Modern Algorithms for Garbage Collection

Outlining modern algorithms for garbage collection on the examples of Go, Python, OCaml and Java

Daniel Huber Matteo Gropp

October 11, 2023

Contents

1 Introduction 3

2 Overview over Garbage Collection 4

3 Comparison with other Memory Management Techniques 5

3.1 Manual Memory Management 5

3.2 Lifetimes and Borrow Checking 5

4 Tradeoffs between Garbage Collection Algorithms 6

5 Overview over current Garbage Collection Algorithms 7

5.1 Go 7

5.2 Python 7

5.3 OCaml 7

5.4 Java 7

Abstract

Summary in english and in german here

1 Introduction

Garbage collection refers to the process of automatically managing heap allocated memory on behalf of the running process by identifying parts of memory that are no longer needed. [1, Introduction] This is often performed by the runtime of a programming language while the program is executing. [2, Introduction]
Most programming languages allocate values with static lifetimes[1] in main memory along with the executable code. Values that are alive for a certain scope are allocated using the call stack[2] without requiring dynamic allocation. These Variables can't escape the scope they were defined in and must be dynamically allocated if accessing them outside of their scope is desired.
This requires the programmer to allocate and deallocate these variables to prevent memory leaks[3] provided the programming language does not perform garbage collection.

```c
typedef struct {
    char *name;
    double age;
} Person;

Person *new_person(char *name, double age) {
  Person *p = malloc(sizeof(Person));
  p->age = age, p->name = name;
  return p;
}
```

Listing 1: C heap allocation example

Listing 1 showcases a possible use case for dynamic memory allocation. The `Person` structure is filled with values defined in the parameters of the `new_person` function. This structure, if stack allocated, would not live longer than the scope of the `new_person` function, thus rendering this function useless. To create and use a `Person` structure outside of its scope, the structure has to be dynamically allocated via the `malloc` function defined in the `#include <stdlib.h>` header.

[1]variable available for the whole runtime of the program [3, Abstract]
[2]stores information about running subroutines / functions[4, 2.2 Call Stacks]
[3]allocated no longer needed memory not deallocated [5, 1.2.1 A Practical Object Ownership Model]

2 Overview over Garbage Collection

3 Comparison with other Memory Management Techniques

3.1 Manual Memory Management

3.2 Lifetimes and Borrow Checking

4 Tradeoffs between Garbage Collection Algorithms

5 Overview over current Garbage Collection Algorithms

5.1 Go

5.2 Python

5.3 OCaml

5.4 Java

References

[1] *A Guide to the Go Garbage Collector.* URL: `https://tip.golang.org/doc/gc-guide` (visited on 10/09/2023).

[2] *The Go Programming Language Specification.* URL: `https://tip.golang.org/ref/spec` (visited on 10/11/2023).

[3] *Beyond static and dynamic scope.* URL: `https://dl.acm.org/doi/abs/10.1145/1837513.1640137` (visited on 10/11/2023).

[4] *Call Stack Coverage for GUI Test-Suite Reduction.* URL: `https://www.cs.umd.edu/~atif/papers/McMasterMemonISSRE2006.pdf` (visited on 10/11/2023).

[5] David L Heine and Monica S Lam. “A practical flow-sensitive and context-sensitive C and C++ memory leak detector”. In: *Proceedings of the ACM SIGPLAN 2003 conference on Programming language design and implementation.* 2003, pp. 168–181.