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

General[TODO delete if no other cathegory]

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

Common Mathematical Functions and Operators

10_2	Numbers' radices are designated with a subscript
$\mathbf{b}$	Vector $\mathbf{b}$
b_i	the i^{th} element of vector $\mathbf{b}$
$\|\|\mathbf{b}\|\|$	Norm of vector $\mathbf{b}$
$\dim \mathbf{b}$	Dimension of vector $\mathbf{b}$
$\mathbf{A}$	Matrix $\mathbf{A}$
$a_{i,\,j}$	Element of matrix $\mathbf{A}$ at the i^{th} row, and the j^{th} column
$\mathbf{A}^{-1}$	Inverse matrix to matrix $\mathbf{A}$
$\mathbf{A}^T$	Transposed matrix to matrix $\mathbf{A}$
$\|\|\mathbf{A}\|\|$	Norm of matrix $\mathbf{A}$
$\operatorname{cond} \mathbf{A}$	Condition number of matrix $\mathbf{A}$
$\operatorname{rank} \mathbf{A}$	Rank of matrix $\mathbf{A}$ — how many independent rows/columns it has
$\max\{a,\, b\}$	Maximum of a and b, a when $a \geq b$, b when $a < b$
$\min\{a,\, b\}$	Minimum of a and b, a when $a \leq b$, b when $a > b$
$O(x)$	The big O notation
$\Theta(x)$	The big Θ notation

Mathematical Terminology

Q	Number of prime number modules
M	A product of individual modules $M = \prod_{i=1}^{Q} m_i$
...	...
...	...
...	...
...	...

Miscellaneous Abbreviations

FPU	Floating Point Unit
...	...
...	...
...	...
...	...

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[TODO citation]. So far humans used computers and automation to make industrial processes as efficient as possible. But now the technology allowes 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. But 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 strusture[5[Tariq,Florence]Design Specification of Cyber-Physical Systems]. They contain both a discrete unit and continuous unit. Most of such devices fit into cathegory of embedded systems, because they monitor variables of the physical world (temperature, pressure, chemical composition, speed, etc.) 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 a lot of emphasis on the creation of digital model of CPS.

An important part of such 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. 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 conjuction of discrete and continuous worlds. For example continuous dynamic of rotating car wheel can be clearly described using 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 behaviour of CPS, verification tools treat models of CPS only as black boxes, not considering its inner structure. This approach have 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 industry area. Then we focus on the verification process itself and the tools generally used in practise. 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. [TODO publication citations] They created a verification tool named S-TaLiRo [TODO citation of S-TaLiRo paper] and also presented their own metric for effective searching for simulation trajectories when verifying a model against given specification [citation of robustness metric paper].

When working with specification and requirements we use Metric Temporal Logic (MTL) developed by Ron Koymans [citation]. 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 efforst were presented on the student seminar PAD 2017 [citation] where we gathered a lot of valuable feedback. This helped us to concreticise 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 a 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 physical world around us with the world of cybernetics, we use a term cyber-physical system. Although certain forms of CPS have been in industril use since 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.[Rajeev Alur Principles of CPS]

Advance in the field of Cyber-physical systems will bring us closer to usage of high-speed, low-cost, and real-time embedded computers in technologies like electric networks that employ advanced monitoring [Smart Grids: A Cyber–Physical Systems Perspective, By Xinghuo Yu and Yusheng Xue], networked autonomous vehicles [E. A. Lee, "Cyber Physical Systems: Design Challenges," 978-0-7695-3132-8] or prosthesis like neural controlled artificial leg [On Design and Implementation of Neural-Machine Interface for Artificial Legs, Xiaorong Zhang, Yuhong Liu]. 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.[Rajeev Alur Principles of CPS] An embedded system consists of hardware and software integrated within a mechanical or and electrical system designed for a specific purpose. 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 continuosly evolving physical environment. Examples of on-board sensors include a global positioning system (GPS) receiver, a camera or an infrared thermal sensor.[Rajeev Alur Principles of CPS] 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.

TODO Chart of CPS

In comparison to the traditional software development architecture, the creation of CPS differs in the emphasis on the security, confidence, reliability and performance of the system. CPS are often used in areas with many safety requirements (medicine [cite S-TaLiRo insulin pump], automotive [cite EMB paper], civil engineering [cite some smarthome papers], avionics, etc.). Apart from embedded systems, CPS will not be operating in a controlled environments and must be robust to unexpected conditions and adaptable to subsystem failures. [E. A. Lee, Cyber Physical Systems: Design Challenges, 2008,978-0-7695-3132-8]. 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 consist of processing huge amount of inputs and delivering immediate reactions. The traditional computing device process an input and produces an output. An example is a program that process an unsorted list of numbers and returns a sorted list (based on given criteria, e.g. in an 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.[Rajeev Alur Principles of CPS]

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.[J. C. Jensen, D. H. Chang and E. A. Lee, A model-based design methodology for cyber-physical systems,978-1-4244-9538-2].

Embedded system is usually constructed from the physical plant and the controller module. The controller contains specific algorithm, designed for capabilities and resources of 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 lifes. Systems where safety requirements have higher priority than other design objectives such as performance and developement cost are called safety-critical. CPS generally fit into this

cathegory. That is why ensurance of a system's corectness during design is of utmost importance and sometimes even mandatory because of goverment regulations.

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

And that is why a different paradigm for developing CPS was created. It is called Model-Based Development and it is increasingly adopted by industry.[S. Mohalik, A. A. Gadkari, A. Yeolekar, K.C. Shashidhar, and S. Ramesh. Automatic test case generation from Simulink/Statefow models using model checking. Softw. Test. Verif. Reliab. 24, 2, 2014, 155-180]

2.2 Model-Based Development

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

In their work Jensen et al. 2011 [J. C. Jensen, D. H. Chang and E. A. Lee, A model-based design methodology for cyber-physical systems,978-1-4244-9538-2] 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 enormous task of creation of CPS into manageable iterations, which can be repeated if needed. 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. [Strathmann and Oehlerking in Verifying properties of an electro-mechanical braking system]. According to traditional design process, we would propose a system design, implement it in real life, test it manually or randomly and then launch 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. [Tuncali, Fainekos, Functional Gradient Descent optimization for automatic test case generation for vehicle controllers]

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. [S. Mohalik, A. A. Gadkari, A. Yeolekar, K.C. Shashidhar, and S. Ramesh. Automatic test case generation from Simulink/Statefow models using model checking. Softw. Test. Verif. Reliab. 24, 2, 2014, 155-180] The debugged models serve as reference during the design, implementation and testing of the downstream artefacts 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). [Dalal SR, Jain A, Karunanithi N, Leaton JM, Lott CM, Patton GC, Horowitz BM. Model-based testing in practice. In Proceedings of the 1999 International Conference on Software Engineering, ICSE' 99, Los Angeles, CA, USA, 1999; 285–294.]

Thanks to MBD we are able to 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 case of our EMBS, an example of a safety reqirement could be formulated as: As soon as braking is requested, the contact between caliper and disc should occur within 23 ms.[Strathmann and Oehlerking in Verifying properties of an electro-mechanical braking system]

2.3 Model-Based Testing

Is it really so important to test and verify CPS? We can present a few examples when perfunctory testing process led to a 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 [https://esamultimedia.esa.int/do x-1819eng.pdf]

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 [http://www.eis.mdx.ac.uk/research/SFC/Reports/TR2002-01.pdf]

3. Intel lost 475 million dollars for replacing Pentium II processors that had a faulty floting-point division unit [Khan, M. A., Ansari, A. Q. (2012). Handbook of Research on Industrial Informatics and Manufacturing Intelligence: Innovations and Solutions (pp. 1-662). Hershey, PA: IGI Global. doi:10.4018/978-1-4666-0294-6]

4. Toyota recalled some vehicles in 2010 for a bug in the anti-lock brake software [https://instituteforpr.org/wp-content/uploads/JFGRA-InfoTrend-case-study-ver-2.pdf]

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 [http://sunnyday.mit.edu/papers/therac.p

Generally software testing requires up to 50% of software development costs and in case of safety-critical applications these costs are even higher. This can be reduced by automatization of test execution or test generation. An 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. [TODO citation]

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 tecnique 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 clearly sees.

Black box testing does not use information of 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 corectness of the output is checked with respect to 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 attemp 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. [S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems, Y. S. R. Annapureddy, C. Liu, G. E. Fainekos and S. Sankaranarayanan]

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 by means of some variables and operations. The models specify pre-conditions 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 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 a 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. [Christel Baier and Joost-Pieter Katoen. 2008. Principles of Model Checking (Representation and Mind Series). The MIT Press.]

TODO: figure of model checking

Model checking is 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. But 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. [Fainekos, "Robustness of Temporal Logic Specifications for Finite State Sequences in Metric Spaces", 2006]

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. [myLibrary 22 - TODO old information -¿ verify]

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 modelling 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 modelling both the continuous and discrete dynamics of the system and because of that it is a perfect choice for modelling CPS. It also can simulate the system using either discrete or continuous solvers and use a fixed-step or a variable-step.[S. Mohalik, A. A. Gadkari, A. Yeolekar, K.C. Shashidhar, and S. Ramesh. Automatic test case generation from Simulink/Statefow models using model checking. Softw. Test. Verif. Reliab. 24, 2, 2014, 155-180] 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 witness of the violation.

2.4 Requirements verification

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

First phase of the V process in MBD is usually refered 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 in terms of acceptable loss. Or we can define safety as the ability of the system to operate without catastrophic failures. [Ian Sommerville. 2015. Software Engineering (10th ed.). Pearson.] There is no such thing as absolute safety. [Leveson, N.: Safeware – System, Safety and Computers, Addison-Wesley Publishing Company, Massachusetts, Bonn, 1995]

In case of safety-critical applications (including CPS), safety standards are rather high. For example an avionic standard DO-178B [RTCA/DO-178B. Software Considerations in Airborne Systems and Equipment Certification, RTCA, Washington D.C., USA, 1992.] requires complete Modified Condition Decision (MCDC) [Chilenski JJ, Miller SP. Applicability of modified condition/decision coverage to software testing. Software Engineering Journal 1994; 9(5):193–200.] coverage for the safety level A software. MCDC is also highly recommended for Automotive Safety Integrity Level D software by the upcoming ISO 26262 functional safety standard [ISO 26262: Road Vehicles—Functional Safety. Standard Under Development, International Organization for Standardization, 2011.] 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 doesn't 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 unexpected jerk when braking could possibly lead to injuries when driving a car with EMBS. In their work Fehnker and Ivančić [Fehnker A., Ivančić F. (2004) Benchmarks for Hybrid Systems Verification. In: Alur R., Pappas G.J. (eds) Hybrid Systems: Computation and Control. HSCC 2004. Lecture Notes in Computer Science, vol 2993. Springer, Berlin, Heidelberg], 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 labelled **A** that have to be reached and cells labelled **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 a number of 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 in [Annpureddy, Yashwanth Liu, Che Fainekos, Georgios Sankaranarayanan, Sriram. (2011). S-TaLiRo: A Tool for Temporal Logic Falsification for Hybrid Systems. 6605. 254-257. 10.1007/978-3-642-19835-9 21.]

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. [Koymans, R. Real-Time Syst (1990) 2: 255. https://doi.org/10.1007/BF01995674] 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 were three general way of how to extend the syntax of temporal logic for specifying real-time systems. 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

Defined in [myLibrary 22].

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. 21[Ouaknine MTLdecidability] MTL is an extension of temporal logic; temporal operators are replaced by time-constrained versions like until, next, since and previous operators. [myLibrary 22] 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 [Koymans, Vytopil and de Roever 1983], 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, Vytopil and de Roever 1983] state that such operators allow us to form an elegant specification of message passing systems. [Lichtenstein, Pnueli and Zuck 1985] 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 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 assume 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. [myLibrary 22 - TODO old information -¿ verify] However, until now the question of the decidability of full MTL over infinite timed words remained open. [from the abstract of On Metric Temporal Logic and Faulty Turing Machines] Koy90 suspect that MTL has a clean complete, finite axiomatization, which is yet to be found. [myLibrary 22 - TODO old information -¿ verify] Koymans does one more notable thing when defining MTL. He 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. [Barry W. Boehm (Ed.). 1989. Software Risk Management. IEEE Press, Piscataway, NJ, USA.]

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/Statefow models using model checking. Softw. Test. Verif. Reliab. 24, 2, 2014, 155-180]

2.8 Previous Results and Related Work

Chapter 3

Overview of Our Approach

In our research, we address focus on the lack of sophisticated algorithms that would help with automated testing during the MBD as described in 2.3

The sample Fig. 3.1 shows ...

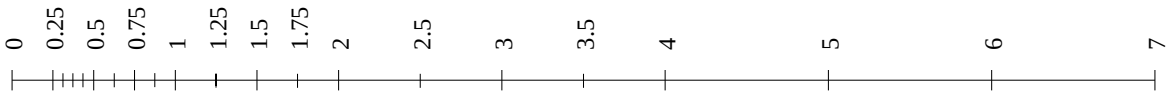

Figure 3.1: Distribution of the floating point numbers. This figure shows a distribution of a sample floating point number set with a precision $t = 3$, and $e_{min} = -1$ and $e_{max} = 3$.

There are two basic floating point data types , as defined by the IEEE 754-2008 [1] standard, are shown in Tab. 3.1.

	Sign [b]	**Exponent** [b]	**Mantissa** [b]	**Prec.** [dig]	**Total** [b]
binary32	1	8	24	8	32
binary64	1	11	53	16	64

Table 3.1: Basic floating point data types.

Chapter 4

Prelimitary Results

4.1 Prelimitary Result 1

4.2 Prelimitary Result 2

4.3 Prelimitary Result 3

4.4 Discussion

4.5 Summary

Chapter 5

Conclusions

5.1 Proposed Doctoral Thesis

Title of the thesis:
TITLE
The author of the report suggests to present the following:

5.1.1 Topic 1

5.1.2 Topic 2

5.1.3 Topic 3

Bibliography

[1] IEEE Computer Society Standards Committee. *IEEE Standard for Floating-Point Arithmetic*. ANSI/IEEE STD 754-2008. The Institute of Electrical and Electronics Engineers, Inc., 2008.

Publications of the Author

[A.1] R. Gortz, F. Tölökö. *On the Carpathian Castle*. Transylvanian Journal of ..., Werst, Romania, 2010.

The paper has been cited in:

- Š. Nováků. *Carpathian Castle Revealed*, International Symposium on Carpathian Legends, 1:319–323, 2010.

[A.2] R. Gortz *Another publication*. 36$^{\text{th}}$ International Conference on ..., pp. 19-24, Štrbské pleso, Slovak Republic, 2010.

Appendix A

...

A.1 ...

Section not in the Table of Contents