19-1

Mutation (State)

$$f(n) = \sum_{i=0}^{n} i$$

```
letrec f = λn. if n==0 then 0
               else n + (f (n-1)) in      } recursive
  f 10
```

```
letrec f' = λn. λa. if n==0 then a                 tail-recursive
                    else f'(n-1, n+a)      }
       f = λn. f' n 0        in
  f 10
```

```
int f (int n) {
①   int sum = 0;
②   for (int while (n > 0) { ③ sum += n; ④ n--; }
⑤   return sum;
}
```

(term in ISWIM)

f 10 = f' 10 0 = f' 9 10 = f' 8 19 = f' 7 27
= f' 6 34 = f' 5 40 = f' 4 45 = f' 3 49 = f' 2 52
= f' 1 54 = f' 0 55 = 55.

C state = loc × n × sum

(not C programs)

(1, 10, 0) → (2, 10, 0) → (3, 10, 10) → (4, 9, 10) → (2, 9, 10)
→ ... →* (2, 5, 40) →* (4, 1, 54) →* (2, 0, 55)
→ (5, 0, 55) ⇒ 55

9-2

Compositional Reasoning

ISWIM: (.... [$(\lambda x.x)\ 5$] .... [M])

Can you answer Q? What does ↑ or → do?

Yes, if I know the bindings ($FV(M) = \emptyset$, definitely)

$FV(M) = \{x, y, z\}$

.... $((\lambda x.$ .... $((\lambda y$ ... $((\lambda z$ .... [M] $)\ z)\ y))\ x)$

C:

```
int x = 5;
   ooo
printf("%d", x);
   ooo
return x;
```

Semantics is a homo-morphism

$f : B \to C,\ g : A \times A \to B$

$f(g(x, y)) = g'(f'(x), f'(y))$

$f' : A \to D \qquad g' : D \times D \to C$

$g(x, y) = (APP\ x\ y)$ "(x y)" $\qquad f = eval$

$f' = eval \qquad g'(x, y) = eval\ (APP\ x\ y)$

```
int x = 1;
return (x++) + (x++);
```

compo. R. ⇒ 4 $\qquad$ actual ⇒ 5

State-ISWIM (Matthias Felleisen)

m := (let ([X m]) m) | .... | (set! X m)

E = (let ([X E]) m) | (let ([X V]) E) | ...
| (set! X E)

E[(let ([X V]) E'[X]] ↦ E[(let ([X V]) E'[V]]

FV(E'[X]) ∋ X

E[(let ([X V]) E'[(set! X V')]]
↦ E[(let ([X V']) E'[V']]

E[(let ([X V]) V')] ↦ E[V']

(λY. X)

M := .... | (set! X m)

P := (let ([X V] ...) M

(let (... [X V] ...) E[(set! X V')]
↦ (let (... [X V'] ...) E[V']

σ = some set distinct from X (σ_1, σ_2, σ_7)

m = ... | σ | (set! σ m) ~~(let ([X V]) m~~

P = (with ([σ V] ...) m)

(with ([σ_1 V_1] ... [σ_n V_n]) E[(λX. m) V])
↦ (with ([σ_1 V_1] ... [σ_n V_n] [σ_{n+1} V]) E[m[X ← σ_{n+1}]]

(with ([σ_1 V_1] ... [σ_x V_x] ... [σ_n V_n]) E[σ_x])
↦ (with ( ) E[V_x])

E[(set σ_x V])

↦ (with ([σ_x V]) E[V])

19-4

CESK: C → code, E → env, S → store, K → continuation

$N, M = X \mid (\lambda X. M) \mid (M\ N) \mid (\text{set! } X\ M) \mid b \mid o^n M \ldots$

$E = \bullet \mid [X \mapsto \sigma]\, E$

$S = \bullet \mid [\sigma \mapsto V]\, S$

$V = b \mid clo(\lambda X. M, E)$

$K = ret \mid fun(N, E, k) \mid arg(V, k) \mid set(\sigma, k) \mid op(o^n, E, V \ldots, M \ldots, k)$

$\langle X, E, S, k \rangle \mapsto \langle S(E(x)), \bullet, S, k \rangle$

$\langle \lambda X. M, E, S, k \rangle \mapsto \langle clo(\lambda X. M, E), \bullet, S, k \rangle$

$\langle M\ N, E, S, k \rangle \mapsto \langle M, E, S, fun(N, E, k) \rangle$

$\langle V, E, S, fun(N, E', k) \rangle \mapsto \langle N, E', S, arg(V, k) \rangle$

$\langle V, E, S, arg(clo(\lambda X. M, E'), k) \rangle$
$\quad \mapsto \langle M, E'[X \mapsto \sigma'], S[\sigma' \mapsto V], k \rangle$

$\langle \text{set! } X\ M, E, S, k \rangle \mapsto \langle M, E, S, set(E(x), k) \rangle$

$\langle V, E, S, set(\sigma, k) \rangle \mapsto \langle V, E, S[\sigma \mapsto V], k \rangle$

$M = \ldots \mid snapshot \mid restore\ M \qquad K = \ldots \mid restore(k)$

$V = sto(S) \mid \ldots$

$\langle snapshot, E, S, k \rangle \mapsto \langle sto(S), E, S, k \rangle$

$\langle sto(S'), E, S, retore(k) \rangle \mapsto \langle 1, E, S', k \rangle$

$\langle restore\ M, E, S, k \rangle \mapsto \langle M, E, S, restore(k) \rangle$