Contents

Preface ix
Acknowledgments xv
Credits xvii
Tables of judgment forms and important functions xviii
Symbols and notation xx

Introduction 1

Part I. Foundations

1 An imperative core 11
- 1.1 Looking at languages 13
- 1.2 The Impcore language 17
- 1.3 Abstract syntax 27
- 1.4 Environments and the meanings of names 28
- 1.5 Operational semantics 29
- 1.6 The interpreter 38
- 1.7 Operational semantics revisited: Proofs 55
- 1.8 Extending Impcore 66
- 1.9 Summary 67
- 1.10 Exercises 72

2 Scheme, S-expressions, and first-class functions 89
- 2.1 Overview of μScheme... 90
- 2.2 Language I: Values, syntax, and initial basis 91
- 2.3 Practice I: Recursive functions on lists of values 98
- 2.4 Records and trees (more data) 107
- 2.5 Combining theory and practice: Algebraic laws 110
- 2.6 Language II: Local variables and `let` 117
- 2.7 Language III: First-class functions, `lambda`, and locations 120
- 2.8 Practice III: Higher-order functions on lists 127
- 2.9 Practice IV: Higher-order functions for polymorphism 131
- 2.10 Practice V: Continuation-passing style 136
- 2.11 Operational semantics 144
- 2.12 The interpreter 152
- 2.13 Extending μScheme with syntactic sugar 162
- 2.14 Scheme as it really is 168
- 2.15 Summary 172
- 2.16 Exercises 176

3 Control operators and a small-step semantics: μScheme+ 201
- 3.1 The μScheme+ language 202
- 3.2 Procedural programming with control operators 205
- 3.3 Operational semantics: Evaluation using a stack 210
- 3.4 Operational semantics: Lowering to a core language 213
- 3.5 A semantics of Core μScheme+ 215
- 3.6 The interpreter 223
- 3.7 Stacks, control, and semantics as they really are 239
- 3.8 Summary 243
- 3.9 Exercises 248

4 Automatic memory management 257
- 4.1 What garbage is and where it comes from 258
- 4.2 Garbage-collection basics 259
- 4.3 The managed heap in μScheme+ 263
- 4.4 Mark-and-sweep collection 266
- 4.5 Copying collection 271
- 4.6 Debugging a collector 280
- 4.7 Mark-compact collection 283
- 4.8 Reference counting 283
- 4.9 Garbage collection as it really is 285
- 4.10 Summary 287
- 4.11 Exercises 292

5 Interlude: μScheme in ML 301
- 5.1 Names and environments... 303
- 5.2 Abstract syntax and values 306
- 5.3 Evaluation 309
- 5.4 Defining and embedding primitives 312
- 5.5 Notable differences between ML and C 314
- 5.6 Free and bound variables... 315
- 5.7 Summary 317
- 5.8 Exercises 320

6 TYPE SYSTEMS FOR IMPCORE AND μSCHEME 327
6.1 Typed Impcore: A statically typed imperative core 329
6.2 A type-checking interpreter for Typed Impcore 337
6.3 Extending Typed Impcore with arrays 343
6.4 Common type constructors 348
6.5 Type soundness 350
6.6 Polymorphic type systems and Typed μScheme 351
6.7 Type systems as they really are 383
6.8 Summary 383
6.9 Exercises 386

7 ML AND TYPE INFERENCE 401
7.1 Nano-ML: A nearly applicative language 402
7.2 Abstract syntax and values of nano-ML 404
7.3 Operational semantics 405
7.4 Type system for nano-ML 407
7.5 From typing rules to type inference 417
7.6 The interpreter 433
7.7 Hindley-Milner as it really is 441
7.8 Summary 441
7.9 Exercises 443

PART II. PROGRAMMING AT SCALE

8 USER-DEFINED, ALGEBRAIC TYPES (AND PATTERN MATCHING) 457
8.1 Case expressions and pattern matching 459
8.2 Algebraic data types in μML 466
8.3 Equational reasoning with case expressions 476
8.4 Syntactic sugar: Patterns everywhere 480
8.5 Type generativity and type equivalence 483
8.6 Abstract syntax and values... 485
8.7 Theory and implementation of user-defined types 486
8.8 Theory and implementation of case expressions 490
8.9 Algebraic data types as they really are 499
8.10 Summary 501
8.11 Exercises 503

9 MOLECULE, ABSTRACT DATA TYPES, AND MODULES 525
9.1 The vocabulary of data abstraction 527
9.2 Introduction to Molecule, part I: Writing client code 528
9.3 Introduction, part II: Implementing an abstraction 530
9.4 The Molecule language 534
9.5 Molecule's initial basis 544
9.6 Program design: Abstractions 545
9.7 Key feature: Inspecting multiple representations 555
9.8 Molecule's type system: Enforcing abstraction 558
9.9 Notes on the interpreter 579
9.10 Abstract data types, modules, and overloading as they really are 580
9.11 Summary 585
9.12 Exercises 589

10 SMALLTALK AND OBJECT-ORIENTATION 609
10.1 Object-oriented programming by example 610
10.2 Data abstraction all over again 625
10.3 The μSmalltalk language 627
10.4 The initial basis of μSmalltalk 636
10.5 Object-oriented programming techniques 654
10.6 Technique I: Method dispatch replaces conditionals 654
10.7 Technique II: Abstract classes 656
10.8 Technique III: Multiple representations... 662
10.9 Technique IV: Invariants in object-oriented programming 673
10.10 Operational semantics 677
10.11 The interpreter 685
10.12 Smalltalk as it really is 700
10.13 Objects and classes as they really are 707
10.14 Summary 707
10.15 Exercises 713

AFTERWORD 727
BIBLIOGRAPHY 733
KEY WORDS AND PHRASES 745
CONCEPT INDEX 749