Contents

List of Figures **xi**

Preface **xv**

I Type Problems in Object-Oriented Languages 1

1 *Introduction* 3

1.1 Type systems in programming languages 4
1.2 Type checking and strongly typed languages 6
1.3 Focus on statically typed class-based languages 12
1.4 Foundations: A look ahead 13

2 *Fundamental Concepts of Object-Oriented Languages* 17

2.1 Objects, classes, and object types 17
2.2 Subclasses and inheritance 22
2.3 Subtypes 24
2.4 Covariant and contravariant changes in types 26
2.5 Overloading versus overriding methods 27
2.6 Summary 32

3 *Type Problems in Object-Oriented Languages* 33

3.1 Type checking object-oriented languages is difficult 33
3.2 Simple type systems are lacking in flexibility 35
3.3 Summary of typing problems 48

4 *Adding Expressiveness to Object-Oriented Languages* 49

4.1 GJ 49
4.2 Even more flexible typing with Eiffel 60
4.3 Summary 69

5 *Understanding Subtypes* 71

5.1 Subtyping for non-object types 72
5.2 Object types 83
5.3 Subtyping for class types 84
5.4 Summary 86

6 *Type Restrictions on Subclasses* 89

6.1 Allowable changes to method types 89
6.2 Instance variable types invariant in subclasses 91
6.3 Changing visibility 92
6.4 Summary 93

7 *Varieties of Object-Oriented Programming Languages* 95

7.1 Multi-methods vs. object-based vs. class-based languages 95
7.2 Well-known object-oriented languages 103
7.3 Summary 111

***Historical Notes and References for Section I* 113**

**II Foundations:
The Lambda Calculus 117**

8 *Formal Language Descriptions and the Lambda Calculus* 119

8.1 The simply-typed lambda calculus 120
8.2 Adding pairs, sums, records, and references 132
8.3 Summary 140

9 *The Polymorphic Lambda Calculus* 141

9.1 Parameterized types and polymorphism 141
9.2 Recursive expressions and types 147
9.3 Information hiding and existential types 151
9.4 Adding subtypes to the polymorphic lambda calculus 156
9.5 Summary 165

***Historical Notes and References for Section II* 167**

III Formal Descriptions of Object-Oriented Languages 171

10 *$\mathcal{SOOL}$, a Simple Object-Oriented Language* 173

10.1 Informal description and example 173
10.2 Syntax and type-checking rules 176
10.3 Summary 200

11 *A Simple Translational Semantics of Objects and Classes* 201

11.1 Representing objects at runtime 201
11.2 Modeling $\mathcal{SOOL}$ types in $\Lambda^{P}_{<:}$ 203
11.3 Modeling $\mathcal{SOOL}$ expressions in $\Lambda^{P}_{<:}$ 207
11.4 Modeling classes — first try 212
11.5 Problems with modeling subclasses 218
11.6 Summary 223

12 *Improved Semantics for Classes* 225

12.1 (Re-)Defining classes 225
12.2 A correct subclass encoding 232
12.3 Summary and a look ahead 233

13 *$\mathcal{SOOL}$'s Type System Is Safe (and Sound)* 239

13.1 The translation of $\mathcal{SOOL}$ to $\Lambda^{P}_{<:}$ is sound 239
13.2 The translation is well defined 255
13.3 $\mathcal{SOOL}$ is type safe 258
13.4 Errors 260
13.5 Summary 262

14 *Completing $\mathcal{SOOL}$: `super`, `nil`, Information Hiding, and Multiple Inheritance* 263

14.1 Using methods from superclasses 263
14.2 Translating `nil` 266
14.3 A complication with `self` 271
14.4 Finer control over information hiding 272
14.5 Multiple inheritance 275
14.6 Summary 279

***Historical Notes and References for Section III* 283**

IV Extending Simple Object-Oriented Languages 289

15 *Adding Bounded Polymorphism to $\mathcal{SOOL}$* 291

15.1 Introducing $\mathcal{PSOOL}$ 291
15.2 Translational semantics of $\mathcal{PSOOL}$ 296
15.3 Summary 297

16 *Adding `MyType` to Object-Oriented Programming Languages* 299

16.1 Typing `self` with `MyType` 300
16.2 $\mathcal{MOOL}$: Adding `MyType` to $\mathcal{SOOL}$ 309
16.3 Translational semantics of $\mathcal{MOOL}$ 319
16.4 Soundness of translation for $\mathcal{MOOL}$ 322
16.5 Summary 330

17 *Match-Bounded Polymorphism* 331

17.1 Benefits of match-bounded polymorphism 331
17.2 Introducing $\mathcal{PMOOL}$ 333
17.3 Examples and comparison with F-bounded polymorphism 334
17.4 Translational semantics of $\mathcal{PMOOL}$ 335
17.5 Soundness of the translation of $\mathcal{PMOOL}$ 342
17.6 Summary 347

18 *Simplifying: Dropping Subtyping for Matching* 349

18.1 Can we drop subtyping? 349
18.2 Introducing hash types 352
18.3 Type-checking rules 356
18.4 An informal semantics of hash types 360
18.5 Summary 361

***Historical Notes and References for Section IV* 363**

***Bibliography* 367**

***Index* 379**