compatible closure of beta as:

$$C[(\lambda X.M)\ V] \to_v C[M[X \leftarrow V]]$$

beta rule

$$C[\ ] \to_v C[\ ]$$

$(+\ 2\ (+\ 3\ 4)) \to_v (+\ 2\ 7)$

$C = (+\ 2\ \blacksquare)$

$C[(+\ 3\ 4)] \to C[7]$

$C = \blacksquare$
| $(\lambda X.\ C)$
| $(C\ M)$
| $(M\ C)$
| $(o^n\ M \ldots C\ M \ldots)$

$P = (+\ (+\ 1\ 2)\ (+\ 3\ 4))$

$C_1 = (+\ \blacksquare\ (+\ 3\ 4))$

$C_2 = (+\ (+\ 1\ 2)\ \blacksquare)$

$P = C_1[(+\ 1\ 2)] \Rightarrow C[3]$
$\quad\ C_2[(+\ 3\ 4)] \Rightarrow C[7]$

$(\lambda X.\ (+\ 1\ 2)) = (\lambda X.\ \blacksquare)[(+\ 1\ 2)]$

$V = b$
| $(\lambda X.M)$

- Unique Decomposition Theorem (FALSE for C)

$\forall M.\ M = V$ or $\exists$ a unique C s.t.
$M = C[(V_1\ V_2)]$ or $M = C[o^n\ V_1 \ldots V_n]$

Evaluation contexts satisfy unique decomp.

$$C = \blacksquare \mid (\lambda X.\, C) \mid (C\ M) \mid (M\ C) \mid (o^n\ M \ldots C\ M \ldots)$$

$$E = \blacksquare \mid \underline{\qquad} \mid (E\ M) \mid (V\ E) \mid (o^n\ V \ldots E\ M \ldots)$$

Eval-ctx-based semantics $\longmapsto_v$

$$E[(\lambda X.M)\ V] \longmapsto_v E[M[X \leftarrow V]]$$

$$E[(o^n\ V_1 \ldots V_n)] \longmapsto_v E[\delta(o^n, V_1, \ldots, V_n)]$$

The Standard Reduction (SR)

$$E[M] \longmapsto_v E[M'] \quad \text{if } M \vee M'$$

$\longmapsto\!\!\!\!\rightarrow_v$ (refl-transitive closure of $\longmapsto_v$)

$$eval_v^S(M) = \begin{cases} b & \text{if } M \longmapsto\!\!\!\!\rightarrow_v b \\ 'fun & \text{if } M \longmapsto\!\!\!\!\rightarrow_v \lambda X.N \end{cases}$$

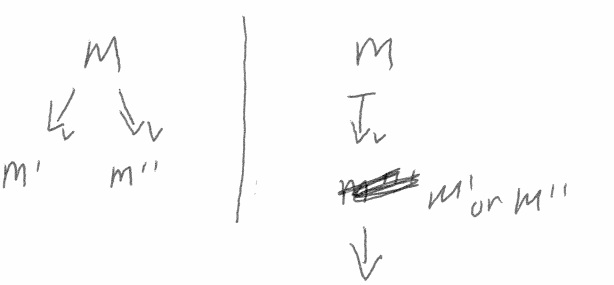
$\underbrace{eval_v^S}_{SR} = \underbrace{eval_v}_{\text{compatible closure}}$?

part 1. lamddas

part 2 non-det

$(\lambda X.M) \longrightarrow_v (\lambda X.M')$ if $M \to_v M'$

BUT $(\lambda X.M) \not\longmapsto\!\!\!\!\rightarrow_v$



Church-Rosser Theorem

```
Program SR (Program p) {

rec [ Program p' = SR1 (p);
      if (p') { return SR(p'); }
      else    { return p; } }

iter [ while (Program p' = SR1 (p);)
         p = p'
       ret p }
```

```
class FilledContext {
  Context E;
  Program m;
}
```

```
Program⊥ SR1 (Program p) {                    // FilledContext⊥ / NULL
  FilledContext⊥ fc = split(p);               // p = E[m] or p = V
  if (!fc) { return NULL; }                   // done because p is a value
  Program⊥ m' = V(fc.m);                      // NULL if m is stuck
  if(!m') { ret NULL; }                       // error   m = (5 5)
  ret fill(fc.E, m');                         // E[m']
}
```

```
split : Program -> FilledContext⊥
V     : Program -> Program⊥
fill  : Context x Program -> Program
```

```
P⊥ V (P p) {
  if (p is app ∧ p.left is lambda) {
        // p = ((λX.m) V)
    ret p.left.body.subst(p.right)
          // m[X ← V]
  } else if (p is a primapp) {
    case (p.op) { ... }
  } else { ret NULL; }
```

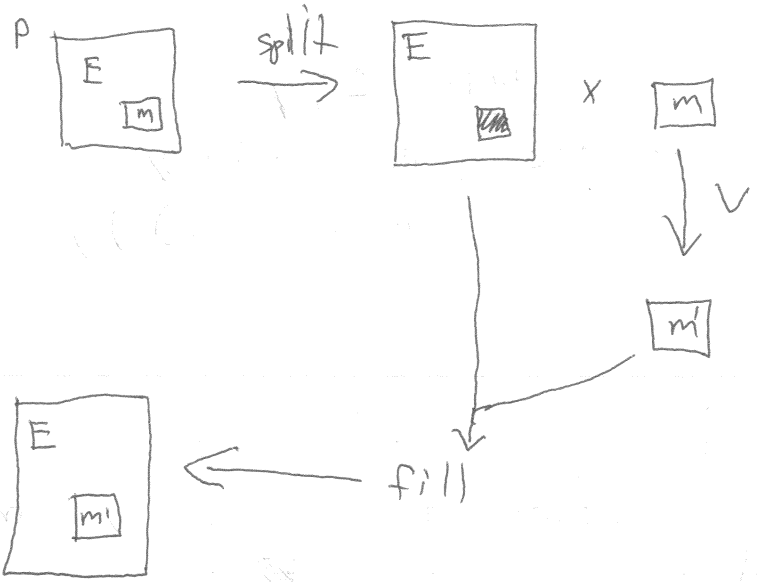
6-4/

split: program → Filled Context⊥

```
FC⊥ split(p p) {
  if (p is a value) {
     ret NULL;
  } else if( p is an app ) {
     if (p.left a value) {
        if (p.right a value) {
           ret ( [redacted] , p )
        } else {
      E[m] = split(p.right)
           ret ((p.left E) , m)
        }
     } else {
        ret E[m] = split(p.left);
        [ ret ((E p.right) , m)
     }   newFC( new OnLeft(E, p.right), m)
  } else if (p is prim). {
     ...
  }
}
```

6-5/

fill : Context x Program => Program



Context an Interface

fill (▨ , p) = p ▨ = new Hole();

fill (onLeft(E, m) , p) = (E m) = new OnLEFT(E,m)
app (fill(E,p) , m)

fill (onRight (V, E) , p) = (V E) = new OnR(V, E)
app (V , fill (E, p))

```
class OnLeft implements Context {
  public:
    Context E; Program m;
    OnLeft (C E, P m) {
       this.E = E;  this.M = M;
    }
    Program fill (Program p) {
       return new Application ( E.fill(p) , M ) ;
    }
}
```

6-6

"(+ 1 (+ 2 3))"

parse →

```
new Add( new Constant(1),
         new Add( new C(2),
                  new C(3) ))
```