In *Pearls of Functional Algorithm Design* Richard Bird takes a radically new approach to algorithm design, namely design by calculation. The body of the text is divided into 30 short chapters, called pearls, each of which deals with a particular programming problem. These problems, some of which are completely new, are drawn from sources as diverse as games and puzzles, intriguing combinatorial tasks and more familiar algorithms in areas such as data compression and string matching.

Each pearl starts with the statement of the problem expressed using the functional programming language Haskell, a powerful yet succinct language for capturing algorithmic ideas clearly and simply. The novel aspect of the book is that each solution is calculated from the problem statement by appealing to the laws of functional programming.

*Pearls of Functional Algorithm Design* will appeal to the aspiring functional programmer, students and teachers interested in the principles of algorithm design, and anyone seeking to master the techniques of reasoning about programs in an equational style.

**Richard Bird** is Professor of Computer Science at Oxford University and Fellow of Lincoln College, Oxford.
PEARLS OF FUNCTIONAL ALGORITHM DESIGN

RICHARD BIRD

University of Oxford
Dedicated to my wife, Norma.
# Contents

 Preface ix  
 1 The smallest free number 1  
 2 A surpassing problem 7  
 3 Improving on saddleback search 12  
 4 A selection problem 21  
 5 Sorting pairwise sums 27  
 6 Making a century 33  
 7 Building a tree with minimum height 41  
 8 Unravelling greedy algorithms 50  
 9 Finding celebrities 56  
 10 Removing duplicates 64  
 11 Not the maximum segment sum 73  
 12 Ranking suffixes 79  
 13 The Burrows–Wheeler transform 91  
 14 The last tail 102  
 15 All the common prefixes 112  
 16 The Boyer–Moore algorithm 117  
 17 The Knuth–Morris–Pratt algorithm 127  
 18 Planning solves the Rush Hour problem 136  
 19 A simple Sudoku solver 147  
 20 The *Countdown* problem 156  
 21 Hylomorphisms and nexuses 168  
 22 Three ways of computing determinants 180  
 23 Inside the convex hull 188
<table>
<thead>
<tr>
<th>Chapter</th>
<th>Title</th>
<th>Page</th>
</tr>
</thead>
<tbody>
<tr>
<td>24</td>
<td>Rational arithmetic coding</td>
<td>198</td>
</tr>
<tr>
<td>25</td>
<td>Integer arithmetic coding</td>
<td>208</td>
</tr>
<tr>
<td>26</td>
<td>The Schorr–Waite algorithm</td>
<td>221</td>
</tr>
<tr>
<td>27</td>
<td>Orderly insertion</td>
<td>231</td>
</tr>
<tr>
<td>28</td>
<td>Loopless functional algorithms</td>
<td>242</td>
</tr>
<tr>
<td>29</td>
<td>The Johnson–Trotter algorithm</td>
<td>251</td>
</tr>
<tr>
<td>30</td>
<td>Spider spinning for dummies</td>
<td>258</td>
</tr>
<tr>
<td></td>
<td><strong>Index</strong></td>
<td>275</td>
</tr>
</tbody>
</table>
Preface

In 1990, when the *Journal of Functional Programming* (JFP) was in the stages of being planned, I was asked by the then editors, Simon Peyton Jones and Philip Wadler, to contribute a regular column to be called *Functional Pearls*. The idea they had in mind was to emulate the very successful series of essays that Jon Bentley had written in the 1980s under the title “Programming Pearls” in the *Communications of the ACM*. Bentley wrote about his pearls:

Just as natural pearls grow from grains of sand that have irritated oysters, these programming pearls have grown from real problems that have irritated programmers. The programs are fun, and they teach important programming techniques and fundamental design principles.

I think the editors had asked me because I was interested in the specific task of taking a clear but inefficient functional program, a program that acted as a specification of the problem in hand, and using equational reasoning to calculate a more efficient one. One factor that stimulated growing interest in functional languages in the 1990s was that such languages were good for equational reasoning. Indeed, the functional language GOFER, invented by Mark Jones, captured this thought as an acronym. GOFER was one of the languages that contributed to the development of Haskell, the language on which this book is based. Equational reasoning dominates everything in this book.

In the past 20 years, some 80 pearls have appeared in the JFP, together with a sprinkling of pearls at conferences such as the *International Conference of Functional Programming* (ICFP) and the *Mathematics of Program Construction Conference* (MPC). I have written about a quarter of them, but most have been written by others. The topics of these pearls include interesting program calculations, novel data structures and small but elegant domain-specific languages embedded in Haskell and ML for particular applications.
My interest has always been in algorithms and their design. Hence the title of this book is *Pearls of Functional Algorithm Design* rather than the more general *Functional Pearls*. Many, though by no means all, of the pearls start with a specification in Haskell and then go on to calculate a more efficient version. My aim in writing these particular pearls is to see to what extent algorithm design can be cast in a familiar mathematical tradition of calculating a result by using well-established mathematical principles, theorems and laws. While it is generally true in mathematics that calculations are designed to simplify complicated things, in algorithm design it is usually the other way around: simple but inefficient programs are transformed into more efficient versions that can be completely opaque. It is not the final program that is the pearl; rather it is the calculation that yields it. Other pearls, some of which contain very few calculations, are devoted to trying to give simple explanations of some interesting and subtle algorithms. Explaining the ideas behind an algorithm is so much easier in a functional style than in a procedural one: the constituent functions can be more easily separated, they are brief and they capture powerful patterns of computation.

The pearls in this book that have appeared before in the JFP and other places have been polished and repolished. In fact, many do not bear much resemblance to the original. Even so, they could easily be polished more. The gold standard for beauty in mathematics is *Proofs from The Book* by Aigner and Ziegler (third edition, Springer, 2003), which contains some perfect proofs for mathematical theorems. I have always had this book in mind as an ideal towards which to strive.

About a third of the pearls are new. With some exceptions, clearly indicated, the pearls can be read in any order, though the chapters have been arranged to some extent in themes, such as divide and conquer, greedy algorithms, exhaustive search and so on. There is some repetition of material in the pearls, mostly concerning the properties of the library functions that we use, as well as more general laws, such as the fusion laws of various folds. A brief index has been included to guide the reader when necessary.

Finally, many people have contributed to the material. Indeed, several pearls were originally composed in collaboration with other authors. I would like to thank Sharon Curtis, Jeremy Gibbons, Ralf Hinze, Geraint Jones and Shin-Cheng Mu, my co-authors on these pearls, for their kind generosity in allowing me to rework the material. Jeremy Gibbons read the final draft and made numerous useful suggestions for improving the presentation. Some pearls have also been subject to close scrutiny at meetings of the Algebra of Programming research group at Oxford. While a number of flaws and errors have been removed, no doubt additional ones have been introduced. Apart
from those mentioned above, I would like to thank Stephen Drape, Tom Harper, Daniel James, Jeffrey Lake, Meng Wang and Nicholas Wu for many positive suggestions for improving presentation. I would also like to thank Lambert Meertens and Oege de Moor for much fruitful collaboration over the years. Finally, I am indebted to David Tranah, my editor at Cambridge University Press, for encouragement and support, including much needed technical advice in the preparation of the final copy.

Richard Bird