

# Contents

<b>Preface</b>	<b>ix</b>
<b>Part One</b>	<b>3</b>
<b>1 Category: The Essence of Composition</b>	<b>3</b>
1.1 Arrows as Functions . . . . .	3
1.2 Properties of Composition . . . . .	5
1.3 Composition is the Essence of Programming . . . . .	7
1.4 Challenges . . . . .	8
<b>2 Types and Functions</b>	<b>9</b>
2.1 Who Needs Types? . . . . .	9
2.2 Types Are About Composability . . . . .	10
2.3 What Are Types? . . . . .	11
2.4 Why Do We Need a Mathematical Model? . . . . .	12
2.5 Pure and Dirty Functions . . . . .	14
2.6 Examples of Types . . . . .	15
2.7 Challenges . . . . .	17
<b>3 Categories Great and Small</b>	<b>19</b>
3.1 No Objects . . . . .	19
3.2 Simple Graphs . . . . .	19
3.3 Orders . . . . .	20
3.4 Monoid as Set . . . . .	20
3.5 Monoid as Category . . . . .	23
3.6 Challenges . . . . .	25
<b>4 Kleisli Categories</b>	<b>27</b>
4.1 The Writer Category . . . . .	30
4.2 Writer in Haskell . . . . .	32
4.3 Kleisli Categories . . . . .	34
4.4 Challenge . . . . .	34

<b>5</b>	<b>Products and Coproducts</b>	<b>37</b>
5.1	Initial Object . . . . .	37
5.2	Terminal Object . . . . .	39
5.3	Duality . . . . .	40
5.4	Isomorphisms . . . . .	40
5.5	Products . . . . .	42
5.6	Coproduct . . . . .	45
5.7	Asymmetry . . . . .	48
5.8	Challenges . . . . .	49
5.9	Bibliography . . . . .	50
<b>6</b>	<b>Simple Algebraic Data Types</b>	<b>51</b>
6.1	Product Types . . . . .	51
6.2	Records . . . . .	54
6.3	Sum Types . . . . .	55
6.4	Algebra of Types . . . . .	58
6.5	Challenges . . . . .	61
<b>7</b>	<b>Functors</b>	<b>63</b>
7.1	Functors in Programming . . . . .	65
7.1.1	The Maybe Functor . . . . .	65
7.1.2	Equational Reasoning . . . . .	66
7.1.3	Optional . . . . .	68
7.1.4	Typeclasses . . . . .	69
7.1.5	Functor in C++ . . . . .	70
7.1.6	The List Functor . . . . .	71
7.1.7	The Reader Functor . . . . .	72
7.2	Functors as Containers . . . . .	74
7.3	Functor Composition . . . . .	75
7.4	Challenges . . . . .	77
<b>8</b>	<b>Functoriality</b>	<b>79</b>
8.1	Bifunctors . . . . .	79
8.2	Product and Coproduct Bifunctors . . . . .	81
8.3	Functorial Algebraic Data Types . . . . .	82
8.4	Functors in C++ . . . . .	85
8.5	The Writer Functor . . . . .	86
8.6	Covariant and Contravariant Functors . . . . .	87
8.7	Profunctors . . . . .	89
8.8	The Hom-Functor . . . . .	90
8.9	Challenges . . . . .	91

<b>9</b>	<b>Function Types</b>	<b>93</b>
9.1	Universal Construction . . . . .	94
9.2	Currying . . . . .	97
9.3	Exponentials . . . . .	100
9.4	Cartesian Closed Categories . . . . .	101
9.5	Exponentials and Algebraic Data Types . . . . .	101
9.5.1	Zeroth Power . . . . .	101
9.5.2	Powers of One . . . . .	102
9.5.3	First Power . . . . .	102
9.5.4	Exponentials of Sums . . . . .	102
9.5.5	Exponentials of Exponentials . . . . .	103
9.5.6	Exponentials over Products . . . . .	103
9.6	Curry-Howard Isomorphism . . . . .	103
9.7	Bibliography . . . . .	105
<b>10</b>	<b>Natural Transformations</b>	<b>107</b>
10.1	Polymorphic Functions . . . . .	110
10.2	Beyond Naturality . . . . .	114
10.3	Functor Category . . . . .	115
10.4	2-Categories . . . . .	118
10.5	Conclusion . . . . .	121
10.6	Challenges . . . . .	121
	<b>Part Two</b>	<b>125</b>
<b>11</b>	<b>Declarative Programming</b>	<b>125</b>
<b>12</b>	<b>Limits and Colimits</b>	<b>131</b>
12.1	Limit as a Natural Isomorphism . . . . .	135
12.2	Examples of Limits . . . . .	138
12.3	Colimits . . . . .	142
12.4	Continuity . . . . .	143
12.5	Challenges . . . . .	145
<b>13</b>	<b>Free Monoids</b>	<b>147</b>
13.1	Free Monoid in Haskell . . . . .	148
13.2	Free Monoid Universal Construction . . . . .	149
13.3	Challenges . . . . .	152
<b>14</b>	<b>Representable Functors</b>	<b>153</b>
14.1	The Hom Functor . . . . .	154
14.2	Representable Functors . . . . .	156
14.3	Challenges . . . . .	159
14.4	Bibliography . . . . .	159

<b>15 The Yoneda Lemma</b>	<b>161</b>
15.1 Yoneda in Haskell . . . . .	165
15.2 Co-Yoneda . . . . .	167
15.3 Challenges . . . . .	167
15.4 Bibliography . . . . .	168
<b>16 Yoneda Embedding</b>	<b>169</b>
16.1 The Embedding . . . . .	170
16.2 Application to Haskell . . . . .	171
16.3 Preorder Example . . . . .	172
16.4 Naturality . . . . .	173
16.5 Challenges . . . . .	174
<b>Part Three</b>	<b>177</b>
<b>17 It's All About Morphisms</b>	<b>177</b>
17.1 Functors . . . . .	177
17.2 Commuting Diagrams . . . . .	177
17.3 Natural Transformations . . . . .	178
17.4 Natural Isomorphisms . . . . .	179
17.5 Hom-Sets . . . . .	180
17.6 Hom-Set Isomorphisms . . . . .	180
17.7 Asymmetry of Hom-Sets . . . . .	181
17.8 Challenges . . . . .	181
<b>18 Adjunctions</b>	<b>183</b>
18.1 Adjunction and Unit/Counit Pair . . . . .	183
18.2 Adjunctions and Hom-Sets . . . . .	187
18.3 Product from Adjunction . . . . .	190
18.4 Exponential from Adjunction . . . . .	193
18.5 Challenges . . . . .	194
<b>19 Free/Forgetful Adjunctions</b>	<b>195</b>
19.1 Some Intuitions . . . . .	197
19.2 Challenges . . . . .	199
<b>20 Monads: Programmer's Definition</b>	<b>201</b>
20.1 The Kleisli Category . . . . .	202
20.2 Fish Anatomy . . . . .	204
20.3 The do Notation . . . . .	205

<b>21 Monads and Effects</b>	<b>209</b>
21.1 The Problem . . . . .	209
21.2 The Solution . . . . .	210
21.2.1 Partiality . . . . .	210
21.2.2 Nondeterminism . . . . .	211
21.2.3 Read-Only State . . . . .	212
21.2.4 Write-Only State . . . . .	214
21.2.5 State . . . . .	214
21.2.6 Exceptions . . . . .	215
21.2.7 Continuations . . . . .	215
21.2.8 Interactive Input . . . . .	217
21.2.9 Interactive Output . . . . .	218
21.3 Conclusion . . . . .	219
<b>22 Monads Categorically</b>	<b>221</b>
22.1 Monoidal Categories . . . . .	224
22.2 Monoid in a Monoidal Category . . . . .	228
22.3 Monads as Monoids . . . . .	229
22.4 Monads from Adjunctions . . . . .	230
<b>23 Comonads</b>	<b>233</b>
23.1 Programming with Comonads . . . . .	234
23.2 The Product Comonad . . . . .	234
23.3 Dissecting the Composition . . . . .	235
23.4 The Stream Comonad . . . . .	236
23.5 Comonad Categorically . . . . .	238
23.6 The Store Comonad . . . . .	240
23.7 Challenges . . . . .	241
<b>24 F-Algebras</b>	<b>243</b>
24.1 Recursion . . . . .	245
24.2 Category of F-Algebras . . . . .	247
24.3 Natural Numbers . . . . .	249
24.4 Catamorphisms . . . . .	250
24.5 Folds . . . . .	251
24.6 Coalgebras . . . . .	252
24.7 Challenges . . . . .	254
<b>25 Algebras for Monads</b>	<b>257</b>
25.1 T-algebras . . . . .	259
25.2 The Kleisli Category . . . . .	261
25.3 Coalgebras for Comonads . . . . .	263
25.4 Lenses . . . . .	263
25.5 Challenges . . . . .	265

<b>26 Ends and Coends</b>	<b>267</b>
26.1 Dinatural Transformations . . . . .	268
26.2 Ends . . . . .	269
26.3 Ends as Equalizers . . . . .	272
26.4 Natural Transformations as Ends . . . . .	272
26.5 Coends . . . . .	273
26.6 Ninja Yoneda Lemma . . . . .	276
26.7 Profunctor Composition . . . . .	277
<b>27 Kan Extensions</b>	<b>279</b>
27.1 Right Kan Extension . . . . .	281
27.2 Kan Extension as Adjunction . . . . .	282
27.3 Left Kan Extension . . . . .	284
27.4 Kan Extensions as Ends . . . . .	286
27.5 Kan Extensions in Haskell . . . . .	288
27.6 Free Functor . . . . .	290
<b>28 Enriched Categories</b>	<b>293</b>
28.1 Why Monoidal Category? . . . . .	294
28.2 Monoidal Category . . . . .	294
28.3 Enriched Category . . . . .	296
28.4 Preorders . . . . .	297
28.5 Metric Spaces . . . . .	298
28.6 Enriched Functors . . . . .	299
28.7 Self Enrichment . . . . .	300
28.8 Relation to 2-Categories . . . . .	301
<b>29 Topoi</b>	<b>303</b>
29.1 Subobject Classifier . . . . .	304
29.2 Topos . . . . .	307
29.3 Topoi and Logic . . . . .	307
29.4 Challenges . . . . .	308
<b>30 Lawvere Theories</b>	<b>309</b>
30.1 Universal Algebra . . . . .	309
30.2 Lawvere Theories . . . . .	310
30.3 Models of Lawvere Theories . . . . .	313
30.4 The Theory of Monoids . . . . .	314
30.5 Lawvere Theories and Monads . . . . .	315
30.6 Monads as Coends . . . . .	317
30.7 Lawvere Theory of Side Effects . . . . .	319
30.8 Challenges . . . . .	320
30.9 Further Reading . . . . .	320

<b>31 Monads, Monoids, and Categories</b>	<b>323</b>
31.1 Bicategories . . . . .	323
31.2 Monads . . . . .	327
31.3 Challenges . . . . .	330
31.4 Bibliography . . . . .	330

<b>Preface</b>	<b>331</b>
<b>Appendices</b>	<b>331</b>
<b>Index</b>	<b>331</b>
<b>Acknowledgments</b>	<b>333</b>
<b>Colophon</b>	<b>334</b>
<b>Copyright notice</b>	<b>335</b>

I will attempt, in the space of a few paragraphs, to convince you that this book is written for you, and whatever objections you might have to learning one of the most abstract branches of mathematics in your "spare spare time" are totally unfounded.

My optimism is based on several observations. First, category theory is a treasure trove of extremely useful programming ideas. Haskell programmers have been tapping this resource for a long time, and the ideas are slowly percolating into other languages; but this process is too slow. We need to speed it up.

Second, there are many different kinds of math, and they appeal to different audiences. You might be allergic to calculus or algebra, but it doesn't mean you won't enjoy category theory. I would go as far as to argue that category theory is the kind of math that is particularly well suited for the needs of programmers. That's because category theory — rather than dealing with particulars — deals with structure. It deals with the kind of structure that makes programs composable.

Composition is at the very root of category theory — it's part of the definition of the category itself. And I will argue strongly that composition is the essence of programming. We've been composing things forever, long before your great engineer came up with the idea of a subroutine. Some time ago the principles of

---

we regularly send the truth this world is a few sentences at <http://www.gutenberg.org> or search "category theory" on YouTube.