

# Table of Contents

|  |    |
|--|----|
| Sorting Morphisms .....  | 1  |
| <i>Lex Augusteijn</i>  |    |
| 1 Introduction .....   | 1  |
| 2 Morphisms on Lists .....   | 2  |
| 2.1 The List Catamorphism .....  | 2  |
| 2.2 The List Anamorphism .....   | 4  |
| 2.3 The List Hylomorphism .....  | 5  |
| 2.4 Insertion Sort .....   | 6  |
| 2.5 Selection Sorts .....  | 7  |
| 3 Leaf Trees .....   | 9  |
| 3.1 The Leaf-Tree Catamorphism .....                                     | 9  |
| 3.2 The Leaf-Tree Anamorphism .....                                      | 10 |
| 3.3 The Leaf-Tree Hylomorphism .....                                     | 11 |
| 3.4 Merge Sort .....   | 12 |
| 4 Binary Trees .....   | 13 |
| 4.1 The Tree Catamorphism .....  | 13 |
| 4.2 The Tree Anamorphism .....   | 14 |
| 4.3 The Tree Hylomorphism .....  | 14 |
| 4.4 Quicksort .....  | 15 |
| 4.5 Heap Sort .....  | 16 |
| 5 Paramorphisms .....  | 18 |
| 5.1 The List Paramorphism .....  | 18 |
| 5.2 Insert As Paramorphism .....   | 18 |
| 5.3 Remove As Paramorphism .....   | 19 |
| 6 Generalizing Data Structures .....                                     | 20 |
| 6.1 Generalizing Quicksort .....   | 20 |
| 6.2 Generalizing Heap Sort .....   | 21 |
| 7 Conclusions .....  | 23 |
| Generic Programming – An Introduction – .....                            | 28 |
| <i>Roland Backhouse, Patrik Jansson, Johan Jeuring, Lambert Meertens</i> |    |
| 1 Introduction .....   | 28 |
| 1.1 The Abstraction-Specialisation Cycle .....                           | 28 |
| 1.2 Genericity in Programming Languages .....                            | 29 |
| 1.3 Path Problems .....  | 30 |
| 1.4 The Plan .....   | 33 |
| 1.5 Why Generic Programming? .....                                       | 35 |
| 2 Algebras, Functors and Datatypes .....                                 | 36 |
| 2.1 Algebras and Homomorphisms .....                                     | 36 |
| 2.2 Functors .....   | 43 |

|   |  |     |
|---|--|-----|
| 2.3   | Polynomial Functors .....  | 46  |
| 2.4   | Datatypes Generically .....  | 54  |
| 2.5   | A Simple Polytypic Program .....                                   | 67  |
| 3   | PolyP .....  | 68  |
| 3.1   | Regular Functors in PolyP .....                                    | 69  |
| 3.2   | An Example: <code>psum</code> .....                                | 70  |
| 3.3   | Basic Polytypic Functions .....                                    | 72  |
| 3.4   | Type Checking Polytypic Functions .....                            | 73  |
| 3.5   | More Examples of Polytypic Functions .....                         | 75  |
| 3.6   | PolyLib: A Library of Polytypic Functions .....                    | 76  |
| 4   | Generic Unification .....  | 83  |
| 4.1   | Monads and Terms .....   | 85  |
| 4.2   | Generic Unification .....  | 89  |
| 5   | From Functions to Relations .....                                  | 94  |
| 5.1   | Why Relations? .....   | 94  |
| 5.2   | Parametric Polymorphism .....                                      | 95  |
| 5.3   | Relators .....   | 99  |
| 5.4   | Occurs-In .....  | 101 |
| 6   | Solutions to Exercises .....                                       | 104 |
| Generic Program Transformation .....                              |  | 116 |
| <i>Oege de Moor and Ganesh Sittampalam</i>                        |  |     |
| 1   | Introduction .....   | 116 |
| 2   | Abstraction versus Efficiency .....                                | 117 |
| 2.1   | Minimum Depth of a Tree .....                                      | 117 |
| 2.2   | Decorating a Tree .....  | 118 |
| 2.3   | Partitioning a List .....  | 119 |
| 3   | Automating the Transition: Fusion and Higher Order Rewriting ..... | 120 |
| 4   | The MAG System .....   | 125 |
| 4.1   | Getting Acquainted .....   | 125 |
| 4.2   | Accumulation Parameters .....                                      | 128 |
| 4.3   | Tupling .....  | 131 |
| 4.4   | Carrying On .....  | 133 |
| 5   | Matching Typed $\lambda$ -Expressions .....                        | 134 |
| 5.1   | Types .....  | 134 |
| 5.2   | Expressions .....  | 135 |
| 5.3   | Substitutions .....  | 138 |
| 5.4   | Matching .....   | 138 |
| 6   | Concluding Remarks .....   | 140 |
| 7   | Answers to Exercises .....   | 143 |
| Designing and Implementing Combinator Languages .....             |  | 150 |
| <i>S. Doaitse Swierstra, Pablo R. Azero Alcocer, João Saraiva</i> |  |     |
| 1   | Introduction .....   | 150 |
| 1.1   | Defining Languages .....   | 150 |
| 1.2   | Extending Languages .....  | 151 |

|     |  |     |
|-----|--|-----|
| 1.3 | Embedding Languages .....  | 151 |
| 1.4 | Overview .....   | 152 |
| 2   | Compositional Programs .....                                     | 153 |
| 2.1 | The Rep_Min Problem .....  | 153 |
| 2.2 | Table_Formatting .....   | 159 |
| 2.3 | Defining Catamorphisms .....                                     | 170 |
| 2.4 | Discussion .....   | 175 |
| 3   | Attribute Grammars .....   | 177 |
| 3.1 | The Rep_Min Problem .....  | 177 |
| 3.2 | The Table_Formatting Problem .....                               | 181 |
| 3.3 | Comparison with Monadic Approach .....                           | 184 |
| 4   | Pretty Printing .....  | 185 |
| 4.1 | The General Approach .....                                       | 187 |
| 4.2 | Improving Filtering .....  | 188 |
| 4.3 | Loss of Sharing in Computations .....                            | 193 |
| 4.4 | Discussion .....   | 196 |
| 5   | Strictification .....  | 201 |
| 5.1 | Introduction .....   | 201 |
| 5.2 | Pretty Printing Combinators Strictified .....                    | 201 |
| 6   | Conclusions .....  | 203 |
|     | Using MetaML: A Staged Programming Language .....                | 207 |
|     | <i>Tim Sheard</i>  |     |
| 1   | Why Staging? .....   | 207 |
| 2   | Relationship to Other Paradigms .....                            | 210 |
| 3   | Introducing MetaML .....   | 211 |
| 3.1 | The Bracket Operator: Building Pieces of Code .....              | 212 |
| 3.2 | The Escape Operator: Composing Pieces of Code .....              | 213 |
| 3.3 | The run Operator: Executing User-Constructed Code .....          | 214 |
| 3.4 | The lift Operator: Another Way to Build Code .....               | 215 |
| 3.5 | Lexical Capture of Free Variables: Constant Pieces of Code ..... | 216 |
| 4   | Pattern Matching Against Code .....                              | 217 |
| 5   | A Staged Term Rewriting System .....                             | 218 |
| 6   | Safe Reductions under Brackets .....                             | 222 |
| 6.1 | Safe-Beta .....  | 222 |
| 6.2 | Safe-Eta .....   | 222 |
| 6.3 | Safe-Let-Hoisting .....  | 223 |
| 7   | Non-standard Extensions .....                                    | 223 |
| 7.1 | Higher Order Type Constructors .....                             | 223 |
| 7.2 | Local Polymorphism .....   | 224 |
| 7.3 | Monads .....   | 225 |
| 7.4 | Monads in MetaML .....   | 226 |
| 7.5 | An Example Monad .....   | 226 |
| 7.6 | Safe Monad-Law-Normalization Inside Brackets .....               | 227 |
| 8   | From Interpreters to Compilers Using Staging .....               | 228 |
| 8.1 | The While-Language .....   | 228 |

|   |  |     |
|---|--|-----|
| 8.2   | The Structure of the Solution .....        | 229 |
| 8.3   | Step 1: Monadic Interpreter .....          | 231 |
| 8.4   | Step 2: Staged Interpreter .....           | 233 |
| 9   | Typing Staged Programs .....               | 236 |
| 9.1   | Type Questions Still to be Addressed ..... | 236 |
| 10  | Conclusion .....                           | 238 |
| 11  | Exercises .....                            | 238 |
| Cayenne — A Language with Dependent Types ..... |  | 240 |
| <i>Lennart Augustsson</i>                       |  |     |
| 1   | Introduction .....                         | 240 |
| 1.1   | The Type of <code>printf</code> .....      | 241 |
| 1.2   | The Set “Package” .....                    | 242 |
| 1.3   | The <code>Eq</code> Class .....            | 243 |
| 2   | Core Cayenne .....                         | 246 |
| 2.1   | Functions .....                            | 246 |
| 2.2   | Data Types .....                           | 247 |
| 2.3   | Records .....                              | 248 |
| 2.4   | The Type of Types .....                    | 248 |
| 3   | Full Cayenne .....                         | 249 |
| 3.1   | Hidden Arguments .....                     | 249 |
| 3.2   | Syntactic Sugar .....                      | 250 |
| 3.3   | Modules .....                              | 251 |
| 4   | The Cayenne Type System .....              | 252 |
| 4.1   | Translucent Sums .....                     | 252 |
| 4.2   | Typing and Evaluation Rules .....          | 253 |
| 4.3   | Type Checking .....                        | 254 |
| 4.4   | Undecidability in Practice .....           | 257 |
| 5   | Cayenne as a Proof System .....            | 258 |
| 6   | Implementation .....                       | 258 |
| 6.1   | Erasing Types .....                        | 258 |
| 6.2   | Keeping Types .....                        | 260 |
| 6.3   | The Current Implementation .....           | 260 |
| 7   | Related Work .....                         | 260 |
| 8   | Future Work .....                          | 261 |
| 9   | Acknowledgments .....                      | 261 |
| A   | The <code>Eq</code> Class .....            | 264 |
| B   | The Tautology Function .....               | 266 |
| Haskell as an Automation Controller .....       |  | 268 |
| <i>Daan Leijen, Erik Meijer, James Hook</i>     |  |     |
| 1   | Introduction .....                         | 268 |
| 2   | Minuscule Introduction to Haskell .....    | 269 |
| 3   | Using COM Components .....                 | 270 |
| 3.1   | MS Agents in Haskell .....                 | 271 |
| 3.2   | Exercises .....                            | 273 |

|     |  |     |
|-----|--|-----|
| 4   | Essential COM .....                    | 273 |
| 4.1 | Interface Types .....                  | 274 |
| 4.2 | Inheritance .....                      | 274 |
| 4.3 | IDL .....                              | 275 |
| 5   | Automation .....                       | 276 |
| 5.1 | Using Automation .....                 | 276 |
| 5.2 | Methods .....                          | 277 |
| 5.3 | Properties .....                       | 277 |
| 5.4 | HaskellDirect .....                    | 278 |
| 5.5 | Exercises .....                        | 278 |
| 6   | Advanced Automation .....              | 278 |
| 6.1 | Variants .....                         | 279 |
| 6.2 | Optional Arguments .....               | 279 |
| 7   | Advanced Example .....                 | 280 |
| 7.1 | Webster .....                          | 281 |
| 7.2 | Exercises .....                        | 282 |
| 8   | Interacting with other Languages ..... | 282 |
| 8.1 | The Script Server Interfaces .....     | 283 |
| 8.2 | Exporting Values from Haskell .....    | 284 |
| 8.3 | Visual Basic and Haskell .....         | 285 |
| 8.4 | Importing Values into Haskell .....    | 286 |
| 8.5 | Handling Events .....                  | 286 |
| 8.6 | Exercises .....                        | 288 |
| 9   | Conclusions .....                      | 288 |

Advanced Functional Programming

Third International School, AFP'98, Braga, Portugal,

September 12-19, 1998, Revised Lectures

Swierstra, S.D.; Henriques, P.R.; Oliveira, J.N. (Eds.)

1999, XIV, 298 p., Softcover

ISBN: 978-3-540-66241-9