

Contents

Acknowledgments xi

1 Introduction 1

- 1.1 Whence This Book? 1
- 1.2 Why Coq? 3
 - 1.2.1 Based on a Higher-Order Functional Programming Language 3
 - 1.2.2 Dependent Types 3
 - 1.2.3 An Easy-to-Check Kernel Proof Language 4
 - 1.2.4 Convenient Programmable Proof Automation 5
 - 1.2.5 Proof by Reflection 6
- 1.3 Why Not a Different Dependently Typed Language? 6
- 1.4 Engineering with a Proof Assistant 7
- 1.5 Prerequisites 8
- 1.6 Using This Book 8
 - 1.6.1 Reading This Book 9
 - 1.6.2 The Tactic Library 10
 - 1.6.3 Installation and Emacs Setup 11

2 Some Quick Examples 13

- 2.1 Arithmetic Expressions over Natural Numbers 13
 - 2.1.1 Source Language 14
 - 2.1.2 Target Language 17
 - 2.1.3 Translation 18
 - 2.1.4 Translation Correctness 19
- 2.2 Typed Expressions 28
 - 2.2.1 Source Language 28
 - 2.2.2 Target Language 31
 - 2.2.3 Translation 33

2.2.4 Translation Correctness 35

I	Basic Programming and Proving	39
3	Introducing Inductive Types	41
3.1	Proof Terms	41
3.2	Enumerations	43
3.3	Simple Recursive Types	47
3.4	Parameterized Types	51
3.5	Mutually Inductive Types	53
3.6	Reflexive Types	56
3.7	An Interlude on Induction Principles	60
3.8	Nested Inductive Types	64
3.9	Manual Proofs about Constructors	69
4	Inductive Predicates	73
4.1	Propositional Logic	74
4.2	What Does It Mean to Be Constructive?	80
4.3	First-Order Logic	81
4.4	Predicates with Implicit Equality	82
4.5	Recursive Predicates	86
5	Infinite Data and Proofs	93
5.1	Computing with Infinite Data	94
5.2	Infinite Proofs	98
5.3	Simple Modeling of Nonterminating Programs	106
II	Programming with Dependent Types	111
6	Subset Types and Variations	113
6.1	Introducing Subset Types	113
6.2	Decidable Proposition Types	121
6.3	Partial Subset Types	124
6.4	Monadic Notations	126
6.5	A Type-Checking Example	127
7	General Recursion	133
7.1	Well-Founded Recursion	134
7.2	A Nontermination Monad Inspired by Domain Theory	140
7.3	Co-inductive Nontermination Monads	147

7.4	Comparing the Alternatives	151
8	More Dependent Types	155
8.1	Length-Indexed Lists	155
8.2	The One Rule of Dependent Pattern Matching in Coq	159
8.3	A Tagless Interpreter	161
8.4	Dependently Typed Red-Black Trees	167
8.5	A Certified Regular Expression Matcher	178
9	Dependent Data Structures	185
9.1	More Length-Indexed Lists	185
9.2	Heterogeneous Lists	189
9.2.1	A Lambda Calculus Interpreter	191
9.3	Recursive Type Definitions	193
9.4	Data Structures as Index Functions	196
9.4.1	Another Interpreter Example	200
9.5	Choosing between Representations	205
10	Reasoning about Equality Proofs	207
10.1	The Definitional Equality	207
10.2	Heterogeneous Lists Revisited	211
10.3	Type Casts in Theorem Statements	217
10.4	Heterogeneous Equality	222
10.5	Equivalence of Equality Axioms	227
10.6	Equality of Functions	229
11	Generic Programming	233
11.1	Reifying Datatype Definitions	233
11.2	Recursive Definitions	236
11.2.1	Pretty-Printing	239
11.2.2	Mapping	242
11.3	Proving Theorems about Recursive Definitions	243
12	Universes and Axioms	251
12.1	The Type Hierarchy	251
12.1.1	Inductive Definitions	255
12.1.2	Deciphering Baffling Messages about Inability to Unify	259
12.2	The Prop Universe	261
12.3	Axioms	265
12.3.1	The Basics	265

12.3.2	Axioms of Choice	270
12.3.3	Axioms and Computation	273
12.3.4	Methods for Avoiding Axioms	275
III	Proof Engineering	285
13	Proof Search by Logic Programming	287
13.1	Introducing Logic Programming	287
13.2	Searching for Underconstrained Values	295
13.3	Synthesizing Programs	298
13.4	More on auto Hints	302
13.5	Rewrite Hints	304
14	Proof Search in Ltac	309
14.1	Some Built-in Automation Tactics	309
14.2	Ltac Programming Basics	310
14.3	Functional Programming in Ltac	318
14.4	Recursive Proof Search	323
14.5	Creating Unification Variables	330
15	Proof by Reflection	339
15.1	Proving Evenness	339
15.2	Reifying the Syntax of a Trivial Tautology Language	342
15.3	A Monoid Expression Simplifier	345
15.4	A Smarter Tautology Solver	348
15.4.1	Manual Reification of Terms with Variables	354
15.5	Building a Reification Tactic That Recurses under Binders	357
IV	The Big Picture	361
16	Proving in the Large	363
16.1	Ltac Antipatterns	363
16.2	Debugging and Maintaining Automation	372
16.3	Modules	380
16.4	Build Processes	384

17 Reasoning about Programming Language Syntax**389**

- 17.1 Dependent de Bruijn Indices 390
- 17.2 Parametric Higher-Order Abstract Syntax 397
 - 17.2.1 Functional Programming with PHOAS 399
 - 17.2.2 Verifying Program Transformations 402
 - 17.2.3 Establishing Term Well-Formedness 408
 - 17.2.4 A Few Additional Remarks 409

Conclusion 411**References 413****Index 419**