Combining Generational and Conservative Garbage Collection: Framework and Implementations

Alan Demers
Mark Weiser
Barry Hayes
Hans Boehm
Daniel Bobrow
Scott Shenker

Xerox Palo Alto Research Center
Palo Alto, Ca 94304

SUMMARY

Two key ideas in garbage collection are *generational collection* and *conservative pointer-finding*. Generational collection and conservative pointer-finding are hard to use together, because generational collection is usually expressed in terms of copying objects, while conservative pointer-finding precludes copying. We present a new framework for defining garbage collectors. When applied to generational collection, it generalizes the notion of younger/older to a partial order. It can describe traditional generational and conservative techniques, and lends itself to combining different techniques in novel ways. We study in particular two new garbage collectors inspired by this framework. Both these collectors use conservative pointer-finding. The first one is based on a rewrite of an existing trace-and-sweep collector to use one level of generation. The second one has a single parameter, which controls how objects are partitioned into generations; the value of this parameter can be changed dynamically with no overhead. We have implemented both collectors and present measurements of their performance in practice.

This appeared in the Conference Record of the 17th Annual ACM Symposium on Principles of Programming Languages, January 17-19, 1990, pp. 261-269.

I. Introduction

Garbage collectors fall into two general classes: reference-counting and tracing. In this paper we consider only tracing collectors. A tracing garbage collector works by starting with a root set of memory objects, following the pointers found there to other memory objects that should be preserved, and so on recursively, until all objects accessible from the roots have been found. Inaccessible objects are garbage and can be reclaimed.

Garbage collection has a colorful past. It is considered essential by some programming subcultures, such as those from a Lisp heritage, and is considered superfluous or dangerous by other subcultures, such as those from a systems programming or real-time background. However, there has been some use of garbage collection in systems programming [Rovner85] [Weiser89] [Cardelli88], and even real-time constraints are possible [Baker78] [Appel88]. In general, interest in garbage collection is growing.

Garbage collection is almost never shared among multiple language implementations. Instead, every language with garbage collection does it differently, even idiosyncratically, because collection usually depends on implementation assumptions about the uses of pointers. There is, however, a technique for identifying pointers that is nearly language-independent. This technique, called *conservative pointer-finding*, identifies a superset of the true pointers, in effect by treating every word of a memory object as if it *might* possibly contain a pointer [Boehm88]. Conservative pointer-finding precludes copying objects when an object is copied, every pointer to that object must be updated to refer to the new location, but conservative pointer-finding cannot distinguish pointers from integers,