J. Functional Programming **7** (2): 227–229, March 1997. Printed in the United Kingdom

Book review

Garbage Collection: Algorithms for Automatic Dynamic Memory Management by Richard Jones and Rafael Lins, John Wiley & Sons, 1996.

`http://www.ukc.ac.uk/computer_science/Html/Jones/GC/gcbook.html`

This book is an excellent and up-to-date survey of garbage collection algorithms and techniques. Functional programmers have always needed garbage collection, to reclaim unused data structures and function closures, but the rest of the world is now discovering that safe programming requires an automatic garbage collector.

It is remarkable that garbage collection – which is at heart just the simple problem of calculating graph reachability – is still a thriving research area after nearly four decades of work. Mark-sweep collection (McCarthy, 1960) and reference counting (Collins, 1960) were first described in 1960, and the 1981 survey paper by Cohen (1981) cited 111 papers on garbage collection; Jones and Lins cite almost 500 references. What's all the fuss about?

Perhaps the problem is that garbage collectors are subject to many conflicting demands. The collector imposes overhead on any other algorithm the programmer may implement, so it must be extremely time-efficient; it must use as little auxiliary space as possible; it must cause no internal or external fragmentation; it must not interrupt the program with long pauses for collection; it must find pointers even within the registers and stack frames rearranged by a highly optimizing compiler; it must not require special-purpose hardware tag bits or machine instructions; it must not be confused by cycles of pointers; it must be able to operate even while the program is simultaneously fetching and storing pointers; it must work on a distributed computation; it must not disturb the locality of reference of a cached or virtual-memory computation; and it must be simple to implement. If no single algorithm has all these properties, then no algorithm can strictly dominate all the others. Each new research paper claims superiority, but along a dimension subtly different from the previous paper. It is not hard to see why so much has been written, and why further research continues.

The authors explain the fundamental algorithms and practically all the modern improvements, discuss the controversies and give the reader useful criteria for making judgments. Their book is useful as an introduction to the subject and each different style of garbage collection (as listed in the chapter summaries below), and also as a bibliographic reference pointing to the best papers of the recent (and 'classical') literature.

Chapter 1 explains what garbage collection is and why it is useful in software engineering, and discusses many of the issues – demands placed on a collector and the different kinds of collection algorithms that meet these demands – that are

covered in the rest of the book. Chapter 2 introduces the three 'classical' algorithms – reference-counting, mark-sweep, and copying collection. Like most chapters, it has a very useful *issues to consider* section describing what user requirements can lead to the selection of one algorithm over another.

Chapter 3 covers reference counting, including deferred reference counting, limited-size 'sticky' reference count fields, partial mark-sweep algorithms, special hardware, and several other variations. After an *issues to consider* section discussing factors that may influence a decision to use or avoid reference counts, it concludes with a discussion of related literature.

Chapter 4 describes mark-sweep collection, mark-stack management (including pointer reversal), mark-bit placement, and lazy sweeping. It concludes with a comparison of mark-sweep versus copying collection on issues such as space usage, locality of reference, time complexity, and object mobility, and remarks that "the choice of collector now depends as much on the kind of application it is to support as on the intrinsic properties of the collector itself." Chapter 5 discusses several algorithms for mark-compact collection.

Chapter 6 covers copying collection: Cheney's algorithm, cheap allocation in a contiguous freespace, multiple fromspaces and tospaces, and traversal strategies to improve locality. As usual it concludes with a discussion of strengths and weaknesses of the algorithms covered, and a discussion of the relevant literature.

Chapter 7 covers generational collection: the generational hypothesis, multi-generation heaps, pause times, root sets, intergenerational pointers, promotion policies, organization of generations, and age recording. It also describes several methods for recording pointers from old to new generations.

Chapter 8 describes concurrent collection, using the *tricolor marking* abstraction to explain several different algorithms. It concludes by comparing the different synchronization costs incurred by different styles of algorithm, and a lengthy discussion of the literature.

Chapter 9 describes conservative collection, for programming languages or compilers which do not accurately describe to the collector the locations of pointers within data and stack frames. Important recent techniques such as *blacklisting* are described, as well as the possible pointer-hiding behavior of an optimizing compiler. Chapter 10 describes several approaches to collection for C++.

Chapter 11, 'Cache-conscious garbage collection', describes how modern cache architectures work, explains the typical memory-access patterns of mark-sweep and copying collectors, and shows the results of empirical measurements by several researchers on the cache performance of garbage-collected systems.

Chapter 12 covers distributed collection. It introduces several addressing models for distributed systems, explains notions of active versus passive objects, access methods, cyclic structures, and synchronization. Then it describes nine different distributed mark-sweep algorithms and implementations, briefly mentions distributed copying algorithms, covers eight distributed reference-counting algorithms, and summarizes a few algorithms for collecting actors.

Many of the algorithms are presented in detailed pseudocode so that the reader can learn how they really work. There are many quantitative and qualitative discussions

of experience from the research literature. The prose is accessible to the novice but not tedious to the experienced. There is a glossary, a substantial bibliography, and an index.

The main weakness is an inadequate explanation of how a compiler should describe the layout of stack frames and heap objects to a collector. Many readers of this book will have a compiler to which they wish to attach a collector; their first step must (usually) be to outfit their compiler to propagate type information all the way to the back end. An entire chapter could profitably be devoted to this topic, perhaps starting with Hudson *et al.* (1991) and Diwan, Moss and Hudson (1992).

Another useful reference on the subject is a well-written survey paper by Wilson (1997), which covers basic algorithms (reference counts, mark-sweep, copying), has a very coherent discussion of several incremental algorithms, covers generational algorithms well, and covers low-level implementation issues such as compiler interfaces. But – as a survey paper and not a textbook – it does not provide detailed pseudocode for each algorithm, and cites only about a third as many papers.

The Jones and Lins book will likely be the definitive text and reference book on garbage collection for the twentieth century. It presents a very good picture of the state of the art up to 1995. Is it possible that the furious pace of garbage-collection research will abate, or will a book review in the year 2010 discuss the remarkable vigor of research and the 1000+ citations in the latest book of that day?

References

Cohen, J. (1981) Garbage collection of linked data structures. *Computing surveys*, **13**(3), 341–367.

Collins, G. E. (1960) A method for overlapping and erasure of lists. *Communications of the ACM*, **3**(12), 655–657.

Diwan, A., Moss, E. and Hudson, R. (1992) Compiler support for garbage collection in a statically typed language. *Proc. ACM SIGPLAN '92 Conf. on Prog. Lang. Design and Implementation*, pp. 273–282. ACM Press.

Hudson, R., Moss, E., Diwan, A. and Weight, C. F. (1991) *A language-independent garbage collector toolkit*. Technical report TR 91-47, University of Massachusetts at Amherst, Department of Computer Science.

McCarthy, J. (1960). Recursive functions of symbolic expressions and their computation by machine – I. *Communications of the ACM*, **3**(1), 184–195.

Wilson, P. R. (1997) Uniprocessor garbage collection techniques. *ACM Computing Surveys* (to appear).

Andrew W. Appel
Princeton University