Lab 5 Report: Mark–Sweep Garbage Collector

Astitwa Saxena
2025MCS3005

Abhishek Gupta
2025MCS2963

1 Introduction

This report describes the design and implementation of a stop-the-world mark–sweep garbage collector integrated into the Virtual Machine developed in previous labs. The objective of this assignment is to provide automatic memory management for heap-allocated objects such as pairs, functions, and closures, while ensuring correctness, safety, and efficiency. The garbage collector identifies live objects through root discovery, marks all reachable objects, and reclaims memory occupied by unreachable objects. This document explains the design, integration, and correctness of the garbage collector as required in Lab 5.

2 Description of GC Design and Integration

The garbage collector is implemented using a stop-the-world **mark–sweep** strategy and is fully integrated into the Virtual Machine (VM). All dynamically allocated objects such as pairs, functions, and closures are stored in a linked list representing the heap. Each object contains a mark bit and references to other objects.

Garbage collection is explicitly invoked using a dedicated VM instruction. When this instruction is executed, the VM pauses execution, performs garbage collection, and then resumes normal program execution. This explicit triggering allows controlled testing and evaluation of GC behavior.

The overall GC process consists of two phases:

1. Mark phase: All reachable objects are identified and marked.
2. Sweep phase: All unmarked objects are reclaimed.

This design ensures that memory is reclaimed safely and that no unreachable objects remain allocated.

3 Root Identification Strategy

Roots are the starting points for reachability analysis. In this VM, roots are identified from:

- The operand stack: all values of type `VAL_OBJ`.
- The VM memory array: all memory slots that contain object references.

These roots represent all objects that are directly accessible by the program at any moment. Any object that can be reached by following references starting from these roots is considered live and must not be collected.

The following logic illustrates root discovery:

```
for (auto& v : operandStack)
    if (v.type == VAL_OBJ) mark(v.obj);
```

4 Mark and Sweep Implementation Details

4.1 Mark Phase

The mark phase performs a traversal of the object graph starting from the roots. Each object is visited and marked as reachable. From each marked object, its internal references are followed:

- For a pair object, the left and right fields are explored.
- For a closure object, the function and environment references are explored.

An explicit stack is used instead of recursion to avoid stack overflow when processing deep object graphs or cyclic structures. Each object is marked only once, preventing infinite loops in the presence of cycles.

4.2 Sweep Phase

The sweep phase scans the entire heap. For each object:

- If the object is not marked, it is unreachable and is deleted.
- If the object is marked, it is preserved and its mark bit is reset for the next GC cycle.

This process ensures that all unreachable memory is reclaimed while reachable objects remain intact.

5 Discussion of Correctness and Limitations

5.1 Correctness

The garbage collector is correct for the following reasons:

- All objects reachable from the root set are preserved.
- All unreachable objects are reclaimed during the sweep phase.
- Cyclic references are handled correctly, as marking is based on reachability rather than reference counts.
- Deep object graphs are processed safely without stack overflow due to the use of an explicit traversal stack.
- The system avoids memory leaks and unsafe memory access.

The implementation is validated using test cases that cover:

- Reachable vs. unreachable objects.
- Transitive reachability.
- Cyclic references.
- Deep object graphs.
- Stress allocation and full heap cleanup.

5.2 Limitations

- The GC is stop-the-world and pauses VM execution during collection.
- Garbage collection must be triggered explicitly by the program.
- The system does not implement generational or incremental collection.
- No heap compaction is performed, so fragmentation may occur over time.

6 Conclusion

The mark–sweep garbage collector is successfully implemented and integrated into the virtual machine. The design correctly manages heap allocation, identifies roots accurately, traverses object graphs safely, and reclaims unreachable memory. The implementation satisfies all the requirements specified for Lab 5 and demonstrates a complete and reliable memory management system.