FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fault-Based Analysis of Cyber Physical Systems

DISSERTATION

zur Erlangung des akademischen Grades

Doktorin der Technischen Wissenschaften

eingereicht von

Niveditha Manjunath, M.Sc.
Matrikelnummer 01646820

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ.-Prof. Dott. Ric. Ezio Bartocci
Zweitbetreuung: Dr. Dejan Ničković

Diese Dissertation haben begutachtet:

Dr. Thao Dang	Asst. Prof. Taylor T. Johnson

Wien, 2. Februar 2021

Niveditha Manjunath

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

FAKULTÄT FÜR !NFORMATIK
Faculty of Informatics

Fault-Based Analysis of Cyber Physical Systems

DISSERTATION

submitted in partial fulfillment of the requirements for the degree of

Doktorin der Technischen Wissenschaften

by

Niveditha Manjunath, M.Sc.
Registration Number 01646820

to the Faculty of Informatics

at the TU Wien

Advisor: Univ.-Prof. Dott. Ric. Ezio Bartocci
Second advisor: Dr. Dejan Ničković

The dissertation has been reviewed by:

Dr. Thao Dang	Asst. Prof. Taylor T. Johnson

Vienna, 2nd February, 2021

Niveditha Manjunath

Technische Universität Wien
A-1040 Wien ▪ Karlsplatz 13 ▪ Tel. +43-1-58801-0 ▪ www.tuwien.ac.at

Erklärung zur Verfassung der Arbeit

Niveditha Manjunath, M.Sc.
Schloßhofer Straße 20/1/4, 1210 Wien

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 2. Februar 2021

Niveditha Manjunath

To Mom and Dad

Acknowledgements

I express my sincere gratitude to my advisors Assoc. Prof. Dr. Ezio Bartocci and Dr. Dejan Ničković who transcended their roles and duties of an advisor. From mentoring me to spending countless hours in guiding me through the maze of this Ph.D. They have been extremely patient, supportive and involved in my pursuit. For creating and providing opportunities to fill my lacuna of knowledge in various subjects and expand my horizons, I cannot thank them enough.

Over the course of work described in this thesis, I have had several collaborations with partners from other universities and industry. In particular, I am thankful to Dieter Haerle, Thomas Ferrère, Fabrizio Pastore, Roderick Bloem and Benedikt Maderbacher. I feel honored to have visited Leonardo Mariani at the University of Milano-Bicocca and benefited from his many advices. I would also like to thank COST-ARVI for funding my STSM in Milan. I would like to thank the members of my proficiency evaluation committee, Ao.Univ.-Prof. Bernhard Aichernig, Prof. Andreas Steininger and Ao.Univ.Prof. Gernot Salzer for their feedback and recommendations. I greatly appreciate the effort of Dr. Thao Dang and Asst. Prof. Taylor T. Johnson in reviewing my dissertation and their advice for improving it.

I am grateful to have had the opportunity to work at Austrian Institute of Technology (AIT). I am thankful to Cristinel Mateis with whom I have had several brainstorming sessions on various topics at AIT apart from our collaborations. My stay at AIT has been a delightful one and for this I would like to thank my colleagues Stefan, Thorsten, Andreas, Rupert, Otto, Agi, Neha, Ankit, Sandra and Willibald with a special mention to Mario and Dieter who handled my never ending IT issues patiently.

I am thankful to all my friends with whom I have shared several adventures, especially Gaurav, Sachin, Gautam and Chinmay, who have helped me overcome several challenges. I would also like to thank all my teachers who have been a champion of pursuits and my uncle, Rajashekar, who kindled in me an interest towards engineering early on and has guided me ever since. I express my warmest appreciation to mother-in-law, Swarajyalakshmi, to whom the simplest of my achievements was a cause for celebration.

My deepest love to my parents, Rajalakshmi and Manjunath, to whom I shall remain eternally grateful. They have been my source of support and strength in all my endeavors, shaping me to be the individual I am today. Finally, and most importantly, my special gratitude extends to my partner in crime and husband, Abilash, for his unconditional support and efforts to keep my sanity alive. I could not have done this without you.

Kurzfassung

Cyber-Physische Systeme (CPS) bestehen aus rechnergestützten, vernetzten und verteilen Systemen, die eine physikalische Umgebung überwachen und kontrollieren. Heutzutage sind diese allgegenwärtig in etlichen sicherheitskritischen Systemen, u.a. in autonomen Fahrzeugen oder medizinischen Geräten. Deshalb ist es wichtig, die Korrektheit von CPS sicherzustellen bevor diese eingesetzt werden. Ihre Komplexität und Heterogenität macht sie anfällig für Fehler die schwer zu finden sind. Diese Fehler propagieren über Zeit und Raum und produzieren sichtbares Fehlverhalten, räumlich entfernt vom Ursprung des Fehler und sie sind typischerweise das Resultat einer komplexen Kombination von Ereignissen.

Diese Eigenschaften von Fehlern machen es schwierig sie zu erkennen, aber auch arbeitsintensiv sie zu lösen. Wir sind bestrebt Methoden zur Fehleranalyse von CPS zu entwickeln, die helfen können Fehler zur Entwurfszeit zu erkennen und zu erklären. Wir entwickeln theoretische, methodische und praktische Lösungen um dieses Ziel zu erreichen und konzentrieren uns dabei auf folgende Themen: (1) Methoden um Fehler zu lokalisieren, (2) Ansätze um Fehler zu erklären und (3) Algorithmen die Tests erzeugen die eine Abdeckung erzielen.

Die wissenschaftlichen Resultate wurden in Prototypen und Tools implementiert. Die Lösungskonzepte in dieser Doktorarbeit wurden auf öffentlich zugänglichen Benchmarks und industriellen Anwendungsfällen aus verschiedenen Bereichen evaluiert um die Anwendbarkeit unserer Forschung zu zeigen.

Abstract

Cyber-physical systems (CPS) consist of computational networked and distributed components monitoring and controlling the physical environment. They are nowadays ubiquitous in many safety-critical applications including autonomous vehicles or smart medical devices. Therefore ensuring the correctness of CPS before their deployment is essential. The complexity and heterogeneity of CPS also makes them vulnerable to faults that are challenging to detect, understand and correct. These faults can propagate in time and space, producing observable misbehavior that is hard to trace to its origin because it is typically a result of complex combination of events.

In this thesis, we develop methods for failure analysis of a CPS that help exposing and explaining faults. We develop theoretical, methodological and practical solutions to achieve this goal by focusing on the following topics: (1) methods to localize and explain failures, and (2) testing procedures that facilitate fault analysis. The scientific outcomes of this were implemented as prototypes and tools.

The solutions developed in this thesis have been evaluated on publicly available benchmarks and industrial use cases from several domains to demonstrate the applicability of our research.

Contents

Kurzfassung xi

Abstract xiii

Contents xv

Thesis Publications xvii

1 Introduction 1
- 1.1 Problem Statement and Research Gap . . . 4
- 1.2 Aim of the Thesis . . . 5
- 1.3 Methodology . . . 6
  - 1.3.1 State of the Art . . . 6
  - 1.3.2 Specification . . . 9
  - 1.3.3 Models . . . 9
  - 1.3.4 Methods . . . 9
  - 1.3.5 Platforms & Tools . . . 11
- 1.4 Contributions of the Author per Chapter . . . 11
- 1.5 Thesis Contributions . . . 12
  - 1.5.1 Fault Localization . . . 12
  - 1.5.2 Failure Explanation . . . 14
  - 1.5.3 CPSDebug . . . 17
  - 1.5.4 Adaptive Testing . . . 19
  - 1.5.5 Coverage Analysis of Tests . . . 20
- 1.6 Structure of the Thesis . . . 24
- 1.7 Bibliography . . . 25

2 Fault Localization 37

3 Failure Explanation 39

4 CPSDebug 41

5 Adaptive Testing 43

6 Coverage Analysis of Production Tests **45**

7 Conclusion and Future Work **47**
7.1 Conclusion 47
7.2 Future Work 47

List of Figures **49**

Acronyms **51**

Thesis Publications

This thesis is based on the following research papers published in international conference proceedings and journals. The authors are listed in alphabetical order in all papers except ITC'18, where the authors are listed in the order of their contribution.

- Ezio Bartocci, Thomas Ferrère, Niveditha Manjunath, Dejan Ničković: Localizing Faults in Simulink/Stateflow Models with STL, *ACM International Conference on Hybrid Systems: Computation and Control*, HSCC 2018, Porto, Portugal, April 11-13, 2018, pp 197-206
- Niveditha Manjunath, Dieter Haerle, Stephen Sabanal, Herbert Eichinger, Hermann Tauber, Andreas Machne, Christian Manthey, Mikko Väänänen, Radu Grosu, Dejan Ničković: Production Tests Coverage Analysis in the Simulation Environment, *IEEE International Test Conference*, ITC 2018, Phoenix, USA, October 29 - November 1, 2018, pp 1-7
- Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Ničković: Automatic Failure Explanation in CPS Models, *International Conference on Software Engineering and Formal Methods*, SEFM 2019, Oslo, Norway, September 18-20, 2019, pp 69-86
- Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Ničković: CPSDebug: Automatic Failure Explanation in CPS Models, *International Journal on Software Tools for Technology Transfer*, STTT 2021, pp 1-14
- Ezio Bartocci, Niveditha Manjunath, Leonardo Mariani, Cristinel Mateis, Dejan Ničković, Fabriozio Pastore: CPSDebug: A Tool for Explanation of Failures in Cyber-Physical Systems, *ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, Los Angeles/Virtual, USA, July 18-22, 2020, pp 569-572
- Ezio Bartocci, Roderick Bloem, Benedikt Maderbacher, Niveditha Manjunath, Dejan Ničković: Adaptive Testing for Specification Coverage, *Computing Research Repository (CoRR) in arXiv* and is under submission

CHAPTER 1

Introduction

Science and Engineering has created a huge impact on our society, influencing and transforming every aspect of our lives, from housing, transportation to communication. Cyber-Physical Systems (CPSs) are ubiquitous in modern technology. They are utilized substantially in several domains including ones deemed safety-critical in nature such as automotive, avionics and medicine. The arrival of emerging competitive technologies in the previous decade has created a paradigm shift in the landscape of applications for CPSs. The automotive industry is headed towards driver-less cars with Tesla leading at the helm of *Autonomous Driving* [GS20] and *Smart Mobility* has been initiated in cities like Vienna to make public transport more sustainable [Neu19]. *Smart Buildings* are improving urban development by combining smart energy systems with intelligent buildings supporting in the construction of a convenient, connected and energy efficient infrastructure [Gre15]. The fourth industrial revolution is happening today with the digital transformation of manufacturing and value creation processes [Bar15]. With improved communication and remote monitoring, it is now possible to have modular structured factories that uses Artificial Intelligence (AI) to manufacture custom-made products quite inexpensively. Robotics and Drones have been applications of CPS since their emergence. In recent times, Amazon has joined the pioneers in having autonomous robots deliver goods or packages [Vin19] and on the other hand as using Drones for surveillance and tracking [Nag20, Vol20] has been popular for a while, they have also been successful in making a reputation for themselves indoors within the warehousing and logistics industries [Pre20].

Cyber-Physical System (CPS) refer to systems constituting computationally complex devices and networked elements that interact with their physical environment using sensors and actuators. Although this is a very concise definition of CPS, in reality these systems are far more complex and require sophisticated design methods to implement intricate interactions between its components, sub-systems and its environment. These are highly collaborative, intelligent, automated systems that combine computation, communication and control [Lee15]. CPS focus on conjoining the engineering conventions of the cyber and physical worlds.

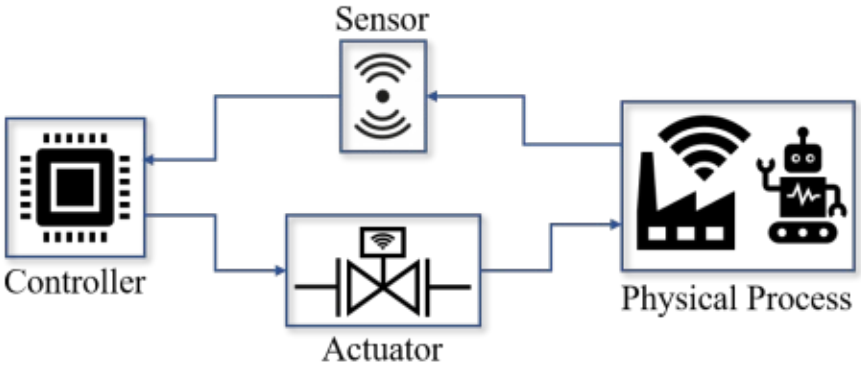

Figure 1.1: Illustration of a typical Cyber-Physical System (CPS)

As advancements in technology has simplified our lives, it has in contrast complicated the development of CPS. The design of these systems may not only involve Physics to model the behavior of the system and its interaction with the environment, but also complexity in terms of the computational devices such as controllers or processors. Depending on the application of CPS, its architecture may necessitate the system to contain distributed components.

CPSs incorporate continuous dynamics from physical systems such as the sensors or actuators and discrete dynamics from computational devices and the interaction between them is non-trivial since they use different models of computation. These systems exhibit real-time interactions with sophisticated and unpredictable environments that can involve other physical and computational elements, infrastructures and human-in-the-loop. Communication between components is essential as segments of the system may be distributed over large geographical areas. However, due to their inherent complexity, there are usually no sufficiently accurate models of environments.

Since the system relies heavily on transducing elements to observe and interact with its environment, the accuracy of measurements plays an important role. Often, there is a network of sensors and actuators in the plant. The construction of sensors, as well as the conditions of the environment can induce deviations and uncertainties in sensor measurements, resulting in an inaccurate knowledge about the environment. Depending on the type of sensors, such as radars or cameras, they are also affected by sensitivity and external factors such as light, vibration or noise. Sensors are finally prone to offset errors which are unique to every component manufactured.

Another source of complexity in CPS stems from their compositional and hierarchical nature. Frequently, the inter-connected network of heterogeneous sub-systems for a complex CPS are designed independently. The design of such composite systems require knowledge from several distinct engineering fields. With onset of paradigms involving AI, we more and more often see machine learning components being actively incorporated into systems. These components are to a large extent black box and opaque, which means one does not fully understand how they work. Machine learning components are not engineered using the classical development paradigms and they do not follow conventional approaches of design or testing. Therefore, there is no structured way to ensure their conformance to the design specification and requirements.

Development of industrial, surveillance or healthcare systems requires reliability and dependability. A failure in such systems may lead to catastrophic ramifications not only incurring significant monetary losses but also potential loss of human lives. For example, the autonomous Uber car was unable to classify the pedestrian as she was not near the sidewalk leading to the crash which was fatal for the pedestrian ([McC19]). Similarly, at the heart of the two Boeing 737 Max devastating crashes, which occurred 5 months apart, was the Manoeuvring Characteristics Augmentation System - or MCAS software that was designed to help from preventing stalling. Investigation found that this software repeatedly pushed the plane's nose down and the pilots were unable to gain back control leading to hundreds of lives lost, the planes being grounded and eventually Boeing facing severe fines and lawsuits [Leg19]. There have also been other incidents with mobility such as a pedestrian collision with an autonomous shuttle bus in Vienna where the vehicle activated the alarm and applied brake to avoid or cushion the collision within 1.6 seconds of detecting the object. The pedestrian in question got away with slight bruises [Por19]. Although failures tend to be expensive in most cases, they also serve as a motivator in improving existing technologies. The 2003 power grid blackout is one such example for emergence of smarter Power Grid technologies ([AS17]).

As systems grow in size and complexity rapidly, the challenges in its design process and hence its verification also grows in multitude. With introduction of smart or intelligent CPSs, classical engineering approaches no longer suffice. To ensure the correctness of a CPS, verification must be addressed at various levels - from hardware integrated circuits (ICs), to embedded software components and system-level. Therefore, to simplify the development process and reduce the complexity that engineers face at design time, new frameworks have been defined, one of which is Model-based Design (MBD).

Model-based Design is a model-centric framework that enables cost-effective development of heterogeneous dynamic systems such as CPSs [NM18] and reducing design time. It emphasizes the use of a common design environment throughout the development process. A complete model of a CPS contains all aspects of the system including algorithms, logic, mathematical representations and software relevant to physical processes, coupling environment and embedded controllers that represent the end-to end behavior of the system [JCL11]. It facilitates identification and correction of errors through integration testing, generation of embedded software code, refinement of design by tuning parameters using multidomain simulation, development and reuse of test suites and lastly, aids in understanding essential properties of such models.

The impact of a technology can be evaluated by how quickly and enthusiastically it is incorporated by industry and the value of MBD for CPS can only be emphasized by the abundance of tools available for development. We give several examples of MBD tools that support various aspects of CPS development. SysML[1] and AADL[2] are used for system-level modeling, LabVIEW[3] can perform instrument control and industrial automation focusing on measurements, MATLAB Simulink[4] is employed to create complex models of controllers in a dynamic simulation envi-

[1] http://www.omgsysml.org/
[2] http://www.aadl.info/
[3] https://www.ni.com/en-us/shop/labview.html
[4] http://www.mathworks.com/products/simulink.html

ronment, Modelica[5] supports physical system modeling and Ansys 2020 R2[6] can collaborate multiphysics modelling.

Any system at inception or during implementation is not entirely bug free. There exists several classes of faults in CPSs due to its hierarchical and heterogeneous nature. They can be systematic faults that are applicable to both hardware and software which are usually introduced during development or random faults that mostly occur in hardware. These random faults could be classified into permanent faults introduced during the product life-cycle which exist until either repaired or removed or they could be transient faults that occur unpredictably once or a few times during the lifetime of the product due to external influences, for example an SRAM bit flip [Cat01] due to radiation. In this thesis, we only focus on systematic and permanent random fault models.

Debugging of faults is typically a difficult, laborious and an expensive task. The cost of error detection and correction significantly grows in every subsequent step of the development process. Therefore it is essential to ensure correctness of the system as early as possible. It is also time-critical and impacts time to market. More than 50% of the development effort is spent on verification to maintain coherence from concept design to finished product [Lam05]. Verification and Validation (V&V) [DFP+16] of CPS either by exhaustive or simulation-based techniques facilitate checking the compliance of system to design constraints and performance. With CPSs being multidisciplinary systems known for their complexity and heterogeneity, there are multiple theoretical and practical challenges to verify such models.

1.1 Problem Statement and Research Gap

Detecting, localizing and debugging faults is a common activity in any software and harware development process. In virtually any complex system faults exist, and they are very likely to cause failures. Many CPS applications are safety-critical and it is utterly important to address faults and the resulting failures before the product is released.

While there has been tremendous progress in test methodologies and automation for CPS, very little attention has been given to the problem of automated and systematic support for debugging. Fault localization and explanation are key steps in unerstanding and debugging faults. As a consequence, there is an urgent need for methods and tools that can assist the engineer in understanding detected faults and hence facilitate their correction.

Fault localization and explanation are not isolated activities – an accurate localization and a precise explanation are highly sensitive to the quality of the test suite used to detect the fault. In essence, more we know about the system, better fault analysis we can make. It follows that there is a strong correlation between the test coverage and the ability to precisely locate and explain faults. Consequently, it is necessary to (1) analyze existing test suites with faults in mind, and (2) devise procedures for automatically generating test suites that achieve a good coverage and hence can facilitate fault localization and explanation.

[5] https://www.modelica.org/

[6] https://www.ansys.com/en-gb/products/release-highlights

1.2 Aim of the Thesis

This thesis develops solutions for fault-based analysis in CPS. As we have already seen, CPS are heterogeneous systems whose definition varies from an system represented as model to a hardware circuit at the most rudimentary level. At distinct phases and levels of modeling and development of CPS design, the approach for validation, verification and debugging varies. To ensure a fault-free design of such diverse systems, the faults must first be identified, captured and localized as early as possible in design time, leading us to the following question.

RQ1: How to automatically localize faults in CPS models?

In complex CPS, while verification enables early detection of faults, it is not sufficient to debug the models. Debugging industrial sized models manually is highly challenging and a laborious process along with needing field expertise. To simplify and aid in the debugging process, it is not only sufficient to locate the fault but also be able to explain the cause of failures automatically, leading us to the following question.

RQ2: How to automatically explain failures in CPS models?

In the litterature on software diagnosis, there are two major approaches to localize and explain faults – model-based diagnosis and statistical methods for fault analysis. CPS are in many respects more complex than traditional software and it is clear that a single fault analysis method cannot give satisfactory outcomes. In order to improve the fault analysis in CPS, it is therefore important to combine all the information about the system that is available. This includes information about CPS requirements expressed in the form of temporal logic specifications, the structure and the architecture of the CPS model and the simulation data illustrating individual behaviors of the CPS system.

RQ3: How to combine logic-based, model-based and data-driven reasoning to improve debugging information?

The confidence in correctness of the design can be achieved by introducing a notion of coverage to the testing activity. Fault localization and explanation is highly sensitive to the quality of the test suite and its coverage. Hence, it is of extreme importance to have good notions of coverage for CPS models, resulting in the following research question.

RQ4: How can one define a good test coverage for CPS?

While having a good notion coverage allows us to quantify coverage over the existing test suite, this is alone not sufficient. One needs techniques that are able to effectively generate test suites that achieve this coverage.

RQ5: How to generate tests that achieve the good test coverage?

For legacy systems, there are already test suites that are available. Given a specific fault model, it is important to know how much coverage does the existing suite achieves and which test can be associated to which fault – an information that can greatly facilitate fault localization and significantly improve fault explanation.

RQ6: How to measure coverage of specific fault models?

1.3 Methodology

In the previous section, we discussed the goals of this thesis in form of research questions that serves as the beginning to defining our methodological approach. We address both theoretical and practical challenges while providing solutions in the field of fault-based analysis of CPS. Considering that we tackle diverse challenges, the approaches to solving them is also entirely divergent. Figure 1.2 illustrates an overview of areas employed in this thesis for creating solutions to the research questions.

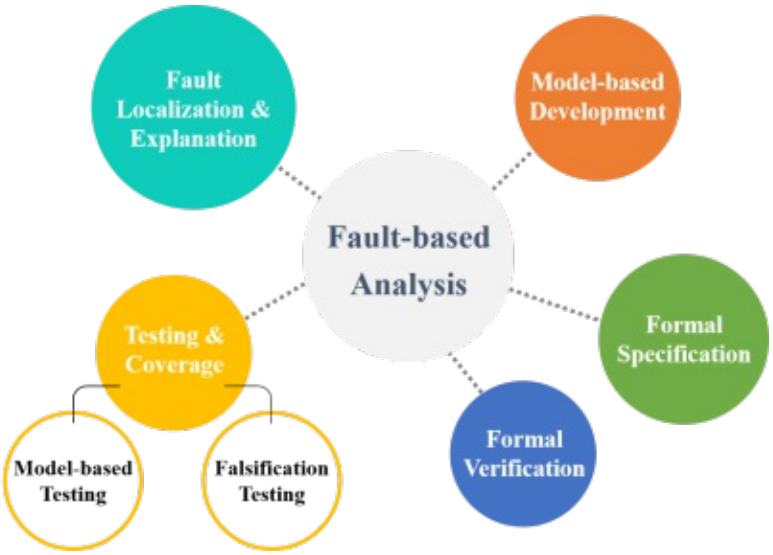

Figure 1.2: Overview of *Fault-based Analysis*

1.3.1 State of the Art

Improvements in usage patterns and performance demands of Cyber-Physical Systems (CPSs) has changed the dynamics of CPS. These changes necessitate improvements in the design and engineering approach [GRSS11, KM15]. Model-based Design (MBD) is a software development methodology that is based on the V-model [Mat20] where a system model at the center of the design process, through defining requirements, design, implementation and verification. It provides a common platform for design of systems with mathematical tools to address challenges associated with communication, control and signal processing. This is integrated it with testing to ensure a valid system is being designed at every stage of the development process. It brings value in accelerated development. It is also observed that MBD supports in early detection and correction of errors. Reuse of models or tests and adapting designs are also simplified as there exists a common environment. While model-based development has become the norm in developing complex CPS, there have been several distinct efforts to streamline the development process of heterogeneous systems by combining the design and testing into the same environment [LFW+16, AVT17] including the ones that involve AI components [DDS19, DFG+19].

Exhaustive verification techniques perform conformance checking of the system model to its specification aiming at obtaining full correctness and making V&V more systematic. Specifications describe the intended behavior of the system and formalisms such as *temporal logics* are used to specify properties of the system in a declarative manner. Model checking was initially introduced as a method for exhaustively verifying the correctness of finite-state systems models as Kripke structures with respect to their specifications [CE81, QS82, CJPG08]. The model of a system represented as a Kripke structure, computing over-approximations of reachable sets of states and checking if all possible executions of the system conforms to the specification guaranteeing correctness or finding counterexamples. From explicit state model checking, this field has evolved into symbolic state model checking [BCM+92] and bounded model checking [BCC+03] where the latter explores finite length paths incrementally to find a counterexample instead of exhaustive model exploration. Exhaustive verification is also used with focus on Hybrid systems [ACHH93, MMP91, Alu11]. Hybrid systems are dynamic systems that exhibit both continuous and discrete behaviors which are also highlighted features of CPSs. Formal verification of Hybrid systems has exhibited impressive results in recent years [Mal13, CÁS13, BFGH17, DDM04, CGMT15, Fre15, SJJ17, FQM+16, BAJD17]. However, these methods still suffer from scalability issues and state-space explosions as CPS models may contain large number of continuous and discrete state variables in the interconnected components. This led to model checking usually being applied to prove absence of bugs, where although they are unable to determine if a system is safe, they may compute witnesses when the system is unsafe [PKV09]. Consequently, simulation-based testing is preferred for verification of systems during design and run-time in the industry.

Testing [ZSM17] in the industry is largely performed in an ad-hoc manner based on engineering experience to uncover faults in the system and expose corner-case violations. *Directed testing* is carried out where generated input vectors target specific features and functionality in regions the designers are aware and *random testing* is performed to steer away from the biased regions and explore other neighbourhoods [Lam05]. Randomized testing has been popular for verification of specifications for large CPS. However, such a process could be error prone as it usually verifies a subset of the state space [DZS+15] and uniform random choice of test inputs is not always optimal [ALFS11] necessitating the evolution of hybrid systems and CPS verification into more rigorous and systematic testing approaches.

A practical solution is to use a hybrid technique that combines formal methods with simulation-based testing [BGH+99]. One class of such methods that is used to detect faults in industrial size systems is falsification-based testing [BDD+18, SF12, NSF+10]. As the name implies, falsification refers to a process of discovering witnesses that prove violation of a specification on a system. It is an iterative search-based approach where in every iteration the inputs of the system are optimized to steer the system closer to the property violation. The search input trace that induces the system to violate the property is guided by the quantitative evaluation of the specification, called robustness degree. Robustness degree is a quantitative measure of how far a trace is from satisfying or violating a specification. Falsification testing is reduced to an optimization problem where the goal is to minimize robustness. S-Taliro [ALFS11] and Breach [Don10] are two of the most prominent tools for falsification of real-time temporal logic

specifications. Since this approach is aimed at detecting bugs in designs, it either terminates immediately after successfully finding the first counterexample or does not witness a violation. We obtain no additional information about the correctness of system or coverage of tests.

In general, coverage is a measure of confidence in the correctness of the system and completeness of a set of tests applied to a design or a model [BGH+99]. Coverage is used as a metric to ensure that the design was tested as thoroughly as possible [BCHN06]. It is not only tedious but unrealistic to test an system-under-test (SUT) with all admissible inputs, therefore effort has been invested to define and implement notions of coverage [ND07] for hybrid systems and CPS. Coverage metrics can be defined for example as state or transition exercised in discrete systems or lines of code executed in software which are described as functional coverage and code coverage. Coverage can also be used to measure progress in verification [DS14]. One needs to identify features that should be verified, define and plan coverage goals. ISO 26262 [Eecgsa18] that sets regulations for functional safety of road vehicles lists some typical goals as coverage of statement, decision or branch. However, fault coverage along with test coverage is especially popular in the semiconductor industry. Coverage-driven verification is a verification methodology that has been established based on the trials and experiences of the digital hardware industry. Major vendors in the semiconductor industry provide Verification IPs that can be integrated directly into the existing environment. However, often there exists test suites such as production tests that are developed in-house with seldom the notion of coverage defined. These tests are designed with the objective of verifying specific features of a design and expose faults, typically permanent random faults.

Testing and runtime verification are complementary approaches extensively used in the development process in industry. Since it is impossible to assess the performance and correctness during design time, it can be augmented by integrating oracles for after deployment. Runtime assurance (RTA) [S+01] ensures safe operation of CPS or other systems that cannot be statically verified. Runtime Verification (RV), as the name implies, is a dynamic analysis technique that deals with the detection of specification violations on a single execution of a system [LS09, CM05, NSBJ15, FHR13]. It is a light-weight formal method that combines formal specifications with individual analysis of the systems providing a scalable solution for verification of both hardware and software systems [Mal16]. Evaluation of the execution trace against the system specification is known as *monitoring* and the generated decision procedure behaving as the oracle for the property is referred to as a *monitor* [FHR13]. The SUT may be instrumented to generate the events or traces for the monitor to consume, regardless the system may also be observed as a black-box, monitoring only the observables in case of CPS. Monitoring maybe performed online during the actual execution of the system or it may be performed offline on a recorded execution trace [MN04]. Since the notion of robustness degree permits expression beyond binary satisfaction, monitoring also conveys the degree of satisfaction.

From a Monitoring Oriented Programming Environment for Java that instruments software using AspectJ [KHH+01] implemented in JavaMOP [CR05] to the Real-Time Analog Monitoring Tool (RTAMT) [NM07, NY20] that implements both an offline monitoring and an incremental algorithm to evaluate the satisfaction of requirements expressed in Signal Temporal Logic (STL), several application domains have benefited using RV. But as in case of model checking, detecting

error in alone may not provide sufficient insight into the violation in runtime verification and often manual effort is required in diagnosing the system.

1.3.2 Specification

A formal specification provides a precise description of the requirements the system should satisfy when designed. This facilitates automatic verification techniques in which the system is checked algorithmically against the requirement encoded in temporal logic [TSL04, Pnu86]. The use of temporal logic in system design turned a new chapter in formal verification [Pnu77, Pnu81, Var09]. Temporal logic provides a formalism to describe the *input-output behavior* of a reactive system being designed should satisfy [DMB+12]. These systems being are viewed as a network of subsystems or blocks which communicate between each other using signals. Temporal logic can be used to express real-time temporal safety and performance properties of CPS.

Signal Temporal Logic (STL) [MN13] extends continuous-time Metric Temporal Logic (MTL) [Koy90] with numerical predicates over real-valued variables [BDD+18]. It allows engineers to describe properties of the system or component interfaces related to events and the temporal distances between them for real-valued variables in real-time. This has been extended over dense-time in recent years [MN04, MNP08]. It was proposed to equip temporal logic with a quantitative interpretation in [FP06, FP09] whose adaptation to STL was studied in [DM10]. This alternative semantics can be used to compute *real-valued robustness* degree as opposed to boolean satisfaction indicating the distance of observed signal from satisfaction or violation of the specification [FND+19]. The monitoring algorithms were developed in [MN04, MN13] and implemented as tool in [NM07]. Signal Temporal Logic (STL) is used with both *past* and *future* operators in this thesis. We interpret STL with its classical semantics as defined in [MN13]. Future temporal operators may use either weak or strong semantics in case of truncated paths as in [EFH+03].

In the context of Chapters 2-5, we consider formal specifications of the SUT as a set of formulas written in STL that the system is expected to model. The reference simulations obtained by simulating non-faulty schematic serves as a specification to compute the distance metric in Chapter 6.

1.3.3 Models

To evaluate our methods and approaches, finding sizable case studies was imperative. Simulink Benchmarks obtained from various venues have been utilized as use cases for illustrating and evaluating our approach in Chapters 2-5. The SPICE schematics and simulations used in Chapter 6 was provided by an industrial partner.

1.3.4 Methods

A fault is a condition that causes the system or a component to behave in an anomalous manner and a failure is the inability of a system to perform according to its specification. Typically, debugging is a laborious and tedious task during the design of CPSs. The software engineering

community has been well invested since several decades to provide techniques that assists the developer to localize faults and explain failures, a comprehensive survey of which is be found in [WGL+16]. *Spectrum-based fault localization* (SBFL) [AZVG07] is a well-established statistical technique that ranks the program components with their likelihood of involvement for the observed faults by measuring the code coverage in successful and failed tests. It has been applied in [DJMP18, LNB+16, LLN+16, LNB+17] to localize faults in Simulink CPS models with comparable accuracy when applied to software systems [LLN+16]. Alternatively, fault diagnosis of model-based techniques has been investigated in control-theory from a different perspective. [FD97] explores a fault detection system based on observable outputs and an on-line parameter estimation approach using state-based observers is discussed in [DA97]. Among a few complementary approaches, [DST17] presents a method to identify the most important segments in the input of a counterexample that falsifies a Spectrum-based Fault Localization (SBFL) requirement. Fault detection can be combined with localization techniques to narrow the search for identifying the cause, one such approach is model slicing. An extensive survey on model slicing is compiled in [ACH+13].

Failure explanation aims at producing information about the failures and their root cause. Some approaches combine mining and dynamic analysis to reveal [NSF+10] and explain failures [BMP09, BWW16, MPP10] in the context of component-based systems. Hynger [JBD15, NHB+18] exploits invariant generation using Daikon [EPG+07] to detect mismatches between the actual and inferred specification for Simulink/Stateflow models. It has also been used for mining invariants and monitoring them for faults or attacks in runtime [BJD17]. Incidentally, methods like [DDP15] can be used to infer the good behavior of the system, which can further be used to classify bad behaviours and identify its cause. Ideally, the inferred properties should be stronger than the actual specification and the mismatch indicates presence of possible specification violations in the model.

The previously discussed fault localization methods assume an existing test suite. However, since the fault localization and failure explanation is as good as the tests, this can be improved by generating suitable test cases [ADTP10, BFLT06, DDD+15, BDF+16, LNB+17, RFZO12] additionally. Although, typical objective of testing involves ensuring the implementation satisfies system specification known as specification-based testing [Kuh99], there could be several objectives of testing that then defines the approach. Fault-based testing [Mor90] relies on injecting a fault into an implementation of the system and executing the test that can detect it or not, a typical example being mutation testing [JH10]. It is also called Defect-oriented Testing (DOT) in context of hardware or circuit schema [Sac13]. Falsification-based testing [AFS+13, ADD+17, AH15, ALFS11, ALFS11, AF10, BS18, DZS+15, NSF+10, SF12, YF17] systematically searches for an input that falsifies the specification. Quantitative semantics [FP09] can be used to indicate the distance of the trace from violating the specification. A model of the SUT is exploited to derive test cases in model-based testing [ABJ+15, ALN13, DZS+15, FKS+17, Tre08]. Adaptive testing [ABJ+15, BFG+19, BHRS15] requires that the test cases adapt based on the observed behavior of SUT in runtime to achive the goal.

The thesis utilizes the existing knowledge base available and builds on them. For example, we combine statistics-based spectral methods with logic-based trace diagnostics and model slicing to

perform system diagnostics.

1.3.5 Platforms & Tools

Development of prototypes and implementation of tools simulating CPS models for the work described in Chapters 2-5 was implemented in MATLAB 2018b with backward compatibility upto 2016b. The adaptive testing algorithm in Chapter 5 was implemented in Python 3.7 with an interface to the MATLAB engine. The fault extraction procedure in Chapter 6 was performed in Cadence Virtuoso and the prototype for test ordering procedure was implemented in C++.

In order to implement the tools and develop prototypes to the research questions discussed in the section 1.2, several tools have been employed. RTAMT library is used to monitor STL specifications and compute the test oracles in Chapters 2-5. In Chapter 2, the trace diagnostics performed using RTAMT is extended to compute epochs on tests labelled with *failed* verdict. In our current implementation of CPSDebug, the tool is interfaced with Daikon and TkT to perform specification mining as described in Chapter 4.

1.4 Contributions of the Author per Chapter

1. **Chapter 2:** The author's contribution is the development of the algorithmic concepts leading to fault localization framework, most of its implementation and experimental work, that resulted from discussions with Ezio Bartocci, Thomas Ferrère and Dejan Ničković.

2. **Chapter 3:** The author's contribution is the implementation of a prototype and experimental work. The formulation of failure explanation methodology resulted from extensive discussions with Ezio Bartocci, Cristinel Mateis, Leonardo Mariani and Dejan Ničković. An initial version of this work has been refined under the guidance of Leonardo Mariani.

3. **Chapter 4:** The author's majority contribution is towards proposing the tool architecture, implementing and integrating it with external tools and libraries along with experimental work presented working in collaboration with Dejan Ničković for integration of the RTAMT library used by the tool and Fabrizio Pastore for extension of TkT with features necessary for the analysis.

4. **Chapter 5:** The author's main contribution was the development of search-based testing method based on Particle Swarm Optimization (PSO), its integration with MATLAB Simulink and the experimental work. The interface with cooperative strategies was performed in close collaboration with Benedikt Madebacher. The development of algorithmic concepts that formed the framework for adaptive testing resulted from discussions with Ezio Bartocci, Roderick Bloem and Dejan Ničković.

5. **Chapter 6:** The author's contribution is development of concepts for the test ordering procedure, experimental work and most of the implementation for fault extraction and test selection procedure. The algorithmic concepts were developed resulting from discussions with Dieter Haerle and Dejan Ničković.

1.5 Thesis Contributions

This thesis provides theoretical, methodological and practical contributions to two main challenges, (1) localization and explanation of faults detected in a CPS, and (2) systematic generation of test cases in a way that can specifically help understanding and explaining faults. The thesis contributed to both, the low- component- and product-level of CPS (Chapter 6) and system-level concept design (Chapters 2-5). The contributions of papers presented in Chapters 2 to 6 are outlined in this section. We structure this section according to the major topics that are addressed in this thesis - fault localization in CPS, algorithms for failure explanation, adaptive testing for specification coverage and fault coverage analysis.

1.5.1 Fault Localization

Development of cyber-physical systems is a complex process. Even with existing advancements in verification technologies, debugging remains tedious and challenging, most often requiring domain knowledge to identify and correct faults in the design. With MATLAB and Simulink/Stateflow becoming synonymous with model-based design (MBD) of control software in CPS, more and more need arises to aid the engineer in not only detecting a failure but be able to localize faults quickly in design time.

[FMN15] presents a trace diagnostics procedure for continuous-time specification languages for MTL, MITL and STL. This method detects the violation of observable signals for a continuous-time temporal logic specification and provides a *temporal implicant* as diagnostics, a set of sub-signals that are sufficient to explain the violation. This technique observes the model as a black-box not aiming to localize faults withing the model but provides valuable spatial and temporal information on the observable variables involved in the violation. In Chapter 2, we present a approach that builds on [FMN15] combining it with model-based and statistical reasoning to automatically localize faults in Simulink/Stateflow models. The approach consists of three stages: (1) specification monitoring and trace (epoch) diagnostics; (2) model slicing; and (3) spectral analysis. The approach assumes a hybrid system in form a Simulink/Stateflow model, a set of STL specifications to be used as oracles and a test suite.

Monitoring and Trace Diagnostics

We begin the *monitoring and trace diagnostics* step by simulating the model with existing test suite (Figure 1.3(a)). Using the monitoring procedure, we then evaluate the observable traces against the STL specification (Figure 1.3(b)). If we detect a specification violation, we use the trace diagnostics procedure to identify all segments of traces responsible for the detected violation (Figure 1.3(c)). From this step, faults can be explained by the identified time segments on traces at the observable interface of the model.

Slicing and Mapping

The previous step identifies violations only at the interface and since we intend to localize faults within the model, we need a procedure to map the trace diagnostics results to the internal signals

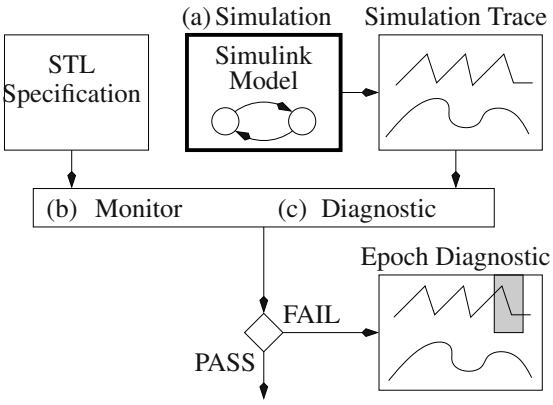

Figure 1.3: System diagnostics procedure overview – monitoring and trace diagnostics.

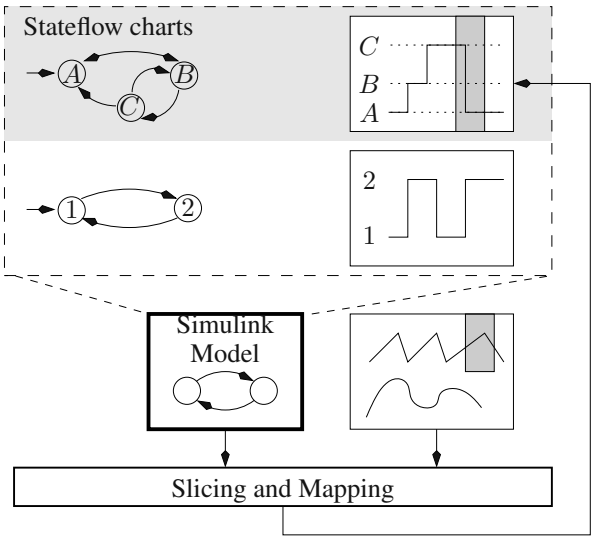

Figure 1.4: System diagnostics procedure overview – model slicing.

of the model. The *model slicing* procedure identifies all components in the model, particularly in Stateflow charts (see shaded light-gray in Figure 1.4), that maps to the outcome of previous step. The procedure then maps the time segments computed by trace diagnostics to the trace that records evolution of the chart's states and transitions over time (see dark-gray band in the trace of highlighted Stateflow chart in Figure 1.4).

Spectrum-based Fault Localization

We apply the previous steps over all tests in the test suite and record the outcome an a tabular form (see Figure 1.5). Each test comprises of a sequence of segments with each segment corresponding to a column in the table with an associated verdict label. The states and transitions of the Stateflow charts are represented as rows in the table. If a state or transition was active during a segment, the

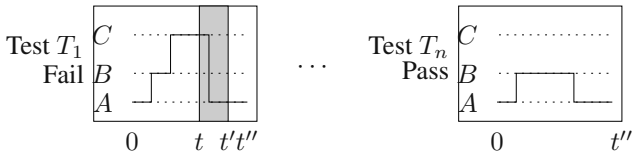

Figure 1.5: System diagnostics procedure overview – spectrum-based fault localization.

corresponding cell is marked with a boolean flag. Finally, the procedure applies Spectrum-based Fault Localization (SBFL) [AZVG07] to compute the states and transitions that are most likely to cause the detected fault.

Chapter 2 answers RQ1 and RQ3 along with addressing RQ2 by developing a fault localization procedure that combines logical-reasoning with model slicing and spectral analysis. Trace diagnostics identifies the segments of observable signals that violate the specification. Model slicing then maps these signals to their origin blocks and spectrum-based fault localization computes the states and transitions capturing the behavior that are most likely to detect the fault.

1.5.2 Failure Explanation

It is often seen that identifying the causes of failures in a CPS model can be as challenging as locating faults themselves [Lee08]. In Chapter 3 we present CPSDebug, a framework for explaining failures in Simulink models. The workflow assumes a target CPS model along with a test suite and a set of model *specifications*. It follows the sequential processes of *testing*, *mining* and *explanation* to produce a failure explanation.

Testing

The workflow of CPSDebug begins by instrumenting the SUT that is a Simulink/Stateflow model. This is a crucial pre-processing step that makes the simulation engine to record every Real, Boolean or Enumeration type internal signals of the model. Similarly, CPSDebug also instruments state machines and look-up tables. *Instrumentation* is defined inductively on the model performed in a bottom-up fashion. In the testing phase, CPSDebug first simulates the model with available *test cases* against the instrumented model. Simulation produces a trace corresponding to each test case and the trace consists of time-value pairs for every instrumented variable. We transform the informal model specification into an STL formula that can be automatically evaluated by a *monitor* to determine the nature of traces. CPSDebug labels every trace *Pass* or *Fail* verdict depending on whether the trace satisfies the STL property or violates it.

Mining

CPSDebug selects all traces labeled with *Pass* verdict and exploits them for *property mining*. It performs several intermediate steps ahead of property inference to facilitate mining. Initially,

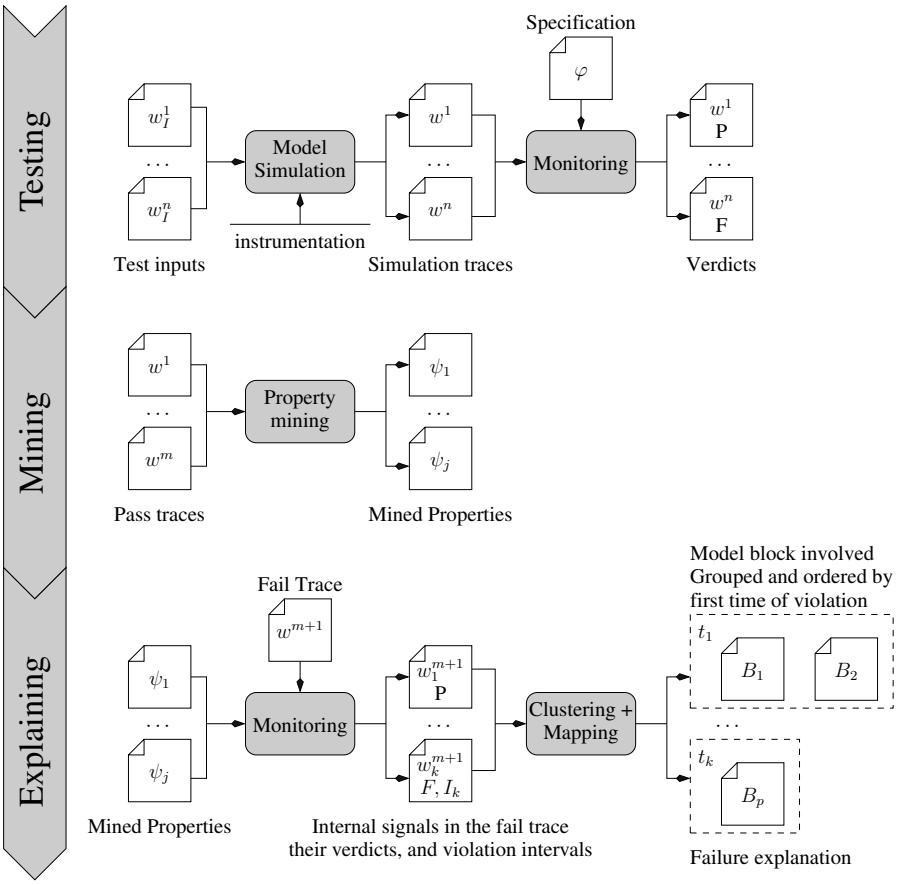

Figure 1.6: Overview of failure explanation framework.

CPSDebug performs cross-correlation between internal signals (or trace variables) to reduce them to a subset of significant trace variables. Pearson method is used to compute cross-correlation co-efficient $P(v_1, v_2)$ between two variables v_1 and v_2, i.e. $P(v_1, v_2) = \frac{cov(v_1,v_2)}{\sigma_{v1}\sigma_{v2}}$ defined in terms of covariance and their standard deviations. CPSDebug non-deterministically removes one of the two variables (and its associated traces) from further analysis when the cross-correlation co-efficient is greater than 0.99, i.e. $P(v_1, v_2) > 0.99$.

CPSDebug associates the variables to it domain and its parent Simulink/Stateflow block. It collects all observations from all samples in traces associated with the variables in the domain. It then infers a set of properties related to the specific block and domain using Daikon [EPG+07] and TkT [PMM17] automata learning. CPSDebug collects the inferred properties from all domains and blocks. Each Daikon property p is transformed to an STL assertion of type $\square p$. TkT properties are in form of timed automata that describes the behavior of state variables and do not need further transformation for use.

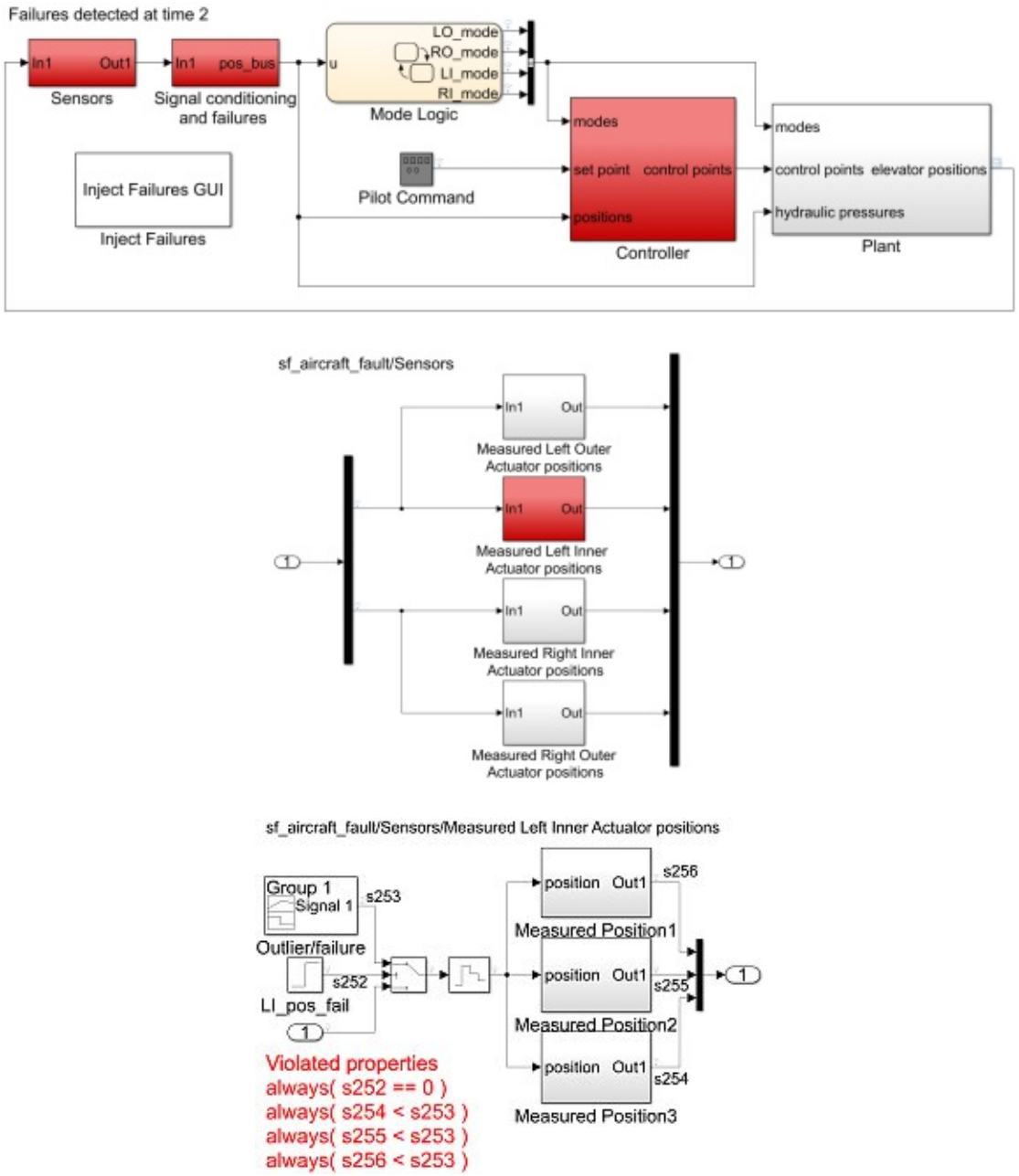

Figure 1.7: Failure explanation as a sequence of snapshots - part of the first snapshot.

Explaining

In this phase, CPSDebug analyses a trace w from a failing execution to produce a failure explanation. The trace is evaluated against the mined properties and *monitors* return the signals that violate the properties and time intervals in which they are violated. The internal signals involved in the violated properties are labeled *Fail* and the remaining signals of the trace as *Pass*

along with the violation time intervals of the violated properties.

In the *Clustering + Mapping* step, CPSDebug (1) clusters the resulting fail-annotated signal instances by their violation time intervals and (2) maps them to their corresponding model blocks that have some of the fail-annotated signal instances as internal signals. The failure explanation generated by CPSDebug captures the origin of the fault and its propagation in space and time. It is a sequence of snapshots of the system, one for each cluster of property violations. The snapshot represents a new relevant state of the system, and the sequence shows how the execution progresses from the violation of the set of properties to the signal violation of the specification (see 1.7). In particular, it contains (1) approximate time of occurrence of the violations in the cluster, (2) model blocks from which the reported violations of the cluster originate, (3) the properties violated by the cluster, that represent the existence of anomalies in the cluster, and (4) internal signals that participate in the violation of properties associated with the cluster. The engineer can exploit this to understand the failure to localize the root cause of the problem.

Chapter 3 answers RQ2 and RQ3 along with addressing RQ1 by developing an automatic failure explanation framework that follows the phases of testing, mining and explanation. The procedure uses Daikon to infer global properties of the system and TkT automata learning for inferring time-dependent behavior of stateful components which are then used to produce failure explanations.

1.5.3 CPSDebug

In the previous section, we discussed CPSDebug, a framework that combined logic based testing, specification mining and failure analysis to automatically identify multiple causes of failures in CPS. The tool CPSDebug constructed based on the discussed framework and its architecture is presented in Chapter 4. The tool is implemented in MATLAB and is available at https://gitlab.com/nmanjunath/cpsdebug.git for download. CPSDebug can be executed from the command line in any environment equipped with Java 8, Python and MATLAB 2016b to 2018b. It can be easily integrated withing automated scripts and fully supports CPS models designed in Simulink/Stateflow but does not support designs that exploit other toolboxes such as Simscape or SimEvents.

The tool takes as input CPS models designed in Simulink/Stateflow, a test suite and a set of specifications written in Signal Temporal Logic (STL). These parameters are included as apart of the configuration file which influences the behavior of the tool. CPSDebug produces a failure explanation table per violation as output which is an ordered sequence of identified snapshots each of which includes the signals that violate the mined properties, the time of violation, the block that the signal belongs to and the violated properties. The high-level architecture of CPSDebug as illustrated in Figure 1.8 comprises of *Property Miner*, *Monitor + Trace Diagnoser* and *Failure Mapper* components along with the *Model Instrumenter*.

The objective of *Model Instrumenter* is to instrument the model under analysis which enables logging of values assigned to every internal signal and variables. It navigates inductively through hierarchical structure the CPS model in a bottom-up fashion adding tracing capabilities to each

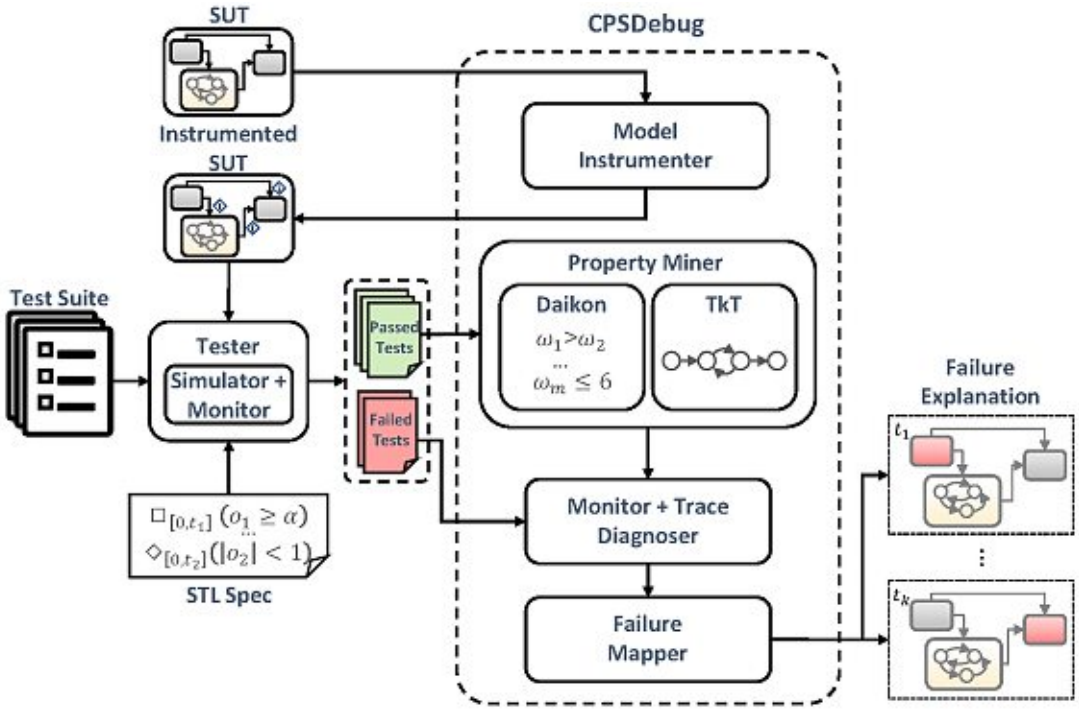

Figure 1.8: Architecture of CPSDebug.

element. It also automatically assigns a unique identifier to every Real, Integer, Boolean and Enumerated-type signal. It is also able to record the transitions of a state machine or compute a trace that emulates the indices of look-up table access.

CPSDebug uses MATLAB Simulation Engine as a *Tester* to simulate the available test cases. In principle, one can interface any suitable test executor with CPSDebug. The output traces from the tests are then automatically evaluated by a *Monitor* against an STL formula and the traces are then labeled with a *Pass* verdict if they satisfy the formula or a *Fail* in case they violate. CPSDebug uses RTAMT library [NY20] to encode and monitor STL specifications and label the outcome of tests.

To mine properties that are a representative of the CPS model it is essential to collect comprehensive data during the testing phase. The traces labeled with a *Pass* verdict is then selected by the *Property Miner* for inference. It is also responsible for excluding correlated signals and grouping traces per component. Multiple mining solutions can be integrated and run in our tool. Daikon and TkT are incorporated in the current version to derive universal and real-time state-based properties respectively.

The *Monitor + Trace Diagnoser* evaluates the traces labeled with *Fail* verdict against the mined properties and produces failure explanations. The evaluation results in a list of signals that violated the mined properties and the time at which these signals first violate the properties. CPSDebug uses RTAMT to detect universal property violations and TkT for real-time state-based

property violations.

These fail-annotated signals are clustered by their violation times and mapped to their corresponding origin components by the *Failure Mapper*. A sequence of snapshots are created using the violated signals and their corresponding origin blocks, one snapshot for each cluster of property violations. This is presented as *Failure Explanation* to the engineer who can then reconstruct the anomalous behaviors observed during the failing executions and identify the root event as well as the component responsible.

Chapter 4 presents the architecture and implementation of the tool for localizing faults and explaining failures answering RQ2 and RQ3 while addressing RQ1. Given a SUT, formal specification and a test suite, the tool automatically captures the faults in the SUT and as failure explanation produces a series of snapshots that can be investigated to identify the cause of the failure.

1.5.4 Adaptive Testing

Falsification-based testing [NSF+10, AFS+13] is a simulation-based testing approach that uses robustness to guide the SUT to specification violation. This method has been effective to detect bugs, however when no violations are witnessed we do obtain much information about the correctness of the system. Falsification typically stops on detecting the first violation effectively detecting at most one fault. Confidence in correctness of design can achieved by a notion of *coverage*, i.e. if the design passes all tests in a test suite that covers a sufficient number variety of tests according to the chosen coverage metric, the design is considered to be correct. In Chapter 5, we present a novel adaptive test methodology for generating tests that maximize coverage [TSL04] of the specification, an overview of which we present in Figure 1.9. We define our coverage metric on the specification as opposed to the typical definition coverage metrics that are defined on the model or the program being tested.

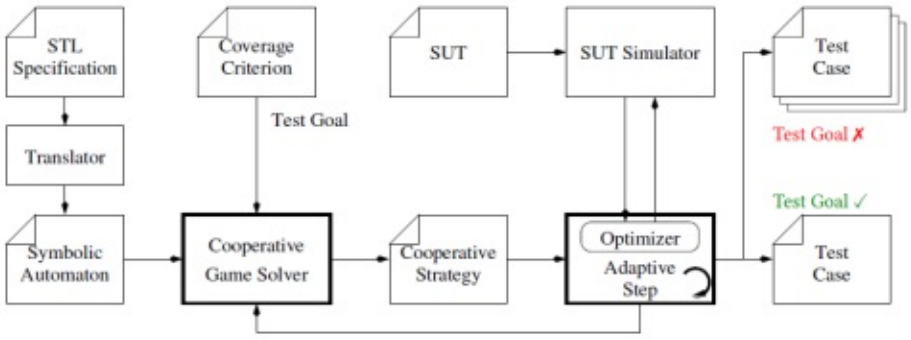

Figure 1.9: Adaptive Testing - overview.

The adaptive testing approach begins with the system requirements formalized as an STL specification. The specification is first translated into its equivalent symbolic automaton. We define

coverage over this automaton and use it to derive test objectives. Given a symbolic automaton and an objective, we interpret it as a symbolic reachability game that is played between the SUT and the tester. The goal of the game is for the tester to bring the SUT to a target state. We assume a setting where the SUT is not adversarial to the tester. Starting from the initial state, the tester moves first by picking inputs and the SUT responds by picking the output.

A *test strategy* determines the value that the tester picks in each move based on the previously visited locations. The state-space of the game is partitioned into three zones: (1) a *safe* zone where the tester can advance the game without the cooperation from the SUT, (2) a *possibly winning* zone where the tester can advance if the SUT is willing to cooperate, and (3) a *no-hope* zone from which the tester cannot advance regardless of the SUT. The outcome of the game is a *cooperative strategy*.

To execute this strategy when in a *possibly winning* location, the tester must steer the SUT to pick a move that allows the tester to advance in the game. We denote this as the *adaptive step*. The problem of guiding the SUT with a cooperative move is formulated as an optimization problem where we use Particle Swarm Optimization (PSO) to direct the SUT behavior in the desired direction efficiently.

The outcome of this approach is set of test cases, one of which reaches the target state. A test suite is an accumulation of tests where each test is an set of finite input traces. These test cases are used to measure coverage and to define the next target. This process is repeated until a desired level of coverage is achieved or a pre-defined budget of maximum allowed simulations is spent.

> Chapter 5 answers RQ4 and RQ5 by defining the notion of specification coverage and developing an adaptive testing approach combining reachability games with numerical optimization to generate tests that maximize specification coverage.

1.5.5 Coverage Analysis of Tests

Field returns have a negative impact in form of huge costs and potential loss of reputations. A fault free product is a consequence of robust test suites. Especially in the semiconductor industry, where production defects are random faults induced by the complex manufacturing process, it is essential to obtain a reliable fault coverage. To perform the coverage analyses of production tests, the evaluation must be conducted in a simulation environment to enable injection of faults into the circuit schematic or system-under-test (SUT). However, long simulation times of production tests can be a huge obstacle and exhaustive fault coverage analysis requires iterative injection of all faults and in worst case simulating all available tests. We address this challenge by developing the procedure in Chapter 6. It presents a method to identify and extract all fault models in an analog and mixed-signal (AMS) circuit schematic and develop a test ordering procedure, a heuristic that enables to choose a test that is most likely to capture the fault. Modeled faults are then injected in the circuit schematic and simulated with test chosen by our heuristic to evaluate the coverage of existing tests.

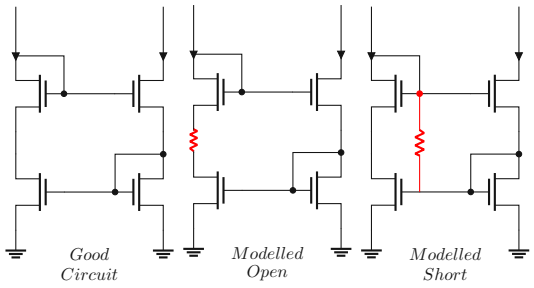

Figure 1.10: Fault injection models.

To extract fault models from a schematic, we must first identify the commonly occurring faults in production. The most common faults in produced ICs are shorts and opens in metal layers [KK07]. An open can be represented by the parasitic resistance (R) from an R-only extraction and a short by the parasitic capacitance (C) from a C-only extraction of a circuit layout. Given a schematic and its corresponding layout, we can extract its parasitics from the layout. To inject an open into the schematic, the corresponding parasitic resistor is assigned a very high resistance value, typically in the order of MΩ. Similarly, the parasitic capacitor can be replaced by a resistor with negligible resistance to emulate a short. Injection of both fault models in schematic is illustrated in Figure 1.10. We extract all parasitics from the layout and compile them into a database of fault models, a snippet of which is shown in Figure 1.11.

Fault Model	Parasitic Element	Parasitance Value	Schematic Netname1	Schematic Netname2	Layout Netname1	Layout Netname2	X-coordinate	Y-coordinate
Short	C21	3.5487E-12	out	gnd	out	gnd	2.56	-40.214
Short	C22	4.5147E-15	net014	I7/I3/net9	152	2942	588.02	412.526
Short	C23	2.7142E-16	I17/close	I17/I15/gate	421	287	212.754	415.847
Open	Rw_m1_124	3.041E-03	I3/net04#254	I3/net04#255	587#254	587#255	2209.0542	-751.36
Open	Rw_m1_129	1.0151	osc#32	osc#33	614#32	614#33	-186.235	58.785
Open	Rw_m1_132	7.064E-04	I2/net34#123	I2/net34#124	1298#123	1298#124	153.354	1354.443
Open	Rw_m1_135	0.4862	net021#74	net021#75	374#74	374#75	-243.153	156.65

Figure 1.11: A segment of the fault database.

The simulation environment consists of a SUT, test suite and the fault database extracted from SUT. The SUT is represented as a tuple of a finite set of real-valued input and output variables and a function that maps the input signals to output signals. An individual *test* in a test suite refers to a set of input signals. Ideally, to obtain the coverage analysis, every fault in the fault database must be injected into the SUT and the mutated system must be tested with a test from the test suite until the mutant is *killed*. We assume a fault, a SUT, its fault induced version and a test from the test suite. The distance of SUT from its mutated version for the given test is the difference in measured observable behavior that the fault induces in SUT over the test. Given a constant that defines an acceptable distance between a SUT and mutated SUT, we consider a test *kills* a mutant if the computed distance is greater than the constant. However, since such exhaustive testing is clearly unrealistic, we propose to execute the test that can catch the fault first, therefore minimizing the testing effort.

Computing the precedence order is as expensive and impractical as to executing all tests on the faulty model. Instead, we propose to use a heuristic that approximates this order with a *score* function denoted by $\sigma(SUT, fault)$ that is fast to compute. This score function assigns a value in $[0, 1]$ to every test-value $(test, fault)$ pair which gives the likelihood of $test$ detecting $fault$. We compute the score function based on two main aspects, (1) *distance score*, and (2) *activity score*. The distance score is a measure of how far the fault is from an observable output. We consider that a fault closer to an observable output is more likely to be detected by a test that affects that output. To compute the *activity score*, we execute the SUT with a test T without injected fault. Each observable output may have a certain activity for the executed test. A measure of how much activity a test creates in an observable output is represented as *activity score*. If a test creates considerable activity in a non-faulty simulation, we consider that the test is most likely to affect the same output when the fault is injected. We also take the simulation time of the tests into account for test ordering. If multiple tests have similar distance and activity scores, we choose the test with shortest simulation time.

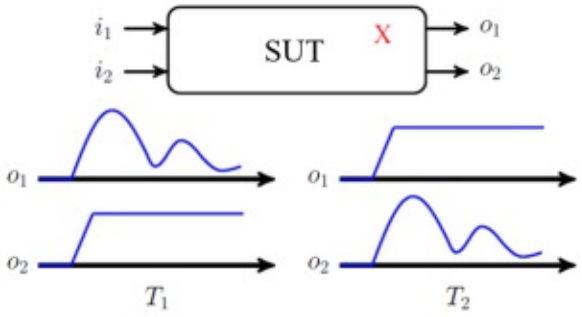

Figure 1.12: Combining distance and activity measures.

We illustrate importance of having both *distance* and *activity* scores in Figure 1.12. We assume a SUT with two outputs o_1 and o_2, a fault X that is close to o_1, and two tests T_1 and T_2. While the test T_1 creates some activity in o_1, the output o_2 is only powered up and the situation is reversed for the test T_2. By combining the distance and activity information, we can deduce that the test T_1 is more likely to detect the fault X than T_2 according to our ordering criterion.

To compute the *Distance Score*, we assume a SUT in form of a SPICE netlist and a fault from the fault database. From the netlist, we generate a *connectivity* graph that contains (1) a finite set of vertices representing all nets and instances from SPICE netlist, (2) a set of input vertices corresponding to input variables, (3) a set of output vertices corresponding to output variables, (4) a finite set of edges, and (5) a function that maps every vertex to a weight. To define edges, we first identify basic components (NMOS and PMOS FETs, resistors, capacitors, diodes and connecting pads) and use the graph translation rules from [Eic13]. We illustrate the schematic netlist to graph translation with an example in Figure 1.13. The instance of the netlist is represented by a square node and the edge between the instances are represented by a circular node in the graph. To avoid trivial path computations, we also remove insignificant connections from the power supply components to all the nets and instances from the analysis. We map a fault to a vertex from the finite set of vertices and compute weight inductively as: (1) if current vertex is the position of

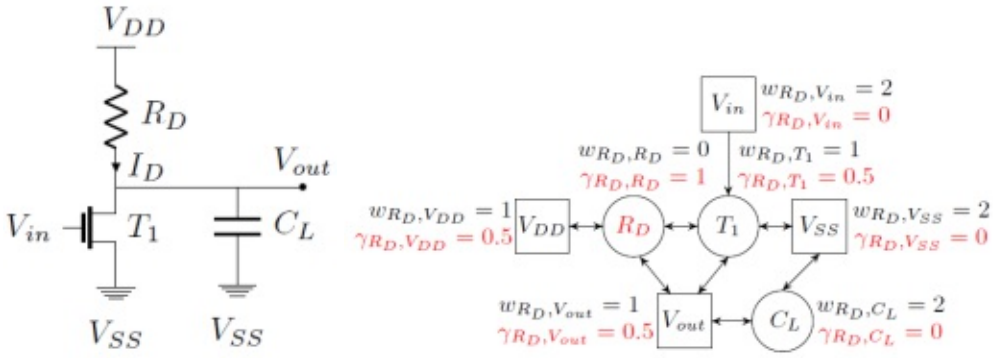

Figure 1.13: (a) Circuit schematic of an NMOS Inverter, (b) Graph translation of an NMOS Inverter with computed weights and normalized distance.

injected fault, the weight of the vertex is 0, and (2) the weight of any vertex in the graph is the minimum edge distance from the fault. We normalize the weights to scale between $[0, 1]$. The *distance score* of an observable output in the SUT with respect to a fault is directly obtained from its weight.

Computation of *Activity Score* requires simulation of SUT with the test without any injected fault. We then record all the output signals and assign the activity score of an observable output under the test and compute the rate of change of the signal. We then normalize the activity values in the interval [0,1].

For a given SUT, test and injected fault, we compute the *Overall Score* for a certain output as a product of the *distance* and *activity* scores from that output. Finally, the *Overall Score* assigned to a test for a given fault is the minimum of overall scores computed from all outputs for the given fault and test.

Once the *Overall Score* is computed for all $(fault, test)$ pairs, the test ordering procedure creates a precedence order of the tests based on the computed overall scores for each fault, indicating that the test with maximum score is most likely to capture the injected fault. Tests with shorter simulation times are prioritized in case of similar overall scores.

This approach illustrated on an AMS circuit could in theory be adapted to analyze and evaluate coverage of any pre-existing test suites for hierarchical system or models comprising a notion of distance and levels of abstraction as long as a well-defined fault model exists.

Chapter 6 describes an approach to determine fault coverage of the existing test suite and develops a test precedence method that, for a given fault, orders tests according to their likelihood to capture the fault answering RQ4 and RQ6.

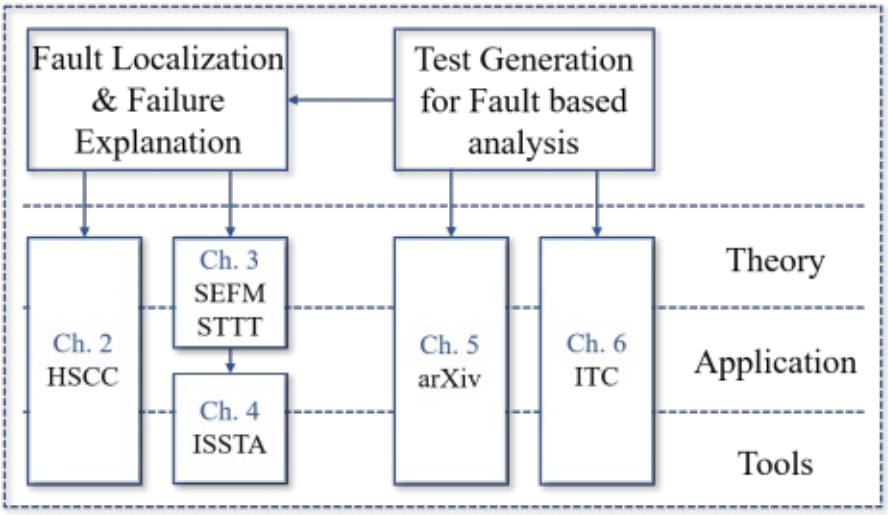

Figure 1.14: Structure of the thesis: reference of the papers to chapters.

1.6 Structure of the Thesis

This thesis explores approaches of fault localization, failure explanation and test generation with intent of fault-based analysis. It consists of five peer-reviewed conference papers and one journal representing research activities carried out between 2016 and 2020. The structure of thesis and classification of the contributions are presented in Figure 1.14. The thesis is structured as follows:

- Chapter 2 proposes a procedure to localize faults in CPS modeled as Simulink/Stateflow designs. The approach combines logic-based reasoning with model slicing and statistical-based spectral methods.
- Chapter 3 presents a framework to explain failures automatically in CPS models. The workflow follows testing, mining and explanation phases to produce failure explanations.
- Chapter 4 discusses the architecture and implementation of CPSDebug, a tool for localizing faults and explaining failures in CPS models.
- Chapter 5 proposes a method for adaptive generation of tests with specification coverage for STL. This approach combines cooperative reachability games with search based-testing to explore the system model and exercise different parts of the specification.
- Chapter 6 presents a fault extraction and a test ordering procedure to analyze the fault coverage of an existing production test suite.

1.7 Bibliography

[ABJ+15] Bernhard K Aichernig, Harald Brandl, Elisabeth Jöbstl, Willibald Krenn, Rupert Schlick, and Stefan Tiran. Killing strategies for model-based mutation testing. *Software Testing, Verification and Reliability*, 25(8):716–748, 2015.

[ACH+13] Kelly Androutsopoulos, David Clark, Mark Harman, Jens Krinke, and Laurence Tratt. State-based model slicing: A survey. *ACM Computing Surveys (CSUR)*, 45(4):1–36, 2013.

[ACHH93] Rajeev Alur, Costas Courcoubetis, Thomas A. Henzinger, and Pei Hsin Ho. Hybrid automata: An algorithmic approach to the specification and verification of hybrid systems. In Robert L. Grossman, Anil Nerode, Anders P. Ravn, and Hans Rischel, editors, *Hybrid Systems*, pages 209–229, Berlin, Heidelberg, 1993. Springer Berlin Heidelberg.

[ADD+17] Arvind S. Adimoolam, Thao Dang, Alexandre Donzé, James Kapinski, and Xiaoqing Jin. Classification and coverage-based falsification for embedded control systems. In *Computer Aided Verification - 29th International Conference, CAV 2017, Heidelberg, Germany, July 24-28, 2017, Proceedings, Part I*, pages 483–503, 2017.

[ADTP10] Shay Artzi, Julian Dolby, Frank Tip, and Marco Pistoia. Directed test generation for effective fault localization. In *Proceedings of the 19th international symposium on Software testing and analysis*, pages 49–60, 2010.

[AF10] Yashwanth Singh Rahul Annapureddy and Georgios E Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *IECON 2010-36th Annual Conference on IEEE Industrial Electronics Society*, pages 91–96. IEEE, 2010.

[AFS+13] Houssam Abbas, Georgios E. Fainekos, Sriram Sankaranarayanan, Franjo Ivancic, and Aarti Gupta. Probabilistic temporal logic falsification of cyber-physical systems. *ACM Trans. Embed. Comput. Syst.*, 12(2s):95:1–95:30, 2013.

[AH15] Takumi Akazaki and Ichiro Hasuo. Time robustness in mtl and expressivity in hybrid system falsification. In *International Conference on Computer Aided Verification*, pages 356–374. Springer, 2015.

[ALFS11] Yashwanth Annpureddy, Che Liu, Georgios Fainekos, and Sriram 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*, pages 254–257. Springer, 2011.

[ALN13] Bernhard K Aichernig, Florian Lorber, and Dejan Ničković. Time for mutants—model-based mutation testing with timed automata. In *International Conference on Tests and Proofs*, pages 20–38. Springer, 2013.

[Alu11] Rajeev Alur. Formal verification of hybrid systems. In *Proceedings of the ninth ACM international conference on Embedded software*, pages 273–278, 2011.

[AS17] Massoud Amin and Phillip F. Schewe. Preventing blackouts: Building a smarter power grid. *Scientfic American*, 2017.

[AVT17] Aivo Anier, Jüri Vain, and Leonidas Tsiopoulos. Dtron: a tool for distributed model-based testing of time critical applications. *Proceedings of the Estonian Academy of Sciences*, 66(1), 2017.

[AZVG07] Rui Abreu, Peter Zoeteweij, and Arjan JC Van Gemund. On the accuracy of spectrum-based fault localization. In *Testing: Academic and industrial conference practice and research techniques-MUTATION (TAICPART-MUTATION 2007)*, pages 89–98. IEEE, 2007.

[BAJD17] O. A. Beg, H. Abbas, T. T. Johnson, and A. Davoudi. Model validation of pwm dc–dc converters. *IEEE Transactions on Industrial Electronics*, 64(9):7049–7059, 2017.

[Bar15] Massimo Barbato. Inside amberg: Industry 4.0 in action. *Chartered Management Institute*, 2015.

[BCC+03] Armin Biere, Alessandro Cimatti, Edmund M. Clarke, Ofer Strichman, and Yunshan Zhu. Bounded model checking. *Adv. Comput.*, 58:117–148, 2003.

[BCHN06] Janick Bergeron, Eduard Cerny, Alan Hunter, and Andy Nightingale. *Verification methodology manual for SystemVerilog*. Springer Science & Business Media, 2006.

[BCM+92] Jerry R Burch, Edmund M Clarke, Kenneth L McMillan, David L Dill, and Lain-Jinn Hwang. Symbolic model checking: 1020 states and beyond. *Information and computation*, 98(2):142–170, 1992.

[BDD+18] Ezio Bartocci, Jyotirmoy Deshmukh, Alexandre Donzé, Georgios Fainekos, Oded Maler, Dejan Ničković, and Sriram Sankaranarayanan. Specification-based monitoring of cyber-physical systems: A survey on theory, tools and applications. In *Lectures on Runtime Verification*, pages 135–175. Springer, 2018.

[BDF+16] Sergiy Bogomolov, Alexandre Donzé, Goran Frehse, Radu Grosu, Taylor T. Johnson, Hamed Ladan, Andreas Podelski, and Martin Wehrle. Guided search for hybrid systems based on coarse-grained space abstractions. *Int. J. Softw. Tools Technol. Transf.*, 18(4):449–467, 2016.

[BFG+19] Roderick Bloem, Goerschwin Fey, Fabian Greif, Robert Könighofer, Ingo Pill, Heinz Riener, and Franz Röck. Synthesizing adaptive test strategies from temporal logic specifications. *Formal methods in system design*, 55(2):103–135, 2019.

[BFGH17] Sergiy Bogomolov, Goran Frehse, Mirco Giacobbe, and Thomas A. Henzinger. Counterexample-guided refinement of template polyhedra. In Axel Legay and Tiziana Margaria, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 23rd International Conference, TACAS 2017, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2017, Uppsala, Sweden, April 22-29, 2017, Proceedings, Part I*, volume 10205 of *Lecture Notes in Computer Science*, pages 589–606, 2017.

[BFLT06] Benoit Baudry, Franck Fleurey, and Yves Le Traon. Improving test suites for efficient fault localization. In *Proceedings of the 28th international conference on Software engineering*, pages 82–91, 2006.

[BGH+99] Mike Benjamin, Daniel Geist, Alan Hartman, Gerard Mas, Ralph Smeets, and Yaron Wolfsthal. A study in coverage-driven test generation. In *Proceedings of the 36th annual ACM/IEEE Design Automation Conference*, pages 970–975, 1999.

[BHRS15] Roderick Bloem, Daniel Hein, Franz Röck, and Richard Schumi. Case study: Automatic test case generation for a secure cache implementation. In *International Conference on Tests and Proofs*, pages 58–75. Springer, 2015.

[BJD17] O. A. Beg, T. T. Johnson, and A. Davoudi. Detection of false-data injection attacks in cyber-physical dc microgrids. *IEEE Transactions on Industrial Informatics*, 13(5):2693–2703, 2017.

[BMP09] Anton Babenko, Leonardo Mariani, and Fabrizio Pastore. Ava: automated interpretation of dynamically detected anomalies. In *Proceedings of the eighteenth international symposium on Software testing and analysis*, pages 237–248, 2009.

[BS18] Luca Bortolussi and Simone Silvetti. Bayesian statistical parameter synthesis for linear temporal properties of stochastic models. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 396–413. Springer, 2018.

[BWW16] Mitra Tabaei Befrouei, Chao Wang, and Georg Weissenbacher. Abstraction and mining of traces to explain concurrency bugs. *Formal Methods in System Design*, 49(1-2):1–32, 2016.

[CÁS13] Xin Chen, Erika Ábrahám, and Sriram Sankaranarayanan. Flow*: An analyzer for non-linear hybrid systems. In Natasha Sharygina and Helmut Veith, editors, *Computer Aided Verification - 25th International Conference, CAV 2013, Saint Petersburg, Russia, July 13-19, 2013. Proceedings*, volume 8044 of *Lecture Notes in Computer Science*, pages 258–263. Springer, 2013.

[Cat01] Anthony Cataldo. Sram soft errors cause hard network problems. *EETimes*, 2001.

[CE81] Edmund M Clarke and E Allen Emerson. Design and synthesis of synchronization skeletons using branching time temporal logic. In *Workshop on Logic of Programs*, pages 52–71. Springer, 1981.

[CGMT15] Alessandro Cimatti, Alberto Griggio, Sergio Mover, and Stefano Tonetta. Hycomp: An smt-based model checker for hybrid systems. In Christel Baier and Cesare Tinelli, editors, *Tools and Algorithms for the Construction and Analysis of Systems - 21st International Conference, TACAS 2015, Held as Part of the European Joint Conferences on Theory and Practice of Software, ETAPS 2015, London, UK, April 11-18, 2015. Proceedings*, volume 9035 of *Lecture Notes in Computer Science*, pages 52–67. Springer, 2015.

[CJPG08] Baier Christel, Katoen Joost-Pieter, and Larsen Kim Guldstrand. Principles of model checking, 2008.

[CM05] SÈverine Colin and Leonardo Mariani. Run-time verification, volume 3472 of lecture notes in computer science, chapter 18, 2005.

[CR05] Feng Chen and Grigore Roşu. Java-mop: A monitoring oriented programming environment for java. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 546–550. Springer, 2005.

[DA97] Laurian Dinca and Tunc Aldemir. An on-line parameter estimation scheme for fault diagnosis. *IFAC Proceedings Volumes*, 30(18):289–294, 1997.

[DDD+15] Tommaso Dreossi, Thao Dang, Alexandre Donzé, James Kapinski, Xiaoqing Jin, and Jyotirmoy V. Deshmukh. Efficient guiding strategies for testing of temporal properties of hybrid systems. In *NASA Formal Methods - 7th International Symposium, NFM 2015, Pasadena, CA, USA, April 27-29, 2015, Proceedings*, pages 127–142, 2015.

[DDM04] Thao Dang, Alexandre Donzé, and Oded Maler. Verification of analog and mixed-signal circuits using hybrid system techniques. In *Formal Methods in Computer-Aided Design, 5th International Conference, FMCAD 2004, Austin, Texas, USA, November 15-17, 2004, Proceedings*, pages 21–36, 2004.

[DDP15] Thao Dang, Tommaso Dreossi, and Carla Piazza. Parameter synthesis through temporal logic specifications. In *FM 2015: Formal Methods - 20th International Symposium, Oslo, Norway, June 24-26, 2015, Proceedings*, pages 213–230, 2015.

[DDS19] Tommaso Dreossi, Alexandre Donzé, and Sanjit A Seshia. Compositional falsification of cyber-physical systems with machine learning components. *Journal of Automated Reasoning*, 63(4):1031–1053, 2019.

[DFG+19] Tommaso Dreossi, Daniel J Fremont, Shromona Ghosh, Edward Kim, Hadi Ravanbakhsh, Marcell Vazquez-Chanlatte, and Sanjit A Seshia. Verifai: A toolkit for the formal design and analysis of artificial intelligence-based systems. In *International Conference on Computer Aided Verification*, pages 432–442. Springer, 2019.

[DFP+16] Parasara Sridhar Duggirala, Chuchu Fan, Matthew Potok, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, Stanley Bak, Sergiy Bogomolov, Taylor T. Johnson, Luan Viet

Nguyen, Christian Schilling, Andrew Sogokon, Hoang-Dung Tran, and Weiming Xiang. Tutorial: Software tools for hybrid systems verification, transformation, and synthesis: C2e2, hyst, and tulip. In *2016 IEEE Conference on Control Applications, CCA 2016, Buenos Aires, Argentina, September 19-22, 2016*, pages 1024–1029, 2016.

[DJMP18] Jyotirmoy Deshmukh, Xiaoqing Jin, Rupak Majumdar, and Vinayak S. Prabhu. Parameter optimization in control software using statistical fault localization techniques. In *Proceedings of the 9th ACM/IEEE International Conference on Cyber-Physical Systems*, ICCPS '18, page 220–231. IEEE Press, 2018.

[DM10] Alexandre Donzé and Oded Maler. Robust satisfaction of temporal logic over real-valued signals. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 92–106. Springer, 2010.

[DMB+12] Alexandre Donzé, Oded Maler, Ezio Bartocci, Dejan Nickovic, Radu Grosu, and Scott A. Smolka. On temporal logic and signal processing. In *Automated Technology for Verification and Analysis - 10th International Symposium, ATVA 2012, Thiruvananthapuram, India, October 3-6, 2012. Proceedings*, pages 92–106, 2012.

[Don10] Alexandre Donzé. Breach, A toolbox for verification and parameter synthesis of hybrid systems. In Tayssir Touili, Byron Cook, and Paul B. Jackson, editors, *Computer Aided Verification, 22nd International Conference, CAV 2010, Edinburgh, UK, July 15-19, 2010. Proceedings*, volume 6174 of *Lecture Notes in Computer Science*, pages 167–170. Springer, 2010.

[DS14] Thao Dang and Noa Shalev. Test coverage estimation using threshold accepting. In *Automated Technology for Verification and Analysis - 12th International Symposium, ATVA 2014, Sydney, NSW, Australia, November 3-7, 2014, Proceedings*, pages 115–128, 2014.

[DST17] Ram Das Diwakaran, Sriram Sankaranarayanan, and Ashutosh Trivedi. Analyzing neighborhoods of falsifying traces in cyber-physical systems. In *Proceedings of the 8th International Conference on Cyber-Physical Systems*, pages 109–119, 2017.

[DZS+15] Adel Dokhanchi, Aditya Zutshi, Rahul T Sriniva, Sriram Sankaranarayanan, and Georgios Fainekos. Requirements driven falsification with coverage metrics. In *2015 International Conference on Embedded Software (EMSOFT)*, pages 31–40. IEEE, 2015.

[Eecgsa18] ISO/TC 22/SC 32 Electrical, electronic components, and general system aspects. Iso 26262-1:2018 road vehicles — functional safety. *ISO*, 2018.

[EFH+03] Cindy Eisner, Dana Fisman, John Havlicek, Yoad Lustig, Anthony McIsaac, and David Van Campenhout. Reasoning with temporal logic on truncated paths. In Warren A. Hunt Jr. and Fabio Somenzi, editors, *Computer Aided Verification, 15th*

International Conference, CAV 2003, Boulder, CO, USA, July 8-12, 2003, Proceedings, volume 2725 of *Lecture Notes in Computer Science*, pages 27–39. Springer, 2003.

[Eic13] Michael Eick. *Structure and signal path analysis for analog and digital circuits*. PhD thesis, Technische Universität München, 2013.

[EPG+07] Michael D. Ernst, Jeff H. Perkins, Philip J. Guo, Stephen McCamant, Carlos Pacheco, Matthew S. Tschantz, and Chen Xiao. The daikon system for dynamic detection of likely invariants. *Sci. Comput. Program.*, 69(1-3):35–45, 2007.

[FD97] Paul M Frank and Xianchun Ding. Survey of robust residual generation and evaluation methods in observer-based fault detection systems. *Journal of process control*, 7(6):403–424, 1997.

[FHR13] Yliès Falcone, Klaus Havelund, and Giles Reger. A tutorial on runtime verification. *Engineering dependable software systems*, 34:141–175, 2013.

[FKS+17] Andreas Fellner, Willibald Krenn, Rupert Schlick, Thorsten Tarrach, and Georg Weissenbacher. Model-based, mutation-driven test case generation via heuristic-guided branching search. In *Proceedings of the 15th ACM-IEEE International Conference on Formal Methods and Models for System Design*, MEMOCODE '17, page 56–66, New York, NY, USA, 2017. Association for Computing Machinery.

[FMN15] Thomas Ferrère, Oded Maler, and Dejan Nickovic. Trace diagnostics using temporal implicants. In *Automated Technology for Verification and Analysis - 13th International Symposium, ATVA 2015, Shanghai, China, October 12-15, 2015, Proceedings*, pages 241–258, 2015.

[FND+19] Thomas Ferrère, Dejan Nickovic, Alexandre Donzé, Hisahiro Ito, and James Kapinski. Interface-aware signal temporal logic. In *Proceedings of the 22nd ACM International Conference on Hybrid Systems: Computation and Control*, pages 57–66, 2019.

[FP06] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications. In *Formal Approaches to Software Testing and Runtime Verification*, pages 178–192. Springer, 2006.

[FP09] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theoretical Computer Science*, 410(42):4262–4291, 2009.

[FQM+16] Chuchu Fan, Bolun Qi, Sayan Mitra, Mahesh Viswanathan, and Parasara Sridhar Duggirala. Automatic reachability analysis for nonlinear hybrid models with C2E2. In Swarat Chaudhuri and Azadeh Farzan, editors, *Computer Aided Verification - 28th International Conference, CAV 2016, Toronto, ON, Canada, July 17-23, 2016, Proceedings, Part I*, volume 9779 of *Lecture Notes in Computer Science*, pages 531–538. Springer, 2016.

[Fre15] Goran Frehse. Reachability of hybrid systems in space-time. In Alain Girault and Nan Guan, editors, *2015 International Conference on Embedded Software, EMSOFT 2015, Amsterdam, Netherlands, October 4-9, 2015*, pages 41–50. IEEE, 2015.

[Gre15] Philipp Greiner. Research 2.0: How to build a smart city. *TEDx Vienna*, 2015.

[GRSS11] Holger Giese, Bernhard Rumpe, Bernhard Schätz, and Janos Sztipanovits. Science and engineering of cyber-physical systems (dagstuhl seminar 11441). *Dagstuhl Reports*, 1(11):1–22, 2011.

[GS20] Brenda Goh and Yilei Sun. Tesla 'very close' to level 5 autonomous driving technology, musk says. *Reuters*, 2020.

[JBD15] Taylor T Johnson, Stanley Bak, and Steven Drager. Cyber-physical specification mismatch identification with dynamic analysis. In *Proceedings of the ACM/IEEE Sixth International Conference on Cyber-Physical Systems*, pages 208–217, 2015.

[JCL11] Jeff C. Jensen, Danica H. Chang, and Edward A. Lee. A model-based design methodology for cyber-physical systems. In *Proceedings of the 7th International Wireless Communications and Mobile Computing Conference, IWCMC 2011, Istanbul, Turkey, 4-8 July, 2011*, pages 1666–1671, 2011.

[JH10] Yue Jia and Mark Harman. An analysis and survey of the development of mutation testing. *IEEE transactions on software engineering*, 37(5):649–678, 2010.

[KHH+01] Gregor Kiczales, Erik Hilsdale, Jim Hugunin, Mik Kersten, Jeffrey Palm, and William G Griswold. An overview of aspectj. In *European Conference on Object-Oriented Programming*, pages 327–354. Springer, 2001.

[KK07] Israel Koren and C. Mani Krishna. *Fault-Tolerant Systems*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 1st edition, 2007.

[KM15] Siddhartha Kumar Khaitan and James D. McCalley. Design techniques and applications of cyberphysical systems: A survey. *IEEE Systems Journal*, 9(2):350–365, 2015.

[Koy90] Ron Koymans. Specifying real-time properties with metric temporal logic. *Real-time systems*, 2(4):255–299, 1990.

[Kuh99] D Richard Kuhn. Fault classes and error detection capability of specification-based testing. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 8(4):411–424, 1999.

[Lam05] William Lam. *Hardware Design Verification: Simulation and Formal Method-Based Approaches*. Prentice Hall, 2005.

[Lee08] Edward A Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, pages 363–369. IEEE, 2008.

[Lee15] Edward Lee. The past, present and future of cyber-physical systems: A focus on models. *Sensors*, 15(3):4837–4869, Feb 2015.

[Leg19] Theo Legget. Boeing 737 max lion air crash caused by series of failures. *BBC*, 2019.

[LFW+16] Peter Gorm Larsen, John Fitzgerald, Jim Woodcock, Peter Fritzson, Jörg Brauer, Christian Kleijn, Thierry Lecomte, Markus Pfeil, Ole Green, Stylianos Basagiannis, et al. Integrated tool chain for model-based design of cyber-physical systems: The into-cps project. In *2016 2nd International Workshop on Modelling, Analysis, and Control of Complex CPS (CPS Data)*, pages 1–6. IEEE, 2016.

[LLN+16] Bing Liu, Lucia, Shiva Nejati, Lionel C Briand, and Thomas Bruckmann. Simulink fault localization: an iterative statistical debugging approach. *Software Testing, Verification and Reliability*, 26(6):431–459, 2016.

[LNB+16] Bing Liu, Shiva Nejati, Lionel Briand, Thomas Bruckmann, et al. Localizing multiple faults in simulink models. In *2016 IEEE 23rd International Conference on Software Analysis, Evolution, and Reengineering (SANER)*, volume 1, pages 146–156. IEEE, 2016.

[LNB+17] Bing Liu, Shiva Nejati, Lionel C Briand, et al. Improving fault localization for simulink models using search-based testing and prediction models. In *2017 IEEE 24th International Conference on Software Analysis, Evolution and Reengineering (SANER)*, pages 359–370. IEEE, 2017.

[LS09] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *The Journal of Logic and Algebraic Programming*, 78(5):293–303, 2009.

[Mal13] Oded Maler. Algorithmic verification of continuous and hybrid systems. In *15th International Workshop on Verification of Infinite-State Systems, INFINITY 2013*, pages 48–69, 2013.

[Mal16] Oded Maler. Some thoughts on runtime verification. In *International Conference on Runtime Verification*, pages 3–14. Springer, 2016.

[Mat20] Mathworks. Model-based design with simulink. *Mathworks*, 2020.

[McC19] Phil McCausland. Self-driving uber car that hit and killed woman did not recognize that pedestrians jaywalk. *NBC News*, 2019.

[MMP91] Oded Maler, Zohar Manna, and Amir Pnueli. From timed to hybrid systems. In *In Real-Time: Theory in Practice, REX Workshop, Mook, The Netherlands, June 3-7, 1991*, pages 447–484, 1991.

[MN04] Oded Maler and Dejan Nickovic. Monitoring temporal properties of continuous signals. In *Formal Techniques, Modelling and Analysis of Timed and Fault-Tolerant Systems*, pages 152–166. Springer, 2004.

[MN13] Oded Maler and Dejan Nickovic. Monitoring properties of analog and mixed-signal circuits. *Int. J. Softw. Tools Technol. Transf.*, 15(3):247–268, 2013.

[MNP08] Oded Maler, Dejan Nickovic, and Amir Pnueli. Checking temporal properties of discrete, timed and continuous behaviors. In *Pillars of computer science*, pages 475–505. Springer, 2008.

[Mor90] Larry J. Morell. A theory of fault-based testing. *IEEE Transactions on Software Engineering*, 16(8):844–857, 1990.

[MPP10] Leonardo Mariani, Fabrizio Pastore, and Mauro Pezze. Dynamic analysis for diagnosing integration faults. *IEEE Transactions on Software Engineering*, 37(4):486–508, 2010.

[Nag20] Devanjana Nag. Wow! indian railways introduces drone-based surveillance system to protect railway assets; details. *Financial Express*, 2020.

[ND07] Tarik Nahhal and Thao Dang. Test coverage for continuous and hybrid systems. In *International Conference on Computer Aided Verification*, pages 449–462. Springer, 2007.

[Neu19] Karl Neuwinger. Driverless buses in vienna's seestadt on the road again. *Metropole*, 2019.

[NHB+18] Luan V Nguyen, Khaza Anuarul Hoque, Stanley Bak, Steven Drager, and Taylor T Johnson. Cyber-physical specification mismatches. *ACM Transactions on Cyber-Physical Systems*, 2(4):1–26, 2018.

[NM07] Dejan Nickovic and Oded Maler. Amt: A property-based monitoring tool for analog systems. In *International Conference on Formal Modeling and Analysis of Timed Systems*, pages 304–319. Springer, 2007.

[NM18] Gabriela Nicolescu and Pieter J Mosterman. *Model-based design for embedded systems*. Crc Press, 2018.

[NSBJ15] Luan Viet Nguyen, Christian Schilling, Sergiy Bogomolov, and Taylor T. Johnson. Runtime verification for hybrid analysis tools. In *Runtime Verification - 6th International Conference, RV 2015 Vienna, Austria, September 22-25, 2015. Proceedings*, pages 281–286, 2015.

[NSF+10] Truong Nghiem, Sriram Sankaranarayanan, Georgios Fainekos, Franjo Ivancić, Aarti Gupta, and George J. Pappas. Monte-carlo techniques for falsification of temporal properties of non-linear hybrid systems. In *HSCC '10: Proceedings of the 13th ACM international conference on Hybrid systems: computation and control*, New York, NY, USA, 2010. Association for Computing Machinery.

[NY20] Dejan Nickovic and Tomoya Yamaguchi. Rtamt: Online robustness monitors from stl. *arXiv preprint arXiv:2005.11827*, 2020.

[PKV09] Erion Plaku, Lydia E Kavraki, and Moshe Y Vardi. Falsification of ltl safety properties in hybrid systems. In *International Conference on Tools and Algorithms for the Construction and Analysis of Systems*, pages 368–382. Springer, 2009.

[PMM17] Fabrizio Pastore, Daniela Micucci, and Leonardo Mariani. Timed k-tail: Automatic inference of timed automata. In *2017 IEEE International Conference on Software Testing, Verification and Validation, ICST 2017, Tokyo, Japan, March 13-17, 2017*, pages 401–411. IEEE Computer Society, 2017.

[Pnu77] Amir Pnueli. The temporal logic of programs. In *18th Annual Symposium on Foundations of Computer Science (sfcs 1977)*, pages 46–57. IEEE, 1977.

[Pnu81] Amir Pnueli. The temporal semantics of concurrent programs. *Theoretical computer science*, 13(1):45–60, 1981.

[Pnu86] Amir Pnueli. *Applications of temporal logic to the specification and verification of reactive systems: A survey of current trends*, pages 510–584. Springer Berlin Heidelberg, Berlin, Heidelberg, 1986.

[Por19] Jon Porter. Pedestrian collision puts vienna's driverless bus trial on hold. *The Verge*, 2019.

[Pre20] Preetipadma. How drones are accelerating in the warehousing and logistics industry. *Analytics Insight*, 2020.

[QS82] Jean-Pierre Queille and Joseph Sifakis. Specification and verification of concurrent systems in cesar. In *International Symposium on programming*, pages 337–351. Springer, 1982.

[RFZO12] Jeremias Röβler, Gordon Fraser, Andreas Zeller, and Alessandro Orso. Isolating failure causes through test case generation. In *Proceedings of the 2012 international symposium on software testing and analysis*, pages 309–319, 2012.

[S+01] Lui Sha et al. Using simplicity to control complexity. *IEEE Software*, 18(4):20–28, 2001.

[Sac13] Manoj Sachdev. *Defect oriented testing for CMOS analog and digital circuits*, volume 10. Springer Science & Business Media, 2013.

[SF12] Sriram Sankaranarayanan and Georgios Fainekos. Falsification of temporal properties of hybrid systems using the cross-entropy method. In *HSCC '12: Proceedings of the 15th ACM international conference on Hybrid systems: computation and control*, New York, NY, USA, 2012. Association for Computing Machinery.

[SJJ17] Andrew Sogokon, Paul B. Jackson, and Taylor T. Johnson. Verifying safety and persistence properties of hybrid systems using flowpipes and continuous invariants. In *NASA Formal Methods - 9th International Symposium, NFM 2017, Moffett Field, CA, USA, May 16-18, 2017, Proceedings*, pages 194–211, 2017.

[Tre08] Jan Tretmans. Model based testing with labelled transition systems. In *Formal methods and testing*, pages 1–38. Springer, 2008.

[TSL04] Li Tan, Oleg Sokolsky, and Insup Lee. Specification-based testing with linear temporal logic. In *Proceedings of the 2004 IEEE International Conference on Information Reuse and Integration, 2004. IRI 2004.*, pages 493–498. IEEE, 2004.

[Var09] Moshe Y. Vardi. From philosophical to industrial logics. In Ramaswamy Ramanujam and Sundar Sarukkai, editors, *Logic and Its Applications, Third Indian Conference, ICLA 2009, Chennai, India, January 7-11, 2009. Proceedings*, volume 5378 of *Lecture Notes in Computer Science*, pages 89–115. Springer, 2009.

[Vin19] James Vincent. Amazon has made its own autonomous six-wheeled delivery robot. *The Verge*, 2019.

[Vol20] Gian Volpicelli. The uk is spending big on migrant-tracking drones to surveil the seas. *Wired*, 2020.

[WGL+16] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8):707–740, 2016.

[YF17] Shakiba Yaghoubi and Georgios Fainekos. Hybrid approximate gradient and stochastic descent for falsification of nonlinear systems. In *2017 American Control Conference (ACC)*, pages 529–534. IEEE, 2017.

[ZSM17] Justyna Zander, Ina Schieferdecker, and Pieter J Mosterman. *Model-based testing for embedded systems*. CRC press, 2017.

CHAPTER 2

Localizing Faults in Simulink/Stateflow Models with STL

Abstract. Fault-localization is considered to be a very tedious and time-consuming activity in the design of complex Cyber-Physical Systems (CPS). This laborious task essentially requires expert knowledge of the system in order to discover the cause of the fault. In this context, we propose a new procedure that aids designers in debugging Simulink/Stateflow hybrid system models, guided by Signal Temporal Logic (STL) specifications. The proposed method relies on three main ingredients: (1) a monitoring and a trace diagnostics procedure that checks whether a tested behavior satisfies or violates an STL specification, localizes time segments and interfaces variables contributing to the property violations; (2) a slicing procedure that maps these observable behavior segments to the internal states and transitions of the Simulink model; and (3) a spectrum-based fault-localization method that combines the previous analysis from multiple tests to identify the internal states and/or transitions that are the most likely to explain the fault. We demonstrate the applicability of our approach on two Simulink models from the automotive and the avionics domain.

Keywords. Software Testing and Debugging, Model Verification and Validation.

The content of this chapter has been published: E. Bartocci, T. Ferrère, N. Manjunath, D. Ničković: Localizing Faults in Simulink/Stateflow Models with STL, *ACM International Conference on Hybrid Systems: Computation and Control*, HSCC 2018, Porto, Portugal, April 11-13, 2018, pp 197-206. DOI:10.1145/3178126.3178131

CHAPTER 3

CPSDebug: Automatic Failure Explanation in CPS Models

Abstract. Debugging Cyber-Physical System (CPS) models is a cumbersome and costly activity. CPS models combine continuous and discrete dynamics – a fault in a physical component manifests itself in a very different way than a fault in a state machine. Furthermore, faults can propagate both in time and space before they can be detected at the observable interface of the model. As a consequence, explaining the reason of an observed failure is challenging and often requires domain-specific knowledge.

In this paper we propose CPSDebug, a novel approach that combines testing, specification mining, and failure analysis, to automatically explain failures in Simulink/Stateflow models. In particular, we address the hybrid nature of CPS models by using different methods to infer properties from continuous and discrete state variables of the model. We evaluate CPSDebug on two case studies, involving two main scenarios and several classes of faults, demonstrating the potential value of our approach.

Keywords. Cyber-Physical Systems, Testing, Debugging, Model-based Development, Property Mining, Failure Explanation.

The content of this chapter has been published: E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, D. Ničković: CPSDebug: Automatic Failure Explanation in CPS Models, *International Journal on Software Tools for Technology Transfer*, STTT 2021, pp 1-14. DOI:10.1007/s10009-020-00599-4

CHAPTER 4

CPSDebug: A Tool for Explanation of Failures in Cyber-Physical Systems

Abstract. Debugging Cyber-Physical System models is often challenging, as it requires identifying a potentially long, complex and heterogenous combination of events that resulted in a violation of the expected behavior of the system. In this paper we present CPSDebug, a tool for supporting designers in the debugging of failures in MATLAB Simulink/Stateflow models. CPSDebug implements a gray-box approach that combines testing, specification mining, and failure analysis to identify the causes of failures and explain their propagation in time and space. The evaluation of the tool, based on multiple usage scenarios and faults and direct feedback from engineers, shows that CPSDebug can effectively aid engineers during debugging tasks.

Keywords. Cyber-Physical Systems, Model-based Development, Testing, Specification Mining, Debugging, Failure Explanation.

The content of this chapter has been published: E. Bartocci, N. Manjunath, L. Mariani, C. Mateis, D. Ničković, F. Pastore: CPSDebug: A Tool for Explanation of Failures in Cyber-Physical Systems, *ACM SIGSOFT International Symposium on Software Testing and Analysis*, ISSTA 2020, Los Angeles/Virtual, USA, July 18-22, 2020, pp 569-572. DOI:10.1145/3395363.3404369

CHAPTER 5

Adaptive Testing for Specification Coverage

Abstract. Ensuring correctness of cyber-physical systems (CPS) is an extremely challenging task that is in practice often addressed with simulation-based testing. Formal specification languages, such as Signal Temporal Logic (STL), are used to mathematically express CPS requirements and thus render the simulation activity more systematic and principled. We propose a novel method for adaptive generation of tests with specification coverage for STL. To achieve this goal, we devise cooperative reachability games that we combine with numerical optimization to create tests that explore the system in a way that exercise various parts of the specification. To the best of our knowledge our approach is the first adaptive testing approach that can be applied directly to MATLAB™ Simulink/Stateflow models. We implemented our approach in a prototype tool and evaluated it on several illustrating examples and a case study from the avionics domain, demonstrating the effectiveness of adaptive testing to (1) incrementally build a test case that reaches a test objective, (2) generate a test suite that increases the specification coverage, and (3) infer what part of the specification is actually implemented.

Keywords. Adaptive Testing, Cooperative Games, Cyber-Physical Systems, Falsification-based Testing, Signal Temporal Logic.

The content of this chapter is published: E. Bartocci, R. Bloem, B. Maderbacher, N. Manjunath, D. Ničković: Adaptive Testing for Specification Coverage, *Computing Research Repository (CoRR) in arXiv* DOI:arXiv:2010.06674v2 and is under submission

CHAPTER 6

Production Tests Coverage Analysis in the Simulation Environment

Abstract. In the semiconductor industry, field returns have a negative impact with large costs and potential loss of reputation. As a consequence, a good coverage of the production tests with respect to the common manufacturing defects is essential to ensure the quality of the product to be delivered. Defect simulation is imperative to obtain coverage, however long simulation duration of the production tests can be a huge obstacle. Hence, there is an emergent need for novel methodologies to obtain coverage analysis of Analog and Mixed-Signal (AMS) chip production tests. In this paper, we address several aspects that are necessary to develop such a methodology. We first propose a method to identify a fault model that mimics the common manufacturing defects and extract all such faults from the Device-under Test (DUT) layout, we then develop a test ordering procedure that for a given fault selects the test from an existing test suite that is the most likely to detect the fault. The test ordering technique allows to avoid the execution of many tests during the coverage analysis and thus save considerable amounts of simulation time.

We demonstrate the applicability and efficiency of the resulting techniques on an AMS design from Infineon Technologies AG.

Keywords. Coverage Analysis, Analog and Mixed-signal Validation, Production Testing

The content of this chapter has been published: N. Manjunath, D. Haerle, S. Sabanal, H. Eichinger, H. Tauber, A. Machne, C. Manthey, M. Väänänen, R. Grosu, D. Ničković: Production Tests Coverage Analysis in the Simulation Environment, *IEEE International Test Conference*, ITC 2018, Phoenix, USA, October 29 - November 1, 2018, pp 1-7. DOI:10.1109/TEST.2018.8624870

CHAPTER 7

Conclusion and Future Work

7.1 Conclusion

In this thesis, we addressed the challenges in fault-based analysis of CPS. We introduced theoretical concepts, described methodological frameworks and provided practical solutions for various types of fault analysis:

- Fault localization and failure explanation in CPS models, and
- Test analysis and generation for fault localization and failure explanation.

The outcomes of this thesis were implemented in prototype tools and applied to both publicly available, but also industrial case studies. This thesis provides necessary means for systematic and automated analysis of faults in CPS, thus facilitating the tedious debugging activities in the development process.

7.2 Future Work

We belive that the outcomes of this thesis do not only provide solutions to the problem of fault analysis in CPS, but also open many new research opportunities.

In this thesis, we noticed that the quality of explanations depended on the quality of test suites. We addressed this problem by devising the adaptive testing method with specification coverage. We believe that this is not the only plausible solution to this problem and that *testing for explaining* is a topic that did not receive sufficient attention in the research community. Inspired by falsification testing, we plan to investigate test generation methods that are specifically guided by the quality of test explanations.

In the current setting, CPSDebug uses one failing test to explain the failures. We plan to study whether presence of multiple failing tests can improve failure explanation. Fault injection was

performed manually in this work. To evaluate the effectiveness of CPSDebug by performing systematic experiments, development of an automatic fault injection procedure would be necessary.

Specification coverage of sizeable and complex CPS demands for a large number of simulation budget. While the adaptive testing as presented in Chapter 5 performs simulations serially, the algorithm enlists parallelization to perform multiple simulations concurrently. Implementation of the testing approach to execute several input prefixes simultaneously would speed up the execution of the algorithm significantly and render this approach more applicable.

We develop a heuristic to perform test ordering that aids in obtaining quicker fault coverage in Chapter 6. A complementary approach that could be pursued is by ordering faults on their likelihood or a combination of both. Critical regions of the circuit layout can be defined based on several aspects of the design, technology and application. Extracted faults from this region may be prioritized over the others.

Finally, we studied in this thesis the problem of fault analysis. The outcomes of our analysis can greatly help the engineer to debug the CPS model. The debugging process remains nevertheless manual. We plan to investigate the ambitious problem of model repair, the automatic debugging of the model, as the natural next step building on top of the results in this thesis.

List of Figures

1.1 Illustration of a typical Cyber-Physical System (CPS) 2
1.2 Overview of *Fault-based Analysis* 6
1.3 System diagnostics procedure overview – monitoring and trace diagnostics. . . . 13
1.4 System diagnostics procedure overview – model slicing. 13
1.5 System diagnostics procedure overview – spectrum-based fault localization. . . . 14
1.6 Overview of failure explanation framework. 15
1.7 Failure explanation as a sequence of snapshots - part of the first snapshot. 16
1.8 Architecture of CPSDebug. 18
1.9 Adaptive Testing - overview. 19
1.10 Fault injection models. 21
1.11 A segment of the fault database. 21
1.12 Combining distance and activity measures. 22
1.13 (a) Circuit schematic of an NMOS Inverter, (b) Graph translation of an NMOS Inverter with computed weights and normalized distance. 23
1.14 Structure of the thesis: reference of the papers to chapters. 24

Acronyms

AI Artificial Intelligence. 1, 2, 6

AMS Analog and Mixed-Signal. 20, 23, 45

CPS Cyber-Physical System. 1, 2, 4–6, 8, 9, 12, 14, 24, 47, 49

CPSs Cyber-Physical Systems. 1, 6, 9

DOT Defect-oriented Testing. 10

DUT Device-under Test. 45

FETs Field-effect Transistors. 22

ICs Integrated Circuits. 3, 21

IP Intellectual Property. 8

MBD Model-based Design. 3, 6, 12

MITL Metric Interval Temporal Logic. 12

MTL Metric Temporal Logic. 9, 12

NMOS N-type Metal-Oxide-Semiconductor. 22, 23, 49

PMOS P-type Metal-Oxide-Semiconductor. 22

PSO Particle Swarm Optimization. 11, 20

RTA Run-Time Assurance. 8

RTAMT Real-Time Analog Monitoring Tool. 8, 11, 18

RV Runtime Verification. 8

SBFL Spectrum-based Fault Localization. 10, 14

SPICE Simulation Program with Integrated Circuit Emphasis. 9, 22

SRAM Static Random Access Memory. 4

STL Signal Temporal Logic. 8, 9, 12, 14, 17, 19, 24

SUT System Under Test. 8–10, 14, 19–21, 23

TkT Timed k-Tail. 11, 17, 18

V&V Verification and Validation. 4, 7