THESIS FOR THE DEGREE OF LICENTIATE OF ENGINEERING

Falsification of Signal-Based Specifications for Cyber-Physical Systems

with applications from the automotive domain

Johan Lidén Eddeland

CHALMERS
UNIVERSITY OF TECHNOLOGY

Department of Electrical Engineering
Chalmers University of Technology
Göteborg, Sweden, 2019

Falsification of Signal-Based Specifications for Cyber-Physical Systems with applications from the automotive domain

Johan Lidén Eddeland

This thesis has been prepared using LaTeX.

Department of Electrical Engineering
Chalmers University of Technology
SE-412 96 Göteborg, Sweden
Phone: +46 (0)31 772 1000
www.chalmers.se

Printed by Chalmers Reproservice
Göteborg, Sweden, December 2019

To Hanna

Abstract

In the development of software for modern Cyber-Physical Systems, testing is an integral part that is rightfully given a lot of attention. Testing is done on many different abstraction levels, and especially for large-scale industrial systems, it can be difficult to know when the testing should conclude and the software can be considered correct enough for making its way into production.

This thesis proposes new methods for analyzing and generating test cases as a means of being more certain that proper testing has been performed for the system under test. For analysis, the proposed approach includes automatically finding how much a given test suite has executed the physical properties of the simulated system.

For test case generation, an up-and-coming approach to find errors in Cyber-Physical Systems is *simulation-based falsification*. While falsification is suitable also for some large-scale industrial systems, sometimes there is a gap between what has been researched and what problems need to be solved to make the approach tractable in the industry. This thesis attempts to close this gap 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 thesis includes a method for automatically transforming a signal-based specification into a formal specification in temporal logic, as well as a modification to the underlying optimization problem that makes falsification more viable in an industrial setting.

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 analysis and generation of tests can be useful, given that one has enough information about the system under test. It is difficult to provide a general solution that will always work best – instead, the challenge lies in identifying which properties of the given system should be taken into account when trying to find potential errors in the system.

Keywords: Testing, Simulation-Based Verification, Falsification, Cyber-Physical Systems.

List of Publications

This thesis is based on the following publications:

[A] **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

[B] **J. Eddeland**, S. Miremadi, M. Fabian and K. Åkesson, "Objective Functions for Falsification of Signal Temporal Logic Properties in Cyber-Physical Systems". *13th IEEE Conference on Automation Science and Engineering (CASE)*, 2017, Xi'an, China.

[C] K. Claessen, N. Smallbone, **J. Eddeland**, Z. Ramezani and K. Åkesson, "Using Valued Booleans to Find Simpler Counterexamples in Random Testing of Cyber-Physical Systems". *14th Workshop on Discrete Event Systems (WODES)*, 2018, Sorrento Coast, Italy.

[D] **J. Lidén Eddeland**, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi and K. Åkesson, "Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans". Submitted for possible journal publication.

[E] **J. Lidén Eddeland** and K. Åkesson, "A Case Study of Optimization Solvers and Objective Functions for Falsification of Cyber-Physical Systems". Submitted for possible conference publication.

Acknowledgments

First of all, I would like to thank my supervisors Knut Åkesson and Sajed Miremadi for always giving me the support I need in my doctoral studies. During times when I have struggled, you have always helped me with encouraging words and fruitful discussions, all of which has contributed greatly to this thesis. Thanks also goes to my assistant supervisor Martin Fabian, who always helps me with proofreading and specifying the theoretical concepts of my research.

I also want to thank everyone I have worked with at Volvo for making it a good workplace for me. Specifically, I want to thank my first manager Isak Öberg for all the help during the start of my time as a PhD student. Andreas Andersson has always been available for good discussion on technical details, and Ulf Eliasson and Johan Alenius have helped me out with specific issues more times than I can count. Team MM and Team Red have always contributed to a great experience at work.

I am thankful to everyone at Chalmers who help make it a good working environment as well, especially the Automation group and the other industrial PhD students whom I share my office with. I am also very grateful to Koen Classen and Nicholas Smallbone for taking their time to discuss interesting concepts with me. I would also like to show my deep appreciation to Alexandre Donzé, who always helps me to solve different problems even though my questions are not always well-posed.

This research has been performed as part of Volvo Cars Industrial PhD Program (VIPP). The 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. I gratefully acknowledge this support.

Finally, I want to thank my family who have endured me from the very beginning. I express my gratitude to my parents and brothers for their continuous love and support. The one who remains to be mentioned is the one who I could not live without: my wife Hanna. Coming home from work every day to you and our sons Vidar and Sixten makes me the happiest person in the world. Thank you for your endless patience and love, I will try to match it for the rest of our lives!

Acronyms

CI:	Continuous Integration
CPS:	Cyber-Physical System
MBT:	Model-Based Testing
MC/DC:	Modified Condition/Decision Coverage
SMT:	Satisfiability Modulo Theories
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 modern software is no exception, which means that we need to systematize how to catch the errors 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 testing of Cyber-Physical Systems (CPSs) are presented, in an effort to find bugs or errors without creating much additional work effort for the engineers designing the system. A CPS is, as the name suggests, a system which consists of both *cyber* and *physical* properties – 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 abstrac-

tion. For example, a simple model of a car could simply 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 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 through space. Performing testing on models is naturally called Model-Based Testing (MBT). 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 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, which indicates that the software size is increasing approximately exponentially. This motivates introducing more automated testing as a complement to the manual testing which is usually already in place in development of modern CPSs.

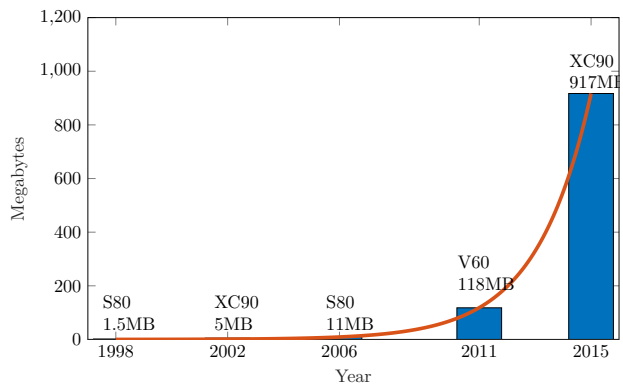

Figure 1.1: A bar chart of the downloadable software size in certain Volvo car models during the years 1998 - 2015.

To test the models that are being developed, one must define what needs to be tested. This is done by defining a *specification*, *i.e.*, the desired behaviour of the system. A specification can be written in natural language, *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 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 absence of errors 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 (for example in a *test suite*, a collection of test cases). There are many types of coverage 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 structure of the code that has actually been tested.

Another approach to generate new test cases is via optimization-based testing (or *falsification*) of CPSs. This approach formulates the problem of generating test cases as an optimization problem, where the objective function measures how far a formal specification is from being falsified, *i.e.*, not being fulfilled. To be clear, whenever falsification is mentioned in this thesis, it refers to the specific optimization-based method of finding counterexamples to temporal logic specifications of CPSs.

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 academical 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 academical methods to be suitable for industrial models as well as academical 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), which is detailed later in this section.

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 model language, for example Modelica or Simulink). The plant, *i.e.,* the physical part of the system which 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 of the plant is still being 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 there are certain aspects that can only be tested by actual driving of 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 verify that the component works as expected by itself. When several software components are created, the next step is for them to be connected to each other 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 1.2 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.

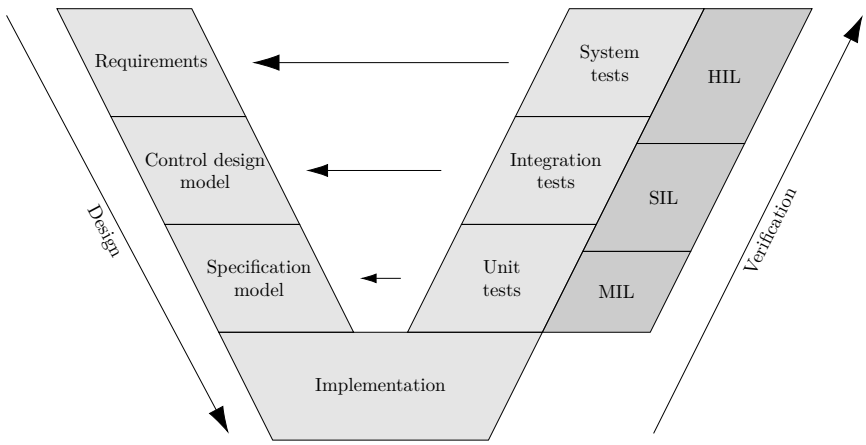

Figure 1.2: An illustration of how MIL, SIL, and HIL testing can be related to unit, integration, and system testing in the V model of software development. Even though each of the testing levels come sequentially in the testing process, there is not a 1:1 correspondence between (for example) MIL/Unit, SIL/Integration, or HIL/System.

Continuous integration

A common way to incorporate testing in industry is to use Continuous Integration (CI). CI is the practice of automatically merging developed code often, in order to more frequently find smaller faults rather than having to fix large errors with many potentially complex causes at larger time intervals. An overview of a typical CI workflow is shown in Figure 1.3.

A sketch of the desired effect of CI, in terms of time spent finding and correcting errors in the developed software, can be seen in Figure 1.4. It is clear that in the ideal case, it is easier to find bugs when there are few of them and there are not many different versions of the software to check against. However, implementing CI also requires 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.

1.2 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 A) and random testing (needed for Paper C and Paper D).

Chapter 3 is about optimization-based testing 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 B, D, and E.

Chapter 4 first includes a summary of related and recent work in related research areas. It also contains the research questions, the methodology, and the main contributions of the thesis.

Chapter 5 summarizes the content of the appended papers.

Chapter 6 contains a conclusion of the work presented in the thesis, and also an outlook on the future work to be done.

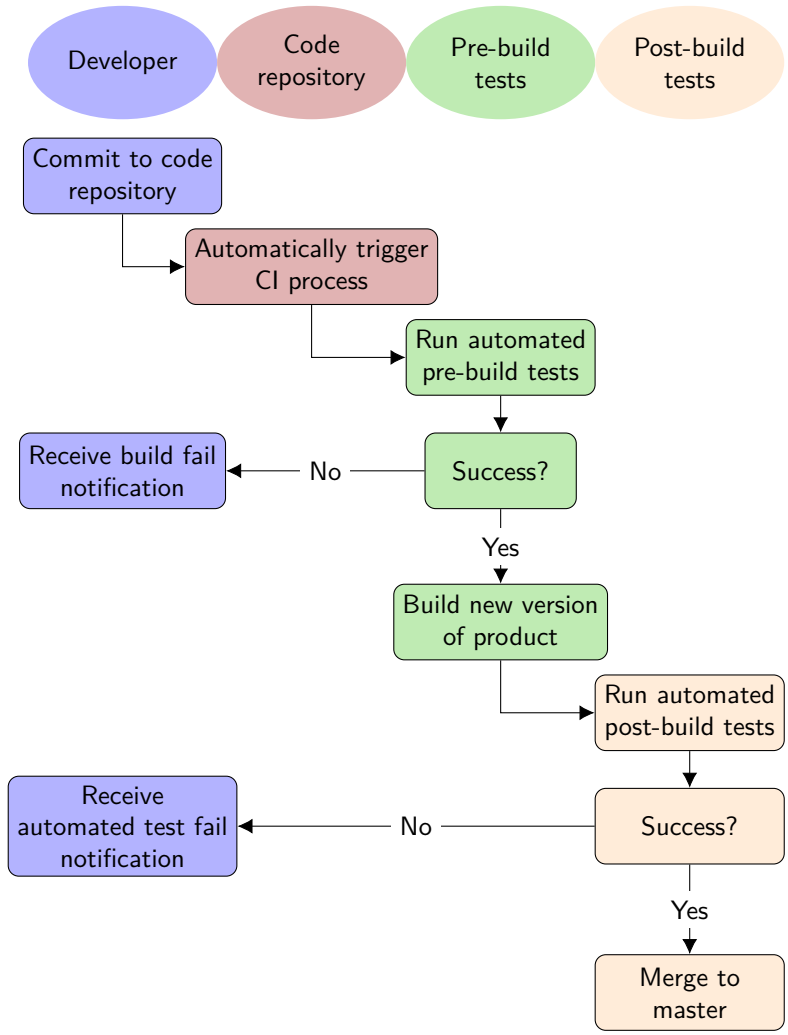

Figure 1.3: A flowchart including typical elements of continuous integration (CI). When a developer pushes their code, the code needs to be built and pass both unit tests and other automated tests before being pushed to the master branch. If the code does not build, or if it fails any tests, the developer 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.

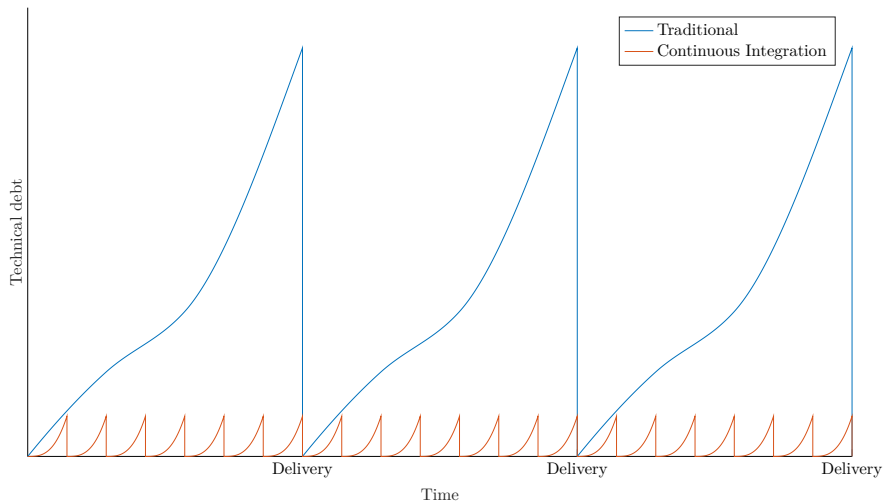

Figure 1.4: A sketch of intended technical debt over time using traditional development methods versus continuous integration. Committing the developed code with high frequency typically also means that the faults are easier to find and less time-consuming to fix.

CHAPTER 2

Software testing

This chapter gives a short insight into what software testing is, and the different kinds of software testing that are related to this thesis. In Section 2.1, there is a brief discussion about why testing is a reasonable approach to verifying behaviour of CPSs. Section 2.2 discusses coverage criteria for testing and how they are used in industry. In Section 2.3, random testing is presented.

2.1 Model checking versus testing

An approach to verify correctness of programs is model checking [1]. Model checking is exhaustive, meaning that if there is an error in the model with regards to the specification, a model checking algorithm that finishes will find it [2]. While this sounds appealing, model checking 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 [3]. 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 model checking methods are very useful for models without the

limitations discussed here, there are several other obstacles to overcome [4] 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 bugs, but testing can still raise the 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.

2.2 Coverage criteria

Testing the inner structure of the SUT is called white-box testing, while testing the system behaviours 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, consider the simple example below.

1: **if** (a and b) or c **then**
2: $\quad x = x + 1$
3: **else**
4: $\quad x = x - 1$
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 [5]. A decision is a Boolean expression composed of *conditions* and zero or more Boolean operators, where a condition is a Boolean expression containing no Boolean operators. In the given example, "(a and b) or c" is a decision, 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 [6]. 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 possible 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 decisions outcome. Instead, an example of cases required to fulfill MC/DC are shown below (where conditions in **bold** can be shown to independently affect the decisions 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. One approach [7] uses the *star discrepancy* to measure how well-filled a set of points is. 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 start discrepancy as a guide.

In another work [8], 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.

2.3 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 [9] (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* [10], where faulty inputs are sent into a program to test its security, and *concolic testing* [11], where random inputs is 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, another 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 $\forall x\ x \geq 0$. Very 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 neighbouring 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 fail 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 *e.g.* [12]).

To summarize, random testing is a form of software testing that can be proficient at finding certain kind of bugs in code, given that the testing problem is set up properly. However, to automatically generate test cases for Cyber-Physical Systems requires several other considerations as well – something that is covered in more detail in Chapter 3.

CHAPTER 3

Optimization-based testing of cyber-physical systems

This chapter starts with a presentation of Cyber-Physical Systems in general and continues with the basics of falsification for CPSs. Falsification of CPSs is the main subject of papers B, C, D, and E. Included in this presentation is the definition of discrete-time signals, Signal Temporal Logic (STL), and the robust satisfaction of STL formulas. Finally, the whole falsification loop is summarized and explained in further detail.

3.1 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) [13]. 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 [14]. 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.

Requirements of CPSs

CPSs are typically safety-critical, meaning that a failure in 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] [15]. 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 D. However, as real industrial requirements are proprietary, only a discussion on hypothetical requirements can be included in this thesis. Consider therefore say that we have the following requirement on 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 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 components. 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 of refining the requirement will be performed; an overview of how the different requirements relate to each other are shown in Figure 3.1. The first refinement follows in Requirement 2:

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 comfortable temperature quickly after start, and

[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 academical contexts.

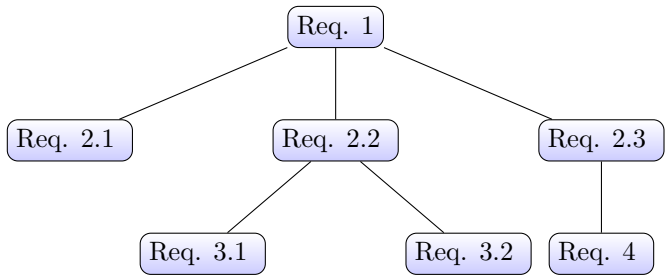

Figure 3.1: A tree describing how different example requirements are related. Each node is a refinement of its parent.

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, in order to keep the example short enough for presentation. The refinement will continue only for the requirements which can be considered software-related, as those are the ones that could be used as specifications in a simulation-based testing environment (such as the testing environment used in most of this thesis).

Clearly, Requirement 2.1 is not software-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 software-related. In order 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* $70°$ *and the current speed is lower than* 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* $200km/h$*, the behaviour 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 more clear it is which signals of the system 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 D. For example, for Requirement 4, one precondition would be that the angle of the accelerator pedal is larger than 70° *and* that the current speed is lower than 200 km/h.

3.2 Discrete-time signals

Falsification relies on monitoring of signals with respect to specifications written in temporal logic. As such, falsification in literature is usually defined in relation to continuous signals [16]. However, in Paper C, there is much discussion about semantics of logic for discrete-time signals, which is why the discrete-time presentation is chosen in this chapter as well. It is possible but not trivial to generalize these definitions to continuous time [17], [18].

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 *labels the time instants of the signal, and the signal takes on continuous values at each of those time instants.*

3.3 Signal temporal logic

Signal Temporal Logic (STL) [19] is an extension of Linear Temporal Logic (LTL) [20], with the additions 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

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

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 [21], the validity of a formula φ with respect to the discrete-time signal x at time instant k is defined as

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

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`	$(x < 0) \vee (x > 10)$
$\square_{[a,b]}$	Timed `always`	$\square_{[0,5]}(x > 20)$
$\Diamond_{[a,b]}$	Timed `eventually`	$\Diamond_{[0,4]}(x = 5)$
$\mathcal{U}_{[a,b]}$	Timed `until`	$(x = 1) \, \mathcal{U}_{[0,5]} \ (y > 10)$

How to interpret these examples is shown in Table 3.2.

Robust satisfaction of STL formulas

One advantage of writing specifications in STL is that there is a notion of robustness defined for them. The robustness of an STL specification is in-

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 0 and 5, x is greater than 20.
$\Diamond_{[0,4]}(x = 5)$:	Between 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 $\bar{t}$ between 0 and 5, y is larger than 10. For all times before this time, *i.e.* for times between 0 and $\bar{t}$, x is equal to 1.

formally how "far away" the specification is from the point of failing. 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 earlier works [21]:

$$\rho(\mu, x, k) = \mu(x[k]) \tag{3.1}$$

$$\rho(\neg\mu, x, k) = -\mu(x[k])) \tag{3.2}$$

$$\rho(\varphi \wedge \psi, x, k) = \min(\rho(\varphi, x, k), \rho(\psi, x, k)) \tag{3.3}$$

$$\rho(\varphi \vee \psi, x, k) = \max(\rho(\varphi, x, k), \rho(\psi, x, k)) \tag{3.4}$$

$$\rho(\Box_{[a,b]}\varphi, x, k) = \min_{k' \in [k+a, k+b]} \rho(\varphi, x, k') \tag{3.5}$$

$$\rho(\Diamond_{[a,b]}\varphi, x, k) = \max_{k' \in [k+a, k+b]} \rho(\varphi, x, k') \tag{3.6}$$

$$\rho(\varphi\ \mathcal{U}_{[a,b]}\psi, x, k) = \max_{k' \in [k+a, k+b]} (\min(\rho(\psi, x, k'), \min_{k'' \in [k, k']} \rho(\varphi, x, k''))) \tag{3.7}$$

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 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 will be a scalar.

To summarize, STL formulas can be used to specify desired behaviours of CPSs, and the robustness of STL formulas are 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.

3.4 Falsification

Falsification of CPSs uses optimization for generating new test cases. The reason STL is introduced in Section 3.3 is because the robustness of STL formulas is used as the objective function for the optimization. An overview of the falsification procedure is shown in Figure 3.2.

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 it is 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 [22]:

The space of permissible input signals is parametrized by m *input parameters* $\mathbf{a} = (a_1, \ldots, a_m)$ that take values from a set $\mathcal{P}_u$. The actual input $\mathbf{u}[k]$

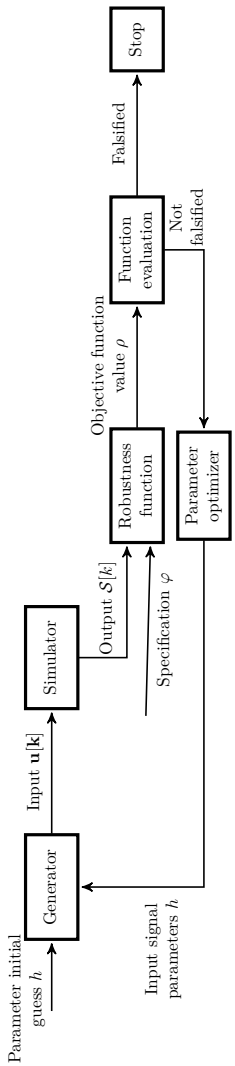

Figure 3.2: A flowchart describing the main falsification procedure.

is created using a generator function g such that $\mathbf{u}[k] = g(v(\mathbf{a}))[k]$, where $v(\mathbf{a}) \in \mathcal{P}_u$ is a valuation of the parameter vector $\mathbf{a}$. The output signal is calculated by the *system* $\mathcal{S}$.

The optimization problem is now formulated as

$$\underset{v(\mathbf{a})\in\mathcal{P}_u}{\text{minimize}} \quad \rho(\varphi,\ \mathcal{S}(g(v(\mathbf{a}))), 0) \tag{3.8}$$

where the initial guess is called $v_i(\mathbf{a})$.

Input generators

The question is how to define suitable $\mathbf{a}$ 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 sake of the optimization problem, it is preferable to use as few parameters as possible, as each parameter increases the dimension of the search space in the minimization problem. 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 actually would falsify the specification. Examples of input generators are shown in Section 3.5.

Robustness function

The standard robustness function ρ presented earlier is the one used by the two biggest falsification tools – the MATLAB toolboxes Breach [23] and S-TaLiRo [24]. As a negative robustness value means that the specification is falsified, it is easy to then check whether the algorithm should terminate or try to find other parameters that results in a robustness value lower than the current one.

In Paper B and Paper C, alternative robust semantics are presented. These are more expressive in some cases, and will make the falsification procedure end up with different results than if the "standard" STL robust semantics are used. It is, however, not possible to decide which robust 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 D and Paper E.

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 [25], Simulated Annealing [26], and Local Stochastic Tabu search [27], 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 [28]. This is not something considered in this thesis as the systems considered are black-box and of considerable size.

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 [29], and it is also used for evaluation purposes in Paper D and Paper E. The model has two inputs; the first input is the throttle 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 7 control points distributed evenly in time. Between these control points, each of which lie in the interval $[0, 100]$, the throttle values are interpolated using a standard MATLAB function.

An example of three generated input scenarios are shown in Figure 3.3.

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 3.3, 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

$$\rho(\varphi, x, 0) = \max_{k' \in [0,20]}(\omega[k'] - 2000). \tag{3.9}$$

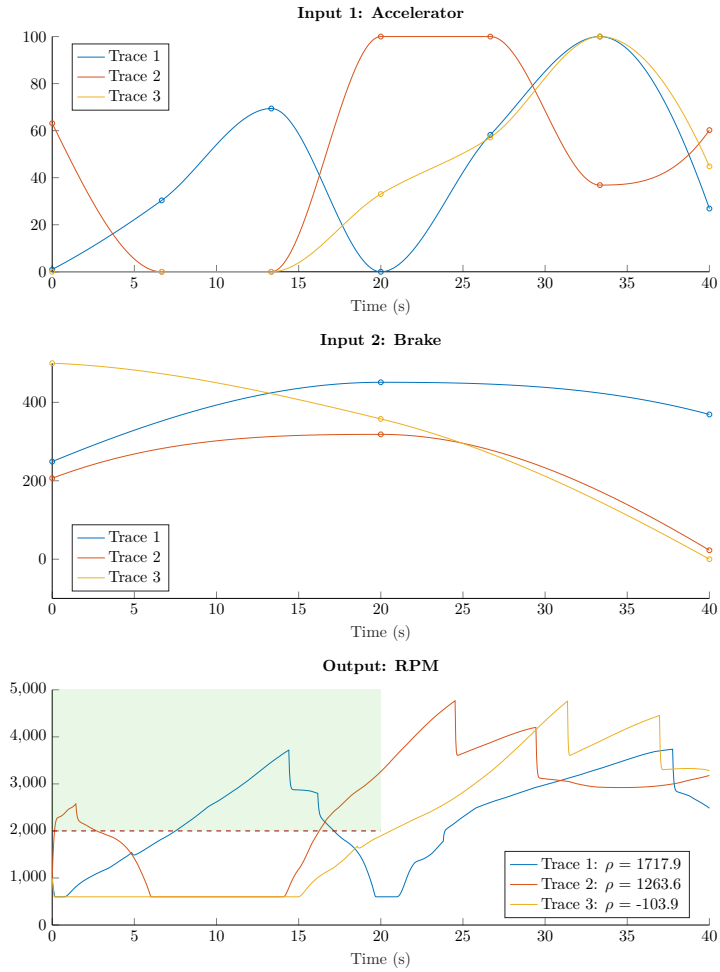

Figure 3.3: An example of three generated input scenarios for falsification of an automatic transmission system of a vehicle. The top figure shows input 1, which is parametrized by 7 control points spread evenly in time. The middle figure shows input 2, which is parametrized by 3 control points spread 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.

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

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 maximum value 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 to the specification.

CHAPTER 4

Related Work and Research Questions

This chapter first contains recent work published in fields that are relevant to this thesis in Signal Temporal Logic and Metric Temporal Logic, as well as in falsification. Following that are the research questions that are the basis of the research performed leading up to the thesis. The method and contributions of the papers appended to the thesis are also included.

4.1 Related work

Testing of CPSs is a wide and growing research area. For the interested, three recent surveys [30]–[32] explore different aspects of testing of CPSs, including different simulation-based approaches as well as monitoring of specifications.

Signal Temporal Logic and Metric Temporal Logic

Signal Temporal Logic (STL) was originally introduced [19] as an extension of Metric Temporal Logic (MTL) [33] with real-valued signals, while MTL itself is an extension of Linear Temporal Logic (LTL) [20]. One of the main focus areas of STL formulas is their quantitative semantics [18] and the efficient

monitoring of such semantics [34]. An extension to this is the ability for online monitoring of STL formulas [35], which for example enables the evaluation of specification fulfillment for partial traces of simulated systems.

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* [36] includes an additional *freezing* operator which makes it possible to specify damped oscillations in a signal, Time-Frequency Logic (TFL) [37] can be used to specify frequency-domain properties in addition to time-domain, and hyperproperties [38] are properties that require two or more execution traces to be evaluated.

A modified version of STL, avSTL [39], 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 robust semantics, which is shown to be preferential in certain systems and specifications with an application to falsification. In somewhat similar fashion, another change to the robust semantics of STL is "edit distance" [40], which is a metric that measures the distance in both time and space of an STL specification. In a related publication [41], 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 [42]. To alleviate the process of using formal specifications, approaches in earlier works have included tools that make is easier to write specifications [43], tools to automatically detect faulty specifications [44], as well as defining template specifications that make it easier for inexperienced testers to formulate the specifications [45].

Falsification of Cyber-Physical Systems

Falsification of CPSs can be performed in different ways, for example as simulation-based, optimization-based, or temporal logic-based, where the key is that falsification typically consists of all three approaches as part of the whole process. Notable falsification tools are S-TaLiro [24] and Breach [46]. An application of falsification is parameter mining of temporal requirements from closed-loop models [22], 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 [47] and falsification using

deep reinforcement learning [48]. Other approaches includes time staging [49], which splits the input generation into temporal parts, and specific procedures for better falsification of request-response specifications [50], [51]. 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 [52].

To evaluate different forms of falsification against each other, different benchmarks [29], [53] of specifications and systems are typically used. Recently, a set of standardized benchmarks have been proposed as part of an annual friendly workshop competition in falsification [54].

4.2 Research questions

The goal of the research performed leading up to this thesis can be summarized in three research questions, all connected to Model-Based Testing of Cyber-Physical Systems. The research questions did not originally all exists 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.

1. *How do optimization-based testing methods compare to random testing in finding software faults for Cyber-Physical Systems?*

This is an important question that needs to be answered in order to know if optimization-based methods are worthwhile to pursue, or if the effort should be spent on simple randomized testing instead. While there will never be a complete answer to this question (as there are infintely many systems to test), the empirical data presented in Paper E indicate that optimization-based methods outperforms random testing for several variations of solvers, specifications and systems.

While it is definitely interesting to compare optimization-based testing and random testing, it is also interesting to find out how optimization-based testing performs for real models in industry. This prompts the next research question.

2. *What is unique about industrial models in the context of optimization-based testing? How can the testing methods be adjusted for these characteristics?*

As the research area of optimization-based testing is quite close to industrial applications, the practical aspects are deemed important. In Paper B, the point was to illuminate some shortcomings in the well-established procedure of falsification of CPSs, specifically when there are discrete-valued signals present in the system (and the specification). These discrete-valued signals correspond to switching behaviour in the systems, which can be difficult to detect efficiently using falsification with the standard robust semantics presented in earlier works. To combat this issue, new objective functions are defined for a certain class of specifications, safety specifications, and it is concluded that for some systems, the new objective functions give better performance than the previously defined ones. In Paper C, Paper D, and Paper E, another more general notion of new objective functions is used, which can be applied to any class of specification. Paper E also highlights two important choices when performing falsification for real systems – the choice of optimization solver, as well as the choice of robust semantics for the STL specifications.

The methods presented in the research area of falsification can also be difficult to apply to real-world models because of the need of theoretical knowledge about the process. Specifically, specifications need to be stated in STL, something that a typical engineer in industry cannot accomplish. This leads us to wanting to investigate the next research question.

3. *How can specifications expressed as Simulink block diagrams already in place in industry be used for optimization-based falsification of Cyber-Physical Systems?*

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

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

4. *What are the strengths and weaknesses of code coverage and mode coverage for models of Cyber-Physical Systems?*

Paper A 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 (which is a model at Volvo Car Corporation). The conclusion is that a mostly automatic approach can be used to analyze how thoroughly the physical behaviour 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 System Under Test 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 should then investigate the physical behaviours 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 which is just used as a check that the test suite fulfills some basic properties.

4.3 Methodology

The purpose of the research presented in this thesis is to create further understanding of the testing problem for Cyber-Physical Systems. The research has been experimental in nature, 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.

As the research area of testing of 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 indus-

trial 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. As the industrial models themselves are proprietary, they cannot be published, however the research area is still interested in the results of applying methods to these proprietary models.

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 A, 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.

In a similar fashion, 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 B, C, D, and E.

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 A, the experiments were performed using OpenModelica and the programming language Python, with test cases previously created by TestWeaver. The experiments in all remaining papers were performed using MATLAB.

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. This means that the tools for research as well as the research itself was chosen based on the information on what the models required to be analyzed. 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 with the purpose of highlighting the contribution of the paper as clearly as possible, in order to make the paper easy to read and understand.

Analysis

As the aim of all research is 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 A, and the Volvo models discussed in papers B and D) are proprietary, exact reproducibility is impossible and the scientific integrity of the author has to be trusted. However, 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 the method of falsification as a means of attempting to verify behaviour for simulations of CPSs has the drawback of never guaranteeing any fulfillment of any property, but it is widely accepted as a reasonable verification strategy since it scales well for complex systems.

4.4 Contributions

As the reader might have guessed from the research questions, the main focus of the research behind this thesis is practical implementation in industry. The field of testing CPSs is itself naturally close to application, but there is always 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.

Contribution 1. *Comparison of optimization-based falsification and randomized testing with a similar setup, finding out whether it is useful to pursue optimization-based falsification with different robust semantics for a set of benchmark problems.*

Contribution 2. *Illustration of the shortcoming of the "standard" objective function used for falsification of CPSs, in the context of industrial applications, as well as definitions of new objective functions that attempt to resolve this issue.*

Contribution 3. *A method for automatic transformation of specifications from Simulink into Signal Temporal Logic, which makes it possible to use temporal logic-based falsification in applications where the testers modeling the specifications have no knowledge of STL or formal specifications.*

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 research contributions stem from novel approaches that have helped advance the area of testing of CPSs towards being more viable for industrial applications.

CHAPTER 5

Summary of included papers

This chapter briefly summarizes the appended papers.

5.1 Paper A

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 System Under Test can be modeled as. The paper also shows how to automatically acquire the modes of the hybrid automaton, given code in a causal 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.

5.2 Paper B

J. Eddeland, S. Miremadi, M. Fabian and K. Åkesson
Objective Functions for Falsification of Signal Temporal Logic Properties in Cyber-Physical Systems
13th IEEE Conference on Automation Science and Engineering (CASE), 2017, Xi'an, China.

This paper discusses issues of using the "standard" falsification technique for some examples of industrial models, namely models where there are discrete-valued signals present in the specification and SUT. The problem is illustrated with some simple examples and one possible solution is presented, which is to alter the objective function used in the falsification problem. Two new objective functions for safety specifications, $MARV$and $RARV$, are shown to increase falsification capabilities for three example specifications from Volvo Car Corporation. The improvement is seen by having objective function values that vary more as a function of the input parameters, even with discrete behaviour in the system which comes from having Boolean and enumerated signals in the software.

5.3 Paper C

K. Claessen, N. Smallbone, **J. Eddeland**, Z. Ramezani and K. Åkesson
Using Valued Booleans to Find Simpler Counterexamples in Random Testing of Cyber-Physical Systems
14th Workshop on Discrete Event Systems (WODES), 2018, Sorrento Coast, Italy.

In this paper, the logic of *Valued Booleans* (vBools) is introduced. A vBool is a combination of a Boolean value and a non-negative real-valued number, which indicates not only if a property is true or false, but also assigns a magnitude indicating how true or false the property is. vBools are very similar

to robust satisfaction of STL, but they come with the ability to add different robust semantics for different formulas (or parts of formulas). vBools are used here to simplify test cases in the tool Quickcheck, a technique for simplification called *shrinking*, and it is shown that for Cyber-Physical Systems which contain real-valued signals, using vBools give counterexamples that are easier to analyze compared to using only Boolean satisfaction of the properties under test.

5.4 Paper D

J. Lidén Eddeland, K. Claessen, N. Smallbone, Z. Ramezani, S. Miremadi and K. Åkesson
Enhancing Temporal Logic Falsification with Specification Transformation and Valued Booleans
Submitted for possible journal publication 2019.

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 still use a tool for falsification, and the transformed specifications actually contain some more information compared to a specification written directly in STL. The second part of the paper discusses the framework of vBools, which has been defined in Paper C and is used in the falsification framework to be able to switch between different objective functions (most notably *max* semantics and *additive* semantics). A comparison of different objective functions (or robustness semantics) is shown for several benchmark examples. The conclusion is that which semantics will perform best during falsification is heavily dependent on both the system and the specification.

5.5 Paper E

J. Lidén Eddeland and K. Åkesson
A Case Study of Optimization Solvers and Objective Functions for Falsification of Cyber-Physical Systems
Submitted to conference.

This paper contains a case study of falsification of CPSs using different

robust semantics of STL and different optimization solvers. The benchmarks used are a subset of the benchmarks used in Paper D, and the falsification setup is the same except for the fact that in this paper, several different optimization solvers are used and compared. The main purpose of the paper is to assert whether or not the results observed in Paper D also applies when considering several different optimization solvers; that is, if performance of different robust semantics in falsification depends on both the system and the specification in similar ways. The conclusion presented is that similar results apply also for different solvers, and also that for the benchmark problems presented, some solvers and strategies are typically better than others.

CHAPTER 6

Concluding Remarks and Future Work

The research of this thesis has been focused on analyzing already generated test cases, as well as improving techniques for generating new test cases for Cyber-Physical Systems. The research has been performed at Volvo Car Corporation with a focus on implementing state-of-the-art research methods to models being developed there.

As the papers appended to this thesis are given in chronological order, the reader can see how the research focus has changed over the time of the project so far. In the beginning, the aim was to look at a specific model of a dog clutch, and see how an analysis of the test result could be designed for more insight into the quality of automatically generated test cases. As there are already many approaches to evaluate test suite quality, the goal was to come up with a novel approach that had not been used before. The result was a type of coverage that is inspired by the physical properties of the SUT, which is adequate for all models that have been modeled in this acausal modeling paradigm. This summarizes one of the main contributions of the thesis.

After the work of Paper A had been performed, the focus shifted from analyzing previously created test cases, into actually generating the test cases themselves. A naturally fitting research area is the optimization-based (or

simulation-based) falsification of CPSs. This is a growing area that gets a lot of attention, and it was interesting to look at for the purpose of 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 Paper B, Paper C, Paper D, and Paper E 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 robust semantics for the logic language STL, and what the semantics could possibly be extended with in order to make it more viable for certain classes of models and systems.

The future work planned for the continuation of the project is still focused on the area of falsification for CPSs and potential areas that can be improved for its use in industrial applications. One of the directions intended is to evaluate the additive semantics presented in Paper C for models at Volvo Car Corporation (in the evaluation in Paper D and Paper E, the semantics were only evaluated for typical benchmark examples). Another direction is to instead focus on the optimization part of the falsification loop, in which we intend to improve the optimization solvers by including other robust semantics (again, most notably the additive semantics). If possible, it would be interesting to also increase test generation quality by considering the coverage criterion presented in Paper A (or possibly some similar criterion), and to construct test suite reduction based on mode coverage. Finally, we will also investigate whether we can use statistical methods to automatically find how sensitive a specification is with respect to a certain input, with the use of robustness. This could then be used to automatically adjust the parametrization of the falsification, to help reduce the dimensionality of the optimization problem.

References

[1] 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.

[2] C. Baier and J.-P. Katoen, *Principles of model checking*. MIT press, 2008.

[3] 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.

[4] 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.

[5] 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.

[6] ISO, "Iso/dis 26262-1 - road vehicles — functional safety — part 1 glossary", Tech. Rep., 2009.

[7] 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.

[8] 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.

[9] 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.

[10] 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.

[11] P. Godefroid, N. Klarlund, and K. Sen, "Dart: Directed automated random testing", in *ACM Sigplan Notices*, ACM, vol. 40, 2005, pp. 213–223.

[12] 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.

[13] 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.

[14] 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.

[15] 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.

[16] A. Donzé and O. Maler, "Robust satisfaction of temporal logic over real-valued signals.", in *FORMATS*, Springer, vol. 6246, 2010, pp. 92–106.

[17] 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.

[18] 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.

[19] 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.

[20] A. Pnueli, "The temporal logic of programs", in *Foundations of Computer Science, 1977., 18th Annual Symposium on*, IEEE, 1977, pp. 46–57.

[21] 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.

[22] 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.

[23] A. Donzé, "Breach, a toolbox for verification and parameter synthesis of hybrid systems.", in *CAV*, Springer, vol. 10, 2010, pp. 167–170.

[24] 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.

[25] 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.

[26] 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.

[27] 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.

[28] 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.

[29] 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.

[30] 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.

[31] 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.

[32] 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.

[33] R. Koymans, "Specifying real-time properties with metric temporal logic", *Real-time systems*, vol. 2, no. 4, pp. 255–299, 1990.

[34] A. Donzé, T. Ferrere, and O. Maler, "Efficient robust monitoring for stl", in *International Conference on Computer Aided Verification*, Springer, 2013, pp. 264–279.

[35] 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.

[36] 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.

[37] 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.

[38] 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.

[39] 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.

[40] 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.

[41] 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.

[42] 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.

[43] 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.

[44] 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.

[45] 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.

[46] 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.

[47] 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.

[48] 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.

[49] G. Ernst, I. Hasuo, Z. Zhang, and S. Sedwards, "Time-staging enhancement of hybrid system falsification", *arXiv preprint arXiv:1803.03866*, 2018.

[50] 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.

[51] 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.

[52] 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.

[53] 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.

[54] G. Ernst, P. Arcaini, A. Donze, G. Fainekos, L. Mathesen, G. Pedrielli, S. Yaghoubi, Y. Yamagata, and Z. Zhang, "Arch-comp 2019 category report: Falsification", *EPiC Series in Computing*, vol. 61, pp. 129–140, 2019.

Part II

Papers