Formal Specification and Design

L. M. G. Feijs &
H. B. M. Jonkers

Cambridge Tracts in Theoretical Computer Science 35

FORMAL SPECIFICATION AND DESIGN

L.M.G. FEIJS & H.B.M. JONKERS
Philips Research Laboratories
Eindhoven

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/9780521434577

© Cambridge University Press 1992

This publication is in copyright. Subject to statutory exception
and to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 1992
This digitally printed first paperback version 2005

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

ISBN-13 978-0-521-43457-7 hardback
ISBN-10 0-521-43457-2 hardback

ISBN-13 978-0-521-43592-5 paperback
ISBN-10 0-521-43592-7 paperback

FORMAL SPECIFICATION AND DESIGN

Cambridge Tracts in Theoretical Computer Science

Managing Editor Professor C.J. van Rijsbergen,
Department of Computing Science, University of Glasgow

Editorial Board

S. Abramsky, Department of Computing Science, Imperial College of Science and Technology
P.H. Aczel, Department of Computer Science, University of Manchester
J.W. de Bakker, Centrum voor Wiskunde en Informatica, Amsterdam
J.A. Goguen, Programming Research Group, University of Oxford
J.V. Tucker, Department of Mathematics and Computer Science, University College of Swansea

Titles in the series

1. G. Chaitin *Algorithmic Information Theory*
2. L.C. Paulson *Logic and Computation*
3. M. Spivey *Understanding Z*
4. G. Revesz *Lambda Calculus, Combinators and Functional Programming*
5. A. Ramsay *Formal Methods in Artificial Intelligence*
6. S. Vickers *Topology via Logic*
7. J-Y. Girard, Y. Lafont & P. Taylor *Proofs and Types*
8. J. Clifford *Formal Semantics & Pragmatics for Natural Language Processing*
9. M. Winslett *Updating Logical Databases*
10. K. McEvoy & J.V. Tucker (eds) *Theoretical Foundations of VLSI Design*
11. T.H. Tse *A Unifying Framework for Stuctured Analysis and Design Models*
12. G. Brewka *Nonmonotonic Reasoning*
14. S.G. Hoggar *Mathematics for Computer Graphics*
15. S. Dasgupta *Design Theory and Computer Science*
17. J.C.M. Baeten (ed) *Applications of Process Algebra*
18. J.C.M. Baeten & W. P. Weijland *Process Algebra*
23. E.-R. Olderog *Nets, Terms and Formulas*
26. P.D. Mosses *Action Semantics*
27. W.H. Hesselink *Programs, Recursion and Unbounded Choice*
29. P. Gärdenfors (ed) *Belief Revision*
30. M. Anthony & N. Biggs *Computational Learning Theory*
33. E.G. Manes *Predicate Transformer Semantics*
34. F. Nielson & H.R. Nielson *Two Level Functional Languages*
35. L. Feijs & H. Jonkers *Formal Specification and Design*

此为试读，需要完整PDF请访问： www.ertongbook.com

Preface

This book is about formal specification and design techniques, including both algebraic specifications and state-based specifications.

The construction and maintenance of complex software systems is a difficult task and although many software projects are started with great expectations and enthusiasm, it is too often the case that they fail to achieve their goals within the planned time and with the given resources. The software often contains errors; attempts to eliminate the errors give rise to new errors, and so on. Moreover, the extension and adaptation of the software to new tasks turns out to be a difficult and tedious task, which seems unsuitable for scientific methods.

This unsatisfactory situation can be improved by introducing precise specifications of the software and its constituent parts. When a piece of software P has a precise specification S say, then 'P satisfies S' is a clear statement that could be verified by reasoning or that could be falsified by testing; users of P can read S and rely on it and the designer of P has a clearly formulated task. When no precise specifications are available, there are hardly any clear statements at all, for what could one say: 'it works' or more often 'it almost works'? Without precise specifications, it becomes very difficult to analyse the consequences of modifying P into P', for example, and to make any clear statements about that modification. Therefore it is worthwhile during the software development process to invest in constructing precise specifications of well-chosen parts of the software system under construction. Writing precise specifications turns out to be a considerable task itself. In many situations the use of natural language, pictures and pseudo-code does not yield specifications of the required level of abstractness and precision. *Formal specification* is an approach to writing precise specifications, building on concepts from mathematical logic. During the past decades, much research and development concerning formal specification techniques has been conducted. Well-known results in this field are the techniques of 'abstract data types' and of 'pre- and postconditions'.

What is the role of 'language' in connection with formal specifications? One can say that, in many respects, the practical progress in software engineering is language-driven: it is hard to introduce methodological concepts unless these are concretely available as constructs in the language in use. This is a major motivation behind the introduction of formal specification languages. In practice it is not enough to have good methodological concepts for writing formal specifications: one needs a language as a vehicle. Throughout this book, the language COLD-K is employed as a vehicle. COLD is an acronym for *Common Object-oriented Language for Design.* This book explains the constructs offered by the language and shows how to use them. The use of formal specification techniques at certain well-chosen points in the design process is one of the key factors – though certainly not the only one – in increasing the quality of the software development process.

A formal specification language is a language whose constructs are derived both from mathematical logic and from programming languages and which has a precise syntax and semantics. If, furthermore the language allows for descriptions at several levels of abstraction, it is called a *wide-spectrum* specification language. The language employed in this book is such a wide-spectrum specification language; other wide-spectrum specification languages are VDM, CIP and RSL. One could also call it a *design language* to emphasise that the language can be used for recording a software system in its intermediate stages of design, ranging from *specification* to *implementation.*

Many techniques such as 'abstract data types', 'abstraction functions', 'invariants', 'pre- and postconditions', 'modular specification' and 'information hiding' can be explained using the constructs offered by COLD-K [1]. The language is in the tradition of VDM [2, 3] and Z [4, 5], but has been influenced by ASL [6], Module Algebra [7], Harel's dynamic logic [8], Scott's E-logic [9, 10] and object-oriented languages. Furthermore it contains a novel notion of 'design' comparable with the structuring mechanisms provided by e.g. HOOD [11].

The language was developed at the Philips Research Laboratories in Eindhoven within the framework of ESPRIT project 432 (also known as METEOR). It has been designed mainly by H.B.M. Jonkers, with technical contributions from C.P.J. Koymans, G.R. Renardel de Lavalette and L.M.G. Feijs. The fact that its well-formedness and semantics are defined mathematically guarantees that descriptions in the language leave no room for ambiguity and that a high level of tool support can be provided. Actually, COLD-K is one out of a sequence of language versions, in which it plays a special role: it is a *kernel language*, serving as a point of departure in the further development of the language. It is meant to be used as the kernel of user- and application-oriented language versions, to be derived by syntactic extensions. All essential semantic features are contained in this kernel language, as well as high level constructs for modularisation, parameterisation and designs. It is important to realise that this language is the forerunner of versions which are much more user-friendly – at least from a syntactic point of view. Indeed, certain aspects of the language are somewhat Spartan, but for the purpose of this book this is hardly a disadvantage.

One of the main goals of this book is to treat the basic concepts underlying algebraic specification techniques. The book shows how algebraic specification techniques can be effectively used in the software development process. Yet, the approach of this book goes far beyond algebraic specifications: it shows how algebraic and state-based techniques can be combined in an integrated approach. The main motivation for using COLD-K is as follows. It is a *formal* language, with a well-defined syntax and semantics; it can be used as an *algebraic* specification language; furthermore, it is an *integrated* language, unifying algebraic and state-based techniques.

The book is divided into three parts. The first part is concerned with algebraic specifications, the second part with state-based specifications. The first and the second part have essentially the same structure, beginning with an introduction of the basic concepts followed by methodological guidelines about setting up a specification. After that the the topics of *large* specifications as well as *implementation* strategies are treated – each in one chapter. In the third part we have three chapters, each of a different nature. In Chapter 9 a number of existence proofs and theoretical discussions are presented. These are related to the earlier chapters, but are not presented there in order not to distract too much from the main line of these chapters. In Chapter 10 a number of additional language constructs are presented informally. In the last chapter (Chapter 11) a pictorial representation of module structures is explained as well as a systematic approach for putting specifications and implementations together in a top-level language construct called *design*. The structure of the book is as follows.

I
1. Introducing the basic concepts,
2. Setting up algebraic specifications,
3. Structuring algebraic specifications,
4. Implementing algebraic specifications.

II
5. From algebras to states,
6. Setting up state-based specifications,
7. Structuring state-based specifications,
8. Implementing state-based specifications.

III
9. Theoretical topics,
10. Additional language constructs,
11. Towards large systems.

Since the goal of the book is not to serve as a language reference manual, we decided not to present *all* language features. Instead of that we restricted ourselves to the most essential aspects of the language. These are covered in depth by the Chapters 1–9, and additional language features as well as constructs for building systems from components are dealt with briefly and informally in Chapters 10 and 11. The syntax of the full language COLD-K is contained in Appendix A.

This book developed from the course material for a post-graduate course given by the authors at the Technical University of Eindhoven and at the University of Nijmegen. Special thanks go to J.A. Bergstra, R.J. Bril and C.A. Middelburg for their contributions, reviewing and discussions supporting the creation of this text.

Contents

Preface xiii

I Algebraic specification 1

1 Introducing the basic concepts **3**

1.1 Introduction 3
1.2 What is a(n algebraic) specification? 4
1.3 Names and signatures 8
1.4 Algebras 12
1.5 Flat algebraic specifications 15
1.6 Terms and assertions 16
1.7 Undefinedness and strictness 21
1.8 Example: specification of switches 23
1.9 Initial algebras 28
1.10 Example: specification of pairs of switches 28
1.11 Example: specification of natural numbers 29

2 Setting up algebraic specifications **33**

2.1 Introduction 33
2.2 Inductive predicate definitions 33
2.3 Horn clauses 36
2.4 Inductive function definitions 39
2.5 Proof obligations and applications 40
2.6 Consistency and categoricity 42
2.7 How to set up an algebraic specification 44
2.8 Example: specification of queues 45
2.9 Example: specification of stacks 50
2.10 Example: specification of bags 52
2.11 Example: specification of symbolic expressions 55

3 Structuring algebraic specifications **59**
3.1 Introduction 59
3.2 Flat schemes 61
3.3 Export schemes 62
3.4 Import schemes 64
3.5 Renaming schemes 66
3.6 Abbreviation schemes 69
3.7 Semantics of normal-form specifications 70
3.8 Hidden names 73

4 Implementing algebraic specifications **79**
4.1 Introduction 79
4.2 Expressions 80
4.3 Term interpretation of expressions 82
4.4 Declarations 83
4.5 Survey of assertions and expressions 85
4.6 Algorithmic predicate definitions 86
4.7 Algorithmic function definitions 87
4.8 From inductive to algorithmic definitions 88
4.9 Implementing an algebraic specification 93
4.10 Example: implementation of sets 94

II State-based specification **111**

5 From algebras to states **113**
5.1 Introduction 113
5.2 What is a state-based specification? 114
5.3 Procedure names and class signatures 116
5.4 States as algebras 118
5.5 Classes 120
5.6 Introducing variables 123
5.7 Procedure definitions 125
5.8 Comparison with imperative programs 127
5.9 From predicate logic to dynamic logic 129
5.10 Classes and specifications 138

6 Setting up state-based specifications **143**
6.1 Kinds of axioms 143
6.2 Example: specification of traffic lights 144
6.2.1 Properties of all states 145
6.2.2 Invariance properties 147
6.2.3 Properties of the initial state 149

此为试读,需要完整PDF请访问: www.ertongbook.com

6.2.4 Properties of state transitions 149
6.3 Example: specification of attributes 153
6.4 Example: specification of buffers 155
6.5 Example: specification of a display 160
6.6 How to set up an axiomatic state-based class description 169
6.7 Discussion 170

7 Structuring state-based specifications 171
7.1 Introduction 171
7.2 Example: specification of a database 173
7.2.1 Tuples and relations 173
7.2.2 Database schemas 175
7.2.3 The contents of a database 178
7.2.4 Tuple variables 180
7.2.5 Expressions and qualifications 183
7.2.6 Well-formedness 188
7.2.7 Semantics of queries 190
7.2.8 Example of an interactive session 194
7.3 Discussion 196

8 Implementing state-based specifications 199
8.1 Introduction 199
8.2 Statements 200
8.3 Algorithmic procedure definitions 204
8.4 Example: implementation of division 204
8.5 Towards an implementation strategy 207
8.6 The implementation strategy 210
8.7 Example: implementation of a line editor 213
8.7.1 Specifying the system 213
8.7.2 Documenting a building block 217
8.7.3 Choosing a representation 218
8.7.4 Adding display-oriented features 224
8.7.5 Implementing the display-oriented features 228
8.7.6 Translation to C 232
8.7.7 Executing the program 237
8.8 Discussion 238

III Advanced techniques **241**

9 Theoretical topics **243**
9.1 Introduction 243
9.2 Undefinedness revisited 243
9.3 Initial algebras 246
9.4 Horn clauses 252
9.5 Origin consistency 254
9.6 Comparing two types of models 259
9.7 The class concept revisited 260

10 Additional language constructs **263**
10.1 Introduction 263
10.2 Liberal scope rules 263
10.3 Free definitions 265
10.4 Parameterisation 268
10.5 Abstraction schemes 270
10.6 Application schemes 271
10.7 Extending the normalization procedure 272
10.8 More complex parameter restrictions 275
10.9 Object creation and procedures with results 276
10.10Variable sort definitions 278
10.11Dependent definitions 279
10.12Example: specification of instances 282
10.13Unifying expressions and statements 283

11 Towards large systems **287**
11.1 Introduction 287
11.2 Graphical representation of modules 288
11.3 Components and designs 291
11.4 Applications 299
11.5 Concluding remarks 301

Bibliography **303**

A Syntax **309**
A.1 General 309
A.2 Concrete syntax 309
A.3 Tokens 310
A.4 Keywords 310
A.5 Comments 311
A.6 Grammar 311
A.7 Operator priorities and associativities 314

A.7.1 Operators in renamings and signatures 315
A.7.2 Operators in assertions and expressions 316

B Standard library 317
B.1 Booleans 317
B.2 Natural numbers 319
B.3 Characters 321
B.4 Tuples 322
B.5 Finite sets 323
B.6 Finite bags 325
B.7 Finite sequences 327
B.8 Finite maps 329

Index 332

List of figures

Fig 1.1 Algebra with sort *Set1* and one object. 5
Fig 1.2 Algebra with two sorts. 6
Fig 1.3 Signature of the specification of integers. 11
Fig 1.4 Signature of the specification of stacks. 12
Fig 1.5 Signature Σ with two sort names `V` and `W`. 14
Fig 1.6 Algebra with signature Σ. 14
Fig 1.7 Algebra where $f^M(x)$ is not defined. 21
Fig 1.8 Algebra with empty sort *Switch*. 23
Fig 1.9 Algebra with sort *Switch* and one object. 25
Fig 1.10 Algebra with sort *Switch* and three objects. 26
Fig 1.11 Algebra with sort *Switch* and two objects. 27
Fig 1.12 Algebra with empty sort *Nat*. 31
Fig 1.13 Natural numbers as a model. 32
Fig 1.14 Non-standard model satisfying `NAT1`, `NAT2`, `NAT3` and `NAT4`. 32
Fig 2.1 Signature of the specification of queues. 45
Fig 2.2 Signature of the specification of stacks. 51
Fig 2.3 Signature of the specification of bags. 53
Fig 2.4 Signature of the specification of symbolic expressions. 55
Fig 3.1 Signature Σ_1 with sort names `A`, `B` and `C`. 71
Fig 3.2 Signature Σ_2 with sort names `A` and `B`. 71
Fig 3.3 Algebra with sorts A, B and C. 71
Fig 3.4 Algebra with sorts A and B. 72
Fig 3.5 Forgetting `C` and `g`. 72
Fig 3.6 Signature of the specification of sets. 75
Fig 4.1 Signature of the specification of lists. 94
Fig 4.2 Implementation of lists. 106
Fig 5.1 Class Signature with sorts, functions and a procedure. 117
Fig 5.2 Signature Σ with two sorts and two functions. 118
Fig 5.3 Σ-algebra with sorts `V` and `W`. 119
Fig 5.4 Class with five states. 120
Fig 5.5 Class signature with procedures `set` and `reset`. 122
Fig 5.6 Signature with one sort and one function. 122
Fig 5.7 Class with procedures `set` and `reset`. 122
Fig 5.8 Class with procedures `set` and `reset`. 123
Fig 5.9 Class with procedure `set` obtained by hiding. 140
Fig 6.1 Road crossing with traffic lights. 145
Fig 6.2 State transition. 150
Fig 6.3 State transition with intermediate state. 151
Fig 6.4 Buffer containing three items. 156
Fig 6.5 Display device with cursor and screen. 162

Fig 7.1 Template of relation with two fields. 176
Fig 7.2 Relation with two fields. 176
Fig 7.3 Outcome of first **retrieve** command. 194
Fig 7.4 Outcome of second **retrieve** command. 195
Fig 7.5 Outcome of third **retrieve** command. 195
Fig 7.6 Sharing of the state-based description `TYPING`. 197
Fig 8.1 Template of an axiomatic state-based specification. 211
Fig 8.2 Template of an algorithmic state-based implementation. 212
Fig 8.3 Buffer with gap. 219
Fig 8.4 Buffer representing string **"hello_world"**. 220
Fig 9.1 Signature Σ_{Scf} with sort, constant and unary function. 246
Fig 9.2 Auto-bisimulation $\underline{\leftrightarrow}$. 261
Fig 10.1 Class signature with one sort and one procedure. 277
Fig 10.2 Class with dynamic object creation. 278
Fig 11.1 Graphical representation of `CLASS` D_1 ... D_n `END`. 288
Fig 11.2 Nested import/export structure. 289
Fig 11.3 Overview of the editor (first part). 290
Fig 11.4 Overview of the editor (display-oriented features). 291

List of tables

Table 1.1 Survey of all terms. 17
Table 1.2 Survey of all assertions. 19
Table 3.1 Survey of all modular schemes. 60
Table 4.1 Survey of all expressions. 81
Table 4.2 Kinds of predicates. 82
Table 4.3 Survey of all assertions. 85
Table 4.4 Survey of all expressions. 85
Table 5.1 Survey of predicate and function definitions. 125
Table 5.2 Survey of procedure definitions. 127
Table 5.3 Survey of all assertions. 136
Table 5.4 Survey of all expressions. 137
Table 7.1 Bottom-up construction of a large state-based specification. 171
Table 8.1 Survey of all statements. 203
Table 8.2 `EDITOR_SPEC` as an instance of the general template. 217
Table 8.3 `EDITOR_IMPL` as an instance of the general template. 223
Table 10.1 Assertions and expressions with extended scope. 264
Table 10.2 Constructs where declarations in *A* extend to *B*. 264
Table 10.3 Constructs which are transparent for object names. 264
Table 10.4 Survey of all schemes. 270
Table 10.5 Survey of sort definitions. 278
Table 11.1 Survey of all components. 292
Table 11.2 Designs. 292
Table A.7.1 Operators in renamings and signatures. 315
Table A.7.2 Operators in assertions and expressions. 316

此为试读，需要完整PDF请访问：www.ertongbook.com