Undergraduate Topics in Computer Science

Undergraduate Topics in Computer Science (UTiCS) delivers high-quality instructional content for undergraduates studying in all areas of computing and information science. From core foundational and theoretical material to final-year topics and applications, UTiCS books take a fresh, concise, and modern approach and are ideal for self-study or for a one- or two-semester course. The texts are all authored by established experts in their fields, reviewed by an international advisory board, and contain numerous examples and problems. Many include fully worked solutions.

More information about this series at http://www.springer.com/series/7592

Peter Csaba Ölveczky

Designing Reliable Distributed Systems

A Formal Methods Approach Based on Executable Modeling in Maude

Springer

Peter Csaba Ölveczky
University of Oslo
Oslo
Norway

ISSN 1863-7310 ISSN 2197-1781 (electronic)
Undergraduate Topics in Computer Science
ISBN 978-1-4471-6686-3 ISBN 978-1-4471-6687-0 (eBook)
DOI 10.1007/978-1-4471-6687-0

Library of Congress Control Number: 2017947868

Printed on acid-free paper

This Springer imprint is published by Springer Nature
The registered company is Springer-Verlag London Ltd.
The registered company address is: The Campus, 4 Crinan Street, London, N1 9XW, United Kingdom

To Cecilia, Roland, and Robert

Foreword

De facto, both individually and socially, all of us rely more and more on software-mediated systems and devices. However, as software disasters and successful cyber-attacks keep piling up, the crucial importance of software quality and reliability, and the sobering realization of how vulnerable our systems are, loom larger and larger. In areas such as avionics, railway systems, microprocessor design, and security protocols, the obvious consequence, namely, the need for *mathematical* methods providing high assurance beyond the insufficient assurance made possible by testing alone is well understood, so that formal methods are applied in practice in such areas. But this is far from being the case in general. In particular, since most systems nowadays are *distributed* systems, which are very hard to test and can have very subtle bugs, the necessary but insufficient role of testing is painfully felt; but the obvious need for stronger verification methods beyond testing is still not fully understood or appreciated in practice.

An important question is why this highly problematic state of affairs remains largely unresolved. It is certainly true that, although big advances in both scalability and automation of formal methods have been made and very important successful formal verification efforts have been carried out, *scalability* is still an important challenge. However, in my view two closely related problems, quite orthogonal to scalability, present a serious obstacle, namely: (1) verifying *designs*, as opposed to verifying *code*, is hindered in practice by the lack of suitable mathematical models for system designs; and (2) there is considerable ignorance about the *mathematical modeling* nature of programming made possible by *declarative languages*. The importance of solving problem (1) is one of effectiveness: design errors can be orders of magnitude more expensive than coding errors and in fact account for most of the critical errors in system development. This does not mean that verifying code is unimportant; however, correct-by-construction code generation from verified designs is a promising alternative to standard code verification and can be a considerably more cost-effective way of achieving code correctness. Problem (2) is quite serious and is self-inflicted. In many prestigious universities worldwide most

undergraduates now only learn to program in imperative languages like C, C++, or Java, and often do not even *know* that it is possible for a program to also be a *mathematical model* of the problem it solves.

The point is that problems (1) and (2) are closely related. A declarative program, that is, a program written in a computational logic and specified as a *theory* in such a logic has two key advantages: (i) it defines a *mathematical model* of the system it executes, which means that the distinction between design and code either evaporates or becomes reduced to one of refining and optimizing a high-level declarative program into a more efficient, yet equivalent, program; and (ii) since a system design specified as a declarative program is already a mathematical object, verifying its properties is typically much easier than verifying them for a program written in an imperative language. This all means that understanding the crucial role of declarative programs as formal executable specifications can greatly help in solving problems (1) and (2) at the same time.

An important distinction to be made is that between what I call *system specification* and *property specification and verification*. A computational system can obviously be programmed. By programming it in a declarative language, we obtain a *mathematical model* of the *system* thus programmed. But only by having a mathematical model of a system is it meaningful at all to *verify* its mathematical *properties*. Such properties need not be expressed in the computational logic of the declarative language in which the system in question has been specified. Indeed, many properties, for example, temporal logic properties or inductive theorems, need not be *executable* at all. This means that system properties to be verified *about* a system design may be specified in various logics in which such properties have a natural and easy expression. This also means that formal verification can be seen as the task of proving that the model defined by a formal, executable specification $\mathscr{S}$—that is, by a declarative program $\mathscr{S}$—*satisfies* a set $\{\varphi_1, \ldots, \varphi_n\}$ of desired properties expressed as formulas $\varphi_1, \ldots, \varphi_n$ in a suitable property specification logic.

All this brings us to the present book, that addresses the above problems (1) and (2) in an excellent and eminently practical way. One of its key contributions to undergraduate CS education is how well it shows students that programming as mathematical modeling in a declarative language such as Maude is: (i) quite easy, (ii) fairly intuitive, and (iii) actually fun to do. Once this is done through many well-chosen examples and exercises, students come to realize, almost as an afterthought, that they have been doing *mathematical modeling* all along. This happens just as for the man who suddenly realized that he had been speaking in prose all his life. This "aha moment" opens the door for discussing issues of *formal correctness* and *formal specification and verification* of system *properties*, so that property logics and their associated verification methods can be naturally introduced and explained.

In the first part of this book, all this is done for *deterministic systems* specified in equational logic as functional programs in Maude. Since the mathematical model defined by an equational program is the *initial algebra* of such a program as an equational theory, students are then introduced to the specification and verification of *inductive properties* satisfied by such initial algebras, and are shown how Maude itself can be used as a simple inductive theorem prover to verify such properties.

Since equational logic is a sublogic of rewriting logic, which is a natural and simple logic in which to specify *distributed systems*, the book then moves in a natural and seamless way from its first part focused on deterministic systems into its second and main part, focused on the executable specification of distributed systems as rewrite theories in Maude. Properties of distributed systems and their specification and verification are then explained. The same gentle and gradual approach is followed in this second part. This is achieved so well and with such a wealth of examples, that the book can also be used as a first introduction to distributed systems, their modeling, and their verification at the undergraduate level. The same gradual method of approach is also followed for the specification and verification of properties. First, the simplest of such properties, namely, *invariants*, are introduced, and explicit-state *reachability analysis* supported by Maude's `search` command is used to automatically verify such invariants, or to do so up to a given depth bound if the system is infinite-state. After this, a gentle, yet quite thorough, introduction to linear-time temporal logic (LTL) and its semantics is given, and many examples are given showing how Maude's LTL model checker can be used to automatically verify LTL properties of a distributed system formally specified as a rewrite theory in Maude. Finally, broader perspectives are opened up by explaining how additional topics such as the specification and verification of real-time and of probabilistic systems can be treated by corresponding extensions of rewriting logic by means of real-time rewrite theories and probabilistic rewrite theories; and at the property level by suitable real-time and probabilistic extensions of temporal logic. Each notion is again illustrated by means of well-chosen examples and exercises.

In summary, this book addresses an important and serious need in undergraduate CS education and, at the same time, the broader need of training a next generation of computer scientists who are well acquainted with both distributed systems and with the mathematical modeling and verification of such systems. Given the present state of affairs, both in the vulnerability of our systems and the serious gaps in mathematical modeling abilities in undergraduate CS education, the appearance of this book could not be more timely. I have been using earlier drafts of this book in a program verification course at the University of Illinois at Urbana-Champaign and plan to recommend the present book to my students as reading material for such a course in the years to come. I am sure that it will be of great help to many other persons teaching programming languages, formal methods, and distributed systems at the undergraduate level and, above all, to the students themselves.

Cabo Palos José Meseguer
June 2017

Preface

The two main goals of this book are to:

1. provide an introduction to formal modeling and analysis of both data types and, in particular, distributed systems; and
2. provide an introduction to distributed computer systems and the challenges of designing and analyzing such systems.

The book is meant to be a first introduction to formal methods and therefore does not assume any previous knowledge about formal methods or distributed systems; it is based on a third-year course at the University of Oslo, but can equally well be taught at the second-year level. Some previous exposure to programming could be useful; likewise, experience with simple recursive functions is helpful but not necessary. There are no prerequisites on the mathematical side.

A distinguishing feature of this book is the significant use of the rewriting-logic-based Maude language and simulation and model checking tool for formally modeling both data types and distributed systems. Data types are specified using a functional programming style that students tend to like. Indeed, a valuable side effect of studying this book is training in writing recursive programs. For formally modeling distributed systems, Maude provides a simple yet intuitive and expressive modeling formalism that is particularly suitable for modeling distributed systems in an object-oriented way. Maude is by now a mature and well-established tool that is increasingly used around the world.

About the Content

As mentioned above, one main goal of this book is to gently introduce students to a wide range of concepts in formal methods, including:

- *verifying* properties about programs and (models of) systems; e.g., proving that a specification/program terminates for all possible inputs, and using equational logic to prove semantic properties;
- logics and inference systems; and
- automated model checking techniques to analyze properties for some—but not all—possible inputs/system configurations.

This book is divided into two parts. The first part deals with specifying the data types needed to model complex distributed systems. This part introduces classical algebraic specification and term rewriting theory, including reasoning about termination, confluence, and inductive equational properties.

The second part deals with formally modeling and analyzing distributed systems in rewriting logic using Maude. This part introduces rewriting logic and object-oriented modeling of distributed systems. It also introduces temporal logic to specify requirements that a system should satisfy. Such models are analyzed using Maude simulations, reachability analysis, and temporal logic model checking, thereby also giving the students a hands-on experience of the state-space explosion problem for distributed systems. As mentioned above, the second main goal of this book is to introduce the students to the problems of designing and analyzing distributed systems. Instead of giving theoretical explanations of these issues, the book tries to convey intuition about distributed systems and their design challenges through a range of examples/case studies in different domains, including: the dining philosophers problem, transport protocols like the alternating bit protocol and the sliding window protocol, classic distributed algorithms such as the distributed two-phase protocol for distributed database systems, distributed mutual exclusion and leader election algorithms, and the NSPK cryptographic protocol. Finally, the book briefly introduces two extensions of standard distributed systems: real-time systems and probabilistic systems.

The book is based on a course that has been given at the University of Oslo for more than 10 years, which implies that the book contains a wealth of exercises, both smaller ones and larger ones suitable for course projects, etc. Most of the executable code presented in this book, as well as other supplementary material, can be found at http://peterol.at.ifi.uio.no/BOOK.

I would like to thank José Meseguer, Dorel Lucanu, Narciso Martí-Oliet, and Ralf Sasse for many insightful and very helpful comments on earlier versions of this book, Indranil Gupta for discussions on distributed systems, Jon Grov for providing the figures used in this book, Si Liu for performing the statistical model checking experiments, Lars Kristiansen for discussions on logic, and Shiji Bijo, Antonio Gonzalez Burgueño, Benjamin Oliver, and Olaf Owe for pointing out mistakes in those earlier drafts. I also thank Hanne Riis Nielson and Ian Mackie for encouraging me to publish this book with Springer, and Simon Rees and Wayne Wheeler for their patience in waiting for it to be finished.

Oslo, Norway
June 2017

Peter Csaba Ölveczky

Contents