

# Contents

<b>1</b>	<b>Introduction</b>	1
1.1	Files for This Chapter	1
1.2	Meta Language and Object Language	1
1.3	A Simple Language of Expressions	2
1.3.1	Expressions Without Variables	2
1.3.2	Expressions with Variables	3
1.4	Syntax and Semantics	5
1.5	Representing Expressions by Objects	6
1.6	The History of Programming Languages	8
1.7	Exercises	9
	References	12
<b>2</b>	<b>Interpreters and Compilers</b>	13
2.1	Files for This Chapter	13
2.2	Interpreters and Compilers	13
2.3	Scope and Bound and Free Variables	14
2.3.1	Expressions with Let-Bindings and Static Scope	15
2.3.2	Closed Expressions	16
2.3.3	The Set of Free Variables	17
2.3.4	Substitution: Replacing Variables by Expressions	17
2.4	Integer Addresses Instead of Names	20
2.5	Stack Machines for Expression Evaluation	22
2.6	Postscript, a Stack-Based Language	23
2.7	Compiling Expressions to Stack Machine Code	25
2.8	Implementing an Abstract Machine in Java	26
2.9	History and Literature	26
2.10	Exercises	26
	References	29
<b>3</b>	<b>From Concrete Syntax to Abstract Syntax</b>	31
3.1	Preparatory Reading	31
3.2	Lexers, Parsers, and Generators	32

3.3	Regular Expressions in Lexer Specifications . . . . .	33
3.4	Grammars in Parser Specifications . . . . .	34
3.5	Working with F# Modules . . . . .	35
3.6	Using <code>fslex</code> and <code>fsyacc</code> . . . . .	36
	3.6.1 Installing and Using <code>fslex</code> and <code>fsyacc</code> . . . . .	37
	3.6.2 Parser Specification for Expressions . . . . .	37
	3.6.3 Lexer Specification for Expressions . . . . .	38
	3.6.4 The <code>ExprPar.fsyacc.output</code> File Generated by <code>fsyacc</code> . . . . .	41
	3.6.5 Exercising the Parser Automaton . . . . .	44
	3.6.6 Shift/Reduce Conflicts . . . . .	45
3.7	Lexer and Parser Specification Examples . . . . .	47
	3.7.1 A Small Functional Language . . . . .	47
	3.7.2 Lexer and Parser Specifications for Micro-SQL . . . . .	48
3.8	A Handwritten Recursive Descent Parser . . . . .	48
3.9	JavaCC: Lexer-, Parser-, and Tree Generator . . . . .	50
3.10	History and Literature . . . . .	53
3.11	Exercises . . . . .	55
	References . . . . .	57
<b>4</b>	<b>A First-Order Functional Language . . . . .</b>	<b>59</b>
	4.1 Files for This Chapter . . . . .	59
	4.2 Examples and Abstract Syntax . . . . .	60
	4.3 Run-Time Values: Integers and Closures . . . . .	61
	4.4 A Simple Environment Implementation . . . . .	62
	4.5 Evaluating the Functional Language . . . . .	62
	4.6 Evaluation Rules for Micro-ML . . . . .	64
	4.7 Static Scope and Dynamic Scope . . . . .	66
	4.8 Type-Checking an Explicitly Typed Language . . . . .	68
	4.9 Type Rules for Monomorphic Types . . . . .	70
	4.10 Static Typing and Dynamic Typing . . . . .	72
	4.10.1 Dynamic Typing in Java and C# Array Assignment . . . . .	73
	4.10.2 Dynamic Typing in Non-generic Collection Classes . . . . .	74
	4.11 History and Literature . . . . .	74
	4.12 Exercises . . . . .	75
	References . . . . .	78
<b>5</b>	<b>Higher-Order Functions . . . . .</b>	<b>81</b>
	5.1 Files for This Chapter . . . . .	81
	5.2 Higher-Order Functions in F# . . . . .	81
	5.3 Higher-Order Functions in the Mainstream . . . . .	82
	5.3.1 Higher-Order Functions in Java 5 . . . . .	82
	5.3.2 Higher-Order Functions in Java 8 . . . . .	84

- 5.3.3 Higher-Order Functions in C# . . . . . 85
- 5.3.4 Google MapReduce . . . . . 86
- 5.4 A Higher-Order Functional Language . . . . . 86
- 5.5 Eager and Lazy Evaluation . . . . . 87
- 5.6 The Lambda Calculus . . . . . 88
- 5.7 History and Literature . . . . . 91
- 5.8 Exercises . . . . . 91
- References . . . . . 96
- 6 Polymorphic Types . . . . . 97**
  - 6.1 Files for This Chapter . . . . . 97
  - 6.2 ML-Style Polymorphic Types . . . . . 97
    - 6.2.1 Informal Explanation of ML Type Inference . . . . . 98
    - 6.2.2 Which Type Parameters May Be Generalized . . . . . 100
  - 6.3 Type Rules for Polymorphic Types . . . . . 101
  - 6.4 Implementing ML Type Inference . . . . . 103
    - 6.4.1 Type Equation Solving by Unification . . . . . 106
    - 6.4.2 The Union-Find Algorithm . . . . . 106
    - 6.4.3 The Complexity of ML-Style Type Inference . . . . . 107
  - 6.5 Generic Types in Java and C# . . . . . 108
  - 6.6 Co-Variance and Contra-Variance . . . . . 110
    - 6.6.1 Java Wildcards . . . . . 111
    - 6.6.2 C# Variance Declarations . . . . . 112
    - 6.6.3 The Variance Mechanisms of Java and C# . . . . . 113
  - 6.7 History and Literature . . . . . 114
  - 6.8 Exercises . . . . . 114
  - References . . . . . 117
- 7 Imperative Languages . . . . . 119**
  - 7.1 Files for This Chapter . . . . . 119
  - 7.2 A Naive Imperative Language . . . . . 120
  - 7.3 Environment and Store . . . . . 121
  - 7.4 Parameter Passing Mechanisms . . . . . 122
  - 7.5 The C Programming Language . . . . . 124
    - 7.5.1 Integers, Pointers and Arrays in C . . . . . 124
    - 7.5.2 Type Declarations in C . . . . . 126
  - 7.6 The Micro-C Language . . . . . 127
    - 7.6.1 Interpreting Micro-C . . . . . 129
    - 7.6.2 Example Programs in Micro-C . . . . . 129
    - 7.6.3 Lexer Specification for Micro-C . . . . . 130
    - 7.6.4 Parser Specification for Micro-C . . . . . 132
  - 7.7 Notes on Strachey’s *Fundamental Concepts* . . . . . 134
  - 7.8 History and Literature . . . . . 137
  - 7.9 Exercises . . . . . 137
  - References . . . . . 140

- 8 Compiling Micro-C** . . . . . 141
  - 8.1 Files for This Chapter . . . . . 141
  - 8.2 An Abstract Stack Machine . . . . . 142
    - 8.2.1 The State of the Abstract Machine . . . . . 142
    - 8.2.2 The Abstract Machine Instruction Set . . . . . 143
    - 8.2.3 The Symbolic Machine Code . . . . . 145
    - 8.2.4 The Abstract Machine Implemented in Java . . . . . 145
    - 8.2.5 The Abstract Machine Implemented in C . . . . . 147
  - 8.3 The Structure of the Stack at Run-Time . . . . . 147
  - 8.4 Compiling Micro-C to Abstract Machine Code . . . . . 148
  - 8.5 Compilation Schemes for Micro-C . . . . . 149
  - 8.6 Compilation of Statements . . . . . 150
  - 8.7 Compilation of Expressions . . . . . 153
  - 8.8 Compilation of Access Expressions . . . . . 154
  - 8.9 Compilation to Real Machine Code . . . . . 155
  - 8.10 History and Literature . . . . . 155
  - 8.11 Exercises . . . . . 156
  - References . . . . . 159
- 9 Real-World Abstract Machines** . . . . . 161
  - 9.1 Files for This Chapter . . . . . 161
  - 9.2 An Overview of Abstract Machines . . . . . 161
  - 9.3 The Java Virtual Machine (JVM) . . . . . 163
    - 9.3.1 The JVM Run-Time State . . . . . 163
    - 9.3.2 The JVM Bytecode . . . . . 165
    - 9.3.3 The Contents of JVM Class Files . . . . . 165
    - 9.3.4 Bytecode Verification . . . . . 169
  - 9.4 The Common Language Infrastructure (CLI) . . . . . 169
  - 9.5 Generic Types in CLI and JVM . . . . . 172
    - 9.5.1 A Generic Class in Bytecode . . . . . 173
    - 9.5.2 Consequences for Java . . . . . 174
  - 9.6 Decompilers for Java and C# . . . . . 175
  - 9.7 Just-in-Time Compilation . . . . . 176
  - 9.8 History and Literature . . . . . 177
  - 9.9 Exercises . . . . . 178
  - References . . . . . 180
- 10 Garbage Collection** . . . . . 183
  - 10.1 Files for This Chapter . . . . . 183
  - 10.2 Predictable Lifetime and Stack Allocation . . . . . 183
  - 10.3 Unpredictable Lifetime and Heap Allocation . . . . . 184
  - 10.4 Allocation in a Heap . . . . . 185
  - 10.5 Garbage Collection Techniques . . . . . 186
    - 10.5.1 The Heap and the Freelist . . . . . 187
    - 10.5.2 Garbage Collection by Reference Counting . . . . . 187

- 10.5.3 Mark-Sweep Collection . . . . . 188
- 10.5.4 Two-Space Stop-and-Copy Collection . . . . . 189
- 10.5.5 Generational Garbage Collection . . . . . 191
- 10.5.6 Conservative Garbage Collection . . . . . 192
- 10.5.7 Garbage Collectors Used in Existing Systems . . . . . 192
- 10.6 Programming with a Garbage Collector . . . . . 193
  - 10.6.1 Memory Leaks . . . . . 193
  - 10.6.2 Finalizers . . . . . 194
  - 10.6.3 Calling the Garbage Collector . . . . . 194
- 10.7 Implementing a Garbage Collector in C . . . . . 195
  - 10.7.1 The List-C Language . . . . . 195
  - 10.7.2 The List-C Machine . . . . . 198
  - 10.7.3 Distinguishing References from Integers . . . . . 198
  - 10.7.4 Memory Structures in the Garbage Collector . . . . . 199
  - 10.7.5 Actions of the Garbage Collector . . . . . 200
- 10.8 History and Literature . . . . . 202
- 10.9 Exercises . . . . . 202
- References . . . . . 207
- 11 Continuations . . . . . 209**
  - 11.1 Files for This Chapter . . . . . 209
  - 11.2 Tail-Calls and Tail-Recursive Functions . . . . . 210
    - 11.2.1 A Recursive but Not Tail-Recursive Function . . . . . 210
    - 11.2.2 A Tail-Recursive Function . . . . . 210
    - 11.2.3 Which Calls Are Tail Calls? . . . . . 212
  - 11.3 Continuations and Continuation-Passing Style . . . . . 212
    - 11.3.1 Writing a Function in Continuation-Passing Style . . . . . 213
    - 11.3.2 Continuations and Accumulating Parameters . . . . . 214
    - 11.3.3 The CPS Transformation . . . . . 214
  - 11.4 Interpreters in Continuation-Passing Style . . . . . 215
    - 11.4.1 A Continuation-Based Functional Interpreter . . . . . 215
    - 11.4.2 Tail Position and Continuation-Based Interpreters . . . . . 217
    - 11.4.3 A Continuation-Based Imperative Interpreter . . . . . 217
  - 11.5 The Frame Stack and Continuations . . . . . 219
  - 11.6 Exception Handling in a Stack Machine . . . . . 220
  - 11.7 Continuations and Tail Calls . . . . . 221
  - 11.8 Callcc: Call with Current Continuation . . . . . 223
  - 11.9 Continuations and Backtracking . . . . . 224
    - 11.9.1 Expressions in Icon . . . . . 224
    - 11.9.2 Using Continuations to Implement Backtracking . . . . . 225
  - 11.10 History and Literature . . . . . 227
  - 11.11 Exercises . . . . . 228
  - References . . . . . 231

<b>12</b>	<b>A Locally Optimizing Compiler</b>	233
12.1	Files for This Chapter	233
12.2	Generating Optimized Code Backwards	233
12.3	Backwards Compilation Functions	234
12.3.1	Optimizing Expression Code While Generating It	236
12.3.2	The Old Compilation of Jumps	238
12.3.3	Optimizing a Jump While Generating It	238
12.3.4	Optimizing Logical Expression Code	240
12.3.5	Eliminating Dead Code	242
12.3.6	Optimizing Tail Calls	242
12.3.7	Remaining Deficiencies of the Generated Code	245
12.4	Other Optimizations	246
12.5	A Command Line Compiler for Micro-C	247
12.6	History and Literature	248
12.7	Exercises	248
	References	251
<b>13</b>	<b>Compiling Micro-SML</b>	253
13.1	Files for This Chapter	253
13.2	Grammar for Micro-SML	254
13.2.1	Example Programs	255
13.2.2	Abstract Syntax	256
13.2.3	Prettyprinting	256
13.2.4	Tail Calls	257
13.2.5	Free Variables	258
13.3	Type Inference for Micro-SML	259
13.3.1	Type Inference Implementation	261
13.3.2	Annotated Type Information	263
13.4	Interpreting Micro-SML	263
13.4.1	Continuations	266
13.4.2	Sequence	267
13.4.3	Functions	267
13.4.4	Exceptions	268
13.5	Compiling Micro-SML	269
13.5.1	Extensions to Abstract Machine Instruction Set	269
13.5.2	Compilation of Primitive Micro-SML Expressions	272
13.5.3	Compilation of Variable Access	273
13.5.4	Compilation of Value Declarations	274
13.5.5	Compilation of Let Expressions and Functions	277
13.5.6	Compilation of Exceptions	278
13.6	Exercises	279
	Reference	281

- 14 Real Machine Code** . . . . . 283
  - 14.1 Files for This Chapter . . . . . 283
  - 14.2 The x86 Processor Family . . . . . 284
    - 14.2.1 Evolution of the x86 Processor Family . . . . . 284
    - 14.2.2 Registers of the x86 Architecture . . . . . 285
    - 14.2.3 The x86 Instruction Set . . . . . 287
    - 14.2.4 The x86 Stack Layout . . . . . 288
    - 14.2.5 An Assembly Code Example . . . . . 288
  - 14.3 Compiling Micro-C to x86 Code . . . . . 290
    - 14.3.1 Compilation Strategy . . . . . 291
    - 14.3.2 Representing x86 Machine Code in the Compiler . . . . . 292
    - 14.3.3 Stack Layout for Micro-C x86 Code . . . . . 293
  - 14.4 The micro-C x86 Compiler . . . . . 295
  - 14.5 Compilation Schemes for Micro-C . . . . . 296
  - 14.6 Compilation of Statements . . . . . 297
  - 14.7 Compilation of Expressions . . . . . 297
  - 14.8 Compilation of Access Expressions . . . . . 300
  - 14.9 Choosing Target Registers . . . . . 301
  - 14.10 Improving the Compiler . . . . . 302
  - 14.11 History and Literature . . . . . 303
  - 14.12 Exercises . . . . . 303
  - References . . . . . 305
- Appendix A: Crash Course in F#** . . . . . 307
- Index** . . . . . 333



<http://www.springer.com/978-3-319-60788-7>

Programming Language Concepts

Sestoft, P.

2017, XV, 341 p. 87 illus., Softcover

ISBN: 978-3-319-60788-7