Contents

Preface **ix**

1 Introduction **1**
1.1 Programming Languages 1
1.2 Syntax, Semantics, and Pragmatics 2
1.3 Goals 5
1.4 PostFix: A Simple Stack Language 5
1.4.1 Syntax 6
1.4.2 Semantics 7
1.4.3 The Pitfalls of Informal Descriptions 12

2 Syntax **17**
2.1 Abstract Syntax 18
2.2 Concrete Syntax 20
2.3 S-Expression Grammars Specify ASTs 21
2.3.1 S-Expressions 22
2.3.2 The Structure of S-Expression Grammars 23
2.3.3 Phrase Tags 28
2.3.4 Sequence Patterns 28
2.3.5 Notational Conventions 30
2.4 The Syntax of PostFix 32

3 Operational Semantics **37**
3.1 The Operational Semantics Game 37
3.2 Small-step Operational Semantics (SOS) 41
3.2.1 Formal Framework 41
3.2.2 Example: An SOS for PostFix 44
3.2.3 Rewrite Rules 46
3.3 Big-step Operational Semantics 66
3.4 Operational Reasoning 72

Draft November 23, 200

3.4.1 Programming Language Properties 72
3.4.2 Deterministic Behavior of EL 73
3.4.3 Termination of PostFix Programs 77
3.4.4 Safe PostFix Transformations 82
3.5 Extending PostFix 92

4 Denotational Semantics 107
4.1 The Denotational Semantics Game 107
4.2 A Denotational Semantics for EL 110
4.2.1 Step 1: Restricted ELMM 111
4.2.2 Step 2: Full ELMM 113
4.2.3 Step 3: ELM 118
4.2.4 Step 4: EL 120
4.2.5 A Denotational Semantics is Not a Program 122
4.3 A Denotational Semantics for PostFix 124
4.3.1 A Semantic Algebra for PostFix 125
4.3.2 A Meaning Function for PostFix 128
4.3.3 Semantic Functions for PostFix: the Details 135
4.4 Denotational Reasoning 139
4.4.1 Program Equality 139
4.4.2 Safe Transformations: A Denotational Approach 140
4.4.3 Technical Difficulties 143
4.4.4 Relating Operational and Denotational Semantics 144
4.4.5 Operational vs. Denotational: A Comparison 152

5 Fixed Points 155
5.1 The Fixed Point Game 155
5.1.1 Recursive Definitions 155
5.1.2 Fixed Points 158
5.1.3 The Iterative Fixed Point Technique 160
5.2 Fixed Point Machinery 166
5.2.1 Partial Orders 166
5.2.2 Complete Partial Orders (CPOs) 174
5.2.3 Pointedness 176
5.2.4 Monotonicity and Continuity 178
5.2.5 The Least Fixed Point Theorem 182
5.2.6 Fixed Point Examples 183
5.2.7 Continuity and Strictness 188
5.3 Reflexive Domains 192
5.4 Summary 193

Draft November 23, 200

6 FL: A Functional Language 195
6.1 Decomposing Language Descriptions . . . 195
6.2 The Structure of FL . . . 196
6.2.1 FLK: The Kernel of the FL Language . . . 197
6.2.2 FL Syntactic Sugar . . . 204
6.2.3 The FL Standard Library . . . 216
6.2.4 Examples . . . 216
6.3 Variables and Substitution . . . 224
6.3.1 Terminology . . . 224
6.3.2 General Properties of Variables . . . 227
6.3.3 Abstract Syntax DAGs and Stoy Diagrams . . . 229
6.3.4 Alpha-Equivalence . . . 232
6.3.5 Renaming and Variable Capture . . . 233
6.3.6 Substitution . . . 234
6.4 An Operational Semantics for FLK . . . 239
6.4.1 An SOS for FLK . . . 239
6.4.2 Example . . . 242
6.5 A Denotational Definition for FLK . . . 248

7 Naming 257
7.1 Parameter Passing . . . 259
7.1.1 Call-by-Name and Call-by-Value: The Operational View . 259
7.1.2 Call-by-Name and Call-by-Value:
The Denotational View . . . 267
7.1.3 Discussion . . . 269
7.2 Name Control . . . 279
7.2.1 Hierarchical Scoping: Static and Dynamic . . . 281
7.2.2 Multiple Namespaces . . . 294
7.2.3 Non-hierarchical Scope . . . 296
7.3 Object-Oriented Programming . . . 302
7.3.1 Semantics of HOOK . . . 305

8 State 313
8.1 What is State? . . . 313
8.1.1 Time, State, Identity, and Change . . . 313
8.1.2 FL Does Not Support State . . . 314
8.1.3 Simulating State In FL . . . 320
8.1.4 Imperative Programming . . . 326
8.2 Mutable Data: FL! . . . 327
8.2.1 Mutable Cells . . . 327

Draft November 23, 2004 -- Draft November 23, 2004 -- D

8.2.2 Examples of Imperative Programming 329
8.2.3 An Operational Semantics for FLK! 332
8.2.4 A Denotational Semantics for FLK! 341
8.2.5 Referential Transparency, Interference, and Purity 349
8.3 Mutable Variables: FLAVAR! 356
8.3.1 Mutable Variables 356
8.3.2 FLAVAR! 357
8.3.3 Parameter Passing Mechanisms for FLAVAR! 359

9 Control 365
9.1 Motivation: Control Contexts and Continuations 365
9.2 Using Procedures to Model Control 368
9.2.1 Multiple-value Returns 369
9.2.2 Non-local Exits 371
9.2.3 Coroutines 378
9.3 A Standard Semantics of FL! 378
9.4 Non-local Exits 395
9.5 Exception Handling 402

10 Data 417
10.1 Products 418
10.1.1 Positional Products 419
10.1.2 Named Products 426
10.1.3 Non-strict Products 428
10.1.4 Mutable Products 436
10.2 Sums 442
10.2.1 Positional Sums 443
10.2.2 Named Sums 447
10.3 Sum-of-Products 449
10.4 Data Declarations 456
10.5 Pattern Matching 464
10.5.1 Introduction to Pattern Matching 464
10.5.2 A Desugaring-based Semantics of `match` 468
10.5.3 Views 480

11 Concurrency 489
11.1 Motivation 489
11.2 Threads 492
11.2.1 MUFL!, a Multi-threaded Language 493
11.2.2 An Operational Semantics for MUFL! 495

Draft November 23, 2004 -- Draft November 23, 2004 -- D

11.2.3 Other Thread Interfaces . . . 498
11.3 Communication and Synchronization . . . 503
11.3.1 Shared Mutable Data . . . 504
11.3.2 Locks . . . 505
11.3.3 Channels . . . 507

12 Simple Types 513
12.1 Static Semantics . . . 513
12.2 An Introduction to Types . . . 515
12.2.1 What is a Type? . . . 515
12.2.2 Dimensions of Types . . . 516
12.2.3 Explicit vs. Implicit . . . 518
12.2.4 Simple vs. Expressive . . . 519
12.3 FL/X: A Language with Monomorphic Types . . . 519
12.3.1 FL/X . . . 519
12.3.2 FL/X Type Checking . . . 525
12.3.3 FL/X Dynamic Semantics and Type Soundness . . . 533
12.4 Typed Data . . . 536
12.4.1 Typed Products . . . 536
12.4.2 Digression: Type Equality . . . 538
12.4.3 Typed Mutable Data . . . 539
12.4.4 Typed Sums . . . 542
12.4.5 Typed Lists . . . 543
12.5 Recursive Types . . . 546

13 Subtyping and Polymorphism 551
13.1 Subtyping . . . 551
13.1.1 Motivation . . . 551
13.1.2 FL/XS . . . 552
13.1.3 Discussion . . . 554
13.2 Polymorphic Types . . . 561
13.3 Descriptions . . . 567
13.4 Kinds and Kind Checking: FL/XSPDK . . . 576

14 Type Reconstruction 583
14.1 Introduction . . . 583
14.2 A Language with Type Reconstruction: FL/R . . . 586
14.3 Unification . . . 590
14.4 A Type Reconstruction Algorithm . . . 591
14.5 Discussion . . . 592

Draft November 23, 2004 -- Draft November 23, 2004 -- D

15 Abstract Types **599**
15.1 Data Abstraction 599
15.1.1 A Point Abstraction 600
15.1.2 Procedural Abstraction is not Enough 601
15.2 Dynamic Locks and Keys 603
15.3 Nonce Types 619
15.4 Dependent Types 628
15.4.1 A Dependent Package System 629
15.4.2 Design Issues with Dependent Types 632
15.5 Modules 636
15.5.1 An Overview of Modules and Linking 636
15.5.2 A First-Class Module System 638

16 Effects Describe Program Behavior **653**
16.1 Types, Effects, and Regions - What, How, and Where 653
16.2 An Effect System for FL/R 657
16.3 Using Effects to Analyze Program Behavior 660
16.3.1 Effect Masking Hides Invisible Effects 660
16.3.2 Effects Describe the Actions of Applets 662
16.3.3 Effects Describe Control Transfers 663
16.3.4 Effects Can Be Used to Deallocate Storage 664
16.4 Reconstructing Types and Effects 665

17 Compilation **673**
17.1 Why do we study compilation? 673
17.2 TORTOISE Architecture and Languages 675
17.2.1 Overview of TORTOISE 675
17.2.2 The Compiler Source Language: FL/R$_{\text{TORTOISE}}$ 677
17.2.3 The Compiler Intermediate Language: SILK 678
17.2.4 Purely Structural Transformations 694
17.3 Transform 1: Desugaring 697
17.4 Transform 2: Type Reconstruction 698
17.5 Transform 3: Globalization 699
17.6 Transform 4: Translation 705
17.7 Transform 5: Assignment Conversion 708
17.8 Transform 6: Renaming 714
17.9 Transform 7: CPS Conversion 718
17.9.1 The Structure of CPS Code 720
17.9.2 A Simple CPS Transform 725
17.9.3 A More Efficient CPS Transform 734

Draft November 23, 2004 -- Draft November 23, 2004 -- D

17.9.4 CPS Converting Control Constructs 745
17.10 Transform 8: Closure Conversion 748
17.10.1 Flat Closures 748
17.10.2 Variations on Flat Closure Conversion 756
17.10.3 Linked Approaches 759
17.11 Transform 9: Lifting 763
17.12 Transform 10: Data Conversion 765
17.13 Garbage Collection 765

A A Metalanguage **769**
A.1 The Basics 769
A.1.1 Sets 770
A.1.2 Tuples 773
A.1.3 Relations 774
A.2 Functions 775
A.2.1 Definition 775
A.2.2 Application 777
A.2.3 More Function Terminology 779
A.2.4 Higher-Order Functions 780
A.2.5 Multiple Arguments and Results 781
A.2.6 Lambda Notation 784
A.2.7 Recursion 787
A.2.8 Lambda Notation is not Lisp! 788
A.3 Domains 790
A.3.1 Motivation 790
A.3.2 Product Domains 791
A.3.3 Sum Domains 793
A.3.4 Sequence Domains 796
A.3.5 Function Domains 798
A.4 Metalanguage Summary 802
A.4.1 The Metalanguage Kernel 802
A.4.2 The Metalanguage Sugar 804

Draft November 23, 2004 -- Draft November 23, 2004 -- D

Draft November 23, 2004 -- Draft November 23, 2004 -- D