Contents

Preface **ix**

Acknowledgements **xii**

Glossary of Notations **xiii**

0 The Challenge of Unification **1**

0.1 Programming paradigms 3
0.2 Levels of abstraction 5
0.3 Varieties of presentation 6
0.4 Alphabets 8
0.5 Signatures 12
0.6 Laws 15
0.7 Challenges that remain 18

1 The Logic of Engineering Design **22**

1.1 Observations and alphabets 24
1.2 Behaviour and predicates 26
1.3 Conjunction 28
1.4 Specifications 31
1.5 Correctness 34
1.6 Abstraction 38
1.7 The ideal and the reality of engineering 41

2 Relations **44**

2.1 Conditional 47

2.2 Composition 48
2.3 Assignment 49
2.4 Non-determinism 51
2.5 The complete lattice of relations 54
2.6 Recursion 56
2.7* Strongest fixed point 61
2.8* Preconditions and postconditions 64
2.9 Variable declarations 68

3 Designs **74**
3.1 The refinement calculus 76
3.2 Healthiness conditions 82

4 Linking Theories **86**
4.1 Subset theories 88
4.2 Galois connections 97
4.3* Prespecification and postspecification 104
4.4* Simulation 109

5 The Algebra of Programs **113**
5.1 Assignment normal form 115
5.2 Non-determinism 118
5.3 Non-termination 119
5.4 Recursion 120
5.5 Iteration 125
5.6 Computability 127
5.7* Completeness 128
5.8* From algebraic to denotational semantics 130

6 Implementation **133**
6.1 Execution 134
6.2 Compilation 141
6.3 Interpretation 148
6.4* Jumps and labels 152

7 Concurrency **160**
7.1 Disjoint processes 162
7.2 Parallel by merge 168
7.3 The spreadsheet principle 175
7.4 Shared array 180
7.5 Synchronisation 182
7.6* Concurrent logic programming 186

8 Communication **194**
8.1 Algebra of Communicating Processes (ACP) 199
8.2 Communicating Sequential Processes (CSP) 207
8.3* Data flow 217

9 High Order Programming **232**
9.1 Procedures without parameters 233
9.2 Parameter mechanism 238
9.3 Functions 244
9.4* Declarative programming 249

10 Operational Semantics **258**
10.1 Derivation of the step relation 259
10.2 Bisimulation 262
10.3 From operations to algebra 269
10.4* From operational to denotational semantics 272
10.5* Operational semantics of CSP 273

Appendix 0: Alphabets **278**

Appendix 1: Shared Variables **279**

Appendix 2: Primitives **280**

Appendix 3: Healthiness Conditions **281**

Bibliography **283**

Index **295**

Preface

A theory of programming explores the principles that underlie the successful practice of software engineering. As in all branches of engineering, the practice comes first, both in importance and in historical order. The rapidly spreading benefits of computer application in modern society are largely due to the efforts and intuitive genius of teams of programmers, who have gained their skills and understanding the hard way, by long practice and experience. But now there is another complementary way for practising software engineers. A study of the relevant scientific theory can enhance their skills, broaden their range, deepen their understanding and strengthen their confidence in the accuracy and reliability of their designs and products. Understanding of a common theory enables experience gained in one language or application to be generalised rapidly to new applications and to new developments in technology or fashion; and it is the theory that maintains the intellectual interest of professional activity throughout a lifetime of achievement. But best of all, development of a comprehensive and comprehensible theory encapsulates the best of the state of the art in the subject, and makes it more readily available to the next generation of entrants to the profession.

The key to further progress is education; that is the goal of this book. It aims to attract and inform a class of student who are committed to practical engineering ideals, and who wish to devote their efforts to study of the relevant scientific foundation. They will probably have exposure already to one or more programming languages, and expect to meet more in their professional careers. An understanding of the basic concepts which underlie this variety will assist in mastery of new methods and notations, and in transfer of hard-won experience from one field of application to another. In a university curriculum, the book may find a place on a course entitled "principles of programming languages" or "programming language semantics". It would be a useful technical basis for a course on program verification, or on a software engineering course which takes seriously the normal engineering concerns of quality, reliability and safety. If such courses do not exist now, perhaps this book will eventually inspire the development of courses entirely devoted to the theory of programming. An introductory course can be based on the first six chapters or so; the remaining chapters lead to the edge of current research.

The book may also attract the interest of students and researchers in mathematics, who look to computing science as a source of new applications and examples, and as a source of new research problems of relevance to the modern world. Those who have already made a contribution to theoretical computing science may be inspired to contribute further towards the ideal of unification. It is certain that progress towards a deeper and broader understanding will depend on the commitment, cooperation, and further major discoveries by specialists in many diverse fields of research.

The book is wide ranging not only in its subject matter but also in its approach and style. The early chapters take a rather general philosophical approach, and the early sections of each chapter devote attention to general background and motivation. The definitions are accompanied by examples and the theorems by meticulous proof. The reader is strongly encouraged to skip the elements that are found uninteresting or incomprehensible; there is a strong possibility that further reading will solve both these problems. Sections marked with an asterisk may be profitably omitted on first reading. The following summary is a further guide to judicious skipping.

The first five chapters justify and introduce the main concepts and methods used throughout the rest of the book. Chapter 0 relates the goal of unification to the achievements of other branches of science and mathematics. It surveys the range of programming methods and languages, and the axes along which they are classified. It summarises the main methods and conclusions of the book, in a manner suitable for readers who enjoy prior acquaintance with programming semantics. It ends with a summary of major challenges which have been left for future research. Chapter 1 draws similar inspiration from a study of general methods of engineering design, which are found to follow exactly the principles of logical reasoning familiar in other mathematical and scientific disciplines. The chapter ends with a summary of the many personal and professional qualities required of the successful engineer, apart from an understanding of the relevant theory and its practice.

Both these chapters may be lightly skipped by a reader who wants to get on to the real substance, which begins in Chapter 2. This is a presentation within predicate calculus of Tarski's theory of relations, enriched with his fixed point theory, and applied to Dijkstra's simple non-deterministic sequential programming language. This language will prove adequate as a framework within which all more complex languages can be defined. But first, a small inconsistency between relation theory and practice must be resolved. This is done in Chapter 3, where *designs* are introduced as a subclass of relations which can be decomposed into the familiar precondition–postcondition pair of programming calculi like VDM. Chapter 4 develops some of the more elegant general results used in the rest of the book. Chapter 5 presents a complete algebra for the sequential programming language, and derives a normal form theorem.

The remaining chapters of the book introduce more advanced programming language features one by one. They may be studied in almost any combination and to any depth, with cross references followed at will. Chapter 6 deals with labels, jumps and machine code, and expounds the principles of correct compilation from a high level language. Chapter 7 introduces parallel processing based on the shared-store paradigm. It gives a common parameterised definition of parallel composition that is reused in the following chapters. Chapter 8 introduces reactive processes, which allow interaction and communication not only on termination of a program but also at intermediate stable states while it is running. Chapter 9 deals with programs that manipulate programs as data. It includes declarative programming, as incorporated in modern functional and logical programming languages. Chapter 10 unifies the general theory of programming with a popular mode of presentation in the form of an operational semantics.

Note: A prerequisite for private reading of the book is some acquaintance with propositional calculus, predicate notation and the concepts of discrete mathematics. In the interests of standardisation, and for the general benefit of software engineers, the mathematical notations are taken largely from the draft international standards for Z and VDM.

Tony Hoare
He Jifeng

Oxford, July 1997

Acknowledgements

The ideas put forward in this book have been inspired or derived from the work of many earlier researchers. The references are a partial acknowledgement of the work that has been most influential on the thinking of the authors; they are not necessarily the earliest source of the ideas.

The following have performed a sterling service to the reader by comments on earlier drafts of this work: R. Backhouse, J. Baeten, J.W. de Bakker, J.A. Bergstra, O.-J. Dahl, E.W. Dijkstra, P. Gardiner, M. Hennessy, R. Joshi, B. von Karger, H. Langmaack, G. Lowe, G. McCusker, A.J.R.G. Milner, J. Misra, C.C. Morgan, O. Owe, F. Page, J. Parrow, G.D. Plotkin, A. Ravn, W.P. de Roever, A.W. Roscoe, A. Sampaio, S.A. Schneider, D.S. Scott, M. Sintzoff, A. Stevens, F. Vaandrager.

The research reported in this book was supported by EPSRC research grant GR/K58708 "Linking theories for computing science", the ESPRIT Basic Research Actions 3104 and 7071 ProCoS and 3006 CONCUR, and the Admiral B.R. Inman Centennial Chair in Computing Theory at the University of Texas at Austin, and the Newton Institute Seminar on the Semantics of Programming Languages, and the James Martin Chair of Computing at the University of Oxford.

Glossary of Notations

Cross references

Section 3.2	Chapter 3 Section 2
3.2**L1**	Law 1 in Section 3.2
Theorem 3.2.3	Theorem 3 in Section 3.2
Lemma 3.2.7	Lemma 7 in Section 3.2
Example 3.2.5	Example 5 in Section 3.2
Table 3.2.4	Table 4 in Section 3.2
Diagram 3.2.8	Diagram 8 in Section 3.1
Exercise 3.2.10	Exercise 10 in Section 3.1
□	end of an example or proof

Logic

$=$	equals
$\neq$	is distinct from
$=_{df}$	is defined by
iff	if and only if
$[P]$	P is true for all values of variables in its alphabet
$P \wedge Q$	P and Q (both true)
$P \vee Q$	P or Q (one or both true)
$\neg P$	not P (P is not true)
$P \Rightarrow Q$	if P then Q

$P \equiv Q$	P if and only if Q
$\exists x : T \bullet P$	there exists an x in set T such that P
$\forall x : T \bullet P$	for all x in set T, P
$P \vdash Q$	Q can be validly deduced from P
$P \dashv\vdash Q$	$P \vdash Q$ and $Q \vdash P$

Sets

$\in$	is a member of
$\notin$	is not a member of
$\{\}$	the empty set
$\{a\}$	the singleton set containing only a
$\{x : T \mid P(x)\}$	the set of all x in T such that $P(x)$
$\{f(x) \mid P(x)\}$	the set of all the values $f(x)$ such that $P(x)$
$S \cup T$	S union T
$S \cap T$	S intersect T
$S \setminus T$	S minus T
$S \subseteq T$	S is contained in T
$S \supseteq T$	S contains T
$\mathcal{N}$	the set of natural numbers
$\bigcup C$	union of collection C of sets
$\bigcap C$	intersection of collection C of sets

Sequences

$<>$	the empty sequence
$<a>$	the sequence containing only a
$s\frown t$	catenation of sequences s and t
s_0	the head of sequence s
$tail(s)$	the tail of sequence s
$\#s$	the length of s
A^*	set of all sequences with elements from set A
$s \downarrow E$	the subsequence of s omitting elements outside E
$s \vert\vert\vert t$	the set of all interleavings of sequences s and t

Functions

id_S	the identity function on set S
$\{i \mapsto e\}$	the singleton function that maps i to e
$F : S \to T$	F is a total mapping from S to T
$domain(F)$	the domain of function F
$image(F)$	the image of function F
$X \lhd F$	function F with domain restricted to set X
$F(x)$	the member of T to which F maps x
F^{-1}	inverse of F
$F \circ G$	F composed with G, mapping x to $F(G(x))$
$\lambda x : S \bullet F(x)$	the function that maps each x in S to $F(x)$
$F \oplus G$	the function $\lambda x \bullet (G(x)$ if $x \in domain(G)$ else $F(x))$
or_P	the function $\lambda X \bullet (P \vee X)$
and_Q	the function $\lambda X \bullet (Q \wedge X)$
pre_J	the function $\lambda X \bullet (J; X)$
$post_K$	the function $\lambda X \bullet (X; K)$
imp_P	the function $\lambda X \bullet (P \Rightarrow X)$

Alphabets

1.1	αP	the alphabet of relation P
1.3	$in\alpha P$	the input alphabet of relation P
1.3	$out\alpha P$	the output alphabet of relation P
6.1	$\alpha l P$	the set of continuations of program P
6.4	$\alpha l_0 P$	the set of entry points of labelled program P
6.4	$\alpha l' P$	the set of exit points of labelled program P
8.0	$\mathcal{A}P$	the set of all actions possible for process P
8.3	$inchan\, P$	the set of input channels of process P
8.3	$outchan\, P$	the set of output channels of process P

Observations

1.1	x	the initial value of variable x
1.1	x'	the final value of variable x

3.0	ok	the program has started
3.0	ok'	the program has reached a stable state
6.1	l	control variable of the execution mechanism
7.6	q	the initial observation of a logic program (called a question)
7.6	a'	the resulting sequence of answers
8.0	tr	an arbitrary trace of the specified process
8.0	ref	an arbitrary refusal of the specified process
8.0	$wait$	the specified process is in an intermediate observable state

Relations

1.5	$[P \Rightarrow Q]$	P implies Q (everywhere)
2.4	**true**	the universal relation
2.5	$P \sqcup Q$	intersection of P and Q
2.5	$\top$	miracle (the top of the lattice)
2.5	**false**	the empty relation
2.6	$\mu X_{\mathbf{S}} \bullet P(X)$	the weakest solution in **S** of $X = P(X)$
2.7	$\nu X_{\mathbf{S}} \bullet P(X)$	the strongest solution in **S** of $X = P(X)$
2.8	$p\{Q\}r$	Hoare triple: on precondition p, execution of Q ensures postcondition r
2.8	$b_\perp$	assertion b
2.8	$b^\top$	assumption b
2.9	**var** x	declaration of variable x
2.9	**end** x	termination of the scope of variable x
2.9	P_{+x}	P with its alphabet augmented by x and x'
3.1	$P \vdash Q$	the relation $ok \wedge P \Rightarrow ok' \wedge Q$
3.1	$\mathcal{D}e$	expression e is defined
4.0	$P \sqsupseteq Q$	P is a refinement of Q, i.e. $[P \Rightarrow Q]$
4.3	P/Q	the weakest prespecification of Q through P
4.3	$Q \backslash P$	the weakest postspecification of Q through P
10.1	$\rightarrow$	step relation on machine states
10.2	$\sim_s$	strong bisimulation
10.2	$\approx_w$	weak bisimulation

10.3	$\overset{*}{\rightarrow}$	the reflexive transitive closure of $\rightarrow$
10.3	$(\mathtt{s}, \mathtt{P})\uparrow$	$(\mathtt{s}, \mathtt{P})$ is a divergent machine state
10.3	$(\mathtt{s}, \mathtt{P}) \sqsubseteq (\mathtt{t}, \mathtt{Q})$	$(\mathtt{t}, \mathtt{Q})$ refines $(\mathtt{s}, \mathtt{P})$
10.3	$\mathtt{P} \sim \mathtt{Q}$	$\mathtt{P}$ simulates $\mathtt{Q}$ and vice versa
10.4	$\overset{a}{\rightarrow}$	step accompanied by action a

Sequential programming language

2.1	$P \lhd b \rhd Q$	P if b else Q
2.2	$P; Q$	P then Q
2.3	$x := e$	assign value of e to variable x
2.3	$I\!I$	skip (do nothing, but terminate)
2.4	$P \sqcap Q$	non-deterministic choice of P and Q
2.4	$\perp$	abort

Labelled programming language

6.1	P^*	repeated execution of the step relation P
6.1	$P \Vert Q$	assembly of step relations P and Q
6.2	$\langle s, P, f \rangle$	the target code: $\mathbf{var}\, l; (l = s)^{\top}; P^*; (l = f)_{\perp}; \mathbf{end}\, l$
6.3	$\mathtt{P}$	text of program P
6.4	$P : S \Rightarrow F$	P with S as its entries and F as its exits
6.4	$P^{\backslash H}$	P without the exit labels in H
6.4	${}^{H/}P$	P without the entry labels in H

Concurrent programming language

7.1	$l : P(x)$	the relation $P(l.x)$, with observations labelled by l
7.1	$P \| Q$	disjoint parallel composition of P and Q
7.2	$\|_M$	parallel composition with the merge operation M

Logic programming language

7.6	**no**	always gives an empty sequence of answers
7.6	**yes**	the program which copies the question as its answer
7.6	$K \| \| L$	interleaves the answers produced by K and L

7.6	K **or** L	catenates the answers produced by K and L
7.6	L^*	apply L to each of the questions and catenate all the answers
7.6	K **and** L	feed the answers produced by K to L^*
7.6	!	cut : take the first answer
7.6	$\neg L$	**yes** if L has no answer, else **no**

ACP (Algebra of Communicating Processes)

8.1	δ	the deadlocked process
8.1	$\mathbf{do}_A(a)$	do a then terminate (abbreviated to a)
8.1	$a; P$	do a then P
8.1	$P + Q$	P choice Q
8.1	$\Sigma_{i \in I} P_i$	choice from the family $\{P_i \mid i \in I\}$
8.1	$P \vert\vert\vert Q$	P interleave Q
8.1	$P \vert\vert_{ACP} Q$	P in parallel with Q, with selective synchronisation
8.1	$\varrho_E P$	process P without events in E (encapsulation)

CSP (Communicating Sequential Processes)

8.2	$SKIP$	the process which does nothing but terminate successfully
8.2	$STOP$	the deadlocked process
8.2	$CHAOS$	the worst process, whose behaviour is unpredictable
8.2	$a \rightarrow P$	a then P
8.2	$P \| Q$	P choice Q
8.2	$P \vert\vert\vert Q$	P interleave Q
8.2	$P \vert\vert_{CSP} Q$	P in parallel with Q, with synchronisation of identical actions
8.2	$P \backslash E$	P with events in E hidden
8.2	$P \gg Q$	P chained to Q
8.2	$P \oslash Q$	the composite choice $(P \| Q) \sqcap Q$

Data flow

8.3	$c.m$	communication of value m on channel c
8.3	$\mathcal{A}_c(P)$	the set of messages which P can communicate on c
8.3	$[P]$	P with each of its channels buffered

8.3	$a?x \leadsto P$	wait for an input from channel a then P
8.3	$c!e \leadsto P$	output e on channel c then P
8.3	$P/a.m$	P after input of m from channel a
8.3	$P\|_{DF}Q$	parallel composition of data flow processes P and Q

High order programming

9.1	$\{[P]\}$	predicate P named as a constant
9.3	$\tau\ res \bullet P$	the *res* such that P (unique description)
9.4	$P \overset{\bullet}{\|} Q$	P if it does not stop, else Q (priority choice)

Precedence

arithmetic operators	bind tightest
;	
$\lhd b \rhd$	
$\wedge,\ \vee,\ \sqcap,\ \sqcup$	
$\cap,\ \cup$	
$\Rightarrow,\ \equiv$	bind loosest