GC control loop for OCaml 5
Design document

Stephen Dolan and Damien Doligez

October 29, 2025

1 Prerequisites and assumptions

✓ Stephen Dolan's mark-delay patch (rebased and adapted by Nick Barnes, #13580), which delays the marking of roots until sweeping is mostly done. We'll make the simplifying assumption that sweeping is completed before the roots are marked.

• Stephen Dolan's accounting patch, oxcaml#3618, which ensures that the work done to mark the roots and the marking work done by the write barrier are counted as normal marking work.

✓ Sadiq Jaffer's patch, #13616, which ensures that the sweeping work is independent of the size and shape of the free list (formally, it is bounded by a linear function of live+garbage). Note that this may also be covered by oxcaml#3618. To be checked in `shared_heap.c/pool_sweep`.

• Another patch to change the ephemeron GC work accounting. The marking of ephemerons should be counted as normal marking (each block counting for its size when it gets marked). This is to get rid of arbitrary (up to quadratic) amounts of work when the structure of ephemeron graphs is complex. The upside is that this will make the GC behaviour predictable (in terms of space), the downside is that the GC pauses will have less-predictable length, hurting the worst-case latency of pauses. In practice this should not be a problem because the users of ephemerons (therorem-provers) are not dependent on real-time behaviour.

• Yet another patch to insert an Idle phase between Sweep and Mark. In this phase, the GC does not do any work while the mutator is allocating as usual.

We assume the following properties of the OCaml runtime:

• OCaml 5 ensures that the sweeping work is constant during one GC cycle (i.e. allocations done during sweeping do not increase work-to-do).

- The heap is large enough and the mutator allocations are smooth enough that we can ignore all quantization issues and treat all numbers as reals.

About words: we count allocations and work in units of (heap) words, including header and contents of each block. For marking and sweeping work, treating one block is counted as a number of work units equal to its size (with header). This is a simple-minded model of the actual cost of doing the work. It allows us to bound the time spent in a slice (hence the maximum latency of the GC) although it doesn't give a very good uniformity (the latency distribution is rather wide).

When we talk about *amount of data* we indicate a number of words (allocated or collected, live or dead, etc.) including the header words.

Steady-state assumption (SSA): whenever x words are allocated by the mutator, x (other) words are made unreachable. This is a simplifying assumption that lets us study the optimal solution to our problem: make the program use an amount of memory proportional to the size of the live data. We expect most programs to have at least some phases that violate this assumption. The GC behaviour during these phases will be studied in another document.

A GC cycle is composed of the following points and phases:

- (point) *beginning of cycle n*, rotation of the colours

- Sweep phase

- Idle phase

- (point) marking of roots

- Mark phase

- (point) *end of cycle n, beginning of cycle n + 1*

2 Simple heap

In this section we study the case of a program that does not make use of custom blocks or ephemerons.

2.1 Notations

We define the following variables:

o is the `overhead` parameter of the runtime system, given by the user.

L_n is the amount of live (marked and reachable) data at the beginning of cycle n

F_n is the amount of floating garbage (marked and unreachable) at the beginning of cycle n

G_n is the amount of collectible garbage (unmarked and unreachable) at the beginning of cycle n

S_n is the amount of data allocated during the Sweep phase of cycle n
I_n is the amount of data allocated during the Idle phase of cycle n
M_n is the amount of data allocated during the Mark phase of cycle n
s is the amount of GC work done per word allocated in the Sweep phase
m is the amount of GC work done per word allocated in the Mark phase

Note that s and m are the control parameters of the GC: in each major GC slice, we know how much the program allocated since the latest slice, and we have to decide how much GC work to do in reaction.

$$\beta = o/100 \tag{1}$$

This is the overhead setting expressed in a convenient unit.

$$\sigma = s/m \tag{2}$$

This is the ratio of sweeping to marking "speed". This will be chosen to make the GC pauses of the Sweep and Mark phases approximately equal in real-time. This is normally larger than 1 because it turns out that the GC code needs to do more work to mark x words of memory than to sweep the same amount. Note that this ratio is a function of the GC code rather than the memory behaviour of the program. We will choose a reasonable constant after studying the run-time behaviour of the code.

$$O_n = F_n + G_n \tag{3}$$

We call this *raw overhead*. This is our measure of the memory overhead of the system. It is composed of all the garbage that is present at the beginning of the GC cycle. In an ideal world, the free list is empty at the beginning of the cycle because we are starting to Sweep and replenish the free list. In reality, the free list may be non-empty, and there will be other overhead caused by auxiliary data structures and fragmentation. Since these cannot be controlled by changing the GC speed, we keep them out of our equations.

$$Q_n = O_n/L_n \tag{4}$$

This is the relative overhead: how much more memory we need per word of live data.

The goal of the GC pacing system is to choose s and m such that Q_n stays as close as possible to β, so we will look for a solution with the constraint:

$$Q = \beta \tag{5}$$

2.2 Steady state

From the SSA, we immediately deduce that S_n, I_n, M_n are also the amounts of garbage created during the Sweep, Idle, Mark phases respectively. We also get:

$$L_{n+1} = L_n \tag{6}$$

and in fact the amount of live data is constant through the GC cycle. We will drop the subscript and call this amount L.

Because the GC is a snapshot-at-beginning incremental GC, the floating garbage at the beginning of cycle $n + 1$ is what was allocated between the marking of roots and the end of cycle n:

$$F_{n+1} = M_n \tag{7}$$

The collectible garbage is composed of the floating garbage from the previous cycle, plus what became unreachable before the marking of roots:

$$G_{n+1} = F_n + S_n + I_n \tag{8}$$

The amount of work done during Sweep must be equal to the amount of live data plus the amount of garbage, because the free list is not swept.

$$sS_n = L + F_n + G_n \tag{9}$$

The amount of work done during Mark must be equal to the amount of live data at root marking time, which is L. This is because data allocated during marking is allocated marked and does not need to be marked again.

$$mM_n = L \tag{10}$$

2.3 Choice of I

I_n is how long we keep the GC in Idle, measured in memory allocated by the mutator during this time. We can choose whatever value we want for I_n, it is not constrained by the equations above. We are interested in running the GC in two modes:

- normal mode: when L is reasonably large, we don't use the Idle phase, so we set $I_n = 0$.

- small-heap mode: when L is very small, we can avoid running the major GC at full speed, at the cost of a small amount of memory (raw overhead), even though that translates to an unbounded relative overhead. To this end, we choose a constant J (for example, at the same order of magnitude as the minor heap size) and decide we must allocate J words before we start marking:

$$J = S_n + I_n \tag{11}$$

4

2.4 Solving for s and m

We can use the SSA to simplify notations: Assuming o (and β), s, m are constant, we also have L_n, M_n, F_n constant. We write them as L, M, F without subscript.

2.4.1 Normal mode

In normal mode, we have $I_n = 0$, thus

$$
\begin{aligned}
G_{n+1} &= F + S_n \\
&= F + L/s + F/s + G_n/s
\end{aligned}
\tag{12}
$$

This converges on a constant G iff $s > 1$. When $s \leq 1$, the sequence G will keep growing and memory use is unbounded.

We now assume $s > 1$. We also assume that G_n has converged to its limit, which we call G.

S_n, O_n, A_n, Q_n are now constants too, we'll write S, O, A, Q.

We get the following equations:

$$
\begin{aligned}
Q &= \beta \\
\sigma &= s/m \\
I &= 0 \\
Q &= O/L \\
O &= F + G \\
G &= F + S + I \\
F &= M \\
sS &= L + F + G \\
mM &= L
\end{aligned}
$$

These equations are mostly linear and we can eliminate all the uppercase variables to get[1]

$$
\begin{aligned}
s &= 1 + (2\sigma + 1)/\beta \\
m &= s/\sigma
\end{aligned}
$$

2.4.2 Small-heap mode

From (8) and (11) we get:

$$
G_{n+1} = F + J
\tag{13}
$$

---

[1] Checked by Rocq, with the additional conditions $L \neq 0$, $s \neq 0$, $Q \neq 0$

G is again constant, and so are S, I, O, Q. We get the following equations:

$$J = S + I$$
$$O = F + G$$
$$A = L + O$$
$$sS = L + F + G$$
$$mM = L$$
$$Q = O/L$$
$$F = M$$
$$G = M + J$$

We get[2]

$$A = sJ - sI$$
$$A \leq sJ$$

This gives us a nice bound on the active heap size, which is also a bound on the absolute overhead.

The relative overhead will be:

$$Q = 2/m + J/L \tag{14}$$

As expected, this can get arbitrarily large as L approaches 0.

2.4.3 Smooth transition between modes

At the transition between modes, we have $I = 0$ and $J = S$. Computing Q from the small-heap formula (14), we get:

$$Q = (2\sigma + 1)/(s - 1) \tag{15}$$

which is also the Q value from the normal mode, so there is no discontinuity between the modes.

3 Managing off-heap memory

Off-heap memory is memory allocated outside the heap, but linked from the heap. Since it gets deallocated (typically through finalisers) when the corresponding heap blocks are freed, it is in fact managed by the GC. We want the GC pacing logic to take it into account. Off-heap memory is normally managed through custom blocks, most importantly bigarrays (but also I/O buffers, etc.).

---

[2]Checked by Rocq, with the additional conditions $I \geq 0$, $s > 0$

3.1 Notations

We define a few more variables:

L'_n is the amount of off-heap live data (linked from live heap blocks) at the beginning of cycle n

F'_n is the amount of off-heap floating garbage (linked from floating garbage heap blocks and not live) at the beginning of cycle n

G'_n is the amount of off-heap collectible garbage (linked only from collectible garbage heap blocks) at the beginning of cycle n

S'_n is the amount of off-heap data allocated during the Sweep phase of cycle n

I'_n is the amount of off-heap data allocated during the Idle phase of cycle n

M'_n is the amount of off-heap data allocated during the Mark phase of cycle n

We introduce two more control parameters for the GC:

s' is the is the amount of GC work done per off-heap word allocated in the Sweep phase

m' is the is the amount of GC work done per off-heap word allocated in the Mark phase

3.2 Strenthening the SSA

We complete the SSA with a new assumption:

Extended steady-state assumption (SSAx): Whenever x' words are allocated off-heap, x' (other) off-heap words become unreachable.

We also introduce the Allocation Rate Assumption (ARA): that the ratio of off-heap to on-heap allocations is a constant e'. We get:

$$S'_n = e'S_n \tag{16}$$

$$I'_n = e'I_n \tag{17}$$

$$M'_n = e'M_n \tag{18}$$

From SSAx, we get:

$$L'_{n+1} = L'_n \tag{19}$$

Thus the amount of off-heap live data is constant and we'll call it L'.

We also have:

$$F'_{n+1} = M'_n \tag{20}$$

$$G'_{n+1} = F'_n + S'_n + I'_n \tag{21}$$

The GC never touches or even looks at off-heap data, so the amount of work to do isn't affected by off-heap data. Equations (9) and (10) become invalid and are replaced with:

$$sS_n + s'S'_n = L_n + F_n + G_n \tag{22}$$

$$mM_n + m'M'_n = L_n \tag{23}$$

7

3.3 New definition of overhead

We define the off-heap overhead to be all the off-heap garbage:

$$O'_n = F'_n + G'_n \tag{24}$$

We want to manage the on-heap and off-heap overhead as a whole, so we define the total space overhead to be the sum of the two:

$$\dot{O}_n = O_n + O'_n \tag{25}$$

Unlike L (which is measured indirectly by the Mark and Sweep phases), we notice that the magnitude of L' has no influence on the GC: the off-heap space is invisible to it. As a consequence, we will use a definition of relative overhead that disregards L': we make it relative only to L. For programs with a large L'/L factor, this might be a problem because the GC will aim for a very small overhead relative to the total live size $L + L'$. Our best solution for the moment is to count on the user to compensate by increasing the o parameter.

$$\dot{Q}_n = \dot{O}_n / L_n \tag{26}$$

The goal of the pacing logic is now to choose not only s and m, but also s' and m' to keep $\dot{Q}_n$ close to β.

We will choose s' and m' such that:

$$s'/m' = \sigma \tag{27}$$

As noted above ((2)), the ratio between s' and m' should be chosen as a function of the real-time behaviour of the marking and sweeping code. It doesn't change when the GC work is in response to off-heap allocation.

3.4 Choice of I and I'

I_n and I'_n are related by the ARA and (17), so there's only one thing to choose. The interesting value is the total active size $A + A'$ of the program rather than the way it is distributed between on-heap and off-heap data. As before, we'll distinguish two modes:

- Normal mode: when $L+L'$ is reasonably large, we want to use $I_n = I'_n = 0$

- Small-memory mode: when $L + L'$ is small, we want to slow down the GC by extending the Idle phase, and let the relative overhead get larger than the user setting. As before, we choose one constant, now called $\dot{J}$ and let:

$$\dot{J} = S_n + S'_n + I_n + I'_n \tag{28}$$

  In other words, we switch from Idle to Mark phase when $\dot{J}$ words have been allocated (on- and off-heap) since the beginning of the cycle.

3.5 Solving for s, m, s', m'

Using SSAx, we have L' constant; assuming s' and m' are constant and using (18), (20), (23), we infer that M' and F' are also constant.

3.5.1 Normal mode

We have the following equations:

$$S'_n = e'S_n$$
$$sS_n + s'S'_n = L + F + G_n$$
$$G_{n+1} = F + S_n$$

From which we get:

$$G_{n+1} = F + \frac{L + F + G_n}{s + e's'} \tag{29}$$

This converges on a constant G iff $s + e's' > 1$, otherwise garbage grows unboundedly.

We now assume $s > 1$, which ensures $s + e's' > 1$ for all values of e', and we also assume that G_n has converged to G.

S_n is now constant, and so is S'_n. We'll write S and S'.

From (16), (17), (18), (21) we get:

$$G'_{n+1} = e'G_{n+1} \tag{30}$$

Thus G'_n, O'_n, A'_n, $\dot{O}_n$, $\dot{Q}_n$ are also constant, we'll write G', O', A', $\dot{O}$, $\dot{Q}$.

Our goal is now to keep $\dot{Q}$ close to β, so we replace (5) with:

$$\dot{Q} = \beta \tag{31}$$

We have the following equations:

$$s = \sigma m$$
$$s' = \sigma m'$$
$$I = 0$$
$$I' = 0$$
$$M' = e'M$$
$$S' = e'S$$
$$F = M$$
$$F' = M'$$
$$G = F + S + I$$
$$G' = F' + S' + I'$$
$$sS + s'S' = L + F + G$$
$$mM + m'M' = L$$
$$O = F + G$$
$$O' = F' + G'$$
$$\dot{O} = O + O'$$
$$\dot{Q} = \dot{O}/L$$

Solving for s and s' as a function of $\dot{Q}$, we get[3]

$$s + e's' = 1 + (1 + e')(2\sigma + 1)/\dot{Q} \tag{32}$$

If $e' = 0$ we get the same equation as in the simple heap. If $e' > 0$, we get to choose s and s' as a function of e' and $\dot{Q}$. It is convenient to choose s independently of e', which means using the same solution as in the simple case:

$$s = 1 + (2\sigma + 1)/\beta \tag{33}$$

This leaves us with

$$s' = (2\sigma + 1)/\beta \tag{34}$$
$$s' = s - 1 \tag{35}$$

3.5.2   Small-memory mode

In small-memory mode, we will keep the same s and s' as for normal mode, and concentrate on the choice of $\dot{J}$.

---

[3] Checked by Rocq, with the additional conditions $L \neq 0$, $e' \geq 0$, $s > 0$, $s' > 0$, $\dot{Q} \neq 0$

We have the following equations:

$$s = m\sigma$$
$$s' = m'\sigma$$
$$G = F + S + I$$
$$G' = F' + S' + I'$$
$$F' = M'$$
$$F = M$$
$$S' = e'S$$
$$I' = e'I$$
$$M' = e'M$$
$$mM + m'M' = L$$
$$sS + s'S' = L + F + G$$
$$\dot{J} = S + S' + I + I'$$
$$O = F + G$$
$$O' = F' + G'$$
$$\dot{O} = O + O'$$
$$s = 1 + (2\sigma + 1)/\beta$$
$$s' = (2\sigma + 1)/\beta$$

We get [4]

$$\dot{O} \le \dot{J} + \frac{2L}{m'} \le \beta L + \dot{J}$$
$$\dot{O} \le \dot{J}(2 + 2\sigma + \beta)$$

The first line shows that small-memory mode cannot add more than a fixed amount of overhead $\dot{J}$, which is under direct control of the user.

The second line provides an absolute bound expressed directly in terms of $\dot{J}$ and β.

3.5.3   Transition between the modes

Our small-memory mode bound is pretty loose, so we can't say much about the transition between the modes. We would like to prove that it's continuous as a function of $I + I'$ but that's not obvious from the formulas.

We would also like to give a bound on the size of live memory at the transition point, to show that small-memory mode is indeed restricted to small heaps. Unfortunately, there is no constraint on L' in the above equations, so at best we'll be able to bound L.

---

[4]Checked by Rocq with the additional conditions $m' > 0$, $\sigma > 0$, $\beta > 0$, $e' >= 0$, $M >= 0$, $L >= 0$

4   Ephemerons

To deal with ephemerons, the GC has to maintain a global list of all ephemerons, and uses two more phases, inserted between the Mark phase and the end of the cycle:

- Ephe_mark, where the GC repeatedly goes through the list of ephemerons in order to mark their data alive according to some conditions (wich are irrelevant here). In the worst case, this phase will use time quadratic in the length of the list. Since this time cannot be predicted, we choose to ignore it: treat this phase as a part of the Mark phase, and count work units normally when blocks are marked. Note that this degrades the real-time performance of the GC for programs that make significant use of ephemerons with complex data structures (plain weak arrays are not affected).

- Ephe_sweep, where the GC goes once through the list of ephemerons to remove the garbage from the list and erase the weak pointers as needed.

Note that ephemerons are in a way dual to off-heap memory: they do not use memory, and they do add workload to the GC.

4.1   Notations

We add more notations:

L_n'' is the amount of live ephemerons at the beginning of cycle n.

F_n'' is the amount of floating garbage ephemerons at the beginning of cycle n.

G_n'' is the amount of collectible garbage ephemerons at the beginning of cycle n.

S_n'' is the amount of ephemerons allocated during the Sweep phase of cycle n.

I_n'' is the amount of ephemerons allocated during the Idle phase of cycle n.

M_n'' is the amount of ephemerons allocated during the Mark phase of cycle n.

W_n, W_n', W_n'' are the amounts of heap data, off-heap data, ephemerons allocated during the Ephe_sweep phase. (W stands for "weak")

s'' is the amount of GC work done per word of ephemerons allocated during the Mark phase.

m'' is the amount of GC work done per word of ephemerons allocated during the Mark phase.

w, w', w'' are the amounts of work done per word of heap data, off-heap data, ephemerons allocated during the Ephe_sweep phase.

4.2 Strenghtening the SSA (again)

We introduce a new SSA for ephemerons:

Steady-state assumption for ephemerons (SSAw): Whenever x'' words of ephemerons are allocated, x'' (other) ephemeron words become unreachable.

We also introduce the Allocation Rate Assumption for ephemerons (ARAw): the ratio of ephemerons to on-heap allocations is a constant e''. We get:

$$S_n'' = e'' S_n \tag{36}$$
$$I_n'' = e'' I_n \tag{37}$$
$$M_n'' = e'' M_n \tag{38}$$
$$W_n'' = e'' W_n \tag{39}$$

We also extend the ARA to the Ephe_sweep phase:

$$W_n' = e' W_n \tag{40}$$

From SSAw, we get:

$$L_{n+1}'' = L_n'' \tag{41}$$

Thus the amount of ephemerons is constant and we'll call it L''.
We have to replace (7) and (20) with:

$$F_{n+1} = M_n + W_n \tag{42}$$
$$F_{n+1}' = M_n' + W_n' \tag{43}$$

We also have:

$$F_{n+1}'' = M_n'' + W_n'' \tag{44}$$
$$G_{n+1}'' = F_n'' + S_n'' + I_n'' \tag{45}$$

Note that, at the beginning of Ephe_mark, the ephemeron list contains not only all the marked ephemerons: live L'' and floating garbage of the next cycle F_{n+1}'' but also all the garbage ephemerons of the next cycle G_{n+1}'' that will be removed from the list in this phase and reclaimed during the Sweep phase of the next cycle. However, for the garbage ephemerons, the GC does no work beyond removing them from the list, unlike the other ones, for which it has to examine each field. Since adding the term G_{n+1}'' to (39) makes the stability condition really hard to study, we will simplify things by not counting them as work, thus incurring another slight degradation of the real-time performance.

Equations (22) and (23) become invalid and are replaced with:

$$sS_n + s'S_n' + s''S_n'' = L + F_n + G_n \tag{46}$$
$$mM_n + m'M_n' + m''M_n'' = L \tag{47}$$
$$wW_n + w'W_n' + w''W_n'' = L'' + F_{n+1}'' \tag{48}$$

4.3   Overhead

The ephemerons cause more GC work, but they do not occupy any memory beyond what is already accounted for, so our definitions of overhead does not need to be changed.

4.4   New parameters

We have five more parameters to choose: s'', m'', w, w', w'', and the goal is still to keep $\dot{Q}_n$ close to β.

As we will see, there is additional overhead associated to the amount of live ephemerons present in the heap. To deal with this, we introduce a new parameter β'' and change our goal from (31) to:

$$\beta \leq \dot{Q} \leq \beta + \beta'' \tag{49}$$

4.5   Choice of I, I', I''

Once again, I, I', I'' are related by ARA and ARAw and (17) and (37), so we have only one variable to set. We'll keep $\dot{J}$ as our primary variable.

4.6   Solving for s, s', s'', m, m', m'', w, w', w''

Assuming m'' constant and using (18), (38), (47) we conclude that M, M', M'' are also constant.

Then, using (48), (44), (40), (39), (42), (43) we get that W, W', W'', F, F', F'' are constant.

4.6.1   Normal mode

We have:

$$S'_n = e' S_n$$
$$S''_n = e'' S_n$$
$$s S_n + s' S'_n + s'' S''_n = L + F + G_n$$
$$G_{n+1} = F + S_n$$

We get:

$$G_{n+1} = F + L/(s + s'e' + s''e'') + F/(s + s'e' + s''e'') + G_n/(s + s'e' + s''e'') \tag{50}$$

This converges on a constant G iff $s + s'e' + s''e'' > 1$. We once again assume $s > 1$, which ensures the condition is true for all values of e' and e''. We also assume that G_n has converged to its limit, which we again call G.

Then S, S', S'', G', G'', O, O', $\dot{O}$, $\dot{Q}$ are also constant.

We have the following:

$$s/m = \sigma \qquad\qquad s'/m' = \sigma$$
$$I = 0 \qquad\qquad I' = 0 \qquad\qquad I'' = 0$$
$$\qquad\qquad M' = e'M \qquad\qquad M'' = e''M$$
$$\qquad\qquad S' = e'S \qquad\qquad S'' = e''S$$
$$F = M + W \qquad\qquad F' = M' + W' \qquad F'' = M'' + W''$$
$$G = F + S + I \qquad\qquad G' = F' + S' + I'$$
$$O = F + G \qquad\qquad O' = F' + G' \qquad \dot{O} = O + O'$$
$$\dot{Q} = \dot{O}/L \qquad sS + s'S' + s''S'' = L + F + G$$
$$mM + m'M' + m''M'' = L$$
$$wW + w'W' + w''W'' = L'' + F''$$

Let us define the following intermediate variables to make the formulas easier to handle:

$$\ddot{s} = s + e's' + e''s'' \tag{51}$$
$$\ddot{m} = m + e'm' + e''m'' \tag{52}$$
$$\ddot{w} = w + e'w' + e''w'' \tag{53}$$
$$\overline{s} = \ddot{s} - 1 \tag{54}$$
$$\overline{w} = \ddot{w} - e'' \tag{55}$$

We get:

$$\overline{s}S = L + 2(M + W) \tag{56}$$
$$\ddot{m}M = L \tag{57}$$
$$\overline{w}W = L'' + e''M \tag{58}$$

Then we can compute $\dot{O}$ as a function of L, L'', e', e'', M, $\overline{s}$, $\ddot{m}$, $\overline{w}$:

$$\dot{O} = (e' + 1)(S + 2(M + W)) \tag{59}$$

$$\frac{\dot{O}}{e' + 1} = \frac{L}{\overline{s}} + \frac{2(M + W)}{\overline{s}} + 2(M + W) \tag{60}$$

$$\frac{\dot{O}}{e' + 1} = \frac{L}{\overline{s}} + 2(M + W)\frac{\ddot{s}}{\overline{s}} \tag{61}$$

$$\frac{\dot{O}}{e' + 1} = \frac{L}{\overline{s}} + \frac{2L\ddot{s}}{\ddot{m}\overline{s}} + \frac{2(L'' + e''M)\ddot{s}}{\overline{w}\overline{s}} \tag{62}$$

$$\frac{\dot{O}}{e' + 1} = \frac{L}{\overline{s}} + \frac{2L\ddot{s}}{\ddot{m}\overline{s}} + \frac{2(L'' + \frac{e''L}{\ddot{m}})\ddot{s}}{\overline{w}\overline{s}} \tag{63}$$

15

We make some simplifying decisions in order to cut down the number of variables: we make $\ddot{m}$ and $\overline{w}$ both proportional to $\ddot{s}$ and introduce a new intermediate variable γ:

$$\ddot{m} = \ddot{s}/\sigma \tag{64}$$

$$\overline{w} = 2\ddot{s}/\gamma \tag{65}$$

Then we get:

$$\frac{\dot{O}}{e'+1} = \frac{L}{\overline{s}} + 2\sigma\frac{L}{\overline{s}} + \gamma\frac{L'' + \frac{e''M}{\ddot{m}}}{\overline{s}} \tag{66}$$

And finally[5]

$$\dot{Q}\overline{s} = (e'+1)(1 + 2\sigma + \gamma(\frac{L''}{L} + \frac{e''}{\ddot{m}})) \tag{67}$$

L'' is the amount of live ephemerons and L is the amount of live data, which includes ephemerons. Thus L''/L is between 0 and 1. Likewise, e'' is between 0 and 1, and we have:

$$\frac{1}{\overline{s}}(e'+1)(1 + 2\sigma) \le \dot{Q} \le \frac{1}{\overline{s}}(e'+1)(1 + 2\sigma + \gamma(\frac{L''}{L} + \frac{e''}{\ddot{m}})) \tag{68}$$

We can take L'' and e'' out of the picture by loosening the upper bound:

$$\frac{1}{\overline{s}}(e'+1)(1 + 2\sigma) \le \dot{Q} \le \frac{1}{\overline{s}}(e'+1)(1 + 2\sigma + \gamma(1 + \frac{1}{m''})) \tag{69}$$

By matching this against (49), we get:

$$\beta = \frac{1}{\overline{s}}(e'+1)(2\sigma + 1) \tag{70}$$

$$\beta'' = \frac{1}{\overline{s}}(e'+1)\gamma(1 + \frac{1}{m''}) \tag{71}$$

This gives :

$$\gamma = \frac{\beta''}{\beta}(2\sigma + 1)\frac{m''}{m'' + 1} \tag{72}$$

Now we have:

$$\overline{s} = \frac{e'+1}{\beta}(2\sigma + 1) \tag{73}$$

$$s - 1 + e's' + e''s'' = (e'+1)(2\sigma + 1)/\beta \tag{74}$$

---

[5]Checked by Rocq, with the additional conditions $\gamma \neq 0$, $\sigma \neq 0$, $e' \geq 0$, $e'' \geq 0$, $L \neq 0$, $s > 0$, $s' \geq 0$, $s'' \geq 0$

This bound is attained in the absence of ephemerons (i.e. when $L'' = e'' = 0$) and we then get:

$$s - 1 + e's' = (e' + 1)(2\sigma + 1)/\beta \tag{75}$$

This is exactly equivalent to (32) and we have to choose the same s and s' as before for this to be true for all values of e'. Thus we remain compatible with ephemeron-free programs.

$$s = 1 + (2\sigma + 1)/\beta \tag{76}$$
$$s' = (2\sigma + 1)/\beta \tag{77}$$

Note that the above does not constrain the value of m'' (or s''). We use (64) and (65) to derive the remaining coefficients:

$$m = s/\sigma \tag{78}$$
$$m' = s'/\sigma \tag{79}$$
$$w = 2s/\gamma \tag{80}$$
$$w' = 2s'/\gamma \tag{81}$$
$$w'' = 2s''/\gamma + 1 \tag{82}$$

Now we have to choose s''. If we set it close to 0, then γ also goes to 0 and w, w' get arbitrarily large. If we set s'' to be very large, m'' will also be large. In the first case, we get large latencies in the Ephe_sweep phase, while in the second case we get large latencies when allocating ephemerons in the other phases.

In order to strike a balance between the two extremes (and to get simpler formulas) we will choose $s'' = \sigma + 1$.

We then have:

$$\gamma = \frac{\beta''}{\beta}(\sigma + 1) \tag{83}$$
$$s'' = \sigma + 1 \tag{84}$$
$$m'' = s''/\sigma \tag{85}$$
$$\tag{86}$$

We can do a smoke check with some reasonable values for the parameters:

$$\beta = 1 \tag{87}$$
$$\beta'' = 0.2 \tag{88}$$
$$\sigma = 3 \tag{89}$$

We get:

$$\gamma = 0.8 \tag{90}$$
$$s = 8 \tag{91}$$
$$s' = 7 \tag{92}$$
$$s'' = 4 \tag{93}$$
$$m = 2.67 \tag{94}$$
$$m' = 2.33 \tag{95}$$
$$m'' = 1.33 \tag{96}$$
$$w = 20 \tag{97}$$
$$w' = 17.5 \tag{98}$$
$$w'' = 11 \tag{99}$$

There is some imbalance between the phases, but there is not much to gain by increasing s'', so the user may have to change β'' and strike a balance between space overhead and latency. Note that β'' is a really loose bound on the space overhead incurred by ephemerons, since it is only approached (not even reached) when all the heap data and allocations are made up of ephemerons.

The right user setting might be β''/β rather than β'' itself, since the formulas only depend on the ratio.

4.6.2   Small-memory mode

We have $G_{n+1} = F + S_n + I_n = F + J$, which is constant. Then S, S', S'', G', G'', O, O', $\dot{O}$, $\dot{Q}$ are also constant.

We have the following:

$$S + I = J \qquad\qquad\qquad S' + I' = J' \qquad\qquad \dot{J} = J + J'$$

$$M' = e'M \qquad\qquad M'' = e''M$$

$$I' = e'I$$

$$W' = e'W \qquad\qquad W'' = e''W$$

$$S' = e'S \qquad\qquad S'' = e''S$$

$$s = \sigma m \qquad\qquad s' = \sigma m' \qquad\qquad s'' = \sigma m''$$

$$w = 2s/\gamma \qquad\qquad w' = 2s'/\gamma \qquad w'' = 2s''/\gamma + 1$$

$$F = M + W \qquad\qquad F' = M' + W' \qquad\qquad F'' = M'' + W''$$

$$G = F + S + I \qquad\qquad G' = F' + S' + I'$$

$$O = F + G \qquad\qquad O' = F' + G' \qquad\qquad \dot{O} = O + O'$$

$$sS + s'S' + s''S'' = L + F + G$$

$$mM + m'M' + m''M'' = L$$

$$wW + w'W' + w''W'' = L'' + F''$$

$$s = 1 + (2\sigma + 1)/\beta \qquad\qquad s' = (2\sigma + 1)/\beta \qquad\qquad s'' = \sigma + 1$$

$$\gamma = (\sigma + 1)\beta''/\beta$$

We can prove[6]

$$\dot{O} \leq (\beta + \beta'')L + \dot{J} \tag{100}$$

Once again, small-memory mode cannot add more than a fixed amount of overhead $\dot{J}$, which is under direct control of the user.

---

[6]Checked by Rocq, with the additional conditions $\sigma > 0$, $\beta > 0$, $\beta'' > 0$, $e' \geq 0$, $e'' \geq 0$, $M \geq 0$, $0 \leq L'' \leq L$.