Garbage Collection Techniques

Notes by Bogdan Stroe

The paper covers the main algorithms for garbage collecting. First are described the basic techniques, and performances problems of these algorithms are discussed. Next are presented refinements of basic algorithms, with the final goal of obtaining a functional real time garbage collecting system.

Next, we present a summary of each section of the paper:

1. Introduction
   - application GB's;
   - GB in programming languages;
   - support from compiler;

   - the two phase abstraction: garbage detection and reclaiming the garbage;
   - garbage detection based on *reachability* from a *root* set;
   - the *root* set contains globally visible variables of active procedures and local variables of active procedures, and the registers used by active procedures;
   - assumption: RTT information;

2. Basic GC techniques
   - three approaches: *reference counting*, *marking*, *copying*.

   2.1 Reference counting
   Advantages: incremental in nature;
   Problems: *efficiency* (cost per program instruction, short lived variables) and *effectiveness* (cycles);

   2.2 Mark-Sweep Collection
   - two phases: marking the reachable graph and reclaiming the garbage (usually into free lists;
     Problems: fragmentation (possible solution is keeping free lists for different sizes), cost of collection is proportional with the size of the heap, and locality of reference.

   2.3 Mark-Compact Collection
   - live objects are compacted together, the rest of the space is free;
   - advantages: no fragmentation, easy allocation, improved locality;
   - problems: several (two or three) passes through the whole heap;

   2.4 Copying Garbage Collection
   - heap divided in two contiguous *semispaces*;

- memory is allocated easily;
- once we run out of memory, live data **from** current space is copied **to** the other space, and then the roles are reversed;
- copying traversal Cheney algorithm;
- forwarding pointers to replace the copied objects;

3. Incremental Tracing Collectors

3.1 Tricolor Marking
- black (checked), white (un-known) and grey (reacheable, but not checked yet) objects; different representations of this abstraction;
- the mutator should act inside a wave-front of grey objects;
- no pointers from black to white objects allowed;
- two approaches to coordinate the mutator and the collector: *read barrier* and *write barrier;*
- two types of write barrier: *snapshot at beginning* and *incremental update;*

3.2 Baker's Incremental Copying
- uses a read barrier;
- copying is made in BF search;
- new objects allocated in *to-space;*(i.e. they are black)
- a read operation may result in a copying of an object;

3.3 The Treadmill
- non-copying version of Baker's scheme, using double-linked lists;
- new objects allocated "black";

3.4 Snapshot-at-Beginning write-barrier algorithms
- we construct the reachability graph as it is at the beginning of the process;
- if we write to a location, we save the old value in a stack; this old value become a root variable;
- no objects can be freed during collection;

3.5 Incremental Update Write-Barrier Algorithms
- if we write a pointer to a white object in a black object we make the latter grey;
- new objects are white (less conservative);
- Steele's variation: new objects are allocated black or white depending on their referents;

4. Generational Garbage Collection
- some objects (few) live long and most of them live short;
- multiple subheaps with varying scavenge frequencies;
- detecting intergenerational references (e.g. indirection table for each generation);
- additional questions: advancement policy, heap organization, traversal algorithms, collection scheduling, intergenerational references.