

Contents

Preface	1
1 Functional Programming with the Booleans	13
1.1 Declaring the Datatype of Booleans	13
1.1.1 Aside: type levels	15
1.1.2 Constructors <code>tt</code> and <code>ff</code>	15
1.1.3 Aside: some compiler directives	15
1.2 First steps interacting with Agda	16
1.3 Syntax declarations	16
1.4 Defining boolean operations by pattern matching: negation	18
1.4.1 Aside: space around operators and in files	19
1.5 Defining boolean operations by pattern matching: and, or	20
1.6 The if-then-else operation	23
1.6.1 Some examples with if-then-else	24
1.7 Conclusion	25
1.8 Exercises	26
2 Introduction to Constructive Proof	27
2.1 A First Theorem about the Booleans	27
2.1.1 Types as formulas: the Curry-Howard isomorphism	28
2.1.2 Proofs as programs: more Curry-Howard	29
2.2 Universal Theorems	30
2.2.1 Proving the theorem using pattern matching	31
2.2.2 An alternative proof and implicit arguments	32
2.2.3 Using holes in proofs	33
2.3 Another Example, and More On Implicit Arguments	34
2.4 Theorems with Hypotheses	37
2.4.1 The absurd pattern	38
2.4.2 An alternative proof	39
2.4.3 Matching on equality proofs	41
2.4.4 The <code>rewrite</code> directive	42
2.5 Going deeper: Curry-Howard and constructivity	43
2.6 Further examples	47
2.7 Conclusion	48

2.8	Exercises	49
3	Natural Numbers	51
3.1	Peano Natural Numbers	52
3.2	Addition	53
3.2.1	Simplest theorems about addition	55
3.2.2	Associativity of addition and working with holes	59
3.2.3	Commutativity of addition and helper lemmas	62
3.3	Multiplication	64
3.3.1	Distributivity of multiplication and choosing the induction variable	65
3.3.2	Commutativity of multiplication	67
3.3.3	Associativity of multiplication	68
3.4	Arithmetic Comparison	69
3.4.1	Dotted variables	72
3.4.2	An equality test	73
3.5	Even/odd and mutually recursive definitions	75
3.6	Conclusion	76
3.7	Exercises	76
4	Lists	79
4.1	The List Datatype and Type Parameters	79
4.2	Basic Operations on Lists	81
4.2.1	Length of a list	81
4.2.2	Appending two lists	81
4.2.3	Mapping a function over a list	83
4.2.4	Filtering a list	84
4.2.5	Removing an element from a list, and lambda abstractions	85
4.2.6	Selecting the n 'th element from a list, and the <code>maybe</code> type	86
4.2.7	Reversing a list (easy but inefficient way)	87
4.2.8	Efficiently reversing a list	87
4.3	Reasoning about List Operations	89
4.3.1	Distributing length over append	89
4.3.2	Associativity of list append	90
4.3.3	Length of filtered lists, and the <code>with</code> construct	91
4.3.4	Filter is idempotent, and the <code>keep</code> idiom	95
4.3.5	Reverse preserves list length	98
4.4	Conclusion	100
4.5	Exercises	100
5	Internal Verification	103
5.1	Vectors	103
5.1.1	The Vector Datatype	104
5.1.2	Appending vectors	105
5.1.3	Head and tail operations on vectors	106
5.1.4	Using types to express properties of other vector operations	107

5.2	Binary Search Trees	110
5.2.1	The <code>bst</code> module, and <code>bool-relations</code>	110
5.2.2	The <code>bst</code> datatype	114
5.2.3	Searching for an element in a binary search tree	115
5.2.4	Inserting an element into a binary search tree	116
5.3	Sigma Types	117
5.3.1	Why Sigma and Pi?	121
5.4	Braun Trees	122
5.4.1	The <code>braun-tree</code> datatype, and sum types	122
5.4.2	Inserting into a Braun tree	124
5.4.3	Removing the minimum element from a Braun tree	127
5.5	Discussion: Internal vs. External Verification	129
5.6	Conclusion	132
5.7	Exercises	132
6	Type-level Computation	135
6.1	Integers	135
6.1.1	The \mathbb{Z} datatype	136
6.1.2	Addition on integers	137
6.1.3	Antisymmetric integer comparison	139
6.2	Formatted Printing	139
6.2.1	An unsuccessful attempt	140
6.2.2	A working solution	142
6.3	Proof by Reflection	146
6.3.1	The datatype of representations, and its semantics	147
6.3.2	The list simplifier	148
6.3.3	Proving that the simplifier preserves the semantics	151
6.4	Conclusion	152
6.5	Exercises	153
7	Generating Agda Parsers with <code>gratr</code>	155
7.1	A Primer on Grammars	156
7.1.1	Derivations	157
7.1.2	From derivations to syntax trees	158
7.1.3	Regular-expression operators for grammars	162
7.2	Generating Parsers with <code>gratr</code>	162
7.2.1	Compiling the emitted parsers	164
7.2.2	Running the emitted executable	165
7.2.3	Modifying the emitted code to process parse trees	166
7.2.4	Reorganizing rules to resolve ambiguity	167
7.3	Conclusion	170
7.4	Exercises	170
8	A Case Study: Huffman Encoding and Decoding	173
8.1	The Files	174
8.2	The Input Formats	175

8.3	Encoding Textual Input	175
8.3.1	Computing word frequencies using tries	177
8.3.2	Initializing a priority queue with Huffman leaves	179
8.3.3	Processing the priority queue to compute a Huffman tree	182
8.3.4	Computing the code from the Huffman tree	183
8.3.5	Encoding the input words using the computed code	183
8.4	Decoding Encoded Text	185
8.4.1	Code trees	185
8.4.2	Computing the code tree	185
8.4.3	Decoding the input	187
8.5	Conclusion	188
8.6	Exercises	188
9	Reasoning about Termination	189
9.1	Termination Proofs	189
9.1.1	Defining termination in Agda	190
9.1.2	Termination of greater-than on natural-numbers	192
9.1.3	Proving termination using a measure function	193
9.1.4	Using termination proofs for well-founded recursion	195
9.1.5	Nontermination with negative datatypes	197
9.2	Operational Semantics for SK Combinators	198
9.2.1	Mathematical syntax and semantics for combinators	199
9.2.2	Defining the syntax and semantics in Agda	201
9.2.3	Termination for S-free combinators	203
9.2.4	Defining lambda abstractions using combinators	207
9.3	Conclusion	213
9.4	Exercises	215
10	Intuitionistic Logic and Kripke Semantics	217
10.1	Positive Propositional Intuitionistic Logic (PPIL)	218
10.2	Kripke structures	222
10.3	Kripke semantics for PPIL	225
10.4	Soundness of PPIL	229
10.4.1	Monotonicity proof	230
10.4.2	Soundness proof	232
10.5	Completeness	234
10.5.1	A universal structure	234
10.5.2	Completeness and soundness with respect to \mathcal{U}	236
10.5.3	Concluding completeness and universality	240
10.5.4	Composing soundness and completeness	241
10.6	Conclusion	247
10.7	Exercises	247
	Quick Guide to Symbols	249
	Commonly Used Keyboard Commands	251

Some Extra Emacs Definitions	253
References	255
Index	259