Time Required for Garbage Collection in Retention Block-Structured Languages

D. M. Berry and A. Sorkin

Received March 1975; revised February 1977

This paper investigates the time requirements for an implementation of retention block-structured languages that uses a garbage collector as its sole means for recovering inaccessible storage. The usual three-pass mark-compactify-and-update garbage collector is optimized to eliminate the need for the third pass in the event that the executed program is lifetime well-stacking.

**KEY WORDS:** Block-structured languages; retention vs. deletion; contour model; garbage collection; time estimates.

“You probably work for the City of Los Angeles.”
The first author’s neighbor on the plane back to Los Angeles in September 1974, after seeing the first half of the title of this paper.

1. INTRODUCTION

In block-structured languages, upon entry to a block or procedure, storage is allocated for the identifiers declared in the block or procedure. There are two choices as to when to deallocate this storage:

1. Upon exit from the block or procedure,
2. When the storage becomes inaccessible.

---

1 Supported (in part) by the United States Energy Research and Development Administration, Contract No. E(04-3)-34, PA 214, and (in part) by the National Science Foundation, Grant No. DCR75-08659.
2 Computer Science Department, UCLA, Los Angeles, California.
The former is referred to as the deletion strategy because storage for a block or procedure is deleted (deallocated) automatically upon exit. The latter is referred to as the retention strategy because it is possible for storage for a block or procedure to be retained after exit. These two strategies are not equivalent, as there exist numerous examples of programs making use of pointer, label, and procedure values, which show the difference between these two strategies. For examples, see Refs. 3, 4, 7, 8, 12, and 20. The difference shows up in the form of a dangling reference in the deletion strategy which does not appear in the retention strategy.

In the deletion strategy, storage for blocks and procedures is allocated and deallocated in a last-in-first-out order. Therefore, storage for blocks and procedures can be managed efficiently on a pushdown stack. However, deletion gives rise to dangling references in the use of pointer, label, and procedure values. This is dealt with in one of two ways which compromises either security or generality:

1. There is no attempt to prevent or detect dangling references, and errors, which are hard for the programmer to detect, can occur. This is the case in PL/I-F, in which a pure stack model with no run time checks can be used.\(^{(19)}\)

2. There are restrictions on the use of pointer, label, and procedure values which require the addition of some run-time checks to the basic stack model. This is the case with ALGOL 68, in which the run time checks are to prevent dangling references, and with Deletion Parallel Euler, in which the run time checks are to detect dangling references.\(^{(7,17,18)}\)

Retention is both more secure and more general in that it eliminates all possibility of dangling references, and it therefore removes all restrictions on the use of pointer, label, and procedure values. This is the case with GEDANKEN, PAL, Oregano, and Retention Parallel Euler.\(^{(2,4,7,15,21)}\) However, retention is considered to be less efficient than deletion, as it requires more sophisticated storage management techniques, including the use of a free list, reference count management, and/or garbage collection.

While retention is clearly nicer, its inefficiency appears to be the major obstacle to its general use. Most programs that are written are single task programs that use integer, real, Boolean, and character string values almost exclusively. They use pointers only for passing parameters by reference, and they use labels and procedures only as constants. Furthermore, these procedures return only integer, real, Boolean, and character string values. These programs do not require retention, and run correctly on an implementation using the deletion strategy. We call programs that run correctly on a deletion implementation, e.g., the stack model, well-stacking (WS). It is only the rare program doing list manipulation or demonstrating a principle of computer