

PURELY FUNCTIONAL DATA STRUCTURES

Most books on data structures assume an imperative language like C or C++. However, data structures for these languages do not always translate well to functional languages such as Standard ML, Haskell, or Scheme. This book describes data structures from the point of view of functional languages, with examples, and presents design techniques so that programmers can develop their own functional data structures. It includes both classical data structures, such as red-black trees and binomial queues, and a host of new data structures developed exclusively for functional languages. All source code is given in Standard ML and Haskell, and most of the programs can easily be adapted to other functional languages.

This handy reference for professional programmers working with functional languages can also be used as a tutorial or for self-study.

Cambridge University Press
0521663504 - Purely Functional Data Structures
Chris Okasaki
Frontmatter
[More information](#)

PURELY FUNCTIONAL DATA STRUCTURES

CHRIS OKASAKI
COLUMBIA UNIVERSITY



Cambridge University Press
0521663504 - Purely Functional Data Structures
Chris Okasaki
Frontmatter
[More information](#)

PUBLISHED BY THE PRESS SYNDICATE OF THE UNIVERSITY OF CAMBRIDGE
The Pitt Building, Trumpington Street, Cambridge, United Kingdom

CAMBRIDGE UNIVERSITY PRESS
The Edinburgh Building, Cambridge CB2 2RU, UK www.cup.cam.ac.uk
40 West 20th Street, New York, NY 10011-4211, USA www.cup.org
10 Stamford Road, Oakleigh, Melbourne 3166, Australia
Ruiz de Alarcón 13, 28014 Madrid, Spain

© Cambridge University Press 1998

This book is in copyright. Subject to statutory exception and
to the provisions of relevant collective licensing agreements,
no reproduction of any part may take place without
the written permission of Cambridge University Press.

First published 1998
First paperback edition 1999

Typeface Times 10/13 pt.

A catalog record for this book is available from the British Library

Library of Congress Cataloging in Publication data is available

ISBN 0 521 63124 6 hardback
ISBN 0 521 66350 4 paperback

Transferred to digital printing 2003

Contents

<i>Preface</i>	<i>page ix</i>
1 Introduction	1
1.1 Functional vs. Imperative Data Structures	1
1.2 Strict vs. Lazy Evaluation	2
1.3 Terminology	3
1.4 Approach	4
1.5 Overview	4
2 Persistence	7
2.1 Lists	7
2.2 Binary Search Trees	11
2.3 Chapter Notes	15
3 Some Familiar Data Structures in a Functional Setting	17
3.1 Leftist Heaps	17
3.2 Binomial Heaps	20
3.3 Red-Black Trees	24
3.4 Chapter Notes	29
4 Lazy Evaluation	31
4.1 $\$$ -notation	31
4.2 Streams	34
4.3 Chapter Notes	37
5 Fundamentals of Amortization	39
5.1 Techniques of Amortized Analysis	39
5.2 Queues	42
5.3 Binomial Heaps	45
5.4 Splay Heaps	46
5.5 Pairing Heaps	52

vi	<i>Contents</i>	
5.6	The Bad News	54
5.7	Chapter Notes	55
6	Amortization and Persistence via Lazy Evaluation	57
6.1	Execution Traces and Logical Time	57
6.2	Reconciling Amortization and Persistence	58
6.2.1	The Role of Lazy Evaluation	59
6.2.2	A Framework for Analyzing Lazy Data Structures	59
6.3	The Banker's Method	61
6.3.1	Justifying the Banker's Method	62
6.3.2	Example: Queues	64
6.3.3	Debit Inheritance	67
6.4	The Physicist's Method	68
6.4.1	Example: Binomial Heaps	70
6.4.2	Example: Queues	72
6.4.3	Example: Bottom-Up Mergesort with Sharing	74
6.5	Lazy Pairing Heaps	79
6.6	Chapter Notes	81
7	Eliminating Amortization	83
7.1	Scheduling	84
7.2	Real-Time Queues	86
7.3	Binomial Heaps	89
7.4	Bottom-Up Mergesort with Sharing	94
7.5	Chapter Notes	97
8	Lazy Rebuilding	99
8.1	Batched Rebuilding	99
8.2	Global Rebuilding	101
8.2.1	Example: Hood–Melville Real-Time Queues	102
8.3	Lazy Rebuilding	104
8.4	Double-Ended Queues	106
8.4.1	Output-Restricted Deques	107
8.4.2	Banker's Deques	108
8.4.3	Real-Time Deques	111
8.5	Chapter Notes	113
9	Numerical Representations	115
9.1	Positional Number Systems	116
9.2	Binary Numbers	116
9.2.1	Binary Random-Access Lists	119
9.2.2	Zeroless Representations	122

<i>Contents</i>		vii
9.2.3	Lazy Representations	125
9.2.4	Segmented Representations	127
9.3	Skew Binary Numbers	130
9.3.1	Skew Binary Random-Access Lists	132
9.3.2	Skew Binomial Heaps	134
9.4	Trinary and Quaternary Numbers	138
9.5	Chapter Notes	140
10	Data-Structural Bootstrapping	141
10.1	Structural Decomposition	142
10.1.1	Non-Uniform Recursion and Standard ML	143
10.1.2	Binary Random-Access Lists Revisited	144
10.1.3	Bootstrapped Queues	146
10.2	Structural Abstraction	151
10.2.1	Lists With Efficient Catenation	153
10.2.2	Heaps With Efficient Merging	158
10.3	Bootstrapping To Aggregate Types	163
10.3.1	Tries	163
10.3.2	Generalized Tries	166
10.4	Chapter Notes	169
11	Implicit Recursive Slowdown	171
11.1	Queues and Deques	171
11.2	Catenable Double-Ended Queues	175
11.3	Chapter Notes	184
A	Haskell Source Code	185
	<i>Bibliography</i>	207
	<i>Index</i>	217

Preface

I first began programming in Standard ML in 1989. I had always enjoyed implementing efficient data structures, so I immediately set about translating some of my favorites into Standard ML. For some data structures, this was quite easy, and to my great delight, the resulting code was often both much clearer and much more concise than previous versions I had written in C or Pascal or Ada. However, the experience was not always so pleasant. Time after time, I found myself wanting to use destructive updates, which are discouraged in Standard ML and forbidden in many other functional languages. I sought advice in the existing literature, but found only a handful of papers. Gradually, I realized that this was unexplored territory, and began to search for new ways of doing things.

Eight years later, I am still searching. There are still many examples of data structures that I just do not know how to implement efficiently in a functional language. But along the way, I have learned many lessons about what *does* work in functional languages. This book is an attempt to codify these lessons. I hope that it will serve as both a reference for functional programmers and as a text for those wanting to learn more about data structures in a functional setting.

Standard ML Although the data structures in this book can be implemented in practically any functional language, I will use Standard ML for all my examples. The main advantages of Standard ML, at least for presentational purposes, are (1) that it is a strict language, which greatly simplifies reasoning about how much time a given algorithm will take, and (2) that it has an excellent module system that is ideally suited for describing these kinds of abstract data types. However, users of other languages, such as Haskell or Lisp, should find it quite easy to adapt these examples to their particular environments. (I provide Haskell translations of most of the examples in an appendix.) Even

C or Java programmers should find it relatively straightforward to implement these data structures, although C's lack of automatic garbage collection can sometimes prove painful.

For those readers who are not familiar with Standard ML, I recommend Paulson's *ML for the Working Programmer* [Pau96] or Ullman's *Elements of ML Programming* [Ull94] as introductions to the language.

Other Prerequisites This book is not intended as a first introduction to data structures in general. I assume that the reader is reasonably familiar with basic abstract data types such as stacks, queues, heaps (priority queues), and finite maps (dictionaries). I also assume familiarity with the basics of algorithm analysis, especially “big-Oh” notation (e.g., $O(n \log n)$). These topics are frequently taught in the second course for computer science majors.

Acknowledgments My understanding of functional data structures has been greatly enriched by discussions with many people over the years. I would particularly like to thank Peter Lee, Henry Baker, Gerth Brodal, Bob Harper, Haim Kaplan, Graeme Moss, Simon Peyton Jones, and Bob Tarjan.