
Fundamental Structures of Computer Science

SED

Fundamental Structures of Computer Science

William A. Wulf
Mary Shaw
Paul N. Hilfinger

Computer Science Department
Carnegie-Mellon University

Lawrence Flon

Computer Science Department
University of Southern California



ADDISON-WESLEY PUBLISHING COMPANY

Reading, Massachusetts • Menlo Park, California

London • Amsterdam • Don Mills, Ontario • Sydney

Reproduced from camera-ready copy provided by the authors. The copy was prepared at Carnegie-Mellon University on a photocomposer operated by the SCRIBE text editing system.

Library of Congress Cataloging in Publication Data

Main entry under title:

Fundamental structures of computer science.

Includes bibliographical references and index.

1. Electronic digital computers—Programming.
2. Data structures (Computer science) I. Wulf, William Allan.

QA76.7.F86 001.6'42 79-12374
ISBN 0-201-08725-1

Copyright © 1981 by Addison-Wesley Publishing Company, Inc. Philippines copyright 1981 by Addison-Wesley Publishing Company, Inc.

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 the prior written permission of the publisher. Printed in the United States of America. Published simultaneously in Canada. Library of Congress Catalog Card No. 79-12374

ISBN 0-201-08725-1
CDEFGHIJK-DO-898765432

Preface

This textbook is designed to support an intermediate-level course in computer science. It is intended for students who have at least one or two semesters of programming and problem-solving experience and who wish to prepare for more advanced topics in computing. Ideally, the students will also have a semester's background in discrete mathematics.

The content and organization of the textbook was motivated by a single premise—that *programming is an engineering discipline*. Like other engineering disciplines, programming is concerned with building things, particularly *real* things of enormous practical importance. And, like other engineering, *good* programming is rooted in the application of science. Understanding that science is essential to being a good programmer.

This textbook is also an introduction to the science that underlies good programming. Many of the concepts involved are mathematical. Software concepts such as program organization, data structures, performance analysis, and so on, have traditionally been taught only as programming techniques. In this textbook we support these programming techniques by relating programming ideas to the underlying mathematical concepts of automata, formal languages, data types, and so on. Through examples we illustrate the relation and utility of the mathematical concepts to practical contexts.

Much of the material presented in this textbook has traditionally been taught (in greater depth) in advanced undergraduate and graduate courses. We believe, however, that it is both necessary and appropriate to introduce this material at an early stage. Just as introductory calculus and physics are

prerequisite to most engineering courses, the material presented here forms an appropriate background for courses in software engineering and advanced computer science. For the organization of this material, see the matrix on page 8 and the discussion immediately preceding it.

Using the Book

We originally designed this textbook for a second course in computer science, and since 1974 have used various generations of it for a one-semester sophomore course. One semester is not enough to cover all the material now contained in the textbook, but in a semester we can generally cover most of Parts One and Two and the theoretical material in Part Three. We also cover several of the example chapters in Part Four. Homework usually involves substantial programming exercises. Although we are aware that this is a large amount of material to cover in one semester, we have *not* observed that our students have any great difficulty with the *level* of the material.

Although we strongly believe that the material in the textbook is best presented early, that is in a second course in computer science, there are other ways that the book can be used. For example,

- As the basis of a two-semester course: This textbook contains ample material for a full-year course. In such a course, however, we would not only cover this material, but also emphasize advanced programming techniques and more advanced data structure representations.
- Merged with a software engineering course: Most of the material in this text is among the formal prerequisites for a rigorous treatment of software engineering. Half of a third or fourth year course in software engineering might profitably be spent on the chapters about formal models, specification, verification, and performance analysis.
- Incorporated into a theoretical computer science course: Much of the material in this course is predicated on a knowledge of discrete mathematics and formal logic and therefore constitutes a meaningful application of these formal mathematical ideas.
- As in-house training material for practicing programmers: Most of today's practicing programmers were educated when computing was little more than a collection of tricks. That's no longer the case. In order to compete with the recent graduates, the experienced professional will need to master the material we cover—especially the more formal material.

The language used throughout this book is essentially PASCAL. The material we cover, however, applies to programming in general, rather than to any particular language, and so we have felt free to deviate from PASCAL from time to time in presenting some material. Appendix C summarizes the major deviations. The programs in the example chapters have all been executed with a PASCAL compiler for the DEC PDP-10 originally developed at the University of Hamburg, Germany. We have, however, tried to avoid including in our examples any statements peculiar to that compiler.

Computer Science has made enormous advances in the past 10 to 15 years. It is no longer simply an art or an ad hoc collection of programming tricks. There is now a genuine body of science and mathematics that the programmer of the future must know and use. This book is hardly the last word that will be written on the subject, but it is a good beginning for new students and a sourcebook for the experienced professional.

Acknowledgments

The authors gratefully acknowledge the support and assistance of

- All those students over the past six years who have suffered through drafts of our material,
- Anita Andler, Sharon Carmack, and Dorothy Josephson, who (re)*typed much of the material, adapting to a series of dynamically developing computerized document production systems,
- The graduate students who helped teach sections of the course and who contributed material and comments: Sten Andler, Lee Coopridner, David Notkin, Gary Feldman, Lanny Forgy, Brian Reid, and Elaine Rich.
- Brian Reid, who developed and maintains SCRIBE, the document production system that allowed the manuscript of this textbook to be almost automatically produced,
- Bob Morgan, who gave some of us strong personal support,
- Jon Bentley, whose experiences with using the textbook in an industrial setting gave us new insights into ways to teach the material,
- A host of readers and reviewers, both known to us and anonymous, whose comments were invaluable, notably Terry Baker, Jon Bentley, Ron Brender, Mike Horowitz, David Jefferson, John Kender, David Lamb, David Levine, Tony Ralston, Brian Reid, Peggy Ross, Len Shustek, John Sowa, Bob Sproull, and Judy Zinnikas,

- The National Science Foundation and the Defense Advanced Research Projects Agency, which provided support for the research environment in which we learned much of what we teach here.

Pittsburgh, Pennsylvania
April 1980

W.A.W.
M.S.
P.N.H.
L.F.

Contents

Part One/Fundamental Structures of Control	9
CHAPTER 1 FINITE STATE MODELS	11
1.1 Finite State Machines	11
State Transition Functions	14
State Transition Diagrams	16
Special FSM's: Recognizers and Generators; Null Transitions	17
Bigger Inputs and Outputs	19
1.2 Nondeterministic FSM's	20
1.3 Languages and Regular Expressions	25
Languages	25
Regular Expressions	26
1.4 Equivalence of Regular Expressions and FSM's	29
Example: Construction of NDFSR from RE	32
1.5 Characterizing the "Power" of Computational Models	33
A Limitation of FSM's	34
1.6 Further Reading	36
CHAPTER 2 MORE MODELS OF CONTROL: FLOWCHARTS AND PROGRAMS	49

2.1	Flowcharts	49
	The Relation Between FSM's, RE's, and Flowchart Programs	53
2.2	Simple Programming Language Control	55
	FCL/1	55
	FCL/2	56
2.3	Equivalence of Flowcharts and Programs	56
2.4	Further Reading	59
CHAPTER 3	ADDITIONAL CONTROL STRUCTURES	65
3.1	Selection Constructs	66
	The "case" Statement	66
	The "numberedcase" Statement	67
3.2	Iterative Constructs	68
	The "repeat" and "loop" Statements	68
	The "for" Statement	69
3.3	Routines	71
	Routine Invocation	72
	Parameters	73
	Value-returning and Value-less Routines	74
	Recursive Routines	75
	The Power of Routines	77
3.4	Summary: Equivalence, Power, and Convenience	77
3.5	Further Reading	78
CHAPTER 4	THE REPRESENTATION OF CONTROL	85
4.1	Representation of FCL/2 Constructs	86
	The Representation of Sequential Control	87
	The Representation of "if-then-else"	87
	The Representation of "while"	87
4.2	Representation of Other Control Constructs	88
	Representation of "repeat"	88
	Representation of "numberedcase"	88
4.3	Representation of Routines	90
4.4	Further Reading	92
CHAPTER 5	FORMAL SPECIFICATION AND PROOF OF PROGRAMS	101
5.1	Specification of Programs	102
5.2	Program Proofs	105

5.3	Computing wp: Language Semantics	106
	The Null Statement	108
	Sequencing	108
	The “if” Statement	108
	The Assignment Statement	109
	Two Examples	111
	Embedded Assertions	112
	Weakest Precondition for the “while” Statement	112
	The “for” Statement	120
	The Procedure Call	123
	An Example with Procedures	125
	Dealing with Parameters	126
	Summary of Weakest Preconditions	127
5.4	Weaknesses in Correctness Arguments	129
	Errors in Specification	129
	Errors in Proofs	130
5.5	Validation: Testing Versus Verification	130
5.6	Further Reading	131

CHAPTER 6 DETERMINING EFFICIENCY OF COMPUTATIONS 145

6.1	Two Easy Examples	146
	Polynomial Evaluation	146
	Series Evaluation	148
	Improvements Come in Different Flavors	149
6.2	Order Arithmetic	150
6.3	Gathering and Using Information about Efficiency	153
6.4	Experimental Determination of Performance	154
	Counting Operations	155
	Measuring Time	156
6.5	Analytic Determination of Performance	157
	Verifying Statements about Operation Counts	158
	A Formal Technique for the Analysis of Execution Time	160
	Sample Values for Constants	164
	Examples Revisited	166
	Another Example	168
6.6	Further Reading	172

Part Two/Fundamental Structures of Data 179

CHAPTER 7 MATHEMATICAL MODELS OF DATA 181

7.1	Introduction	181
7.2	Memory, Variables, Names, and Values	182
	A Functional Description of Memory	182
	An Axiomatic Description of Memory	184
	Relating the Functional and Axiomatic Descriptions	186
7.3	Type	187
7.4	Describing Types	189
	Syntactic Specification	190
	Semantic Specification	191
7.5	Further Reading	194
CHAPTER 8	DATA IN PROGRAMMING LANGUAGES	203
8.1	Scalar Types	203
	Enumerated Types	204
8.2	Names and References	206
	Use of References in Assignments	209
	Use of References for Parameter Passing	211
	Explicit Reference Types	212
8.3	Structured Types	212
	Arrays	213
	Records	216
8.4	Other Data-Related Issues that Arise in Programming Languages	217
	Unions	217
	Conversions Between Types: Coercions	219
	Variable Declarations	219
	Storage Management and Allocation	223
8.5	Further Reading	223
CHAPTER 9	NONELEMENTARY DATA STRUCTURES	233
9.1	Abstract Structured Types: Linear Structures	233
	Deque	234
	Queues	236
	Stacks	237
	Linear Lists	237
9.2	Abstract Structured Types: Nonlinear Structures	245
	Labeled Graphs	245
	Trees	248
	Finite Homogenous Sets	250
9.3	Further Reading	251
CHAPTER 10	THE REPRESENTATION OF DATA	263

10.1	Representational Techniques for Data	264
	Encoding	265
	Packing	265
	Address Arithmetic	266
	Linking	268
	Concluding Remarks on Representational Techniques	274
10.2	Representation of Stacks	274
10.3	Representation of Deques and Queues	275
	Vector Representation: Circular Buffering	275
	Linked Implementation	279
10.4	Representation of Trees and Graphs	280
	Linked Unlabeled Graph Representations	280
	Linked Tree Representation	283
	Vector Implementation of Binary Trees	283
	Array Representation of Unlabeled Graphs	284
10.5	Representation of Sets	285
	Bit-Vector Representation of Sets	285
	Vector-Pointer Representation of Sets	287
	Linked Representation of Sets	288
10.6	The Representation of Multidimensional Arrays	288
	More on Vector Representation: Descriptors	290
	Address Computations for Multidimensional Arrays	292
	A Slight Variation: Triangular Arrays	293
	Representation of Slices	294
	Illiffe Vectors	295
	Linked Representation of Sparse Arrays	296
10.7	Further Reading	297
CHAPTER 11	CORRECTNESS OF DATA REPRESENTATIONS	315
11.1	An Example: Proof of Type Stack	316
	What We Must Prove	317
	Adequacy of the Representation	319
	Correctness of the Operations	320
11.2	The General Approach	322
11.3	Weakest Preconditions for Dereferenced Assignments	324
11.4	Further Reading	326
CHAPTER 12	SPACE REQUIREMENTS	329
12.1	Static Space Requirements	330
12.2	Dynamic Space Requirements	333
	Space Requirements in Block-Structured Languages	334
	Dynamic Data Structure Allocation	334

12.3	Design Decisions and Representation Trade-Offs	336
12.4	Further Reading	337
	Part Three/The Interaction of Control and Data	339
CHAPTER 13	MODELS OF COMPUTATION AND GRAMMARS	341
13.1	Pushdown Automata	342
	DPDA's and NDPDA's	343
	Limitations of PDA's	348
13.2	Turing Machines	349
	Definition of a Turing machine	350
	Interpreters and Universal Turing Machines	352
	Computability	352
13.3	Metalanguages and Productions	354
	Backus-Naur Form (BNF)	356
	Classes of Languages	357
	BNF Description of Type 3 Languages	359
	Embedding	361
	Parse Trees	362
	Ambiguity	364
13.4	Further Reading	364
CHAPTER 14	RECURSION AND RELATED TOPICS	375
14.1	Dynamic Data Types	376
14.2	Recursion	376
14.3	The Power of Recursion	379
14.4	Divide-and-Conquer	381
14.5	Recursive Operations on Dynamic Types: Tree Walking	384
	Alternative Orders and Some Notation	385
	Search Trees	387
	Other Iterative Tree Walks	389
14.6	Further Reading	390
CHAPTER 15	THE INTERPRETATION OF IDENTIFIERS	397
15.1	Scope of Identifiers, Revisited	397
15.2	Extent of Storage, Revisited	401
15.3	Binding Identifiers to Values	403
	Call-by-Value	405
	Call-by-Address	406
	Call-by-Value/Result	406

	Call-by-Result	406
	Call-by-Name	407
15.4	Further Reading	407
CHAPTER 16	RUNTIME REPRESENTATION OF HIGH-LEVEL LANGUAGES	415
16.1	Languages with Static Storage Management	415
16.2	"Stack-Based" Languages	417
16.3	Up-level Addressing and the Display	418
16.4	Array Allocation	423
16.5	Block Structure	423
16.6	Procedure Parameters	424
16.7	Parameter Mechanisms	425
	Call-by-Value	425
	Call-by-Address	426
	Call-by-Value/Result	426
	Call-by-Result	426
	Call-by-Name	426
16.8	Heap Allocation	427
16.9	Further Reading	428
CHAPTER 17	REASONING ABOUT RECURSION	433
17.1	Proving Recursive Programs	433
17.2	Structural Induction	435
17.3	Further Reading	436
CHAPTER 18	ANALYSIS OF RECURSIVE ALGORITHMS	439
18.1	Examples of Cost Functions for Recursive Programs	440
18.2	Familiar Recurrence Relations of Mathematics	441
18.3	Solutions to Example Cost Functions	441
18.4	A General Rule for Solving Divide-and-Conquer Recurrences	443
18.5	Further Reading	443
	Part Four/Case Studies	447
CHAPTER 19	USING FSM'S IN PROGRAMMING: AN EXAMPLE	449
19.1	Background	449
19.2	The Problem	450

19.3	A Solution Based on FSM's	451
	A Generalization of FSM's	452
	Organizing the Lexical Analyzer	453
19.4	Abstract Program for the Solution	457
	An FSM Interpreter	459
	Tables to Drive the FSM Interpreter	460
19.5	Concrete Program for the Solution	462
	Declarations for Concrete Solution	462
	Main Program	463
	Implementation of Action Primitives	464
	Input Format for Tables	464
	Initializing the FSM	466
	Procedures for Input and File Manipulation	467
	Sample Run	469
19.6	Correctness of the Lexical Analyzer	469
	Verifying Properties of Individual Components	469
	Verifying Properties of the FSM Interpreter	471
	Verifying Properties of the Entire System	474
19.7	Performance of the Lexical Analyzer	475
	Cost of Initialization	476
	Cost of Simulation	477
	Total Cost	479
CHAPTER 20	FAST IMPLEMENTATION OF SETS: AN EXAMPLE	483
20.1	Background	483
20.2	The Problem	484
	Operations on the Set Type	484
20.3	An Implementation Using Lists and Trees	486
	Representing Sets with Trees	486
	The Problem of Deleting Elements	488
	Representing Sets with Lists	488
	A Combined Representation	488
20.4	Using the SetofElts Type	490
20.5	Programs for the Solution	491
	The Data Structure	491
	<i>FindElr</i> : An internal routine	492
	The Operation <i>Clear</i>	495
	The Operation <i>Remove</i>	495
	The Operation <i>Insert</i>	496
	The Operation <i>IsEmpty</i>	498
	The Operation <i>IsSubset</i>	499
	The Operation <i>Union</i>	499
	The Operation <i>Intersect</i>	500
	The Operation <i>Compact</i>	500
	The Operation <i>Copy</i>	501

20.6	Correctness of the <i>SetofEls</i> Type	501
	Representation Mapping	502
	Invariants	503
	Initialization	503
	Correctness of Operations	504
20.7	Performance of the <i>SetofEls</i> Type	505
	Speed of the <i>SetofEls</i> Type	506
	Space of the <i>SetofEls</i> Type	508
20.8	Choosing a Representation	509
	Differences and Constraints	509
	Time Comparison for Set Implementations	510
	Space Comparison for Set Implementations	511
	Criteria for Choosing an Implementation	511
CHAPTER 21	GENERATORS: WRITING LOOPS THAT OPERATE ON SETS	519
21.1	Loops that Operate on Set Elements	519
	Specifications for Generator Type	520
21.2	A Solution for the Implementation with Threaded Trees	521
21.3	Programs for the Solution	523
	The Data Structure	523
	Loop Control Operations	523
21.4	Reimplementation of Set operations	525
	The Operation <i>IsSubset</i>	525
	The Operation <i>Union</i>	525
CHAPTER 22	FORMULA MANIPULATION: AN EXERCISE IN DEFINING A DATA TYPE	529
22.1	Some Definitions	529
22.2	The Abstract Type <i>Expression</i>	532
22.3	Solutions to the Original Problems	534
	Reading Expressions	534
	Printing Expressions	535
	Differentiating Expressions	536
	Examples	537
22.4	Implementation of the Expression Data Type	538
22.5	A Performance Problem and its Solution	539
22.6	Summary	543
CHAPTER 23	PRODUCTION SYSTEMS AND SIMPLIFICATION	549
23.1	Production Systems	550
	Patterns	551
	Replacements	552
	Ordering of Productions	553

23.2	Extending the Type <i>Expression</i> with Patterns and Replacements	555
	Extensions to the Specifications of <i>Expression</i>	555
	The Routine <i>ApplyProductions</i>	556
	Implementing the Extended <i>Expression</i> Type	558
23.3	A Simplification Routine	559
23.4	Summary	562
	REFERENCES	565
APPENDIX A	GLOSSARY AND NOTATION	571
A.1	Standard Mathematical and Logical Notation	571
A.2	Standard Definitions from Discrete Mathematics	572
A.3	Notation Introduced in the Text	573
APPENDIX B	STANDARDS FOR EVALUATING PROGRAMS	577
B.1	Introduction	577
B.2	Programs Are Read by People	578
B.3	What Is Involved in Presenting a Program Well?	578
	Program Organization	579
	Comments and Other Documentation	580
	Programming Style	581
	Efficiency and Clarity	582
	Making Reasonable Decisions about Peripheral Matters	582
	Selection of Test Data	583
APPENDIX C	THE EXAMPLE PROGRAMMING LANGUAGE	585
APPENDIX D	A SIMPLE COMPUTER	587
D.1	The Memory Unit	588
D.2	The Input-Output Unit	589
D.3	The Central Processor	589
D.4	An Instruction Set	590
APPENDIX E	COLLECTED PROGRAMS FROM PART FOUR	593
E.1	Lexical Analyzer	593
E.2	The Types SetofElts and SetGen	597
E.3	Differentiator and Simplifier	602
	INDEX	609