

Contents

<i>Preface</i>	<i>page xi</i>
Introduction	1
I.1 Connections between logic and computation	1
I.2 Logical primitives and programming expressivity	3
I.3 The meaning of <i>higher-order logic</i>	5
I.4 Presentation style	6
I.5 Prerequisites	7
I.6 Organization of the book	8
1 First-Order Terms and Representations of Data	10
1.1 Sorts and type constructors	10
1.2 Type expressions	12
1.3 Typed first-order terms	14
1.4 Representing symbolic objects	19
1.5 Unification of typed first-order terms	26
1.6 Bibliographic notes	32
2 First-Order Horn Clauses	34
2.1 First-order formulas	34
2.2 Logic programming and search semantics	38
2.3 Horn clauses and their computational interpretation	43
2.4 Programming with first-order Horn clauses	46
2.5 Pragmatic aspects of computing with Horn clauses	60
2.6 The relationship with logical notions	62
2.7 The meaning and use of types	66
2.8 Bibliographic notes	72

viii	<i>Contents</i>	
3	First-Order Hereditary Harrop Formulas	75
3.1	The syntax of goals and program clauses	75
3.2	Implicational goals	77
3.3	Universally quantified goals	82
3.4	The relationship with logical notions	88
3.5	Bibliographic notes	93
4	Typed λ-Terms and Formulas	96
4.1	Syntax for λ -terms and formulas	97
4.2	The rules of λ -conversion	100
4.3	Some properties of λ -conversion	101
4.4	Unification problems as quantified equalities	105
4.5	Solving unification problems	111
4.6	Some hard unification problems	113
4.7	Bibliographic notes	116
5	Using Quantification at Higher-Order Types	118
5.1	Atomic formulas in higher-order logic programs	118
5.2	Higher-order logic programming languages	122
5.3	Examples of higher-order programming	126
5.4	Flexible atoms as goals	132
5.5	Reasoning about higher-order programs	134
5.6	Defining some of the logical constants	136
5.7	The conditional and negation-as-failure	137
5.8	Using λ -terms as functions	138
5.9	Higher-order unification is not a panacea	143
5.10	Comparison with functional programming	146
5.11	Bibliographic notes	147
6	Mechanisms for Structuring Large Programs	150
6.1	Desiderata for modular programming	150
6.2	A modules language	151
6.3	Matching signatures and modules	156
6.4	The logical interpretation of modules	160
6.5	Some programming aspects of the modules language	165
6.6	Implementation considerations	172
6.7	Bibliographic notes	173
7	Computations over λ-Terms	175
7.1	Representing objects with binding structure	175
7.2	Realizing object-level substitution	180
7.3	Mobility of binders	183
7.4	Computing with untyped λ -terms	185

<i>Contents</i>		ix
7.5	Computations over first-order formulas	195
7.6	Specifying object-level substitution	199
7.7	The λ -tree approach to abstract syntax	203
7.8	The L_λ subset of λ Prolog	204
7.9	Bibliographic notes	208
8	Unification of λ-Terms	211
8.1	Properties of the higher-order unification problem	212
8.2	A procedure for checking for unifiability	214
8.3	Higher-order pattern unification	220
8.4	Pragmatic aspects of higher-order unification	224
8.5	Bibliographic notes	226
9	Implementing Proof Systems	229
9.1	Deduction in propositional intuitionistic logic	229
9.2	Encoding natural deduction for intuitionistic logic	231
9.3	A theorem prover for classical logic	235
9.4	A general architecture for theorem provers	240
9.5	Bibliographic notes	245
10	Computations over Functional Programs	247
10.1	The miniFP programming language	247
10.2	Specifying evaluation for miniFP programs	250
10.3	Manipulating functional programs	255
10.4	Bibliographic notes	258
11	Encoding a Process Calculus Language	261
11.1	Representing the expressions of the π -calculus	261
11.2	Specifying one-step transitions	263
11.3	Animating π -calculus expressions	266
11.4	May- versus must-judgments	269
11.5	Mapping the λ -calculus into the π -calculus	273
11.6	Bibliographic notes	275
Appendix: The Teyjus System		277
A.1	An overview of the Teyjus system	277
A.2	Interacting with the Teyjus system	278
A.3	Using modules within the Teyjus system	283
A.4	Special features of the Teyjus system	285
<i>Bibliography</i>		289
<i>Index</i>		301