

Verification of Programmable Logic Controller Code using Model Checking and Static Analysis

Von der Fakultät für Mathematik, Informatik und Naturwissenschaften
der RWTH Aachen University zur Erlangung des akademischen Grades
eines Doktors der Naturwissenschaften genehmigte Dissertation

vorgelegt von

Diplom-Informatiker
Sebastian Biallas
aus Düsseldorf

Berichter: Universitätsprofessor Dr.-Ing. Stefan Kowalewski
Universitätsprofessor Dr.-Ing. Alexander Fay

Tag der mündlichen Prüfung: 14.7.2016

Diese Dissertation ist auf den Internetseiten der Hochschulbibliothek online verfügbar.

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über <http://dnb.d-nb.de> abrufbar.

Zugl.: D 82 (Diss. RWTH Aachen University, 2016)

Sebastian Biallas
Informatik 11 — Embedded Software
biallas@embedded.rwth-aachen.de

Aachener Informatik Bericht AIB-2016-07

Herausgeber: Fachgruppe Informatik
RWTH Aachen University
Ahornstr. 55
52074 Aachen
GERMANY

ISSN 0935-3232

Copyright Shaker Verlag 2016

Alle Rechte, auch das des auszugsweisen Nachdruckes, der auszugsweisen oder vollständigen Wiedergabe, der Speicherung in Datenverarbeitungsanlagen und der Übersetzung, vorbehalten.

Printed in Germany.

ISBN 978-3-8440-4711-0

Shaker Verlag GmbH • Postfach 101818 • 52018 Aachen
Telefon: 02407 / 95 96 - 0 • Telefax: 02407 / 95 96 - 9
Internet: www.shaker.de • E-Mail: info@shaker.de

ABSTRACT

Programmable Logic Controllers (PLCs, ger. Speicherprogrammierbare Steuerungen) are control devices used in industry to control, operate and monitor plants, machines and robots. PLCs comprise input connectors, typically connected to sensors, output connectors, typically connected to actuators, and a program, which controls the behavior, computing new output values based on the input values and an internal state. Since PLCs operate in safety-critical environments, where a malfunction could seriously harm the environment, humans, or the plant, it is recommended to verify their programs using formal methods.

This dissertation studies the formal methods *model checking* and *static analysis* to prove the correctness of PLC programs. For this, we developed the tool ARCADE.PLC, which allows the automatic application of these methods to PLC programs written in different vendor-specific dialects. It extracts a model from the program by abstract simulation, which over-approximates the possible program behavior. The user is then able to verify whether the model obeys a specification, which can be written in the logic CTL or using automata.

For applying model checking, we demonstrate how the model can be extracted automatically, such that the approach scales to industrial size programs. For this, we introduce two different abstraction techniques: First, we develop an abstraction refinement guided by the model checker that automatically creates an abstracted model by iteratively analyzing counterexamples. Second, we implemented a predicate abstraction that evaluates a formalized program semantics using an SMT solver. Both techniques are evaluated using different programs from industry and academia. Further, we introduce a simplified formalism to write specifications, which is influenced by an automata-based language established in industry. We implement an algorithm to check programs using this formalism and show that this technique is even able to detect errors in the specification. Finally, we detail how counterexamples generated by the model checker can be analyzed automatically to locate the actual erroneous line in the program.

The static analysis we developed is able to detect program errors in a fully automatic way. We detect typical errors such as division by zero and illegal array accesses, but also PLC specific errors, e.g., during a restart. The analysis is based on a value-set analysis, which determines the values of all program variables in each program location. These sets are then verified against the predefined checks or user-provided annotations. We show how to implement this analysis such that it scales to industrial size programs. The approach is evaluated on an industrial case study.

Speicherprogrammierbare Steuerungen (SPSen, engl. Programmable Logic Controller) sind Automatisierungsgeräte, welche zur Steuerung, Regelung und Überwachung von industriellen Anlagen und Maschinen eingesetzt werden. Sie besitzen dazu Eingänge, die mit Sensoren verbunden sind, Ausgänge, die mit Aktuatoren verbunden sind und ein Programm, welches die Ausgänge in Abhängigkeit der Eingänge und eines internen Speichers belegt. Da SPSen häufig in kritischen Bereichen eingesetzt werden, in denen eine Fehlfunktion Gefahren für Mensch, Umwelt oder die Anlage bergen kann, ist die Korrektheit des Programms zu prüfen.

Diese Dissertation untersucht die formalen Methoden *Model-Checking* und *Statische Analyse*, um die Korrektheit von PLC-Programmen zu beweisen. Wir haben dazu das Tool ARCADE.PLC geschrieben, welches es ermöglicht, diese Techniken vollautomatisch auf PLC-Programme verschiedener Hersteller anzuwenden. Es extrahiert durch abstrakte Simulation ein Modell, welches sämtliches Programmverhalten widerspiegelt. Der Benutzer kann dann überprüfen, ob das Modell einer Spezifikation entspricht, welche er in der Logik CTL formulieren muss oder als Automaten eingeben kann.

Zum Bereich Model-Checking zeigen wir in dieser Dissertation, wie das Modell automatisch abstrahiert werden kann, so dass der Ansatz auch für industrielle Programme skaliert. Es werden dazu zwei verschiedene Abstraktionstechniken eingeführt: Eine durch den Model-Checker gesteuerte Abstraktionsverfeinerung erstellt ein abstrahiertes Modell iterativ durch Analyse von Gegenbeispielen. Außerdem haben wir eine automatische Prädikat-Abstraktion implementiert, welche mithilfe einer SMT-Solvers die formalisierte Programmsemantik auf Prädikaten auswertet. Beide Techniken werden anhand verschiedener Programme evaluiert. Weiterhin führen wir einen vereinfachten Spezifikationsformalismus ein, welcher sich an einer in der Industrie etablierten Automaten-sprache orientiert. Wir implementieren einen Algorithmus, um Programme mit diesem Formalismus zu überprüfen und zeigen, dass durch diese Technik auch Spezifikationsfehler entdeckt werden können. Schließlich zeigen wir noch, wie vom Model-Checker gefundene Gegenbeispiele analysiert werden können, um die eigentlich fehlerhafte Programmzeile automatisch zu lokalisieren.

Die von uns implementierte Statische Analyse kann vollautomatisch Programmfehler entdecken. Dazu gehören beispielsweise eine Division durch Null, unerlaubte Array-Zugriffe oder PLC-spezifische Fehler z.B. beim Neustart. Die Analyse basiert auf einer Wertebereichsanalyse, welche eine Übermenge der Werte aller Variablen in allen Programmstellen berechnet. Wir zeigen, wie diese Analyse skalierbar implementiert werden kann. Der Ansatz wird an einer großen industriellen Fallstudie ausgewertet.

ACKNOWLEDGEMENTS

I worked on this dissertation while I was employed as a research assistant at the chair *Informatik 11 — Embedded Software* at the RWTH Aachen University. This work would not have been possible without the support of many others. First of all, I have to thank Prof. Dr.-Ing. Stefan Kowalewski for giving me the opportunity to join his group, for supporting my thesis, and for the excellent collaboration during this time. I would also like to thank Prof. Dr.-Ing. Alexander Fay for serving as a second supervisor and for helpful remarks. Furthermore, I thank Prof. Dr. Bernhard Rumpe and apl. Prof. Dr. Thomas Noll for participating in my examination committee.

I have to thank Dr. Bastian Schlich, whom I first met when I was a student while he was a postdoctoral researcher at the chair. He introduced me to the topic of formal verification of embedded software and supervised by Diploma thesis. Later, he went to ABB and was able to establish an industrial research collaboration. I would also like to thank his colleagues, especially Dr. Stefan Hauck-Stattemann, for the great collaboration during this time.

Furthermore, I would like to thank all my former colleagues and friends for the great time I had at the chair. In particular, I enjoyed the numerous interesting discussions I had with Dr. Jörg Brauer even outside the work environment.

I also have to thank Dr. Ralf Huuck for giving me the possibility to join his group at NICTA during a research visit. I learned a lot about the static analysis of C and C++ programs and how to scale such an analysis to industrial size programs.

I especially have to thank my students. They wrote excellent bachelor and master theses, implemented algorithms and user interfaces, performed case studies, and contributed as co-authors of publications. This work would not have been possible without them.

Financially, my work was supported by the *Deutsche Forschungsgemeinschaft*. Further, I was supported by the DFG research training group 1298 *Algorithmic Synthesis of Reactive and Discrete-Continuous Systems* and the DFG Cluster of Excellence on *Ultra High Speed Mobile Information and Communication*. I am very grateful for this support and I also have to thank these groups for many interesting and stimulating discussions.

Finally, I have to thank my parents and my sister for support and proof-reading.

Sebastian Biallas
July 2016, Berlin

CONTENTS

1	INTRODUCTION	1
1.1	Formal Verification of PLC Code	2
1.2	Contribution & Outline	3
1.2.1	Model Checking	3
1.2.2	Static Analysis	5
1.2.3	Combining Model Checking and Static Analysis	6
1.3	Related Work	6
1.4	Bibliographic Notes & Contributions by the Author	7
2	FORMAL VERIFICATION OF PLC CODE	9
2.1	A Brief History of Programmable Logic Controllers	9
2.2	Status Quo	9
2.2.1	Program Organization Units	10
2.2.2	Modes of Operation	10
2.2.3	Programming Languages	11
2.2.4	Variables, Data Types, Lifetime and Scope	12
2.2.5	General Organization	14
2.2.6	Timers	15
2.2.7	Function Block Calls	15
2.2.8	Standard & Vendor-Specific Extensions	16
2.3	PLCOPEN	17
2.4	Formal Verification using Model Checking	18
2.4.1	Kripke Structures	18
2.4.2	CTL Formulae	18
2.4.3	Counterexamples and Witnesses	19
2.5	Model Checking PLC Programs	20
2.5.1	Concrete Model	20
2.5.2	Abstract Model for PLC Programs	21
3	IMPLEMENTATION	25
3.1	ARCADE.PLC	25
3.2	Organization	26
3.3	Generic Simulator and Abstract Domains	27
3.3.1	Lattices	27
3.3.2	Intervals	28
3.3.3	Bitsets	28
3.3.4	Extensions	28
3.3.5	Reduced Product	29
3.4	Translation to the Intermediate Representation	30
3.4.1	Parsers	30

3.4.2	Annotations using Pragmas	30
3.4.3	Pragmatic & Practical Considerations	31
3.4.4	Instructions	32
4	COUNTEREXAMPLE-GUIDED ABSTRACTION REFINEMENT	37
4.1	Approach	37
4.1.1	Related Work	38
4.1.2	Contributions & Outline	39
4.2	Worked Example	39
4.3	Constraint Solver	41
4.3.1	Constraints on Abstract Values	42
4.3.2	Constraints on Expressions	43
4.3.3	Transforming Constraints	44
4.4	Refinements	45
4.4.1	Refinement of Input Variables	46
4.4.2	Refinement of Local Variables	47
4.5	State Space Organization	50
4.5.1	Counterexample Analysis	51
4.5.2	Worked Example	51
4.6	Case Studies	54
4.7	Conclusion	56
5	PREDICATE ABSTRACTION	59
5.1	Overview & Outline	59
5.2	Related Work	59
5.3	Worked Example	60
5.4	Encoding of PLC semantics in FOL	61
5.4.1	Encoding of Variables and the Program	61
5.4.2	Translating PLC Programs as FOL Formulae	63
5.4.3	Encoding of Timers	65
5.4.4	Succinct Representation of Control-Flow Automata	67
5.5	Predicate Abstraction	67
5.5.1	Implementation of the Predicate Abstraction	68
5.5.2	Scoping of Predicates	70
5.6	Case Study	71
5.7	Conclusion	72
6	MODEL CHECKING USING SAFETY AUTOMATA SPECIFICATIONS	73
6.1	Motivation & Overview	73
6.1.1	Bibliographic Notes & Related Work	74
6.1.2	Contribution & Outline	75
6.2	Safety Automata	75
6.2.1	Formalization	75
6.2.2	Simplifications & Conventions	76
6.2.3	Relation to CTL	77
6.3	A Model Checking Algorithm for Safety Automata	77

6.3.1	On-the-fly Checking	77
6.3.2	Counterexamples	78
6.3.3	Extensions	78
6.4	Checking PLCOPEN Safety Function Blocks	80
6.5	Detecting Over-Specifications	82
6.5.1	Detecting Over-Specifications in Safety Automata	82
6.5.2	Detection of a Faulty Specification	83
6.6	Concluding Discussion & Future Work	84
6.6.1	Automata Compared to CTL	84
6.6.2	Future Work	84
7	FAULT LOCALIZATION IN COUNTEREXAMPLES	87
7.1	Approach	87
7.2	Motivating Example	88
7.3	Trace Comparison	90
7.3.1	Preliminaries	90
7.3.2	Analysis of the Last Cycle	91
7.3.3	Analysis of a Trace	93
7.3.4	Correction Candidates	93
7.3.5	Case Study	95
7.3.6	Discussion	96
7.4	Candidate Exclusion	97
7.4.1	Testing Multiple Lines at Once	98
7.4.2	Testing Multiple Cycles	98
7.4.3	Coincidental Correctness & Preconditions	99
7.4.4	Multiple Necessary Error Candidates	101
7.4.5	Case Study	101
7.5	Discussion & Comparison	101
7.6	Related Work	102
7.7	Conclusion & Future Work	103
8	STATIC ANALYSIS OF PLC PROGRAMS	105
8.1	Approach	105
8.1.1	Contribution & Outline	106
8.1.2	Related Work	106
8.2	Static Analysis Process	107
8.2.1	Pointer Analysis	108
8.2.2	Control-Flow-Graph Builder	109
8.2.3	Static Analyses Dataflow Framework	111
8.2.4	Live Variable & Reaching Definition Analysis	112
8.2.5	Value-Set Analysis	113
8.2.6	Value-Set Analysis with Sparse Memory States	114
8.2.7	Widening	115
8.2.8	Post-Analysis	115
8.3	Localization of Function Block Variables	116

8.4	Initializations & Partial Unrolling	118
8.4.1	Retain Variables	119
8.5	Implementation of Checks	119
8.6	Case Studies	122
8.6.1	Industrial Programs	122
8.6.2	Specific Warning: Illegal GetStructComponent / PutStruct- Component	123
8.6.3	PLCOPEN Safety Function Blocks	124
8.7	Calculation of Summaries	124
8.8	Conclusion & Future Work	125
9	STATIC ANALYSIS & MODEL CHECKING INTERPLAY	127
9.1	Verification of a Safety Application	127
9.1.1	Modular Abstractions	128
9.1.2	Selecting Modular Refinements using Forward Slicing	129
9.1.3	State Space Reduction using Liveness Analysis	130
9.1.4	Final Analysis	131
9.2	Using the Model Checker to Augment Static Analysis Results	132
9.3	Conclusion	132
10	CONCLUSION	135
10.1	Formal Methods in Practice	135
10.2	Future Work	136
	Bibliography	139