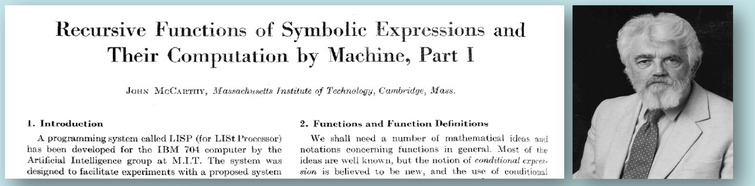
Down for the Count? Getting Reference Counting Back in the Ring

By Rifat Shahriyar†, rifat.shahriyar@anu.edu.au. Supervisors: Steve Blackburn† and Daniel Frampton$^{\delta}$

Garbage collection (GC) is Ubiquitous

- Born 52 years ago



Recursive Functions of Symbolic Expressions and Their Computation by Machine, Part I

JOHN MCCARTHY, *Massachusetts Institute of Technology, Cambridge, Mass.*

1. Introduction

A programming system called LISP (for LISt Processor) has been developed for the IBM 704 computer by the Artificial Intelligence group at M.I.T. The system was designed to facilitate experiments with a proposed system

2. Functions and Function Definitions

We shall need a number of mathematical ideas and notations concerning functions in general. Most of the ideas are well known, but the notion of *conditional expression* is believed to be new, and the use of conditional

A Method for Overlapping and Erasure of Lists

GEORGE E. COLLINS, *IBM Corp., Yorktown Heights, N. Y.*

Abstract. An important property of the Newell-Shaw-Simon scheme for computer storage of lists is that data having multiple occurrences need not be stored at more than one place in the computer. That is, lists may be "overlapped." Unfortunately, overlapping poses a problem for subsequent erasure. Given a list that is no longer needed, it is desired to erase just those parts that do not overlap other lists. In LISP, McCarthy employs an elegant but inefficient solution to the problem. The present paper describes a general method which enables efficient erasure. The method employs interspersed reference counts to describe the extent of

McCarthy's solution is very elegant, but unfortunately it contains two sources of inefficiency. First and most important, the time required to carry out this reclamation process is nearly independent of the number of registers reclaimed.[1] Its efficiency thereby diminishes drastically as the memory approaches full utilization.[2] Second, the method as used by McCarthy required that a bit (McCarthy uses the sign bit) be reserved in each word for

- Two ideas underpin large literature:
  - **Tracing** [McCarthy60]
  - **Reference Counting** [Collins60]
- However
  - ✔ Tracing used in all high performance GCs
  - ✘ Reference counting (RC) only in non-performance critical settings

Reference Counting vs. Tracing

- **Advantages**
  - ✔ Immediate
  - ✔ Incremental (Reclaim as-you-go)
  - ✔ Object-local
  - ✔ Overhead distributed
  - ✔ Very simple (Trivial implementation for naïve RC)
- **Disadvantages**
  - ✘ Maintain count (Time and space overheads)
  - ✘ Cycles (Can't be collected)
  - ✘ Complex (High performance implementation about as complex as tracing)

The Challenge

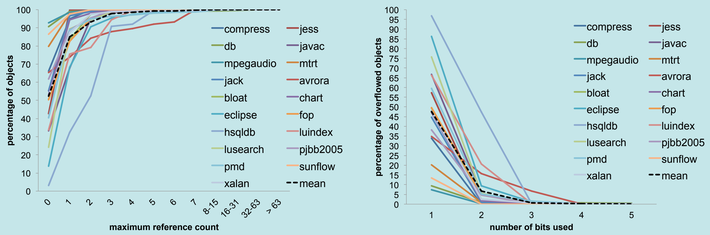
- ✔ One of the two fundamental GC algorithms
- ✔ Many advantages
- ✘ Neglected by all performance-conscious VMs
- ✘ High performance RC is 30% slower than mark-sweep (MS)

Can we get RC back in the ring?

† Computer Systems Group, Research School of Computer Science, College of Engineering & Computer Science, Australian National University, $^{\delta}$ Microsoft

Storing the Count

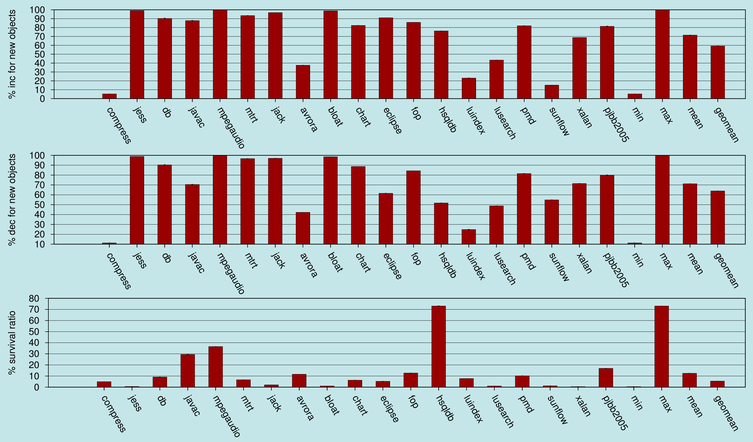
- **Space**
  - Dedicated word (32 bits) per object
  - Steal bits from each object's header
- **Findings**
  - Max count < 8 in most cases
  - Very few overflows (The percentage of stuck objects is very low)



- **Design: Handling stuck objects**
  - Auxiliary data structure to store count of the overflowed objects
  - Ignore them let backup tracing collect stuck objects
  - Restore them let backup tracing restore stuck counts

Maintaining the Count

- **Types**
  - Deferred RC ignores changes to stacks and register
  - Coalescing RC ignores many changes to heap
- **Findings**
  - New objects are the source of most incs and decs
  - Survival ratio is low
  - New objects a fruitful focus for optimization



Handling of New Objects

- **Existing**
  - Implicitly dirty (marked as dirty and inc enqueued for next collection)
  - Implicitly live (initial count of one and dec enqueued for next collection)
- **New**
  - Implicitly clean (lazily dirty at collection time, non-surviving never processed)
  - Implicitly dead (lazily increment at collection time, available to free list if no inc)

Performance Improvement

- ✔ Our improved RC is 24% faster than the standard RC
- ✔ Standard RC was 30% slower then MS, but our improved RC performs the same
- ✔ Improved RC is only 2% slower than URC and 3% slower than Immix
- ✔ It performs same as Sticky MS but 10% slower than Sticky Immix (current project)

RC is back in the ring

Australian National University

ANU College of Engineering & Computer Science