Mälardalen University Press Licentiate Theses
No. 182

MODEL CHECKING-BASED SOFTWARE TESTING FOR FUNCTION-BLOCK DIAGRAMS

Eduard Enoiu

2014

School of Innovation, Design and Engineering

ISBN 978-91-7485-166-3
ISSN 1651-9256
Printed by Arkitektkopia, Västerås, Sweden

Abstract

Software testing becomes more complex, more time-consuming, and more expensive. The risk that software errors remain undetected and cause critical failures increases. Consequently, in safety-critical development, testing software is standardized and it requires an engineer to show that tests fully exercise, or cover, the logic of the software. This method often requires a trained engineer to perform manual test generation, is prone to human error, and is expensive or impractical to use frequently in production. To overcome these issues, software testing needs to be performed earlier in the development process, more frequently, and aided by automated tools.

We devised an automated test generation tool called CompleteTest that avoids many of those problems. The method implemented in the tool and described in this thesis, works with software written in Function Block Diagram language, and can provide tests in just a few seconds. In addition, it does not rely on the expertise of a researcher specialized in automated test generation and model checking. Although CompleteTest itself uses a model checker, a complex technique requiring a high level of expertise to generate tests, it provides a straightforward tabular interface to the intended users. In this way, its users do not need to learn the intricacies of using this approach such as how coverage criteria can be formalized and used by a model checker to automatically generate tests. If the technique can be demonstrated to work in production, it could detect and aid in the detection of errors in safety-critical software development, where conventional testing is not always applicable and efficient.

We conducted studies based on industrial use-case scenarios from Bombardier Transportation AB, showing how the approach can be applied to generate tests in software systems used in the safety-critical domain. To evaluate the approach, it was applied on real-world programs. The results indicate that it is efficient in terms of time required to generate tests and scales well for most of the software. There are still issues to resolve before the technique can be applied to more complex software, but we are already working on ways to overcome them. In particular, we need to understand how its usage in practice can vary depending on human and software process factors.

”The good thing about science is that it’s true whether or not you believe in it. That is why it works.”
Neil deGrasse Tyson

Acknowledgments

First and foremost, I would like to thank my three supervisors, Dr Adnan Čaušević, Associate Professor Daniel Sundmark and Professor Paul Pettersson. They been very supportive in the last three years of my studies; always available to provide advice and support when needed. I'd like to thank Associate Professor Cristina Seceleanu for encouraging me to pursue an academic career. I want to give a special thanks to my industrial mentor Ola Sellin for giving me the opportunity to work in Bombardier Transportation. I will forever be indebted to them for all they have given me.

Many thanks to my family for their love and support through the 10 years I've been at university. Thanks to my fiance, Raluca, for believing in me and being there for me through the hard times.

I'd also like to show my gratitude to all my colleagues at Mälardalen University and Bombardier Transportation in Västerås for encouraging me into interesting collaborations, and for offering friendly advice.

Finally, I'd like to thank VINNOVA whose financial support via the ATAC research project, has made this thesis possible.

Eduard Paul Enoiu
Strömsholm, Sweden
October 7, 2014

List of Publications

Papers Included in the Licentiate Thesis [1]

Paper A *Model-based Test Suite Generation for Function Block Diagrams using the UPPAAL Model Checker.* Eduard Paul Enoiu, Daniel Sundmark, Paul Pettersson. In the Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 158 - 167, ISBN: 978-1-4799-1324-4, 2013, IEEE.

Paper B *MOS: An Integrated Model-based and Search-based Testing Tool for Function Block Diagrams.* Eduard Paul Enoiu, Kivanc Doganay, Markus Bohlin, Daniel Sundmark, Paul Pettersson. In the First International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pages 55 - 60, ISBN: 978-1-4673-6284-9, 2013, IEEE.

Paper C *Using Logic Coverage to Improve Testing Function Block Diagrams.* Eduard Paul Enoiu, Daniel Sundmark, Paul Pettersson. In the Proceedings of the 25th IFIP WG 6.1 International Conference on Testing Software and Systems, volume 8254, pages 1 - 16, Lecture Notes in Computer Science, 2013, Springer.

Paper D *Automated Test Generation using Model-Checking: An Industrial Evaluation.* Eduard Paul Enoiu, Adnan Čaušević, Thomas J. Ostrand, Elaine J. Weyuker, Daniel Sundmark, Paul Pettersson. Accepted for Publication in the International Journal on Software Tools for Technology Transfer, 2014, Springer.

[1] The included papers have been reformatted to comply with the thesis layout.

Other Relevant Publications

Enablers and Impediments for Collaborative Research in Software Testing: An Empirical Exploration.
Eduard Paul Enoiu, Adnan Čaušević. Proceedings of the 2014 International Workshop on Long-term Industrial Collaboration on Software Engineering, 2014, ACM.

A Methodology for Formal Analysis and Verification of EAST-ADL Models.
Eun-Young Kang, Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Pierre Yves Schnobbens, Paul Pettersson. International Journal of Reliability Engineering and System Safety, 2013, Springer.

ViTAL : A Verification Tool for EAST-ADL Models using UPPAAL PORT.
Eduard Paul Enoiu, Raluca Marinescu, Cristina Seceleanu, Paul Pettersson. Proceedings of the 17th IEEE International Conference on Engineering of Complex Computer Systems, 2012, IEEE.

Extending EAST-ADL for Modeling and Analysis of System's Resource-Usage.
Raluca Marinescu, Eduard Paul Enoiu. IEEE 36th Annual Computer Software and Applications Conference Workshops (COMPSACW), 2012, IEEE.

A Design Tool for Service-oriented Systems.
Eduard Paul Enoiu, Raluca Marinescu, Aida Čaušević, and Cristina Seceleanu. Proceedings of the 9th International Workshop on Formal Engineering approaches to Software Components and Architectures, 2012, Elsevier.

A SysML Model for Code Correction and Detection Systems.
Stefan Stancescu, Lavinia Neagoe, Raluca Marinescu, Eduard Paul Enoiu. Proceedings of the 33rd International Convention on Information and Communication Technology, Electronics and Microelectronics, 2010, IEEE.

Contents

List of Figures

List of Tables

I

Thesis

Chapter 1

Introduction

To this day software testing is one of the biggest research directions in software engineering. Wong et al. [28] indicated that for 37% of the top scholars in software engineering, their research focus includes *software testing*. As time has progressed software testing research provided a case for technologies, methods, and knowledge invoking changes in companies.

Technological, organisational and economic factors profoundly influence the quality of software testing worldwide. Since the beginnings of software testing, we have tried to address complexity, whilst improving productivity through the use of more smarter techniques and tools. We have progressed from testing software in terms of low-level functionality to automatically generating tests for the system as whole. From structural testing, via data flow testing, to model-based testing, automated test generation and mutation analysis: testing software is arguably becoming more advanced than the software we produce.

1.1 Software Testing

Software testing is an engineering approach to quality assurance having the purpose of analyzing and executing the software in order to find errors [16]. This method often requires a trained tester to perform manual test generation, is prone to human error, and is expensive to use frequently in production. To overcome some of these issues, software testing needs to be performed earlier in the development process and aided by automated tools.

Obviously, the list of impediments and issues related to software testing

is long. This thesis addresses some of these issues. It was conducted within the ATAC (Advanced Test Automation for Complex and Highly-Configurable Software-intensive Systems) project, started in 2012 by 15 European partners. The project aim was to develop, enhance, and deploy high performance methods and tools for automated quality assurance of large and distributed software-intensive systems. The results presented in this thesis were strongly related to the ATAC project.

1.2 Model Checking

Like other engineering disciplines, today's software testing is using models of the system-under-test. Many notations are used for software models, from formal - mathematical descriptions of the software to semi-formal notations such as the Unified Modeling Language (UML). Historically using models to aid software testing has played a minor role in software engineering practice. Within the last decade model-checking has turned out to be a useful technique for generation of test cases from models [10]. A model checker is a tool for formal verification. There are many different efficient model checkers freely available, therefore it is easy to experiment with such an approach. The several different ways model checking has been used for test case generation illustrates its flexibility [26, 27]. Consequently, such an approach is also chosen in this thesis. However, one of the problems in using model-checking for testing industrial software systems is the limited application to domain-specific languages used in practice.

1.3 Safety-Critical Software Development

In safety-critical software development as the complexity of the programs increases, the importance of performing thorough testing and certification becomes evident [3]. Safety-critical and real-time software systems implemented in Programmable Logic Controllers (PLCs) are used in many real-world industrial application domains. One of the programming languages defined by the *International Electrotechnical Commission* (IEC) for PLCs is the *Function Block Diagram* language. Programs developed in Function Block Diagram are transformed into program code, which is compiled into machine code automatically by using specific engineering tools provided by PLC vendors. The motivation for using Function Block Diagram as the target language in this thesis comes from the fact that it is the standard in many industrial PLC systems,

such as the ones in the railway transportation domain. According to a Sandia National Laboratories study [23] from 2007, PLCs are widely used in a large number of industries with a global market of approx. $ 8.99 billion.

1.4 Structural Testing

Depending on the type of software system to be developed, different testing methods and strategies come in many different forms. In order to reason about these techniques, test criteria are used for evaluating the *adequacy* reached by a certain test. A test criterion is formulated using so called *coverage items*. These items should be exercised during testing in order for the criterion to be satisfied. For example, in statement coverage, statements are coverage items [29]. Usually, testers describe the extent to which a criterion is exercised by using the ratio between the number of coverage items exercised in testing and the overall number of coverage items in the software under test.

A test criterion defined on the actual or abstract representation of the software implementation is called a structural test criterion. Examples of structural test criteria include exercising all execution paths or all variable definition-use paths in the software.

In the software engineering process, testing is performed at different levels, e.g., unit, integration and system testing [3]. Basically, testing is performed from the lowest level of software development with functions tested in isolation (Unit Testing) to system or subsystem integration testing of two or more units (Integration Testing and System Testing), where the whole system configuration is incorporated and executed on the intended target hardware. In general, both structural and functional criteria is considered in lower levels of testing. In system-level and integration testing mostly functional criteria are considered because of the architectural-inherent problems for structural criteria.

Some of the structural test criteria investigated in practice with respect to the coverage items are:

- **Statement Coverage.** The most fundamental and most widely used structural test criterion. According to Zhu et al. [29] the statement coverage is satisfied if "*for all nodes n in the flow graph, there is at least one path p such that node n is on the path p*".

- **Branch Coverage.** Widely used because of the similarity to statement coverage. Again, as defined by Zhu et al. [29] the branch coverage is

satisfied if "*for all edges e in the flow graph, there is at least one path p such that p contains the edge e*".

- **Modified Condition/ Decision Coverage (MC/DC).** Used because it is a strict requirement in the safety-critical software development, especially in the railway industry. According to Chilenski and Miller [7], the MC/DC criterion is satisfied if "*every point of entry and exit in the program has been invoked at least one, every condition in a decision in the program gas taken on all possible outcomes at least once, and each condition has been shown to independently affect the decision's outcome.*"

1.5 Thesis Overview

In this thesis, our goal is to help testing practitioners to automatically generate tests for safety-critical software systems developed in Function Block Diagram language. One example of industrial application includes the use of structural coverage which needs to be demonstrated on the developed programs. There has been little research on using coverage criteria for Function Block Diagram programs in an industrial setting. In some cases coverage is analyzed at the code level [9]. Even if at the code level, coverage is used, there is no much use of analysing the generated code because the code generation scheme is not standardised and there is no direct mapping of the code structure to the original Function Block Diagram program. Hence, it is advantageous to propose and evaluate an automated test generation method tailored to Function Block Diagram software.

The following research contributions were included in this thesis:

- *A framework suitable for transforming Function Block Diagram programs to a formal representation of both its functional and timing behavior.* For this, we implemented a transformation to timed automata, a well known model introduced by Alur and Dill [1]. The choice of timed automata as the target language is motivated primarily by its precise semantics and tool support for experimentation. The transformation reflects the characteristics of the Function Block Diagram language by constructing a model which assumes a *read-execute-write* semantics. The translation method consists of four separate steps. The first three steps involve mapping all the interface elements and the existing timing annotations. The latter step produces a behavior for every block in the program. These steps are independent of timed automata and thus

are generic in the sense that they could also be used when translating a Function Block Diagram program to another target language. This allowed us to investigate further a test case generation technique based on model checking.

- *A test generation technique based on model-checking, tailored for logic coverage of Function Block Diagram programs.* There have been a number of testing techniques using model-checkers, e.g., [5, 20, 21]. However, these techniques are not directly applicable to Function Block Diagram programs. Our main goal with this contribution was to show evidence that logic coverage can be used on Function Block Diagram programs based on the transformed timed automata model. This copes with both functional and timing behavior of an Function Block Diagram program. We showed how a model-checker can be used to generate test cases for covering a Function Block Diagram program.

- *A testing tool for safety critical applications and its application on a large scale case study.* The method implemented in the tool and described in this thesis can automatically provide tests and it does not rely on the expertise of a researcher specialized in model checking. The tool provides a straightforward tabular interface to the intended users.

We used the tools and methods included in this thesis in a large case study based on industrial use-case scenarios from Bombardier Transportation AB, showing how the approach can be applied to generate tests. To evaluate the approach, it was applied on real-world programs.

Chapter 2

Research Summary

This chapter presents the research problem tackled in this thesis and lists the research goals relevant to the problem while pointing out the scientific contributions of the thesis including the published papers. To provide a quick overview of the most common topics included in this thesis, Figure 2.1 contains a word cloud that we generated using all scientific papers contributing to this thesis.

2.1 Problem Statement and Research Goals

In software development, test engineers are required to validate the software against their specifications as well as to show that tests exercise, or cover, the structure of the software. Consequently, the use of automated test generation techniques has been proposed by several researchers [18]. The past years have witnessed increasing research within software testing, especially in the automatic creation and analysis of tests given a model and a set of testing goals (i.e., structural or functional). The limited application to real-world industrial projects, however, impacts the transfer of test generation technologies. Thus, there is a need to validate these approaches against relevant industrial systems such that more knowledge is built on how to efficiently use them in practice.

The approach considered in this thesis is the usage of model-checking for automated test generation. Specifically, we focus on testing Function Block Diagram software because it is the standard in many industrial software systems, such as in the railway domain. Although this was considered before by

Figure 2.1: Word cloud generated using the contributions included in this thesis

researchers [10], there are a few practical solutions that can generally be applied and used in an industrial setting.

Based on the above discussion, we identify our general research problem as: *The need to address both structural and functional testing of Function Block Diagram software in an applicable and efficient way.*

In order to refine this general problem, we narrow our focus based on different perspectives. Firstly, we consider that in order to use model-checking for testing, practitioners needs to employ a testing framework equipped with efficient and effective model-checking methods and tools that can be applied for various test purposes. Secondly, software systems, such as in the railway domain, typically require a certain degree of structural coverage which must be demonstrated on the developed software [6].

Therefore, we specify our research problem as an overall goal of our research efforts:

Overall Goal. *To enable the usage of an applicable automated test generation framework for Function Block Diagram software.*

Since this goal is too abstract to be directly addressed, we have further divide it into three more concrete research goals. In order to be able to provide a framework for testing Function Block Diagrams, one needs an expressive

and well-defined technique that would support both structural and functional testing of Function Block Diagrams. A formalization of the Function Block Diagram software is then needed, in order to achieve an unambiguous model that can be formally analyzed. This motivation justifies our first research goal:

RG 1. *Develop a transformation to a formal description of a model for Function Block Diagram software.*

The first research goal is the basis for the next two research goals, in that it provides a model of the Function Block Diagram programs that can be formally analyzed. The next step is to propose and demonstrate the use of a model-checker for testing of Function Block Diagrams, which gives rise to the second research goal as follows:

RG 2. *Develop a model-checking based technique and associated tool support for functional and structural testing of Function Block Diagram software.*

To address the second research goal, we developed a testing technique based on the UPPAAL model checker. Many benefits emerge from developing this method, including the ability to automatically generate test cases for real industrial software systems described in Function Block Diagram language.

To support testers and developers when testing Function Block Diagram programs we have formulated the third research goal as follows:

RG 3. *Evaluate the applicability and usefulness of the proposed framework by testing a real-world software system in an industrial context.*

The last research goal is based on the proposed framework for testing Function Block Diagrams and aims at providing evidence on the efficiency and applicability of the proposed framework.

2.2 Research Methodology

Perkman et al. [19] is distinguishing between three types of collaborative research methodologies between industry and academia: opportunity driven, commercialization-driven and research-driven. In 2012, a research-driven collaboration was established between Bombardier Transportation AB, a large manufacturer of trains and Mälardalen University both located in Västerås, Sweden. As shown in Figure 2.2 this cooperation is driven by a methodology encapsulating our common research opportunities. The vision of this methodology is to improve the state of the practice in automated test generation and

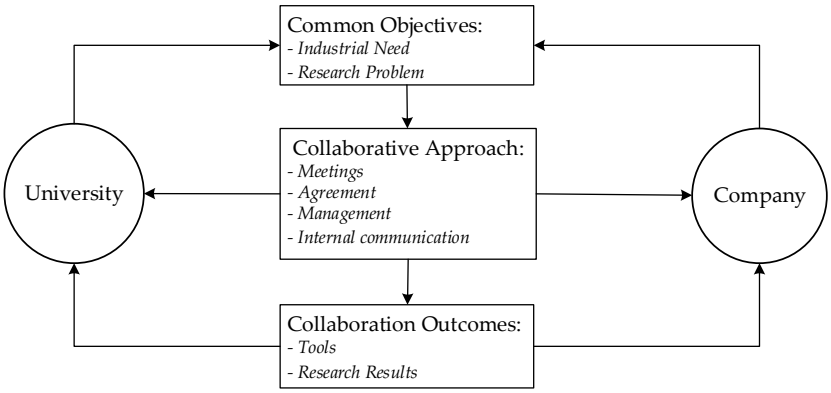

Figure 2.2: Model of Collaborative Research Methodology

evaluation through design, implementation and conduct of relevant research that could be translated into software testing policy and practice. A major emphasis was made on using available research in the area of *automated test generation*.

As shown in Figure 2.2 the research was build upon *common objectives*. Both partners were keen to demonstrate the industrial efficacy of the new and uncertain automated test generation technology. The collaborative approach demonstrates that the university and the company can together obtain tools and applied research results which they could not achieve independently.

Our research starts with finding a problem or opportunity, and ends with proposing a solution for that problem while building knowledge in the area of software testing. We identify a general research problem from software testing and provide a solution to it by refining and narrowing down the general problem. First the overall goal is decomposed into clearer research goals. The research is performed by giving clear descriptions, using prototype implementations, and evaluating the framework on industrial examples.

2.3 Contributions

In this section, we map the contributions of the thesis to the goals formulated earlier. The relation between each contribution and the research questions is presented in Table 2.1.

	RG 1	RG 2	RG 3
Paper A	✓	✓	
Paper B			✓
Paper C		✓	
Paper D	✓	✓	✓

Table 2.1: Contribution of the individual papers to the research goals

2.3.1 Paper A

Model-based Test Suite Generation for Function Block Diagrams using the UPPAAL Model Checker.
Eduard Paul Enoiu, Daniel Sundmark, and Paul Pettersson. In the Sixth International Conference on Software Testing, Verification and Validation Workshops (ICSTW), pages 158 - 167, ISBN: 978-1-4799-1324-4, 2013, IEEE.
Summary. In the first paper, we propose a framework for test generation using a model checker and by that we address RG 1 and RG 2. We propose a translation of FBD programs into timed automata models. We present in detail this approach using the UPPAAL model-checker in the context of a model-based approach towards unit testing. For the translation of a program into timed automata, a set of rules are presented. On the basis of this model, a model checker has been used for generating test suites.
My contribution. The development of the concept was done by the first author. I implemented the models, prototype tools, and performed the experiments.

2.3.2 Paper B

MOS: An Integrated Model-based and Search-based Testing Tool for Function Block Diagrams.
Eduard Paul Enoiu, Kivanc Doganay, Markus Bohlin, Daniel Sundmark, Paul Pettersson. Published in the 1st International Workshop on Combining Modelling and Search-Based Software Engineering (CMSBSE), pages 55 - 60, ISBN: 978-1-4673-6284-9, 2013, IEEE.
Summary. Based on Paper A and aimed at increasing confidence on the results for RG 3, this paper presents a combined model and search-based approach to testing Function Block Diagrams in practice, as well as several specific implications. The approach is aimed at safety critical applications described in Function Block Diagram language, and supports both a model-based and a search-based approach. In Paper B, and to achieve RG 3, we describe the ar-

chitecture of the tool, its workflow process, and a small descriptive case study in which the tool has been applied in a real industrial setting to test a train control management system.
My contribution. The first two authors are the main contributors of the paper focusing on model-based and search-based approach respectively, with the other co-authors having academic advisory role.

2.3.3 Paper C

Using Logic Coverage to Improve Testing Function Block Diagrams.
Eduard Paul Enoiu, Daniel Sundmark, Paul Pettersson. Published in Testing Software and Systems, Proceedings of the 25th IFIP WG 6.1 International Conference ICTSS 2013, volume 8254, pages 1 - 16, Lecture Notes in Computer Science, 2013, Springer.
Summary. As a direct result of Paper A, we address RG 2 in order to improve testing of Function Block Diagrams. We generate tests that cover the structure of Function Block Diagrams by using logic coverage criteria. One way of dealing with structural testing is to approach it as a model checking problem, such that model checking tools automatically create tests. We start from the framework introduced in Paper A and we show how logic coverage criteria can be formalised and used by a model checker to provide tests.

Not suprisingly, we observe that for more complicated logic coverage criteria, test cases result in longer tests than for simpler logic coverage criteria. Further, we note that the use of timer elements in the language is influencing the test generation efficiency in terms of generation time and used memory.
My contribution. I am the main author of the paper, with my co-authors having academic and industrial advisory role. I implemented the models, the concept, and performed the experiments.

2.3.4 Paper D

Automated Test Generation using Model-Checking: An Industrial Evaluation
Eduard Paul Enoiu, Adnan Čaušević, Elaine Weyuker, Tom Ostrand, Daniel Sundmark and Paul Pettersson. Accepted for Publication in the International Journal on Software Tools for Technology Transfer, 2014, Springer.
Summary. We continue this collection of papers with a paper detailing the development of a tool used in practice for automatic test generation and a large

case study with more elaborate empirical evaluation of the use of model checking for testing. To address RG 3 we measure the efficiency of using logic coverage for Function Block Diagram programs. In Paper D, we further show how a tool for test case generation that aims to satisfy logic coverage on Function Block Diagrams can be eficiently implemented using a model checker. To further address RG 1 and RG 2 we describe improvements to the technique proposed in Paper B and present a toolbox in which logic coverage criteria can be formalized and used by a model-checker to generate test cases. We carried out an extensive empirical study of the method by applying the toolbox to 157 real-world industrial programs developed at Bombardier Transportation AB. The results indicate that model checking is suitable for handling logic coverage for real-world Function Block Diagram programs, but also revealed some potential limitations of the toolbox when used for test generation such as the usage of manual expected outputs. The evaluation showed that the toolbox is efficient in terms of time required to generate tests that satisfy logic coverage and that it scales well for most of the programs.

My contribution. The first author is the main contributor of the paper focusing on both theoretical and experimental results, with the other co-authors having academic and industrial advisory role.

Chapter 3

Related Work

3.1 Function Block Diagrams and IEC 61131-3

PLCs are widely used in different control systems from nuclear power plants to traffic control systems. A PLC is an industrial real-time computer, integrated with a processor, a main memory, linked together by a common bus. Programs running on a PLC execute in a loop, in which the iteration follows the *"read-execute-write"* semantics. This ensures that the PLC reads all inputs, executes the computation, and then writes to its output, all without interruption. Function Block Diagram, a PLC programming language standardized by IEC 61131-3, is popular because of its graphical notations and its usefulness in applications with a high degree of data flow between control components.

The IEC 61131-3 standard proposes a hierarchical software architecture for structuring and running any Function Block Diagram program. This architecture specifies the syntax and semantics of a unified control software based on a PLC configuration, resource allocation, task control, program definition, function and function block repository, and program code [11, 17, 25].

3.2 Model Checking-Based Test Generation

A model checker has been used to find test cases to various criteria and from programs in a variety of formal languages [5, 12]. In addition, Black et al. [2] discuss the problems encountered in using a model-checker for test case generation for full-predicate coverage and explain why logic coverage criteria

is not directly applicable for model-checking. Rayadurgam et al. [20] present an alternative method that modifies instead the system model and are obtaining MC/DC adequate test cases using a model-checking approach. Similarly to our work, the system model is annotated and the properties to be checked are expressible as a single test sequence. However, this technique is not coping with the timing behavior of a Function Block Diagram program as we do and only MC/DC criteria is investigated. We provide an approach to generate test cases for different logic criteria that are directly applicable to Function Block Diagram programs.

Similar to this work, Rayadurgam and Heimdahl [21] have defined a complete formal framework that can be used for coverage based test-case generation using a model checker. For a detailed overview of testing with model checkers we refer the reader to Fraser et al. [10].

The idea of using model-checkers for verifying and testing Function Block Diagram programs is not new [24, 8]. These two approaches use the UPPAAL model checker and UPPAAL TRON for verification of Function Block Diagram programs, however they translate their model for functional verification. Soliman et al. [24] provide an automatic transformation to timed automata and their verification methodology is used to check the model against safety requirements. In contrast to the online model-based testing approach used in [8] we generate test suites for offline execution.

3.3 Testing Function Block Diagram Software

Previous contributions in testing of Function Block Diagram programs range from a simulation-based approach [22] to verification of the actual Function Block Diagram program code [4, 13]. The technique in [4] is based on Petri Nets models. In comparison to our work, they are not coping with the internal structure of the PLC logical and timing aspects. It is our opinion that testing Function Block Diagram programs can be complemented by using a model-checker as presented in this thesis.

Similar to our work there have been some attempts to focus on Function Block Diagram testing [14, 13]. These works are focusing on structural testing techniques and are proposing a solution based on the logical aspects of the Function Block Diagram. The criteria used in this thesis is tailored for the usage of a model checker and is complementing previous work in this area.

Related to this work but outside the PLC testing community, the most notable efforts have been focusing on test coverage for data flow languages. For

example, for the Lustre language there are contributions [15] describing an activation condition concept that can be used when data flows from an input edge to an output edge. While this approach studied the effect of structural coverage criteria on the overall program, we study the ability to generate test cases and its effect on the test artifacts, i.e., predicates and clauses, tailored for Function Block Diagram programs.

Chapter 4

Conclusions and Future Work

To our knowledge, not much theoretical work and experimental data is available regarding testing for Function Block Diagrams. In our work we have defined a model-based test generation method tailored for Function Block Diagram programs and demonstrated how to use a tool for model checking the implementation in order to ensure compliance to quality requirements including unit testing. As a consequence of these results we have developed our own framework to support both a model and search-based testing approach which can include specific coverage measurements. One way of dealing with test case generation for ensuring program coverage is to approach it as a model-checking problem, such that model-checking tools automatically create test cases. We showed how logic coverage criteria can be formalized and used by a model-checker to provide test cases for ensuring this coverage on safety-critical software described in Function Block Diagram language.

The testing framework presented in this thesis is based on both our previous work and the related work in this field. It is an attempt to automatically compute tests using a model checker for Function Block Diagrams. We provide evidence for the usage of logic coverage as an improvement to testing of Function Block Diagrams.

There are still issues to resolve before the technique can be applied to more complex programs and in production for different companies, but we are already working on ways to overcome them. In particular, we need to understand how its usage in practice can vary depending on technological and human fac-

tors. In addition we are currently investigating this approach on a larger case study. In addition, we want to extend the evaluation to measure both efficiency and effectiveness of our approach.

Bibliography

[1] R. Alur and D. Dill. Automata for Modeling Real-time Systems. *Automata, languages and programming*, pages 322–335, 1990.

[2] P. Ammann, P. E. Black, and W. Ding. Model Checkers in Software Testing. In *NIST-IR 6777, National Institute of Standards and Technology Report*, 2002.

[3] P. Ammann and J. Offutt. *Introduction to software testing*. Cambridge University Press, 2008.

[4] L. Baresi, M. Mauri, A. Monti, and M. Pezze. PLCTools: Design, Formal Validation, and Code Generation for Programmable Controllers. *IEEE International Conference on Systems, Man, and Cybernetics*, 4:2437–2442, 2000.

[5] P. Black. Modeling and Marshaling: Making Tests from Model Checker Counter-examples. In *Proceedings of the 19th Digital Avionics Systems Conference*, volume 1, pages 1B3–1. IEEE, 2000.

[6] E. CENELEC. 50128: Railway applications-communication, signalling and processing systems-software for railway control and protection systems. *CENELEC European Committee for Electrotechnical Standardization, Central Secretariat: rue de Stassart*, 35, 2014.

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

[8] L. da Silva, L. de Assis Barbosa, K. Gorgônio, A. Perkusich, and A. Lima. On the Automatic Generation of Timed Automata Models from Function Block Diagrams for Safety Instrumented Systems. *34th Annual Conference of IEEE Industrial Electronics, 2008. IECON 2008.*, pages 291–296, 2008.

[9] K. Doganay, M. Bohlin, and O. Sellin. Search based testing of embedded systems implemented in iec 61131-3: An industrial case study. *International Conference on Software Testing, Verification and Validation Workshops*, 2013.

[10] G. Fraser, F. Wotawa, and P. E. Ammann. Testing with Model Checkers: a Survey. In *Journal on Software Testing, Verification and Reliability*, volume 19, pages 215–261. Wiley Online Library, 2009.

[11] W. A. Halang. Languages and Tools for the Graphical and Textual System Independent Programming of Programmable Logic Controllers. *Microprocessing and Microprogramming Journal*, 27(1):583–590, 1989.

[12] H. S. Hong, I. Lee, O. Sokolsky, and H. Ural. A Temporal Logic-Based Theory of Test Coverage and Generation. In *Tools and Algorithms for the Construction and Analysis of Systems*, pages 327–341. Springer, 2002.

[13] E. Jee, S. Kim, S. Cha, and I. Lee. Automated Test Coverage Measurement for Reactor Protection System Software implemented in Function Block Diagram. *Computer Safety, Reliability, and Security*, pages 223–236, 2010.

[14] E. Jee, J. Yoo, S. Cha, and D. Bae. A Data Flow-based Structural Testing Technique for FBD Programs. *Information and Software Technology*, 51(7):1131–1139, 2009.

[15] A. Lakehal and I. Parissis. Lustructu: A Tool for the Automatic Coverage Assessment of Lustre Programs. *16th IEEE International Symposium on Software Reliability Engineering, 2005.*, pages 10–pp, 2005.

[16] G. J. Myers, C. Sandler, and T. Badgett. *The art of software testing*. John Wiley & Sons, 2011.

[17] M. Öhman, S. Johansson, and K. Årzén. Implementation Aspects of the PLC standard IEC 1131-3. *Control Engineering Practice*, 6(4):547–555, 1998.

[18] A. Orso and G. Rothermel. Software testing: a research travelogue (2000–2014). *Proceedings of the IEEE International conference on Software Engineering (ICSE), Future of Software Engineering*, 2014.

[19] M. Perkmann and K. Walsh. Engaging the scholar: Three types of academic consulting and their impact on universities and industry. *Research Policy*, 37(10):1884–1891, 2008.

[20] S. Rayadurgam and M. Heimdahl. Generating MC/DC Adequate Test Sequences Through Model Checking. In *NASA Goddard Software Engineering Workshop Proceedings*, pages 91–96. IEEE, 2003.

[21] S. Rayadurgam and M. P. Heimdahl. Coverage Based Test-Case Generation using Model Checkers. In *International Conference and Workshop on the Engineering of Computer Based Systems*, pages 83–91. IEEE, 2001.

[22] S. Richter and J. Wittig. Verification and Validation Process for Safety IC Systems. *Nuclear Plant Journal*, 21(3):36–36, 2003.

[23] M. D. Schwartz, J. Mulder, J. Trent, and W. D. Atkins. Control System Devices: Architectures and Supply Channels Overview. *Sandia National Laboratories Sandia Report SAND2010-5183*, 2010.

[24] D. Soliman, K. Thramboulidis, and G. Frey. Function Block Diagram to UPPAAL Timed Automata Transformation Based on Formal Models. *Information Control Problems in Manufacturing*, 14(1):1653–1659, 2012.

[25] J. Thieme and H. Hanisch. Model-based Generation of Modular PLC Code using IEC61131 Function Blocks. In *Proceedings of the International Symposium on Industrial Electronics*, volume 1, pages 199–204. IEEE, 2002.

[26] W. Visser, K. Havelund, G. Brat, S. Park, and F. Lerda. Model checking programs. *Automated Software Engineering*, 10(2):203–232, 2003.

[27] W. Visser, C. S. Psreanu, and S. Khurshid. Test input generation with java pathfinder. *ACM SIGSOFT Software Engineering Notes*, 29(4):97–107, 2004.

[28] W. E. Wong, T. Tse, R. L. Glass, V. R. Basili, and T. Y. Chen. An assessment of systems and software engineering scholars and institutions (2003–2007 and 2004–2008). *Journal of Systems and Software*, 84:162–168, 2011.

[29] H. Zhu, P. Hall, and J. May. Software unit test coverage and adequacy. *ACM Computing Surveys (CSUR)*, 29(4):366–427, 1997.