Book review

Purely Functional Data Structures by Chris Okasaki, Cambridge University Press, 1998

This is a well-written book about purely functional data structures. Actually, it is more than that: the main emphasis is on general *techniques* for designing data structures, which sets it apart from most other textbooks on data structures. The book provides a wealth of ideas and examples in a compact 220 pages. Every functional programmer who finds herself in need of an efficient data structure should definitely read this book. Even anyone who is more than casually interested in data structures should read this book. That said, however, the reader must be familiar both with the basics of data structures and with a functional language — preferably Standard ML (SML) including its module system as SML is used for the examples.

Are functional data structures different from imperative ones? Yes they are — and for one main reason. Functional programmers love referential transparency — the ability to substitute equals for equals — which is why assignments are frowned upon. Unfortunately, assignments are indispensable for many data structures including simple ones like doubly linked lists and complex ones like the quad edge data structure (Guibas & Stolfi, 1985). Not using assignments, however, has its merits: purely functional data structures are automatically *persistent*. A data structure is called persistent if, after an update, the old version of the data structure is still available for processing. By contrast, imperative data structures are typically *ephemeral*: an update destroys the old version of the data structure. So Chris Okasaki's book really deals with persistent data structures, which should make it worthwhile for a wider audience including imperative programmers.

The book is divided into three parts. The first part serves as an introduction, the second explores the relationship between lazy evaluation and amortization, and the third is concerned with general design techniques.

Part I. Chapter 1 sets the stage by delineating the difference between functional and imperative data structures and between eager and lazy evaluation. Chapter 2 explains how functional languages achieve persistence through path copying. Actually the chapter serves a double purpose since it also illustrates the use of signatures, structures and functors for implementing abstract data types. This mingling is a bit unfortunate as it possibly distracts from the main thread. Chapter 3 covers three data structures which are easily implemented in a functional setting: leftist heaps, binomial queues, and red-black trees. In each case it is a delight to look at the code which is concise and clear. Insertion into a red-black tree, for instance, takes 13 lines. By contrast, the imperative version in (Cormen *et al.*, 1991) occupies 43 lines even though symmetric cases are omitted.

Part II. Data structures with amortized time bounds are usually simpler than their worst-case counterparts making them the data structure of choice for many applications. Unfortunately, the classical approaches to amortization do not work in a persistent setting. Moreover, for a long time amortization and persistence were considered to be incompatible. Using lazy evaluation this Gordian knot can be cut. Chapter 4 introduces an extension of SML (a strict language) which supports lazy evaluation. Chapter 5 reviews the classical techniques of amortization and gives well chosen examples illustrating their use: Gries' FIFO queues, splay heaps, and pairing heaps. Chapter 6 explains in great detail the use of lazy evaluation to implement persistent, amortized data structures. Again, the techniques are supported by several carefully analyzed implementations of FIFO queues and priority queues. Chapter 7 shows how to convert an amortized data structure into a worst-case one by systematically scheduling delayed computations.

Part III. Chapter 8 describes the techniques of batched, global and lazy rebuilding and provides several implementations of FIFO queues and deques based on these ideas. Chapter 9 deals with so-called numerical representations. A container type is termed a numerical representation if it is designed in close analogy with a number system. The basic idea is to model a container with n elements on the representation of n and to model the operations on the container on the corresponding arithmetic functions. Chris Okasaki develops this technique into a fine art. He presents an abundance of number systems, explains their properties, and shows how to turn them into efficient implementations of flexible arrays and priority queues. Chapter 10 discusses three different flavours of data-structural bootstrapping: bootstrapping unbounded data structures from bounded ones, bootstrapping efficient data structures from inefficient ones, and bootstrapping data structures for compound types from data structures for atomic types. Data-structural bootstrapping is applied, for instance, to implement optimal priority queues. Finally, Chapter 11 combines ideas from the previous two chapters into a framework called implicit recursive slowdown. The examples illustrating this technique culminate in an implementation of catenable deques that are optimal in an amortized sense.

Complete implementations in Standard ML are given for all data structures. Furthermore, there is an appendix providing Haskell translations of most of the examples. Unfortunately, the Haskell code is neither explained nor documented. For example, one major difference between Haskell and Standard ML is the use of type classes rather than structures and functors. Lack of explanation here is definitely a shortcoming and should be corrected in a second edition. The text is complemented by exercises and bibliographic remarks at the end of each chapter. Occasionally, there are 'hints to practitioners' which recommend data structures that perform well in practice. In summary, this is a great book from a competent author and I heartily recommend it.

References

Cormen, Thomas H., Leiserson, Charles E., & Rivest, Ronald L. (1991). *Introduction to algorithms.* Cambridge, Massachusetts: The MIT Press.

Guibas, Leonidas, & Stolfi, Jorge. (1985). Primitives for the manipulation of general subdivisions and computation of Voronoi diagrams. *ACM transactions on graphics*, **4**(2), 74–123.

Ralf Hinze