

Principles of Concurrent and Distributed Programming

Visit the *Principles of Concurrent and Distributed Programming, Second Edition* Companion Website at www.pearsoned.co.uk/ben-ari to find valuable **student** learning material including:

- Source code for all the algorithms in the book
- Links to sites where software for studying concurrency may be downloaded.



We work with leading authors to develop the strongest educational materials in computing, bringing cutting-edge thinking and best learning practice to a global market.

Under a range of well-known imprints, including Addison-Wesley, we craft high quality print and electronic publications which help readers to understand and apply their content, whether studying or at work.

To find out more about the complete range of our publishing, please visit us on the World Wide Web at: www.pearsoned.co.uk

Principles of Concurrent and Distributed Programming

Second Edition

M. Ben-Ari



Harlow, England • London • New York • Boston • San Francisco • Toronto • Sydney • Singapore • Hong Kong
Tokyo • Seoul • Taipei • New Delhi • Cape Town • Madrid • Mexico City • Amsterdam • Munich • Paris • Milan

Pearson Education Limited
Edinburgh Gate
Harlow
Essex CM20 2JE
England

and Associated Companies throughout the world

Visit us on the World Wide Web at:
www.pearsoned.co.uk

First published 1990
Second edition 2006

© Prentice Hall Europe, 1990
© Mordechai Ben-Ari, 2006

The right of Mordechai Ben-Ari to be identified as author of this work has been asserted by him in accordance with the Copyright, Designs and Patents Act 1988.

All rights reserved. No part of this publication may be reproduced, stored in a retrieval system, or transmitted in any form or by any means, electronic, mechanical, photocopying, recording or otherwise, without either the prior written permission of the publisher or a licence permitting restricted copying in the united Kingdom issued by the Copyright Licensing Agency Ltd, 90 Tottenham Court Road, London W1T 4LP.

All trademarks used herein are the property of their respective owners. The use of any trademark in this text does not vest in the author or publisher any trademark ownership rights in such trademarks, nor does the use of such trademarks imply any affiliation with or endorsement of this book by such owners.

ISBN-13: 978-0-321-31283-9
ISBN-10: 0-321-31283-X

British Library Cataloguing-in-Publication Data

A catalogue record for this book is available from the British Library

Library of Congress Cataloging-in-Publication Data

A catalog record for this book is available from the Library of Congress

10 9 8 7 6 5 4 3 2 1
10 09 08 07 06

Printed and bound by Henry Ling Ltd, at the Dorset Press, Dorchester, Dorset

The publisher's policy is to use paper manufactured from sustainable forests.

Contents

Preface	xi
1 What is Concurrent Programming?	1
1.1 Introduction	1
1.2 Concurrency as abstract parallelism	2
1.3 Multitasking	4
1.4 The terminology of concurrency	4
1.5 Multiple computers	5
1.6 The challenge of concurrent programming	5
2 The Concurrent Programming Abstraction	7
2.1 The role of abstraction	7
2.2 Concurrent execution as interleaving of atomic statements	8
2.3 Justification of the abstraction	13
2.4 Arbitrary interleaving	17
2.5 Atomic statements	19
2.6 Correctness	21
2.7 Fairness	23
2.8 Machine-code instructions	24
2.9 Volatile and non-atomic variables	28
2.10 The BACI concurrency simulator	29
2.11 Concurrency in Ada	31

2.12	Concurrency in Java	34
2.13	Writing concurrent programs in Promela	36
2.14	Supplement: the state diagram for the frog puzzle	37
3	The Critical Section Problem	45
3.1	Introduction	45
3.2	The definition of the problem	45
3.3	First attempt	48
3.4	Proving correctness with state diagrams	49
3.5	Correctness of the first attempt	53
3.6	Second attempt	55
3.7	Third attempt	57
3.8	Fourth attempt	58
3.9	Dekker's algorithm	60
3.10	Complex atomic statements	61
4	Verification of Concurrent Programs	67
4.1	Logical specification of correctness properties	68
4.2	Inductive proofs of invariants	69
4.3	Basic concepts of temporal logic	72
4.4	Advanced concepts of temporal logic	75
4.5	A deductive proof of Dekker's algorithm	79
4.6	Model checking	83
4.7	Spin and the Promela modeling language	83
4.8	Correctness specifications in Spin	86
4.9	Choosing a verification technique	88
5	Advanced Algorithms for the Critical Section Problem	93
5.1	The bakery algorithm	93
5.2	The bakery algorithm for N processes	95
5.3	Less restrictive models of concurrency	96

5.4	Fast algorithms	97
5.5	Implementations in Promela	104
6	Semaphores	107
6.1	Process states	107
6.2	Definition of the semaphore type	109
6.3	The critical section problem for two processes	110
6.4	Semaphore invariants	112
6.5	The critical section problem for N processes	113
6.6	Order of execution problems	114
6.7	The producer–consumer problem	115
6.8	Definitions of semaphores	119
6.9	The problem of the dining philosophers	122
6.10	Barz’s simulation of general semaphores	126
6.11	Udding’s starvation-free algorithm	129
6.12	Semaphores in BACI	131
6.13	Semaphores in Ada	132
6.14	Semaphores in Java	133
6.15	Semaphores in Promela	134
7	Monitors	145
7.1	Introduction	145
7.2	Declaring and using monitors	146
7.3	Condition variables	147
7.4	The producer–consumer problem	151
7.5	The immediate resumption requirement	152
7.6	The problem of the readers and writers	154
7.7	Correctness of the readers and writers algorithm	157
7.8	A monitor solution for the dining philosophers	160
7.9	Monitors in BACI	162

7.10	Protected objects	162
7.11	Monitors in Java	167
7.12	Simulating monitors in Promela	173
8	Channels	179
8.1	Models for communications	179
8.2	Channels	181
8.3	Parallel matrix multiplication	183
8.4	The dining philosophers with channels	187
8.5	Channels in Promela	188
8.6	Rendezvous	190
8.7	Remote procedure calls	193
9	Spaces	197
9.1	The Linda model	197
9.2	Expressiveness of the Linda model	199
9.3	Formal parameters	200
9.4	The master–worker paradigm	202
9.5	Implementations of spaces	204
10	Distributed Algorithms	211
10.1	The distributed systems model	211
10.2	Implementations	215
10.3	Distributed mutual exclusion	216
10.4	Correctness of the Ricart–Agrawala algorithm	223
10.5	The RA algorithm in Promela	225
10.6	Token-passing algorithms	227
10.7	Tokens in virtual trees	230
11	Global Properties	237
11.1	Distributed termination	237

11.2	The Dijkstra–Scholten algorithm	243
11.3	Credit-recovery algorithms	248
11.4	Snapshots	250
12	Consensus	257
12.1	Introduction	257
12.2	The problem statement	258
12.3	A one-round algorithm	260
12.4	The Byzantine Generals algorithm	261
12.5	Crash failures	263
12.6	Knowledge trees	264
12.7	Byzantine failures with three generals	266
12.8	Byzantine failures with four generals	268
12.9	The flooding algorithm	271
12.10	The King algorithm	274
12.11	Impossibility with three generals	280
13	Real-Time Systems	285
13.1	Introduction	285
13.2	Definitions	287
13.3	Reliability and repeatability	288
13.4	Synchronous systems	290
13.5	Asynchronous systems	293
13.6	Interrupt-driven systems	297
13.7	Priority inversion and priority inheritance	299
13.8	The Mars Pathfinder in Spin	303
13.9	Simpson’s four-slot algorithm	306
13.10	The Ravenscar profile	309
13.11	UPPAAL	311
13.12	Scheduling algorithms for real-time systems	312

A	The Pseudocode Notation	317
B	Review of Mathematical Logic	321
B.1	The propositional calculus	321
B.2	Induction	323
B.3	Proof methods	324
B.4	Correctness of sequential programs	326
C	Concurrent Programming Problems	331
D	Software Tools	339
D.1	BACI and jBACI	339
D.2	Spin and jSpin	341
D.3	DAJ	345
E	Further Reading	349
	Bibliography	351
	Index	355

Supporting Resources

Visit www.pearsoned.co.uk/ben-ari to find valuable online resources

Companion Website for students

- Source code for all the algorithms in the book
- Links to sites where software for studying concurrency may be downloaded.

For instructors

- PDF slides of all diagrams, algorithms and scenarios (with \LaTeX source)
- Answers to exercises

For more information please contact your local Pearson Education sales representative or visit www.pearsoned.co.uk/ben-ari

Preface

Concurrent and distributed programming are no longer the esoteric subjects for graduate students that they were years ago. Programs today are inherently concurrent or distributed, from event-based implementations of graphical user interfaces to operating and real-time systems to Internet applications like multiuser games, chats and ecommerce. Modern programming languages and systems (including Java, the system most widely used in education) support concurrent and distributed programming within their standard libraries. These subjects certainly deserve a central place in computer science education.

What has not changed over time is that concurrent and distributed programs cannot be “hacked.” Formal methods *must* be used in their specification and verification, making the subject an ideal vehicle to introduce students to formal methods. Precisely for this reason I find concurrency still intriguing even after forty years’ experience writing programs; I hope you will too.

I have been very gratified by the favorable response to my previous books *Principles of Concurrent Programming* and the first edition of *Principles of Concurrent and Distributed Programming*. Several developments have made it advisable to write a new edition. Surprisingly, the main reason is not any revolution in the *principles* of this subject. While the superficial technology may change, basic concepts like *interleaving*, *mutual exclusion*, *safety* and *liveness* remain with us, as have the basic constructs used to write concurrent programs like *semaphores*, *monitors*, *channels* and *messages*. The central problems we try to solve have also remained with us: *critical section*, *producer–consumer*, *readers and writers* and *consensus*. What has changed is that concurrent programming has become ubiquitous, and this has affected the choice of language and software technology.

Language: I see no point in presenting the details of any particular language or system, details that in any case are likely to obscure the *principles*. For that reason, I have decided not to translate the Ada programs from the first edition into Java programs, but instead to present the algorithms in pseudocode. I believe that the high-level pseudocode makes it easier to study the algorithms. For example, in the

Byzantine Generals algorithm, the pseudocode line:

```
for all other nodes
```

is much easier to understand than the Java lines:

```
for (int i = 0; i < numberOfNodes; i++)  
    if (i != myID)
```

and yet no precision is lost.

In addition, I am returning to the concept of *Principles of Concurrent Programming*, where concurrency simulators, not concurrent programming languages, are the preferred tool for teaching and learning. There is simply no way that extreme scenarios—like the one you are asked to construct in Exercise 2.3—can be demonstrated without using a simulator.

Along with the language-independent development of models and algorithms, explanations have been provided on concurrency in five languages: the Pascal and C dialects supported by the BACI concurrency simulator, Ada¹ and Java, and Promela, the language of the model checker Spin. Language-dependent sections are marked by ^L. Implementations of algorithms in these languages are supplied in the accompanying software archive.

A word on the Ada language that was used in the first edition of this book. I believe that—despite being overwhelmed by languages like C++ and Java—Ada is still the best language for developing complex systems. Its support for concurrent and real-time programming is excellent, in particular when compared with the trials and tribulations associated with the concurrency constructs in Java. Certainly, the protected object and rendezvous are elegant constructs for concurrency, and I have explained them in the language-independent pseudocode.

Model checking: A truly new development in concurrency that justifies writing a revised edition is the widespread use of *model checkers* for verifying concurrent and distributed programs. The concept of a state diagram and its use in checking correctness claims is explained from the very start. Deductive proofs continue to be used, but receive less emphasis than in the first edition. A central place has been given to the Spin model checker, and I warmly recommend that you use it in your study of concurrency. Nevertheless, I have refrained from using Spin and its language Promela exclusively because I realize that many instructors may prefer to use a mainstream programming language.

¹All references to Ada in this book are to Ada 95.

I have chosen to present the Spin model checker because, on the one hand, it is a widely-used industrial-strength tool that students are likely to encounter as software engineers, but on the other hand, it is very “friendly.” The installation is trivial and programs are written in a simple programming language that can be easily learned. I have made a point of using Spin to verify all the algorithms in the book, and I have found this to be extremely effective in increasing my understanding of the algorithms.

An outline of the book: After an introductory chapter, Chapter 2 describes the abstraction that is used: the interleaved execution of atomic statements, where the simplest atomic statement is a single access to a memory location. Short introductions are given to the various possibilities for studying concurrent programming: using a concurrency simulator, writing programs in languages that directly support concurrency, and working with a model checker. Chapter 3 is the core of an introduction to concurrent programming. The critical-section problem is the central problem in concurrent programming, and algorithms designed to solve the problem demonstrate in detail the wide range of pathological behaviors that a concurrent program can exhibit. The chapter also presents elementary verification techniques that are used to prove correctness.

More advanced material on verification and on algorithms for the critical-section problem can be found in Chapters 4 and 5, respectively. For Dekker’s algorithm, we give a proof of freedom from starvation as an example of deductive reasoning with temporal logic (Section 4.5). Assertional proofs of Lamport’s fast mutual exclusion algorithm (Section 5.4), and Barz’s simulation of general semaphores by binary semaphores (Section 6.10) are given in full detail; Lamport gave a proof that is partially operational and Barz’s is fully operational and difficult to follow. Studying assertional proofs is a good way for students to appreciate the care required to develop concurrent algorithms.

Chapter 6 on semaphores and Chapter 7 on monitors discuss these classical concurrent programming primitives. The chapter on monitors carefully compares the original construct with similar constructs that are implemented in the programming languages Ada and Java.

Chapter 8 presents synchronous communication by channels, and generalizations to rendezvous and remote procedure calls. An entirely different approach discussed in Chapter 9 uses logically-global data structures called spaces; this was pioneered in the Linda model, and implemented within Java by JavaSpaces.

The chapters on distributed systems focus on algorithms: the critical-section problem (Chapter 10), determining the global properties of termination and snapshots (Chapter 11), and achieving consensus (Chapter 12). The final Chapter 13 gives an overview of concurrency in real-time systems. Integrated within this chapter

are descriptions of software defects in spacecraft caused by problems with concurrency. They are particularly instructive because they emphasize that some software really does demand precise specification and formal verification.

A summary of the pseudocode notation is given in Appendix A. Appendix B reviews the elementary mathematical logic needed for verification of concurrent programs. Appendix C gives a list of well-known problems for concurrent programming. Appendix D describes tools that can be used for studying concurrency: the BACI concurrency simulator; Spin, a model checker for simulating and verifying concurrent programs; and DAJ, a tool for constructing scenarios of distributed algorithms. Appendix E contains pointers to more advanced textbooks, as well as references to books and articles on specialized topics and systems; it also contains a list of websites for locating the languages and systems discussed in the book.

Audience: The intended audience includes advanced undergraduate and beginning graduate students, as well as practicing software engineers interested in obtaining a scientific background in this field. We have taught concurrency successfully to high-school students, and the subject is particularly suited to non-specialists because the basic principles can be explained by adding a very few constructs to a simple language and running programs on a concurrency simulator. While there are no specific prerequisites and the book is reasonably self-contained, a student should be fluent in one or more programming languages and have a basic knowledge of data structures and computer architecture or operating systems.

Advanced topics are marked by ^A. This includes material that requires a degree of mathematical maturity.

Chapters 1 through 3 and the non-^A parts of Chapter 4 form the introductory core that should be part of any course. I would also expect a course in concurrency to present semaphores and monitors (Chapters 6 and 7); monitors are particularly important because concurrency constructs in modern programming languages are based upon the monitor concept. The other chapters can be studied more or less independently of each other.

Exercises: The exercises following each chapter are technical exercises intended to clarify and expand on the models, the algorithms and their proofs. Classical problems with names like the *sleeping barber* and the *cigarette smoker* appear in Appendix C because they need not be associated with any particular construct for synchronization.

Supporting material: The companion website contains an archive with the source code in various languages for the algorithms appearing in the book. Its address is:

<http://www.pearsoned.co.uk/ben-ari>.

Lecturers will find slides of the algorithms, diagrams and scenarios, both in ready-to-display PDF files and in \LaTeX source for modification. The site includes instructions for obtaining the answers to the exercises.

Acknowledgements: I would like to thank:

- Yifat Ben-David Kolikant for six years of collaboration during which we learned together how to really teach concurrency;
- Pieter Hartel for translating the examples of the first edition into Promela, eventually tempting me into learning Spin and emphasizing it in the new edition;
- Pieter Hartel again and Hans Henrik Løvengreen for their comprehensive reviews of the manuscript;
- Gerard Holzmann for patiently answering innumerable queries on Spin during my development of jSpin and the writing of the book;
- Bill Bynum, Tracy Camp and David Strite for their help during my work on jBACI;
- Shmuel Schwarz for showing me how the frog puzzle can be used to teach state diagrams;
- The Helsinki University of Technology for inviting me for a sabbatical during which this book was completed.

M. Ben-Ari
Rehovot and Espoo, 2005

