GC V: generational GC

David Bremner

April 7, 2021

Generational Garbage Collection

Generational Hypothesis

Most objects die young

- die means become garbage
- empirically well supported
- like many good ideas, goes back to Self in the 1980s

Our examples have lots of young garbage

sum

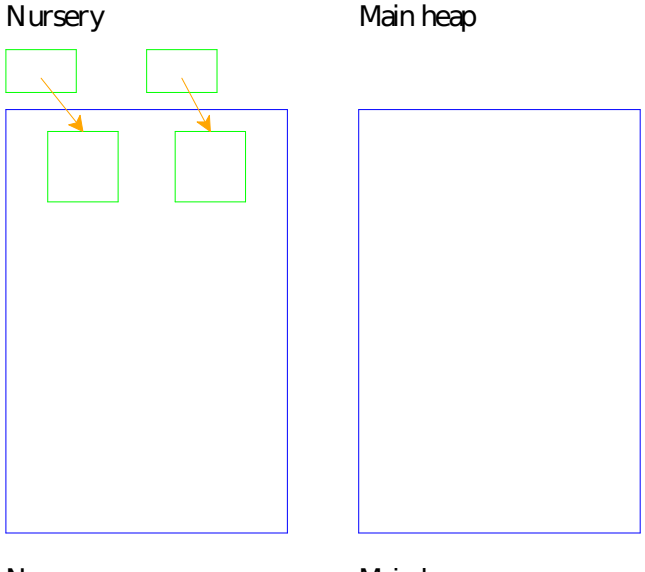
```
(allocator-setup "mark-sweep-free-list.rkt" 160)
(define (sum lst)
  (cond
    [(empty? lst) 0]
    [else (+ (first lst) (sum (rest lst)))]))

(sum '(1 2 3 4 5 6 7 8 9 10 11 12 13 14 15 16 17 18))
```

Our examples have lots of young garbage

	0	1	2	3	4	5	6	7	8	9
0	#f	'clos	sum	0	'flat	1	'flat	2	'flat	3
10	'flat	4	'flat	5	'flat	6	'flat	7	'flat	8
20	'flat	9	'flat	10	'flat	11	'flat	12	'flat	13
30	'flat	14	'flat	15	'flat	16	'flat	17	'flat	18
40	'flat	empty	'cons	38	40	'cons	36	42	'cons	34
50	45	'cons	32	48	'cons	30	51	'cons	28	54
60	'cons	26	57	'cons	24	60	'cons	22	63	'cons
70	20	66	'cons	18	69	'cons	16	72	'cons	14
80	75	'cons	12	78	'cons	10	81	'cons	8	84
90	'cons	6	87	'cons	4	90	'flat	#f	'flat	#f
100	'flat	#f	'flat	#f	'flat	#f	'flat	#f	'flat	#f
110	'flat	#f	'flat	#f	'flat	#f	'flat	#f	'flat	#f
120	'flat	#f	'flat	#f	'flat	#f	'flat	#f	'flat	#f
130	'flat	#f	'flat	#t	'flat	0	'flat	18	'flat	35
140	'flat	51	'flat	66	'flat	80	'flat	93	'flat	105
150	'flat	116	'flat	126	'flat	135	'flat	143	'flat	150

- heap state just before first gc
- before about location 95 is live
- all of the `'flat #f` are garbage
- other stuff is potentially live

Generational Garbage Collection



How it (mostly) works

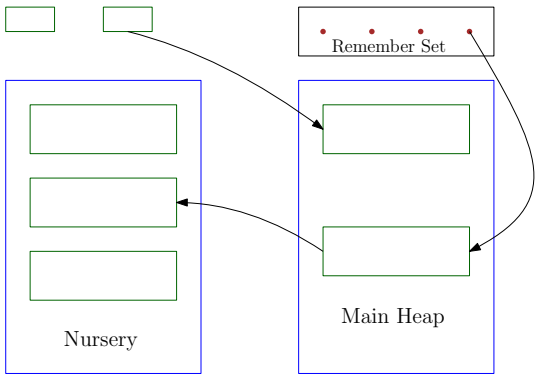
Nursery

- Collected often
- Cheap to collect as long as mostly dead objects
- Typically small

Main (Tenured) heap

- Large enough to contain peak “live” data
- Marking and sweeping both expensive
- Don’t collect often

Intergenerational Pointers

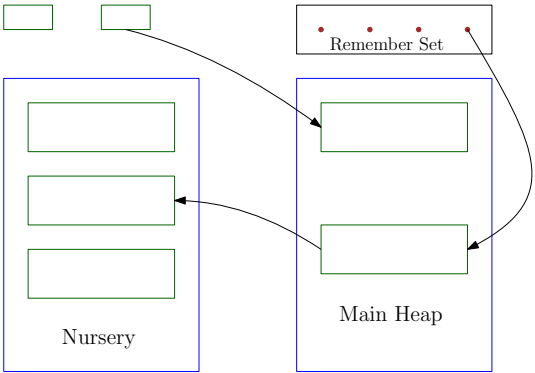


- Can't ignore main heap when GCing nursery.
- Avoiding marking main heap is the point of gen. GC

metadata Track pointers from main heap to nursery

conservative Assume they're live

Updating Remember Sets



- old-young points are created by mutation
- write barriers catch mutation

Write Barriers

```
(define (gc:set-first! pr-loc new)
  (cond
    [(gc:cons? pr-loc)
     (define loc (track/loc pr-loc))
     (heap-set! (+ loc 1) new)
     (when (and (2nd-gen? loc)
                (at-to-space? new))
       (table/alloc (+ loc 1) new))]
    [else (error 'set-first! "non pair at ~s"
      pr-loc)]))

(define (track/loc loc-or-root)
  (define loc (->location loc-or-root))
  (case (heap-ref loc)
    [(flat pair proc) loc]
    [(frwd) (heap-ref (+ loc 1))]
```

Heap layout

	0	1	2	3	4	5	6	7	8	9
0	1	'free	'free	'free	'free	'free	'free	'free	'free	'free
10	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
20	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
30	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
40	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
50	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
60	'free-n	#f	150	'free	'free	'free	'free	'free	'free	'free
70	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
80	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
90	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
100	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
110	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
120	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
130	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
140	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
150	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
160	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
170	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
180	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
190	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
200	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
210	60	212	'free	'free	'free	'free	'free	'free	'free	'free
220	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
230	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free

Nursery

Table

Two stage GC

```
(define (malloc n . extra-roots)
  (define addr (heap-ref (alloc-word)))
  (cond
    [(enough-to-space? addr n)
     (heap-set! (alloc-word) (+ addr n))
     addr]
    [else
     (collect-garbage extra-roots)
     (switch/sweep-tospace n)]))

(define (forward/loc loc)
  (cond
    [(at-to-space? loc)
     (case (heap-ref loc)
       ; ⋮
```

Quicksort Demo

sort

```
(define (sort l less?)
  (if (empty? l) l
      (let* ([pivot (first l)]
             [left? (λ (x) (less? x pivot))]
             [left (filter left? (rest l))]
             [right? (λ (x) (not (less? x pivot)))]
             [right (filter right? (rest l))])
        (append (append (sort left less?)
                        (cons pivot empty))
                (sort right less?)))))
(sort '(3 1 4 2) (λ (a b) (< a b)))
```

Heap State

	0	1	2	3	4	5	6	7	8	9
0	39	'out	'flat	#t	'frwd	198	'flat	#t	'flat	#t
10	'flat	#f	'flat	#t	'frwd	195	4	'flat	#t	'flat
20	#f	'flat	#t	'frwd	192	14	'flat	#f	'flat	#f
30	'free	'flat	#t	'pair	104	192	'pair	83	33	'free
40	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
50	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
60	'proc	sort	0	'proc	filter1	0	'proc	filter	0	'proc
70	reverse1	0	'proc	reverse	0	'proc	append	0	'proc	not
80	0	'flat	4	'flat	1	'flat	3	'proc	temp179	2
90	92	108	'pair	85	95	'pair	83	98	'pair	81
100	101	'pair	104	106	'flat	2	'flat	empty	'proc	temp294
110	0	'flat	empty	'pair	104	116	'flat	empty	'pair	83
120	111	'flat	empty	'pair	83	113	'pair	104	129	'flat
130	empty	'proc	temp250	2	123	108	'pair	83	139	'flat
140	empty	'flat	empty	'pair	104	146	'flat	empty	'flat	empty
150	'pair	104	141	'pair	104	156	'flat	empty	'proc	temp250
160	2	92	108	'flat	empty	'pair	85	168	'flat	empty
170	'pair	83	153	'pair	81	176	'flat	empty	'proc	temp250
180	2	173	108	'flat	empty	'pair	81	188	'flat	empty
190	'flat	empty	'pair	85	195	'pair	81	198	'flat	empty
200	'free-n	#f	10	'free	'free	'free	'free	'free	'free	'free
210	200	212	'free	'free	'free	'free	'free	'free	'free	'free
220	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free
230	'free	'free	'free	'free	'free	'free	'free	'free	'free	'free

Nursery

Table

Acknowledgements / References

- Lecture 22 based in part on slides by Vincent St. Amour.
- Generational collector, write barrier example, based on code from the Master's Thesis of Yixi Zhang `https://github.com/yixizhang/racket-gc/`
- The GC Handbook.