MASSACHVSETTS INSTITVTE OF TECHNOLOGY
Department of Electrical Engineering and Computer Science
6.001—Structure and Interpretation of Computer Programs
Fall 2007

# Recitation 25 Solutions
# Garbage

## Memory

Assume that memory consists of many cells, each of which contains a number and a tag. For now, just look at tags "N" (number), "P" (pointer to another cell), and "E" (null: the empty list).

For example:

| | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| the-cars | N3 | N4 | P0 | N3 | N5 | P2 | N2 | N3 | P1 | N4 | |
| the-cdrs | E0 | E0 | P4 | P5 | P0 | P6 | P5 | P3 | P3 | N5 | |

Using this implementation of cons cells, `(cons a b)` does these steps:

1. Look for the next free location ($i$)
2. Store $a$ at `the-cars[`$i$`]`
3. Store $b$ at `the-cdrs[`$i$`]`
4. Return $P_i$, a pointer to the new cell.

Draw a box and pointer diagram starting from P5.

Note the not everything in memory is useful. For example, consider what happens when evaluating `(define foo (map square (list 1 2 3)))`.

At the end of evaluating, there are no pointers to the list `(1 2 3)` so those cells are wasted – those pairs are *garbage*.

What's Garbage? Garbage is anything that cannot have any influence on the future computation. Everything else we need to keep around. We define the Root Set to be the set of all pointers in the machine registers and stack. The cells we might need are defined as those cells that can be reached by a succesion of car and cdr operations starting from the root set.

## Reference Counting

1. Attach a counter to each pair in memory.
2. When a new pointer is connected to that pair, increment the counter.

3. When a pointer is removed, decrement the counter.
4. Any cell with a 0 counter is garbage, and can be re-used.

Draw a box-and-pointer diagram, keeping a reference counter with each cell:

```
(define a (list 1 2 3))
(set-cdr! a 4)
(set-cdr! a a)
(set! a 1)
```

What's the problem here?

## Stop and Copy

Here's the main idea to Stop and Copy:

1. Split memory in half (Working memory and Copy memory).
2. Keep a free pointer to the free part of memory.
3. When Memory runs out, stop computation and begin garbage collection.
   (a) Place scan and free pointers at the start of the Copy memory.
   (b) Copy the Root Set over to copy memory, incrementing free. In each location that's copied over, put the "Broken Heart" token in the car and the forwarding address in the cdr.
   (c) Starting at the scan pointer, check the car and the cdr. If either is a pointer, look at the location in Working memory. If it's already been copied (i.e. it has a "Broken Heart"), update the reference. Otherwise, copy the location and put the "Broken Heart" in the old location.
   (d) Repeat until scan = free.
   (e) Swap the roles of the Working and Copy memory.

The register maxchine code for this algorithm is in the text.

For example, let's perform stop-and-copy on the following with Root Set = P5 :

| Working | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| the-cars | N3 | N4 | P0 | N3 | N5 | P2 | N2 | N3 | P1 | N4 | |
| the-cdrs | E0 | E0 | P4 | P5 | P0 | P6 | P5 | P3 | P3 | N5 | |

| Copy | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| the-cars | | | | | | | | | | | |
| the-cdrs | | | | | | | | | | | |

This method sounds expensive, because half of memory is idle except when using the Garbage Collector. However, it goes just about as fast as possible. It only examines valid locations. Most memory is garbage, and never even gets looked at. We see a very common tradeoff here: Pay a lot of space, get a lot of time.

## Mark-sweep

Here's the main idea to Mark-Sweep:

1. Add a Mark Bit to each location in memory.
2. Keep a free pointer to the head of the free list.
3. When Memory runs out, stop computation, clear the Mark Bits and begin garbage collection.
4. Mark
   (a) Start at the root and follow the accessible structure (keeping a stack of where you still need to go).
   (b) Mark every cell you visit.
   (c) Stop when you see a marked cell, so you don't go into a cycle.
5. Sweep
   (a) Start at the end of memory, and build a new free list.
   (b) If a cell is unmarked, then it's garbage, so hook it into the free list.

For example, let's perform Mark-Sweep on the following with Root Set = P5 :

| Working | 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9 | 10 |
|---|---|---|---|---|---|---|---|---|---|---|---|
| the-cars | N3 | N4 | P0 | N3 | N5 | P2 | N2 | N3 | P1 | N4 | |
| the-cdrs | E0 | E0 | P4 | P5 | P0 | P6 | P5 | P3 | P3 | N5 | |
| marks | | | | | | | | | | | |

Stack:

Free Pointer:

This method is a good deal more compact than stop-and-copy. However, it also always looks through all of memory, so it is fairly slow.

In this example, we needed a stack to keep track of where we still need to explore. This can actually be done without a stack by keeping track of recursive calls using the data structures in memory themselves.