9-2 | CEk — Control, Environment, Kontinuation

$\langle V, \langle fn, \lambda X.M, k\rangle\rangle \longmapsto_{Ck} \langle M[X \leftarrow V], k\rangle$

$\langle V, \langle fn, \lambda X.M, \quad, k\rangle\rangle \longmapsto_{CEk} \langle M, \mathcal{E}[X \leftarrow V], k\rangle$

$\langle 7, \langle fn, \lambda X.X, mt\rangle\rangle \longmapsto_{Ck} \langle X[X \leftarrow 7], mt\rangle$
$= \langle 7, mt\rangle$

$\mathcal{E} = \bullet$

$\langle X, \mathcal{E}, k\rangle \longmapsto_{CEK} \langle \mathcal{E}(x), \mathcal{E}, k\rangle$ $\qquad \mathcal{E}[X \leftarrow V]$

$\langle (M\ N), \mathcal{E}, k\rangle \longmapsto_{CEK} \langle M, \mathcal{E}, \langle ar, N, \mathcal{E}, k\rangle\rangle$

① $\langle V, \mathcal{E}, \langle ar, N, \mathcal{E}', k\rangle\rangle \longmapsto_{CEK} \langle N, \mathcal{E}', \langle fn, V, \mathcal{E}, k\rangle\rangle$

$\mathcal{E} = \lambda X, Y \ldots [Y \leftarrow 11]$

② $\langle V, \mathcal{E}, \langle fn, \lambda X.M, \mathcal{E}', k\rangle\rangle \longmapsto_{CEK} \langle M, \mathcal{E}'[X \leftarrow V], k\rangle$

$((\lambda X. \bullet)[X \leftarrow 8][X \leftarrow 7])$
$(\lambda X. X)$
$7)\ 8)$

$V_{Ck} = \lambda X.M \mid b$ $\qquad$ ↙ a closure

$V_{CEk} = b \mid \langle clo, \lambda X.M, \mathcal{E}\rangle$

$\langle \lambda X.M, \mathcal{E}, k\rangle \longmapsto_{CEk} \langle \langle clo, \lambda X.M, \mathcal{E}\rangle, \mathcal{E}, k\rangle$

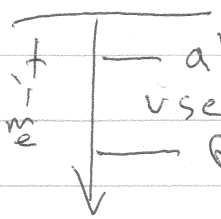
①' $\ldots \longmapsto_{CEk} \langle N, \mathcal{E}', \langle fn, V, k\rangle\rangle$

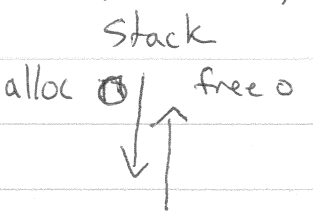
②' $\langle V, \mathcal{E}, \langle fn, \langle clo, \lambda X.M, \mathcal{E}'\rangle, k\rangle\rangle$
$\longmapsto_{CEk} \langle M, \mathcal{E}'[X \leftarrow V], k\rangle$

10-1 | Memory Management

When is a value no longer needed?

rule 1 (code c, env. e, kont k) {
  k' = <fn, c, ↖> → <ar, N, ε, k> free(k)
  return cek(c', e', k')



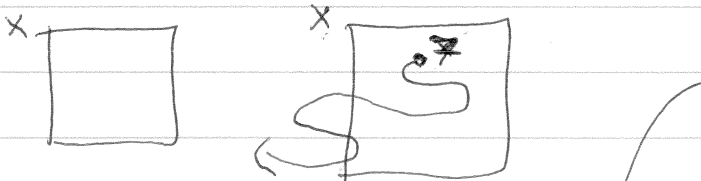


Stack Discipline

Heap ≠ Stack
» malloc » local variables

```
f() {
  x = malloc(7)
  return x }
g() {
  int x = 8
  return &x; }
```

When done (at app level)
When out-of-scope

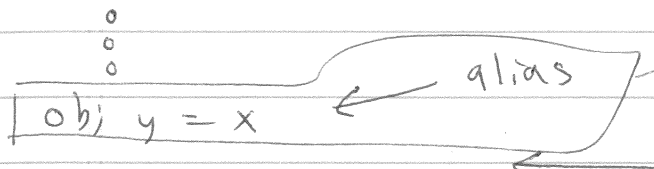


What is a good memory manager?

1' └ simulate infinite memory — CORRECT
  └ frees space when extent is over (ie free when done) — COMPLETE
1 └ don't free stuff we use — SOUNDness
  └ time efficient (don't take time from program)
  └ mem efficient (close to ideal mem size)
  └ low latency (mm runs are short)

10-2 | 100% completeness at low low prices

1. obj x = malloc();



line 87.

88. if (f()) { — is x live (is x's extent ⊂ [88,91])
89. print x — g(12) → h(19, 20)
90. 3 — COMPLETEness depends
91. exit — on knowing what f() can return — TM-Halts(M)

---

malloc() / free() as a person (Jay Method)

- sound — possible, but mistakes happen → NO
- completeness — anything (leak = tiff) (mem hog = Δ>>)
- mem — lg n + lg σ
- time — lg n
- latency — naive → high latency possible/queuing → low latency