Texts in Computer Science

Editors
David Gries
Fred B. Schneider

For other titles published in this series, go to
www.springer.com/series/3191

V.S. Alagar • K. Periyasamy

Specification of Software Systems

2nd edition

Springer

Prof. V.S. Alagar
Dept. Computer Science and Software Eng.
Concordia University
St. Catherine Street West 1515
H3G 1M8 Montreal, Québec
Canada
alagar@cs.concordia.ca

Prof. K. Periyasamy
Computer Science Department
University of Wisconsin-La Crosse
State Street 1725
54601 La Crosse, WI
USA
kasi@cs.uwlax.edu

Series Editors
David Gries
Department of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853-7501, USA

Fred B. Schneider
Department of Computer Science
Upson Hall
Cornell University
Ithaca, NY 14853-7501, USA

ISSN 1868-0941
e-ISSN 1868-095X
ISBN 978-0-85729-276-6
e-ISBN 978-0-85729-277-3
DOI 10.1007/978-0-85729-277-3
Springer London Dordrecht Heidelberg New York

British Library Cataloguing in Publication Data
A catalogue record for this book is available from the British Library

Library of Congress Control Number: 2011922494

Cover design: deblik

Printed on acid-free paper

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

Preface to the Second Edition

This is the second edition of the textbook in which most of the concepts introduced in the first edition are extended and updated, and a significant amount of new material has been added. While preserving the original intent of focusing on **software specification**, this edition emphasizes the practice of formal methods for **specification and verification activities** for different types of software systems and at different stages of developing the software systems. This expanded view is reinforced both in the organization of the book and in the presentation of its contents. The primary driving force for writing the second edition came from Springer-Verlag, London, who expressed a great desire and strong interest in catering to the growing needs of students and researchers in the area of Formal Software Engineering.

Background and Motivation

Although during the initial stages of formal methods research there was only a marginal use of formal methods in industry, new languages, techniques and tools developed during early 1990s have spurred great interest in adapting formal methods in industries. In fact, the 1990s witnessed an explosion of new developments in formal methods research. NASA Langley Research Center was the first hub of formal methods research and practice. The researchers at Langley focused on large scale commercial projects that are suitable for injecting formal methods. They felt that the industries were reluctant to use formal methods because of inadequate tools, inadequate background, and lack of adequate examples. However, this situation started changing gradually during 1995–2004 when new directions of research and development of tools in the three areas *Software Specification Methods*, *Model Checking*, and *Theorem Proving* provided a great spur for formal development activity in industries. Most notably, software engineers at nuclear power stations, aerospace and transportation industries used formal methods to formally specify and verify the properties of *safety critical* parts in systems. In 1998, the fully automatic driverless subway was launched in Paris Metro and, in 2006, the fully automatic driverless shuttle servicing the various terminals at Roissy Airport, Paris was launched. With success stories such as these arose a desire in academia and industries to learn formal methods more systematically. In order to choose a method that is appropriate for a specific application that demands concepts such as *causality*, *concurrency*, and *conflict avoidance* a certain level of expertise in formal methods education is necessary. Getting to know that it is possible to mix different abstractions from different languages to model heterogeneous systems is an asset for an efficient development process. Nowadays formal methods often are bundled up with tools, many available as open source software, to support architectural principles of generality and orthogonality. In view of these spectrum of changes and success stories, software engineers now have several case studies to learn from and choose languages and methods with a rich repertoire of appropriate concepts for their intended applications. In keeping up with this trend this second edition is offered. In writing this second edition, the expectation is that formal methods will be well integrated into the teaching of software engineering programs. In this hope, topics related to the integration of formal methods in software development process are discussed quite early in the text and are followed by presentations of abstraction principles, formalism definitions, notations of formalism, and a wide variety of fairly detailed specification examples.

What is New in the Second Edition?

Old material has been updated to improve both content and presentation. Some chapters in the first edition of the text have undergone extensive revisions. In some cases, an old chapter has been split into two or more chapters and in each of them extensive new material have been added. New chapters that discuss Object-Z, B-Method, and Calculus of Communicating Systems have been added. The entire book has been structured into six parts. The distinguishing features of this restructured and expanded second edition are as follows.

Part I The first part of the book introduces specification fundamentals. The material is presented in four chapters. An elaborate introduction to the role of specification is followed by discussions on specification activities and specification qualities. The first part concludes with a discussion on abstraction principles, illustrated with a domain abstraction example.

Part II The second part introduces the basics of formalism, automata notations used in formal languages, study extensions to the basic automata notation, and concludes with a discussion on the classification of formal specification techniques. This material is presented in four chapters. The chapters that discuss automata and extended state machine notations are almost self-contained. A variety of examples that arise in software construction are taken up for formal modeling using different variations of state machines. Top-down and bottom-up constructions of formal models, their sequential and parallel compositions are discussed and illustrated with examples.

Part III The third part of the book is entirely devoted to logic. Propositional logic, predicate logic, and temporal logic are treated in three separate chapters. The presentation focuses on introducing the logics as formal languages, and hence introduces their syntax, semantics, and reasoning methods in succession. The expressive power of predicate logic is illustrated for representing knowledge, policies, as well as serving as axiomatic system for program verification. For the latter purpose, Hoare axioms are presented and illustrated by verifying simple sequential programs. Temporal logic chapter gives a detailed discussion of the syntax, and semantics of linear temporal logic. Many examples from reactive systems and concurrent systems are chosen to emphasize the expressivity of the logic languages in specifying such systems and their properties. A discussion of axiomatic proof method and model checking are included.

Part IV Most of the model-based specification languages are based on set theory and first-order predicate logic. Therefore, it is essential to have a strong background in set theory and relations. This part of the book includes one chapter on set theory and relations. Most parts of this chapter are retained from the previous edition of the book.

Part V Three specification methods are discussed to illustrate the property-oriented approach to specifications. Two of the chapters, *Algebraic Specifications* and *Larch*, are left unchanged. The chapter *Calculus of Communicating Systems* is new and it discusses Milner's algebraic approach to specifying communication and concurrency. Some examples discussed in Temporal Logic chapter are drawn in here to strike a comparison between the two approaches. An effort has been made to make the discussion in this chapter simple, rigorous, and self-contained.

Part VI This part is devoted to model-based specification techniques. Four such techniques are described in detail. These are VDM-SL, Z, Object-Z and the B-Method. Material on VDM-SL and Z are retained from the previous edition of the book, while the bibliographic references have been updated. Two new chapters have been introduced, one for Object-Z and another for the B-Method. The material for new chapters are presented in the same style as in the old chapters. Also, the two new chapters include extensive examples and case studies, and provide a detailed tutorial of the techniques introduced in those chapters.

How to Use the Book

In the second edition of the book, we have added considerable new material and we have also restructured the chapters into various parts. Consequently, those who have used the first edition may see a different layout of the book. The book includes several different specification techniques grouped into various categories. In addition, it also includes chapters with necessary mathematical background for these techniques. Because of the diverse nature of these techniques, the book can be used by different groups of people for different purposes. Below we suggest a few streams of course offerings to fit different curriculum needs.

1. Chapters in Part I are required for further reading of the book.
2. Based on Part I, Part II, Chaps. 9, 10 of Part III, and Part IV a one-semester undergraduate course within a software engineering program can be given. This course is intended to be an Introduction to Formal Software Engineering Methods. The course can be extended into another semester by covering the material from one of the four specification languages discussed in Chaps. 16 through 19, and choosing a project in which the students would write a complete specification and analyze the specification. The examples and case studies given in these chapters would help the students to achieve this goal.
3. Chapters in Part II, Part III, and Chap. 15 from Part V can be offered as a one-semester course for senior undergraduate students or first year graduate students in computer science and computer engineering programs. This course is intended to be an Introduction to Formal Methods.
4. Parts V and VI are devoted to various formal specification techniques. Each chapter in these parts gives a thorough tutorial of one specification technique. Together with the mathematical fundamentals described in Part IV, each chapter in Parts V and VI can be individually used to teach a one-semester course on a particular specification technique. The course will introduce the formal method in some depth, choosing appropriate tools suggested in the bibliographic notes of these chapters. It is suitable to teach this course at senior undergraduate level or at the graduate level provided that the students are exposed to some of the mathematical fundamentals described in Parts I through IV before taking this course. Alternately, a quick overview of the fundamentals can be covered in few weeks and the rest of the semester can be spent on the syntax and semantics of the chosen specification technique.
5. An advanced graduate-level course can be taught using any one of the techniques discussed in Parts V and VI with emphasis on developing complete specification for a fairly large problem. This would involve refinement, proof obligation, and implementation.
6. Another option would be to teach an advanced graduate-level course that requires the students to critically compare the techniques in each group and write a report. For example, one course could be taught on model-based specification techniques, all chapters in Part VI. Students in this course will get an in-depth understanding of the techniques and also would be able to choose the appropriate technique for a given problem.
7. Practitioners of formal methods, especially those who use formal methods for industrial applications, can use this book as a reference. In particular, the chapters in Parts V and VI have been written in such a way that a practitioner who is familiar with one technique can quickly jump start with another technique with little time. The examples and case studies in each chapter in these two parts provide sufficient information for a practitioner to start writing the specification for a new application without much preparation time.

Intended Audience

This book is written to serve as a text book for students in Software Engineering, Computer Science, Computer Engineering and Information Systems Engineering. Software professionals who want to familiarize themselves with formal methods can use this book as a good reference. The wide coverage of various formal specification techniques and the tutorial nature of descriptions of each individual technique make the book as a good resource for formal methods, all in one place. The bibliographic notes given at the end of each chapter provokes the reader to expand their horizon beyond the materials discussed in that chapter along with information on tool support.

Acknowledgments

Our sincere thanks go to the editorial board of Springer-Verlag, London whose persistent persuasion gave us sufficient motivation to engage in this venture. Many people have assisted us in bringing out the second edition of this book. First of all, those who have helped us during the first edition deserve a second round of applause. During the extensive revisions and additions to the second edition we received great support from Lei Feng, Pankaj Goyal, Naseem Ibrahim, Diep Mai, Ka Lok Man, Mubarak Mohammad, Shiri Nematollaah, and Olga Ormandjieva. We express our sincere thanks for their dedication and timely support.

Preface

This is a textbook on **software specification** emphasizing formal methods that are relevant to requirements and design stages of software development. The aim of the book is to teach the fundamental principles of formal methods in the construction of modular and verifiable formal specifications. The book introduces several formal specification techniques and illustrates the expressive power of each technique with a number of examples.

General Characteristics

Traditional textbooks on software engineering discuss the difficulties and challenges that lie on the path from requirements analysis to implementation of a software product. Most of these books describe some techniques in detail and give hints on implementation of these techniques. Only a few among them deal with important software engineering principles and techniques, and discuss how a particular technique may be used to implement a given principle. There is very little exposure in these books to a rigorous approach to, or a systematic study of, the construction of verifiable software. Those who have acquired an understanding of the fundamental principles of software engineering from traditional textbooks will find the following characteristics of this book quite relevant to the practice of software engineering.

- *The book deals with specification.*
  The principal characteristic of this book is to discuss formalisms that provide a theoretical foundation for the principles of software engineering, and are appropriate to the requirements and design stages of software development. We discuss the concept of abstraction, the need for formalism in software development, the mathematical basis of formal methods, components of a formal system, specification languages, different levels of rigor in applying languages, and the need for tool support to use formal methods for different stages of software development. We discuss the relationship between specifications and implementations, as well as subjecting specifications to rigorous analyses and formal proofs.

- *The book emphasizes mathematical principles.*
  Formal approaches to software development can be understood and practiced by studying the mathematics they use. A primary objective of the book is to relate discrete mathematical structures to the study of abstract data types, and to bring students to the level of mathematical maturity where they can write and reason about small specifications. Once the students acquire the basic mathematical skills that a formalism is based on, mastery of formal specification languages, techniques for refinements, and proofs become easy to understand and apply. We believe that the use of tools and techniques become effective when their underlying principles are properly understood.
- *The book teaches formal specification languages.*
  Unlike many recent books that are devoted to one formal specification language, we discuss four specification languages to emphasize their design philosophies and their practical applicability. We also discuss formal specifications based on set theory and logic without regard to any specification language. The purpose here is to teach the reader that these mathematical abstractions form the formal basis of the four specification languages. The languages discussed in the book are OBJ3, VDM, Z, and Larch. We illustrate their expressive power for different classes of applications. We expect that our treatment of the subject will prepare the reader to learn more sophisticated languages and tools that may be developed in the future. It is our belief that mastery of these languages will allow the reader to choose the language that is suitable for a given application.
- *The book presents proofs.*
  Informal arguments conducted in conjunction with a formal specification often lead to a proof construction, which can be presented in a justifiable manner. Proofs ensure a measure of certainty on claims that can be made of specified system properties. We present proofs in rigorous as well as in formal styles. We avoid lengthy proofs, and put more emphasis on modeling, specification, and rigorous reasoning of the specifications.
- *The book presents engineering principles.*
  This book discusses the general principles for data refinement, operation refinement, and interface specification, and illustrates how these are constructed for particular specification languages. The presentation in the book aims to enable the reader to understand *why* a particular technique is important and *how* to apply the technique.

Audience

This book is designed to be used as a textbook by students of computer science, software engineering, and information engineering. Software professionals who want to learn formal specification languages and use formal methods in their work will find the material in the book useful for serious self-study. The sections on bibliographic notes give a broad account of work related to the topic discussed in each chapter; this should help software professionals to identify industrial applications and learn from the experience reported on the use of tools.

Background Knowledge

The book is designed for undergraduates, and beginning graduate-level students in computer science, computer engineering, software engineering, and information engineering. We assume that the reader has completed an undergraduate course in discrete mathematics. The reader must be fluent in programming and must have completed or must be doing a course in software engineering. An exposure to undergraduate-level theoretical computer science course, or attainment of a certain level of *mathematical maturity* which enables the reader to abstract, conceptualize, and analytically reason about abstracted concepts will be an asset.

Organization and Content

Several specification languages, formal methods, and tools based on them have been developed by different research groups. Some of these methods are practiced by industries and government organizations such as NASA. Books devoted to one particular specification language or method have been published recently. Organizing the essential material to explore four specification languages in one textbook poses a challenge. We have organized this textbook based on the view that a reader should learn the following:

- where and how to integrate formalism in the development process,
- a mathematical basis, and
- the formal specification methods.

These are organized as follows:

- The first three chapters debate the questions: Why do we study formal specification? How do we integrate formal methods in a development process? What are the attributes for a formal specification language?
- Chapters 4 and 5 introduce the concept of abstraction and formalism, and discuss extensions to BNF and finite state machines, the two formal notations that the reader might have used in earlier courses.
- Chapters 6 and 7 discuss specifications based on logic, set theory and relations, and include material on proofs. Although the examples subjected to proofs are small, the structure of formal proofs is brought out clearly. These two chapters must be read carefully by those readers who want to review their mathematical knowledge.
- Chapters 8–11 describe the specification languages OBJ3, VDM, Z and Larch. We discuss the algebraic-specification methodology in Chap. 8, and include a tutorial on OBJ3. In Chap. 9, we introduce VDM, a model-based specification language. Chapter 10 deals with Z, another leading model-based notation built around set theoretical foundation. In Chap. 11, we discuss Larch and Larch/C++ specification languages. Our goal is to treat specification languages from abstract to concrete levels. Whereas representational details are ignored in an algebraic-specification language, VDM and Z specification languages use abstract data types as models for representing information of software

systems. The Larch family of languages are geared toward interface specification, and clearly separate the shareable abstraction from the programming language details. In our opinion, these four languages are representatives of several specification languages used for specifying sequential systems, and their features can be utilized in different application areas.

While the material in the first seven chapters should withstand the passage of time, it is likely that some of the material in Chaps. 8–11 may become outdated because of changes to the specification languages. The language OBJ3 has been around for a number of years, and its design principles are sound. The reader is expected to learn these principles; the syntax of the language or how OBJ3 system interprets a specification are secondary. We have used the ISO standardized notation for VDM in this book. The Z notation is also being standardized by ISO; however, the standardization process is not yet complete. Therefore, we have adopted an earlier version of Z. The Larch Shared Language (LSL), in which abstractions are developed, resembles an algebraic-specification language. However, the semantics of LSL is based on first-order logic. Given the impressive LSL library constructed by Guttag and Horning, we do not expect the syntax and the semantics of traits in the library to change much. However, the Larch/C++ interface specification language may undergo changes. The reader is advised to refer to the web page for Larch/C++ maintained by Gary Leavens for any update on the language. Since interface specification must be related to programming, and C++ is widely used in industry, we hope that the choice of Larch/C++ bridges the gap between design and implementation issues to be resolved by software professionals.

Exercises

All chapters include a section on exercises. There are three types:

- Exercises based on the basic concepts and aimed at extending the basic knowledge; these exercises include specifications and simple proofs.
- Extensions to examples discussed in the chapter; these require integration of the material discussed in the chapter.
- Project-oriented exercises that require complete specifications and proofs.

Case Studies

Case studies are used in Chaps. 8–11 to illustrate the features of OBJ3, VDM, Z, and Larch specification approaches. Each case study is chosen to demonstrate the integration of different concepts and features from a particular specification language. For example, the *Window* specification discussed in Chap. 8 demonstrates the integrated use of modular development and parametric specification concepts in OBJ3. This specification can be incrementally extended with additional operations, views, and theories toward reusing it in the design of another window management system. The *Network* example given in Chap. 9 is a simple version of a communication network. We have given a rigorous proof that the specification supports safe communication of messages between any two nodes in the network. The *Automated billing system* example presented in Chap. 10 is an instance of a real-life commercial application, which can be extended to suit a more complex situation. The case study in Chap. 11 presents Larch/C++ interface specifications for the two Rogue Wave library classes RWZone, and RWFile. These two examples are chosen to illustrate the applicability of Larch/C++ specification language to software products in commercial class libraries. The case studies may be read at different times and may be adapted or reused for different purposes.

Lab Components

The material in Chaps. 8–11 may be taught with tool-supported laboratory projects. In order to ensure that the students use the tool effectively, the instructors must (1) provide a solid foundation on theoretical issues, and (2) give assignments on simple specifications which can be done by pencil and paper. This will give students sufficient familiarity with the subject matter before they start learning to use the tools. The differences in syntactic conventions, and even minor differences in semantics between the specification language and the language employed by the tool must be overcome by the student. This implies that laboratory projects may only be introduced closer to the end of teaching the language; only then can the students' knowledge be expected to grow.

How to Use the Book

This book has evolved from the lecture notes prepared by the first author eight years ago. The notes were revised every year both for content and style. From the experience gained by both of us from the same notes in teaching different courses at different universities, we made extensive revisions to the notes in the last two years. However, the overall structure of the notes has not changed. Since the structure has withstood changes to the specification language details, such as syntax, we are confident that the different sequences as suggested below would fit different curriculum needs:

1. Chapters 1 through 3 are required for further reading of the book.
2. Chapters 4 and 5 may be read partially as well as simultaneously.
3. Based on the first seven chapters, a one-semester (13–14 weeks) undergraduate course within a software engineering program or computer science program or computer engineering program can be given.
4. Depending on the mathematical background of students in an information engineering program, material from Chaps. 1 through 7 may be selected and supplemented with basic mathematics to offer a one-semester course.

5. A two-semester course for graduates or senior undergraduates in software engineering, computer engineering, computer science, and information engineering programs can be given as follows:
   (a) Chapters 1 through 7 may be covered in semester I. One of the following sequences for semester II may be followed:
      - Chapters 8, 9
      - Chapters 8, 10
      - Chapters 9, 11
      - Chapters 10, 11
6. An advanced graduate-level course can be given by choosing the material from Chaps. 8 through 11 and supplementing it with intensive laboratory sessions requiring the verified development of a large project. This type of course requires tool support; for example, LP can be used with Larch, a theorem prover such as EVES or PVS may be used with Z or VDM. The material in the book may be supplemented with published papers in the area.

Acknowledgements

Our sincere thanks go to the many students and people who have helped us to create this book. We are grateful to the students of COMP 648 Systems Requirements Specification at Concordia University, and 74.716 Formal Specifications and Design at the University of Manitoba, for pointing out many of the errors in previous versions of the lecture notes.

Our deepest sense of gratitude go to Darmalingum Muthaiyen, who critically read the entire book, and gave us valuable feedback and corrections. His thorough reading and suggestion for presentable style have contributed greatly to the current version of the book.

We sincerely express our thanks to Dennis Lovie, Jonathan Jacky, and Randolph Johnson for reading and providing critical reviews on the Z notation. In particular, we greatly appreciate Randolph Johnson's comments on the semantics of some of the notations which helped us improve the chapter to its current version. We followed up Dennis Lovie's suggestions on uniformity of names and descriptions in the examples, which enhanced the readability of the chapter.

Jimmy Cheng and David So helped us in typesetting the first version of lecture notes eight years ago. Many of the LaTeX commands defined by them have been used in typesetting the current version of the book.

Finally, our thanks go toward everyone whose work has inspired us in writing this book.

Contents

Part I Specification Fundamentals

Part II Formalism Fundamentals

Part III Logic

Part IV Mathematical Abstractions for Model-Based Specifications

Part V Property-Oriented Specifications

Part VI Model-Based Specifications