PROGRAMS, RECURSION AND UNBOUNDED CHOICE

Cambridge Tracts in Theoretical Computer Science

Titles in the series

PROGRAMS, RECURSION AND UNBOUNDED CHOICE

Predicate-Transformation Semantics and Transformation Rules

WIM H. HESSELINK
Department of Computer Science
University of Groningen

CAMBRIDGE
UNIVERSITY PRESS

CAMBRIDGE UNIVERSITY PRESS
Cambridge, New York, Melbourne, Madrid, Cape Town, Singapore, São Paulo

Cambridge University Press
The Edinburgh Building, Cambridge CB2 2RU, UK

Published in the United States of America by Cambridge University Press, New York

www.cambridge.org
Information on this title: www.cambridge.org/9780521404365

First published 1992
This digitally printed first paperback version 2005

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

ISBN-13 978-0-521-40436-5 hardback
ISBN-10 0-521-40436-3 hardback

ISBN-13 978-0-521-01829-6 paperback
ISBN-10 0-521-01829-3 paperback

CONTENTS

PREFACE

This book is about programs as mathematical objects. We focus on one of the aspects of programs, namely their functionality, their meaning or semantics. Following Dijkstra we express the semantics of a program by the weakest precondition of the program as a function of the postcondition. Of course, programs have other aspects, like syntactic structure, executability and (if they are executable) efficiency. In fact, perhaps surprisingly, for programming methodology it is useful to allow a large class of programs, many of which are not executable but serve as partially implemented specifications.

Weakest preconditions are used to define the meanings of programs in a clean and uniform way, without the need to introduce operational arguments. This formalism allows an effortless incorporation of unbounded nondeterminacy. Now programming methodology poses two questions. The first question is, given a specification, to design a general program that is proved to meet the specification but need not be executable or efficient, and the second question is to transform such a program into a more suitable one that also meets the specification.

We do not address the methodological question how to design, but we concentrate on the mathematical questions concerning semantic properties of programs, semantic equality of programs and the refinement relation between programs. We provide a single formal theory that supports a number of different extensions of the basic theory of computation.

The correctness of a program with respect to a specification is for us only one of its semantic properties. We are equally interested in its incorrectness or its equivalence to other programs. For example, we provide formal rules to prove that a recursive procedure does not meet its specification.

The book can be used for courses on predicate–transformation semantics of various sizes. For an introductory course, I prefer to use Chapters 1, 2, 3 and 4, but Chapter 4 can be replaced by subjects from Chapter 5 or by Chapter 6, possibly followed by 7. The more powerful methods of program transformation are contained in Chapters 10, 11 and 12; these chapters should be accessible after 4, 6

and 7 have been covered. Chapter 14 on temporal predicate transformers can be treated directly after Chapter 4.

Some of the material grew out of courses of lectures on programming or programming theory delivered at the University of Groningen. Other parts of the book grew out of my need for a non–operational understanding of standard implementation techniques like the use of stacks for the implementation of recursive procedures. Chapters 14 and 15 were directly inspired by [Morris 1990].

I offer sincere thanks to A. de Bruin, J.J. Lukkien, J. Morris, R. Reinds, J.C.S.P. van der Woude and J. von Wright for critically reading drafts of the book. I also want to express here my deepest gratitude to the three persons who have stimulated and guided my transition from mathematics to computer science: J.W. de Bakker, E.W. Dijkstra and J.L.A. van de Snepscheut. Van de Snepscheut was for five years the inspiring leader of our department. Under his direction I got the opportunity to spend a sabbatical year at the University of Texas at Austin, in 1986/1987. There I worked with Dijkstra, who converted me to a new mathematical discipline, ten years after I had obtained my doctorate in mathematics. Finally, De Bakker showed continuous interest in my computer science papers and encouraged me to write this book. Of course, there were many more persons who stimulated my development but no enumeration can do justice to all of them.

Groningen, the Netherlands — Wim H. Hesselink
June 1991

LIST OF SYMBOLS

Symbols are listed by section number.

Postfix operators

$-^*$ 4.3
$-^\odot$ 4.3, 10.1
$-^e$ (where $e = 0$ or 1) 4.4
$\heartsuit$ 0.0

Special parentheses

$[\ \ ]$ 0.0, 1.1
$[\ /\ ]$ 10.1
$\{\ \ \}$ 0.6, 2.1
$[\![\ \ ]\!]$ 6.4

Programming operators

! 1.3
? 1.3
; 1.4, 4.7, 6.2
$\|$ 1.4, 4.7, 6.2
$[\![]\!]_f$ 15.2

Relational operators

$\cong$ 0.3, 1.3
$=_V$ 1.6
$\neq_V$ 1.6
$\vdash$ 2.7
$\leq$ 4.1
$\sqsubseteq$ 5.1
$<$ 5.6
$\approx$ 11.3, 12.3
$\ll$ 12.2, 13.3