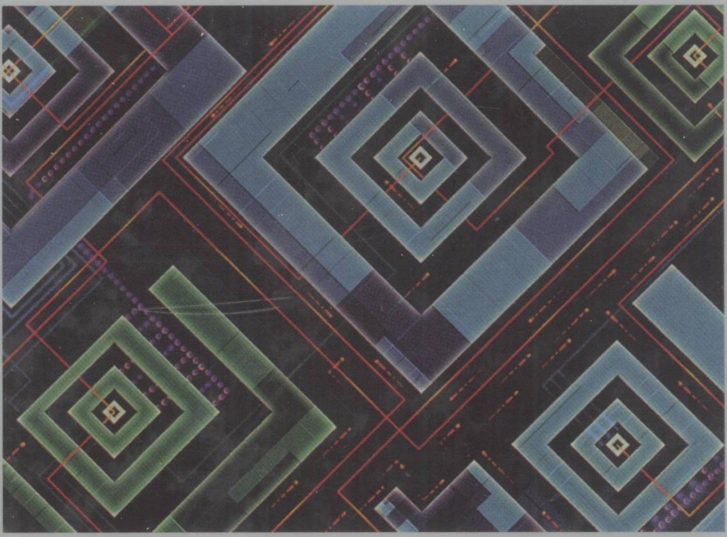
GRADUATE TEXTS IN COMPUTER SCIENCE

Verification of Sequential and Concurrent Programs



Krzysztof R. Apt
Ernst-Rüdiger Olderog

SECOND EDITION

Springer

TP31
A655
E.2

Krzysztof R. Apt   Ernst-Rüdiger Olderog

VERIFICATION OF SEQUENTIAL AND CONCURRENT PROGRAMS

Second Edition

Springer

Krzysztof R. Apt
Stichling Mathematisch Centrum
Centrum voor Wiskunde en Informatica
Kruislaan 413
1098 SJ Amsterdam
Amsterdam, The Netherlands

Ernst-Rüdiger Olderog
Christian Albrechts University
Fachbereich 10-Theoretische Informatik
2900 Oldenburg
Oldenburg, Germany

Series Editors
David Gries
Fred B. Schneider

Department of Computer Science
Cornell University
Upson Hall
Ithaca, NY 14853-7501, USA

Library of Congress Cataloging-in-Publication Data
Apt, Krzysztof R., 1949–
Verification of sequential and concurrent programs / Krzysztof R. Apt, Ernst-Rüdiger Olderog. — 2nd ed.
p. cm. — (Graduate texts in computer science)
Includes bibliographical references and indexes.
ISBN 0-387-94896-1 (hardcover : alk. paper)
1. Computer software—verification. I. Olderog, E.-R. II. Title. III. Series: Graduate texts in computer science (Springer-Verlag New York Inc.)
QA76.76.V47A67 1997
005.1´4—dc21 96-29771

Printed on acid-free paper.

Production managed by Allan Abrams; manufacturing supervised by Johanna Tschebull.
Photocomposed copy provided from the authors' LaTeX file.
Printed and bound by Hamilton Printing Co., Rensselaer, NY.
Printed in the United States of America.

9 8 7 6 5 4 3 2 1

ISBN 0-387-94896-1 Springer-Verlag New York Berlin Heidelberg SPIN 10523521

Preface

Computer programs are becoming more and more part of systems that we use or rely on in our daily lives. Numerous examples include booking terminals in travel agencies, automatic teller machines, ever more sophisticated services based on telecommunication, signaling systems for cars and trains, luggage handling systems at airports or automatic pilots in airplanes.

For the customers of travel agencies and banks and for the passengers of trains and airplanes the proper functioning and safety of these systems is of paramount importance. Money orders should reflect the right bank accounts and the airplanes should stay on the desired route. Therefore the underlying computer programs should work correctly; that is they should satisfy their requirements. A challenge for computer science is to develop methods that ensure program correctness.

Common to the applications mentioned above is that the computer programs have to coordinate a number of system components that can work concurrently, for example the terminals in the individual travel agencies accessing a central database or the sensors and signals used in a distributed railway signaling system. So to be able to verify such programs we need to have at our disposal methods that allow us to deal with correctness of concurrent programs, as well.

Structure of This Book

The aim of this book is to provide a systematic exposition of one of the most common approaches to program verification. This approach is usually called assertional, because it relies on the use of assertions that are attached to program control points. Starting from sequential programs we proceed in a systematic manner to concurrent programs, both parallel and distributed.

We consider here sequential programs in the form of deterministic and nondeterministic programs, and concurrent programs in the form of parallel and distributed programs. Parallel programs consist of several sequential components that can access shared memory. By contrast, distributed programs consist of components with local memory that can communicate only by sending and receiving messages.

For each of these classes of programs their input/output behavior in the sense of so-called partial and total correctness is studied. For the verification of these correctness properties an axiomatic approach involving assertions is used. This approach was initiated by Hoare in 1969 for deterministic programs and extended by various researchers to other classes of programs. It is combined here with the use of program transformations.

For each class of programs a uniform presentation is provided. After defining the syntax we introduce a structured operational semantics as originally proposed by Hennessy and Plotkin in 1979 and further developed by Plotkin in 1981. Then proof systems for the verification of partial and total correctness are introduced.

The use of these proof systems is demonstrated with the help of case studies. In particular, solutions to classical problems such as producer/consumer and mutual exclusion are formally verified. Each chapter concludes with a list of exercises and bibliographic remarks.

The exposition assumes elementary knowledge of programming languages and logic. Therefore this book belongs to the area of programming languages but at the same time it is firmly based on mathematical logic. All prerequisites are provided in the preparatory Chapter 2.

In Chapter 3 Hoare's approach to program verification is explained for a simple class of deterministic sequential programs, known as **while**-programs. Next parallel programs with shared variables are studied. Since these are much more difficult to deal with, they are introduced in a stepwise manner in Chapters 4, 5 and 6. We base our presentation on the approach by Owicki and Gries originally proposed in 1976 and on an extension of it by the authors and de Boer dealing with total correctness.

Nondeterministic sequential programs are studied in Chapter 7. The presentation is based on the work of Dijkstra from 1976 and Gries from 1981.

Study of this class of programs also serves as a preparation for dealing with distributed programs in Chapter 8. The verification method presented there is based on a transformation of distributed programs into nondeterministic ones proposed by the first author in 1986. In Chapter 9 the issue of fairness is studied in the framework of nondeterministic programs. The approach is based on the method of explicit schedulers developed by the authors in 1983.

Teaching from This Book

In the first lecture the zero search example in Chapter 1 should be discussed. This example demonstrates which subtle errors can arise during the design of parallel programs. Next we recommend moving on to Chapter 3 on deterministic programs and before each of the sections on syntax, semantics and verification, to refer to the corresponding sections of the preparatory Chapter 2.

Afterwards the treatment of parallel programs in Chapters 4, 5 and 6 can follow. Then, after a short presentation of Chapter 7 on nondeterministic programs, Chapter 8 on distributed programs can be treated. Alternatively, it is possible to skip Chapters 4 – 6 and move immediately to Chapters 7 and 8.

Chapter 9 on fairness covers a more advanced topic and can be used during specialized seminars.

Changes in the Second Edition

The first edition of this book appeared in the series entitled "Texts and Monographs in Computer Science." The series title reflected both aspects of the book: it could be used as a textbook and some parts of it had a monographic character.

This edition of the book appears in the series entitled "Graduate Texts in Computer Science" and reflects the shift of emphasis. It is aimed to be used as a textbook and consequently it substantially differs from the first edition. In particular:

- The book has been substantially shortened. This makes it possible to use it in its entirety as a textbook for a one semester course on program verification.

- The order of the chapters has been rearranged. In this edition the presentation of nondeterministic programs (previously Chapter 4) has been moved and appears now as Chapter 7. In this way parallelism is introduced earlier and the transition from deterministic programs (Chapter 3) to parallel programs (Chapters 4–6) and from nondeterministic programs (Chapter 7) to distributed programs (Chapter 8) is smoother.

- The subject of systematic program derivation from specifications is now introduced earlier, in Chapter 3. This recognizes the growing importance of this field.

- Various points, such as the discussion of the proof rules and of semantics, are more systematically explained and amplified. In particular, there is a clearer presentation of the completeness proofs in Chapter 3.

- In our view fairness is a more advanced topic the study of which can be skipped in a basic course on program verification. However, it is an important concept pervading many areas of computing. In Chapter 9 we deal with program verification under the fairness assumption in its simplest setting: that of nondeterministic programs studied in Chapter 7. The method provided there can be used for a study of various fairness notions for several classes of programs, including all those studied in this book.

This edition of the book follows in many ways its German edition which appeared as a textbook in the series "Springer-Lehrbuch" in 1994. It reflects our experiences in teaching a course on program verification to graduate students at the Universities of Amsterdam, Oldenburg and Pisa.

Acknowledgments

The authors of this book have collaborated, often together with other colleagues, on the topic of program verification since 1979. During this time we have benefited very much from discussions with Jaco de Bakker, Frank de Boer, Luc Bougé, Ed Clarke, Edsger W. Dijkstra, Nissim Francez, David Gries, Tony Hoare, Shmuel Katz, Leslie Lamport, Hans Langmaack, Jay Misra, Amir Pnueli, Gordon Plotkin, Willem Paul de Roever, Fred Schneider, Jonathan Stavi and Jeffery Zucker. Many thanks to all of them.

Also, we would like to thank Franco Turini for useful comments on this version of the book. Most of the LATEXscript of the first edition was typeset by Thomas Woo at the University of Texas at Austin. The changes for this

second edition were performed by the authors themselves with the expert assistance of Arvid Hülsebus and Christian Kühnke at the University of Oldenburg. The bibliography style used in this book has been designed by Sam Buss; Anne Troelstra deserves credit for drawing our attention to it.

Finally, we would like to thank the staff of Springer-Verlag, in particular Allan Abrams and Martin Gilchrist, for the efficient and professional handling of all the stages of the production of this book.

Amsterdam, The Netherlands — Krzysztof R. Apt
Oldenburg, Germany — Ernst-Rüdiger Olderog

Contents

1 Introduction

Program verification is a systematic approach to proving the correctness of programs. Correctness means that the programs enjoy certain desirable properties. For sequential programs these properties are delivery of correct results and termination. For concurrent programs, that is, those with several active components, the properties of interference freedom, deadlock freedom and fair behavior are also important.

The emphasis in this book is on verification of concurrent programs, in particular of parallel and distributed programs where the components communicate either via shared variables or explicit message passing. Such programs are usually difficult to design, and errors are more a rule than an exception. Of course, we also consider sequential programs because they occur as components of concurrent ones.

1.1 An Example of a Concurrent Program

To illustrate the subtleties involved in the design of concurrent programs consider the following simple problem.

Problem Let f be a function from integers to integers with a zero. Write a concurrent program *ZERO* that finds such a zero.

The idea is to solve the problem by splitting it into two subproblems that can be solved independently, namely finding a positive and a nonpositive zero. Here z is called a *positive zero* of f if $z > 0$ and $f(z) = 0$, and it is called a *nonpositive zero* if $z \leq 0$ and $f(z) = 0$. We are now looking for sequential programs S_1 and S_2 solving the two subproblems such that the parallel execution of S_1 and S_2 solves the overall problem. We write $[S_1 \| S_2]$ for a parallel composition of two sequential programs S_1 and S_2. Execution of $[S_1 \| S_2]$ consists of executing the individual statements of S_1 and S_2 in parallel. The program $[S_1 \| S_2]$ terminates when both S_1 and S_2 terminate.

Solution 1

Consider the following program S_1:

```
$S_1 \equiv found := \textbf{false};\ x := 0;$
        while $\neg found$ do
            $x := x + 1;$
            $found := f(x) = 0$
        od.
```

S_1 terminates when a positive zero of f is found. Similarly, the following program S_2 terminates when a nonpositive zero of f is found:

```
$S_2 \equiv found := \textbf{false};\ y := 1;$
        while $\neg found$ do
            $y := y - 1;$
            $found := f(y) = 0$
        od.
```

Thus the program

$$ZERO\text{-}1 \equiv [S_1 \| S_2],$$

the parallel composition of S_1 and S_2, appears to be a solution to the problem. Note that the Boolean variable *found* can be accessed by both components S_1 and S_2. This shared variable is used to exchange information about termination between the two components. □