve a better understanding and explore the properties of a model of the system. Namely, the user may pose a number of questions in a fragment of Computation Tree Logic (CTL) where the atomic propositions are replaced by a placeholder and the algorithm will try to find the set of atomic propositions for which the temporal logic formula evaluates to true. Since the first paper [32], several authors have studied the problem and proposed different versions and approaches [31, 35, 74, 136]. A related approach is based on specification mining over temporal logic templates [144] rather than special placeholders in a specific formula.

4.8.2 Parameter Mining Over Timed Automata

Timed automata (TA) are extensions of finite-state machines with clocks, which impose timing requirements on the transitions of accepting runs. It turns out that, most problems, such as universality (deciding whether the automaton accepts all timed words over its alphabet), inclusion (whether the language of automaton A is a subset of the language of automaton B) and equivalence (whether two automata are equivalent) are undecidable for timed automata. In [14], the authors prove this by reduction from halting problem for 2-counter machines (equivalent to Turing Machines). However, one of the advances in this area was the result that checking emptiness of the language of a timed automaton is decidable [14]. The key to the solution lies in the fact that the infinite state space can be partitioned in finitely many equivalence classes. Namely, they show that a finite state region automaton that accepts exactly the set of words accepted by a timed automaton can be constructed. Note that, this result, in conjunction with the fact that Timed Automata are closed under intersection, means that we can conduct model checking. In more detail, given a TA A of the system, and a TA B representing "bad behaviors" that are undesirable in the system, we can find the product automaton $C = A \times B$. If the language of C is not empty, then there exists a bad behavior in the system. The complexity of deciding emptiness is exponential in the number of clocks and the length of the constant in the timing constraints. It is known to be PSPACE-Complete [14].

The problem of Parametric Temporal Logic over timed automata attracted the interest of many researchers who were seeking to extend verification results to real-time systems. In [17], the authors present Parametric Timed Automata, with the goal to provide answers to problems as, "given a real-time system S, one may wish to verify a property p of the system as long as the deadline d of an action is less than the

delay r in receiving an acknowledgment, $r > d$". In that work, rather than temporal logic, the authors use timed automata with parametric timing constraints. They provide results on the problem of whether there exists parameters (language emptiness problem) so that the Parametric Timed Automaton has an accepting run and they show that the results are closely related to the number of clocks. For automata with one parametrically constrained clock, emptiness can be decided. Namely, it can be determined whether there exists a parameter such that the language of the automaton is nonempty. However, they show that for three (or more) parametrically constrained clocks, the problem is recognizable but undecidable. Other decision problems with regards to parametric timed automata are studied in [28, 51].

In [28], the authors investigate L/U Timed Automata, where the timing constraint appear either as a lower bound or as an upper bound. In other words, the set of the parameters which can occur as a lower bound in a parametric clock constraint is disjoint from set of parameters which can occur as an upper bound. For the class of L/U Timed automata, the emptiness and universality problems are decidable. The problems are PSPACE-Complete. Also, the authors provide a parameter synthesis algorithm similar in nature to the solution provided in [15].

In [52], the authors present Parametric Metric Interval Temporal Logic (PMITL). There, given a L/U Timed Automata A and a PMITL ϕ specification the authors provide algorithms to determine whether there exists a parameter valuation v such that there is a timed sequence in L(A) that satisfies ϕ with parameter v. Since PMITL is closed under semantic negation, we can conduct model-checking using this approach. The computational complexity of the problem is in EXPSPACE and follows from the satisfiability and model-checking problems for MITL formulas [16].

4.8.3 Parameter Mining Over Hybrid Systems

Hybrid systems are modeled by hybrid automata (HA). HA are extensions of finite state machines with the ability to model continuous behavior through algebraic and differential equations. One subclass of Hybrid Automata are Linear Hybrid Automata (LHA). A subclass of LHA are Timed Automata (TA). Therefore undecidability results for TA immediately hold for HA. In [12], the authors prove that the reachability problem for LHA is undecidable. The result follows from the undecidability of the halting problem for nondeterministic 2-counter machines. Therefore, since it is undecidable for LHA it is undecidable for HA in general.

The model checking of parametric temporal logic specifications over hybrid systems utilizes similar approaches as these methods do. The parametric identification of temporal properties over signals is presented in [21]. Where given a signal, the authors compute the subset of the parameter space that renders the formula satisfied by the trace. Although the logic is defined over dense-time real valued signals, the signal analyzed is a sampled signal which is linearly interpolated. In [91], the authors present a simulation based approach for mining state and timing parameters for Parametric Metric Temporal Logic specification over hybrid systems. There, the authors provide a parameter synthesis algorithm for Parametric Signal Temporal Logic (PSTL), a similar formalism to MTL. To conduct parameter synthesis for multiple parameters, a binary search is utilized to set the parameter value for each parameter in sequence. After a set of parameters is proposed, a stochastic optimization algorithm is utilized to search for trajectories that falsify the specification. If it fails to do so, the algorithm stops, otherwise this two step process continues until the termination condition is met. The authors in [65, 124] define a parametric temporal logic called quantifier free Linear Temporal Logic over real valued signals. However,

they focus on the problem of determining system parameters such that the system satisfies a given property rather than on the problem of exploring the properties of a given system.

In [99], the authors present an inference algorithm that finds temporal logic properties of a system from data. The authors introduce a reactive parametric signal temporal logic and define a partial order over it to aid the property definition process.

In the following, a detailed comparison between the method presented in this thesis and the work in [91] is presented.

4.8.4 Comparison to the Parameter Synthesis Method

In [91], the authors provide a parameter synthesis algorithm for Parametric Signal Temporal Logic (PSTL), a similar formalism to MTL. To conduct parameter synthesis for multiple parameters, a binary search is utilized to set the parameter value for each parameter in sequence. After a set of parameters is proposed, a stochastic optimization algorithm is utilized to search for trajectories that falsify the specification. If it fails to do so, the algorithm stops, otherwise this two step process continues until the termination condition is met.

In the following, we present three main differences between the method proposed here ($\mathcal{A}$) and the method proposed in [91] ($\mathcal{B}$). First, $\mathcal{A}$ is a best effort algorithm for which the termination condition is the number of tests the system engineer is interested to conduct. Clearly, the more tests, the better the search space is explored. Since the parameter mining problem is presented as a single optimization problem, runtime is not directly affected by the number of parameters in the specification. In contrast, in $\mathcal{B}$, the runtime of the algorithm through binary search is affected by the number of parameters in the PSTL formula. For each iteration of the binary

search, multiple robustness computations have to be conducted, which for systems that output a large trace and contain complex specifications, could become costly. The second step in $\mathcal{B}$ is the falsification of the parameters proposed. This algorithm needs to be performed on every iteration, until a falsification is found. If a falsifying trajectory is not found, the stopping condition is met and the parameters are returned. Second, in $\mathcal{A}$, the parameters returned are the "best" parameters for which a falsifying trajectory is found. In $\mathcal{B}$, the proposed parameters are parameters for which no falsifying trajectory is found. Proving that a specification holds for hybrid systems, in general, is undecidable and, therefore the failure to find a falsifying trajectory does not imply that one does not exist. Third, in $\mathcal{A}$, through the priority function, we enable the system engineer to have flexibility when assigning weights and priorities to parameters. In $\mathcal{B}$, parameter synthesis through binary search implicitly prioritizes one parameter over others.

We compare the two methods using the Simuquest Enginuity high-fidelity Engine model and the Automotive Transmission model. To enable the comparison of the two methods, we have implemented the $\mathcal{B}$ method in S-TaLiRo. Note that the simulation time is 60s. The experimental results are presented in Table 4.2. For the $\mathcal{A}$ method, the number of simulations and robustness computations is predefined. On the other hand, for the $\mathcal{B}$ method, these numbers vary following the reasons presented in the previous paragraph. As a result, the difference in computation time between the two methods is significant. Due to the significant differences between the two algorithms, in terms of guarantees provided, it is not possible to compare the quality of the solutions. While the mined parameters with method $\mathcal{A}$ guarantee falsification of the specification, the mined parameters with method $\mathcal{B}$ do not.

The results for the AT model can be reproduced by running the experiments in S-TaLiRo version 1.6 [126], with *Multi Parametric Toolbox (MPT)* version 3.0.

4.9 Conclusion

An important stage in Model Based Development (MBD) of software for CPS is the formalization of system requirements. We advocate that Metric Temporal Logic (MTL) is an excellent candidate for formalizing interesting design requirements. In this thesis, we have presented a solution on how we can explore system properties using Parametric MTL (PMTL) [21]. Based on the notion of robustness of MTL [68], we have converted the parameter mining problem into an optimization problem which we approximate using S-TaLiRo [19, 126]. We have presented a method for mining multiple parameters as long as the robustness function has the same monotonicity with respect to all the parameters. We demonstrated that our method can provide interesting insights to the powertrain challenge problem [40]. Finally, we utilized the method on an industrial size engine model and examples from related works.

4.10 Future Work

There are two possible extensions of this work. One, as the number of parameters in parametric MTL formulas increases, it becomes more and more challenging to visualize the parameter falsification domain. With the current implementation, our framework can visualize up to three parameters. It would be interesting to develop visualization methods for the Pareto front with more than three parameters. Methods that enable the practitioner to explore the falsification domain and inspect the relationship between parameters. Two, the framework presented here enables parameter mining for specifications that satisfy the monotonicity property with respect to the robustness of the parametric MTL formula. An interesting problem would be to consider cases of parameter mining for mixed monotonicity specifications.

Chapter 5

PRACTICAL ALGORITHMS FOR CONFORMANCE TESTING

5.1 Introduction

One of the benefits of Model-Based Design is that it enables the iterative development of CPS. During this process, a series of models and implementations are developed, with varying fidelity, for the same underlying CPS. One of the problems that arises, is to quantify how "close" two systems are to each other. We refer to this as the conformance problem.

Due to the non-linear, hybrid nature of these systems, in general, it is not possible to certify that two systems are conformant to each other. We can only detect non-conformance. In this work, we review and adopt the semi-formal approach for conformance testing of CPS presented in [3]. We propose an automated black/grey box conformance testing framework. In this framework, we simulate two systems with the same input signal and then evaluate the output signals with our notion of conformance. A stochastic optimizer is then utilized to select the next input signal. Our method returns the most non-conformant behavior found after running a predetermined number of tests. Our framework is based on a notion of closeness between signals that encapsulates both spatial and temporal differences. Spatial difference by itself is not sufficient since in many cases, two system trajectories, due to factors such as jitter, may be slightly shifted. This shift is generally acceptable in model based design and therefore we do not wish to classify this as non-conformant behavior.

Finally, we introduce and integrate code coverage metrics for conformance testing

of Cyber-Physical Systems. By doing so, we can ensure that particular locations of the control logic have been covered in the testing process.

We illustrate our methods with an industrial scale high-fidelity engine model from Toyota.

In this chapter:

- We review and adopt the notion of conformance introduced in [3].
- We introduce code coverage methods for conformance testing of CPS.
- We illustrate our work on an industrial scale high-fidelity engine model from Toyota.

5.2 Problem Formulation

In the conformance testing framework, we consider two systems: Σ_1 for the Model and Σ_2 for the implementation. Utilizing the notation from Section 2.1, both systems can be viewed as functions:

$$\Delta_{\Sigma_1} : X_0 \times \mathbf{U} \to Y^N \times \mathfrak{T}$$

$$\Delta_{\Sigma_2} : X_0 \times \mathbf{U} \to Y^N \times \mathfrak{T}$$

The systems take as an imput the same initial conditions $x_0 \in X_0$ and input signals $u \in \mathbf{U}$ to produce:

Output signal $\mathbf{y}_1 : N \to Y$ and timing function $\tau_1 : N \to \mathbb{R}_+$ for Δ_{Σ_1}.

Output signal $\mathbf{y}_2 : N \to Y$ and timing function $\tau_2 : N \to \mathbb{R}_+$ for Δ_{Σ_2}.

We recall that $\mu_1 = (\mathbf{y}_1, \tau_1)$ and $\mu_2 = (\mathbf{y}_2, \tau_2)$ are referred to as *timed state sequences* (TSS). TSS is a widely accepted model for reasoning about real-time systems [11]. A timed state sequence can represent a computer-simulated trajectory of a CPS

or the sampling process that takes place when we digitally monitor physical systems. We remark that a timed state sequence can represent both the internal state of the software/hardware (usually through an abstraction) and the state of the physical system.

Our high level goal is to determine whether there exists a pair of (initial conditions, input signal) that cause the model and its implementation to produce significantly different outputs; and if such a pair exists, to find it and present it to the user.

We define conformance over two trajectories as follows.

Definition 5.2.1 (*$(\delta, \varepsilon) - closeness$* **[3]**) *Two timed state sequences, or* trajectories, *$\mu_1 = (\mathbf{y}_1, \tau_1)$ and $\mu_2 = (\mathbf{y}_2, \tau_2)$ are $(\delta, \varepsilon) - close$ if*
(a) for all $i \in N$, there exists $k \in N$ where

$$|\tau_1(i) - \tau_2(k)| < \delta \text{ and } \|\mathbf{y}_1(i) - \mathbf{y}_2(k)\| < \varepsilon$$

(b) for all $i \in N$, there exists $k \in N$ where

$$|\tau_2(i) - \tau_1(k)| < \delta \text{ and } \|\mathbf{y}_2(i) - \mathbf{y}_1(k)\| < \varepsilon$$

If these conditions are met we say that μ_1 and μ_2 are conformant with degree (δ, ε).

The definition says that within any time window of size 2δ, there must be a time when the trajectories are within ε or less of each other. Allowing some 'wiggle room' in both time and space is important for conformance testing: when implementing a Model, there are inevitable errors. These are due to differences in computation precision, clock drift in the implementation, the use of inexpensive components, unmodeled environmental phenomena, etc, leading to the Implementation's output to differ in value from the Model's output, and to have different timing characteristics.

Problem 5.2.1 (System Conformance) *Given two Systems Σ_1 and Σ_2, for every $\mu_1 \in \mathcal{L}(\Sigma_1)$ and $\mu_2 \in \mathcal{L}(\Sigma_2)$, show that μ_1 and μ_2 are conformant with degree (δ, ε), where*

$$\mathcal{L}(\Sigma_i) = \{(\mathbf{y}_i, \tau_i) \mid \exists x_0 \in X_0 \,.\, \exists u \in \mathbf{U} \,.\, (\mathbf{y}_i, \tau_i) = \Delta_{\Sigma_i}(x_0, u)\}.$$

In many cases, the implementation of the model is given as a black-box or gray-box where the internals of the model are hidden or partially known, respectively. In this case, we can only analyze the system by observing the input-output behavior. For gray-box systems, we might have partial information such as a subset of states of the system or the current locations or modes of the underlying control logic of the system. Therefore, in general, Prob. 5.2.2 does not lend itself to complete analytical techniques. However, since we can detect non-conformant behavior, we can pose this problem as a falsification problem. Namely, we can convert this problem to a testing problem where we search for non-conformant behavior.

Problem 5.2.2 ((δ, ε) - Conformance Falsification) *Given two Systems Σ_1 and Σ_2, find (x_0, u) s.t. the resulting traces $\mu_1 = \Delta_{\Sigma_1}(x_0, u)$ and $\mu_2 = \Delta_{\Sigma_2}(x_0, u)$ are non-conformant with degree (δ, ε).*

An overview of the solution is presented in Fig. 5.1. Given two systems, the sampler produces a point x_0 from the set of initial conditions, and an input signal u. The initial conditions and input signals are passed to the system simulator for both systems which returns two trajectories. The trajectories are then analyzed and a conformance degree is returned. Here, either δ or ϵ are fixed a priori. The conformance degree is used by the stochastic sampler to decide on next initial conditions and inputs. The process terminates once a maximum number of tests is reached. The method returns non-conformant outputs (if they are found) along with the corresponding initial conditions and input signals.

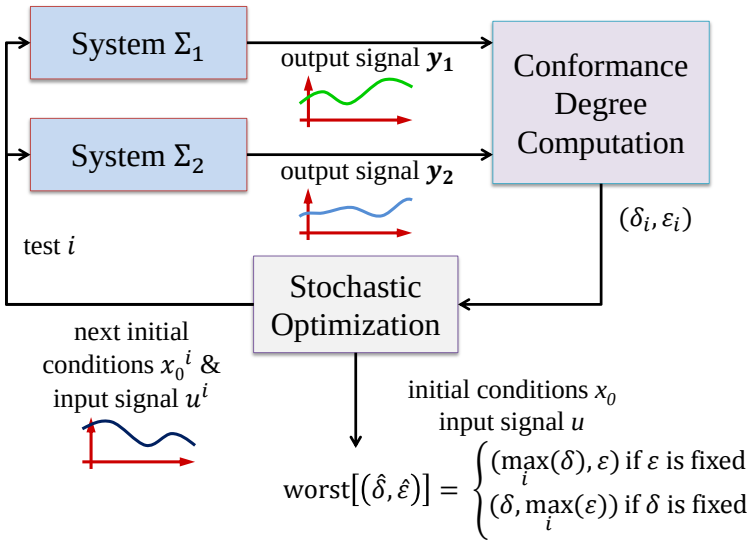


Figure 5.1: Overview of the solution to Prob. 5.2.2: the conformance falsification problem for CPS.

5.3 Code Coverage for CPS

In software testing, many techniques and standards have been developed to ensure that the developed software operates as intended. Techniques ranging from static/dynamic analysis [60] and fault injection [113] to mutation testing [18] and code coverage [140]. In the following we investigate code coverage techniques for conformance falsification of CPS.

Code coverage is of particular importance when dealing with CPS. In order to ensure that the testing process includes parts of the system that are rarely visited/tested. In closed-loop testing, a CPS is composed of a controller and a plant, as illustrated in Fig. 5.2. The physical representation of the system that needs to be controlled is called the plant. The controller processes signal data and generates an actuation signal using a control law that has a specific objective. From the test-

ing perspective, the control logic can be instrumented automatically [57] and we can conduct coverage over the states of the resulting finite state machine.

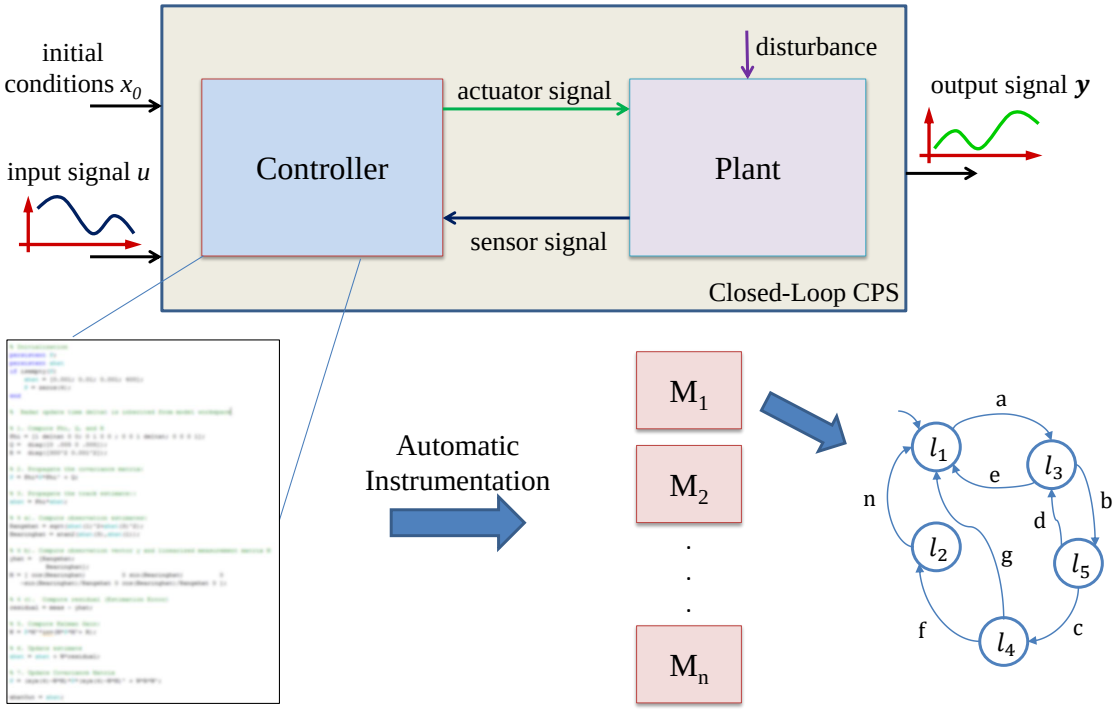


Figure 5.2: Typical closed-loop cyber-physical system. The discrete logic of the controller code is automatically instrumented and modeled using Extended Finite State Machines (EFSM) [10]. Namely, for each conditional statement code block in the controller code, an EFSM is generated.

Example 5.3.1 (Model Instrumentation) *We illustrate the system instrumentation process with our running example from Section 2.6.3. Given a simulink model of the hybrid nonlinear system, by analysing the function block of the controller code, we are able to extract the control logic of the controller. In addition, we modify the system so that we are able to observe in which location the system is in when analyzing the output trajectories. This process is illustrated in Fig. 5.3.*

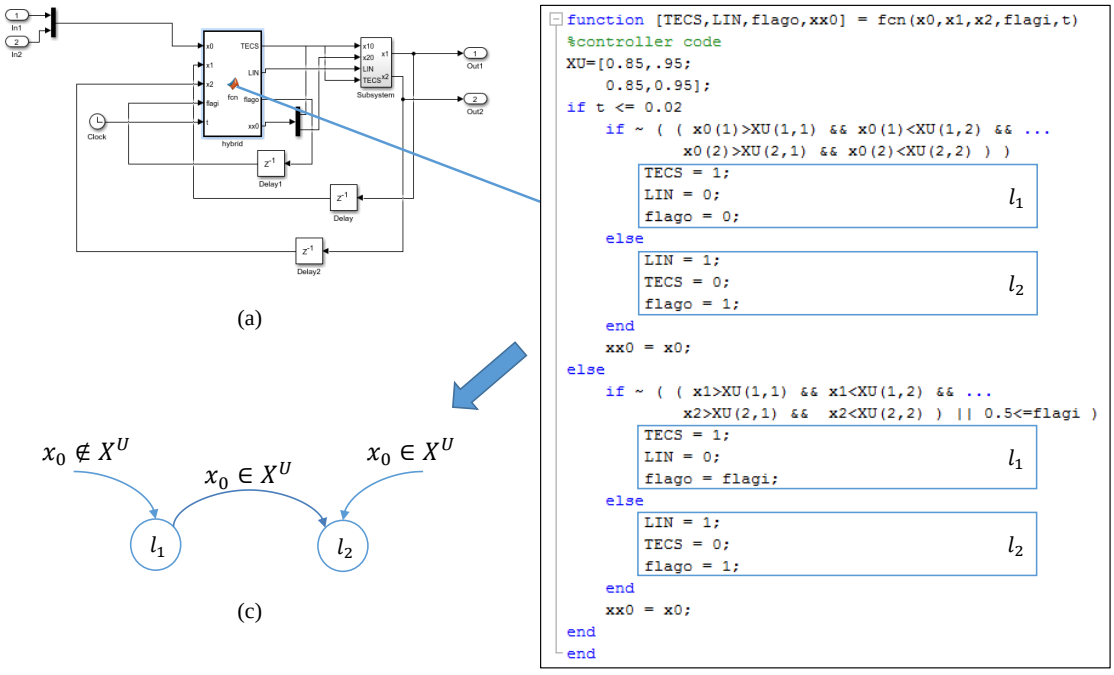
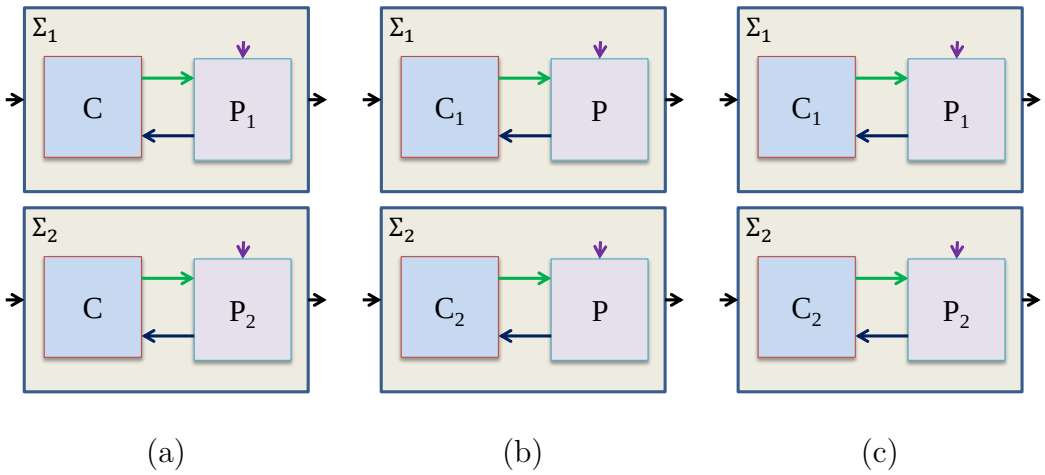


Figure 5.3: Model instrumentation of the hybrid nonlinear system (HS) presented in Section 2.6.3. (a) Simulink model of the HS system. (b) Controller code of the HS simulink model inside the function block. (c) Automatically extracted control logic from the controller code.

In a typical V process in MBD, as presented in Fig. 1.2, a model is developed iteratively. In this iterative process, depending on the development stage, various conformance testing scenarios may be encountered. In Fig. 5.4, we list the three possible scenarios. In the following, we will provide a conformance testing solution for each. First, we will consider the case where the controller is the same in both the model and the implementation but the plant is different. After, we will consider the case when the plant is the same but the controller is different.



(a) (b) (c)

Figure 5.4: Conformance testing scenarios in the V process in MBD. (a) Two systems with the same controller and different plant. (b) Two systems with different controllers and same plant. (c) Two systems with different controllers and different plants.

5.4 Controller Coverage with Different Plants

In the right hand side of the V process, presented in Fig. 1.2, the model gets developed and tested through Model in the loop (MIL), Software in loop (SIL), Processor in loop (PIL), Hardware in loop (HIL) and implementation stages. Typically, the main components that change in this process are the plant and the controller of the model: from the most abstract (Model), to the most realistic (Implementation). In this section, we consider the case where the controller remains the same in both the model and implementation but the plant is different.

First, we extract the controller code through an instrumentation process to obtain the structure of the control logic of the controller but not the actual numerical computations. In this process, for every conditional statement code block, an Extended Finite State Machine (EFSM) M_i is generated. Depending on the size of the resulting machines, the practitioner may desire to test all or some specific combinations of

locations for each M_i. In most practical applications, the number of machines and locations is large and therefore testing all the combinations is not a feasible task. We discuss this case further in Section 5.6. In the rest of this section, we assume that the practitioner has a predetermined combination of locations that they would like to cover.

For a particular combination of locations, the problem is posed as an optimization problem. We modify the solution presented in Fig. 5.1 by changing the cost function such that the system is driven towards the target locations. We accomplish this by utilizing the hybrid distance metric introduced in [2]. We refer the reader to [2] for the technical details. The intuition behind the metric is as follows. When the current location is different from the target location, then the distance is the distance to the closest guard that will enable the transition to the next control location that reduces the path distance. When the system is in the target locations, the cost function is set to the (δ, ε) conformance metric. In this way, not only do we attempt to reach the target locations, but also attempt to find the most non-conformant behavior there.

5.5 Coverage with Different Controllers

Throughout the entire V-process of MBD, it is often the case that the controller is either modified, abstracted and refined depending on the application. When we encounter this case, we would like to ensure conformance between the previous and the new version of the controller. To achieve this, coverage of both controller locations is necessary. Given system Σ_1 with controller C_1 and Σ_2 with controller C_2, we extract the controller logic through an automatic process $\mathcal{A}(C_1) = M$ and $\mathcal{A}(C_2) = N$. Next, we conduct conformance testing with coverage over the product automaton $\mathcal{P} = M \times N$ which contains the states of both controller locations. The search process follows similarly as in Sect 5.4.

5.6 Covering Arrays for Code Coverage of CPS

In many cases, the number of conditional statements inside the controller code is very large, and generating system simulations to cover all the possible interactions is not feasible. In this section, we propose the use of covering arrays [43] to enable conformance testing of a subset of interactions for the conditional statements reached in the testing process. In particular, we will utilize methods from the Covering Array literature, to improve on existing conformance testing methods. Covering arrays have been successfully utilized in a variety of applications such as blackbox testing of hardware systems, learning an unknown Boolean function, sequence alignment of DNA, compressing inconsistent data, etc.

For conformance testing, consider the following example. A controller has 4 conditional statement blocks where M_1 has 15 locations, M_2 has 10 locations, M_3 has 8 locations and M_4 has 25 locations. To check all interactions of the locations we would have to attempt to reach $15 \times 10 \times 8 \times 25 = 30,000$ combinations of locations with a predetermined number of tests each. This approach becomes infeasible very quickly. However, we can utilize covering arrays to drastically decrease the number of interactions of locations. As a trade-off, we do not test for all interactions of locations but cover n-interactions. For example, a 3-interaction covering array reduces the number of interactions of locations to 3750, while a 2-interaction covering array reduces it to 376 location interactions. Furthermore, there is research that shows that a significant number of bugs in software can be found within n-interactions [105]. With covering arrays, the number of sequences of locations needed for coverage, for a fixed number of interactions, grows logarithmically in the number of parameters [105].

Covering arrays have been studied extensively over the last 40 years. Researchers have been primarily focused on the problem of determining the minimum size of

covering arrays. For an overview, see [75]. In general, there has been significant progress in determining upper bounds for the minimum size of the covering arrays. Furthermore, many algorithms [69, 34, 43] and tools [42] have been developed for the construction of covering arrays. In our framework, we utilize the AETG covering array generation system [42].

With covering arrays, we can define seeds, or combinations of locations that must be covered by the covering array. We can define avoids, or interactions that do not need to be covered. We can define constraints, which are interactions that should not be covered. We can also define the strength of some factors over others. All this allows for a very flexible framework when conducting conformance testing.

5.7 Case Study: Toyota Engine Controller

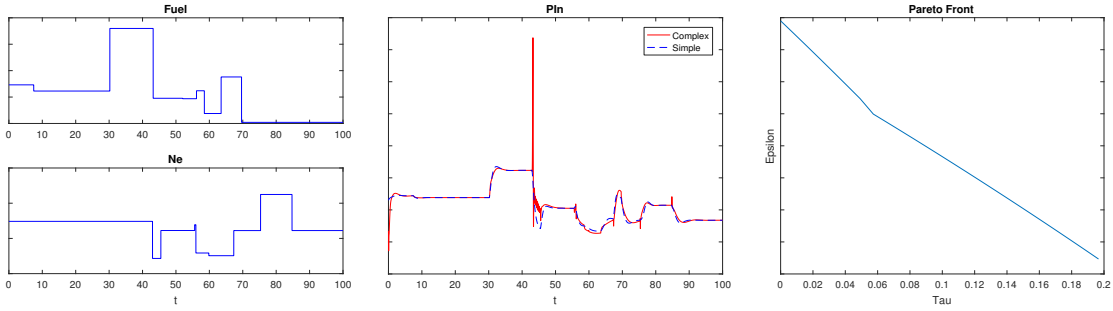
We consider two models of varying fidelity of an internal combustion engine. The models are academic, but of industrial complexity, and are provided by the Toyota Technical Center [1] . The first model Σ_1 is a high-fidelity engine plant and controller while the second one Σ_2 is a simplified, polynomial approximation of the plant model in Σ_1. In this case, with conformance testing, we aim to explore how close the two models are and whether we can utilize Σ_2 for model predictive control [38, 39, 85, 86]. Both Σ_1 and Σ_2 are modeled in Matlab/Simulink. Σ_1 has 1878 blocks, including 10 integrator blocks, 47 lookup tables, 19 saturation blocks, 27 switch blocks, and 44 subsystem blocks. The model of the plant has 11 continuous and 68 discrete states. The controller is defined in a function block and contains ≈ 500 lines of code with multiple if-else conditional statements. Model Σ_2 has 1858 blocks, including 47 lookup tables, 17 saturation blocks, 27 switch blocks and 49 subsystem blocks with

[1] We note that due to confidentiality agreements, we have removed the unit measurements of the y-axes from the figures presented in this section.

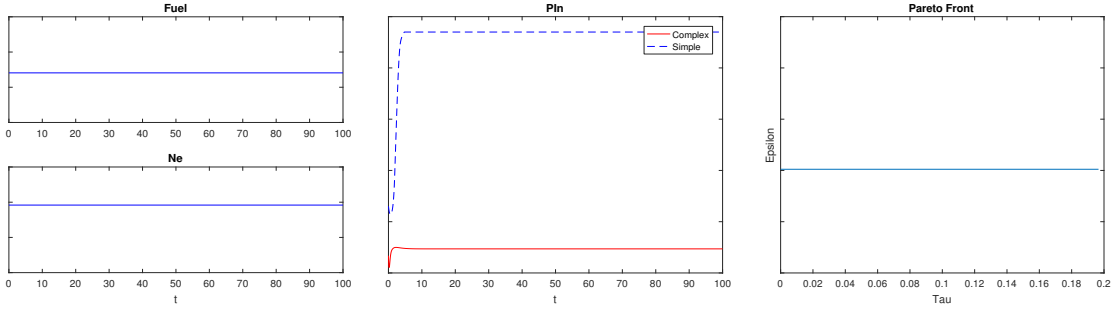
no continuous states and 60 discrete states. Both Σ_1 and Σ_2 have two inputs and one output each. The inputs are the *Fuel Inject Rate* and the *Engine Speed* (*Ne*). The output is the Pressure of Intake Manifold (*Pin*). In the following, we run several experiments with different optimization functions. We also provide partial coverage of the input space as well as branch coverage for the controller code.

5.7.1 Simulated Annealing

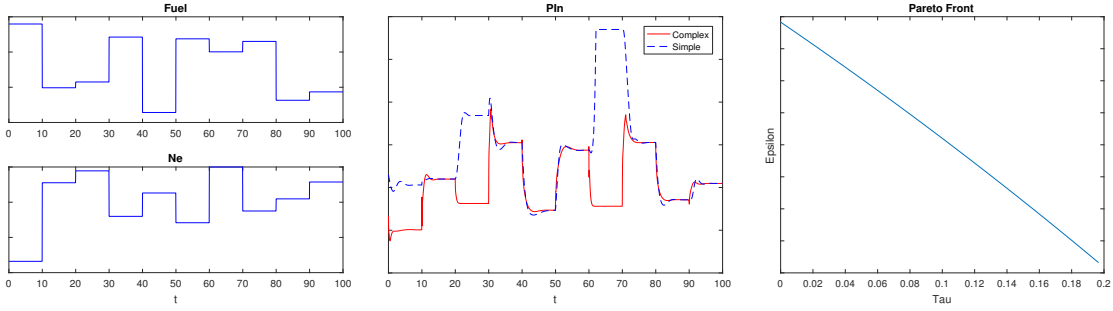
The search space over system inputs is defined over control points which are interpolated to generate an input signal as defined in Section 2.5. Here, both inputs have 10 control points. The search space also includes timing of the control points. Since the initial and terminal timing control points are fixed at the start and end of the simulation, respectively, for each input, we have an additional 8 search variables. The signals are interpolated through a piecewise constant interpolation function. An example input signal is presented in Fig. 5.5 (a) (Left). In total, we have 36 search variables. The stochastic optimizer utilized is a simulated annealing algorithm [2]. After 1000 tests, we found the system inputs that generate the outputs in Fig. 5.5. In the left figure, the resulting system inputs are presented. Figure 5.5 (a) (Middle) shows the system outputs, where the red out is from the complex model Σ_1, and the blue line is from the simplified model Σ_2. The figure on the right is the Pareto front over (δ, ϵ) which illustrates the ϵ difference over the δ range. The ϵ value at $\delta = 0$ represents the instantaneous difference between the outputs of the two models at the same time t, while the ϵ value at $\delta = 0.2$ represents the largest difference between the outputs of the two models while comparing the values of the two signals in a moving window of width 0.2s. Note that the Pareto Front is over 1000 tests and is therefore an under-approximation of the *true* Pareto Front over all system behaviors. We can guarantee that (δ, ϵ) is at least as large as shown in Fig. 5.5 (a) (Right).



(a) Simulated Annealing



(b) Grid Search



(c) Controller Branch Coverage

Figure 5.5: Experimental results for various conformance testing methods. Left: System inputs for the *Fuel Inject Rate* and the *Engine Speed* (*Ne*). Middle: System output for the *Pressure of Intake Manifold Pin*. Right: Pareto front over all tested system behaviors that illustrates the ε difference over the δ range.

5.7.2 Grid Search

One of the goals in the conformance testing process is to provide a level of input search space coverage. To do so, we developed a grid search algorithm which divides the input search space in a grid. Formally, search is conducted over the following set $S = \{(x, y) : x \in [min(U_1) : \frac{range(U_1)}{g-1} : max(U_1)]\ and\ y \in [min(U_2) : \frac{range(U_2)}{g-1} : max(U_2)]\}$, where U_1, U_2 are input signals for inputs 1 and 2, respectively. Here, g is the granularity of the grid. The results of the grid search algorithm are presented in Fig. 5.5 (b). The middle figure shows highly non-conformant behavior between Σ_1 and Σ_2, indicating a possible singularity in either model at this particular input. Here, since the maximal difference between the two trajectories is constant over a period of $2 \times \delta$, the Pareto Front on the right figure is flat.

In the future, we plan to implement a grid search algorithm which includes several grids with respect to time and the input signals are interpolated linearly between them. This way, we can also study switching behaviors of the system.

5.7.3 Controller Branch Coverage

The next step in the analysis is to conduct conformance testing while making sure that we have controller branch coverage. We will consider three if-else blocks to be of particular importance. Namely M_1 with 12 branches, M_2 with 2 branches, and M_3 with 4 branches. These if-else blocks are instrumented automatically from the Simulink model. The instrumentation process extracts the branch information from the Embedded Matlab code function block and passes it to S-TaLiRo for conformance testing. We follow the approach presented in Section 5.4 which is related to the testing approach in [57].

After running our algorithm, we found non-conformant behavior in locations

$(6, 1, 4)$ for M_1, M_2, and M_3. The results are presented in Fig. 5.5 (c). The target locations $(6, 1, 4)$ were reached after 803 tests. After reaching the target locations, the stochastic optimizer was focused on maximizing the (δ, ε) metric between trajectories.

Following our testing results, Ken Butts, an Executive Engineer from the Powertrain Control Department at Toyota Technical Center provided the following statement:

"*Their tool has pointed out where the high-fidelity model is fragile and producing erroneous results. It is good to know that the polynomial model performs well at these cases.*"

5.8 Related Works

Conformance notions have been studied in the past. Tretmans [141] defined an Input-Output conformance (IOCO) notion for discrete labeled transition systems. This notion is defined as a relationship where the implementation does not generate an output that is not producible by the specification. Also, the implementation always produces an output when it is required by the specification. Later, Van Osch [117] extended IOCO to hybrid transition systems (HTS) by incorporating continuous-time inputs.

In [145], the authors extend the work by [141] where the implementation is a black box that generates trajectories. In this framework, the specification is represented as a timed automaton. Here, every trajectory in the language of the implementation needs to satisfy the timed automaton representing the specification.

In the work by Brandl et al. [29] the authors present a method for conformance checking through qualitative reasoning techniques. The method utilzes mutation-based test case generation on action systems for IOCO [141] conformance checking.

In [49], the authors propose a notion of conformance based on the Skorokhod met-

ric. This notion captures both timing and spacial differences between trajectories and supports transference of properties in the development process. However, the physical interpretation and computation of the Skorokhod distance is not as straightforward as for the (τ, ϵ) metric [3].

In [94], the authors provide an overview of confrmance testing methods for hybrid systems. They compare different notions of robustness for conformance testing and they list current challenges in the area.

5.9 Conclusions and Future Work

In this chapter, we presented a conformance testing framework to test how "close" two systems are. We presented a black/gray box framework that includes controller code coverage methods for improved testing. Next, we discussed the use of covering array techniques to dramatically reduce the number of tests necessary to test n-interactions of the branching conditions in the controller code. Finally, we demonstrated our methods with prototype high-fidelity models from Toyota.

In the future, we plan to incorporate a learning algorithm to obtain the regular language of the transitions between location n-tuples and estimate the accompanying probabilities for non-conformant behavior. This could possibly be represented as a Markov Decision Process. This would make it easy to observe "problematic" or non-conformant location changes and facilitate the debugging process.

BIBLIOGRAPHY

[1] H. Abbas and G. Fainekos. Linear hybrid system falsification through local search. In *Automated Technology for Verification and Analysis*, volume 6996 of *LNCS*, pages 503–510. Springer, 2011.

[2] H. Abbas, G. E. Fainekos, S. Sankaranarayanan, F. Ivancic, and A. Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Transactions on Embedded Computing Systems*, 12(s2), May 2013.

[3] H. Abbas, B. Hoxha, G. Fainekos, J. V. Deshmukh, J. Kapinski, and K. Ueda. Conformance testing as falsification for cyber-physical systems. Technical Report arXiv:1401.5200, January 2014.

[4] H. Abbas, B. Hoxha, G. Fainekos, J. V. Deshmukh, J. Kapinski, and K. Ueda. Wip abstract: Conformance testing as falsification for cyber-physical systems. In *Cyber-Physical Systems (ICCPS), 2014 ACM/IEEE International Conference on*, pages 211–211. IEEE, 2014.

[5] H. Abbas, B. Hoxha, G. Fainekos, and K. Ueda. Robustness-guided temporal logic testing and verification for stochastic cyber-physical systems. In *Cyber Technology in Automation, Control, and Intelligent Systems (CYBER), 2014 IEEE 4th Annual International Conference on*, pages 1–6. IEEE, 2014.

[6] E. Ábrahám-Mumm, U. Hannemann, and M. Steffen. Verification of hybrid systems: Formalization and proof rules in pvs. In *Engineering of Complex Computer Systems, 2001. Proceedings. Seventh IEEE International Conference on*, pages 48–57. IEEE, 2001.

[7] T. Akazaki and I. Hasuo. Time robustness in mtl and expressivity in hybrid system falsification. In *International Conference on Computer Aided Verification*, pages 356–374. Springer, 2015.

[8] A. Alfonso, V. Braberman, N. Kicillof, and A. Olivero. Visual timed event scenarios. In *Proceedings of the 26th Int. Conference on Software Engineering*, pages 168–177. IEEE Computer Society, 2004.

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

[10] R. Alur. *Principles of cyber-physical systems*. MIT Press, 2015.

[11] R. Alur, C. Courcoubetis, and D. Dill. Model-checking for real-time systems. In J. Mitchell, editor, *5th Annual IEEE Symp. on Logic in Computer Science (LICS)*, pages 414–425. IEEE Computer Society Press, June 1990.

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

[13] R. Alur, C. Courcoubetis, T. A. Henzinger, and P.-H. Ho. *Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems*. Springer, 1993.

[14] R. Alur and D. L. Dill. Theory of timed automata. *Theoretical Computer Science*, 126(2):183–235, 1994.

[15] R. Alur, K. Etessami, S. La Torre, and D. Peled. Parametric temporal logic for model measuring. *ACM Trans. Comput. Logic*, 2:388–407, July 2001.

[16] R. Alur, T. Feder, and T. A. Henzinger. The benefits of relaxing punctuality. *Journal of the ACM*, 43:116–146, 1996.

[17] R. Alur, T. A. Henzinger, and M. Y. Vardi. Parametric real-time reasoning. In *Proceedings of the twenty-fifth annual ACM symposium on Theory of computing*, pages 592–601. ACM, 1993.

[18] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2016.

[19] Y. S. R. Annapureddy, C. Liu, G. E. Fainekos, and S. Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Tools and algorithms for the construction and analysis of systems*, volume 6605 of *LNCS*, pages 254–257. Springer, 2011.

[20] E. Asarin, O. Bournez, T. Dang, and O. Maler. Approximate reachability analysis of piecewise linear dynamical systems. In *Hybrid Systems: Computation and Control*, volume 1790 of *LNCS*, pages 21–31. Springer, 2000.

[21] E. Asarin, A. Donzé, O. Maler, and D. Nickovic. Parametric identification of temporal properties. In *Runtime Verification*, volume 7186 of *LNCS*, pages 147–160. Springer, 2012.

[22] M. Autili, P. Inverardi, and P. Pelliccione. Graphical scenarios for specifying temporal properties: an automated approach. *Automated Software Engineering*, 14(3):293–340, 2007.

[23] S. Bacherini, A. Fantechi, M. Tempestini, and N. Zingoni. A story about formal methods adoption by a railway signaling manufacturer. In *FM 2006: Formal Methods*, pages 179–189. Springer, 2006.

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

[25] S. Bak, S. Bogomolov, and T. T. Johnson. Hyst: a source transformation and translation tool for hybrid automaton models. In *Proceedings of the 18th International Conference on Hybrid Systems: Computation and Control*, pages 128–133. ACM, 2015.

[26] N. M. Bardh Hoxha and G. Fainekos. Vispec : A graphical tool for elicitation of mtl requirements. In *Proceedings of the 2015 IEEE/RSJ International Conference on Intelligent Robots and Systems*, 2015.

[27] A. Bhatia and E. Frazzoli. Incremental search methods for reachability analysis of continuous and hybrid systems. In *Hybrid Systems: Computation and Control*, volume 2993 of *LNCS*, pages 142–156. Springer, 2004.

[28] L. Bozzelli and S. La Torre. Decision problems for lower/upper bound parametric timed automata. *Formal Methods in System Design*, 35(2):121–151, 2009.

[29] H. Brandl, M. Weiglhofer, and B. K. Aichernig. Automated conformance verification of hybrid systems. In *Quality Software (QSIC), 10th International Conference on*, pages 3–12. IEEE, 2010.

[30] G. Brat, D. Drusinsky, D. Giannakopoulou, A. Goldberg, K. Havelund, M. Lowry, C. Pasareanu, A. Venet, W. Visser, and R. Washington. Experimental evaluation of verification and validation tools on martian rover software. *Formal Methods in System Design*, 25(2-3):167–198, 2004.

[31] G. Bruns and P. Godefroid. Temporal logic query checking. In *Proceedings of the 16th Annual IEEE Symposium on Logic in Computer Science*, pages 409 – 417. IEEE Computer Society, 2001.

[32] W. Chan. Temporal-logic queries. In *Proceedings of the 12th International Conference on Computer Aided Verification*, volume 1855 of *LNCS*, pages 450–463, London, UK, 2000. Springer.

[33] R. N. Charette. This car runs on code. *IEEE spectrum*, 46(3):3, 2009.

[34] M. Chateauneuf, C. J. Colbourn, and D. L. Kreher. Covering arrays of strength three. *Designs, Codes and Cryptography*, 16(3):235–242, 1999.

[35] M. Chechik and A. Gurfinkel. Tlqsolver: A temporal logic query checker. In *Proceedings of the 15th International Conference on Computer Aided Verification*, volume 2725, pages 210–214. Springer, 2003.

[36] T. Chen, M. Diciolla, M. Z. Kwiatkowska, and A. Mereacre. A simulink hybrid heart model for quantitative verification of cardiac pacemakers. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 131–136. ACM, 2013.

[37] X. Chen, E. Abraham, and S. Sankaranarayanan. Flow*: An analyzer for nonlinear hybrid systems. In *Computer-Aided Verification (CAV)*, volume 8044 of *LNCS*, pages 258–263. Springer-Verlag, 2013.

[38] R. Choroszucha, J. Sun, and K. Butts. Closed-loop model order reduction and mpc for diesel engine airpath control. In *American Control Conference (ACC), 2015*, pages 3279–3284. IEEE, 2015.

[39] R. Choroszucha, J. Sun, and K. Butts. Nonlinear model order reduction for predictive control of the diesel engine airpath. In *American Control Conference (ACC), 2016*, pages 5081–5086. IEEE, 2016.

[40] A. Chutinan and K. R. Butts. Dynamic analysis of hybrid system models for design validation. Technical report, Ford Motor Company, 2002.

[41] A. Chutinan and B. Krogh. Verification of polyhedral invariant hybrid automata using polygonal flow pipe approximations. In *Hybrid Systems: Computation and Control*, volume 1569 of *LNCS*, pages 76–90. Springer, 1999.

[42] D. M. Cohen, S. R. Dalal, M. L. Fredman, and G. C. Patton. The aetg system: An approach to testing based on combinatorial design. *IEEE Transactions on Software Engineering*, 23(7):437–444, 1997.

[43] C. J. Colbourn and J. H. Dinitz. *Handbook of combinatorial designs*. CRC press, 2006.

[44] M. Conrad and I. Fey. Testing automotive control software. In *Automotive Embedded Systems Handbook*. CRC Press, 2008.

[45] J. C. Corbett, M. B. Dwyer, J. Hatcliff, S. Laubach, C. S. Păsăreanu, R. Bby, and H. Zheng. Bandera: Extracting finite-state models from java source code. In *Software Engineering, 2000. Proceedings of the 2000 International Conference on*, pages 439–448. IEEE, 2000.

[46] T. Dang. *Verification and Synthesis of Hybrid Systems*. PhD thesis, INPG, 2000.

[47] K. Deb. *Multi-objective optimization using evolutionary algorithms*, volume 16. John Wiley & Sons, 2001.

[48] Y. Deng, A. Rajhans, and A. A. Julius. Strong: A trajectory-based verification toolbox for hybrid systems. In *Quantitative Evaluation of Systems*, pages 165–168. Springer, 2013.

[49] J. V. Deshmukh, R. Majumdar, and V. S. Prabhu. Quantifying conformance using the skorokhod metric. In *International Conference on Computer Aided Verification*, pages 234–250. Springer, 2015.

[50] A. Deshpande, A. Gollu, and P. Varaiya. Shift: A formalism and a programming language for dynamic networks of hybrid automata. In P. J. Antsaklis, W. Kohn, A. Nerode, and S. Sastry, editors, *Hybrid Systems*, volume 1273 of *LNCS*, pages 113–133. Springer, 1996.

[51] B. Di Giampaolo, S. La Torre, and M. Napoli. Parametric metric interval temporal logic. In A.-H. Dediu, H. Fernau, and C. Martin-Vide, editors, *Language and Automata Theory and Applications*, volume 6031 of *LNCS*, pages 249–260. Springer, 2010.

[52] B. Di Giampaolo, S. La Torre, and M. Napoli. Parametric metric interval temporal logic. In *Language and Automata Theory and Applications*, pages 249–260. Springer, 2010.

[53] A. Dokhanchi, B. Hoxha, and G. Fainekos. On-line monitoring for temporal logic robustness. In *Runtime Verification*, volume 8734 of *LNCS*, pages 231–246. Springer, 2014.

[54] A. Dokhanchi, B. Hoxha, and G. Fainekos. Metric interval temporal logic specification elicitation and debugging. In *Formal Methods and Models for Codesign (MEMOCODE), 2015 ACM/IEEE International Conference on*, pages 70–79. IEEE, 2015.

[55] A. Dokhanchi, B. Hoxha, and G. Fainekos. Formal requirement debugging for testing and verification of cyber-physical systems. *arXiv preprint arXiv:1607.02549*, 2016.

[56] A. Dokhanchi, B. Hoxha, C. E. Tuncali, and G. Fainekos. An efficient algorithm for monitoring practical tptl specifications. In *Formal Methods and Models for System Design (MEMOCODE), 2016 ACM/IEEE International Conference on*, pages 184–193. IEEE, 2016.

[57] A. Dokhanchi, A. Zutshi, R. T. Sriniva, S. Sankaranarayanan, and G. Fainekos. Requirements driven falsification with coverage metrics. In *Proceedings of the 12th International Conference on Embedded Software*, pages 31–40. IEEE Press, 2015.

[58] A. Donze. Breach, a toolbox for verification and parameter synthesis of hybrid systems. In *Computer Aided Verification*, volume 6174 of *LNCS*, pages 167–170. Springer, 2010.

[59] A. Donze and O. Maler. Robust satisfaction of temporal logic over real-valued signals. In *Formal Modelling and Analysis of Timed Systems*, volume 6246 of *LNCS*. Springer, 2010.

[60] V. D'silva, D. Kroening, and G. Weissenbacher. A survey of automated techniques for formal software verification. *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, 27(7):1165–1178, 2008.

[61] P. S. Duggirala, S. Mitra, and M. Viswanathan. Verification of annotated models from executions. In *Proc. of the Eleventh ACM Int. Conf. on Embedded Software*, page 26. IEEE Press, 2013.

[62] P. S. Duggirala, S. Mitra, M. Viswanathan, and M. Potok. C2e2: A verification tool for stateflow models. In *TACAS*, pages 68–82, 2015.

[63] D. L. Dvorak and M. Lyu. Nasa study on flight software complexity. *NASA office of chief engineer*, 2009.

[64] J. Eker, J. Janneck, E. A. Lee, J. Liu, X. Liu, J. Ludvig, S. Sachs, and Y. Xiong. Taming heterogeneity - the ptolemy approach. *Proceedings of the IEEE*, 91(1):127–144, Jan. 2003.

[65] F. Fages and A. Rizk. On temporal logic constraint solving for analyzing numerical data time series. *Theor. Comput. Sci.*, 408(1):55–65, 2008.

[66] G. Fainekos, S. Sankaranarayanan, K. Ueda, and H. Yazarel. Verification of automotive control applications using s-taliro. In *Proceedings of the American Control Conference*, 2012.

[67] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications. In *Formal Approaches to Testing and Runtime Verification*, volume 4262 of *LNCS*, pages 178–192. Springer, 2006.

[68] G. E. Fainekos and G. J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[69] M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn. Refining the in-parameter-order strategy for constructing covering arrays. *Journal of Research of the National Institute of Standards and Technology*, 113(5):287, 2008.

[70] G. Frehse, C. L. Guernic, A. Donz, S. Cotton, R. Ray, O. Lebeltel, R. Ripado, A. Girard, T. Dang, and O. Maler. Spaceex: Scalable verification of hybrid systems. In *Proceedings of the 23d CAV*, 2011.

[71] S. Gao. *Computable analysis, decision procedures, and hybrid automata: a new framework for the formal verification of cyber-physical systems.* PhD thesis, PhD thesis, Carnegie Mellon University, 2012.

[72] S. Gao, S. Kong, and E. M. Clarke. dreal: An smt solver for nonlinear theories over the reals. In *International Conference on Automated Deduction*, pages 208–214. Springer, 2013.

[73] A. Girard. Reachability of uncertain linear systems using zonotopes. In *Hybrid Systems: Computation and Control*, volume 3414 of *LNCS*, pages 291–305, 2005.

[74] A. Gurfinkel, B. Devereux, and M. Chechik. Model exploration with temporal logic query checking. *SIGSOFT Softw. Eng. Notes*, 27(6):139–148, 2002.

[75] A. Hartman. Software and hardware testing using combinatorial covering suites. In *Graph theory, combinatorics and algorithms*, pages 237–266. Springer, 2005.

[76] T. A. Henzinger. The theory of hybrid automata. In *Proceedings of the 11th Annual Symposium on Logic in Computer Science*, pages 278–292. IEEE Computer Society Press, 1996.

[77] M. Hinchey, C. Wang, and M. Josh. Formal methods for system/software engineering: Nasa & army experiences. `https://www.nasa.gov/sites/default/files/585641main_FormalMethodsforSystemSoftwareEngineering.pdf`. Accessed: 2017-07-04.

[78] G. J. Holzmann. The model checker spin. *IEEE Transactions on software engineering*, (5):279–295, 1997.

[79] G. J. Holzmann. The logic of bugs. In *Proc. of the 10th ACM SIGSOFT symp. on Foundations of soft. eng.*, pages 81–87. ACM, 2002.

[80] B. Hoxha, H. Abbas, and G. Fainekos. Benchmarks for temporal logic requirements for automotive systems. In *Workshop on Applied Verification for Continuous and Hybrid Systems*, 2014.

[81] B. Hoxha, H. Abbas, and G. Fainekos. Using s-taliro on industrial size automotive models. *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.

[82] B. Hoxha, H. Bach, H. Abbas, A. Dokhanchi, Y. Kobayashi, and G. Fainekos. Towards formal specification visualization for testing and monitoring of cyber-physical systems. In *Int. Workshop on Design and Implementation of Formal Tools and Systems*. October 2014.

[83] B. Hoxha, A. Dokhanchi, and G. Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *International Journal on Software Tools for Technology Transfer*, pages 1–15, 2017.

[84] B. Hoxha and G. Fainekos. Pareto front exploration for parametric temporal logic specifications of cyber-physical systems. In *Workshop on Monitoring and Testing of Cyber-Physical Systems*, 2016.

[85] M. Huang, H. Nakada, S. Polavarapu, K. Butts, and I. Kolmanovsky. Rate-based model predictive control of diesel engines. In *7th IFAC Symposium on Advances in Automotive Control*, pages 177 – 182, 2013.

[86] M. Huang, H. Nakada, S. Polavarapu, R. Choroszucha, K. Butts, and I. Kolmanovsky. Towards combining nonlinear and predictive control of diesel engines. In *American Control Conference (ACC), 2013*, pages 2846–2853. IEEE, 2013.

[87] D. Isbell, M. Hardin, and J. Underwood. Mars climate orbiter team finds likely cause of loss. *NASA news release*, 1999.

[88] ISO/IEC/IEEE 29148:2011(E). International Standard - Systems and software engineering - Life cycle processes - Requirements engineering. pages 1–94, Dec 2011.

[89] J.-B. Jeannin, K. Ghorbal, Y. Kouskoulas, R. Gardner, A. Schmidt, E. Zawadzki, and A. Platzer. A formally verified hybrid system for the next-generation airborne collision avoidance system. In *TACAS*, volume 9035, pages 21–36, 2015.

[90] Z. Jiang, M. Pajic, and R. Mangharam. Cyber-physical modeling of implantable cardiac medical devices. *Proceedings of the IEEE*, 100(1):122–137, 2012.

[91] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia. Mining requirements from closed-loop control models. In *Proceedings of the 16th international conference on Hybrid systems: computation and control*, pages 43–52. ACM, 2013.

[92] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts. Simulation-guided approaches for verification of automotive powertrain control systems. In *American Control Conference (ACC), 2015*, pages 4086–4095. IEEE, 2015.

[93] J. Kapinski, J. V. Deshmukh, X. Jin, H. Ito, and K. Butts. Simulation-based approaches for verification of embedded control systems: An overview of traditional and advanced modeling, testing, and verification techniques. *IEEE Control Systems*, 36(6):45–64, 2016.

[94] N. Khakpour and M. R. Mousavi. Notions of conformance testing for cyber-physical systems: Overview and roadmap. In *LIPIcs-Leibniz International Proceedings in Informatics*, volume 42. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2015.

[95] E. S. Kim, M. Arcak, and S. A. Seshia. Directed specifications and assumption mining for monotone dynamical systems. In *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, pages 21–30. ACM, 2016.

[96] G. Klein, J. Andronick, K. Elphinstone, G. Heiser, D. Cock, P. Derrin, D. Elkaduwe, K. Engelhardt, R. Kolanski, M. Norrish, et al. sel4: formal verification of an operating-system kernel. *Communications of the ACM*, 53(6):107–115, 2010.

[97] S. A. Klugman, H. H. Panjer, and G. E. Willmot. *Loss models: from data to decisions*, volume 715. John Wiley & Sons, 2012.

[98] S. Kong, S. Gao, W. Chen, and E. M. Clarke. dreach: δ-reachability analysis for hybrid systems. In *TACAS*, volume 15, pages 200–205, 2015.

[99] Z. Kong, A. Jones, A. Medina Ayala, E. Aydin Gol, and C. Belta. Temporal logic inference for classification and prediction from data. In *Proceedings of the 17th international conference on Hybrid systems: computation and control*, pages 273–282. ACM, 2014.

[100] P. Koopman. *Better Embedded System Software*. Drumnadrochit Education LLC, 2010.

[101] Y. Kouskoulas, D. W. Renshaw, A. Platzer, and P. Kazanzides. Certifying the safe design of a virtual fixture control algorithm for a surgical robot. In C. Belta and F. Ivancic, editors, *Hybrid Systems: Computation and Control (part of CPS Week 2013), HSCC'13, Philadelphia, PA, USA, April 8-13, 2013*, pages 263–272. ACM, 2013.

[102] R. Koymans. Specifying real-time properties with metric temporal logic. *Real-Time Systems*, 2(4):255–299, 1990.

[103] H. Kugler, D. Harel, A. Pnueli, Y. Lu, and Y. Bontemps. Temporal logic for scenario-based specifications. In *Tools and Alg. for the Construction and Analysis of Systems*, pages 445–460. Springer, 2005.

[104] D. R. Kuhn and R. Chandramouli. Cost effective uses of formal methods in verification and validation. In *Foundations Verification and Validation Workshop*, 2002.

[105] D. R. Kuhn, D. R. Wallace, and A. M. Gallo. Software fault interactions and implications for software testing. *IEEE transactions on software engineering*, 30(6):418–421, 2004.

[106] J. Legriel, C. Le Guernic, S. Cotton, and O. Maler. Approximating the pareto front of multi-criteria optimization problems. In *TACAS*, pages 69–83. Springer, 2010.

[107] N. G. Leveson and C. S. Turner. An investigation of the therac-25 accidents. *Computer*, 26(7):18–41, 1993.

[108] C. Lignos, V. Raman, C. Finucane, M. Marcus, and H. Kress-Gazit. Provably correct reactive control from natural language. *Autonomous Robots*, 38(1):89–105, 2015.

[109] J.-L. Lions, L. Lbeck, J.-L. Fauquembergue, G. Kahn, W. Kubbat, S. Levedag, L. Mazzini, D. Merle, and C. O'Halloran. Ariane 5, flight 501 failure, report by the inquiry board. Technical report, CNES, July 1996.

[110] O. Maler and D. Nickovic. Monitoring temporal properties of continuous signals. In *Proceedings of FORMATS-FTRTFT*, volume 3253 of *LNCS*, pages 152–166, 2004.

[111] R. Muradore, D. Bresolin, L. Geretti, P. Fiorini, and T. Villa. Robotic surgery. *Robotics & Automation Magazine, IEEE*, 18(3):24–32, 2011.

[112] R. H. Myers, D. C. Montgomery, and C. M. Anderson-Cook. *Response surface methodology: process and product optimization using designed experiments*. Wiley & Sons.

[113] R. Natella, D. Cotroneo, and H. S. Madeira. Assessing dependability with software fault injection: A survey. *ACM Computing Surveys (CSUR)*, 48(3):44, 2016.

[114] T. Nghiem, S. Sankaranarayanan, G. E. Fainekos, F. Ivancic, A. Gupta, and G. J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 211–220. ACM Press, 2010.

[115] W. L. Oberkampf and T. G. Trucano. Verification and validation in computational fluid dynamics. *Progress in Aerospace Sciences*, 38(3):209–272, 2002.

[116] D. of Defense. Dod modeling and simulation (m&s) verification, validation, and accreditation (vv&a), December 2009.

[117] M. Osch. Hybrid input-output conformance and test generation. In K. Havelund, M. Nez, G. Rou, and B. Wolff, editors, *Formal Approaches to Software Testing and Runtime Verification*, volume 4262 of *Lecture Notes in Computer Science*, pages 70–84. Springer Berlin Heidelberg, 2006.

[118] D. G. V. Oss. Computer software in civil aircraft. In *Digital Avionics Systems Conference, 1991. Proceedings., IEEE/AIAA 10th*, pages 324–330. IEEE, 1991.

[119] E. Plaku, L. E. Kavraki, and M. Y. Vardi. Falsification of ltl safety properties in hybrid systems. In *Proc. of the Conf. on Tools and Algorithms for the Construction and Analysis of Systems (TACAS)*, volume 5505 of *LNCS*, pages 368 – 382, 2009.

[120] A. Platzer and E. M. Clarke. Computing differential invariants of hybrid systems as fixedpoints. *Formal Methods in System Design*, 35(1):98–120, 2009.

[121] A. Platzer and J.-D. Quesel. KeYmaera: A hybrid theorem prover for hybrid systems. In A. Armando, P. Baumgartner, and G. Dowek, editors, *International Joint Conference on Automated Reasoning*, volume 5195 of *LNCS*, pages 171–178. Springer, 2008.

[122] J.-D. Quesel, S. Mitsch, S. Loos, N. Aréchiga, and A. Platzer. How to model and prove hybrid systems with keymaera: A tutorial on safety. *International Journal on Software Tools for Technology Transfer*, 18(1):67, 2016.

[123] N. Ramdani, N. Meslem, and Y. Candau. Reachability of uncertain nonlinear systems using a nonlinear hybridization. In *HSCC '08: Proceedings of the 11th international workshop on Hybrid Systems*, pages 415–428, Berlin, Heidelberg, 2008. Springer-Verlag.

[124] A. Rizk, G. Batt, F. Fages, and S. Soliman. On a continuous degree of satisfaction of temporal logic formulae with applications to systems biology. In *International Conference on Computational Methods in Systems Biology*, number 5307 in LNCS, pages 251–268. Springer, 2008.

[125] H. Roehm, T. Heinz, and E. C. Mayer. Stlinspector: Stl validation with guarantees. In *Proceedings of the 29th International Conference on Computer Aided Verification*, 2017.

[126] S-TaLiRo: Temporal Logic Falsification Of Cyber-Physical Systems. `https://sites.google.com/a/asu.edu/s-taliro/s-taliro`, 2013. [Online; accessed April-2014].

[127] A. Saadat. Defect information report. `http://www-odi.nhtsa.dot.gov/acms/cs/jaxrs/download/doc/UCM450071/RCDNN-14V053-0945.pdf`, 2014.

[128] S. Sankaranarayanan and G. Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *ACM International Conference on Hybrid Systems: Computation and Control*, 2012.

[129] S. Sankaranarayanan and G. Fainekos. Simulating insulin infusion pump risks by in-silico modeling of the insulin-glucose regulatory system. In *International Conference on Computational Methods in Systems Biology*, 2012. [To Appear].

[130] S. Sankaranarayanan, H. Homaei, and C. Lewis. Model-based dependability analysis of programmable drug infusion pumps. In *Formal modeling and analysis of timed systems*, pages 317–334. Springer, 2011.

[131] C. Scott. Industry-nominated technology brakethroughs of nsf industry/university cooperative research centers. *Washington DC: National Science Foundation*, 2012. Online at: `http://faculty.washington.edu/scottcs/NSF/2012/NSF_Compendium_2012-WEB.pdf`.

[132] C. Scott. Industry-nominated technology brakethroughs of nsf industry/university cooperative research centers. *Washington DC: National Science Foundation*, 2014. Online at: `http://faculty.washington.edu/scottcs/NSF/2014/NSF_Compendium_2014.pdf`.

[133] H. J. Seltman. *Experimental design and analysis*. Pittsburgh: Carnegie Mellon University, 2012.

[134] B. I. Silva and B. H. Krogh. Formal verification of hybrid systems using CheckMate: a case study. In *Proceedings of the American Control Conference*, volume 3, pages 1679 – 1683, June 2000.

[135] Simuquest. Enginuity. `http://www.simuquest.com/products/enginuity`. Accessed: 2013-10-14.

[136] A. Singh, C. Ramakrishnan, and S. A. Smolka. Query-based model checking of ad hoc network protocols. In *Proceedings of Concurrency Theory*, pages 603–619. Springer, 2009.

[137] M. H. Smith, G. J. Holzmann, and K. Etessami. Events and constraints: A graphical editor for capturing logic requirements of programs. In *Requirements Engineering, 2001. Proceedings. Fifth IEEE International Symposium on*, pages 14–22. IEEE, 2001.

[138] S. Srinivas, R. Kermani, K. Kim, Y. Kobayashi, and G. Fainekos. A graphical language for ltl motion and mission planning. In *Robotics and Biomimetics (ROBIO), 2013 IEEE International Conference on*, pages 704–709. IEEE, 2013.

[139] L. Tan, J. Kim, O. Sokolsky, and I. Lee. Model-based testing and monitoring for hybrid embedded systems. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration*, pages 487–492, 2004.

[140] T. Thüm, S. Apel, C. Kästner, I. Schaefer, and G. Saake. A classification and survey of analysis strategies for software product lines. *ACM Computing Surveys (CSUR)*, 47(1):6, 2014.

[141] J. Tretmans. Testing concurrent systems: A formal approach. In *CONCUR 1999 Concurrency Theory*, pages 46–65. Springer, 1999.

[142] S. Tripakis and T. Dang. *Model-Based Design for Embedded Systems*, chapter Modeling, Verification and Testing using Timed and Hybrid Automata, pages 383–436. CRC Press, 2009.

[143] R. Vinter, M. Loomes, and D. Kornbrot. Applying software metrics to formal specifications: A cognitive approach. In *Software Metrics Symposium, 1998. Metrics 1998. Proceedings. Fifth International*, pages 216–223. IEEE, 1998.

[144] A. Wasylkowski and A. Zeller. Mining temporal specifications from object usage. In *24th IEEE/ACM International Conference on Automated Software Engineering*, 2009.

[145] M. Woehrle, K. Lampka, and L. Thiele. Conformance testing for cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 11(4):84:1–84:23, Jan. 2013.

[146] T. Wongpiromsarn, S. Mitra, A. Lamperski, and R. M. Murray. Verification of periodically controlled hybrid systems: Application to an autonomous vehicle. *ACM Trans. Embed. Comput. Syst.*, 11(S2):53:1–53:24, Aug. 2012.

[147] H. Yang, B. Hoxha, and G. Fainekos. Querying parametric temporal logic properties on embedded systems. In *Int. Conference on Testing Software and Systems*, 2012.

[148] B. Yordanov, J. Tmov, I. ern, J. Barnat, and C. Belta. Formal analysis of piecewise affine systems through formula-guided refinement. *Automatica*, 49(1):261 – 266, 2013.

[149] P. Zhang, B. Li, and L. Grunske. Timed property sequence chart. *Journal of Systems and Software*, 83(3):371–390, 2010.

[150] P. Zuliani, A. Platzer, and E. M. Clarke. Bayesian statistical model checking with application to simulink/stateflow verification. In *Proceedings of the 13th ACM International Conference on Hybrid Systems: Computation and Control*, pages 243–252, 2010.

[151] A. Zutshi, J. V. Deshmukh, S. Sankaranarayanan, and J. Kapinski. Multiple shooting, cegar-based falsification for hybrid systems. In *Proceedings of the 14th International Conference on Embedded Software*, page 5. ACM, 2014.

[152] A. Zutshi, S. Sankaranarayanan, J. V. Deshmukh, and J. Kapinski. A trajectory splicing approach to concretizing counterexamples for hybrid systems. In *IEEE Conference on Decision and Control*, 2013.

APPENDIX A

PROOFS

A.1 Lemma 4.4.1

Proof A.1.1 (sketch) *Without loss of generality, we will prove only case (i) of Lemma 4.4.1.1. Case (ii) is symmetric with respect to the temporal operator and Lemma 4.4.1.2 is symmetric in terms of monotonicity. The proof is by induction on the structure of the formula and it is similar to the proofs that appeared in [68].*

For completeness of the presentation, we consider the case $[\![\phi_1 \mathcal{U}_{\langle \alpha,\theta \rangle} \phi_2]\!](\mu, i)$*, where* $\langle \in \{[, (\}$ *and* $\rangle \in \{],)\}$*. The other cases are either similar or they are based on the monotonicity of the* max *and* min *operators. We remark that the* max *and* min *operators preserve monotonicity. Let* $\theta_1 \leq \theta_2$*, then we want to show that:*

$$[\![\phi_1 \mathcal{U}_{\langle \alpha,\theta_1 \rangle} \phi_2]\!](\mu, i) \leq [\![\phi_1 \mathcal{U}_{\langle \alpha,\theta_2 \rangle} \phi_2]\!](\mu, i) \tag{A.1}$$

To show that (A.1) holds, we utilize the robust semantics for MTL given in Definition 4.3.3 and observe that:

$$[\![\phi_1 \mathcal{U}_{\langle \alpha,\theta_2 \rangle} \phi_2]\!](\mu, i) = \max_{j \in \tau^{-1}(\tau(i)+\langle \alpha,\theta_2 \rangle)} \big(\min([\![\phi_2]\!](\mu, j), \min_{i \leq k < j} [\![\phi_1]\!](\mu, k))\big) =$$

$$\max \Bigg(\max_{j \in \tau^{-1}(\tau(i)+\langle \alpha,\theta_1 \rangle)} \big(\min([\![\phi_2]\!](\mu, j), \min_{i \leq k < j} [\![\phi_1]\!](\mu, k))\big),$$

$$\max_{j \in \tau^{-1}(\tau(i)+\overline{\langle}\theta_1,\theta_2\rangle)} \big(\min([\![\phi_2]\!](\mu, j), \min_{i \leq k < j} [\![\phi_1]\!](\mu, k))\big) \Bigg) =$$

$$\max \Big([\![\phi_1 \mathcal{U}_{\langle \alpha,\theta_1 \rangle} \phi_2]\!](\mu, i), [\![\phi_1 \mathcal{U}_{\overline{\langle}\theta_1,\theta_2\rangle} \phi_2]\!](\mu, i) \Big) \geq [\![\phi_1 \mathcal{U}_{\langle \alpha,\theta_1 \rangle} \phi_2]\!](\mu, i)$$

where $\overline{\langle} \in \{[, (\}$ *such that* $\langle \alpha, \theta_1 \rangle \cap \overline{\langle}\theta_1, \theta_2\rangle = \emptyset$ *and* $\langle \alpha, \theta_1 \rangle \cup \overline{\langle}\theta_1, \theta_2\rangle = \langle \alpha, \theta_2 \rangle$*.*

A.2 Lemma 4.4.2

Proof A.2.1 (sketch) *The proof is by induction on the structure of the formula and it is similar to the proofs that appeared in [68]. For completeness of the presentation, we consider the base case* $[\![p[\theta]]\!](\mu, i)$*. Let* $\theta_1 \leq \theta_2$*, then* $\mathcal{O}(p[\theta_1]) \subseteq \mathcal{O}(p[\theta_2])$*. We will only present the case for which* $\mathbf{y}(i) \notin \mathcal{O}(p[\theta_2])$*. We have:*

$$\begin{gathered}
\mathcal{O}(p_j[\theta_1]) \subseteq \mathcal{O}(p_j[\theta_2]) \implies \\
\mathbf{dist}_d(\mathbf{y}(i), \mathcal{O}(p_j[\theta_1])) \geq \mathbf{dist}_d(\mathbf{y}(i), \mathcal{O}(p_j[\theta_2])) \implies \\
\mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p_j[\theta_1])) \leq \mathbf{Dist}_d(\mathbf{y}(i), \mathcal{O}(p_j[\theta_2])) \implies \\
[\![p_j[\theta_1]]\!](\mu, i) \leq [\![p_j[\theta_2]]\!](\mu, i)
\end{gathered}$$

A.3 Theorem 4.4.1

Proof A.3.1 (sketch) *The proof is by induction on the structure of the formula. The base case is given by Lemmas 4.4.1 and 4.4.2.*

Consider the first case where $\phi[\vec{\theta}] = \phi_1[\vec{\theta}] \mathcal{U}_{\mathcal{I}[\theta_s]} \phi_2[\vec{\theta}]$*. Let* $\vec{\theta}, \vec{\theta'} \in \overline{\mathbb{R}}^n_{\geq 0}$*, where* $\vec{\theta} \preceq \vec{\theta'}$*. Let* $i, j, k \in N$*. Then* $\mathcal{I}[\theta_s] \subseteq \mathcal{I}[\theta'_s]$ *and, for all* i*, by the induction hypothesis we have*

$$\begin{gathered}
[\![\phi_1[\vec{\theta}]]\!](\mu, i)) \leq [\![\phi_1[\vec{\theta'}]]\!](\mu, i)) \\
[\![\phi_2[\vec{\theta}]]\!](\mu, i)) \leq [\![\phi_2[\vec{\theta'}]]\!](\mu, i))
\end{gathered}$$

Therefore, for all $i \leq k < j$

$$\min_{i \leq k < j}([\![\phi_1[\vec{\theta}]]\!](\mu, k)) \leq \min_{i \leq k < j}([\![\phi_1[\vec{\theta'}]]\!](\mu, k)) \tag{A.2}$$

$$\min([\![\phi_2[\vec{\theta}]]\!](\mu, j), \min_{i \leq k < j}([\![\phi_1[\vec{\theta}]]\!](\mu, k))) \leq$$

$$\min([\![\phi_2[\vec{\theta'}]]\!](\mu, j), \min_{i \leq k < j}([\![\phi_1[\vec{\theta'}]]\!](\mu, k))) \tag{A.3}$$

Then by (A.2) and (A.3) we have

$$[\![\phi[\vec{\theta}]]\!](\mu, i) = [\![\phi_1[\vec{\theta}]\, \mathcal{U}_{\mathcal{I}[\theta_s]} \phi_2[\vec{\theta}]]\!](\mu, i) =$$

$$\sup_{j \in \tau^{-1}(\tau(i) + \mathcal{I}[\theta_s])} \left(\min([\![\phi_2[\vec{\theta}]]\!](\mu, j), \min_{i \leq k < j} [\![\phi_1[\vec{\theta}]]\!](\mu, k))\right)$$

$$\leq$$

$$\sup_{j \in \tau^{-1}(\tau(i) + \mathcal{I}[\theta'_s])} \left(\min([\![\phi_2[\vec{\theta'}]]\!](\mu, j), \min_{i \leq k < j} [\![\phi_1[\vec{\theta'}]]\!](\mu, k))\right) =$$

$$[\![\phi_1[\vec{\theta'}]\, \mathcal{U}_{\mathcal{I}[\theta'_s]} \phi_2[\vec{\theta'}]]\!](\mu, i) = [\![\phi[\vec{\theta'}]]\!](\mu, i)$$

Therefore,

$$[\![\phi[\vec{\theta}]]\!](\mu, i) \leq [\![\phi[\vec{\theta'}]]\!](\mu, i) \tag{A.4}$$

A.4 Proposition 4.5.1

Proof A.4.1 *If* $[\![\phi[\vec{\theta^*}]]\!](\mu^*) \leq 0$*, then* $[\![\phi[\vec{\theta^*}]]\!](\Sigma) \leq 0$*. Since* $[\![\phi[\vec{\theta}]]\!](\Sigma)$ *is non-increasing with respect to* $\vec{\theta}$*, then for all* $\vec{\theta} \succeq \vec{\theta^*}$*, we also have* $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$*.*

A.5 Proposition 4.5.2

Proof A.5.1 *The interesting case to prove here is when we have* $\vec{\theta}$ *such that* $[\![\phi[\vec{\theta}]]\!](\Sigma) \geq 0$ *and we have* $\vec{\theta'}$ *such that* $[\![\phi[\vec{\theta'}]]\!](\Sigma) < 0$*. See Fig. 4.6 (Left) for an illustration of the arrangement of parameter valuations for a two parameter specification. In this case,*

$$\gamma = \|\vec{\overline{\theta}}\| \geq \|\vec{\theta'}\| \geq \|\vec{\theta}\|$$
$$\textit{and}$$
$$[\![\phi[\vec{\theta}]]\!](\Sigma) \geq 0 \implies$$
$$\|\vec{\theta}\| + \gamma + [\![\phi[\vec{\theta}]]\!](\Sigma) \geq \|\vec{\theta'}\|$$

Therefore, if the problem in Eq. (4.3) is feasible, then the optimum of Eq. (4.3) and Eq. (4.4) is the same.

A.6 Proposition 4.5.3

Proof A.6.1 *If* $[\![\phi[\vec{\theta^*}]]\!](\mu^*) \leq 0$*, then* $[\![\phi[\vec{\theta^*}]]\!](\Sigma) \leq 0$*. Since* $[\![\phi[\vec{\theta}]]\!](\Sigma)$ *is non-decreasing with respect to* $\vec{\theta}$*, then for all* $\vec{\theta} \preceq \vec{\theta^*}$*, we also have* $[\![\phi[\vec{\theta}]]\!](\Sigma) \leq 0$*.*

A.7 Proposition 4.5.4

Proof A.7.1 *The interesting case to prove here is when we have* $\vec{\theta}$ *such that* $[\![\phi[\vec{\theta}]]\!](\Sigma) < 0$ *and we have* $\vec{\theta'}$ *such that* $[\![\phi[\vec{\theta'}]]\!](\Sigma) \geq 0$*. See Fig. 4.6 (Right) for an illustration of the arrangement of parameter valuations for a two parameter specification. In this case*

$$\gamma = -\|\vec{\bar{\theta}}\|,\ [\![\phi[\vec{\theta'}]]\!](\Sigma) \geq 0 \text{ and}$$
$$\|\vec{\bar{\theta}}\| \geq \|\vec{\theta'}\| \geq \|\vec{\theta}\| \implies$$
$$\|\vec{\theta}\| \geq \|\vec{\theta'}\| + \gamma - [\![\phi[\vec{\theta'}]]\!](\Sigma)$$

Therefore, if the problem in Eq. (4.3) is feasible, then the optimum of equations (4.3) and (4.4) is the same.