

Contents

1	Introduction	1
2	An imperative core	5
2.1	The Impcore language	8
2.2	Abstract syntax	13
2.3	Environments	15
2.4	Operational semantics	16
2.5	The interpreter	23
2.6	Operational semantics revisited: Proofs	49
2.7	Further reading	51
2.8	Exercises	53
3	Scheme, S-expressions, and first-class functions	61
3.1	Overview of μ Scheme and this chapter	63
3.2	Language I: S-expressions	65
3.3	Practice I: Recursive functions on lists	68
3.4	Interlude: Laws of applicative programming	75
3.5	Language II: Local variables and <code>let</code>	82
3.6	Practice II: Recursive functions on binary trees	83
3.7	Language III: First-class functions	87
3.8	Practice III: Higher-order functions on lists	96
3.9	Practice IV: Higher-order functions for polymorphism	100
3.10	Practice V: Continuation-passing style	106
3.11	Syntax and values of μ Scheme	113
3.12	Operational semantics	115
3.13	The initial basis	123
3.14	The interpreter	125
3.15	Large example: A metacircular evaluator	141
3.16	Scheme as it really is	147
3.17	Further reading	151
3.18	Exercises	152
4	Automatic Memory Management	173
4.1	What garbage is and where it comes from	174
4.2	Garbage-collection basics	176
4.3	The managed heap in μ Scheme	179
4.4	Mark-and-sweep garbage collection	183
4.5	Copying garbage collection	189

4.6	Debugging a garbage collector	198
4.7	Reference counting	200
4.8	Garbage collection as it really is	202
4.9	Summary	204
4.10	Exercises	206
5	Interlude: μScheme in ML	213
5.1	Environments	214
5.2	Abstract syntax and values	215
5.3	Evaluation	217
5.4	Primitives	219
5.5	Evaluating definitions	221
5.6	The read-eval-print loop	222
5.7	Initializing and running the interpreter	223
5.8	Building and exporting a program	223
5.9	Free and bound variables	225
5.10	Exercises	227
6	Type Systems for Impcore and μScheme	229
6.1	Typed Impcore: a statically typed imperative core	232
6.2	A type-checking interpreter for Typed Impcore	241
6.3	Extending Typed Impcore with arrays	248
6.4	Interlude: common type constructors	252
6.5	Type soundness	254
6.6	Polymorphic type systems and Typed μ Scheme	255
6.7	Type systems as they really are	275
6.8	Glossary	276
6.9	Further reading	277
6.10	Exercises	277
7	μML and type inference	283
7.1	μ ML: a nearly applicative language	285
7.2	Abstract syntax and values of μ ML	287
7.3	Operational semantics	288
7.4	Type system for μ ML	291
7.5	From type rules to type inference	301
7.6	The interpreter	318
7.7	Hindley-Milner as it really is	332
7.8	Further reading	333
7.9	Exercises	334
8	Haskell, Lazy Evaluation, and Monadic I/O	341
9	CLU and data abstraction	343
9.1	Data abstraction	347
9.2	The language	349
9.3	Implementation	367
9.4	Example—Polynomials	367
9.5	Clu as it really is	371
9.6	Summary	377

9.7	Type theory of μClu	379
9.8	The interpreter	382
9.9	Exercises	391
10	Smalltalk and object-orientation	395
10.1	Introduction to messages, classes, and inheritance	399
10.2	The $\mu\text{Smalltalk}$ language	407
10.3	The initial basis	412
10.4	Extended example—Discrete-event simulation	427
10.5	Predefined classes and objects	445
10.6	Interpreter and operational semantics	467
10.7	Smalltalk as it really is	496
10.8	Summary	503
10.9	Exercises	506
11	Prolog and logic programming	523
11.1	Logic as a programming language	525
11.2	The language	527
11.3	Small examples	544
11.4	Implementation	551
11.5	Larger example—The blocks world	559
11.6	Prolog as it really is	564
11.7	Prolog and mathematical logic	566
11.8	Summary	577
11.9	Exercises	579
A	Supporting code for Impcore	585
A.1	Uninteresting interfaces	585
A.2	Uninteresting implementations	586
B	Supporting code for μScheme	603
B.1	Uninteresting μScheme code	603
B.2	Uninteresting parts of the μScheme interpreter	603
C	Supporting code for garbage collection	621
C.1	Object-visiting procedures for mark-and-sweep collection	621
C.2	Root-scanning procedures for copying collection	624
C.3	Other supporting code	627
C.4	Adding root tracking to the μScheme interpreter	628
C.5	Implementing the root stack	641
C.6	Placeholders for exercises	641
D	Lexical analysis, parsing, and reading using ML	643
D.1	Controlling actions by using lazy streams	645
D.2	Managing parsing errors	651
D.3	Stream transformers, which act as parsers	652
D.4	Lexical analyzers: transformers of characters	659
D.5	Parsers: reading tokens and source-code locations	660
D.6	An interactive reader	667
D.7	Further reading	670

E	Supporting code for ML μScheme	671
E.1	Tokens of the μ Scheme language	671
E.2	Parsing	672
E.3	Further reading	675
F	Supporting code for Typed Impcore	677
F.1	Printing types and values	677
F.2	Parsing	678
F.3	Evaluation	680
G	Supporting code for Typed μScheme	683
G.1	Printing types and values	683
G.2	Parsing	684
G.3	Evaluation	686
G.4	Primitives of Typed μ Scheme	688
G.5	Initial basis	690
H	Supporting code for μML	693
H.1	Printing types and constraints	693
H.2	Parsing	694
H.3	Initial basis	696
I	Supporting code for μSmalltalk	699
I.1	Lexing and parsing	699
I.2	Support for tracing	705
I.3	Miscellaneous	707
J	Supporting code for μProlog	709
J.1	String conversions	709
J.2	Lexical analysis	710
J.3	Parsing	714
J.4	Command line	719
K	EBNF	721
L	Supporting discriminated unions in C	723
L.1	Lexical analysis	724
L.2	Abstract syntax and parsing	725
L.3	Interface to a general-purpose prettyprinter	727
L.4	C types	729
L.5	Prettyprinting C types	730
L.6	Creating C types from sums and products	731
L.7	Creating constructor functions and prototypes	733
L.8	Writing the output	735
L.9	Implementation of the prettyprinter	737
L.10	Putting everything together	740
	References	741
	Code Index	753

I	Answers to selected hard problems	785
M	Completed garbage collectors	787
M.1	Mark and sweep	787
M.2	Copying collection	791
N	Implementation of Typed μScheme type checking	797
O	Complete implementation of μML type inference	803
O.1	Independent constraint solving	803
O.2	Type inference	804
O.3	Primitives	806
P	Complete implementation of μHaskell	807
Q	Large integers in μSmalltalk	813
Q.1	Bignums	813
Q.2	Testing	817
Q.3	Large integers	818
Q.4	Modifications to <code>SmallInteger</code>	820
Q.5	More test cases	821
R	Complete implementation of μProlog	823
R.1	Substitution	823
R.2	Unification	824

