Contents

1 Introduction **1**

1.1 What is a compiler? . . . 1
1.2 The phases of a compiler . . . 2
1.3 Interpreters . . . 3
1.4 Why learn about compilers? . . . 4
1.5 The structure of this book . . . 5
1.6 To the lecturer . . . 6
1.7 Acknowledgements . . . 7
1.8 Permission to use . . . 7

2 Lexical Analysis **9**

2.1 Introduction . . . 9
2.2 Regular expressions . . . 10
2.2.1 Shorthands . . . 13
2.2.2 Examples . . . 14
2.3 Nondeterministic finite automata . . . 15
2.4 Converting a regular expression to an NFA . . . 18
2.4.1 Optimisations . . . 20
2.5 Deterministic finite automata . . . 22
2.6 Converting an NFA to a DFA . . . 23
2.6.1 Solving set equations . . . 23
2.6.2 The subset construction . . . 26
2.7 Size versus speed . . . 29
2.8 Minimisation of DFAs . . . 30
2.8.1 Example . . . 32
2.8.2 Dead states . . . 34
2.9 Lexers and lexer generators . . . 35
2.9.1 Lexer generators . . . 41
2.10 Properties of regular languages . . . 42
2.10.1 Relative expressive power . . . 42
2.10.2 Limits to expressive power . . . 44

2.10.3 Closure properties 45
2.11 Further reading 46
Exercises 46

3 Syntax Analysis **53**
3.1 Introduction 53
3.2 Context-free grammars 54
3.2.1 How to write context free grammars 56
3.3 Derivation 58
3.3.1 Syntax trees and ambiguity 60
3.4 Operator precedence 63
3.4.1 Rewriting ambiguous expression grammars 64
3.5 Other sources of ambiguity 66
3.6 Syntax analysis 68
3.7 Predictive parsing 68
3.8 *Nullable* and *FIRST* 69
3.9 Predictive parsing revisited 73
3.10 *FOLLOW* 74
3.11 A larger example 77
3.12 LL(1) parsing 79
3.12.1 Recursive descent 80
3.12.2 Table-driven LL(1) parsing 81
3.12.3 Conflicts 82
3.13 Rewriting a grammar for LL(1) parsing 84
3.13.1 Eliminating left-recursion 84
3.13.2 Left-factorisation 86
3.13.3 Construction of LL(1) parsers summarized 87
3.14 SLR parsing 88
3.15 Constructing SLR parse tables 90
3.15.1 Conflicts in SLR parse-tables 94
3.16 Using precedence rules in LR parse tables 95
3.17 Using LR-parser generators 98
3.17.1 Declarations and actions 99
3.17.2 Abstract syntax 99
3.17.3 Conflict handling in parser generators 102
3.18 Properties of context-free languages 104
3.19 Further reading 105
Exercises 105

4 Scopes and Symbol Tables **113**
4.1 Introduction . . . 113
4.2 Symbol tables . . . 114
4.2.1 Implementation of symbol tables . . . 115
4.2.2 Simple persistent symbol tables . . . 115
4.2.3 A simple imperative symbol table . . . 117
4.2.4 Efficiency issues . . . 117
4.2.5 Shared or separate name spaces . . . 118
4.3 Further reading . . . 118
Exercises . . . 118

5 Interpretation **121**
5.1 Introduction . . . 121
5.2 The structure of an interpreter . . . 122
5.3 A small example language . . . 122
5.4 An interpreter for the example language . . . 124
5.4.1 Evaluating expressions . . . 124
5.4.2 Interpreting function calls . . . 126
5.4.3 Interpreting a program . . . 128
5.5 Advantages and disadvantages of interpretation . . . 128
5.6 Further reading . . . 130
Exercises . . . 130

6 Type Checking **133**
6.1 Introduction . . . 133
6.2 The design space of types . . . 133
6.3 Attributes . . . 135
6.4 Environments for type checking . . . 135
6.5 Type checking expressions . . . 136
6.6 Type checking of function declarations . . . 138
6.7 Type checking a program . . . 139
6.8 Advanced type checking . . . 140
6.9 Further reading . . . 143
Exercises . . . 143

7 Intermediate-Code Generation **147**
7.1 Introduction . . . 147
7.2 Choosing an intermediate language . . . 148
7.3 The intermediate language . . . 150
7.4 Syntax-directed translation . . . 151
7.5 Generating code from expressions . . . 152
7.5.1 Examples of translation . . . 155

7.6 Translating statements . . . 156
7.7 Logical operators . . . 159
7.7.1 Sequential logical operators . . . 160
7.8 Advanced control statements . . . 164
7.9 Translating structured data . . . 165
7.9.1 Floating-point values . . . 165
7.9.2 Arrays . . . 165
7.9.3 Strings . . . 171
7.9.4 Records/structs and unions . . . 171
7.10 Translating declarations . . . 172
7.10.1 Example: Simple local declarations . . . 172
7.11 Further reading . . . 172
Exercises . . . 173

8 Machine-Code Generation **179**
8.1 Introduction . . . 179
8.2 Conditional jumps . . . 180
8.3 Constants . . . 181
8.4 Exploiting complex instructions . . . 181
8.4.1 Two-address instructions . . . 186
8.5 Optimisations . . . 186
8.6 Further reading . . . 188
Exercises . . . 188

9 Register Allocation **191**
9.1 Introduction . . . 191
9.2 Liveness . . . 192
9.3 Liveness analysis . . . 193
9.4 Interference . . . 196
9.5 Register allocation by graph colouring . . . 199
9.6 Spilling . . . 200
9.7 Heuristics . . . 202
9.7.1 Removing redundant moves . . . 205
9.7.2 Using explicit register numbers . . . 205
9.8 Further reading . . . 206
Exercises . . . 206

10 Function calls **209**
10.1 Introduction . . . 209
10.1.1 The call stack . . . 209
10.2 Activation records . . . 210
10.3 Prologues, epilogues and call-sequences . . . 211

10.4 Caller-saves versus callee-saves . . . 213
10.5 Using registers to pass parameters . . . 215
10.6 Interaction with the register allocator . . . 219
10.7 Accessing non-local variables . . . 221
10.7.1 Global variables . . . 221
10.7.2 Call-by-reference parameters . . . 222
10.7.3 Nested scopes . . . 223
10.8 Variants . . . 226
10.8.1 Variable-sized frames . . . 226
10.8.2 Variable number of parameters . . . 227
10.8.3 Direction of stack-growth and position of FP . . . 227
10.8.4 Register stacks . . . 228
10.8.5 Functions as values . . . 228
10.9 Further reading . . . 229
Exercises . . . 229

11 Analysis and optimisation **231**
11.1 Data-flow analysis . . . 232
11.2 Common subexpression elimination . . . 233
11.2.1 Available assignments . . . 233
11.2.2 Example of available-assignments analysis . . . 236
11.2.3 Using available assignment analysis for common subexpression elimination . . . 237
11.3 Jump-to-jump elimination . . . 240
11.4 Index-check elimination . . . 241
11.5 Limitations of data-flow analyses . . . 244
11.6 Loop optimisations . . . 245
11.6.1 Code hoisting . . . 245
11.6.2 Memory prefetching . . . 246
11.7 Optimisations for function calls . . . 248
11.7.1 Inlining . . . 249
11.7.2 Tail-call optimisation . . . 250
11.8 Specialisation . . . 252
11.9 Further reading . . . 254
Exercises . . . 254

12 Memory management **257**
12.1 Introduction . . . 257
12.2 Static allocation . . . 257
12.2.1 Limitations . . . 258
12.3 Stack allocation . . . 258

12.4 Heap allocation . . . 259
12.5 Manual memory management . . . 259
12.5.1 A simple implementation of `malloc()` and `free()` . . . 260
12.5.2 Joining freed blocks . . . 263
12.5.3 Sorting by block size . . . 264
12.5.4 Summary of manual memory management . . . 265
12.6 Automatic memory management . . . 266
12.7 Reference counting . . . 266
12.8 Tracing garbage collectors . . . 268
12.8.1 Scan-sweep collection . . . 269
12.8.2 Two-space collection . . . 271
12.8.3 Generational and concurrent collectors . . . 273
12.9 Summary of automatic memory management . . . 276
12.10 Further reading . . . 277
Exercises . . . 277

13 Bootstrapping a compiler **281**
13.1 Introduction . . . 281
13.2 Notation . . . 281
13.3 Compiling compilers . . . 283
13.3.1 Full bootstrap . . . 285
13.4 Further reading . . . 288
Exercises . . . 288

A Set notation and concepts **291**
A.1 Basic concepts and notation . . . 291
A.1.1 Operations and predicates . . . 291
A.1.2 Properties of set operations . . . 292
A.2 Set-builder notation . . . 293
A.3 Sets of sets . . . 294
A.4 Set equations . . . 295
A.4.1 Monotonic set functions . . . 295
A.4.2 Distributive functions . . . 296
A.4.3 Simultaneous equations . . . 297
Exercises . . . 297

List of Figures

2.1 Regular expressions 11
2.2 Some algebraic properties of regular expressions 14
2.3 Example of an NFA 17
2.4 Constructing NFA fragments from regular expressions 19
2.5 NFA for the regular expression (a|b)*ac 20
2.6 Optimised NFA construction for regular expression shorthands . . 21
2.7 Optimised NFA for [0-9]+ 21
2.8 Example of a DFA 22
2.9 DFA constructed from the NFA in figure 2.5 29
2.10 Non-minimal DFA 32
2.11 Minimal DFA 34
2.12 Combined NFA for several tokens 38
2.13 Combined DFA for several tokens 39
2.14 A 4-state NFA that gives 15 DFA states 44

3.1 From regular expressions to context free grammars 56
3.2 Simple expression grammar 57
3.3 Simple statement grammar 57
3.4 Example grammar 59
3.5 Derivation of the string aabbbcc using grammar 3.4 59
3.6 Leftmost derivation of the string aabbbcc using grammar 3.4 . . . 59
3.7 Syntax tree for the string aabbbcc using grammar 3.4 61
3.8 Alternative syntax tree for the string aabbbcc using grammar 3.4 . 61
3.9 Unambiguous version of grammar 3.4 62
3.10 Preferred syntax tree for 2+3*4 using grammar 3.2 63
3.11 Unambiguous expression grammar 66
3.12 Syntax tree for 2+3*4 using grammar 3.11 67
3.13 Unambiguous grammar for statements 68
3.14 Fixed-point iteration for calculation of *Nullable* 71
3.15 Fixed-point iteration for calculation of *FIRST* 72
3.16 Recursive descent parser for grammar 3.9 81

3.17 LL(1) table for grammar 3.9 82
3.18 Program for table-driven LL(1) parsing 83
3.19 Input and stack during table-driven LL(1) parsing 83
3.20 Removing left-recursion from grammar 3.11 85
3.21 Left-factorised grammar for conditionals 87
3.22 SLR table for grammar 3.9 90
3.23 Algorithm for SLR parsing 91
3.24 Example SLR parsing 91
3.25 Example grammar for SLR-table construction 92
3.26 NFAs for the productions in grammar 3.25 92
3.27 Epsilon-transitions added to figure 3.26 93
3.28 SLR DFA for grammar 3.9 94
3.29 Summary of SLR parse-table construction 95
3.30 Textual representation of NFA states 103

5.1 Example language for interpretation 123
5.2 Evaluating expressions 125
5.3 Evaluating a function call 127
5.4 Interpreting a program 128

6.1 The design space of types 134
6.2 Type checking of expressions 137
6.3 Type checking a function declaration 139
6.4 Type checking a program 141

7.1 The intermediate language 150
7.2 A simple expression language 152
7.3 Translating an expression 154
7.4 Statement language 156
7.5 Translation of statements 158
7.6 Translation of simple conditions 159
7.7 Example language with logical operators 161
7.8 Translation of sequential logical operators 162
7.9 Translation for one-dimensional arrays 166
7.10 A two-dimensional array 168
7.11 Translation of multi-dimensional arrays 169
7.12 Translation of simple declarations 173

8.1 Pattern/replacement pairs for a subset of the MIPS instruction set . 185

9.1 Gen and kill sets 194
9.2 Example program for liveness analysis and register allocation . . . 195

9.3 *succ*, *gen* and *kill* for the program in figure 9.2 196
9.4 Fixed-point iteration for liveness analysis 197
9.5 Interference graph for the program in figure 9.2 198
9.6 Algorithm 9.3 applied to the graph in figure 9.5 202
9.7 Program from figure 9.2 after spilling variable a 203
9.8 Interference graph for the program in figure 9.7 203
9.9 Colouring of the graph in figure 9.8 204

10.1 Simple activation record layout 211
10.2 Prologue and epilogue for the frame layout shown in figure 10.1 . 212
10.3 Call sequence for $x :=$ `CALL` $f(a_1,\ldots,a_n)$ using the frame layout shown in figure 10.1 213
10.4 Activation record layout for callee-saves 214
10.5 Prologue and epilogue for callee-saves 214
10.6 Call sequence for $x :=$ `CALL` $f(a_1,\ldots,a_n)$ for callee-saves 215
10.7 Possible division of registers for 16-register architecture 216
10.8 Activation record layout for the register division shown in figure 10.7 216
10.9 Prologue and epilogue for the register division shown in figure 10.7 217
10.10 Call sequence for $x :=$ `CALL` $f(a_1,\ldots,a_n)$ for the register division shown in figure 10.7 218
10.11 Example of nested scopes in Pascal 223
10.12 Adding an explicit frame-pointer to the program from figure 10.11 224
10.13 Activation record with static link 225
10.14 Activation records for `f` and `g` from figure 10.11 225

11.1 Gen and kill sets for available assignments 235
11.2 Example program for available-assignments analysis 236
11.3 *pred*, *gen* and *kill* for the program in figure 11.2 237
11.4 Fixed-point iteration for available-assignment analysis 238
11.5 The program in figure 11.2 after common subexpression elimination. 239
11.6 Equations for index-check elimination 242
11.7 Intermediate code for for-loop with index check 244

12.1 Operations on a free list 261