Czech Technical University in Prague
Faculty of Information Technology
Department of Digital Design

Automated Testing of Models of Cyber-Physical Systems

by

Tomáš Apeltauer

A Doctoral Study Report submitted to
the Faculty of Information Technology,
Czech Technical University in Prague

Doctoral degree study programme: Informatics

Prague, September 2018

Supervisor:
doc. Dipl.-Ing. Dr. techn. Stefan Ratschan
Department of Digital Design
Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9
160 00 Prague 6
Czech Republic

Abstract

Area of verification...

Keywords:
keyword1, keyword2, keyword3, keyword4, keyword5.

Acknowledgement

This research has been partially supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS ..., and by the ...

Contents

List of Figures

List of Tables

Abbreviations

CPS	Cyber-Physical Systems
MBD	Model-Based Development
MTL	Metric Temporal Logic
GPS	Global Positioning System
EMBS	Electro-Mechanical Braking System
MBT	Model-Based Testing
FSM	Finite-State Machine
UML	Unified Modeling Language
SAT	Boolean satisfiability
MCDC	Modified Condition Decision
LTL	Linear Temporal Logic
CTL	Computation Tree Logic
MTL	Metric Temporal Logic
iPendulum	Inverted Pendulum
PID	Proportional Integral Derivative

Chapter 1

Introduction

At the beginning of the 21.st century, human race enters into a new era of industrial revolution generally called Industry 4.0 [38]. So far humans used computers and automation to make industrial processes as efficient as possible. But now the technology allows us to create Cyber-Physical Systems (CPS) and integrate them into the industrial process. All the work thus can be passed to fully autonomous devices that man will only oversee, giving us more space for something humans do the best, intellectual creativity. However, if we are to put all the work on CPS, we must make sure that such devices will be as safe and secure as possible.

1.1 Motivation

CPS are specific in their structure [35]. They contain both a discrete unit and continuous unit. Most of such devices fit into category of embedded systems because they monitor variables of the physical world (temperature, pressure, chemical composition, speed, and so forth) and also react based on the values of such variables. The manufacturing process of CPS is still very expensive. To address this issue, many companies all over the world use Model-based design (MBD) for prototyping and upgrading their products. MBD puts much emphasis on the creation of the digital model of CPS.

An important part of such a process is model verification. Usually, an engineer has a list of requirements that CPS must comply in order to be labeled as safe an secure. The manual process of verification of models of CPS is very time consuming and limited. That is why several verification tools have been developed to help companies by running automated tests simulations against a set of requirements. These tools use complex search algorithms to find a simulation trajectory that violates given requirement(s). It is not a trivial task, because of the conjunction of discrete and continuous worlds. For example, the continuous dynamic of rotating car wheel can be clearly described using a set of differential equations, but when an Anti-lock braking system discrete controller locks the wheel, none of these equations holds.

1.2 Problem Statement

In addition to a vast complexity of behavior of CPS, verification tools treat models of CPS only as black boxes, not considering its inner structure. This approach has its limitations. Testing a model without the knowledge of its inner structure will never be as effective as if we would test it with structure and contextual analysis. That is why we aim to propose new algorithms for automated testing of models of CPS with the consideration of their inner structure.

The first objective of our research is to gather useful models of CPS, preferably the ones used in the industrial area. Then we focus on the verification process itself and the tools generally used in practice. We try to find use cases when the performance of conventional or academic tools is insufficient and enhance them by providing deep model analysis information.

1.3 Related Work/Previous Results

This research is based upon the work of the research group from Cyber-Physical Systems Laboratory at Arizona State University [4, 12, 36, 17]. They created a verification tool named S-TaLiRo [4] and also presented their metric for effective searching for simulation trajectories when verifying a model against given specification [12].

When working with specification and requirements, we use Metric Temporal Logic (MTL) developed by Ron Koymans [21]. This way of specifying demands on CPS is suitable because MTL allows us to formulate restrictions as: ”There is a maximum number of time units so that each occurrence of an event E is responded to within this bound”.

Our efforts were presented on the student seminar PAD 2017 [TA.1] where we gathered much valuable feedback. This helped us to concretise our goals and form reasonable milestones.

1.4 Structure of the Report

The report is organized into ... chapters as follows:

1. *Introduction*: Describes the motivation behind our efforts together with our goals. There is also a list of contributions of this report.

2. *Background and State-of-the-Art*: Introduces the reader to the necessary theoretical background and surveys the current state-of-the-art.

3. *Overview of Our Approach*: ...

4. *Prelimitary Results*: ...

5. *Conclusions*: Summarizes the results of our research, suggests possible topics of your doctoral thesis and further research, and concludes the report.

Chapter 2

Background and State-of-the-Art

In the last decade, we have seen a dramatic decrease in the cost of certain computation technologies and such phenomenon gave birth of a new family of embedded control systems that are much better prepared for fluent, realistic interaction with the continuous physical world around them. For systems that combine the physical world around us with the world of cybernetics, we use a term cyber-physical system. Although certain forms of CPS have been in industrial use since the 1980s, only recently has the technology for processors, wireless communication, and sensors matured to allow the production of components with impressive capabilities at a low-cost [1].

Advance in the field of Cyber-physical systems will bring us closer to the usage of high-speed, low-cost, and real-time embedded computers in technologies like electric networks that employ advanced monitoring [39], networked autonomous vehicles [23] or prosthesis like neural-controlled artificial leg [40]. CPS are a research priority for both, government agencies (National Science Foundation) and industry (automotive, avionics, medical devices).

2.1 Cyber-Physical Systems

The concept of a cyber-physical system is a generalization of embedded systems. An embedded system consists of hardware and software integrated within a mechanical or an electrical system designed for a specific purpose. As shown in Figure 2.1, CPS consist of a computational unit, sensors, actuators and a physical world which it must observe and react on it. In a CPS the controller consists of discrete software concurrent components, operating in multiple modes of operation, interacting with the continuously evolving physical environment. Examples of onboard sensors include a global positioning system (GPS) receiver, a camera or an infrared thermal sensor [1]. CPS are reactive systems which interact with its environment in an ongoing manner. There is an endless loop of data collection and input evaluation throughout the time.

In comparison to the traditional software development architecture, the creation of CPS differ in the emphasis on the security, confidence, reliability, and performance of the system.

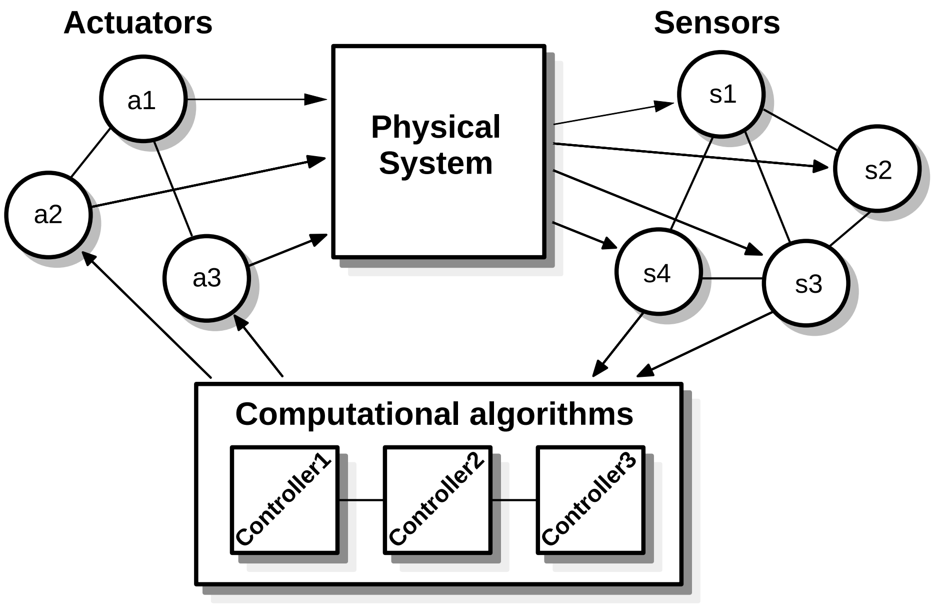

Figure 2.1: Architecture of Cyber-Physical Systems

CPS are often used in areas with many safety requirements (medicine [32], automotive [34], civil engineering [6], avionics, etc.). Apart from embedded systems, CPS will not be operating in a controlled environment and must be robust to unexpected conditions and adaptable to subsystem failures [23]. An example of such systems is an autopilot system used on Airbus aircraft. It is a device used to guide an aircraft without direct assistance from the pilot. Modern autopilots are capable of controlling every part of the flight from just after take-off to landing and are normally integrated with the flight management system.

2.1.1 Reactive Computation

CPS are intended to seamlessly interact with the physical world around in an infinite feedback loop. Such real-time computing can be very challenging because it usually consists of processing a huge amount of inputs and delivering immediate reactions. The traditional computing device process input and produces an output. An example is a program that processes an unsorted list of numbers and returns a sorted list (based on given criteria, e.g., in ascending order).

A reactive system, in contrast, interacts with its environment in an ongoing manner via inputs and outputs. As a typical example of reactive computation consider a program for a cruise controller in a car. CPS are reactive systems [1].

The design of a complex cyber-physical system — especially one with heterogeneous subsystems distributed across networks — is a demanding task. Commonly employed design techniques are sophisticated and include mathematical modeling of physical systems, formal models of computation, simulation of heterogeneous systems, software synthesis, verification, validation, and testing [19].

Embedded system is usually constructed from the physical plant and the controller module. The controller contains a specific algorithm, designed for capabilities and resources of a given embedded system. In industry production area a Model-driven development paradigm has been deployed and successfully tested for development of embedded systems. Unfortunately, when we move from simple programs to more complex software systems and particularly to cyber-physical systems, former design techniques and tools are no longer applicable.

During the process of creation and implementation of an autopilot system, we expect a high level of assurance in the correct behavior of the system. If it would be the other way around, any error can lead to unacceptable consequences such as losses of lives. Systems where safety requirements have higher priority than other design objectives such as performance and development cost are called safety-critical. CPS generally fit into this category. That is why insurance of a system's correctness during design is of utmost importance and sometimes even mandatory because of government regulations.

Former approach of system development is divided into several phases: design, implementation, testing, and validation. The more suitable and practical approach is to write mathematically precise requirements of the desired system, design models of system components and using analysis tools check if the system meets the requirements. Usage of formal models and verification fits for the area of safety-critical applications.

Moreover, that is why a different paradigm for developing CPS was created. It is called Model-Based Development, and it is increasingly adopted by industry [28].

2.2 Model-Based Development

The goal of modeling in system design is to provide mathematical abstractions to manage the complexity of the design. In the context of reactive systems, the basic unit of modeling is a component that interacts with its environment via inputs and outputs [1]. What exactly is a model? A model is defined as a small object, usually built to scale, that represents in detail another, often more massive object: a schematic description of a system, theory, or phenomenon that accounts for its known or inferred properties and may be used for further study of its characteristics.

In their work Jensen et al. 2011 [19] propose a 10-step methodology for developing cyber-physical systems:

1. State the problem
2. Model physical processes
3. Characterize the problem
4. Derive a control algorithm

5. Select models of computation
6. Specify hardware
7. Simulate
8. Construct
9. Synthesize software
10. Verify, validate and test

This approach helps designers break the enormous task of creation of CPS into manageable iterations, which can be repeated if needed. The main goal is to identify any bugs or errors as soon as possible and preferably before the construction phase. For example, we would like to design new experimental electro-mechanical braking system (EMBS). Such system consists of an electric engine, a brake caliper, a brake disc, and a wheel. TODO add figure. The brake disc is connected to the wheel, so that contact between caliper and disc will result in vehicle deceleration [34]. According to the traditional design process, we would propose a system design, implement it in real life, test it manually or randomly and then launch the pilot project. In case of such complicated device as EMBS conventional approach is unsatisfactory. How can we be sure whether the manually or randomly generated scenarios capture the worst-case conditions for the system under test? An alternative approach is to utilize a Model-Based Development (MBD) framework and use the models to simulate the system and intelligently search for corner cases [36].

MBD is gaining increasing acceptance in software and systems engineering practice. This is especially true in the domain of automotive embedded control software design and development, where harsh time-to-market constraints have expedited its adoption. In this methodology, a set of context-specific models are built at the outset of the system development process. These models are executed and tested to validate the requirements, to check the fidelity of the model (e.g., no unreachable elements) and to verify platform-independent design choices [28]. The debugged models serve as a reference during the design, implementation, and testing of the downstream artifacts in the system development process. Methods and tools to support the aforementioned testing-related activities in MBD are studied under the purview of Model-Based Testing (MBT) [10].

Thanks to MBD we can use search-based methods to detect corner cases that violate the safety requirements. We refer to such process as falsification because these methods strive to generate counterexamples that disprove or falsify safety requirements. In the case of our EMBS, an example of a safety requirement could be formulated as As soon as braking is requested, the contact between caliper and disc should occur within 23 ms [34].

2.3 Model-Based Testing

Is it so important to test and verify CPS? We can present a few examples when a perfunctory testing process led to catastrophic failure:

1. The Ariane 5 rocket exploded 36 seconds after the lift-of; the amount lost was of half a billion dollars. The failure was caused by an uncaught exception [7]
2. An error in the software of the baggage handling system delayed of 9 months the opening of the Denver airport, with a loss of 1.1 million dollars per day [11]
3. Intel lost 475 million dollars for replacing Pentium II processors that had a faulty floating-point division unit [20]
4. Toyota recalled some vehicles in 2010 for a bug in the anti-lock brake software [15]
5. Because of an error in the radiation therapy machine Therac-25, some patients were exposed to an overdose of radiation and six of them died [24]

Generally, software testing requires up to 50% of software development costs, and in the case of safety-critical applications, these costs are even higher. This can be reduced by automatization of test execution or test generation. Exhaustive testing is not feasible in practice, and the input domain may be infinite (e.g., avionic system fed with input sequences). Since exhaustive testing is not feasible, we have to select a subset of inputs, and that is why a test generation process for CPS is an active topic in an academic environment [31].

White box testing considers the inner structure of a testing subject. The internal structure/design/implementation of the item being tested is known to the tester. This technique is widely used in software testing where the tester chooses inputs to exercise paths through the code and determines the appropriate outputs. Programming know-how and the implementation knowledge is essential. This method is named so because the software program, in the eyes of the tester, is like a white/transparent box; inside which one sees.

Black box testing does not use information about the internal structure. It observes the testing item only through its interface and considers only the requirements of the system. Tests are only derived from the requirements. MBT is a kind of black box testing. Inputs are applied to the testing item, and the output is observed. The correctness of the output is checked concerning the given expected output. When we are speaking about the safety requirements that describe the expected output, we refer to it as falsification. As stated above these methods attempt to generate counterexamples that disprove or falsify safety requirements, thus disprove the expected output. There are tools such as S-TaLiRo that can automate the process of falsification [4].

We can identify the following families of modeling notations:

2.3.0.1 Transition-based notations

They describe transitions between states of the system. E.g., Finite-State Machine (FSM), Unified Modeling Language (UML) state machines, Statecharts, Labeled Transition Systems, etc.

2.3.0.2 Input-domain notations

They describe the inputs and their constraints. E.g., combinatorial testing, feature modeling.

2.3.0.3 Pre/post notations

They describe the system using some variables and operations. The models specify preconditions that must be satisfied before an operation and post-conditions that must be guaranteed after the operation execution. E.g., Java Modeling Language, Spec#, etc.

2.3.0.4 History-based notations

They describe the allowable traces of the system behavior over time. They are good for describing the interactions among components. E.g., message-sequence charts, UML sequence diagrams, temporal logics.

2.3.0.5 Functional notations

They describe the model as a set of mathematical functions.

2.3.0.6 Operational notations

They describe a system as a set of executing processes. E.g., process algebras, Petri nets, ASMs.

2.3.0.7 Stochastic notations

They describe a probabilistic model of the inputs of the system.

2.3.0.8 Data-flow notations

They model the flow of data (rather than control flow)

The model itself could contain errors. Test generation depends on the notation used. Some approaches are based on:

1. exploration (simulation) of the model
2. logical solvers (Boolean satisfiability (SAT) problem solvers)
3. model checkers (SPIN, NuSMV, etc.) which check more complicated temporal properties

2.3.1 Model checking

Model checking is an automated formal verification technique. It aims to discover whether an abstract description $\mathcal{M}$ of a system satisfies property φ, i.e.,

$$\mathcal{M} \models \varphi \tag{2.1}$$

The technique explores the state space of $\mathcal{M}$ to check whether property φ holds. State-of-the-art model checkers can handle state spaces of about 10^8 to 10^9 states with explicit state-space enumeration. Using clever algorithms and tailored data structures, larger state spaces (10^{20} up to even 10^{476} states) can be handled for specific problems [5].

TODO: a figure of model checking

Model checking is a generally used technique for the verification of the properties of software and hardware systems. Commonly we represent a system properties by modal or temporal logic formulas with the Boolean-valued semantics. However, when we operate in an area of systems whose state space is some general metric space, the model checking problem becomes difficult and in most of the cases undecidable [14].

In case of linear-time logics, the model checking problem is equally difficult as checking satisfiability; that is, it is undecidable for dense-time logics capable of expressing punctuality and EXPSPACE-complete for the discrete-time logics. The running time of the model checking algorithms depends singly exponentially on the size of the implementation and doubly exponentially on the size of the specification formula [2].

For modeling the system, we usually use higher level notations like FSM, UML statecharts, Simulink models, etc. MBD methodologies across different industries use a variety of modeling languages. For the development of embedded control software in the automotive or aerospace domain is Simulink/Stateflow language a popular choice. Simulink/Stateflow language allows modeling both the continuous and discrete dynamics of the system, and because of that, it is a perfect choice for modeling CPS. It also can simulate the system using either discrete or continuous solvers and use a fixed-step or a variable-step [28]. If we are to use a model checking software, we must provide a translation from these higher level notations to the notation of the model checker. If a property φ does not hold, the model checker returns a counterexample acting as a witness of the violation.

2.4 Requirements verification

TODO: if there is time, definition of MCDC coverage.

The first phase of the V process in MBD is usually referred to as Requirements/Specifications. It consists of defining the constraints under which a system operates and is developed. There are different types of requirements:

- User requirements - statements in natural language plus diagrams of the services the system provides and its operational constraints

- System requirements - a structured document setting out detailed descriptions of the system's functions, services, and operational constraints. Defines what should be implemented so may be part of a contract between client and contractor

- Functional requirements - statements of services the system should provide, how the system should react to particular inputs and how the system should behave in particular situations

- Non-functional requirements - constraints on the services or functions offered by the system such as timing constraints, constraints on the development process, standards

- Domain requirements - constraints on the system from the domain of operation

All these categories are then divided into subcategories. Safety requirements are part of Non-functional requirements. They ensure the safety of systems, where safety is defined regarding acceptable loss. Alternatively, we can define safety as the ability of the system to operate without catastrophic failures [33]. There is no such thing as absolute safety [25].

In the case of safety-critical applications (including CPS), safety standards are rather high. For example an avionic standard DO-178B [27] requires complete Modified Condition Decision (MCDC) [9] coverage for the safety level A software. MCDC is also highly recommended for Automotive Safety Integrity Level D software by the ISO 26262 functional safety standard [18] in the automotive domain.

There are several ways how to form requirements. One approach is to use an automatic test generating tool, but since in case of CPS the input domain is infinite, we can only try to achieve complete MCDC coverage, and that does not give us guarantee that we have covered all the corner cases. Another approach is to use previous steps of the MBD process and verify the model against a set of safety requirements. The verification process is typically a simulation. Safety requirements can be generated from formal models or can be defined by the design team. These requirements then have to be translated into temporal logic, because temporal logics permit to describe properties regarding the evolution of the system over time, which comes handy during the simulation.

If we take our EMBS, we can form another safety requirement: The brake caliper velocity upon contact should be less than 2 mm/s to limit jerk. Any kind of an unexpected jerk when braking could lead to injuries when driving a car with EMBS. In their work Fehnker and Ivančić [16], present couple of benchmarks that combine discrete and continuous components and thus can be treated as CPS. They provide them together with the requirements:

2.4.0.1 Navigation benchmark

An object that moves in the $\mathbb{R}^2$ plane. The plane is divided into cells, and there are cells labeled **A** that have to be reached and cells labeled **B** that ought to be avoided.

2.4.0.2 Leak Test benchmark

The benchmark deals with the detection of leaks in a pressurized network. The verification problem is to show that leaking valves are detected properly. Each segment has to comply following:

- If a segment is tested and if none of its upstream valve leaks, then the bubbling should not start
- If a segment is tested and if an upstream valve leaks, then the bubbling should not start. We assume that the model is deadlock-free, and that time can pass
- If the root segment is tested, the test should detect correctly whether or not the downstream valve leaks

2.4.0.3 Room Heating benchmark

It deals with a house with some rooms that are heated by a limited number of heaters. The temperature in each room depends on the temperature of the adjacent rooms, on the outside temperature, and on whether a heater is in the room. The number of heaters is assumed to be smaller than the number of rooms, and each room may have at most one heater. We aim to verify that:

- The temperature in all rooms is always above a given threshold
- All rooms get eventually a heater
- In all rooms, there will be eventually no heater

The last benchmark was used by Fainekos, Annapureddy and Sankaranarayanan in their work [4].

2.5 Metric Temporal Logic

As we stated above, if we use model checkers, our requirements have to be translated into some kind of temporal logic, because we have to test CPS behavior in time (usually using a simulation). Temporal logic is intended for reasoning about situations changing in time. Its semantics makes a clear distinction between the static aspect of a situation, represented by a state, and the dynamic aspect, the relation (in time) between states [21]. It cannot fully capture the concept of metric time, and traditional temporal operators are insufficient for the specification of quantitative temporal requirements (so-called hard real-time constraints). Depending on the nature of the problems we wish to formalize, we use either a first-order temporal logic or the propositional fragment. Back in 90' there was three general way of how to extend the syntax of temporal logic for specifying real-time systems [2]. These were:

1. Trace semantics - sacrifices information about internal structure of a system
2. Interleaving semantics - sacrifices information about the simultaneity of activities
3. Fictitious-clock semantics - sacrifices information about the precise times of activities

Most popular temporal logics are Linear Temporal Logic (LTL) and Computation Tree Logic (CTL). The most suitable logic for models of CPS is Metric temporal logic (MTL).

MTL is a prominent specification formalism for real-time systems [29]. MTL is an extension of temporal logic; temporal operators are replaced by time-constrained versions like until, next, since and previous operators. MTL transforms the traditional temporal operators:

- G (the formula is always true) in computer science represented as
- F (the formula is at least once in the future true) in comp. science represented as
- H (the formula has always been true)
- P (the formula was at least once in the past true)
- Possibly X (next)
- Possibly Y (previous)

And adds:

- D (until)
- E (since)

MTL extends predicate logic with these operators to get a first-order temporal logic. Back in 1990's temporal logics was insufficient for the task of describing the real-time systems, it contained only qualitative temporal operators. There was a huge need for a formal specification method which would provide quantitative timing properties relating occurrences of events. For example:

- Maximal distance between an event and its reaction, e.g., event A is followed by event B within 3 time units (a typical promptness requirement)
- Exact distance between events, e.g., every event A is followed by an event B in exactly 7 time units (timer and time-out)
- Minimal distance between events, e.g., 2 consecutive events A are at least 5 time units apart (assumption about the rate of the input from the environment)
- Periodicity, e.g., an event E occurs regularly with a period of 4 time units

- Bounded response time, e.g., there is a maximum number of time units so that each occurrence of an event E is responded to within this bound

According to Koymans et al. [22], the operators X (next) and Y (previous) lack the abstractness needed to achieve a fully abstract semantics of concurrent programs. It can be shown that the temporal operator until already suffices for expressive completeness over the natural numbers. There is no special need to introduce past operators when working over the natural numbers. However, Koymans et al. state that such operators allow us to form an elegant specification of message passing systems. In their work Lichtenstein et al. [26] show theoretical results that past operators are very useful. Koymans add a distance function d(t, t') which gives us a measure as to how far t and t' are apart in time. This function is transitive, irreflexive and comparable. Also, a structure (trojuhelnik, +, 0) is created, and 6 rules, such as commutativity, associativity, etc. (see Koymans 1990) are defined. By adding more definitions, Koymans came into a definition of a metric point structure T which represents a kind of micro-macro time.

With MTL we can express 5 typical quantitative temporal properties for CPS: Maximal distance: see SotA Exact distance: see SotA Minimal distance: see SotA Periodicity (with period delta): see SotA Bounded response time: see SotA

MTL is linear-time logic that assumes both, the interleaving and fictitious-clock abstractions; it is defined over a point-based weakly-monotonic integer-time semantics. For MTL, the exact complexity of the satisfiability problems is known and independent of interval-based or point-based, synchronous (i.e., strictly-monotonic) or asynchronous (i.e., weakly-monotonic) interpretation: EXPSPACE-complete [2]. However, until now the question of the decidability of full MTL over infinite timed words remained open [30]. Koymans suspect that MTL has a clean, complete, finite axiomatization, which is yet to be found [21]. He also does one more notable thing when defining MTL. Koymans proves that "All first-order sentences over linear orders are definable in metric temporal logic."

2.6 Verification process

The evaluation of whether or not a product, service or system complies with a regulation, requirement, specification, or imposed condition [8].

2.7 Simulink and Stateflow

TODO see section in [S. Mohalik, A. A. Gadkari, A. Yeolekar, K.C. Shashidhar, and S. Ramesh. Automatic test case generation from Simulink/Stateflow models using model checking. Softw. Test. Verify. Reliab. 24, 2, 2014, 155-180]

2.8 Previous Results and Related Work

Chapter 3

Overview of Our Approach

In our research, we address to focus on the lack of sophisticated algorithms that would help with automated testing during the MBD as described in 2.3. To our knowledge, for the higher level notations like Simulink, there are only algorithms which treat the models as a black box. The inner structure of the model is not taken into consideration in any way. We aim to create algorithms that will fill this gap and enhance the automated testing process of models of CPS by fully exploiting information stored in the structure of models. Our research plan is divided into goals:

- Collaborate with the industry companies and gain access to models of CPS commonly used in practice
- Examine tools for automated testing of models of CPS and determine their drawbacks
- Develop new approaches and algorithms for automated testing of models of CPS

We have established a collaboration with the Czech company HUMUSOFT, spol. s r.o. Their areas of expertise include control systems, technical computing, model-based design, and business process simulation. Activities of the company cover both development of own products and solutions and representing other IT companies offering complementary products and services on the local market. HUMUSOFT is a reseller of MathWorks, Inc., U.S.A., for the Czech Republic and Slovakia. They provide marketing and value-added services for technical computing and simulation software MATLAB and Simulink.

We have acquired a model of EV powertrain test bench at Roztoky Science and Technology Park. The powertrain consists of a battery pack, a power inverter using DTC and an induction motor. For each of these components, there is a mathematical description, the methodology of system identi

cation and a model validation by comparing simulation results with real device measurement. There is a control algorithm of the test bench to simulate the EV drive along a track with known altitude by the defned speed of the vehicle in respect to drive resistance forces. Control algorithm includes safety features to avoid test bench failure and is implemented in dSpace DS1103. The communication between actuators is established by the CAN communication protocol and the RS232 communication protocol.

We have joined our forces with Prof. Li Qiu from The Hong Kong University of Science and Technology and work together on a model of Inverted Pendulum (iPendulum). The system consists of a cart and a rod. The cart, with a mass Mc, slides on a stainless steel shaft and is equipped with a motor. A rod, attached with a ball, is mounted on the cart whose axis of rotation is perpendicular to the direction of the motion of the cart. The rod has an evenly distributed mass Mp and a length L. The ball, with a mass Mb, can be regarded as a mass point. The card position x(t) and the pendulum angle theta(t) can be measured. The input is the force f(t) applied on the cart.

For our experiments, we are using MATLAB/Simulink software (R2017b 64-bit), together with Stateflow package as described in TODO add SotA chapter about it. We use a cloud Windows Embedded 8 x64 OS with Intel(R) Xeon(R) CPU E5-2620 v3 @ 2.40Ghz and 8GB RAM. We measure the performance of the S-TaLiRo tool during the falsification process for different models.

S-TaLiRo searches for trajectories of minimal robustness in Simulink/Stateflow simulations. It uses randomized testing based on stochastic optimization techniques (Monte-Carlo, Ant-Colony optimization, etc.) [3]. It searches for counterexamples to Metric Temporal Logic (MTL) properties for CPS. This goal is achieved by a minimization of a robustness metric [14]. In its core, S-TaLiRo uses a combination of stochastic sampling together with Simulink simulation and a typically a certain form of optimization. This way it aims to find the smallest robustness for a model which is desirable because traces with lower robustness value are closer in the distance to falsifying traces. If the tool detects negative robustness, we acquire a trace which falsifies temporal logic properties. We refer to it as a witness trajectory. Robustness is calculated by TaLiRo module, but the computation is based on the results of convex optimization problems used to compute signed distances.

In order to run different experiments on different CPS and other real-time systems, we have to adapt to the usage of MTL in S-TaLiRo tools. Traditional MTL operators are used in S-TaLiRo MATLAB scripts according to Tab. 3.1. For each of these operators, we can also specify time bounds $[ab\}$ where a and b are non-negative integer values, and we use the round bracket for b when b is infinity, else use TODO find the name of chlupata zavorka. Values of a,b are lower/upper bounds not on simulation time, but the number of samples. The actual sample time constraints can be derived from the sampling value of the "Rate Transition" block. TODO: find what block it is.

A few examples of MTL specification in S-TaLiRo:

- Bounded response time - Always between 3 to 5 samples in the past 'p1' implies eventually 'p2' within 1 sample: $phi_1 = [.]_{[}3, 5](p1- ><>_{[}\ 0, 1]p2);$

- Until - 'p1' is true until 'p2' becomes true after 4 and before 7 samples: $phi_2 = p1U_{[}4, 7]p2;$

Language representation	MTL symbol	S-TaLiRo
not	MTL not	!
and	MTL and	/\
or	MTL and	\/
if ... then ...	MTL	$\rightarrow$
if and only if	MTL	$< - >$
it is always going to be the case	G (diamond)	[]
at least once in the future	F (square)	<>
it has always been the case	H	[.]
at least once in the past	P	$< . >$
phi will be true until a time when theta is true	until	U
phi has been true since a time when theta was true	since	S
phi has to hold at the next state	next	X
phi had to hold at the preceding state	previous	P

Table 3.1: Metric Temporal Logic operators in context of the S-TaLiRo tool.

- Eventually - 'p1' eventually will become true between 1 and 6 samples: $phi_3 =<>_{[}$ $1, 6]p1$;

In MATLAB scripts for S-TaLiRo, we have to set up at least one φ property of a system that we model in order to run a falsification process. For example in case of a simple two degree-of-freedom Proportional Integral Derivative (PID) controller for setpoint tracking (TODO add reference to a figure) if we want to specify that a predicate p1 holds always, it would look like this:

```
1 model = 'dc_pid_2dof';
2 time = 50;
3 cp_array = 4;
4 input_range = [25 60];
5 X0 = [];
6 phi = '[]p1';
7 pred.str = 'p1';
8 pred.A = [1];
9 pred.b = [0.3];
```

We can see that in addition to the definition of the MTL requirement there is also a definition of the predicate on lines 7 - 9. On line number 7 is simply the definition of the name of the predicate. This can be any name we choose, and it can contain numeric digits, but it has to start by a lowercase letter (e.g., isGateOpen2). Next two lines define the condition which is then represented by a predicate. It uses two variables TODO variables into italic or something pred. A and pred.b which are then translated into an equation TODO put into an equation mode $A.x <= b$. As S-TaLiRo treats the models of CPS only as a black box, this condition is related to the output variable(s). In our model under test, we link the desired information channels into outputs, and these outputs are then observed by S-TaLiRo tool. Suppose we have our model of a two-degree of freedom PID controller for setpoint tracking and only output from such model is the shaft speed of the motor. Then we can define our predicate as something like "The the shaft speed of the motor will always be less than or equal to 0.30." which is exactly what we can read from the lines 8 - 9. If we fit our values into the equation, we get: $1.x <= 0.3$. We can also define multiple requirements on multiple output variables, so in the end, we end up with matrices of equations for each predicate. We can define multiple predicates.

We see much potential in going beyond and observe not only the output variables but also the way how they are composed and the values of the variables that contribute to this composition. Such an approach inevitably includes an analysis of the inner structure of the models, and it is components. Because we are talking about a modular higher level notation, where a user can create its components, we are facing an uneasy task.

Chapter 4

Prelimitary Results

Here we will present experiments using the tools and approach we discussed in previous chapters. We are using S-TaLiRo "A tool for temporal logic falsification" to test acquired models of CPS. The testing is based upon falsification process of properties that define or restrict the behavior of given CPS. All properties are written in MTL so a model of CPS can be evaluated in time using MATLAB/Simulink simulation process.

We aim to discover cases when the performance of S-TaLiRo is insufficient for practical use, and thus this tool should be replaced/enhanced in order to handle falsification for a wide range of models of CPS.

4.1 Air-Fuel Ratio Control System

The first system that we used for our experiments was a model of Air-Fuel Ratio Control System. The model represents a fuel control system for a gasoline engine. The system is highly robust in that individual sensor failures are detected, and the control system is dynamically reconfigured for uninterrupted operation. Traditional signal flow is handled in Simulink while changes in control configuration are implemented in Stateflow.

The air-fuel ratio is computed by dividing the air mass flow rate (pumped from the intake manifold) by the fuel mass flow rate (injected at the valves). The ideal (i.e., stoichiometric) mixture ratio provides a good compromise between power, fuel economy, and emissions. The target air-fuel ratio for this system is 14.6. Typically, a sensor determines the amount of residual oxygen present in the exhaust gas. This gives a good indication of the mixture ratio and provides a feedback measurement for closed-loop control. If the sensor indicates a high oxygen level, the control law increases the fuel rate. When the sensor detects a fuel-rich mixture, corresponding to a very low level of residual oxygen, the controller decreases the fuel rate.

There are four fault switches for simulation of a sensor fault and also a throttle command input. If there is 1 on the switch input, then sensor readings equal to zero, or in case of exhaust gas to 12. We have connected S-TaLiRo to all switches and throttle input and let it watch over air/fuel ratio by sending it as an output from the model to the MATLAB workspace. We defined a simple MTL requirement: "Air/Fuel ratio has to stay under 20". There are no initial conditions, and we restricted the input signal of the throttle from 10 to 20, and it will have 24 control points. We also changed the interpolation function for the signal to linear interpolation in order to mimic the original input throttle signal as much as possible. All four switches range from 0 to 1. Each simulation run will take 50-time samples. For Formally it is written as:

```
time = 50;
init_cond = [];
input_range = [10 20; 0 1; 0 1; 0 1; 0 1];
cp_array = [24 2 2 2 2];

phi = '[]p';
pred.str = 'p';
pred.A = [1];
pred.b = [20];
```

S-TaLiRo was able to falsify given requirement by the initial sample with initial robustness value -214.7867 [13]. It ran for 17.8684 seconds. We have used variable-step simulation with default ode45 solver.

TODO: link to the plots that will be in the appendix

It is clear that such trivial MTL restriction is too simple for S-TaLiRo. We will continue experimenting with more complex requirements, and we will explore S-TaLiRo's performance on other models of CPS (e.g., the EV powertrain model, EMBS model, Missile Guidance System model, etc.).

Chapter 5

Conclusions

Automated testing of complex systems such as CPS, is a hot topic in the industry area. We focus on the tools and algorithms that are available to the companies who aim for the next industry revolution and thus need a robust and solid wrokflow which enables them to produce new generation of smart devices. In order to deliver the safety-critical system to the market they have to comply with many laws and strict regulations. This cannot be done without clever and efficient testing during the MBD process.

Although there exists a MBD framework for building CPS, we still lack the ecessary tools to automate the whole process. In the last decade we have build enough theory to easily produce simple real-time systems, but we still heavily depend on prototype testing. In order to achieve more efficient and competetive production we need to utilize MBT together with model checking, requirements verification and higher level modeling notations.

We see a lot of potential in analyzing the inner structure of models of CPS and usage of this information for a better automated testing process. We have similar interests as do researchers Georgios Fainekos and Sriram Sankaranarayanan, but we intend to go beyond and come up with better algorithms for MBT.

We experimentally evaluated performance of the S-TaLiRo TODO: by a macro tool on a Air-Fuel Ratio Control System (see Chapter 3) and proposed next steps. We plan to put also other models of CPS under the test and possibly negotiate an access to models used in the industry.

5.1 Proposed Doctoral Thesis

Title of the thesis:

Automated Testing of Models of Cyber-Physical Systems

The author of the report suggests to continue in a reasearch of the following:

5.1.1 Disadvanteges of black box approach to the Model Based Testing process

We have already started this research by using an S-TaLiRo tool and finding its limits. We are going to map areas where similar tools perform well and also areas where these tools are not usable. We will analyze the connection between black box approach to the automated testing and the limits these tools have.

5.1.2 A white box approach to the Model Based Testing process

With the knowledge of the areas where black box approach is insufficient, we can start a research on innovative white box approach that will take the inner structure of the models into consideration. This approach will be tested and then compared to the black box approach.

5.1.3 Hybrid Dynamic Systems

Since there is also mathematically precise framework of Hybrid Dynamic Systems [37] and there exist a lot of algorithms for verification of such systems, we will focus on an adoption of some of the principles of these algorithms and their usage in our white box approach.

5.1.4 Industry modeling standards in context of model checking

In the industry companies use a lot of different standards and form their own components. These components are then used in models of CPS which makes the automated testing process harder. We will focus on gathering industrial standards, workflows and methodics in order to be able to adjust our inovative white box testing approach for a practical use in the industry area.

Bibliography

[1] Rajeev Alur. *Principles of cyber-physical systems.* The MIT Press, Cambridge, Massachusetts, 2015.

[2] Rajeev Alur and Thomas A. Henzinger. Logics and models of real time: A survey. In *Proceedings of the Real-Time: Theory in Practice, REX Workshop*, pages 74–106, Berlin, Heidelberg, 1992. Springer-Verlag.

[3] Y. S. R. Annapureddy and G. E. Fainekos. Ant colonies for temporal logic falsification of hybrid systems. In *IECON 2010 - 36th Annual Conference on IEEE Industrial Electronics Society*, pages 91–96, Nov 2010.

[4] Yashwanth Annapureddy, Che Liu, Georgios Fainekos, and Sriram Sankaranarayanan. S-taliro: A tool for temporal logic falsification for hybrid systems. In *Proceedings of the 17th International Conference on Tools and Algorithms for the Construction and Analysis of Systems: Part of the Joint European Conferences on Theory and Practice of Software*, TACAS'11/ETAPS'11, pages 254–257, Berlin, Heidelberg, 2011. Springer-Verlag.

[5] Christel Baier and Joost-Pieter Katoen. *Principles of Model Checking (Representation and Mind Series).* The MIT Press, 2008.

[6] Heider Berlink, Nelson Kagan, and Anna Helena Reali Costa. Intelligent decision-making for smart home energy management. *Journal of Intelligent & Robotic Systems*, 80(1):331–354, Dec 2015.

[7] Inquiry Board. Ariane 5 - flight 501 failure. In *Inquiry Board repor*, Jul 1996.

[8] Barry W. Boehm, editor. *Software Risk Management.* IEEE Press, Piscataway, NJ, USA, 1989.

[9] J. J. Chilenski and S. P. Miller. Applicability of modified condition/decision coverage to software testing. *Software Engineering Journal*, 9(5):193–200, Sept 1994.

[10] S. R. Dalal, A. Jain, N. Karunanithi, J. M. Leaton, C. M. Lott, G. C. Patton, and B. M. Horowitz. Model-based testing in practice. In *Proceedings of the 21st International Conference on Software Engineering*, ICSE '99, pages 285–294, New York, NY, USA, 1999. ACM.

[11] A.J.M. Donaldson. A case narrative of the project problems with the denver airport baggage handling system. Technical Report TR 2002-01, School of Computer Sciences, Middlesex University, London, England, 01 2002.

[12] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications. In *Proceedings of the First Combined International Conference on Formal Approaches to Software Testing and Runtime Verification*, FATES'06/RV'06, pages 178–192, Berlin, Heidelberg, 2006. Springer-Verlag.

[13] Georgios E Fainekos and George J Pappas. Robustness of temporal logic specifications for finite state sequences in metric spaces. Technical Report MS-CIS-06-05, Department of CIS, University of Pennsylvania, 05 2006.

[14] Georgios E. Fainekos and George J. Pappas. Robustness of temporal logic specifications for continuous-time signals. *Theor. Comput. Sci.*, 410(42):4262–4291, September 2009.

[15] David Fan, David Geddes, and Felix Flory. The toyota recall crisis: Media impact on toyota's corporate brand reputation. *Corporate Reputation Review*, 16(2):99–117, Jun 2013.

[16] Ansgar Fehnker and Franjo Ivančić. Benchmarks for hybrid systems verification. In Rajeev Alur and George J. Pappas, editors, *Hybrid Systems: Computation and Control*, pages 326–341, Berlin, Heidelberg, 2004. Springer Berlin Heidelberg.

[17] Bardh Hoxha, Adel Dokhanchi, and Georgios Fainekos. Mining parametric temporal logic properties in model-based design for cyber-physical systems. *Int. J. Softw. Tools Technol. Transf.*, 20(1):79–93, February 2018.

[18] Road vehicles–functional safety. Standard, International Organization for Standardization, Geneva, CH, 2011.

[19] J. C. Jensen, D. H. Chang, and E. A. Lee. A model-based design methodology for cyber-physical systems. In *2011 7th International Wireless Communications and Mobile Computing Conference*, pages 1666–1671, July 2011.

[20] Mohammad Ayoub Khan, Mohammad Ayoub Khan, and Abdul Quaiyum Ansari. *Handbook of Research on Industrial Informatics and Manufacturing Intelligence: Innovations and Solutions.* IGI Global, Hershey, PA, USA, 1st edition, 2012.

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

[22] Ron Koymans, Jan Vytopil, and Willem P. de Roever. Real-time programming and asynchronous message passing. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 187–197, New York, NY, USA, 1983. ACM.

[23] E. A. Lee. Cyber physical systems: Design challenges. In *2008 11th IEEE International Symposium on Object and Component-Oriented Real-Time Distributed Computing (ISORC)*, volume 00, pages 363–369, 05 2008.

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

[25] Nancy G. Leveson. *Safeware: System Safety and Computers.* Addison-Wesley Professional, New York, NY, USA, 1995.

[26] Orna Lichtenstein, Amir Pnueli, and Lenore D. Zuck. The glory of the past. In *Proceedings of the Conference on Logic of Programs*, pages 196–218, London, UK, UK, 1985. Springer-Verlag.

[27] Burke Maxey. Cots integration in safety critical systems using rtca/do-178b guidelines. In *Proceedings of the Second International Conference on COTS-Based Software Systems*, ICCBSS '03, pages 134–142, London, UK, UK, 2003. Springer-Verlag.

[28] Swarup Mohalik, Ambar A. Gadkari, Anand Yeolekar, K.C. Shashidhar, and S. Ramesh. Automatic test case generation from simulink/stateflow models using model checking. *Softw. Test. Verif. Reliab.*, 24(2):155–180, March 2014.

[29] Joël Ouaknine and James Worrell. On the decidability of metric temporal logic. In *Proceedings of the 20th Annual IEEE Symposium on Logic in Computer Science*, LICS '05, pages 188–197, Washington, DC, USA, 2005. IEEE Computer Society.

[30] Joël Ouaknine and James Worrell. On metric temporal logic and faulty turing machines. In *Proceedings of the 9th European Joint Conference on Foundations of Software Science and Computation Structures*, FOSSACS'06, pages 217–230, Berlin, Heidelberg, 2006. Springer-Verlag.

[31] Stefan Ratschan. Symbolic-numeric problems in the automatic analysis and verification of cyber-physical systems. In *Proceedings of the 2009 Conference on Symbolic Numeric Computation*, SNC '09, pages 7–8, New York, NY, USA, 2009. ACM.

[32] S. Sankaranarayanan, C. Miller, R. Raghunathan, H. Ravanbakhsh, and G. Fainekos. A model-based approach to synthesizing insulin infusion pump usage parameters for diabetic patients. In *2012 50th Annual Allerton Conference on Communication, Control, and Computing (Allerton)*, pages 1610–1617, Oct 2012.

[33] Ian Sommerville. *Software Engineering.* Pearson, 10th edition, 2015.

[34] Thomas Strathmann and Jens Oehlerking. Verifying properties of an electromechanical braking system. In Goran Frehse and Matthias Althoff, editors, *ARCH14-15. 1st and 2nd International Workshop on Applied veRification for Continuous and Hybrid Systems*, volume 34 of *EPiC Series in Computing*, pages 49–56. EasyChair, 2015.

[35] Muhammad Umer Tariq, Jacques Florence, and Marilyn Wolf. Design specification of cyber-physical systems: Towards a domain-specific modeling language based on simulink, eclipse modeling framework, and giotto. In Florian Noyrit, Susanne Graf, and Iulia Dragomir, editors, *Proceedings of the 7th International Workshop on Model-based Architecting and Construction of Embedded Systems co-located with ACM/IEEE 17th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2014)*, volume 1250 of *CEUR Workshop Proceedings*, pages 6–15, Valencia, Spain, 2014. CEUR-WS.org.

[36] C. E. Tuncali, S. Yaghoubi, T. P. Pavlic, and G. Fainekos. Functional gradient descent optimization for automatic test case generation for vehicle controllers. In *2017 13th IEEE Conference on Automation Science and Engineering (CASE)*, pages 1059–1064, Aug 2017.

[37] Arjan J Van Der Schaft and Johannes Maria Schumacher. *An introduction to hybrid dynamical systems*, volume 251. Springer London, 2000.

[38] L. D. Xu, E. L. Xu, and L. Li. Industry 4.0: state of the art and future trends. *International Journal of Production Research*, 56(8):2941–2962, 2018.

[39] X. Yu and Y. Xue. Smart grids: A cyber–physical systems perspective. *Proceedings of the IEEE*, 104(5):1058–1070, May 2016.

[40] X. Zhang, Y. Liu, F. Zhang, J. Ren, Y. L. Sun, Q. Yang, and H. Huang. On design and implementation of neural-machine interface for artificial legs. *IEEE Transactions on Industrial Informatics*, 8(2):418–429, May 2012.

Publications of the Author

[TA.1] T. Apeltauer *Automatické testování modelů kyber-fyzikálních systémů.* 15. ročník semináře Počítačové architektury & diagnostika, Smolenice, Slovensko 2017.

[TA.2] T. Apeltauer *Automated Testing of Models of Cyber-Physical Systems*, Technical Report at Czech Technical University in Prague, Prague, Czech Republic, 2018 (technical report).

[TA.3] Research team member of SGS17/213/OHK3/3T/18, Czech Technical University in Prague.

Appendix A

…

A.1 …

Section not in the Table of Contents