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.

PURELY FUNCTIONAL DATA STRUCTURES

CHRIS OKASAKI
COLUMBIA UNIVERSITY

CAMBRIDGE UNIVERSITY PRESS

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

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

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.