THESIS FOR THE DEGREE OF DOCTOR OF PHILOSOPHY

On Falsification of Large-Scale Cyber-Physical Systems

Johan Lidén Eddeland

Department of Electrical Engineering
Chalmers University of Technology
Gothenburg, Sweden, 2022

On Falsification of Large-Scale Cyber-Physical Systems

JOHAN LIDÉN EDDELAND
ISBN 978-91-7905-750-3

Doktorsavhandlingar vid Chalmers tekniska högskola
Ny serie nr 5216
ISSN 0346-718X

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Gothenburg, Sweden
Phone: +46 (0)31 772 1000

Printed by Chalmers Reproservice
Gothenburg, Sweden, November 2022

To Hanna, Vidar, Sixten, and Signe

Abstract

In the development of modern Cyber-Physical Systems, Model-Based Testing of the closed-loop system is an approach for finding potential faults and increasing quality of developed products. Testing is done on many different abstraction levels, and for large-scale industrial systems, there are several challenges. Executing tests on the systems can be time-consuming and large numbers of complex specifications need to be thoroughly tested, while many of the popular academic benchmarks do not necessarily reflect on this complexity.

This thesis proposes new methods for analyzing and generating test cases as a means for being more certain that proper testing has been performed on the system under test. For analysis, the proposed approach can automatically find out how much of the physical parts of the system that the test suite has executed.

For test case generation, an approach to find errors is *optimization-based falsification*. This thesis attempts to close the gap between academia and industry by applying falsification techniques to real-world models from Volvo Car Corporation and adapting the falsification procedure where it has shortcomings for certain classes of systems. Specifically, the main contributions of this thesis are (i) a method for automatically transforming a signal-based specification into a formal specification allowing an optimization-based falsification approach, (ii) a new collection of specifications inspired by large-scale specifications from industry, (iii) an algorithm to perform optimization-based falsification for such a large set of specifications, and (iv) a new type of coverage criterion for Cyber-Physical Systems that can help to assess when testing can be concluded.

The proposed methods have been evaluated for both academic benchmark examples and real-world industrial models. One of the main conclusions is that the proposed additions and changes to the analysis and generation of tests can be useful, given that one has enough information about the system under test. The methods presented in this thesis have been applied to real-world models in a way that allows for higher-quality products by finding more faults in early phases of development.

Keywords: Testing, Simulation-Based Verification, Formal Requirements, Falsification, Optimization, Test Coverage, Cyber-Physical Systems.

List of Publications

This thesis is based on the following publications. Note that the papers are not presented in chronological order; instead, the order is chosen so as to make the research as easy as possible to understand for the reader.

[A] **Johan Lidén Eddeland**, Koen Claessen, Nicholas Smallbone, Zahra Ramezani, Sajed Miremadi, Knut Åkesson, "Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans". IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 5247-5260, 2020.

[B] **Johan Lidén Eddeland**, Knut Åkesson, "Evaluating Optimization Solvers and Robust Semantics for Simulation-Based Falsification". ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), July, 2020 - Online.

[C] **Johan Lidén Eddeland**, Alexandre Donzé, Sajed Miremadi, Knut Åkesson, "Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems". ARCH20. 7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), July 12, 2020 - Online.

[D] **Johan Lidén Eddeland**, Alexandre Donzé, Knut Åkesson, "Multi-Requirement Testing Using Focused Falsification". Submitted for possible journal publication. This is an extended version of the paper *Multi-Requirement Testing Using Focused Falsification* accepted to and presented at *HSCC 2022: ACM International Conference on Hybrid Systems: Computation and Control.*

[E] **Johan Eddeland**, Javier Gil Cepeda, Rick Fransen, Sajed Miremadi, Martin Fabian, Knut Åkesson, "Automated Mode Coverage Analysis for Cyber-Physical Systems Using Hybrid Automata". The 20th World Congress of the International Federation of Automatic Control, July, 2017 - Toulouse, France.

Specification of my contribution to the included publications:

[A] I implemented the specification transformer and implemented it on the Volvo models for discussion in Section 5.5 in the paper. I implemented the additive semantics in MATLAB and chose the benchmarks models and specifications that were included in the evaluation. I also ran all the experiments in the paper. The co-authors aided in defining the theoretical framework of specification transformation, e.g., using SignalTables and FormulaTables to represent the specifications. I wrote the manuscript with valuable input from all co-authors.

[B] I designed the experiments with the help of the co-author. I performed all experiments and summarized the results in the table and cactus plot presented in the paper. I wrote the manuscript with input from discussions with the co-author.

[C] I created all requirements with inspiration from requirements at Volvo Cars, and I tuned the parameters as described. I ran all the simulations needed to produce the statistics presented in the paper, and I wrote the manuscript with helpful input from all co-authors.

[D] I designed and implemented the MRF and Corners-UR algorithms together with the co-authors. I ran the experiments and generated the data presented in the tables. The theoretical background and discussion was a results of many meetings and discussion together with the co-authors. I wrote the manuscript in collaboration with all of the co-authors.

[E] I defined the main coverage criterion and co-implemented the algorithm for calculating the mode coverage for existing simulations of the dog clutch. I also wrote the manuscript. The co-authors co-implemented the algorithm for calculating mode coverage, implemented visualization of the dog clutch (such as in Figure 6 of the paper), and aided in formulation of the manuscript.

Other publications by the author, not included in this thesis, are:

[F] **Johan Eddeland**, Sajed Miremadi, Martin Fabian, Knut Åkesson, "Objective Functions for Falsification of Signal Temporal Logic Properties in Cyber-Physical Systems". *13th Conference on Automation Science and Engineering (CASE)* Xi'an, China, Aug. 2017.

[G] Koen Claessen, Nicholas Smallbone, **Johan Eddeland**, Zahra Ramezani, Knut Åkesson, "Using Valued Booleans to Find Simpler Counterexamples in Random Testing of Cyber-Physical Systems". *14th IFAC Workshop on Discrete Event Systems (WODES)* Sorrento Coast, Italy, May 2018.

[H] Koen Claessen, Nicholas Smallbone, **Johan Lidén Eddeland**, Zahra Ramezani, Knut Åkesson, Sajed Miremadi, "Applying Valued Booleans in Testing of Cyber-Physical Systems". *2018 IEEE Workshop on Monitoring and Testing of Cyber-Physical Systems (MT-CPS)* Porto, Portugal, Apr. 2018.

[I] Zahra Ramezani, **Johan Lidén Eddeland**, Koen Claessen, Martin Fabian, Knut Åkesson, "Multiple Objective Functions for Falsification of Cyber-Physical Systems". *15th IFAC Workshop on Discrete Event Systems (WODES)* Online, Nov. 2020.

Acknowledgments

So many people have helped me during my years as a Ph.D. student, both professionally and personally, that it is probably impossible to mention everyone. I will do my best, and I am humbled by being given the opportunity to perform research in this project on such an interesting subject (which I did not know anything about beforehand).

First of all, I want to thank my academic supervisor Knut Åkesson and my industrial supervisor Sajed Miremadi for your continuous support and encouragement during the entire project. Even though you have busy schedules, you always had time for discussing interesting problems with me, and you guided me through both the academic challenges and the more hands-on problems which I could not solve myself. Thanks to both of you, I will always have fond memories of my daily work as a Ph.D. student!

I would also like to thank my co-supervisor Martin Fabian. Whenever I needed help with the final details of writing papers, you always stood up to the challenge of meticulously finding every single word with any kind of mistake. Maybe not the help I deserved, but the help I needed, both back then and now when I look back on what I learned about writing and the scientific process.

I am very grateful to Alexandre Donzé for all the help and support during the recent years. You became a mentor to me after all my initial inquiries about Breach and STL, and I am very lucky to have had you with all your expertise become such as big part of the research project.

I would also like to express my gratitude towards Koen Claessen and Nicholas Smallbone for your engaging discussions and contributions to the work I have done. I will never understand how you can get so much done in such a little time, but I am thankful for it!

I also want to thank all my managers at Volvo, especially my first manager Isak Öberg who provided vital support at the start of my time as a Ph.D. student when I wasn't entirely sure how to handle the difficulty of it all. Many thanks also to Sajed, Marcus, Martin, Aboozar, and Carol for helping me out with support and carrying the burden of administration so that I didn't have to.

I also want to acknowledge the support from my colleagues at Volvo. The team spirit of Team MM with Andreas, Ulf, Ulrika, and Niklas always cheered me up when I needed it, and Eduard and Carl are long-term team members I've always been inspired by. Fredrik helped me with hardware, and Ellen, Nithin, and Tiberiu helped me with software whenever I needed their input. In addition to this, I got help and useful comments from many other colleagues who I don't have the space to mention here.

I am grateful for the support from all my Chalmers colleagues in the Automation group as well, especially from Zahra for the tight cooperation in producing interesting research in the same research field. Interesting meetings and discussions with Constantin and Yuvaraj also helped with gain a wider knowledge of related fields of research.

I also want to thank my favorite teacher of all time, Tenzing, for the inspiration during and after physics classes in grades 7–9. Thanks to you I understood more about both what I wanted to do myself, but also about how to inspire and educate others.

Finally, I want to express my sincere gratitude toward my closest family. My parents always supported me in everything I did since I was young. My brothers felt more like a burden than anything else for the first 15 years of my life, but looking back I can see how they helped me become who I am today. I would never have been able to finish this Ph.D. without the never-ending love and support from my wife, Hanna. No matter how bad of a day I can have at work, coming home to you, Vidar, Sixten, and Signe always makes me appreciate our wonderful life together. I love you and our fantastic family!

This work has been performed with support from the Swedish Governmental Agency for Innovation Systems (VINNOVA) project TESTRON 2015-04893 and from the Swedish Research Council (VR) project SyTeC 2016-06204. This support is gratefully acknowledged.

Acronyms

CI:	Continuous Integration
CPS:	Cyber-Physical System
HIL:	Hardware-in-the-Loop
LTL:	Linear Temporal Logic
MIL:	Model-in-the-Loop
MBT:	Model-Based Testing
MC/DC:	Modified Condition/Decision Coverage
MTL:	Metric Temporal Logic
SIL:	Software-in-the-Loop
SMT:	Satisfiability Modulo Theories
STL:	Signal Temporal Logic
SUT:	System Under Test

Contents

Part I

Overview

CHAPTER 1

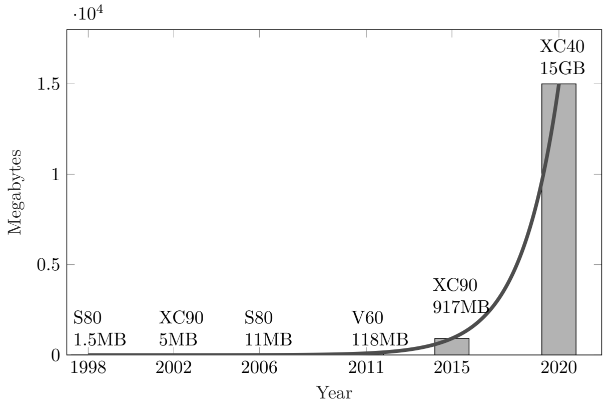
Introduction

When we as humans create something, it is usually a process of trial and error. "Rome was not built in a day" is a proverbial saying that can suggest both that great things take a long time to finish, but perhaps also that there will be mistakes made along the way. Developing software is no exception, which means that we need to systematize how to catch faults so that they do not exist in the final product.

This thesis tackles the problem of testing software. Specifically, techniques to increase the level of automation in the testing of Cyber-Physical Systems (CPSs) [1], to find bugs or faults without creating much additional work effort for the engineers designing the system. A CPS is, as the name suggests, a system that consists of both *cyber* and *physical* components – meaning that there is some software interacting with actual physical components. Some examples of CPSs are cars, industrial robots, and advanced medical devices. A CPS is considered a *hybrid* system in the sense that it contains both discrete and continuous elements.

To efficiently develop modern CPSs, a common design paradigm is to use *models*. A model in this case is a mathematical description of the inner workings of the CPS, and the model can be defined for different levels of abstraction. For example, a simple model of a car could define how a point mass accelerates forward as a function of how hard the driver pushes the gas pedal. However, a more detailed model could take into account the friction between the car tires and the road, the weight of the passengers in the car, the weather and air resistance around the car, and many other characteristics that determine how the car moves. Performing testing on models is naturally called Model-Based Testing (MBT) [2], and there are many methods to apply MBT to CPSs [3]–[6].

MBT typically involves much automatic testing and is becoming more and more useful as the software size in cars is increasing rapidly, which means that relying too much on manual testing does not scale well enough time-wise to be viable for the future. For example, Figure 1.1 shows the historical downloadable software size in certain Volvo cars. This motivates the introduction of more automated testing as a complement to manual testing that is usually already in place in the development of modern CPSs.



,

Figure 1.1: A bar chart of the downloadable software size in certain Volvo car models during the years 1998 - 2020.

To test the components that are being developed, one must define what needs to be tested. This is done by defining a *specification*[1], i.e., the desired behavior of the system. A specification can be written in natural language,

[1]Note that in this thesis, the terms *specification* and *requirement* are used interchangeably. In the appended papers, it is also the case that both terms indicate the same concept.

e.g. "*The car's velocity should be lower than 150 km/h*", or in some more mathematical way, e.g. "$v < 150$". The output of the system given a certain input, a *test case*, can then be evaluated against the specification to see if the test case has passed or failed.

An important and difficult question is how to create new test cases for the System Under Test (SUT). Generating a test case for a model of a CPS typically means coming up with inputs to a simulation of the closed-loop system including both software and the simulated physical components of the system. In the end, the tests are created to find faults if they exist in the system. Testing can never prove the absence of faults in the system, but there are different approaches to guide the generation to this end. One approach is to consider *code coverage* of the software being tested.

As an example, consider the pseudo-code below.

```
if $a$ then
    $x = x + 1$
else
    $x = x - 1$
end if
```

If one wants full statement coverage of the given code, all statements need to be executed, meaning that a needs to take on both values true and false. There are many notions of coverage [7] other than statement coverage, for example, branch coverage and decision coverage, but the main idea of a coverage criterion is to give a number indicating how much of the code that has been tested.

Another approach to generating new test cases is via *falsification* of CPSs. The goal of falsification is to find a counterexample where a given specification does not hold. It is common for the specification to be expressed in a formal language like *Metric Temporal Logic* [8] or *Signal Temporal Logic* [9], [10] (STL), where one can measure how "far" a specification is from being falsified by a test case. Whenever falsification is mentioned in this thesis, it refers to the method of finding counterexamples to temporal logic specifications of CPSs [11].

In short, this thesis distinguishes between *testing*, *falsification*, and *optimization-based falsification* as different levels of ensuring software quality where more exhaustive methods like model checking are not applicable. The relations between the different levels are shown in Figure 1.2.

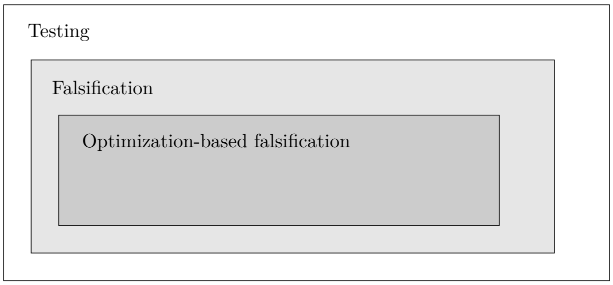


Figure 1.2: An overview of how testing, falsification and optimization-based falsification are related. Optimization-based falsification is a subset of falsification, which itself is a subset of the broader area of testing. This thesis focuses mainly on optimization-based falsification.

No matter which test method is considered, it is clear that testing attracts many practitioners from both academia as well as industry. It is however also clear, as in most research areas, that methods developed in academic contexts are not always found in industry. In other words, there is a discrepancy in methods developed by researchers and methods used in industry. This thesis attempts to diminish this discrepancy by adapting academic methods to be suitable for industrial models as well as academic ones.

1.1 Testing in industry

Testing in an industrial context often becomes difficult because of the sheer scale of software development. When several hundred engineers work together to develop (a part of) a CPS, for example, a component in a car, there are typically different levels of testing performed. It is also common to introduce different automatic methods for faster and more reliable software development. One of these methods is *Continuous Integration* (CI) [12], which is detailed later in this section.

When testing in industry, the SUT can have different characteristics as well. For some systems, we can have access to the source code, while for some systems, the only thing available for tests are pre-compiled binary files. In the latter case, there is no choice but to consider the system a black box, where we can only access input and output values. This also means that in general, to be able to apply new testing methods to industrial systems, the methods need to be able to handle systems that are completely or partially black-box.

Levels of testing

As part of a Model-Based Design approach, the testing levels can include, but are not limited to:

- **Model-in-the-Loop (MIL):** The software component(s) to be tested are modeled and simulated (meaning that no explicit code is written, rather the software components are created using a modeling language, for example, Modelica [13] or Simulink [14]). The plant, i.e., the physical part of the system that the software interacts with, is also simulated.
- **Software-in-the-Loop (SIL):** The modeled software (or *controller*) is code-generated, and then this generated code is tested against a simulated plant.
- **Hardware-in-the-Loop (HIL):** Some component(s) of the actual hardware are used in the testing, while some parts of the plant are still simulated.

The final stage of testing is to physically test the entire system, for example by driving the finished car and trying to evaluate whether all the requirements on the system are fulfilled. The earlier testing phases presented here are the ones that are cheapest and easiest to scale. For MIL and SIL testing, since everything is simulated, the only limiting factor in creating and evaluating new test cases is computational power. For HIL testing, since there is an actual hardware component interacting with the software, the testing needs to be performed in real-time, typically also with additional safety measures since parts could potentially catch fire or be part of similar hazards.

In this thesis, the main focus is on testing environments where the whole system is simulated, e.g. MIL and SIL testing. It should still be noted that all different testing environments are vital for complete testing of the CPS, as MIL and SIL testing for example cannot capture any hardware problems. Similarly, for a car, certain aspects can only be tested by actually driving the car and not in HIL testing.

There is also another aspect of testing in the software development process. When a software component is created, typically the software developer will create *unit tests* to assert that the component works as expected by itself. When several software components are created, the next step is for them to be connected as part of the functionality of the system. Now testing needs to be performed to validate that the interface and interaction between the components work as expected – this is called *integration testing*. When all different parts of the final system are connected, the final testing stage is called *system testing*. Figure 1.3 shows how MIL, HIL, and SIL testing can be related to unit, integration, and system testing in an interpretation of the V model of software development [15].

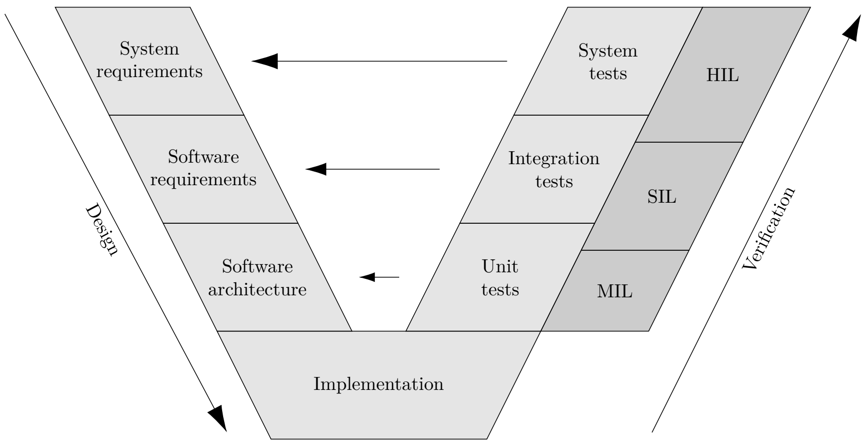


Figure 1.3: An illustration of how MIL, SIL, and HIL testing can be related to unit, integration, and system testing in the V model of software development. Even though each of the testing levels comes sequentially in the testing process, there is not a 1:1 correspondence between (for example) MIL/Unit, SIL/Integration, or HIL/System.

Continuous integration

A common way to incorporate testing in industry is to use Continuous Integration (CI) [12]. CI is the practice of automatically integrating changes of developed code often, to frequently find smaller faults rather than having to fix large faults with many potentially complex causes at larger time intervals. An overview of a typical CI workflow is shown in Figure 1.4.

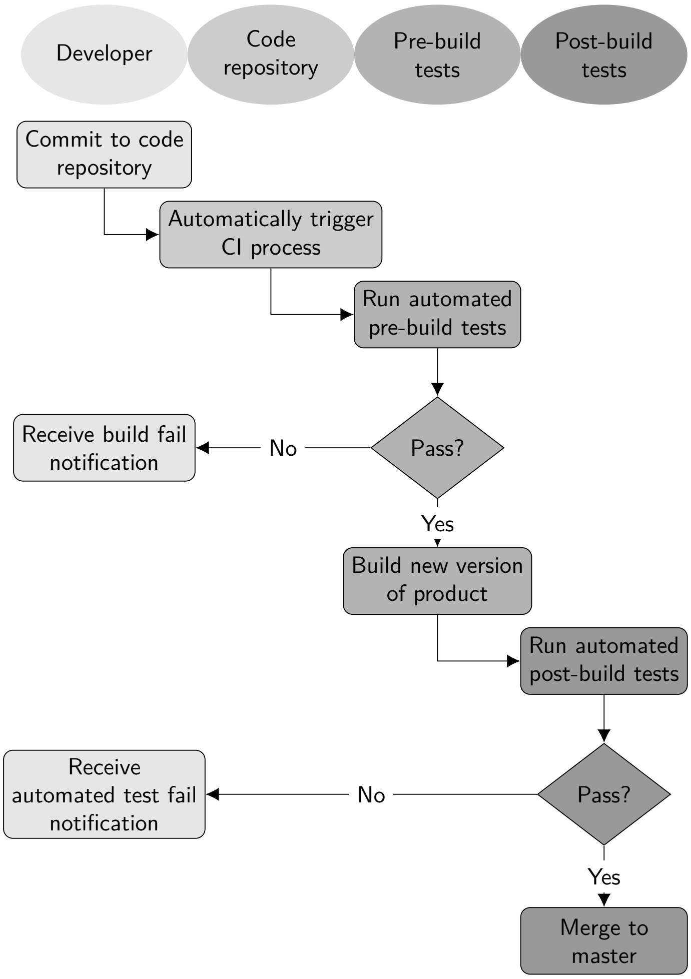

Developer
Code repository
Pre-build tests
Post-build tests
Commit to code repository
Automatically trigger CI process
Run automated pre-build tests
Pass?
No
Receive build fail notification
Yes
Build new version of product
Run automated post-build tests
Pass?
No
Receive automated test fail notification
Yes
Merge to master

Figure 1.4: A flowchart including typical elements of continuous integration (CI). When a developer pushes their code, the code needs to be built and pass both unit tests and other automated tests before being pushed to the master branch. If the code does not build, or if it fails any tests, the developer will be notified and needs to change the code before trying to push again. In the context of this figure, pre-build tests could be unit tests, while post-build tests could be integration tests.

A sketch of the desired effect of CI, in terms of time spent finding and correcting faults in the developed software, can be seen in Figure 1.5. It is clear that in the ideal case, it is easier to find faults when there are few of them and there are not many different versions of the software to check against. However, implementing CI also requires the writing of automated tests and a general change in the way of working (compared to not using CI). As this thesis is focused on automated testing, it can be seen as part of making a CI chain work.

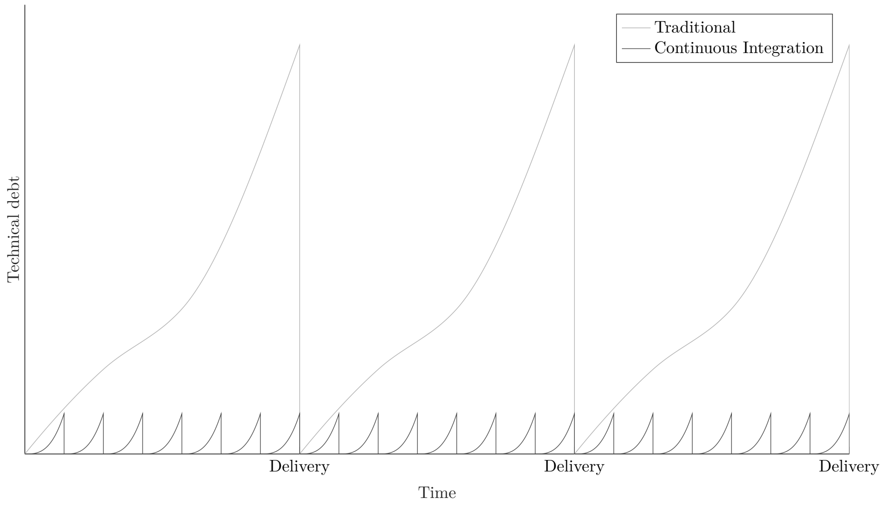


Figure 1.5: A sketch of intended technical debt over time using traditional development methods versus continuous integration. Committing the developed code with a high frequency typically also means that the faults are easier to find and less time-consuming to fix.

1.2 Research questions

The goal of the research performed leading up to this thesis can be summarized in four research questions, all connected to Model-Based Testing of Cyber-Physical Systems. The research questions did not originally all exist as they are presented here, they were rather developed and matured during the progress of the performed research. As such, the research questions are not in any sort of "chronological" order; instead, they are presented in a way that arguably makes them easiest to understand in relation to each other.

Research Question 1. *How can specifications expressed in industrial modeling tools be used for optimization-based falsification of Cyber-Physical Systems?*

One piece of this puzzle is attempted to be solved in Paper A. The main issue is that tools for falsification require specifications written in formal logic, but engineers in industry typically do not have the expertise necessary to write correct specifications in these frameworks. The solution of automatically transforming specifications from Simulink into logic specifications is one of the main contributions of this thesis.

Research Question 2. *How can the optimization-based falsification process be changed to require fewer simulations when considering a single requirement, in the context of large-scale industrial systems?*

As the research area of optimization-based testing is quite close to industrial applications, the practical aspects are deemed important. Paper B presents comparisons between using different optimization problem solvers and quantitative semantics for STL to better understand how falsification can be improved for real-world systems.

Research Question 3. *How can the optimization-based falsification process be changed to require fewer simulations when considering multiple requirements at once?*

This question is the basis for Paper D, where we present an algorithm for Multi-Requirement Falsification. An evaluation of the industry-inspired specification benchmark from Paper C illustrates the differences between the classical single-requirement and the new multi-requirement approaches. This adjustment of the falsification process to consider many requirements at once, together with global sensitivity analysis to provide engineers insight into large systems by use of STL robustness values, is a further step towards bringing academic methods to industrial models.

Finally, when there is a set of generated test cases (no matter the method), there is a need to know how to evaluate them concisely, without losing too much information about the test cases themselves. These thoughts are summarized in the final research question.

Research Question 4. *How can we evaluate how well testing of Cyber-Physical Systems fulfills structural coverage criteria, and how can we then use these criteria to assess when testing is considered finished?*

Paper E tackles this question. More specifically, a new kind of coverage criterion is presented which is defined based on the dynamical equations of the CPS model. This novel *mode coverage* is calculated both for a simple illustrating example, as well as for a use case (a model at Volvo Car Corporation). The conclusion is that a mostly automatic approach can be used to analyze how thoroughly the physical behavior of the model has been tested.

The paper makes it clear that both code coverage and mode coverage can be useful, but for different purposes. Code coverage (MC/DC) is typically used as a minimum requirement for evaluating whether a test suite (a collection of test cases) has exercised the SUT enough. As such, MC/DC is given as a percentage, and testers seldom reflect over *why* a certain degree of MC/DC is fulfilled or not fulfilled. On the other hand, mode coverage is more to be used as a basis for further analysis, especially when the mode coverage is not 100%, since the testers then should investigate the physical behaviors in the modes that are never visited by the test suite. In this way, mode coverage typically requires more work from a tester after the automatic analysis has been performed, in contrast to the established MC/DC code coverage criterion that is just used as a check that the test suite fulfills some basic properties.

1.3 Methodology

The purpose of the research presented in this thesis is to create a further understanding of the testing problem for Cyber-Physical Systems. The research has been experimental, as is common for research in this area, where the basis for evaluation of the research typically consists of different mathematical models of systems (benchmark models). The aim was to address the more practical research problems, which is especially clear in the formulations of research questions 2 and 3. An overview of the research methodology is shown in Figure 1.6. This also corresponds closely to suggested research models in systems engineering [16], where it is clear that industrial research originates from an industrial problem and then evolves towards both an industrial goal and academically viable research.

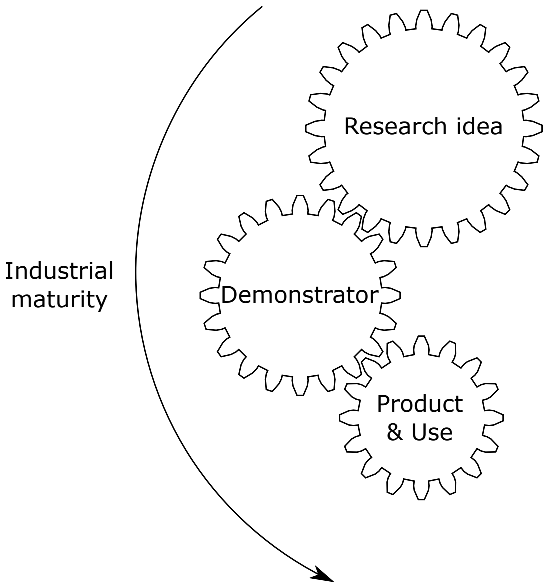


Figure 1.6: An overview of the research methodology used in this thesis. A research challenge is combined with an industrial need to produce a research idea. After some research has been performed, a demonstrator illustrates how to bridge the gap between academia and industry. Finally, the research can result in a product that may be used in real industrial applications as the project grows more mature.

As the research area of testing CPSs is tightly connected to applications, it has been an important goal to conduct research that applies to large-scale systems and not only smaller, simpler systems. While smaller systems have the merit that they are typically easier to analyze and present, the potential negative effect is that they do not always correspond very well to actual systems found in industry. With this in mind, one of the motivations for the conducted research has been to work with methods that scale well for industrial systems, but also to present the results in such ways that the issues of large-scale systems are clearly shown by the use of smaller examples.

Even though most of the results are applied to either previously available smaller benchmark models or proprietary models that we cannot disclose details of, we have also presented new specification models inspired by industrial examples that allows for the greater research community to develop methods more relevant to industrial use cases. While we cannot discuss exactly how the specification benchmark in Paper C was procured, it is obvious that the complexity and number of specifications presented give a novel way of evaluating specification-related algorithms.

The research questions were deliberately designed throughout the project. In the same way that the problems themselves were designed based on the need at Volvo Car Corporation, so were also the research questions posed based on how the industrial solutions could positively affect the research community.

Method

To be able to carry out meaningful research, the academic state-of-the-art was investigated in an extensive literature review. In the research area of testing of CPSs, there are many works that present tools, algorithms, and extensions to previous works that increase testing capabilities using new approaches. The decision to first focus on coverage, the subject of Paper E, was due to a need to analyze a set of test cases for a model already existing at Volvo Car Corporation. The model in question was also included in the paper as a case study.

Similarly, the reason for choosing *falsification* of CPSs as a method to further develop and adapt for industrial models was that there was a need from the industrial perspective to automatically generate new test cases for models present at Volvo Car Corporation. When the falsification approach was applied to several models, we noticed shortcomings of parts of the procedure, which led to subsequent research, and which also led to papers A, B, C, and D.

For all the papers appended to this thesis, the goal of tables and figures with results have always been to illustrate as many aspects of the treated problem as possible. In Paper E, the experiments were performed using OpenModelica [17] and the programming language Python, with test cases previously created by TestWeaver [18]. The experiments in all remaining papers were performed using MATLAB [19].

From the research project's point of view, the models to be tested at Volvo Car Corporation were chosen beforehand, as they are the models being developed at the particular part of the organization where the research takes place. However, for the academic evaluation of research results, the benchmark models had to be chosen somehow. Since there at the start of the project was no concrete set of benchmarks used by the whole research community, several different models were picked from different publications in the field. The benchmark models used for evaluation in the appended papers have appeared in several different papers. As for the smaller examples in the appended papers, they were created to highlight the contribution of the paper as clearly as possible, to hopefully make the paper easy to read and understand.

The structure of the results presented in the papers, such as the tables of falsification rates and cactus plots over optimization solver performance, were typically chosen with respect to how other papers in the field typically present their results. As an example, it is common to present falsification rates and the number of simulations (sometimes with both mean and median values), but not as common to include, e.g., measures for statistical significance and hypothesis tests. It would be possible, at least theoretically, to improve the statistical significance of the presented results, e.g., by having more repetitions of the experiments or by generating more examples (like automatically generated STL formulas to falsify). However, generating more repetitions of the experiments was usually too time-consuming, and automatically generating more STL formulas would go against the goal of the thesis to keep experiments and evaluations as relevant as possible to large-scale industrial systems.

Analysis

As all research aims to be reproducible, that has also been our main goal in the content presented in the appended papers. However, since several of the use cases presented in the papers (particularly the dog clutch model in Paper E, and the Volvo models discussed in Paper A) are proprietary, exact reproducibility is impossible and the scientific integrity of the author has to be trusted. In the cases where there is a use case from Volvo Car Corporation included, there is also a simplified example included to motivate the reasons for the approach presented in the specific paper.

Limitations of the methodology

Choosing methods that are relevant for industrial problems might not result in the research that is the most appealing from a theoretical point of view, however, it is still interesting for the research area in question since the field is in its nature close to application. Choosing falsification as a means of assuring correct behavior of simulations of CPSs has the drawback that falsification and testing never guarantees the absence of faults. However, falsification is still widely accepted as a reasonable strategy for quality assurance since it scales well for complex systems.

1.4 Thesis outline

The thesis is divided into two parts. In the first part, an overview is presented to give the reader the understanding needed for the papers appended in the second part.

Chapter 2 contains a brief overview of why to perform testing for the software that is part of CPSs. There are also presentations of coverage criteria (needed for understanding Paper E) and random testing (needed for Paper A).

Chapter 3 is about optimization-based falsification of CPSs. In this chapter, the falsification process is detailed, including a definition of Signal Temporal Logic for discrete-time signals. These definitions are useful for understanding papers A, B, C, and D.

Chapter 4 summarizes the content of the appended papers as well as the contributions of the thesis.

Chapter 5 contains a conclusion of the work presented in the thesis and an outlook on the future work to be done.

CHAPTER 2

Testing of Cyber-Physical Systems

This chapter gives a short insight into what testing is, and the different kinds of testing that are related to this thesis. Section 2.1 presents details on Cyber-Physical Systems and the requirements of such systems. In Section 2.2, there is a brief discussion about why testing is a reasonable approach to asserting the wanted behavior of CPSs. Section 2.3 discusses coverage criteria for testing and how these are used in industry. In Section 2.4, random testing is presented, and in Section 2.5 the falsification problem is introduced. Finally, in Section 2.6 there is a discussion on the alternative approach of using Reinforcement Learning for falsification.

2.1 Cyber-Physical Systems

Cyber-Physical Systems are systems that interact with the physical environment through the use of sensors (for acquiring information) and actuators (for affecting the physical surroundings) [20]. The main differences from mechatronic systems are that a CPS can be connected to and communicate with other CPSs, and a CPS consists of several different integrated subsystems [21]. As systems get larger and more complex, a CPS can be seen as part of the transition chain going from first a mechatronic system, then to a CPS, and then to a cloud-based system. As an example from the automotive domain, a drivetrain for a vehicle is considered a mechatronic system, while an entire car is considered to be a CPS. In the typical block diagram of a feedback system, as seen in Figure 2.1, this thesis considers testing all parts of the system. This includes the plant, so while the software in the controller is part of the system under test, the testing in the thesis can also be used to find, e.g., errors in how the plant is modeled in a simulated environment.

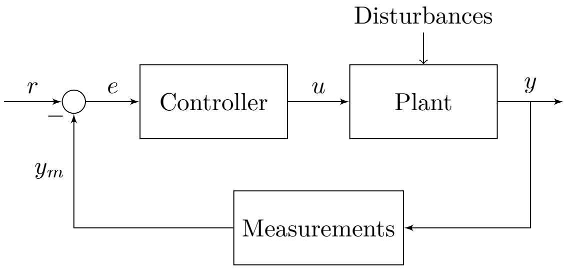


Figure 2.1: A block diagram of a typical feedback system. In this thesis, the entire system is considered as the system under test, meaning that we do not only consider the software as the goal system of the testing activities presented, but also the plant to be controlled.

Requirements of CPSs

CPSs are typically safety-critical, meaning that a failure in the operation of the system can result in serious damage or injury. Therefore, there is much focus in research to make sure that CPSs conform to safety requirements[1] [22]. However, it is not trivial to formulate the requirements to be put on CPSs.

As an extended example, consider a hypothetical requirement on a specific CPS, namely a car. This is to help illustrate the kind of requirements that could exist in industry and therefore also inspire the transformation approach used in Paper A. However, as real industrial requirements are proprietary,

[1]Note that *requirement* and *specification* typically refers to the same thing. However, the word requirement is typically used in industrial contexts, and the word specification is typically used to denote more formal or mathematical objects in academic contexts.

only a discussion on hypothetical requirements can be included in this thesis. Consider therefore that we have the following requirement for the car:

Requirement 1. *The car should be comfortable to drive.*

This requirement is very abstract. On one hand, many would be able to evaluate whether or not this requirement holds after driving the car for a few hours. On the other hand, it is unclear how to formally specify this requirement, and specifically, it is impossible to test that parts of the system, e.g. certain software, fulfills its part of the requirement.

To make the requirement testable, it needs to be broken down into sub-requirements. Of course, a requirement like the one presented can be interpreted in many different ways and will be considered fulfilled in different ways depending on who is asking. Now, different attempts at breaking the requirement down into sub-requirements will be performed; an overview of how the different requirements relate to each other is shown in Figure 2.2. The first refinement follows in Requirement 2:

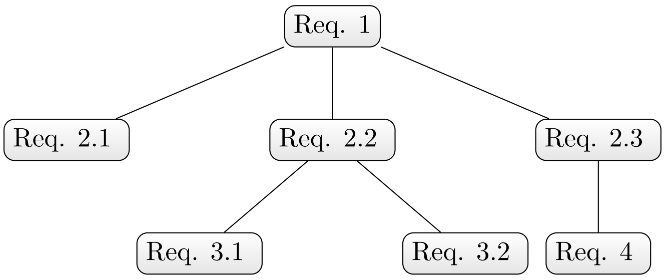


Figure 2.2: A tree describing how different sub-requirements are related. Each node is a refinement of its parent.

Requirement 2. *To be considered comfortable (and therefore fulfill Requirement 1), the car should fulfill all of the following:*

2.1 The car seats should be ergonomic;

2.2 The inside of the car should reach a comfortable temperature quickly after starting, and

2.3 When the driver presses down the accelerator pedal, the car should respond quickly.

It should be clear that Requirement 2 is for most people not enough to describe that a car is comfortable, but this analysis is limited to the three sub-requirements presented, to keep the example short enough for presentation. The refinement will continue only for the requirements that can be considered control-related, as those are the ones that could be used as specifications in a simulation-based testing environment where finding faults in the control system is the main goal (such as the testing environment used in most of this thesis).

Requirement 2.1 is not control-related; it is rather an issue to solve with the hardware of the car. However, in a modern car, both Requirement 2.2 and Requirement 2.3 are likely control-related. For them to be testable, they need to be more clearly defined, so that a simulation environment can somehow evaluate whether the requirement has been fulfilled or not. Another level of refinement is applied, which results in requirements 3 and 4.

Requirement 3. *When the car starts,*

3.1 The inside of the car should reach $21°C$ *within 2 minutes, and*

3.2 The inside temperature of the car should never reach above $23°C$*.*

Requirement 4. *To feel like the car responds quickly to desired acceleration if the angle of the accelerator pedal is larger than or equal to* $70°$ *and the current speed is lower than or equal to* $200\,km/h$*, the gear must be shifted correctly within* 0.5 *seconds and then the maximum acceleration force must be felt by the driver within another* 2 *seconds. Otherwise, if the angle of the accelerator pedal is lower than* $70°$ *or the current speed is higher than* $200\,km/h$*, the behavior of acceleration is defined by another requirement.*

The requirements can be refined even further, but the main point of the refinement process has been shown: the further a requirement is refined, the clearer it is which signals of the system that must be included to evaluate the requirement. It is also typical that to check whether a requirement has been fulfilled by a specific test case, a set of *prerequisites* have to be fulfilled. These prerequisites correspond to different entries in the tables of formulas discussed in the transformation of specifications in Paper A. For example, for

Requirement 4, one precondition would be that the angle of the accelerator pedal is larger than or equal to 70° *and* that the current speed is lower than or equal to 200 km/h.

2.2 Formal verification versus testing

Formal verification includes approaches such as model checking [23], [24] and deductive verification [25], [26] to verify correctness of models. If there is an error with regards to a specification in the model, these methods will find them and provide counterexamples to the specification. If there are no errors, the methods provide formal proof of correctness for the specification. While this sounds appealing, formal verification techniques have limitations and are not possible to use for general industrial CPSs. In fact, the general problem of verifying properties for hybrid systems, i.e., systems with both discrete and continuous dynamics, is undecidable [27]. This means that it has been proven that in the general case, no algorithm can decide whether a certain property for a hybrid system holds or does not hold. In addition to this, while formal verification methods are very useful for models without the limitations discussed here, there are several other obstacles to overcome [28] for model checking to be viable in industry (the most notable being a lack of experience in industry in formalizing the models and specifications to be checked).

With this in mind, we turn to testing instead. Testing is non-exhaustive, meaning that no matter how long we test, we can not *prove* the absence of faults, but testing can still raise confidence in the correctness of the final product. Testing is scalable and usable for complex industrial-sized systems, making it suitable for the research presented in this thesis which is close to application.

2.3 Coverage criteria

Testing the inner structure of the SUT is called white-box testing while testing the system behaviors without considering the inner workings of the SUT is called black-box testing. If the scope is to perform white-box testing, one may be interested in looking at different code coverage criteria for evaluating if the test cases have tested the system appropriately or not. For examples of some common coverage criteria [7], consider the simple example below.

```
1: if ($a$ and $b$) or $c$ then
2:     $x = x + 1$
3: else
4:     $x = x - 1$
5: end if
```

To fulfill *statement coverage*, every statement needs to be executed by the test suite. To fulfill *branch coverage*, every *branch* of the program needs to be executed. In this case, there are two branches; the “if” branch (row 2) and the “else” branch (row 4), which means that “(a **and** b) **or** c” has to evaluate to *true* at least once and *false* at least once in the test suite.

Fulfilling *decision coverage* is sometimes defined as fulfilling branch coverage, and sometimes as making sure that every point of entry and exit in the program has been invoked at least once as well as that every *decision* in the program has taken all possible outcomes at least once [7]. A decision is a Boolean expression composed of *conditions* and zero or more Boolean operators, whereas a condition is a Boolean expression containing no Boolean operators. In the given example, “(a **and** b) **or** c” and “a **and** b” are decisions, while “a”, “b” and “c” are conditions.

Another coverage criterion that covers more than the ones mentioned above is *Modified Condition/Decision Coverage* (MC/DC). MC/DC is especially interesting because it is used widely in industry to validate test suites, and because MC/DC is highly recommended for ASIL D (the highest classification of *Automotive Safety Integrity Level*) in ISO 26262 [29]. MC/DC requires all of the following:

1. Every point of entry and exit in the program has been invoked at least once.

2. Every condition in a decision in the program has taken all possible outcomes at least once.

3. Every decision in the program has taken all possible outcomes at least once.

4. Each condition in a decision has been shown to independently affect that decision’s outcome. A condition is shown to independently affect a decision’s outcome by varying just that condition while holding fixed all other conditions.

Below is a short analysis of what is needed to fulfill each of the points of MC/DC for the given code example.

1. To fulfill the first point, branches of the **if**-statement need to be exited, meaning that "(a and b) or c" needs to evaluate to true at least once, and false at least once.

2. To fulfill the second point, each condition (a, b, and c) must be true at least once, and false at least once.

3. To fulfill the third point, it is enough for this example to fulfill the same things as the first point since there is only one decision present.

4. To fulfill the fourth point is the trickiest. Consider the two cases below

   a = true, b = true, c = true

   a = false, b = false, c = false

   These inputs fulfill the first three points, but they do not fulfill the fourth point. The reason is that for both of these cases, none of the conditions a, b, or c *independently* affect the decision's outcome. Instead, an example of cases required to fulfill MC/DC is shown below (where conditions in **bold** can be shown to independently affect the decision's outcome by keeping the other conditions fixed).

   a = true, **b = true**, c = false

   a = false, b = true, **c = false**

   a = true, **b = false**, **c = false**

   a = false, b = false, **c = true**

Coverage criteria for Cyber-Physical Systems

In the realm of testing Cyber-Physical Systems, different kinds of coverage criteria have also been considered to improve the testing procedure. Several of these coverage notions relate to the continuous state variables in the systems. One approach [30] uses a discretized version of *dispersion*, where dispersion is a notion of coverage for continuous state spaces that measures the largest empty range of a point set based on a set of grid points with a fixed size.

An illustration of the discretized version is shown in Figure 2.3, where the coverage c of the point set P is calculated as

$$c(P) = \frac{1}{\delta} \sum_{j=1}^{n_g} \frac{\min(d_j, \delta)}{n_g}, \tag{2.1}$$

where n_g is the number of points, d_j is the minimum distance from grid point j to the set P, and δ is the grid spacing.

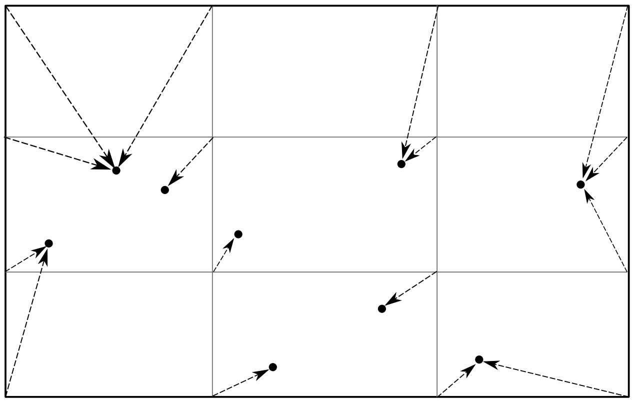


Figure 2.3: An illustration of the discretized version of dispersion, a coverage measure for continuous state spaces. The state space is divided into a 3x3 grid. The coverage measure is the average of the smallest distances from each grid point to the closest point in the point set. Here, the smallest such distances are shown as dashed arrows.

Another notion of coverage is the *star discrepancy* [31], which measures how well-filled a set of points is. In [31], this is applied to the values of the continuous state variables in a test suite for the SUT, and the test generation algorithm presented uses star discrepancy as a guide to find new input values for the SUT. To define the star discrepancy, we first define the *local discrepancy* as follows. Let P be a set of k points inside $\mathcal{B} = [l_1, L_1] \times \ldots \times [l_n, L_n]$. Here, $l_1, \ldots, l_n$ are lower limits and $L_1, \ldots, L_n$ are upper limits, i.e., $\mathcal{B}$ is a hyperrectangle in n dimensions. Let $\mathcal{J}$ be the set of all sub-boxes J of the form $J = \prod_{i=1}^{n} [l_i, \beta_i]$ with $\beta_i \in [l_i, L_i]$. In other words, $\mathcal{J}$ is the set of all possible sub-boxes with the lower corner fixed in the lower corner of the hyperrectangle $\mathcal{B}$. The local discrepancy of the point set P with respect to the sub-box J is defined as

$$D(P, J) = \left| \frac{A(P, J)}{k} - \frac{\text{vol}(J)}{\text{vol}(\mathcal{B})} \right|, \tag{2.2}$$

where $A(P, J)$ is the number of points of P that are inside J, and $\text{vol}(J)$ is the volume of the box J. The star discrepancy of the point set P with respect to the box $\mathcal{B}$ is then defined as

$$D^*(P, \mathcal{B}) = \sup_{J \in \mathcal{J}} D(P, J). \tag{2.3}$$

The star discrepancy is a number $0 < D^*(P, \mathcal{B}) \leq 1$, where a small value means that the points in P are well equidistributed over $\mathcal{B}$. In other words, one way to use the star discrepancy in test generation of CPSs is to try to minimize $D^*(P, \mathcal{B})$ over the continuous state space to make sure that the generated test inputs are evenly spread inside their respective ranges. A two-dimensional illustration of a sub-box $J = [l_1, \beta_1] \times [l_2, \beta_2]$ in the box $\mathcal{B} = [l_1, L_1] \times [l_2, L_2]$ is shown in Figure 2.4 [31].

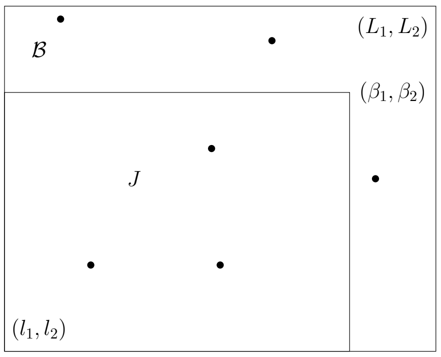


Figure 2.4: An illustration of the notation used in the definition of the star discrepancy. The sub-box $J = [l_1, \beta_1] \times [l_2, \beta_2]$ lies in the box $\mathcal{B} = [l_1, L_1] \times [l_2, L_2]$. This figure is taken from [31].

In [32], the authors present several coverage metrics to be used as evaluation of a requirements-driven falsification tool. The proposed coverage metrics typically include information about the discrete states of the hybrid SUT, such as the time spent in each unique mode of the system.

2.4 Random testing

A testing technique relevant to this thesis is random testing (or *randomized* testing; the terms will be used interchangeably in this thesis). In random testing, the user supplies the testing tool with properties that need to be tested, and generators that define how the inputs to a program can be created [33] (at least for user-defined types where the testing tool cannot know how to generate data otherwise). Some different random testing techniques are *fuzz testing* [34], where faulty inputs are sent into a program to test its security, and *concolic testing* [35], where random inputs are combined with symbolic execution to easier find very specific path conditions leading to bugs in programs.

As an example of random testing, consider the following (faulty) implementation of a function to calculate the absolute value of a number.

```
function ABS($x$)
    if $x > 0$ then
        return $x$
    else
        return $x$                                  ▷ Should be $-x$
    end if
end function
```

Depending on the tool and the type defined for x, one might need to define a generator for x. For example, one alternative is that x should be any signed integer, and another is that x should be a double in the range $[-100, 100]$. For the sake of this example, we choose the second alternative as the input generator for x. A reasonable specification to test against for the absolute value could for example be $\forall x\ x \geq 0$. Many test cases can be generated automatically, where the input of each test case is a random number according to the specified generator. Since approximately 50% of the generated input would fail the specification, the bug in the code would be found easily with random testing.

For the given example, the input is simply a random number, but input generation can easily be generalized to support e.g. random testing of simulated CPSs. For example, consider a model of a car, where the input defines how much to accelerate (a percentage of the full acceleration in the range $[0, 100]$). The input generator would then need to create a time-indexed vector, where for each time the input has a value in $[0, 100]$. Note, however, that it is probably not reasonable to simply generate a new random value for each time instance, independent of neighboring values, as this would be an unrealistic scenario (nobody could push and release their foot on the accelerator pedal tens of times per second). To circumvent this, one could do one of the following (or a combination of both):

- Make sure that the input generator only generates smooth curves, e.g., by generating an appropriate start value and then randomly selecting a second derivative, which is then integrated twice to provide the final input (making sure that the twice integrated values are in the interval $[0, 100]$).
- Generate purely random values for each time, but *shrink* the generated test case when a failure is found. Shrinking keeps simplifying the input of the test case as long as it keeps failing the specification, which could result in a physically reasonable input after the shrinking process finishes (for details, see [36]).

To summarize, random testing is a form of software testing that can be proficient at finding certain kinds of bugs in code, given that the testing problem is set up properly. If each test is executed quickly, it is intuitive that performing random testing can be a smart way to cover many different test scenarios. However, if a test is expensive to execute, e.g., if it requires simulating a complex dynamical system, one could also hypothesize that it is important to minimize the total number of tests generated. In other words, to automatically generate test cases for Cyber-Physical Systems might require several other considerations as well – something that is covered in more detail in Chapter 3.

2.5 Falsification

Falsification is a testing method where the goal is to find a counterexample to a specification of a system, given that we have access to the input-output behavior of the system. Thus, falsification is a black-box method that can be formulated as follows.

- **Given**: a system *model $\mathcal{S}$*, and a *specification φ*.
- **Results for falsification**: a *counterexample* that falsifies the specification, φ, if found in the falsification process.

While Chapter 3 contains detailed information about optimization-based falsification techniques, the main content of this thesis, there are also other approaches for falsifying specifications for CPSs – such as Reinforcement Learning.

2.6 Reinforcement Learning for falsification

An increasingly popular approach is falsification of temporal logic specifications for CPSs using Reinforcement Learning [37] and Deep Reinforcement Learning [38], [39]. Reinforcement Learning trains an agent based on interactions with the environment, in this case via actions sent to stimulate the CPS under test. At each decision step, the agent applies an action defined by its *policy*, and the environment responds by updating the state of the system and supplying a reward to the agent based on the action. While the reward is immediate for each action applied by the agent, the *value function* describes the expected sum of discounted future rewards that an agent can achieve from a certain state while following its policy. The agent's goal is to maximize the value function by optimizing its policy, and the environment is typically modeled as a finite Markov Decision Process [37] – a kind of process that satisfies the Markov Property, meaning that the probability of moving into a specific new state is only influenced by the current state and the agent's current action, i.e., it is conditionally independent on all previous actions and states.

Recent works have shown good falsification results both when generating inputs and rewards interactively during simulation [40], and when generating the entire input to a simulation beforehand and evaluating the reward after the entire simulation concludes [41]. The latter approach generalizes the concept of adaptive stress testing [42], where the goal is to learn an *adversarial policy* which causes the system to not fulfill its specifications.

While this chapter briefly introduces some basic concepts of the general testing problem for CPSs, Chapter 3 takes a deep dive into the core subject of this thesis: *optimization-based* falsification of CPSs.

CHAPTER 3

Optimization-based Falsification of Cyber-Physical Systems

This chapter starts with a presentation of Cyber-Physical Systems in general and continues with the basics of optimization-based falsification of CPSs. Optimization-based falsification of CPSs is the main subject of Papers A, C, B, and D. Included in this presentation is the definition of discrete-time signals, Signal Temporal Logic (STL), and the robust satisfaction of STL formulas. The falsification loop is summarized and explained in further detail with the aid of a simple example. Finally, optimization heuristics and solvers are presented in further detail, and then there is a discussion on further important aspects of applying optimization-based falsification to large-scale systems.

3.1 Discrete-time signals

Falsification relies on the monitoring of signals with respect to specifications written in temporal logic. As such, falsification in literature is usually defined in relation to continuous signals [10]. However, in Paper A, there is an extensive discussion about the semantics of logic for discrete-time signals. Since the research in this thesis considers simulations of CPSs, we naturally deal with discrete signals, which is why the discrete-time presentation is chosen in both Paper A and this chapter. It is possible but not trivial to generalize these definitions to continuous time [43], [44].

Definition 1. *A* discrete-time signal $x[k]$ *is a function* $x\colon I \to \mathbb{R}$ *from a finite subset of* $I \subset \mathbb{Z}$ *to* $\mathbb{R}$*, where* $k \in I$*. The set* I *indexes the time instants of the signal, and the signal takes on continuous values at each of those time instants.*

3.2 Signal temporal logic

Signal Temporal Logic (STL) [9] was originally introduced as an extension of Metric Temporal Logic (MTL) [8] with real-valued signals, while MTL itself is an extension of Linear Temporal Logic (LTL) [45]. The additions of STL compared to LTL consist of real-valued signals and dense time (as opposed to Boolean expressions and modalities). LTL was originally designed for formal verification of software by encoding formulas of what should hold for the systems that should be verified.

The grammar of STL formulas is defined here as

$$\varphi ::= \mu \mid \neg\mu \mid \varphi \wedge \psi \mid \varphi \vee \psi \mid \Box_{[a,b]}\varphi \mid \Diamond_{[a,b]}\varphi \mid \varphi\, \mathcal{U}_{[a,b]}\psi,$$

where μ is a predicate, and φ and ψ are STL formulas. μ is decided by the sign of a function of an underlying signal x, meaning that $\mu \equiv \mu(x[k]) > 0$.

Similarly to [46] and Papers A–D , the validity of a formula φ with respect to the discrete-time signal x at time instant k is defined as

$$\begin{aligned}
(x,k) &\models \mu && \Leftrightarrow \mu(x[k]) > 0 \\
(x,k) &\models \neg\mu && \Leftrightarrow \neg((x,k) \models \mu) \\
(x,k) &\models \varphi \wedge \psi && \Leftrightarrow (x,k) \models \varphi \wedge (x,k) \models \psi \\
(x,k) &\models \varphi \vee \psi && \Leftrightarrow (x,k) \models \varphi \vee (x,k) \models \psi \\
(x,k) &\models \Box_{[a,b]}\varphi && \Leftrightarrow \forall k' \in [k+a, k+b], (x,k') \models \varphi \\
(x,k) &\models \Diamond_{[a,b]}\varphi && \Leftrightarrow \exists k' \in [k+a, k+b], (x,k') \models \varphi \\
(x,k) &\models \varphi\, \mathcal{U}_{[a,b]}\psi && \Leftrightarrow \exists k' \in [k+a, k+b] \quad (x,k') \models \psi \\
& && \quad \wedge \forall k'' \in [k,k'), (x,k'') \models \varphi
\end{aligned}$$

Table 3.1 contains examples to clarify how the different operators behave.

Table 3.1: Examples of different STL operators.

Symbol	Meaning	Example
$\neg$	Logical `NOT`	$\neg(x > 0)$
$\wedge$	Logical `AND`	$(x > 0) \wedge (x < 10)$
$\vee$	Logical `OR`	$(x < 0) \vee (x > 10)$
$\Box_{[a,b]}$	Timed `always`	$\Box_{[0,5]}(x > 20)$
$\Diamond_{[a,b]}$	Timed `eventually`	$\Diamond_{[0,4]}(x = 5)$
$\mathcal{U}_{[a,b]}$	Timed `until`	$(x = 1)\ \mathcal{U}_{[0,5]}\ (y > 10)$

The interpretation of the examples is shown in Table 3.2.

Table 3.2: How to interpret the examples from Table 3.1.

$\neg(x > 0)$:	x is not greater than 0.
$(x > 0) \wedge (x < 10)$:	x is between 0 and 10.
$(x < 0) \vee (x > 10)$:	x is less than 0 or greater than 10.
$\Box_{[0,5]}(x > 20)$:	For all times between and including 0 and 5, x is greater than 20.
$\Diamond_{[0,4]}(x = 5)$:	Between and including times 0 and 4, there is at least one time when x is equal to 5.
$(x = 1)\ \mathcal{U}_{[0,5]}\ (y > 10)$:	For some time k' between and including 0 and 5, y is larger than 10. For all times before this time, i.e., for times between 0 and k' (including 0 but not k'), x is equal to 1.

Robust satisfaction of STL formulas

One of the main focus areas of STL formulas is their quantitative semantics [44] (or *robustness/robust semantics*) and the efficient monitoring of such semantics [47]. An extension to this is the ability for online monitoring of STL formulas [48], which for example enables the evaluation of specification fulfillment for partial traces of simulated systems.

The robustness of an STL specification is informally how “far away” the specification is from being falsified. The robustness ρ is a real-valued function, whose sign indicates if the corresponding specification φ is satisfied or not (negative means non-satisfied, positive means satisfied). $\rho(\varphi, x, k)$ is a function of a specification φ, a signal x (potentially a vector), and a time k at which the robustness is evaluated. The robustness is defined similarly to [46]:

$$\rho(\mu, x, k) = \mu(x[k]) \tag{3.1}$$
$$\rho(\neg\mu, x, k) = -\mu(x[k])) \tag{3.2}$$
$$\rho(\varphi \wedge \psi, x, k) = \min(\rho(\varphi, x, k), \rho(\psi, x, k)) \tag{3.3}$$
$$\rho(\varphi \vee \psi, x, k) = \max(\rho(\varphi, x, k), \rho(\psi, x, k)) \tag{3.4}$$
$$\rho(\Box_{[a,b]}\varphi, x, k) = \min_{k' \in [k+a, k+b]} \rho(\varphi, x, k') \tag{3.5}$$
$$\rho(\Diamond_{[a,b]}\varphi, x, k) = \max_{k' \in [k+a, k+b]} \rho(\varphi, x, k') \tag{3.6}$$
$$\rho(\varphi\, \mathcal{U}_{[a,b]}\psi, x, k) = \max_{k' \in [k+a, k+b]} (\min(\rho(\psi, x, k'), \min_{k'' \in [k,k']} \rho(\varphi, x, k''))) \tag{3.7}$$

Some examples follow. For the specification $\varphi_1 = (x[k] > 3)$, $\mu(x[k])$ is defined as $x[k] - 3$ and the robustness at time 0 is $\rho(\varphi_1, x, 0) = x[0] - 3$. Consider two simulations with resulting signals x_1, x_2 that satisfy φ_1 at time 0, and let us say that $x_1[0] = 5$ and $x_2[0] = 15$. Intuitively, $x_1[0]$ is closer to not satisfying the specification and thus should have a lower robustness value. Indeed, the robustness value of x_1 is 2 and the robustness value of x_2 is 12.

For the specification $\varphi_2 = \Diamond_{[a,b]}(x > 5)$, the robustness at time 0 is $\rho(\varphi_2, x, 0) = \max_{k' \in [a,b]}(x[k'] - 5)$. For this *eventually* operator, the robustness is the maximum of the robustness of its inner formula. It is always the case that the robustness of a specification with respect to a specific signal and time is a scalar.

Extensions of STL

While this thesis is focused on properties that actually can be expressed using STL, it should be noted that some properties need other formalisms to be expressed. For example, STL* [49] includes an additional *freezing* operator which makes it possible to specify damped oscillations in a signal, Time-Frequency Logic (TFL) [50] can be used to specify frequency-domain properties in addition to time-domain, and hyperproperties [51] are properties that require two or more execution traces to be evaluated.

A modified version of STL, avSTL [52], has been proposed, which introduces time-averaged temporal operators in place of $\Box$ and $\Diamond$. These time-averaged operators yield a different robustness value than the standard quantitative semantics, which is shown to be preferential in certain systems and specifications with an application to falsification. In a somewhat similar fashion, another change to the quantitative semantics of STL is “edit distance” [53], which is a metric that measures the distance in both time and space of an STL specification. In a related publication [54], the authors note that the monitoring of STL formulas can be considered as filtering over the signals in the specifications, both for the qualitative and quantitative semantics.

Writing formal specifications is a difficult task [55]. To alleviate the process of using formal specifications, approaches in earlier works have included tools that make it easier to write specifications [56], tools to automatically detect faulty specifications [57], as well as defining template specifications that make it easier for inexperienced testers to formulate the specifications [58].

To summarize, STL formulas can be used to specify desired behaviors of CPSs, and the robustness of STL formulas is used to give a measure of *how much* fulfilled or not a specification is (quantitatively), rather than just measuring whether it is fulfilled or not (qualitatively). One possible use of this robustness is to order a set of test cases based on how close they are to not fulfilling the specification. This naturally leads us to *falsification*, which is covered in the next section.

3.3 Optimization

The aim is to formulate falsification as an optimization problem, where the goal is to minimize the STL robustness value to find counterexamples. While falsification and optimization are not strictly equivalent – the only goal of falsification is a counterexample, while an optimization formulation aims to keep making a counterexample fail “as much as possible” according to some measure – a suitable general optimization problem can be formulated as

$$\begin{aligned} &\min_{\mathbf{x}} f(\mathbf{x}) \\ &s.t.\ \mathbf{x} \in [l, u], \end{aligned} \tag{3.8}$$

where f is the measure of how far the specification is from being falsified (typically the robustness of an STL specification), and $\mathbf{x}$ are the decision variables of the optimization problem, i.e., $\mathbf{x}$ is a vector. The decision variables are bounded according to

$$[l, u] := \{\mathbf{x} \in \mathbb{R}^n \mid l_i \leq a_i \leq u_i,\, i = 1, \ldots, n\},$$

where $l, u \in \mathbb{R}^n$ and $l_i < u_i$ for $i = 1, \ldots, n$, and n is the number of dimensions in the optimization problem, i.e., $\mathbf{x} \in \mathbb{R}^n$. However, it is not clear from this general formulation how the decision variables are related to the input signals of the SUT, or how those input signals are related to the signals used in the STL specification which f measures the robustness of. Those details are shown below in Section 3.4, but there are a few other things to take note of before going into more details on the optimization formulation for falsification.

First, we assume that the SUT is black-box, i.e., that we observe outputs for a given input (for example through simulation), but we cannot access the inner workings of the SUT. Specifically, we assume we have no access to any derivatives of the outputs with regards to the inputs, so there is no option but to use gradient-free methods to solve the optimization problem [59], [60]. Secondly, we assume that the decision variables in $\mathbf{x}$ have lower and upper bounds, which also means that the allowed search space is a hyperrectangle. There are approaches to include more general constraints on the decision variables [61], [62], but this is not trivial and not considered further.

3.4 Optimization-based Falsification

This section considers optimization-based falsification of CPSs as the method to generate new test cases. The reason STL is introduced in Section 3.2 is that the robustness of STL formulas is used as the objective function for the optimization. An overview of the falsification procedure is shown in Figure 3.1.

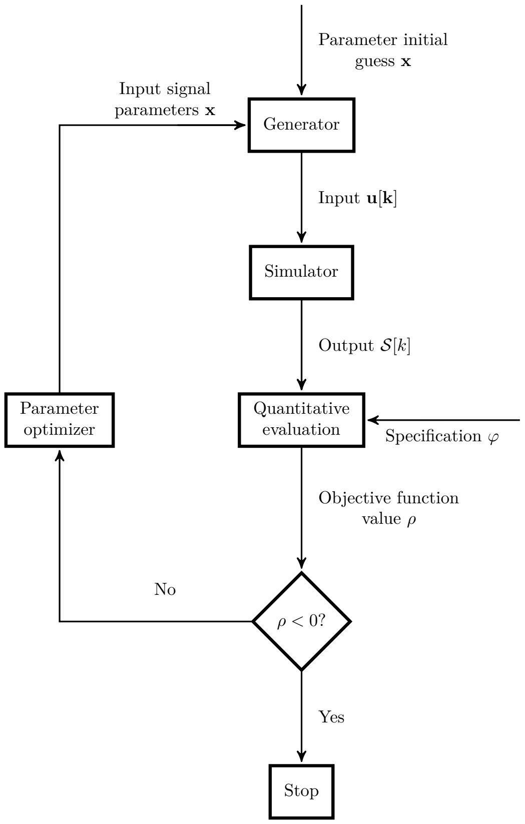

Parameter initial guess $\mathbf{x}$
Input signal parameters $\mathbf{x}$
Generator
Input $\mathbf{u}[\mathbf{k}]$
Simulator
Output $\mathcal{S}[k]$
Parameter optimizer
Quantitative evaluation
Specification φ
Objective function value ρ
No
$\rho < 0$?
Yes
Stop

Figure 3.1: A overview of the optimization-based falsification procedure.

The *Generator* takes the input parametrization to generate an input to the SUT. The *Simulator* generates a simulation trace, which is used together with the requirement φ to evaluate the robustness function for the simulation. The robustness function ρ is evaluated to see whether the specification is falsified or not. If not falsified, new parameters are sampled and the process is repeated. The different parts of this procedure are now presented in more detail.

Falsification can, as stated earlier, be formulated as a minimization problem. The problem is non-linear and thus hard to solve. To be more specific, the objective function is non-linear, which can be seen from how the robustness is defined in (3.1)–(3.7) The constraints, originating from the domain of the parameters, are linear. The falsification is performed in the following way [63].

The space of permissible input signals is parametrized by *n input parameters* $\mathbf{x} = (x_1, \dots, x_n)$ that take values from a set $\mathcal{P}_u$. The actual input $\mathbf{u}[k]$ is created using a generator function g such that $\mathbf{u}[k] = g(v(\mathbf{x}))[k]$, where $v(\mathbf{x}) \in \mathcal{P}_u$ is a valuation of the parameter vector $\mathbf{x}$. The output signal is calculated using the black-box *system* $\mathcal{S}$.

The optimization problem is now formulated as

$$\begin{aligned} &\underset{v(\mathbf{x})\in\mathcal{P}_u}{\text{minimize}} \qquad \rho(\varphi,\ \mathcal{S}(g(v(\mathbf{x}))), 0) \\ &s.t.\ \mathbf{x} \in [l, u], \end{aligned} \tag{3.9}$$

where the initial guess is called $v_i(\mathbf{x})$. Formulating the optimization problem enables the use of general-purpose optimization solvers, which is the main way falsification has been carried out in this research project.

Input generators

The question is how to define suitable $\mathbf{x}$ that parametrize the inputs to the system $\mathcal{S}$. This requires expert knowledge of the system and is not something that can be easily solved in general, since there can be complicated dynamics in $\mathcal{S}$ and, for industrial systems, unknown assumptions on the inputs. For the optimization problem, it is preferable to use as few parameters as possible, as each parameter increases the dimension of the search space. However, if there are too few parameters, the inputs generated might not be expressive enough, and therefore the falsification procedure can miss reasonable test cases that would falsify the specification.

Quantitative evaluation

The standard robustness function ρ presented in (3.1)–(3.7) is the one used by most available falsification tools. As a negative robustness value means that the specification is falsified, the termination criterion is simply that the robustness is negative.

In Paper A and Paper D, alternative quantitative semantics are discussed. These make the falsification procedure end up with different results than if the standard "max" STL quantitative semantics are used, however, for all quantitative semantics the robustness is positive when the specification is fulfilled, and negative when the specification is not fulfilled. It is not possible to decide which quantitative semantics are better to use in general, as the performance depends on both the specification and the system. This is discussed in more detail in Paper A and Paper B.

Parameter optimizer

There has been much research on how to best select new parameter values in the attempt to generate simulations that give lower robustness values and therefore will be closer to falsifying the specification. Specifically for optimizing parameters in falsification, there are publications about Ant Colony Optimization [64], Simulated Annealing [11], a Line Search method [65], and Local Stochastic Tabu search [66], among others. The common denominator for these algorithms is that they do not require a gradient for optimization, which is essential if the SUT is considered black-box. However, there are also approaches to use gradients in the optimization, for example as in [67]. This is not something considered in this thesis as the systems considered are black-box and of considerable size.

Falsification in practice

To evaluate different forms of falsification against each other, different benchmarks [68], [69] of specifications and systems are typically used. Recently, a set of standardized benchmarks has been proposed as part of an annual friendly workshop competition in falsification [70].

Over the last few years, more and more tools have been introduced in the research area of falsification. The two biggest tools S-TaLiRo [71] and Breach [72] have been joined by many others. ARIsTEO [73] is a toolbox developed on top of S-TaLiRo, which is designed to target compute-intensive CPS models by using surrogate models to give faster execution of black-box testing. FalCAuN [74] is a tool that uses black-box checking [75], a method combining active automata learning and model checking, to test Simulink models. The tool falsify [76] uses a deep reinforcement learning algorithm as a grey-box method to falsify safety properties of systems. FalStar [77] uses an adaptive Las-Vegas tree search to generate inputs online during the test. ForeSee [78] uses a new robustness definition called *QB-Robustness* in an attempt to tackle the scale problems of STL specifications, where signals of different scales can mask each other's robustness values, which in turn makes the specification more difficult to falsify. zlscheck uses automatic differentiation to calculate gradients of the robustness with respect to input parameters, to falsify properties for programs written in Zélus. The Stochastic Optimization with Adaptive Restart (SOAR) framework [79] has been applied to the falsification problem [80], and can quantify the robustness uncertainty of generated tests even when no falsifications are found.

An application of falsification is parameter mining of temporal requirements from closed-loop models [63], where one can automatically find out which specifications a system fulfills, given template specifications of a certain form. Other recent modifications to the falsification procedure include falsification of systems with machine learning components [81], time staging [82], which splits the input generation into temporal parts, and specific procedures for better falsification of request-response specifications [83], [84]. Another way to generalize the falsification problem is to put the falsification in an outer loop where different parametrizations of the input signals are considered [85]. This is an approach that could be useful in the case when there is limited knowledge of how the inputs to the SUT typically look.

3.5 Falsification example

In this section, an example of a simplified falsification procedure is presented, the aim of which is to show more details about how the input generators and robustness functions work.

The model used for falsification in this example is a model of the automatic transmission system of a vehicle [68], and it is also used for evaluation purposes in Papers A, B, and D. The model has two inputs; the first input is the throttle

Table 3.3: Summary of input parameters and robustness value for the three traces in the example presented in this section.

Trace	**Input 1 parameters**							**Input 2 parameters**			**Robustness ρ**
1	1	30.4	69.4	0	58.2	100	26.9	249	451.2	369.2	1717.9
2	63.1	0	0	100	100	36.9	60.2	206.5	318.4	22.8	1263.6
3	0	0	0	33	57.1	100	44.8	500	357.8	0	-103.9

of the vehicle, which has a value in the range $[0, 100]$ at all times, and the second input is the brake of the vehicle, which has a value in the range $[0, 500]$ at all times. For the throttle, the input generator uses seven control points distributed evenly in time. Between these control points, each of which lies in the interval $[0, 100]$, the throttle values are interpolated using a standard MATLAB function.

An example of three generated input scenarios is shown in Figure 3.2.

The specification to be falsified is $\varphi = \Diamond_{[0,20]}(\omega \geq 2000)$, and can be interpreted in natural language as *the engine speed reaches 2000 RPM within 20 seconds*. In Figure 3.2, the specification is indicated in the bottom figure with a green box in such a way that a trace of the RPM has to enter the box to fulfill the specification. The robustness function of the specification, using the standard robustness function ρ, is

$$\rho(\varphi, x, 0) = \max_{k' \in [0,20]}(\omega[k'] - 2000). \tag{3.10}$$

In other words, the robustness of each trace is the difference between the maximum value of ω and 2000, in the first 20 seconds only. Table 3.3 shows the information that is available to the optimization solver during falsification, namely the input parameters and the resulting robustness value ρ from simulating the system with the input generated from the given parameters.

For the first trace, the maximum value of the engine speed in the first 20 seconds is 3717.9 at time 14.4 seconds, which gives $\rho = 1717.6$. For the second trace, the maximum value is 3263.6 at time 20 seconds, which gives $\rho = 1263.6$ – this is considered closer to falsification than the first trace. Finally, the third trace has a maximum value of 1896.1 at time 20 seconds, which gives $\rho = -103.9$. Negative robustness indicates that the specification has been falsified, and the given trace is a counterexample of the specification.

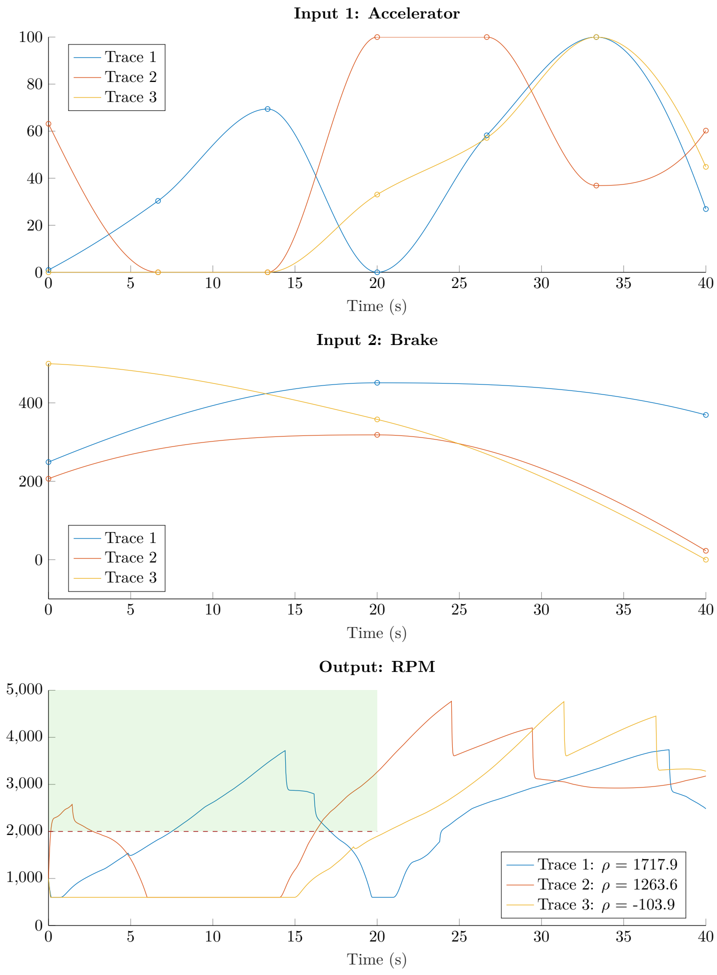


Figure 3.2: An example of three generated input scenarios for falsification of an automatic transmission system of a vehicle. The top figure shows input 1, which is parametrized by 7 control points spread evenly in time. The middle figure shows input 2, which is parametrized by 3 control points spread evenly in time. The bottom figure shows the resulting simulated RPM values for the three traces, and it also indicates a ”green box” which a trace must enter to fulfill the specification under test.

3.6 Optimization approaches and solvers

As mentioned earlier, much research has been performed to determine which gradient-free optimization solvers are efficient to use for the falsification problem in select public benchmark models. This section gives further details on the solvers and heuristics used in the appended papers in order for the reader to get an increased understanding of the methods evaluated in the research leading up to this thesis.

Uniform Random sampling

Uniform Random sampling is a simple way of generating parameter values to constitute a test for the SUT. Each parameter $x_i \in \mathbf{x}$ is independently sampled from a uniform random distribution, which means that each value in the interval $l_i \leq x_i \leq u_i$ has an equal probability of being sampled. Formulated another way, each parameter $a_i \in \mathbf{x}$ is sampled from the distribution which has the probability density function

$$f_{UR}(x_i) = \begin{cases} \frac{1}{u_i - l_i}, & \text{for } l_i \leq x_i \leq u_i \\ 0, & \text{for } x_i < l_i \text{ or } x_i > u_i. \end{cases} \tag{3.11}$$

Uniform Random sampling has been used as the standard baseline case to compare optimization solver performance against in the falsification research community. Findings in appended papers do however point to the fact that many public benchmark examples can be falsified using *corner sampling*, which is detailed next.

Corner sampling

In corner sampling, a corner constitutes a point which is a corner in the hyperrectangle that defines the allowed space of the input parameters $\mathbf{x}$. In other words, a corner sample is a parameter $\mathbf{x}_{corner}$ such that $x_i = l_i$ or $x_i = u_i \; \forall x_i \in \mathbf{x}_{corner}$. Corner sampling is used throughout the different optimization solver comparisons in the appended papers, and specifically, in Paper D, we propose a combination of corner sampling and uniform random sampling as a new baseline to compare optimization solver performance.

Simulated Annealing

A Simulated Annealing solver is used for comparing optimization solver performance in Papers A and B. Simulated Annealing is inspired by annealing in metallurgy, where a material is heated and cooled in a controlled process to affect the properties of the material. The solver is explained with the pseudo-code in Algorithm 1 – the original solver was a part of the S-TaLiRo toolbox [71], but the version included in Papers A and B is altered slightly [85].

Algorithm 1 Simulated Annealing algorithm

```
1: function SimulatedAnnealing(Parameter space $[l, u]$, robustness function $f$)
2:     Generate initial $\mathbf{x} \in [l, u]$
3:     while $f(\mathbf{x}) \geq 0$ do
4:         $\mathbf{x}' \leftarrow$ ProposalScheme($\mathbf{x}$)
5:         $\alpha \leftarrow e^{-\beta(f(\mathbf{x}') - f(\mathbf{x}))}$
6:         $r \leftarrow$ UniformRandomReal$(0, 1)$
7:         if $r \leq \alpha$ then
8:             $\mathbf{x} \leftarrow \mathbf{x}'$
9:         end if
10:    end while
11: end function
```

In line 2, the algorithm first generates an initial sample at random, then the robustness is calculated by simulating the system for the input based on the sample's input parameters. Whenever a negative robustness value is encountered, the algorithm terminates and returns the corresponding counterexample. In line 4, a new parameter vector is sampled according to a proposal scheme based on a normal distribution centered around the current point in the parameter space. The robustness is calculated for this new parameter sample and compared to the existing (best) robustness value, and based on this comparison, line 5 calculates the ratio α as a function of another parameter β. Line 6 samples a uniform random number in $[0, 1]$ and compares it to α to figure out whether the new parameter signal is accepted or rejected.

Note that if the new robustness value is lower than the previous one, $\alpha > 1$ and the new parameter sample is guaranteed to be accepted. Even if the new robustness value is higher than the previous, i.e. a worse sample is encountered, there is a non-zero probability of accepting it anyway. It should also be noted that β is typically updated during execution to allow the algorithm to escape local minima.

SNOBFIT

Stable Noisy Optimization by Branch and Fit, or SNOBFIT [86], is a solver used for comparisons in Papers B and D. It was chosen for evaluation in these papers as it had shown good performance in a review of derivative-free optimization algorithms [87]. SNOBFIT builds local models around generated parameter points to find new parameter points corresponding to lower robustness values. The algorithm generates parameter values belonging to five different classes:

- Class 1 contains at most one point, which is determined from a local quadratic model around the best point.
- Class 2 contains points which are guesses for an approximate local minimizer around a *local point*. A local point in this context is a point where the function value is significantly smaller than the function value of its nearest neighbors.
- Class 3 contains points generated in a similar way to those in Class 2, but from the non-local points instead.
- Class 4 represents the most global aspect of the algorithm by containing points in unexplored regions. These points are generated in large sub-boxes of the current partition of the input space.
- Class 5 contains points that are produced only if the desired number of points in Classes 1-4 cannot be generated. The points in Class 5 are generated in a way to maximize the coverage of the input space.

SNOBFIT has shown good performance in the experiments performed in the appended papers, especially in cases where limited robustness information is available from the system. For example, the specification $\square(\neg(\mathit{fail} = 1))$ for the Boolean signal $fail$ has no valuable robustness information, since it is either true or false and we cannot infer any distance to failure without knowing more about the system where the $fail$ signal is generated. The reason why SNOBFIT performs well even in such scenarios is that it tends to sample more points towards the extreme points of each input parameter's allowed range, i.e., points of Class 5 are more likely to be towards the corners. As mentioned earlier, corner points have been observed to give good falsification results in general for many available public benchmark models.

CMA-ES

Covariance Matrix Adaptation Evolution Strategy, or CMA-ES [88], is used in the comparison between optimization solvers in Paper B. The solver adapts the covariance matrix of a multi-variate normal search distribution to make a population of points converge to the global optimum. On a high level, the CMA-ES algorithm consists of the following points, which are repeated until a stopping criterion is met:

1. Sample a new population of search points from a multivariate normal distribution.

2. Select and recombine a new mean μ of the search distribution as a weighted average of some of the selected points from the sampled points.

3. An *evolution path* is created, which is a sequence of successive steps that is used to update the covariance matrix.

4. The step size of the evolution path is controlled using a number of auxiliary parameters.

5. The covariance matrix is updated using the evolution path (called the rank-one update) and the recombined new mean (called the rank-μ update).

The Nelder-Mead Simplex method

The Nelder-Mead Simplex method [89] is used in the comparison between optimization solvers in Paper B. It uses simplexes (a generalization of a triangle to arbitrary dimensions) to search for the minimum of the objective function starting from $n + 1$ initial samples for an n-dimensional optimization problem. The algorithm is presented in Algorithm 2, where we assume we start with a (randomly sampled) simplex in n dimensions consisting of the points

Algorithm 2 The Nelder-Mead Simplex method

```
function NelderMead(Parameter space $[l, u]$, robustness function $f$)
    while Stopping criterion not fulfilled do
        Arrange $f(\mathbf{x}_1) \leq \ldots \leq f(\mathbf{x}_{n+1})$    ▷ Ordering
        Calculate the centroid $\mathbf{x}_o$ of all points except $\mathbf{x}_{n+1}$
        $\mathbf{x}_r = \mathbf{x}_o + \alpha(\mathbf{x}_o - \mathbf{x}_{n+1})$    ▷ Reflection
        if $f(\mathbf{x}_1) \leq f(\mathbf{x}_r) < f(\mathbf{x}_n)$ then
            $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_r$
        else if $f(\mathbf{x}_r) < f(\mathbf{x}_1)$ then
            $\mathbf{x}_e = \mathbf{x}_o + \gamma(\mathbf{x}_r - \mathbf{x}_o)$    ▷ Expansion
            if $f(\mathbf{x}_e) < f(\mathbf{x}_r)$ then
                $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_e$
            else
                $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_r$
            end if
        else
            if $f(\mathbf{x}_r) < f(\mathbf{x}_{n+1})$ then
                $\mathbf{x}_{oc} = \mathbf{x}_o + \rho(\mathbf{x}_r - \mathbf{x}_o)$    ▷ Outside contraction
                if $f(\mathbf{a_{oc}}) < f(\mathbf{x}_r)$ then
                    $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_{oc}$
                else
                    $\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1), i = 2, 3, \ldots, n+1$    ▷ Shrinking
                end if
            else
                $\mathbf{x}_{ic} = \mathbf{x}_o + \rho(\mathbf{x}_{n+1} - \mathbf{x}_o)$    ▷ Inside contraction
                if $f(\mathbf{x}_{ic}) < f(\mathbf{x}_{n+1})$ then
                    $\mathbf{x}_{n+1} \leftarrow \mathbf{x}_{ic}$
                else
                    $\mathbf{x}_i = \mathbf{x}_1 + \sigma(\mathbf{x}_i - \mathbf{x}_1), i = 2, 3, \ldots, n+1$    ▷ Shrinking
                end if
            end if
        end if
    end while
end function
```

$\mathbf{x}_1, \ldots, \mathbf{x}_{n+1}$. Each iteration attempts to find a new simplex by replacing the vertex with the worst objective function value with a new point.

The algorithm generates the new point in different ways depending on the value of the objective function at the new point. The different ways to generate new points are reflection, expansion, contraction, and shrinking. The algorithm assumes $\alpha > 0$ (reflection parameter), $\gamma > 1$ (expansion parameter), $0 < \rho \leq 0.5$ (contraction parameter), and $0 < \sigma < 1$ (shrinking parameter).

An illustration of these ways to generate new points, inspired by [90], is shown in Figure 3.3.

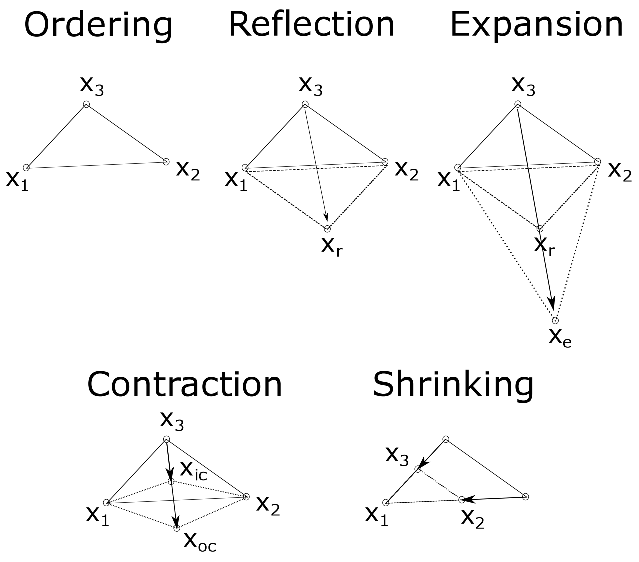


Figure 3.3: Illustration in two dimensions of different ways to generate new points in the Nelder-Mead Simplex method.

Bayesian Optimization

Bayesian Optimization [91] is an optimization method that builds probabilistic models (usually Gaussian Processes) of the function to be minimized. The models are used to choose between exploration, i.e., when to sample points to minimize uncertainty in unexplored regions of the input space, and exploitation, i.e., when to sample points where the predicted model function values are high. A high-level overview of Bayesian optimization is shown in Algorithm 3.

Algorithm 3 Bayesian Optimization

```
1: function BayesianOptimization(Parameter space $[l, u]$, black-box function $f$)
2:     Generate initial input samples, $\mathbf{x}_k, k = 1, \dots, N$
3:     Calculate objective function values, $f(\mathbf{x}_k), k = 1, \dots, N$
4:     Store data $\mathcal{D}_k = \{(\mathbf{x}_1, f(\mathbf{x}_1)), \dots, (\mathbf{x}_k, f(\mathbf{x}_k))\}$
5:     for i = 1,2,... do                    ▷ Repeat until stopping criterion met
6:         Select point $\mathbf{x}_{n+1}$ by optimizing an acquisition function $\alpha$

               $\mathbf{x}_{i+1} = \arg\max_{\mathbf{x}} \alpha(\mathbf{x}; \mathcal{D}_i)$

7:         Calculate objective function value $f(\mathbf{x}_{i+1})$
8:         Augment data $\mathcal{D}_{i+1} = \{\mathcal{D}_i, (\mathbf{x}_{i+1}, f(\mathbf{x}_{i+1})\}$
9:         Update statistical model
10:    end for
11: end function
```

The acquisition function α is what helps decide between exploration and exploitation, which means that it is a vital part of the algorithm. As such, there have been many acquisition functions proposed in the research of Bayesian Optimization; some of the most common ones are called Thompson sampling, probability of improvement, expected improvement, and upper confidence bounds.

The first Bayesian Optimization method aimed at high-dimensional input spaces was REMBO [92], and this method was adapted to be used for falsification as well [93]. However, it requires potentially complex user input, which is why the comparison to Bayesian Optimization in Paper D was performed using the TuRBo algorithm [94] instead. TuRBo has been noted as efficient in a similar context before [95], [96].

TuRBO

Trust region Bayesian optimization algorithm, or TuRBO [94], is a Bayesian Optimization method that is used as part of the Multi-Requirements Falsification in Paper D. The algorithm is specifically designed for problems with many dimensions by using a sequence of local optimization runs inside independent probabilistic models to avoid overexploitation of the search space. It simultaneously keeps track of several different *trust regions*, each of which is a hyperrectangle in the search space. By using Thompson Sampling, the algorithm draws samples from the union of all trust regions and then solves the local optimization problem from which each sample was drawn.

minBO

Minimum Bayesian Optimization, or minBO [97], is a Bayesian Optimization algorithm designed for falsification of conjunctive requirements for CPSs, i.e., requirements of the form $\varphi := \varphi_1 \wedge \varphi_2 \wedge \ldots \wedge \varphi_n$ – note here that n in this context refers to the number of conjunctive requirements, not the number of dimensions. While such a requirement could be falsified directly using the standard robustness function $\rho(\varphi, x, k) = \min_{i=1,\ldots,n} \rho(\varphi_i, x, k)$, minBO instead keeps a Gaussian Process model of each conjunctive component $\varphi_1, \ldots, \varphi_n$. An overview of minBO is shown in Algorithm 4.

Here, $h_i(\mathbf{x})$ is the robustness associated to the i^{th} conjunctive component φ_i. The algorithm first samples b_0 locations in a Latin Hypercube sample, a sampling technique that helps ensure that the set of sampled numbers is representable of the real variability. After this, each iteration of minBO includes n inner BO iteration where GPs are estimated for each of the conjunctive components. Each sample is used to build up every GP so that there are n models $\left(\hat{h}_i(\mathbf{x}), \hat{s}_i^2(\mathbf{x})\right)$. To find out the next location to be sampled, the Expected Improvement (EI) is maximized for each component; the EI is defined here as

$$EI_i(\mathbf{x}) = \mathbb{E}\left[\max\left([y^* - \hat{h}_i(\mathbf{x})]\Phi\left(\frac{y^* - \hat{h}_i(\mathbf{x})}{\hat{s}_i(\mathbf{x})}\right) + \hat{s}_i(\mathbf{x})\phi\left(\frac{y^* - \hat{h}_i(\mathbf{x})}{\hat{s}_i(\mathbf{x})}\right), 0\right)\right], \quad (3.12)$$

where Φ is the Cumulative Distribution Function (CDF) and ϕ is the Probability Density Function (PDF) of the normal distribution. The next point to be sampled is chosen as the point with the highest EI from all of the n models, and the next iteration starts with re-estimating the GPs.

Algorithm 4 minBO: Minimum Bayesian Optimization [97]

```
1: function minBO(Domain $\mathbb{X} \in \mathbb{R}^d, n$ components $\{h_1(\mathbf{x}), \dots, h_n(\mathbf{x})$, objective function $f(\mathbf{x}) = \min(h_1(\mathbf{x}), \dots, h_n(\mathbf{x}))$, initialization budget $b_0$, total budget $T$)
2:     Create initializing Latin Hypercube design $\mathbf{x}_{train}$ with $b_0$ locations from $\mathbb{X}, \mathbf{x}_{train} \in \mathbb{R}^{b_0 \times d}$
3:     Sample $\mathbf{x}_{train}$ over the $n$ composite functions, set $\mathbf{y}^i_{train} = h_i(\mathbf{x}_{train})$ for $i = 1, \dots, n$
4:     Set $t \leftarrow b_0$
5:     while $t < T$ do
6:         for $i = 1, \dots, n$ do
7:             Estimate a GP using the training data $\left\{\mathbf{x}_{train}, \mathbf{y}^i_{train}\right\}$, resulting in $\left(\hat{h}_i(\mathbf{x}), \hat{s}^2_i(\mathbf{x})\right)$ for all $\mathbf{x} \in \mathbb{X}$
8:             $\mathbf{x}^i_{EI} \leftarrow \arg\max_{\mathbf{x} \in \mathbb{X}} EI_i(\mathbf{x})$
9:         end for
10:        $\mathbf{x}^*_{EI} \leftarrow \arg\max_{i=1,\dots,n} \mathbf{x}^i_{EI}$, append $\mathbf{x}^*_{EI}$ to $\mathbf{x}_{train}$
11:        Sample and append $h_i(\mathbf{x}^*_{EI})$ to $\mathbf{y}^i_{train}$, for $i = 1, \dots, n$
12:        $t \leftarrow t + 1$
13:    end while
14: end function
15: return Best observed location and value $\mathbf{x}^*_{minBO}, f(\mathbf{x}^*_{minBO})$
```

3.7 Large-scale falsification

To apply testing to large-scale systems, such as the ones at Volvo Car Corporation considered in several of the appended papers, one must make modifications to the problem formulation and how the problem is solved. The following list specifies some characteristics of the testing problem that relevant tools should handle to be applied to industrial models. The list is based on experiences from the models at Volvo Car Corporation that have been the main subject of the research project.

1. The models can be fully or partially black-box, meaning that we do not have access to all source code in the SUT. For example, code developed by third parties can be included as pre-compiled binaries, so tools that rely on analyzing, e.g., controller code in the model will not work in the general case.

2. The models are of considerable size. As an example, the Volvo models discussed in Paper A include 19846 and 18294 blocks, respectively (a block is a basic modeling construct of Simulink [14]). This means that manual analysis of the test results is likely too complex to be carried out by, e.g., a single engineer.

3. The models can take a long time to simulate. In this context, a simulation time of the order of one minute is considered long (compared to several of the public benchmark examples which take fractions of a second to be simulated).

4. There can be a lot of specifications defined for the models. For example, the models in Paper A have 58 and 36 specifications, respectively.

5. The specifications are typically not defined in STL, or any other similar logic language that is typically used in falsification. The reason why we want to use STL is that the quantitative semantics of STL act as a natural objective function for an optimization problem formualtion.

The rest of the chapter is dedicated to further discussion on these issues, as well as what has been done during this research project to be able to apply the falsification tool Breach [72] to different models at Volvo Car Corporation.

Size and complexity of the system under test

Because the models considered are either fully or partially black-box, the problem formulation in this thesis only assumes that we can supply inputs to the system, simulate it, and then observe its outputs. This fact, coupled with the fact that the models take a long time to simulate, means that the optimization formulation of the falsification problem requires a gradient-free solver that is efficient with respect to the number of simulations needed to find a counter-example. To find the most suitable approach relative to these issues, several of the papers appended to this thesis include different optimization solvers when evaluating falsification performance.

Another aspect of the system complexity is that there is typically a large number of input and output signals, where many of them are discrete, e.g., Boolean signals or enumerations. Robustness of temporal logic specifications is in its nature continuous, meaning that the more discrete signals there are in the specification, the less likely are we to get meaningful robustness values to use during the optimization.

As an example, the robustness of the specification $\varphi_1 := (x > 10)$ has an intuitive meaning when x is a continuous signal: the robustness ρ is $x - 10$ and will vary continuously in value similar to how x does. However, if x is for example an enumeration, we cannot possibly know whether the enumeration value 11 is closer to 10 than 12. This means that in practice, ρ can only ever achieve two possible values for the robustness when x is discrete-valued: a specific positive value when the specification is fulfilled, and a specific negative value when it is not (these values can be chosen arbitrarily).

One way to work around the problem that certain specifications do not necessarily have detailed robustness is to instead consider *temporal* vicinity of the robustness. As an example, if we consider the specification $\varphi_2 := \Diamond_{[0,5]}(x > 10)$ with x discrete-valued, falsification of φ_2 would mean finding a simulation where x is always 10 or lower for the first 5 seconds of simulation. Assume we have two simulations with resulting values of x as shown in Figure 3.4. Even though both of the simulations have x varying between values 9 and 11, the difference in duration at value 9 could suggest that the second simulation is closer to falsifying φ_2.

A generalization of this idea is included in Paper A with additive semantics for Valued Booleans. Where the standard (or *max*) semantics for the robustness of temporal logic specifications would give the same robustness value for

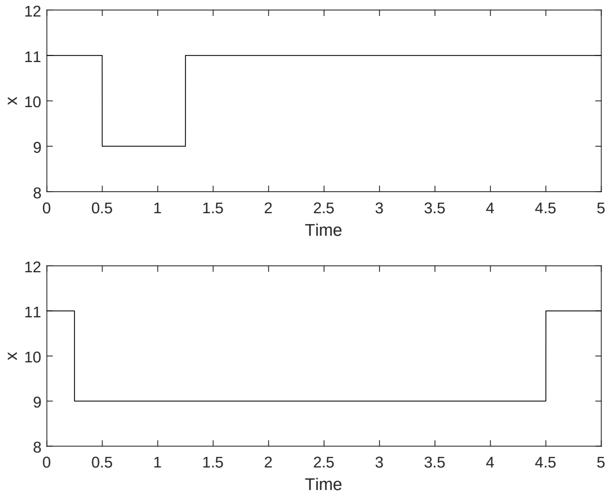


Figure 3.4: An example of two simulations including the discrete-valued signal x. Note that even though both simulations have x varying between 9 and 11, they are not the same and intuitively one could consider the lower graph to be closer to falsifying the specification $\varphi_2 := \Diamond_[0, 5](x > 10)$.

both simulations in the example, additive semantics would give different values based on how long the specifications are fulfilled and not fulfilled, respectively.

Large number of specifications

The falsification problem is usually defined as in Section 3.4, i.e., the goal is to falsify *one* specification for a given system. However, when there are many specifications for the system, one can imagine that there might exist a more efficient procedure than falsifying each specification one by one. This is the exact idea behind the Multi-Requirement Falsification algorithm presented in Paper D, where all specifications of the system under test are considered for falsification at the same time.

One limiting factor of publicly available benchmarks is that they typically do not contain as many specifications as the systems observed at Volvo. To alleviate this issue, Paper C presents a benchmark of specifications inspired by industrial systems.

Expression of the specifications

It is a well-known fact that it is difficult to write formal specifications, even for experts [55], [56]. For complex industrial systems, one typically has to break down high-level requirements stated in natural language to get specifications that can be evaluated against simulation traces.

There have been many approaches [98]–[100] to directly translate natural language requirements into, e.g., temporal logic specifications. In the industrial models considered in this thesis, however, the specifications have already been stated in other forms than just natural language. For example, Paper A introduces a method for generating Signal Temporal Logic specifications from requirements implemented directly in Simulink, which enabled falsification for several Volvo models even though the requirements were not in the correct format for falsification to start with.

CHAPTER 4

Summary of included papers and Contributions

This chapter briefly summarizes the appended papers. Like in the earlier list of papers, the papers are not presented in chronological order; instead, the order is chosen to make the contributions as easy as possible to understand for the reader. Figure 4.1 illustrates how the papers relate to each other.

Paper A includes specification transformation and a discussion on the application of Valued Booleans for Cyber-Physical Systems (CPSs). Valued Booleans (vBools) are defined and applied in an evaluation of different optimization solvers and quantitative semantics in Paper B, while the specification transformation framework is used to produce the industrial benchmark of STL specifications in Paper C. Paper D introduces Multi-Requirement Falsification, which is evaluated on the benchmark problems presented in Paper C, and which uses solver and semantics inspired by the results in Paper B. Paper E is not strictly related to the other papers; instead, it discusses mode coverage and how this can be used to conclude whether generated test suites have tested the System Under Test (SUT) thoroughly or not.

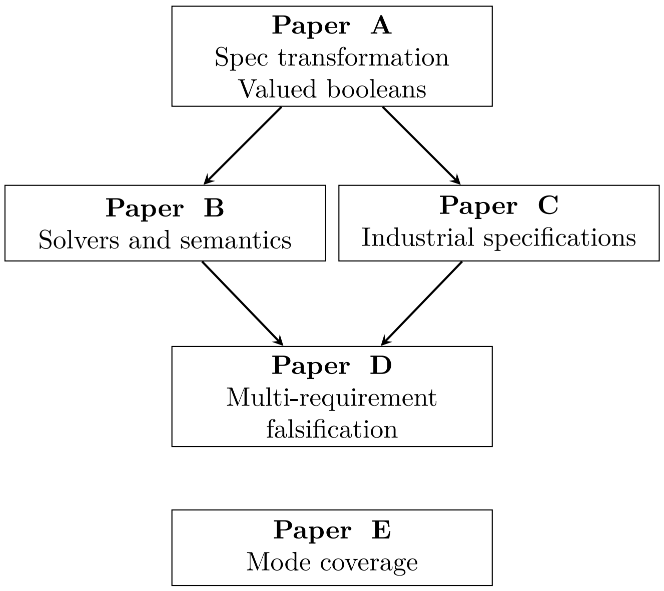


Figure 4.1: Illustration of the relations between the appended papers. An arrow originating from a paper means that an idea or concept from that paper was used in some way in the target paper.

4.1 Paper A

J. Lidén Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi and K. Åkesson
Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans
IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, vol. 39, no. 12, pp. 5247-5260, 2020.
©IEEE DOI: 10.1109/TCAD.2020.2966480

This paper has two main parts. The first part is about automatic transformation from a signal-based framework, in this case Simulink, into STL specifications. This makes it possible for engineers without knowledge of STL to use a tool for optimization-based falsification, and the transformed specifications contain more information compared to a specification written directly in STL. The most notable challenges in transforming specifications in a signal-based context into STL specifications are shown below.

1. The signal-based framework does not differentiate between *signals* and *formulas.*

2. There might be recursive loops in the specifications.

3. We cannot prove that the transformation is correct due to the non-standard semantics of Simulink.

At the end of the paper, some statistics of STL formulas a presented – these 94 formulas have been automatically transformed from the signal-based specifications at Volvo Cars.

The second part of the paper discusses the framework of vBools, which has been defined in earlier works, and is used in the falsification framework to allow the definition of different objective functions (most notably using the *max* semantics and *additive* semantics). A comparison of different objective functions (or robustness semantics) is shown for several benchmark examples: an automatic transmission model, an abstract fuel control model, and a third-order $\Delta - \Sigma$ modulator model. The conclusion is that which semantics will perform best during falsification is heavily dependent on both the system and the specification.

4.2 Paper B

J. Lidén Eddeland, S. Miremadi and K. Åkesson
Evaluating Optimization Solvers and Robust Semantics for Simulation-Based Falsification
7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), 2020

This paper evaluates the difference of certain optimization solvers and Signal Temporal Logic robust semantics for falsification of Cyber-Physical Systems. There are four optimization solvers included in the paper: Simulated Annealing, SNOBFIT, CMA-ES, and Nelder-Mead. These are also compared to a baseline algorithm that performs Uniform Random sampling of the input parameters in the falsification problem. The falsification performance is evaluated for each solver using three different robust semantics: max semantics, additive semantics, and constant semantics. Using constant semantics is equivalent to using only Boolean semantics.

The conclusion is that the falsification performance of a certain solver-semantics combination is dependent on both the system and the specification, but it is clear that using robust semantics overall gives stronger results than using the constant semantics, i.e., only using Boolean satisfaction information in the optimization solver. The performance of using the constant semantics is dependent on specific solver heuristics, but it is still the overall worst-performing semantics.

4.3 Paper C

J. Lidén Eddeland, A. Donzé, S. Miremadi and K. Åkesson
Industrial Temporal Logic Specifications for Falsification of Cyber-Physical Systems
7th International Workshop on Applied Verification of Continuous and Hybrid Systems (ARCH20), 2020

In this paper, a new benchmark of Signal Temporal Logic specifications is proposed. The specifications included in the benchmark are complex and have a structure inspired by actual industrial use cases at Volvo Car Corporation, and the specifications are intended to be used for falsification of Cyber-Physical Systems. The complexity comes from having a large number of STL operators (up to 1562) and a high temporal depth, i.e., total depth of nested temporal operators (up to 5). Since the original specifications are proprietary, the specifications in the paper are modified and obfuscated by designing them for the Automatic Transmission model, a Simulink model commonly used for benchmarking purposes in the falsification research community.

The specification benchmark is a publicly available Git repository that can be continuously updated with new features and/or new specifications over time. The paper details how different specification parameters were tuned – different parameter tunings drastically affect how easy the specifications are to falsify. Two different sets of parameter tuning, one *base* and one *hard*, are presented. Using a simple combination of corner samples and uniform random samples, the falsification rates over 1100 simulations range from 4.1% to 34.2% for the base parameter set and 0% to 0.36% for the hard parameter set. The paper also shows detailed statistics over all 17 included specifications, including the number of operators (ranging from 7 to 1562) and average monitoring time by Breach (ranging from 0.027s to 2.60s).

4.4 Paper D

J. Lidén Eddeland, A. Donzé and K. Åkesson
Multi-Requirement Testing Using Focused Falsification
Submitted for possible journal publication, 2022.
This is an extended version of the paper *Multi-Requirement Testing Using Focused Falsification* accepted to and presented at *HSCC 2022: ACM International Conference on Hybrid Systems: Computation and Control.*

This paper presents a new problem formulation for falsification of Cyber-Physical Systems that considers a large set of requirements instead of just one specification to be falsified for the SUT. The problem formulation is chosen based on experiences from industrial models at Volvo Car Corporation, where each model typically has a large number of requirements that should be monitored.

Two algorithms are presented to solve the modified falsification problem: a baseline algorithm that uses corner sampling as well as uniform random sampling, and a three-phase algorithm called Multi-Requirement Falsification (MRF) which uses corner sampling, sensitivity analysis, and focused falsification as its main parts. A corner is a point in the input parameter space where each parameter has taken either its minimum or its maximum value. The second phase of MRF uses one-factor global sensitivity analysis to approximate which input parameters affect the robustness values of which specifications. In the third phase of MRF, one specification at a time is selected to be focused on in a separate falsification problem, wherein only the parameters that show sensitivity are included in the optimization problem formulation.

It is concluded that the baseline corners-random algorithm is a strong contender since increasing the number of requirements does not affect its performance negatively, but overall the proposed Multi-Requirement Falsification algorithm is better at falsifying difficult requirements. Also, it is noted that the sensitivity analysis for robustness values of STL specifications in itself has been useful for large-scale systems at Volvo Car Corporation, where an illustration of the analysis results can give a good overview of how different parts of the SUT affect each other.

4.5 Paper E

J. Eddeland, J.G. Cepeda, R. Fransen, S. Miremadi, M. Fabian and K. Åkesson
Automated Mode Coverage Analysis for Cyber-Physical Systems Using Hybrid Automata
The 20th World Congress of the International Federation of Automatic Control, 2017, Toulouse, France

This paper presents a new coverage criterion, mode coverage, for testing of Cyber-Physical Systems. Mode coverage is defined based on the modes of the hybrid automaton that the SUT can be modeled as. The paper also shows how to automatically acquire the modes of the hybrid automaton, given code in an acausal modeling language. This procedure relies on Satisfiability Modulo Theories (SMT) analysis, which is used to automatically find the constraints on variables that are logically possible to fulfill.

An analysis of the mode coverage is shown for a use case of a model of a dog clutch from Volvo Car Corporation, where it can be seen that mode coverage provides the new information that some specific physical properties of the system had not been tested at all. The mode coverage is evaluated over 175 test cases, where 25 of the test cases are created manually by engineers, and the remaining 150 test cases are created automatically using the testing software Testweaver. The entire test suite has a mode coverage of 87.5%, covering 7/8 modes in total, where the manual tests on their own have 75% mode coverage and the automatic tests have 50% mode coverage.

4.6 Contributions

The main focus of the research in this thesis is the practical implementation of testing in industry. The field of testing CPSs is itself naturally close to application, but there is a need to come up with new methods that scale well enough to make them usable in complex systems. The contributions are directly connected to the research questions as follows.

Research Question 1. *How can specifications expressed in industrial modeling tools be used for optimization-based falsification of Cyber-Physical Systems?*

Contribution 1: *a method for automatic transformation of specifications from Simulink into Signal Temporal Logic, which makes it possible to use optimization-based falsification in applications where the testers modeling the specifications have no knowledge of STL or formal specifications.*

The specification transformation framework presented in Paper A is an example of a method that can be used without any specific training for engineers developing industrial models in Simulink. A limiting aspect of introducing more formal methods in industrial settings is the fact that it is expensive to teach engineers new tools or languages, for example, temporal logic. By automatically transforming the specifications and therefore enabling the use of more academic tools – such as Breach, which is used extensively throughout the papers appended in this thesis – the integration of research into industry becomes faster.

There is still a need for maintenance of the related tools, such as the transformation tool itself and its potential integration into Breach or other STL monitoring tools. As discussed in Paper A, it is important for the quality of generated STL specifications to in certain places have templates of combinations of different temporal operators in the Simulink specifications. These templates can help give more detailed robustness information about the specifications, even if the specifications with recursive loops in them can still be automatically generated and used for falsification.

Research Question 2. *How can the optimization-based falsification process be changed to require fewer simulations when considering a single requirement, in the context of large-scale industrial systems?*

Contribution 2: *evaluation of different optimization solvers and quantitative semantics for STL specifications over different public benchmark models, which enables more discussion on how to efficiently implement falsification in industry.*

To clarify, Paper A evaluates the falsification performance for several models while varying the quantitative semantics of STL, and Paper B evaluates falsification performance while varying both the quantitative semantics of STL and optimization solver used in the optimization formulation of the falsification problem. While it is difficult to segment different classes of specifications that work better for certain quantitative semantics or optimization solvers, the experiments indicate that there can be some structure of specifications and systems that give certain choices better falsification performance.

Some observations from the experiments follow.

- For most public benchmark models and specifications in the falsification research community, it seems that using quantitative semantics in an optimization problem outperforms pure random testing, given the same input parameterization. The opposite can only be seen for a specific system, the Static Switched system in Paper A, which was designed to have the gradient of the robustness point away from the area in the parameter space where the specification is falsified.
- If a specification is designed to check, e.g., a specific drive cycle, which could be expressed as a conjunction of sub-specifications that individually are easy to fulfill, additive quantitative semantics of STL can be preferable to use. This is to get more detailed information about each clause, such as the specification φ_7 in Paper A.
- As seen in the experiments in Paper B, there is no specific quantitative semantics that always performs better than the others, and there is no specific optimization solver that always performs better than the others. However, the results of the experiments in the paper, such as the cactus plot in Paper B, can serve as a rule of thumb when first choosing semantics and solver for a new specification or set of specifications to falsify for a model.

Research Question 3. *How can the optimization-based falsification process be changed to require fewer simulations when considering multiple requirements at once?*

Contribution 3: *a new framework of multi-requirement falsification, which makes it clearer how the optimization-based falsification can be applied to large-scale systems with many requirements, and also how robustness-based global sensitivity analysis can provide vital information in the process of testing said systems.*

Thanks to the benchmark of specifications inspired by Volvo Cars that were published in Paper C, there is now a new way to evaluate the performance of falsification algorithms over a large set of specifications. Using this benchmark, the MRF algorithm presented in Paper D is evaluated against a proposed baseline algorithm of corner and uniform random samples.

The problem formulation of multi-requirement falsification in itself is considered a contribution – we are interested, like in industry, to falsify as many requirements as possible in the shortest possible time. In the stated formulation, the specifications can be of two different classes, indicating whether the specifications are safety-related or coverage-related.

The parts of the MRF algorithm, all aimed at reducing the number of simulations needed when considering falsification of many requirements at once, are listed below.

- Local predicate normalization, to reduce the effects of the well-known *masking problem* in falsification.
- *Structural sensitivity analysis*, a new kind of sensitivity analysis that can be used to find relations between input parameters and robustness values of specifications.
- A method to find the relevant input parameters to include in an optimization problem, given a specification to falsify.
- Heuristics for automatically choosing specifications to focus on, while still doing random exploration of input parameters that are not considered for the focused specification.

Research Question 4. *How can we evaluate how well testing of Cyber-Physical Systems fulfills structural coverage criteria, and how can we then use these criteria to assess when testing is considered finished?*

Contribution 4: *definition of a new type of coverage criterion, mode coverage, for testing of CPSs, as well as examples of situations where said criterion can be useful and how it compares to code coverage.*

The definition of the new coverage criterion in Paper E is in itself rather intuitive: how many of the modes in the hybrid automaton defining the behavior of the CPS have been visited by the test suite? What makes the application of this coverage definition interesting is the fact that, given an implementation of the system in OpenModelica, the entire procedure to calculate the mode coverage is automatically performed with the help of an SMT solver.

While the example with the dog clutch in Paper E required an extra manual step of translating the existing Simscape code into OpenModelica, the languages have similar syntax and the translation, therefore, did not require much time. The results indicate that it can be easy to miss that even though the test suite might fulfill other coverage criteria, e.g. the commonly used MC/DC, automatic analysis of the mode coverage can highlight important issues with the test suite.

CHAPTER 5

Concluding Remarks and Future Work

The research of this thesis focuses on improving techniques for generating new test cases for Cyber-Physical Systems (CPSs), as well as analyzing already generated test cases. The research has been performed at Volvo Car Corporation with a focus on implementing state-of-the-art research methods useful for testing of the used industrial models.

Optimization-based falsification of CPSs is a growing research area that was interesting to consider for this thesis, as the approaches presented in academic papers are often on the verge of being applied to real industrial problems. As such, the focus of Papers A, B, C, and D is on pieces needed in the falsification framework for it to be properly implemented for complex and large real-world models. The main focus of the research has concerned the definition of a quantitative semantics for the logic formalism Signal Temporal Logic (STL), and what the semantics could be extended with to make it more viable for certain classes of models and systems.

Paper A and Paper B evaluate the performance of both different quantitative semantics for STL, and different optimization solvers, in the context of optimization-based falsification. No specific quantitative semantics or optimization solver can be the best when it comes to falsifying the benchmark problems in the evaluations. However, certain structures in the specifications and models indicate in the results that it can be important to make an informed choice when designing a solution method to a given falsification problem.

A framework presented in Paper A for transforming signal-based specifications into STL specifications allowed optimization-based falsification to be performed for large-scale models at Volvo Car Corporation. This has been key in helping optimization-based falsification become a more mature method to use for industrial models. The signal-based specifications were also an inspiration for the specification benchmark in Paper C, which hopefully can be used by by the falsification research community to create tools that can handle larger and more complex specifications.

The Multi-Requirement Testing algorithm presented in Paper D also indicates an important aspect when it comes to applying falsification methods to industrial models. When there is a large number of complex specifications to falsify in a limited time, it is imperative for an optimization solver to make an informed decision about how to over time shift focus between the different specifications. The algorithm presented, together with an algorithm based on a combination of input parameter corner samples and random exploration, conclude the balance in this thesis between academic value and industrial application.

Finally, Paper E presents results for analysis of previously created test cases. Specifically, the main contribution of the paper is a type of coverage inspired by the physical properties of the System Under Test. The coverage criterion is adequate for systems that are modeled in an acausal way, i.e., using equations to describe the components of the system.

5.1 Future work

The work presented in this thesis can be expanded in multiple ways. Some of the most direct and potentially interesting extensions are included below.

- While Paper A presents a framework for transforming a kind of specifications available for industrial models at the time the paper was written, it would be useful if the transformation approach could be generalized to other kinds of models, for example other tools and languages commonly used in industry. To keep applying academic concepts to industrial models, the tools need to be taken into account, and to do this in a more general way is a challenging problem.

- Further evaluations about using different optimization solvers and semantics could help future algorithms automatically choose settings that could be used for given models and specifications.

- It would be interesting to increase test generation quality by considering the coverage criterion presented in Paper E, and possibly some similar criterion. These kinds of criteria could also be used to reduce the size of the test suite.

- More research could be performed on falsification in later stages of testing, i.e., Hardware-in-the-Loop testing. Having actual hardware as a part of the system under test could create difficulties for the standard falsification approach, such as limitations on the input generation and monitoring due to safety aspects.

- Finally, input parametrization often causes practical issues when implementing falsification in an industrial setting. More research into when different input parametrizations are efficient could help tools become easier to use in practice by test engineers, which also would increase the interest in the research area of optimization-based falsification.

Naturally, the future work considered affects both academic research directions as well as industrial interest in the research subject. In the end, the main goal of my research has been to bridge the gap between academia and industry, so that efficient state-of-the-art methods can be used in the development and testing of large-scale systems.

References

[1] R. Baheti and H. Gill, “Cyber-Physical Systems,” *The impact of control technology*, vol. 12, no. 1, pp. 161–166, 2011.

[2] M. Utting and B. Legeard, *Practical model-based testing: a tools approach.* Elsevier, 2010.

[3] S. A. Asadollah, R. Inam, and H. Hansson, “A Survey on Testing for Cyber Physical System,” in *IFIP International Conference on Testing Software and Systems*, Springer, 2015, pp. 194–207.

[4] J. Kapinski, J. Deshmukh, X. Jin, H. Ito, and K. Butts, “Simulation-Guided Approaches for Verification of Automotive Powertrain Control Systems,” in *2015 American Control Conference (ACC)*, IEEE, 2015, pp. 4086–4095.

[5] E. Bartocci, J. Deshmukh, A. Donzé, G. Fainekos, O. Maler, D. Ničković, and S. Sankaranarayanan, “Specification-Based Monitoring of Cyber-Physical Systems: A Survey on Theory, Tools and Applications,” in *Lectures on Runtime Verification*, Springer, 2018, pp. 135–175.

[6] A. Corso, R. Moss, M. Koren, R. Lee, and M. Kochenderfer, “A Survey of Algorithms for Black-Box Safety Validation of Cyber-Physical Systems,” *Journal of Artificial Intelligence Research*, vol. 72, pp. 377–428, 2021.

[7] K. J. Hayhurst and D. S. Veerhusen, “A Practical Approach to Modified Condition/Decision Coverage,” in *Digital Avionics Systems, 2001. DASC. 20th Conference*, IEEE, vol. 1, 2001, 1B2–1.

[8] R. Koymans, "Specifying Real-Time Properties with Metric Temporal Logic," *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[9] O. Maler and D. Nickovic, "Monitoring Temporal Properties of Continuous Signals," in *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, Springer, 2004, pp. 152–166.

[10] A. Donzé and O. Maler, "Robust Satisfaction of Temporal Logic Over Real-Valued Signals.," in *FORMATS*, Springer, vol. 6246, 2010, pp. 92–106.

[11] H. Abbas, G. Fainekos, S. Sankaranarayanan, F. Ivančić, and A. Gupta, "Probabilistic Temporal Logic Falsification of Cyber-Physical Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 12, no. 2s, p. 95, 2013.

[12] P. M. Duvall, S. Matyas, and A. Glover, *Continuous Integration: Improving Software Quality and Reducing Risk*. Pearson Education, 2007.

[13] M. Tiller, *Introduction to Physical Modeling with Modelica*. Springer Science & Business Media, 2001.

[14] The MathWorks, Inc., Natick, Massachusetts, *Simulink R2013b*, version 0.20.2, 2017.

[15] 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 Magazine*, vol. 36, no. 6, pp. 45–64, 2016.

[16] G. Muller, "Systems Engineering Research Methods," *Procedia Computer Science*, vol. 16, pp. 1092–1101, 2013, Conference on Systems Engineering Research, ISSN: 1877-0509.

[17] P. Fritzson, P. Aronsson, A. Pop, H. Lundvall, K. Nystrom, L. Saldamli, D. Broman, and A. Sandholm, "Openmodelica - A Free Open-Source Environment for System Modeling, Simulation, and Teaching," in *2006 IEEE Conference on Computer Aided Control System Design, 2006 IEEE International Conference on Control Applications, 2006 IEEE International Symposium on Intelligent Control*, IEEE, 2006, pp. 1588–1595.

[18] A. Junghanns, J. Mauss, M. Tatar, *et al.*, “TestWeaver-A Tool for Simulation-Based Test of Mechatronic Designs,” in *6th International Modelica Conference, Bielefeld, March 3*, Citeseer, 2008.

[19] MATLAB, *9.13.0.2049777 (R2022b).* Natick, Massachusetts: The MathWorks Inc., 2022.

[20] V. Bolbot, G. Theotokatos, L. M. Bujorianu, E. Boulougouris, and D. Vassalos, “Vulnerabilities and Safety Assurance Methods in Cyber-Physical Systems: A Comprehensive Review,” *Reliability Engineering & System Safety*, vol. 182, pp. 179–193, 2019.

[21] P. Hehenberger, B. Vogel-Heuser, D. Bradley, B. Eynard, T. Tomiyama, and S. Achiche, “Design, Modelling, Simulation and Integration of Cyber Physical Systems: Methods and Applications,” *Computers in Industry*, vol. 82, pp. 273–289, 2016.

[22] R. Rajkumar, I. Lee, L. Sha, and J. Stankovic, “Cyber-Physical Systems: The Next Computing Revolution,” in *Design Automation Conference*, IEEE, 2010, pp. 731–736.

[23] E. M. Clarke, E. A. Emerson, and J. Sifakis, “Model Checking: Algorithmic Verification and Debugging,” *Communications of the ACM*, vol. 52, no. 11, pp. 74–84, 2009.

[24] C. Baier and J.-P. Katoen, *Principles of Model Checking.* MIT press, 2008.

[25] R. Hähnle and M. Huisman, “Deductive Software Verification: From Pen-and-Paper Proofs to Industrial Tools,” in *Computing and Software Science*, Springer, 2019, pp. 345–373.

[26] A. Platzer and J.-D. Quesel, “KeYmaera: A Hybrid Theorem Prover for Hybrid Systems (System Description),” in *International Joint Conference on Automated Reasoning*, Springer, 2008, pp. 171–178.

[27] T. A. Henzinger, P. W. Kopke, A. Puri, and P. Varaiya, “What’s Decidable About Hybrid Automata?” In *Proceedings of the twenty-seventh annual ACM symposium on Theory of computing*, ACM, 1995, pp. 373–382.

[28] M. Nyberg, D. Gurov, C. Lidström, A. Rasmusson, and J. Westman, "Formal Verification in Automotive Industry: Enablers and Obstacles," in *International Symposium on Leveraging Applications of Formal Methods*, Springer, 2018, pp. 139–158.

[29] ISO, "ISO/DIS 26262-1 - Road Vehicles — Functional Safety — Part 1 Glossary," Tech. Rep., 2009.

[30] J. M. Esposito, J. Kim, and V. Kumar, "Adaptive RRTs for Validating Hybrid Robotic Control Systems," in *Algorithmic Foundations of Robotics VI*, Springer, 2004, pp. 107–121.

[31] T. Dang and T. Nahhal, "Coverage-Guided Test Generation for Continuous and Hybrid Systems," *Formal Methods in System Design*, vol. 34, no. 2, pp. 183–213, 2009.

[32] 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*, IEEE Press, 2015, pp. 31–40.

[33] K. Claessen and J. Hughes, "QuickCheck: A Lightweight Tool for Random Testing of Haskell Programs," *ACM SIGPLAN Notices*, vol. 46, no. 4, pp. 53–64, 2011.

[34] G. Klees, A. Ruef, B. Cooper, S. Wei, and M. Hicks, "Evaluating Fuzz Testing," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, ACM, 2018, pp. 2123–2138.

[35] P. Godefroid, N. Klarlund, and K. Sen, "DART: Directed Automated Random Testing," in *ACM SIGPLAN Notices*, ACM, vol. 40, 2005, pp. 213–223.

[36] T. Arts, J. Hughes, J. Johansson, and U. Wiger, "Testing Telecoms Software with Quviq Quickcheck," in *Proceedings of the 2006 ACM SIGPLAN workshop on Erlang*, ACM, 2006, pp. 2–10.

[37] R. S. Sutton and A. G. Barto, *Reinforcement Learning: An Introduction.* MIT press, 2018.

[38] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing Atari with Deep Reinforcement Learning," *arXiv preprint arXiv:1312.5602*, 2013.

[39] K. Arulkumaran, M. P. Deisenroth, M. Brundage, and A. A. Bharath, “Deep Reinforcement Learning: A Brief Survey,” *IEEE Signal Processing Magazine*, vol. 34, no. 6, pp. 26–38, 2017.

[40] T. Akazaki, S. Liu, Y. Yamagata, Y. Duan, and J. Hao, “Falsification of Cyber-Physical Systems using Deep Reinforcement Learning,” in *International Symposium on Formal Methods*, Springer, 2018, pp. 456–465.

[41] X. Qin, N. Aréchiga, A. Best, and J. Deshmukh, “Automatic Testing With Reusable Adversarial Agents,” *arXiv preprint arXiv:1910.13645*, 2019.

[42] M. Koren, S. Alsaif, R. Lee, and M. J. Kochenderfer, “Adaptive Stress Testing for Autonomous Vehicles,” in *2018 IEEE Intelligent Vehicles Symposium (IV)*, IEEE, 2018, pp. 1–7.

[43] G. E. Fainekos and G. J. Pappas, “Robust Sampling for MITL Specifications,” in *International Conference on Formal Modeling and Analysis of Timed Systems*, Springer, 2007, pp. 147–162.

[44] G. Fainekos and G. Pappas, “Robustness of Temporal Logic Specifications for Continuous-Time Signals,” *Theoretical Computer Science*, vol. 410, no. 42, pp. 4262–4291, 2009.

[45] A. Pnueli, “The Temporal Logic of Programs,” in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, IEEE, 1977, pp. 46–57.

[46] V. Raman, A. Donzé, D. Sadigh, R. M. Murray, and S. A. Seshia, “Reactive Synthesis from Signal Temporal Logic Specifications,” in *Proceedings of the 18th international conference on hybrid systems: Computation and control*, ACM, 2015, pp. 239–248.

[47] A. Donzé, T. Ferrere, and O. Maler, “Efficient Robust Monitoring for STL,” in *International Conference on Computer Aided Verification*, Springer, 2013, pp. 264–279.

[48] J. V. Deshmukh, A. Donzé, S. Ghosh, X. Jin, G. Juniwal, and S. A. Seshia, “Robust Online Monitoring of Signal Temporal Logic,” *Formal Methods in System Design*, vol. 51, no. 1, pp. 5–30, 2017.

[49] L. Brim, P. Dluhoš, D. Šafránek, and T. Vejpustek, "STL*: Extending Signal Temporal Logic with Signal-Value Freezing Operator," *Information and computation*, vol. 236, pp. 52–67, 2014.

[50] A. Donzé, O. Maler, E. Bartocci, D. Nickovic, R. Grosu, and S. Smolka, "On Temporal Logic and Signal Processing," in *International Symposium on Automated Technology for Verification and Analysis*, Springer, 2012, pp. 92–106.

[51] L. V. Nguyen, J. Kapinski, X. Jin, J. V. Deshmukh, and T. T. Johnson, "Hyperproperties of Real-Valued Signals," in *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, ser. MEMOCODE '17, Vienna, Austria: ACM, 2017, pp. 104–113, ISBN: 978-1-4503-5093-8.

[52] T. Akazaki and I. Hasuo, "Time Robustness in MTL and Expressivity in Hybrid System Falsification," in *International Conference on Computer Aided Verification*, Springer, 2015, pp. 356–374.

[53] S. Jakšić, E. Bartocci, R. Grosu, T. Nguyen, and D. Ničković, "Quantitative Monitoring of STL with Edit Distance," *Formal methods in system design*, vol. 53, no. 1, pp. 83–112, 2018.

[54] A. Rodionova, E. Bartocci, D. Nickovic, and R. Grosu, "Temporal Logic as Filtering," in *Proceedings of the 19th International Conference on Hybrid Systems: Computation and Control*, ACM, 2016, pp. 11–20.

[55] 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*, IEEE, 2015, pp. 70–79.

[56] B. Hoxha, N. Mavridis, and G. Fainekos, "VISPEC: A Graphical Tool for Elicitation of MTL Requirements," in *Intelligent Robots and Systems (IROS), 2015 IEEE/RSJ International Conference on*, IEEE, 2015, pp. 3486–3492.

[57] A. Dokhanchi, B. Hoxha, and G. Fainekos, "Formal Requirement Debugging for Testing and Verification of Cyber-Physical Systems," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 17, no. 2, p. 34, 2018.

[58] J. Kapinski, X. Jin, J. Deshmukh, A. Donze, T. Yamaguchi, H. Ito, T. Kaga, S. Kobuna, and S. Seshia, "ST-Lib: A Library for Specifying and Classifying Model Behaviors," SAE Technical Paper, Tech. Rep., 2016.

[59] S. Amaran, N. Sahinidis, B. Sharda, and S. Bury, "Simulation Optimization: A Review of Algorithms and Applications," *Annals of Operations Research*, vol. 240, May 2016.

[60] C. Audet and W. Hare, *Derivative-Free and Blackbox Optimization* (Springer Series in Operations Research and Financial Engineering). Cham: Springer International Publishing, 2017.

[61] Z. Zhang, P. Arcaini, and I. Hasuo, "Constraining Cunterexamples in Hybrid System Falsification: Penalty-Based Approaches," in *NASA Formal Methods Symposium*, Springer, 2020, pp. 401–419.

[62] Z. Zhang, P. Arcaini, and I. Hasuo, "Hybrid System Falsification Under (In)equality Constraints via Search Space Transformation," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 11, pp. 3674–3685, 2020.

[63] X. Jin, A. Donzé, J. V. Deshmukh, and S. A. Seshia, "Mining Requirements from Closed-Loop Control Models," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 11, pp. 1704–1717, 2015.

[64] Y. S. R. Annapureddy and G. E. Fainekos, "Ant Colonies for Temporal Logic Falsification of Hybrid Systems," in *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, IEEE, 2010, pp. 91–96.

[65] Z. Ramezani, K. Claessen, N. Smallbone, M. Fabian, and K. Åkesson, "Testing Cyber–Physical Systems Using a Line-Search Falsification Method," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 41, no. 8, pp. 2393–2406, 2022.

[66] J. Deshmukh, X. Jin, J. Kapinski, and O. Maler, "Stochastic Local Search for Falsification of Hybrid Systems," in *International Symposium on Automated Technology for Verification and Analysis*, Springer, 2015, pp. 500–517.

[67] H. Abbas, A. Winn, G. Fainekos, and A. A. Julius, "Functional Gradient Descent Method for Metric Temporal Logic Specifications," in *American Control Conference (ACC), 2014*, IEEE, 2014, pp. 2312–2317.

[68] B. Hoxha, H. Abbas, and G. Fainekos, "Benchmarks for Temporal Logic Requirements for Automotive Systems," *Proc. of Applied Verification for Continuous and Hybrid Systems*, 2014.

[69] X. Jin, J. V. Deshmukh, J. Kapinski, K. Ueda, and K. Butts, "Powertrain Control Verification Benchmark," in *Proceedings of the 17th international conference on Hybrid systems: computation and control*, ACM, 2014, pp. 253–262.

[70] G. Ernst, P. Arcaini, I. Bennani, A. Chandratre, A. Donzé, G. Fainekos, G. Frehse, K. Gaaloul, J. Inoue, T. Khandait, *et al.*, "ARCH-COMP 2021 Category Report: Falsification with Validation of Results," *EPiC Series in Computing*, vol. 80, pp. 133–152, 2021.

[71] Y. Annpureddy, C. Liu, G. Fainekos, and S. Sankaranarayanan, "S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems," in *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, Springer, 2011, pp. 254–257.

[72] A. Donzé, "Breach, a Toolbox for Verification and Parameter Synthesis of Hybrid Systems," in *International Conference on Computer Aided Verification*, Springer, 2010, pp. 167–170.

[73] C. Menghi, S. Nejati, L. Briand, and Y. I. Parache, "Approximation-Refinement Testing of Compute-Intensive Cyber-Physical Models: An Approach Based on System Identification," in *2020 IEEE/ACM 42nd International Conference on Software Engineering (ICSE)*, IEEE, 2020, pp. 372–384.

[74] M. Waga, "Falsification of Cyber-Physical Systems with Robustness-Guided Black-Box Checking," in *Proceedings of the 23rd International Conference on Hybrid Systems: Computation and Control*, 2020, pp. 1–13.

[75] D. Peled, M. Y. Vardi, and M. Yannakakis, "Black Box Checking," in *Formal Methods for Protocol Engineering and Distributed Systems*, Springer, 1999, pp. 225–240.

[76] Y. Yamagata, S. Liu, T. Akazaki, Y. Duan, and J. Hao, “Falsification of Cyber-Physical Systems Using Deep Reinforcement Learning,” *IEEE Transactions on Software Engineering*, vol. 47, no. 12, pp. 2823–2840, 2020.

[77] G. Ernst, S. Sedwards, Z. Zhang, and I. Hasuo, “Fast Falsification of Hybrid Systems Using Probabilistically Adaptive Input,” in *International Conference on Quantitative Evaluation of Systems*, Springer, 2019, pp. 165–181.

[78] Z. Zhang, D. Lyu, P. Arcaini, L. Ma, I. Hasuo, and J. Zhao, “Effective Hybrid System Falsification Using Monte Carlo Tree Search Guided by QB-Robustness,” in *International Conference on Computer Aided Verification*, Springer, 2021, pp. 595–618.

[79] L. Mathesen, G. Pedrielli, S. H. Ng, and Z. B. Zabinsky, “Stochastic Optimization With Adaptive Restart: A Framework for Integrated Local and Global Learning,” *Journal of Global Optimization*, vol. 79, no. 1, pp. 87–110, 2021.

[80] L. Mathesen, S. Yaghoubi, G. Pedrielli, and G. Fainekos, “Falsification of Cyber-Physical Systems with Robustness Uncertainty Quantification Through Stochastic Optimization with Adaptive Restart,” *2019 IEEE 15th International Conference on Automation Science and Engineering (CASE)*, pp. 991–997, 2019.

[81] T. Dreossi, A. Donzé, and S. A. Seshia, “Compositional Falsification of Cyber-Physical Systems with Machine Learning Components,” *Journal of Automated Reasoning*, vol. 63, no. 4, pp. 1031–1053, 2019.

[82] G. Ernst, I. Hasuo, Z. Zhang, and S. Sedwards, “Time-Staging Enhancement of Hybrid System Falsification,” *arXiv preprint arXiv:1803.03866*, 2018.

[83] A. Dokhanchi, S. Yaghoubi, B. Hoxha, and G. Fainekos, “Vacuity Aware Falsification for MTL Request-Response Specifications,” in *Proceedings of the 13th IEEE Conference on Automation Science and Engineering (CASE’17)*, 2017.

[84] T. Akazaki, “Falsification of Conditional Safety Properties for Cyber-Physical Systems with Gaussian Process Regression,” in *International Conference on Runtime Verification*, Springer, 2016, pp. 439–446.

[85] A. Aerts, B. Tong Minh, M. Reza Mousavi, and M. A. Reniers, "Temporal Logic Falsification of Cyber-Physical Systems: An Input-Signal Space Optimization Approach," in *14th Workshop on Advances in Model Based Testing (A-MOST)*, 2018.

[86] W. Huyer and A. Neumaier, "SNOBFIT-Stable Noisy Optimization by Branch and Fit," *ACM Trans. Math. Softw.*, vol. 35, no. 2, pp. 9–1, 2008.

[87] L. M. Rios and N. V. Sahinidis, "Derivative-Free Optimization: A Review of Algorithms and Comparison of Software Implementations," *Journal of Global Optimization*, vol. 56, no. 3, pp. 1247–1293, 2013.

[88] N. Hansen, "The CMA Evolution Strategy: A Comparing Review," in *Towards a new evolutionary computation*, Springer, 2006, pp. 75–102.

[89] J. A. Nelder and R. Mead, "A Simplex Method for Function Minimization," *The Computer Journal*, vol. 7, no. 4, pp. 308–313, 1965.

[90] H. P. Gavin, "The Nelder-Mead Algorithm in Two Dimensions," *CEE 201L. Duke U*, 2020.

[91] B. Shahriari, K. Swersky, Z. Wang, R. P. Adams, and N. De Freitas, "Taking the Human Out of the Loop: A Review of Bayesian Optimization," *Proceedings of the IEEE*, vol. 104, no. 1, pp. 148–175, 2015.

[92] Z. Wang, F. Hutter, M. Zoghi, D. Matheson, and N. De Feitas, "Bayesian Optimization in a Billion Dimensions via Random Embeddings," *Journal of Artificial Intelligence Research*, vol. 55, pp. 361–387, 2016.

[93] J. Deshmukh, M. Horvat, X. Jin, R. Majumdar, and V. S. Prabhu, "Testing cyber-physical systems through bayesian optimization," *ACM Transactions on Embedded Computing Systems (TECS)*, vol. 16, no. 5s, pp. 1–18, 2017.

[94] D. Eriksson, M. Pearce, J. Gardner, R. D. Turner, and M. Poloczek, "Scalable Global Optimization via Local Bayesian Optimization," *Advances in Neural Information Processing Systems*, vol. 32, 2019.

[95] Z. Ramezani, K. Šehic, L. Nardi, and K. Åkesson, "Falsification of Cyber-Physical Systems using Bayesian Optimization," *arXiv preprint arXiv:2209.06735*, 2022.

[96] Z. Ramezani, "On optimization-based falsification of cyber-physical systems," Ph.D. dissertation, 2022.

[97] L. Mathesen, G. Pedrielli, and G. Fainekos, “Efficient Optimization-Based Falsification of Cyber-Physical Systems With Multiple Conjunctive Requirements,” in *2021 IEEE 17th International Conference on Automation Science and Engineering (CASE)*, IEEE, 2021, pp. 732–737.

[98] R. Nelken and N. Francez, “Automatic Translation of Natural Language System Specifications Into Temporal Logic,” in *International Conference on Computer Aided Verification*, Springer, 1996, pp. 360–371.

[99] C. B. Harris and I. G. Harris, “Generating Formal Hardware Verification Properties from Natural Language Documentation,” in *Proceedings of the 2015 IEEE 9th International Conference on Semantic Computing (IEEE ICSC 2015)*, IEEE, 2015, pp. 49–56.

[100] J. He, E. Bartocci, D. Ničković, H. Isakovic, and R. Grosu, “From English to Signal Temporal Logic,” *arXiv preprint arXiv:2109.10294*, 2021.