Y-V $CC_0$ st = <e, E>

<Fake, E[if ■ et ef]> ↦ <ef, E>

E := ■ | if E e e | (v ... E e ...)

↓ ↓ ↓

Hole() IfC(E, e, e) AppC([List(v), E, List(e)])

↓

AppC → IfC → AppC → IfC ~~→~~ Hole

↓

AppC → IfC → AppC → Hole

Stack

4-2/ $k := k_{ret} \mid k_{if}\ e\ e\ k \mid k_{app}\ \vec{v}\ \vec{e}\ k$

CK  $st = \langle e, k \rangle$

$inject\ e = \langle e, k_{ret} \rangle$

$extract\ \langle v, k_{ret} \rangle = v$

1 $\langle if\ e_c\ e_t\ e_f, k \rangle \mapsto \langle e_c, k_{if}\ e_t\ e_f\ k \rangle$

2 $\langle false, k_{if}\ e_t\ e_f\ k \rangle \mapsto \langle e_f, k \rangle$

3 $\langle v, k_{if}\ e_t\ e_f\ k \rangle \mapsto \langle e_t, k \rangle$

4 $\langle e_0\ e_m \ldots, k \rangle \mapsto \langle e_0, k_{app}\ ()\ (e_m \ldots)\ k \rangle$

5 $\langle v_1, k_{app}\ (v_0 \ldots)\ (e_0\ e_m \ldots)\ k \rangle$
$\mapsto \langle e_0, k_{app}\ (v_1\ v_0 \ldots)\ (e_m \ldots)\ k \rangle$

6 $\langle v_n, k_{app}\ (v_0 \ldots)\ ()\ k \rangle \mapsto \langle \delta(rev(v_n\ v_0 \ldots)), k \rangle$

4-3/ (+ 1 (x 2 3)) ↓ inject

< (+ 1 (x 2 3)), kret > ↓ 4

< +, kapp () (1 (x 2 3)) kret > ↓ 5

< 1, kapp (+) ((x 2 3)) kret > ↓ 5

< (x 2 3), kapp (1 +) () kret > ↓ 4

< x, kapp () (2 3) (kapp (1 +) () kret) > ↓ 5

< 2, kapp (x) (3) (kapp (1 +) () kret) > ↓ 5

< 3, kapp (2 x) () (kapp (1 +) () kret) > ↓ 6

< δ(x,2,3) = 6, kapp (1 +) () kret > ↓ 6

< δ(+,1,6) = 7, kret > ↓ extract

7

4-4/

The K component is like a stack where the values are either ifs or apps

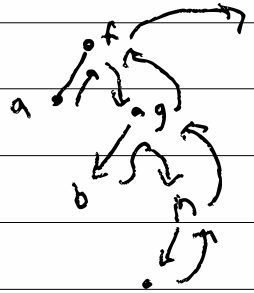
$K$ = Stack < Frames >

Frame = If F (e, e)
| App ($\vec{v}$, $\vec{e}$)

K is called a "continuation"

4-5/ Novices think the stack is the PAST of a computation

f
g
h
→




```
f()
    x = a()
    x = g()
    ret x+1
g()
    y = b()
    y = h()
    ret y+2
h()
    ...
```

4-6/

$Ck(e) \xrightarrow{\text{return}} v$

$c = e \quad , \quad k = k_{ret}()$

while (1) {

case c of

if $e_c$ $e_t$ $e_f$ →

$c = e_c \quad k = k_{if}(e_t, e_f, k)$

app $(e_0 : e_s)$ →

$c = e_0 \quad k = k_{app}([], e_s, k)$

v → case k of

$k_{ret}$ → return v

$k_{if}$ $e_t$ $e_f$ $k'$ →

$c = v\,?\,e_t : e_f \quad , \quad k = k'$

$k_{app}$ vs $e_s$ $k'$ →

case $e_s$ of

[] → $c = \delta(rev(v:vs)$ $k = k'$

$(e_0 : e_s)$ →

$c = e_0 \quad k = k_{app}((v:vs), e_s, k')$

4-7/

```
struct code {
    int type;   // 0 for nums, 1 for bool
                   2 for if, 3 for cas
                   app
union {
    struct num {
        int n; }
    struct if {
        code *c, *t, *f }
```

4-8

Ck in C (or C++) — LL

in Racket, Python, JS, ... — HL

desugar : HL

HL

check "(+ 1 (x 2 3))" "7" → true

→ error o.w

x.c

```
#include "l1.c"
int main() {
  ck(make_app(
    make_prim(PLUS),
    make_num(1),
    make_app(
      make_prim(MULT),
      make_num(2),
      make_num(3))))}
```

```
cc x.c -o x.bin
./x.bin > x.out
read x.out ....
  "7"
```