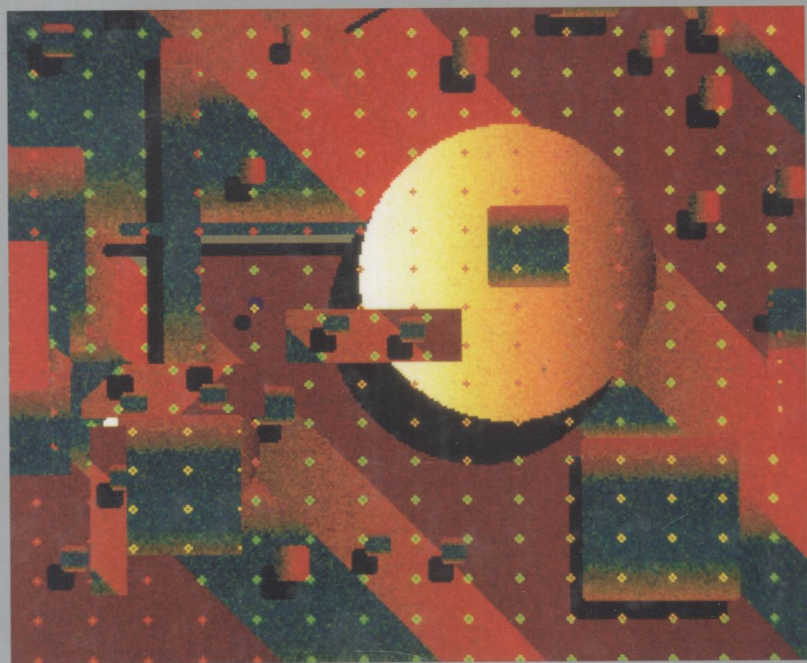


TEXTS IN COMPUTER SCIENCE

Software Reliability Methods



Doron A. Peled



Springer

TP31
P381

Doron A. Peled

SOFTWARE RELIABILITY METHODS

Foreword by Edmund M. Clarke

With 50 Illustrations



E200201401



Springer

Doron A. Peled
Computing Sciences
Bell Labs/Lucent Technologies
Murray Hill, NJ 07974, USA
doron@research.bell-labs.com

Series Editors

David Gries
Department of Computer Science
415 Boyd Graduate Studies Research
Center
The University of Georgia
Athens, GA 30602-7404, USA

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

Library of Congress Cataloging-in-Publication Data
Peled, Doron, 1962—

Software reliability methods / Doron A. Peled.
p. cm. — (Texts in computer science)

Includes bibliographical references and index.

ISBN 0-387-95106-7 (alk. paper)

1. Computer software—Reliability. I. Title. II. Series.

QA76.76.R44 P317 2001

005—dc21

2001018395

Printed on acid-free paper.

© 2001 Lucent Technologies. All Rights Reserved.

All rights reserved. This work may not be translated or copied in whole or in part without the written permission of the publisher (Springer-Verlag New York, Inc., 175 Fifth Avenue, New York, NY 10010 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 of general descriptive names, trade names, trademarks, etc., in this publication, even if they are not especially identified, is not to be taken as a sign that such names, as understood by the Trade Marks and Merchandise Marks Act, may accordingly be used freely by anyone.

Production managed by Allan Abrams; manufacturing supervised by Jerome Basma.

Photocomposed copy prepared from the author's \LaTeX files.

Printed and bound by Maple-Vail Book Manufacturing Group, York, PA.

Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-95106-7

SPIN 10774651

Springer-Verlag New York Berlin Heidelberg

A member of BertelsmannSpringer Science+Business Media GmbH

TEXTS IN COMPUTER SCIENCE

Editors

David Gries

Fred B. Schneider

Springer

New York

Berlin

Heidelberg

Barcelona

Hong Kong

London

Milan

Paris

Singapore

Tokyo

TEXTS IN COMPUTER SCIENCE

- Alagar and Periyasamy*, Specification of Software Systems
- Apt and Olderog*, Verification of Sequential and Concurrent Programs, Second Edition
- Back and von Wright*, Refinement Calculus
- Beidler*, Data Structures and Algorithms
- Bergin*, Data Structure Programming
- Brooks*, C Programming: The Essentials for Engineers and Scientists
- Brooks*, Problem Solving with Fortran 90
- Dandamudi*, Introduction to Assembly Language Programming
- Fitting*, First-Order Logic and Automated Theorem Proving, Second Edition
- Grillmeyer*, Exploring Computer Science with Scheme
- Homer and Selman*, Computability and Complexity Theory
- Immerman*, Descriptive Complexity
- Jalote*, An Integrated Approach to Software Engineering, Second Edition
- Kizza*, Ethical and Social Issues in the Information Age
- Kozen*, Automata and Computability
- Li and Vitányi*, An Introduction to Kolmogorov Complexity and Its Applications, Second Edition

(continued after index)

Foreword by Edmund M. Clarke

It is a great pleasure for me to write the foreword for Doron Peled's new book on software reliability methods. When I first opened the book, I was immediately impressed by the breadth of its coverage. It covers

- specification and modeling,
- deductive verification,
- model checking,
- process algebra,
- program testing, and
- state and message sequence charts.

In addition to describing the individual methods in considerable depth, it also discusses when each method is appropriate and the tradeoffs that are necessary in selecting among them. The different techniques are illustrated by many challenging exercises that can be used in conjunction with state of the art tools. It even tells where to access the tools on the web! I do not know of any other book that covers the same topics with such depth.

The book also describes the process of applying formal methods, starting with modeling and specification, then selecting an appropriate verification technique, and, finally, testing the resulting program. This knowledge is essential in practice, but is rarely covered in software engineering texts. Most books focus on a particular technique like program testing and do not cover other validation techniques or how several techniques can be used in combination. Because Doron has made significant contributions to the development of many of the validation techniques described in the book, his insights are particularly important on this critical issue.

The book is appropriate for a wide spectrum of people involved in the development of software. It is particularly appropriate for an upper level undergraduate level course on software reliability or a master's degree course in software engineering. In fact, it is sufficiently well annotated with pointers to other more advanced papers that it can be used as a reference source for software engineers engaged in code validation or by researchers in formal methods.

Having just completed a book on model checking with Doron, I am immensely impressed with both his talent as a computer scientist and his skill as a writer. I am sure that the present book will be an enormous success. I recommend it with great enthusiasm for anyone who is interested in the problem of software reliability.

Preface

Many books focus on increasing the quality of software through the use of formal methods. However, most books embrace one particular method, and present it as *the* suggested solution for the software reliability problem. This book presents a wider picture of formal methods, through a collection of notations and techniques. It compares them, and discusses their advantages and disadvantages.

One of the main challenges of formal methods is in transferring the technology developed by researchers to the software development community. Recently, we seem to be starting to have a better understanding of the important ingredients of formal methods tools. This manifests itself in the growing acceptance of such tools in the software and hardware development industry. Ideally, formal methods need to be intuitive to use (preferably using graphical interfaces), do not impose on the user an extensive learning period, and incur only small overhead to the development process. Formal methods are much more acceptable today than ten or twenty years ago, in particular in the hardware industry. Yet there is still a lively contention between different approaches.

The focus of this book is on describing the main principles of formal methods, through a collection of techniques. At the time of writing this book, there are already many advanced techniques that are not covered here. Techniques that deal with real-time and hybrid systems, advanced specification formalisms, and special data structures such as binary decision diagrams, were not included. The exclusion of specific material does not mean that the methods presented here are superior to the ones omitted. Nevertheless, the algorithms and methods described here are being used in state-of-the-art software reliability tools. The selection is merely intended to present the subject of formal methods in a way that seems deductive. However, it is impossible to refrain from preferring to include subjects that are closer to one's own research. The main themes used throughout this book are *logic* and *automata theory*. The interested reader can find details of advanced approaches in other books and research papers listed at the end of relevant chapters.

Studying formal methods is incomplete without hands-on experience with some tools. This book includes various exercises and projects, which may be performed using software reliability tools. There are several running examples

that are used in different chapters. An effective way to learn formal methods and their strengths and weaknesses is to follow these examples throughout the different chapters in which they occur. In some cases, a later chapter further elaborates on a running example that was presented as an exercise in a previous chapter. This also serves the purpose of helping readers check their solutions to previous exercises (instead of providing an explicit solution). The readers are encouraged to check if some of the additional intuition gained about the running example may help in improving their solutions to previous exercises.

Most exercises and projects presented here can be performed using a choice of tools. While some of the software reliability tools are subject to nontrivial license fees, many of them can be used free of charge for nonprofit purposes. This usually involves downloading the tool from its worldwideweb page and installing it according to the instructions provided there. At the end of relevant chapters, some tools and their corresponding web pages are listed. Notice that even tools that can be used without acquiring a license often require an agreement letter to be sent to the developers of the tool, committing to their terms of use. In many cases, such terms restrict the use of the tool for academic purposes only, and maintain no responsibility for damage that may be incurred by using it. Since web pages and web addresses tend to change, and since new tools are constantly being constructed, replacing existing ones, one cannot guarantee that the provided web information will remain up to date for long. Moreover, it is not guaranteed that the tools will work under any particular environment.

Different communities have different interests in formal methods. It is of course impossible to present a book that will appeal equally to managers, software developers, quality assurance teams and researchers alike. Nevertheless, I tried to include material that would be interesting to members of each one of these groups. Consequently, the reader may want to skip sections that may seem too theoretical, or too technical. It should be pointed out that the focus of this book is mainly on *techniques* rather than on *methodology*.

Some of the formal methods presented in this book are described together with the corresponding algorithm. Understanding the algorithms is usually not crucial for using the methods, but may give a deeper perspective on how they work. Most of the mathematical proofs involving the methods described were omitted. In some cases, proof sketches are included, to add more intuition.

The author would like to thank the following people for enlightening discussions and comments related to this book: Nina Amla, Christel Baier, David Basin, Shai Ben-David, Roderick Bloem, Glenn Bruns, Ed Clarke, Dennis Dams, Xiaoqun Du, Kousha Etessami, Amy Felty, Elsa Gunter, Doug Howe, Orna Kupferman, Bart Knaack, Bob Kurshan, Bengt Jonsson, Leonid Libkin, Anca Muscholl, Kedar Namjoshi, Wojciech Penczek, Kavita Ravi, Nataraajan Shankar, Natasha Sharygina, Marian Srenby, Richard Tefler, Wolfgang

Thomas, Moshe Vardi, Igor Walukiewicz, Thomas Wilke, Mihalis Yannakakis and Lenore Zuck. Indeed, one of the great benefits of writing such a book is the opportunity to further learn from the suggestions and comments of practitioners and experts of the particular subject.

Quoting from Lewis Carroll's adventure books is hardly original. However, it is little known that Charles Lutwidge Dodgson, who wrote under the pen name Lewis Carroll, was a mathematician interested in the visual representation of logic. His 'biliteral' and 'triliteral' diagrams are predecessors of Karnaugh maps, representing logic in a way that can be easily taught and understood, a recent trend in many formal methods.

Doron Peled, March 2001, Murray Hill, NJ

Contents

1. Introduction	1
1.1 Formal Methods	2
1.2 Developing and Acquiring Formal Methods	5
1.3 Using Formal Methods	7
1.4 Applying Formal Methods	9
1.5 Overview of the Book	11
2. Preliminaries	13
2.1 Set Notation	13
2.2 Strings and Languages	15
2.3 Graphs	16
2.4 Computational Complexity and Computability	20
2.5 Further Reading	27
3. Logic and Theorem Proving	29
3.1 First Order Logic	29
3.2 Terms	30
3.3 First Order Formulas	33
3.4 Propositional Logic	39
3.5 Proving First Order Logic Formulas	39
3.6 Properties of Proof Systems	43
3.7 Proving Propositional Logic Properties	46
3.8 A Practical Proof System	47
3.9 Example Proofs	50
3.10 Machine Assisted Proofs	59
3.11 Mechanized Theorem Provers	61
3.12 Further Reading	61
4. Modeling Software Systems	63
4.1 Sequential, Concurrent and Reactive Systems	64
4.2 States	67
4.3 State Spaces	68
4.4 Transition Systems	71
4.5 Granularity of Transitions	75

4.6	Examples of Modeling Programs	77
4.7	Nondeterministic Transitions	85
4.8	Assigning Propositional Variables to States	86
4.9	Combining State Spaces	88
4.10	The Linear View	90
4.11	The Branching View	91
4.12	Fairness	92
4.13	The Partial Order View	98
4.14	Modeling Formalisms	107
4.15	A Modeling Project	109
4.16	Further Reading	110
5.	Formal Specification	113
5.1	Properties of Specification Formalisms	114
5.2	Linear Temporal Logic	116
5.3	Axiomatizing LTL	121
5.4	Examples of LTL Specification	123
5.5	Automata on Infinite Words	127
5.6	Specification using Büchi-automata	129
5.7	Deterministic Büchi Automata	132
5.8	Alternative Specification Formalisms	133
5.9	Complicated Specifications	136
5.10	Completeness of Specification	136
5.11	Further Reading	138
6.	Automatic Verification	139
6.1	State Space Search	140
6.2	Representing States	143
6.3	The Automata Framework	143
6.4	Combining Büchi Automata	145
6.5	Complementing a Büchi Automaton	151
6.6	Checking Emptiness	152
6.7	A Model Checking Example	154
6.8	Translating LTL into Automata	156
6.9	The Complexity of Model Checking	164
6.10	Representing Fairness	169
6.11	Checking the LTL Specifications	170
6.12	Safety Properties	171
6.13	The State Space Explosion Problem	172
6.14	Advantages of Model Checking	174
6.15	Weaknesses of Model Checking	174
6.16	Selecting Automatic Verification Tools	175
6.17	Model Checking Projects	175
6.18	Model Checking Tools	176
6.19	Further Reading	177

7. Deductive Software Verification	179
7.1 Verification of Flow Chart Programs	180
7.2 Verification with Array Variables	187
7.3 Total Correctness	190
7.4 Axiomatic Program Verification	195
7.5 Verification of Concurrent Programs	202
7.6 Advantages of Deductive Verification	207
7.7 Weaknesses of Deductive verification	208
7.8 Soundness and Completeness of Proof Systems	210
7.9 Compositionality	212
7.10 Deductive Verification Tools	213
7.11 Further Reading	213
8. Process Algebra and Equivalences	215
8.1 Process Algebras	217
8.2 A Calculus of Communicating Systems	218
8.3 An Example: Dekker's Algorithm	225
8.4 Modeling Issues	229
8.5 Equivalences between Agents	230
8.5.1 Trace equivalence	231
8.5.2 Failure equivalence	232
8.5.3 Simulation Equivalence	233
8.5.4 Bisimulation and Weak Bisimulation equivalence	235
8.6 A Hierarchy of Equivalence Relations	237
8.7 Studying Concurrency using Process Algebra	238
8.8 Calculating Bisimulation Equivalence	242
8.9 LOTOS	245
8.10 Process Algebra Tools	247
8.11 Further Reading	247
9. Software Testing	249
9.1 Inspections and Walkthroughs	251
9.2 Control Flow Coverage Criteria	253
9.3 Dataflow Coverage Criteria	259
9.4 Propagating path conditions	261
9.5 Equivalence Partition	267
9.6 Preparing the Code for Testing	267
9.7 Checking the Test Suite	269
9.8 Compositionality	270
9.9 Black Box Testing	272
9.10 Probabilistic Testing	275
9.11 Advantages of Testing	276
9.12 Disadvantages of Testing	277
9.13 Testing Tools	278
9.14 Further Reading	278

10. Combining Formal Methods	279
10.1 Abstraction	279
10.2 Combining Testing and Model Checking	286
10.2.1 Direct Checking	286
10.2.2 Black Box Systems	287
10.2.3 Combination Lock Automata	288
10.2.4 Black Box Deadlock Detection	289
10.2.5 Conformance Testing	290
10.2.6 Checking the Reliability of Resets	294
10.2.7 Black Box Checking	294
10.3 The Cleanroom Method	297
11. Visualization	299
11.1 Using Visualizations for Formal Methods	300
11.2 Message Sequence Charts	300
11.3 Visualizing Flowcharts and State Machines	305
11.4 Hierarchical State Graphs	308
11.5 Visualizing Program Text	312
11.6 Petri Nets	312
11.7 Visualization Tools	314
11.8 Further Reading	316
12. Conclusions	317
References	321
Index	329

List of Figures

2.1	A directed graph	17
2.2	An unfolding of the graph in Figure 2.1	19
2.3	A Turing Machine	20
3.1	Proof trees	51
4.1	Executing commutative transitions	70
4.2	Executing noncommutative transitions	70
4.3	Some states of the Sieve of Eratosthenes for $N = 1$ and $P = 3$	82
4.4	Dekker's mutual exclusion solution	85
4.5	Propositional values for the states in Figure 4.3	87
4.6	Two local state spaces, and their asynchronous composition	90
4.7	The relation between executions and specification	91
4.8	A hierarchy of fairness criteria assumptions	97
4.9	A banking system	99
4.10	A partial order description of the bank	100
4.11	Processes interacting by message passing	104
4.12	A partial order execution of the program in Figure 4.11	105
4.13	Two interleavings	106
5.1	A model of a spring	118
5.2	A Büchi automaton	128
5.3	Two representations for a Büchi automaton	130
5.4	Mutual exclusion	132
5.5	A liveness property	132
5.6	An automaton accepting words where A holds finitely many times	132
5.7	Automaton for identifying words where A holds in even places ...	134
6.1	A DFS traversal of nodes with backward edges	142
6.2	Two automata to be intersected	147
6.3	Intermediate stage of intersection	147
6.4	The intersection of the automata in Figure 6.2	148
6.5	A traffic light model and the negation of a specification	155
6.6	The intersection of the automata in Figure 6.5	157
6.7	The translation algorithm	162

6.8	The initial node	163
6.9	Splitting the node in Figure 6.8	163
6.10	Splitting the right node in Figure 6.9	163
6.11	Generating a successor node	164
6.12	The set <i>Nodes_Set</i> at termination	165
6.13	The automaton with the accepting nodes	166
7.1	Nodes in a flowchart	180
7.2	A flowchart program for <i>integer</i> division	185
7.3	A proof tree	200
8.1	Two simple systems	216
8.2	Repeatedly executing β from $A \triangleq \alpha \beta.A$	223
8.3	The graph for $\alpha.(\beta.(\delta \bar{\delta}) + \gamma)$	224
8.4	An automaton representing a binary variable <i>c1</i>	226
8.5	Agent <i>P1</i>	228
8.6	Agents $\alpha.(\beta + \gamma)$ and $\alpha.\beta + \alpha.(\beta + \gamma)$	233
8.7	Agents $\alpha.\beta.\gamma.Nil + \alpha.\beta.\delta.Nil$ and $\alpha.(\beta.\gamma.Nil + \beta.\delta)$	235
8.8	Agents $\alpha.\beta + \alpha$ and $\alpha.\beta$	235
8.9	Two agents for Exercise 8.5.1	236
8.10	A hierarchy of equivalences	238
8.11	Agents <i>E</i> and <i>C</i>	240
8.12	Process algebra queues	241
8.13	The state spaces for agents <i>A</i> and <i>B</i>	243
9.1	A part of a flowchart	254
9.2	A hierarchy of coverage criteria	258
9.3	A hierarchy of dataflow coverage	261
9.4	A flowchart for the GCD program	265
9.5	A hierarchy of procedure calls	270
9.6	A combined approach of testing and model checking	273
9.7	A graph for testing	274
9.8	A simple Markov Chain	276
10.1	A combined approach of verification and model checking	280
10.2	An <i>n</i> slot buffer and its abstraction	282
10.3	A 4 slot buffer of bits	283
10.4	A combination lock automaton	289
10.5	Two nonconforming automata	290
10.6	A combination lock from a state of a black box automaton	293
10.7	A system where checking the reliability of reset is impossible ...	295
11.1	A Message Sequence Chart	301
11.2	The ITU-120 textual representation of the MSC in Figure 11.1 ..	301
11.3	The partial order between the events in Figure 11.1	302

11.4 An HMSC graph	304
11.5 A template and a matching scenario	305
11.6 Visualization of flowcharts	308
11.7 A simple hierarchical graph	309
11.8 Uniform exits	310
11.9 Concurrency within a state	311
11.10 A Petri Net	313
11.11 A Petri Net for the provisional mutual exclusion algorithm	315

1. Introduction

'Where shall I begin, please your Majesty?' he asked. 'Begin at the beginning,' the King said, very gravely, 'and go on till you come to the end; then stop.'

Lewis Carroll, *Alice's Adventures in Wonderland*

During late 1999, the world waited, with growing concern, for the change of the calendar into the year 2000. The focus was on some potential damage from online computers that control vital systems. This damage could occur because of the mere change of the calendar year and the way years of the twentieth century were traditionally represented in computer memory, using only the least two significant digits from 00 to 99. This surprisingly small detail made some people expect extreme damage. It could have affected electronic systems driven by software, such as verifying traffic control, atomic missiles, nuclear reactors, banking systems, pension plans, electricity and water supply. The US alone spent over 100 billion dollars on combating this, so called, 'Y2K-bug.' Just prior to that date change, some people had escaped into self made shelters, while flashlights and bottled water were a popular demand. Joint teams of the US and Russian military spent the night of December 31 1999 at the North American Aerospace Defense Command (NORAD). Together they monitored the world's skies, as a precaution against a possible computer error that could cause an unruly launch of missiles. Midnight, December 31 1999 has passed into the new millenium with no significant events, except for a few minor glitches.

Computer systems control many aspects of our lives. Telephone systems, store checkout registers, ticket reservation systems, medical systems, financial systems, are all highly computerized. Data communication between computers replaces, in most cases, the use of actual paper money transfer. Computers are even responsible for many of the activities required for flying commercial airplanes. Failure of computerized systems have already caused grave consequences, including fatal accidents, shutting down of vital systems, and loss of money.

The software development industry has grown over the last few decades at an unprecedented pace. Hardware, and in particular memory costs, kept decreasing. The internet has practically transformed the world into a big