

Scalable Techniques for Formal Verification

Sandip Ray

Scalable Techniques for Formal Verification

 Springer

Dr. Sandip Ray
Department of Computer Sciences
University of Texas, Austin
University Station 1
78712-0233 Austin Texas
MS C0500
USA
sandip@cs.utexas.edu

ISBN 978-1-4419-5997-3 e-ISBN 978-1-4419-5998-0
DOI 10.1007/978-1-4419-5998-0
Springer New York Dordrecht Heidelberg London

Library of Congress Control Number: 2010927798

© Springer Science+Business Media, LLC 2010

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer Science+Business Media, LLC, 233 Spring Street, New York, NY 10013, USA), except for brief excerpts in connection with reviews or scholarly analysis. Use in connection with any form of information storage and retrieval, electronic adaptation, computer software, or by similar or dissimilar methodology now known or hereafter developed is forbidden.

The use in this publication of trade names, trademarks, service marks, and similar terms, even if they are not identified as such, is not to be taken as an expression of opinion as to whether or not they are subject to proprietary rights.

Printed on acid-free paper

Springer is part of Springer Science+Business Media (www.springer.com)

To Anindita

*Who showed me that sometimes dreams
can come true*

Preface

This book is about *formal verification*, that is, the use of mathematical reasoning to ensure correct execution of computing systems. With the increasing use of computing systems in safety-critical and security-critical applications, it is becoming increasingly important for our well-being to ensure that those systems execute correctly. Over the last decade, formal verification has made significant headway in the analysis of industrial systems, particularly in the realm of verification of hardware. A key advantage of formal verification is that it provides a mathematical guarantee of their correctness (up to the accuracy of formal models and correctness of reasoning tools). In the process, the analysis can expose subtle design errors. Formal verification is particularly effective in finding corner-case bugs that are difficult to detect through traditional simulation and testing. Nevertheless, and in spite of its promise, the application of formal verification has so far been limited in an industrial design validation tool flow. The difficulties in its large-scale adoption include the following (1) deductive verification using theorem provers often involves excessive and prohibitive manual effort and (2) automated decision procedures (*e.g.*, model checking) can quickly hit the bounds of available time and memory.

This book presents recent advances in formal verification techniques and discusses the applicability of the techniques in ensuring the reliability of large-scale systems. We deal with the verification of a range of computing systems, from sequential programs to concurrent protocols and pipelined machines. Each problem is different, bringing in unique challenges for verification. We show how to ameliorate verification complexity by applying formal analysis judiciously within a hierarchical, disciplined reasoning framework.

The research documented here originally appeared as the author's Ph.D. dissertation. The research owes its existence more to skepticism about the possibility of mathematical analysis at this scale than to anything else. Coming from a background in theory of computation and armed with traditional results in computability and complexity theory, I approached the area with extreme suspicion: how can techniques that are so severely handicapped with undecidability, incompleteness, and (in the better scenarios) lower bounds of computational complexity be successful in practice? The answer to the paradox that I found in my own research is that formal reasoning should not be viewed as a stand-alone tool but rather an integral part of the system design process. In particular, formal reasoning thrives when

it serves to make explicit the intuitions already available to the system designers, helping them effectively articulate the informal justifications behind optimizations and design tweaks. A well-designed reasoning framework permits specification of a problem at a level that matches the intuition of the designer and this turns the analysis into a vehicle for making the intuitive justifications rigorous. Intuitive justification and mathematical rigor must go hand-in-hand, each clearing the way of the other.

Most treatises and research articles on formal verification do not focus on the role of analysis as part of the system design process. Indeed, verification research today is essentially single tracked: the mantra is to develop algorithms and heuristics for providing more automation in the analysis of larger and larger systems. Automation is certainly critical to large-scale system verification. However, with design complexity is increasing faster than automation heuristics can catch up, improvement in raw automation alone is insufficient: one must additionally have disciplined, hierarchical, and customized reasoning infrastructures for different domains. Such infrastructures must permit (1) decomposition of a complex problem into tractable pieces, (2) effective management and control of the overall verification and decomposition, and (3) the ability of different, customized, automated tools to bear upon each individual piece. Furthermore, the act of developing a reusable framework forces thinking about a verification problem at an appropriate level of abstraction. Note that an “abstraction” *does not* mean a “toy.” Rather, the abstraction must capture the essence of the class of systems being analyzed (often by generalization), while eliminating irrelevant complexities. Admittedly, developing a generic, compositional framework for a specific target domain is not easy. Thinking about a problem in a generic manner requires creativity, experience, and practice. However, the cost of doing so is ameliorated by the fact that the same framework can be used over and over on different problems. I have found the act of developing careful, generic frameworks for different problem domains to be both instructive and fruitful, as it exposes the underlying structure of the problems, helps spot connections between various subproblems, and thereby leads to the identification of the verification strategy most suited for the problem at hand. The research presented in this book shows numerous examples of generic verification strategies for different problem domains. The infrastructures described here are intended to serve as a guide to formal verification researchers and practitioners, who intend to pursue mathematical reasoning for large-scale systems. The book is also intended to show system designers how mathematical reasoning can serve as a helpful practical tool for ensuring reliable execution of the systems they design.

This research owes its existence to a number of people, including my teachers, collaborators, friends, and family. Perhaps the biggest debt goes to my Ph.D. supervisor J. Strother Moore for giving me the right balance of freedom, encouragement, and direction to guide the course of this research. Some other key players are (in alphabetical order) Jeff Golden, Warren A. Hunt, Jr., Matt Kaufmann, John Matthews, Erik Reeber, Fei Xie, and Thomas Wahl.

The research presented in this monograph has been supported over the years in part by funding from the National Science Foundation under Grants ISS-041741,

CCF-0916772, and CNS-0910913, by the Defense Advanced Research Projects Agency and National Science Foundation under Grant CNS-0429591, and by the Semiconductor Research Corporation under Grants 02-TJ-1032 and 08-TJ-1849. I am grateful to these agencies for the support. However, any opinions, findings, conclusions, or recommendations expressed herein are mine and do not necessarily reflect the views of any of these agencies.

Finally, I thank my wife Anindita for bearing with me through all the late hours that went behind the book. This book would not have seen the light of the day without her understanding and support.

Austin, TX
November 2009

Sandip Ray

Contents

1	Introduction	1
----------	---------------------	----------

Part I Preliminaries

2	Overview of Formal Verification	9
2.1	Theorem Proving	9
2.2	Temporal Logic and Model Checking	12
2.3	Program Logics, Axiomatic Semantics, and Verification Conditions	17
2.4	Bibliographic Notes	22
3	Introduction to ACL2	25
3.1	Basic Logic of ACL2	25
3.2	Ground Zero Theory	27
3.2.1	Terms, Formulas, Functions, and Predicates	30
3.2.2	Ordinals and Well-Founded Induction	32
3.3	Extension Principles	35
3.3.1	Definitional Principle	36
3.3.2	Encapsulation Principle	40
3.3.3	Defchoose Principle	42
3.4	The Theorem Prover	44
3.5	Structuring Mechanisms	45
3.6	Evaluators	46
3.7	The ACL2 Programming Environment	47
3.8	Bibliographic Notes	48

Part II Sequential Program Verification

4	Sequential Programs	53
4.1	Modeling Sequential Programs	53
4.2	Proof Styles	55
4.2.1	Stepwise Invariants	55
4.2.2	Clock Functions	56
4.3	Comparison of Proof Styles	57

4.4	Verifying Program Components and Generalized Proof Obligations	59
4.5	Discussion	62
4.5.1	Overspecification	62
4.5.2	Forced Homogeneity	63
4.6	Summary	64
4.7	Bibliographic Notes	64
5	Operational Semantics and Assertional Reasoning	65
5.1	Cutpoints, Assertions, and VCG Guarantees	65
5.2	VCG Guarantees and Symbolic Simulation	68
5.3	Composing Correctness Statements	70
5.4	Applications	72
5.4.1	Fibonacci Implementation on TINY	73
5.4.2	Recursive Factorial Implementation on the JVM	75
5.4.3	CBC-Mode Encryption and Decryption	75
5.5	Comparison with Related Approaches	76
5.6	Summary	78
5.7	Bibliographic Notes	78
6	Connecting Different Proof Styles	81
6.1	Soundness of Proof Styles	82
6.2	Completeness	84
6.3	Remarks on Mechanization	88
6.4	Discussion	88
6.5	Summary and Conclusion	90
6.6	Bibliographic Notes	91
Part III Verification of Reactive Systems		
7	Reactive Systems	95
7.1	Modeling Reactive Systems	96
7.2	Stuttering Trace Containment	97
7.3	Fairness Constraints	99
7.4	Discussion	103
7.5	Summary	106
7.6	Bibliographic Notes	107
8	Verifying Concurrent Protocols Using Refinements	109
8.1	Reduction via Stepwise Refinement	110
8.2	Reduction to Single-Step Theorems	110
8.3	Equivalences and Auxiliary Variables	114
8.4	Examples	116
8.4.1	An ESI Cache Coherence Protocol	116
8.4.2	An Implementation of the Bakery Algorithm	119
8.4.3	A Concurrent Deque Implementation	124

8.5	Summary	129
8.6	Bibliographic Notes	129
9	Pipelined Machines	131
9.1	Simulation Correspondence, Pipelines, and Flushing Proofs	131
9.2	Reducing Flushing Proofs to Refinements	134
9.3	A New Proof Rule	136
9.4	Example	137
9.5	Advanced Features	141
9.5.1	Stalls	141
9.5.2	Interrupts	141
9.5.3	Out-of-Order Execution	142
9.5.4	Out-of-Order and Multiple Instruction Completion	142
9.6	Summary	143
9.7	Bibliographic Notes	144

Part IV Invariant Proving

10	Invariant Proving	149
10.1	Predicate Abstractions	151
10.2	Discussion	153
10.3	An Illustrative Example	154
10.4	Summary	156
10.5	Bibliographic Notes	157
11	Predicate Abstraction via Rewriting	159
11.1	Features and Optimizations	163
11.1.1	User-Guided Abstraction	164
11.1.2	Assume Guarantee Reasoning	164
11.2	Reachability Analysis	165
11.3	Examples	166
11.3.1	Proving the ESI	166
11.3.2	German Protocol	168
11.4	Summary and Comparisons	169
11.5	Bibliographic Notes	171

Part V Formal Integration of Decision Procedures

12	Integrating Deductive and Algorithmic Reasoning	175
13	A Compositional Model Checking Procedure	179
13.1	Formalizing a Compositional Model Checking Procedure	180
13.1.1	Finite State Systems	180
13.1.2	Temporal Logic formulas	181
13.1.3	Compositional Procedure	181

13.2	Modeling LTL Semantics	183
13.3	Verification	187
13.4	A Deficiency of the Integration and Logical Issues	190
13.5	A Possible Remedy: Integration with HOL4	192
13.6	Summary and Discussion	193
13.7	Bibliographic Notes	194
14	Connecting External Deduction Tools with ACL2	195
14.1	Verified External Tools	196
14.1.1	Applications of Verified External Tools	200
14.2	Basic Unverified External Tools	203
14.2.1	Applications of Unverified External Tools	204
14.3	Unverified External Tools for Implicit Theories	205
14.4	Remarks on Implementation	209
14.4.1	Basic Design Decisions	209
14.4.2	Miscellaneous Engineering Considerations	211
14.5	Summary	214
14.6	Bibliographic Notes	215
Part VI Conclusion		
15	Summary and Conclusion	219
References		223
Index		237