Small-step review part 1

Evaluation contexts define **where** to reduce

$C ::= (\text{Add } C\ e) \mid (\text{Add } v\ C) \mid [\,] \mid \ldots$

Reduction rules define **how** to reduce

$$\text{Step-add}\ \frac{}{(\text{Add } (\text{Num } n1)\ (\text{Num } n2)) \longrightarrow (\text{Num } n1+n2)}$$

$$\text{Step-context}\ \frac{e1 \longrightarrow e2}{C[e1] \longrightarrow C[e2]}$$

What is **not** in C:

Going "into/under a binder"

(Lam x C) is **not** an evaluation context.
(Rec u C) " " " " " .
(Let x v C) . " " " " " .

$$(\text{Rec } u\ e) \longrightarrow [(\text{Rec } u\ e)/u]\,e$$

Small-step review part 2

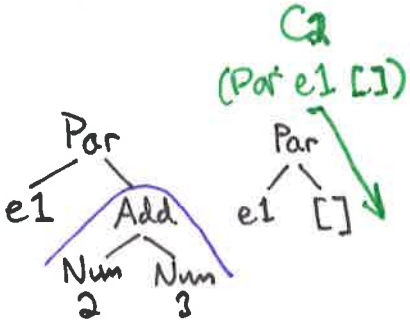
$$\frac{}{(\text{Par } v1\ e2) \longrightarrow v1} \qquad \frac{}{(\text{Par } e1\ v2) \longrightarrow v2}$$

C ::= :
| (Par C e)
| (Par e C)

One possible step:

$$\frac{(\text{Add } (\text{Num } 2)\ (\text{Num } 3)) \longrightarrow (\text{Num } 5)}{(\text{Par } e1\ (\text{Add } (\text{Num } 2)\ (\text{Num } 3))) \longrightarrow (\text{Par } e1\ (\text{Num } 5))}$$

Let C2 = (Par e1 [ ]) = C2



going into 2nd expression in Par

Another possible step

$$\frac{(\text{Add } (\text{Num } 1)\ (\text{Num } 1)) \longrightarrow (\text{Num } 2)}{(\text{Par } (\text{Add } (\text{Num } 1)\ (\text{Num } 1))\ \underbrace{(\text{Add } (\text{Num } 2)\ (\text{Num } 3))}_{e2}) \longrightarrow (\text{Par } (\text{Num } 2)\ e2)}$$

Let C1 = (Par [ ] e2)

C2 does not care what e1 is

C1 does not care what e2 is

Small-step review, part 3

Two derivations are possible (one with C1, one with C2):

overlapping possibilities.

e1 = (Add (Lam x (Num 5)) (Num 1))
e2 = (Add (Num 2) (Num 3))

(Par e1 e2)

Many of submissions claimed this was stuck:

```
[Par (e1 e2)
 (if (= (random 2) 0)
     (step e1)
     (step e2))]
```

incorrect

But...

```
(if (= (random 2) 0)
    (try to step e1
     if #f, step e2)
    (try to step e2
     if #f, step e1))
```

correct

(or (step e1) (step e2))

(Par (Add (Lam x (Num 5)) (Num 1)) (Add (Num 2) (Num 3)))

→ (Par (Add (Lam x (Num 5)) (Num 1)) (Num 5))

→ (Num 5)

Closures

1. Not closures, not environments

```
(Let x (Num 1)
     (Let f (Lam y (Add (Num 10) (Id x)))
          (App (Id f) (Num 0))))
```

Concrete syntax:

```
(Let x 1 (Let f (Lam y (+ 10 x)) (App f 0)))
```

$\longrightarrow$ $[1/x]$ `(Let f (Lam y (+ 10 1)) (App f 0))`

$=$ `(Let f (Lam y (+ 10 1)) (App f 0))`

$\longrightarrow$ $[$`(Lam y (+ 10 1))`$/f]$ `(App f 0)`

$=$ `(App (Lam y (+ 10 1)) 0)`

$\longrightarrow$ $[0/y]$ `(+ 10 1)`

$=$ `(+ 10 1)`

Every Id in every step is bound.

Closures (Clo (x=1, ∅) (Lam y (+10 x)))

x=1

[1/x]

(Let x 1 (Let f (Lam y (+10 x)) (App f 0)))

x=-5

envfx = f=(Lam y (+10 x)), x=1, ∅ [(Lam y (+10 x)) / f]

but I also have [1/x]

[(Lam y (+10 1)) / f]

(Let x 1 (Let f (Lam y (+10 x)) (Let x -5 (App f 0))))

envfx ⊢ (App f 0) ⇓