The store (Σ) is a linear variable
(it's used exactly once)

Rust & Clean can enforce linearity

CESK - machine
- code
- environment
- store
- continuation

K = mt
| $\langle fn, v, k \rangle$
| $\langle ar, c, \mathcal{E}, k \rangle$
| $\langle op, \ldots \rangle$
new → | $\langle set, \sigma, k \rangle$

$\langle (M\ N), \mathcal{E}, \mathcal{S}, k \rangle$
$\mapsto \langle M, \mathcal{E}, S, \langle ar, N, \mathcal{E}, k \rangle \rangle$

aside on Total-CESK

$\langle (call/cc\ M), \mathcal{E}, S, k \rangle$
$\mapsto \langle (M\ \langle cont, k \rangle), \mathcal{E}, S, k \rangle$
$\langle V, \mathcal{E}, S, \langle ar, N, \mathcal{E}', k \rangle \rangle$
$\mapsto \langle N, \mathcal{E}', S, \langle fn, V, k \rangle \rangle$
$\langle \lambda X.M, \mathcal{E}, S, k \rangle \mapsto \langle \langle clo, \lambda X.M, \mathcal{E} \rangle, \mathcal{E}, S, k \rangle$
$\langle V, \mathcal{E}, S, \langle fn, \langle clo, \lambda X.M, \mathcal{E}' \rangle, k \rangle \rangle$
$\mapsto \langle M, \mathcal{E}'[X \mapsto \sigma], S[\sigma \mapsto V], k \rangle$
where $\sigma \notin dom(S)$

```
(let X = 99 in
  ( (begin (set! X 5)
           add1)
    X ) )
```
⇒ 6

$\langle (set!\ X\ M), \mathcal{E}, S, k \rangle$
$\mapsto \langle M, \mathcal{E}, S, \langle set, \mathcal{E}(X), k \rangle \rangle$
$\langle V, \mathcal{E}, S, \langle set, \sigma, k \rangle \rangle$
$\mapsto \langle void, \mathcal{E}, S[\sigma \mapsto V], k \rangle$

```
interpret : env expr -> value
store-interpret : env store expr -> (value , store)
  [(add lhs rhs)
     (lhsv , lhsΣ) := si (Σ , E , lhs)
     (rhsv , rhsΣ) := si (Σ , lhsΣ , rhs)
     ( (+ lhsv rhsv) , rhsΣ )
```

Rules for C pointers:

free() must be called exactly once

How to Enforce

Every function either

(1) frees the pointers it is given

(2) returns them to caller

(1) → give them to such a function
↳ actually call free

(2) → literally return
↳ read-only (not copying)

```
f () {
  p = malloc (...)
  g (p)
```

case1: g only reads pieces of p
we still "own" p

case2: g frees p
we can't mention it again

Goal: $\langle (+\ 2\ 6),\ [\sigma_1 \mapsto 4,\ \sigma_2 \mapsto 2] \rangle \longmapsto \langle (+\ 2\ 6),\ \emptyset \rangle$

$\langle M,\ \Sigma[\sigma_1 \mapsto V_1] \ldots [\sigma_n \mapsto V_n] \rangle \longmapsto \langle M,\ \Sigma \rangle$ [gc-rule]

$\sigma_1 \ldots \sigma_n \notin LS(\langle M, \Sigma \rangle)$

Live Set

$LS(\langle M, \Sigma \rangle) = LS(M) \cup LS(\Sigma)$

$LS([\sigma_1 \mapsto V_1] \ldots [\sigma_n \mapsto V_n]) = \bigcup_{i=1}^{n} LS(V_i)$

$LS(x) = \emptyset$ $\quad LS(\sigma) = \{\sigma\}$

$LS(b) = \emptyset$ $\quad LS(\lambda X.M) = LS(M)$

$LS(M\ N) = LS(M) \cup LS(N)$ $\quad LS(X := M) = LS(M)$

$\langle C, \mathcal{E}, S, k \rangle \longmapsto \langle C, \mathcal{E}, S', k \rangle$

s.t. $\langle LL(C) \cup LL(\mathcal{E}) \cup LL(k), \emptyset, S \rangle \longrightarrow_{gc} \langle \emptyset, L, S \rangle$

(globals, locals, stack)

$S' = \{ \langle \sigma, S(\sigma) \rangle \mid \sigma \in L \}$

$\longrightarrow_{gc} : \langle Set(\sigma), Set(\sigma), Store \rangle$

- $Set(\sigma)$: gray — live, pointing to live, but not processed
- $Set(\sigma)$: black — or live
- root set

$LL(m) = LS(m)$

$LL(\mathcal{E} = [x_0 \mapsto \sigma_0] \ldots [x_n \mapsto \sigma_n]) = \{\sigma_0, \ldots, \sigma_n\}$

$LL(mt) = \emptyset$

$LL(\langle fn, v, k \rangle) = LL(v) \cup LL(k)$

$LL(\langle ar, c, \mathcal{E}, k \rangle) = LL(c) \cup LL(\mathcal{E}) \cup LL(k)$

$LL(\langle set, \sigma, k \rangle) = LL(k) \cup \{\sigma\}$

$\langle G, B, S \rangle \longrightarrow_{gc} \langle G \setminus \sigma_0 \setminus B \cup LL(S(\sigma_0)), B \cup \{\sigma_0\}, S \rangle$ (subtley wrong)

$\sigma_0 \in G$

$(G \cup LL(S(\sigma_0))) \setminus (B \cup \sigma_0)$

Proofs:

- consistent (deterministic)
- completes/halts (doesn't get trapped in cycles)