8-1

CK-machine (C = code string, K =)

St = <M, K>

tr(M) = <M, ret>

untr(<M, ret>) = V

K = ret → ▨

| ar(N, K) → (▨ N)

| fn(V, K) → (V ▨)

| pr(o^n, <V...>, <N...>, K) → (o^n V... ▨ N...)

K is a data structure, like a singly-linked list

represents a context (stored inside-out)

rep : K → E

rep(ret) = ▨

rep(ar(N, K)) = rep(K)[(▨ N)]

rep(fn(V, K)) = rep(K)[(V ▨)]

rep(pr(o^n, <V...>, <N...>, K)) = rep(K)[(o^n V... ▨ N...)]

C: blank(null, K)

improper function calls ↓

proper function calls ↓

1. <(M N), K> $\mapsto_{ck}$ <M, ar(N, K)>
2. <V, ar(N, K)> $\mapsto_{ck}$ <N, fn(V, K)>
3. <V, fn(λX.M, K)> $\mapsto_{ck}$ <M[X←V], K>

C: <M[X←V], blank(null, K)>

/ Scm / pascal / py :

4. <(o M N...), K> $\mapsto_{ck}$ <M, pr(o, <>, <N...>, K)>
5. <V, pr(o, <U...>, <M N...>, K)>

$\mapsto_{ck}$ <M, pr(o, <U... V>, <N...>, K)>

6. <b_n, pr(o^n, <b_1 ... b_{n-1}>, <>, K)>

$\mapsto_{ck}$ <δ(o^n, b_1 ... b_n), K>

((λX. (+ (+ 1 2) X)) 3) $\xrightarrow{tr}$ <((λX. (+ (+ 1 2) X)) 3), ret>

$\xrightarrow{1}$ <(λX. (+ (+ 1 2) X)), ar(3, ret)>

$\xrightarrow{2}$ <3, fn((λX. (+ (+ 1 2) X)), ret)>

$\xrightarrow{3}$ <(+ (+ 1 2) 3), ret> $\xrightarrow{4}$ <(+ 1 2), pr(+, <>, <3>, ret)>

$\xrightarrow{4}$ <1, pr(+, <>, <2>, pr(+, <>, <3>, ret))>

$\xrightarrow{5}$ <2, pr(+, <1>, <>, pr(+, <>, <3>, ret))>

$\xrightarrow{6}$ <3, pr(+, <>, <3>, ret)>

$\xrightarrow{5}$ <3, pr(+, <3>, <>, ret)> $\xrightarrow{6}$ <6, ret> $\xrightarrow{un}$ 6

8-2

$E[m] \longmapsto\!\!\!\!\rightarrow_v E[v]$

iff $\langle m, E\rangle \longmapsto\!\!\!\!\rightarrow_{scc} \langle v, E\rangle$

iff $\langle m, rep^{-1}(E)\rangle \longmapsto\!\!\!\!\rightarrow_{ck} \langle v, rep^{-1}(E)\rangle$

What does K stand for? Stack*

K represents the context and is stored backwards for efficiently

context is our future work

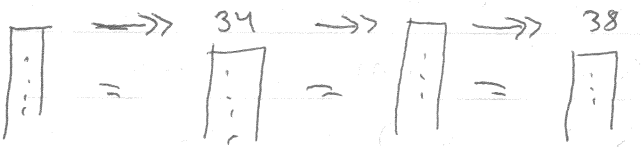
O(1) access to the top and we don't care about rest (we only look at the top)

stack

function call stack

records functions that haven't returned yet (your future)

```
int f(){...
30: g()
34: h()
38: ret x; }
```




```
int f(int x) {
2   int y = 7;
4   g();
6   return x+y;
}
8:  f(10);
12:
```

	s0	s1	s2	s3
6	7	10	12	⊥
7	7	10	12	⊥
7	7	10	12	⊥
17	7	⊥	⊥	⊥

"x+y" => "s0 + s1"

```
movq rax, rsp(-8)
addq rax, rsp(-16)
ret
```

(continuation)

The stack is exactly the K-thing! K = Kontinuation

$\Omega = ((\lambda x.(xx))\ (\lambda x.(xx)))$ finite-space

CK: $\langle (w\ w), ret\rangle \mapsto \langle w, ar(w, ret)\rangle \Rightarrow \langle w, fn(w, ret)\rangle$ (over)

Over CK: $b^3 \Leftarrow b^2 \Leftarrow \Leftarrow \langle (w\ w), blank(null, ret)\rangle$

infinite space