CONTENTS IN DETAIL

FOREWORD by Aaron Patterson **xv**

ACKNOWLEDGMENTS **xvii**

INTRODUCTION **xix**

Who This Book Is For xx
Using Ruby to Test Itself xx
Which Implementation of Ruby? xxi
Overview xxi

1
TOKENIZATION AND PARSING **3**

Tokens: The Words That Make Up the Ruby Language 4
The parser_yylex Function 8
Experiment 1-1: Using Ripper to Tokenize Different Ruby Scripts 9
Parsing: How Ruby Understands Your Code 12
Understanding the LALR Parse Algorithm 13
Some Actual Ruby Grammar Rules 20
Reading a Bison Grammar Rule 22
Experiment 1-2: Using Ripper to Parse Different Ruby Scripts 23
Summary 29

2
COMPILATION **31**

No Compiler for Ruby 1.8 32
Ruby 1.9 and 2.0 Introduce a Compiler 33
How Ruby Compiles a Simple Script 34
Compiling a Call to a Block 38
How Ruby Iterates Through the AST 42
Experiment 2-1: Displaying YARV Instructions 44
The Local Table 46
Compiling Optional Arguments 48
Compiling Keyword Arguments 49
Experiment 2-2: Displaying the Local Table 51
Summary 53

3
HOW RUBY EXECUTES YOUR CODE **55**

YARV's Internal Stack and Your Ruby Stack 56
Stepping Through How Ruby Executes a Simple Script 58
Executing a Call to a Block 61
Taking a Close Look at a YARV Instruction 63
Experiment 3-1: Benchmarking Ruby 2.0 and Ruby 1.9 vs. Ruby 1.8 65

Local and Dynamic Access of Ruby Variables 67
Local Variable Access 67
Method Arguments Are Treated Like Local Variables 70
Dynamic Variable Access. 71
Climbing the Environment Pointer Ladder in C 74
Experiment 3-2: Exploring Special Variables 75
A Definitive List of Special Variables 79
Summary 81

4
CONTROL STRUCTURES AND METHOD DISPATCH 83

How Ruby Executes an if Statement 84
Jumping from One Scope to Another 86
Catch Tables 88
Other Uses for Catch Tables. 90
Experiment 4-1: Testing How Ruby Implements for Loops Internally 90
The send Instruction: Ruby's Most Complex Control Structure. 92
Method Lookup and Method Dispatch 92
Eleven Types of Ruby Methods 93
Calling Normal Ruby Methods. 95
Preparing Arguments for Normal Ruby Methods 95
Calling Built-In Ruby Methods 97
Calling attr_reader and attr_writer 97
Method Dispatch Optimizes attr_reader and attr_writer 98
Experiment 4-2: Exploring How Ruby Implements Keyword Arguments 99
Summary 103

5
OBJECTS AND CLASSES 105

Inside a Ruby Object 106
Inspecting klass and ivptr 107
Visualizing Two Instances of One Class 108
Generic Objects 109
Simple Ruby Values Don't Require a Structure at All 110
Do Generic Objects Have Instance Variables? 111
Reading the RBasic and RObject C Structure Definitions. 112
Where Does Ruby Save Instance Variables for Generic Objects? 113
Experiment 5-1: How Long Does It Take to Save a New Instance Variable?. 113
What's Inside the RClass Structure? 115
Inheritance 118
Class Instance Variables vs. Class Variables. 120
Getting and Setting Class Variables 122
Constants. 124
The Actual RClass Structure 125
Reading the RClass C Structure Definition. 127
Experiment 5-2: Where Does Ruby Save Class Methods? 127
Summary 131

6
METHOD LOOKUP AND CONSTANT LOOKUP 133

How Ruby Implements Modules 135
Modules Are Classes 135
Including a Module into a Class 136
Ruby's Method Lookup Algorithm 138
A Method Lookup Example 139
The Method Lookup Algorithm in Action 140
Multiple Inheritance in Ruby 141
The Global Method Cache 142
The Inline Method Cache 143
Clearing Ruby's Method Caches 143
Including Two Modules into One Class 144
Including One Module into Another 145
A Module#prepend Example 146
How Ruby Implements Module#prepend 150
Experiment 6-1: Modifying a Module After Including It. 151
Classes See Methods Added to a Module Later 152
Classes Don't See Submodules Included Later 152
Included Classes Share the Method Table with the Original Module 153
A Close Look at How Ruby Copies Modules 154
Constant Lookup 155
Finding a Constant in a Superclass 156
How Does Ruby Find a Constant in the Parent Namespace? 157
Lexical Scope in Ruby 158
Creating a Constant for a New Class or Module 159
Finding a Constant in the Parent Namespace Using Lexical Scope 160
Ruby's Constant Lookup Algorithm 162
Experiment 6-2: Which Constant Will Ruby Find First? 162
Ruby's Actual Constant Lookup Algorithm 163
Summary 165

7
THE HASH TABLE: THE WORKHORSE OF RUBY INTERNALS 167

Hash Tables in Ruby 169
Saving a Value in a Hash Table 169
Retrieving a Value from a Hash Table 171
Experiment 7-1: Retrieving a Value from Hashes of Varying Sizes 172
How Hash Tables Expand to Accommodate More Values 174
Hash Collisions 174
Rehashing Entries 175
How Does Ruby Rehash Entries in a Hash Table? 176
Experiment 7-2: Inserting One New Element into Hashes of Varying Sizes 177
Where Do the Magic Numbers *57* and *67* Come From? 180
How Ruby Implements Hash Functions 181
Experiment 7-3: Using Objects as Keys in a Hash 183
Hash Optimization in Ruby 2.0 187
Summary 189

8
HOW RUBY BORROWED A DECADES-OLD IDEA FROM LISP 191

Blocks: Closures in Ruby 192
Stepping Through How Ruby Calls a Block 194
Borrowing an Idea from 1975 196
The rb_block_t and rb_control_frame_t Structures 198
Experiment 8-1: Which Is Faster: A while Loop or Passing a Block to each? 200
Lambdas and Procs: Treating a Function as a First-Class Citizen 203
Stack vs. Heap Memory 204
A Closer Look at How Ruby Saves a String Value 204
How Ruby Creates a Lambda 207
How Ruby Calls a Lambda 209
The Proc Object 211
Experiment 8-2: Changing Local Variables After Calling lambda 214
Calling lambda More Than Once in the Same Scope 216
Summary 217

9
METAPROGRAMMING 219

Alternative Ways to Define Methods 221
Ruby's Normal Method Definition Process 221
Defining Class Methods Using an Object Prefix 223
Defining Class Methods Using a New Lexical Scope 224
Defining Methods Using Singleton Classes 226
Defining Methods Using Singleton Classes in a Lexical Scope 227
Creating Refinements 228
Using Refinements 229
Experiment 9-1: Who Am I? How self Changes with Lexical Scope 231
self in the Top Scope 231
self in a Class Scope 232
self in a Metaclass Scope 233
self Inside a Class Method 234
Metaprogramming and Closures: eval, instance_eval, and binding 236
Code That Writes Code 236
Calling eval with binding 238
An instance_eval Example 240
Another Important Part of Ruby Closures 241
instance_eval Changes self to the Receiver 242
instance_eval Creates a Singleton Class for a New Lexical Scope 243
How Ruby Keeps Track of Lexical Scope for Blocks 244
Experiment 9-2: Using a Closure to Define a Method 246
Using define_method 246
Methods Acting as Closures 247
Summary 248

10
JRUBY: RUBY ON THE JVM **251**

Running Programs with MRI and JRuby . . . 252
How JRuby Parses and Compiles Your Code . . . 254
How JRuby Executes Your Code . . . 255
Implementing Ruby Classes with Java Classes . . . 257
Experiment 10-1: Monitoring JRuby's Just-in-Time Compiler . . . 260
Experiment Code . . . 260
Using the -J-XX:+PrintCompilation Option . . . 261
Does JIT Speed Up Your JRuby Program? . . . 262
Strings in JRuby and MRI . . . 263
How JRuby and MRI Save String Data . . . 264
Copy-on-Write . . . 265
Experiment 10-2: Measuring Copy-on-Write Performance . . . 267
Creating a Unique, Nonshared String . . . 267
Experiment Code . . . 268
Visualizing Copy-on-Write . . . 269
Modifying a Shared String Is Slower . . . 270
Summary . . . 271

11
RUBINIUS: RUBY IMPLEMENTED WITH RUBY **273**

The Rubinius Kernel and Virtual Machine . . . 274
Tokenization and Parsing . . . 276
Using Ruby to Compile Ruby . . . 277
Rubinius Bytecode Instructions . . . 278
Ruby and C++ Working Together . . . 279
Implementing Ruby Objects with C++ Objects . . . 280
Experiment 11-1: Comparing Backtraces in MRI and Rubinius . . . 281
Backtraces in Rubinius . . . 282
Arrays in Rubinius and MRI . . . 284
Arrays Inside of MRI . . . 285
The RArray C Structure Definition . . . 286
Arrays Inside of Rubinius . . . 286
Experiment 11-2: Exploring the Rubinius Implementation of Array#shift . . . 288
Reading Array#shift . . . 288
Modifying Array#shift . . . 289
Summary . . . 292

12
GARBAGE COLLECTION IN MRI, JRUBY, AND RUBINIUS **295**

Garbage Collectors Solve Three Problems . . . 297
Garbage Collection in MRI: Mark and Sweep . . . 297
The Free List . . . 297
MRI's Use of Multiple Free Lists . . . 298
Marking . . . 299
How Does MRI Mark Live Objects? . . . 299

Sweeping . . . 300
Lazy Sweeping . . . 300
The RVALUE Structure . . . 301
Disadvantages of Mark and Sweep . . . 302
Experiment 12-1: Seeing MRI Garbage Collection in Action . . . 302
Seeing MRI Perform a Lazy Sweep . . . 303
Seeing MRI Perform a Full Collection . . . 304
Interpreting a GC Profile Report . . . 305
Garbage Collection in JRuby and Rubinius . . . 309
Copying Garbage Collection . . . 309
Bump Allocation . . . 310
The Semi-Space Algorithm . . . 311
The Eden Heap . . . 312
Generational Garbage Collection . . . 313
The Weak Generational Hypothesis . . . 313
Using the Semi-Space Algorithm for Young Objects . . . 314
Promoting Objects . . . 314
Garbage Collection for Mature Objects . . . 315
References Between Generations . . . 316
Concurrent Garbage Collection . . . 317
Marking While the Object Graph Changes . . . 317
Tricolor Marking . . . 319
Three Garbage Collectors in the JVM . . . 320
Experiment 12-2: Using Verbose GC Mode in JRuby . . . 321
Triggering Major Collections . . . 323
Further Reading . . . 324
Summary . . . 325

INDEX 327