10-1 Mutation (data-structure mutation)

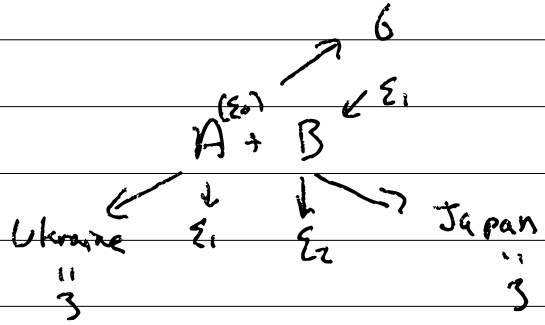
JS	let z=0	Math
x = f(3)	function g() {	x = f(3)
	z++;	
y = f(3)	return	y = f(3)
	a + z; }	
if (x===y)		x = y ?
throw Error		✓

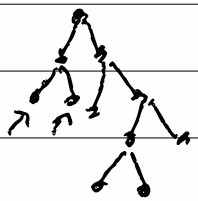
10-2/ Math

$(\Sigma_1, x) = f(\Sigma_0, 3)$ $\quad \Sigma$ – "the store"

$(\Sigma_2, y) = f(\Sigma_1, 3)$

$x = y$?





10-3) $J_5 \Rightarrow J_6$

$v ::= \ldots \mid \sigma$

$e ::= \ldots \mid \text{box } e \mid \text{unbox } e \mid \text{set-box!} \; e \; e$

$\Sigma \mid E[\text{box } v] \rightarrow \Sigma[\sigma \rightarrow v] \mid E[\sigma]$

where $\sigma \notin \Sigma$

$\Sigma \mid E[\text{unbox } \sigma] \rightarrow \Sigma \mid E[\Sigma(\sigma)]$

$\Sigma \mid E[\text{set-box!} \; \sigma \; v] \rightarrow \Sigma[\sigma \rightarrow v] \mid E[v]$

$\emptyset$/

10-4/ let f = let b = box 0 in
λn. set-box! b (+ 1 (unbox b));
(+ n (unbox b))
in (- (f 3) (f 3))

desugar
(x; y) =
let _ = x in y

↙ $\emptyset[\sigma_0 \to 0]$ /
let f = λn. set-box! σ_0 (+ 1 (unbox σ_0));
(+ n (unbox σ_0)) in
(- (f 3) (f 3))

↓ $\emptyset[\sigma_0 \to 0]$ /
(- (set-box! σ_0 (+ 1 (unbox σ_0)); (+ 3 (unbox σ_0)))
(f 3))

⇒ $\emptyset[\sigma_0 \to 1]$ / (- (+ 3 (unbox σ_0)) (f 3))
⇒ $\emptyset[\sigma_0 \to 1]$ / (- 4 (f 3))
⇒ $\emptyset[\sigma_0 \to 2]$ / (- 4 (+ 3 (unbox σ_0)))
⇒ $\emptyset[\sigma_0 \to 2]$ / (- 4 5) → -1

10-5/ v := box v

```
make_box (ob;_t* ) {
  b* = malloc ( ... )
  b -> o = 0;
  return b; }
```

```
box_unbox (b)
  ^el b -> o
box_setbox (b, v)
  b -> o = v;
```

CEK$_3$ ⇒ CESK

st = < e, env, sto, k >

< false, env, sto, kif (env', e$_t$, e$_f$, k) >

↦ < e$_f$, env', sto, k >

10-6) $v = \ldots \mid \text{pair}\ v\ v$

Should we add ...

set-fst! : $(A \times B) \times A \to ()$

```
mpair a b = pair (box a) (box b)
mpair-fst mp = unbox (fst mp)
mpair-fst-set! mp v = set-box (fst mp) v
```

10-7/

["begin"] = ["unit"]

["begin", e] = e

["begin", e_1, e_2, ...] = e_1 ; ["begin", e_2, ...]

(begin0 e_0 e_1 ....)

⇒ (let ans = e_0 in

(begin e_1 ... ans))

(when c e_b ...) ⇒ (if c

(begin e_b ...)

unit)

(unless c e_b ...)

⇒ (when (not c) e_b ...)

10-8/ (while c e_b ...) =>

((λ rec c)
  (when c e_b ... (rec)))

(for (x init limit mc) e_b ...)
=> let x_b = box init in
  while (< (unbox x_b) limit))
    (let x = unbox x_b in e_b ...)
    (set-box! x_b (+(unbox x_b) mc))