13-1/ First-class continuation

e = ... | callcc e

E[callcc v] → E[v (λ (x) abort E[x])]

```
E = (+ 1 ●)
(+ 1 (callcc (λk. (+ 2 (k 3)))))
(+ 1 ((λk. (+2 (k 3)))
        (λx. abort (+ 1 x))))
(+ 1 (+ 2 ((λx. abort (+ 1 x)) 3)))
(+ 1 (+ 2 (abort (+ 1 3))))
(+ 1 3)   →  4
```

13-2) (let/cc k e) => callcc (λ (k) e)

```
(+ 1 (let/cc k (+ 2 (k 3))))
```

C)
```
int f(int x) {
    if (x < 0) return -1;
    if (x == 3) return 0;
    return x*2; }
```

```
(λ (x). let/cc return.
  (when (x < 0) (return -1))
  (when (x = 3) (return 0))
  (* x 2)))
```

J)
```
λ (x).
  (if (x < 0) -1
     (if (x = 3) 0
        (x * 2)))
```

```
desugar ["λ", f, xs, b]
  = ~~let~~ λ f xs
      (let/cc return
        de [b]
```

13-3/

```
while c e_b =
((λ rec () . when c e_b (rec)))

while c e_b =
((λ rec ().
    (let/cc break.
       (when c
            (let/cc continue
                 e_b)
            (rec)))))
```

13-4) return, break, continue — CONTROL constructs

callcc — first-class control

$CEK_4 \to C\underline{E}K_\sigma$ $\qquad v = \ldots \mid \text{kont } k$

$\qquad\qquad\qquad\qquad k = \ldots \mid \text{kcallcc } k$

$\langle \text{callcc } e, env, k \rangle \mapsto \langle e, env, \text{kcallcc } k \rangle$

$\langle v, -, \text{kcallcc } k \rangle \mapsto \langle \text{kont } k, -,$
$\qquad\qquad \text{kapp } (v) \; - \; () \; k \rangle$

$\langle v, -, \text{kapp } (\text{kont } k') \; - \; () \; k \rangle$
$\quad \mapsto \langle v, -, k' \rangle$

$$13\text{-}5/\ \langle\ (+1\ \ (\text{callcc}\ \ \overbrace{(\lambda\ (k)\ \ (+2\ \ (k\ 3)))}^{A})),\ \emptyset,\ \text{kret}\ \rangle$$

$$\langle\ \text{callcc}\ A,\ \emptyset,\ \underbrace{\text{kapp}\ \ (+\ \ 1)\ \ \emptyset\ \ ()\ \ \text{kret}}_{K_1}\ \rangle$$

$$\langle\ A,\ \emptyset,\ \text{kcallcc}\ \ K_1\ \rangle$$

$$\langle\ \text{clo}(A,\emptyset),\ \emptyset,\ \text{kcallcc}\ \ K_1\ \rangle$$

$$\langle\ (+\ 2\ \ (k\ 3)),\ \ \emptyset[k \mapsto \text{kont}\ \ K_1],\ K_1\ \rangle$$

$$\langle\ k\ \ 3,\ \ \text{''}\ \ ,\ \text{kapp}\ \ (+\ \ 2)\ \emptyset\ \ ()\ \ \ K_1\ \rangle$$

$$\langle\ 3,\ \ \text{''},\ \ \text{kapp}\ \ \ (\text{kont}\ \ K_1)\ \ \emptyset\ \ ()$$

$$\text{kapp}\ \ (+\ \ 2)\ \ \emptyset\ \ \ ()\ \ \ K_1\ \rangle$$

$$\langle\ 3,\ \ \emptyset\ ,\ \ K_1\ \rangle$$

$$\langle\ 3,\ \emptyset\ ,\ \ \text{kapp}\ \ (+\ 1)\ \ \emptyset\ \ ()\ \ \text{kret}\ \rangle$$

$$\langle\ 4,\ \emptyset\ ,\ \ \text{kret}\ \rangle\ \longrightarrow\ 4$$

```
13-6) (define last-handler
        (box (λ (x) (abort x))))
      (define throw
        (λ (v) ((unbox last-handler) v)))
      desugar ["try", eb, "catch", ec]
        = trycatch* (λ () eb) ec

  trycatch* := (λ (body newhandler)
    (let oldh = unbox newhandler in
     (let/cc here (set-box! last-handler
                     (λ (x) (set-box! last-handler oldh)
                            (here (newhandler x)))))
        (begin0 (body) (set-box! last-handler oldh)))))
```

13-7) generator

```
let f = make-generator
          (λ (yield)   (yield 0)
                       (yield 2)
                       (yield 4))
    in (+ (f) (f))
  → 2

let evens = make-generator
    (λ (yield)   ((λ rec (i). (yield i) (rec (+ i 2)))
                   0))
in   (evens) → 0        (evens) → 6        JS/Python
     (evers) → 2
     (evens) → 4
```

13-8/

```
make-generator := (λ f.
  let f-in-progress = box( inl false) in
  (λ () (let/cc local
     (case (unbox f-in-progress) of
        inl _ → let current = box local in
          (f (λ (ans)
             (set-box! current
               (let/cc next. (set-box! f-in-progress
                                      (inr next))
               ((unbox current) ans)))))
       inr resume →
         resume local))))
```

13-9) $v ::= \ldots \mid \text{Kont } k$

$\langle v, -, \text{Kcallc } k \rangle$

$\quad \mapsto \langle \text{Kont } k, \emptyset, \text{Kapp } (v)\ \emptyset\ ()\ k \rangle$

$J_{10} \longrightarrow J_8$

(call/cc)   (no call/cc)   CPS - continuation passing style

$f(x) \longrightarrow f(x, k)$

$x+1 \qquad k(x+1)$

$\text{call/cc} ::= \lambda v k.\ v\ k\ k$