(plus one one)

⇒ (λn.λm.λf.λz. (n f) ((m f) z)) one one

⇒ (λm.λf.λz. (one f) ((m f) z)) one

⇒ (λf.λz. (one f) ((one f) z))

⇒ (λf.λz. (λa. f a) ((λb. f b) z))

⇒ (λf.λz. (λa. f a) (f z))

⇒ (λf.λz. f (f z)) := two

one f

⇒ (λf.λz. f z) f

⇒ (λz. f z)

prescriptive – how languages ought to be
descriptive – how languages are

λ-calculus add stuff
ISWIM – if you see what I mean

M, N, L, K = X — variables
| (λX. M) — functions (not abstractions)
| (M N) — application (i.e. function call)
| b — constants from set B
| (O^n M ... n) — n-arity primitive fuctions from set O_n

B = {true, false} ∪ Naturals
O_0 = { rand }
O_1 = { not, negate }
O_2 = { add, sub, div, mul, and, or, ==, lt, gt }

((two (λn. add 1 n)) 0) ⇒ 2
add 1 (add 1 0) ⇒⇒

$$X[X \leftarrow N] = N$$
$$Y[X \leftarrow N] = Y$$
$$(M\ N)[X \leftarrow L] = (M[X \leftarrow L]\ \ N[X \leftarrow L])$$

$$(\lambda X.\, M)[X \leftarrow N] = (\lambda X.\, M)$$
$$(\lambda Y.\, M)[X \leftarrow N] = (\lambda Z.\, M[Y \leftarrow Z][X \leftarrow N])$$
$$(Z \notin M \text{ or } N)$$

$$b[X \leftarrow N] = b$$ constants are opaque

$$(o^n\ M_1 \ldots M_n)[X \leftarrow N] = (o^n\ M_1[X \leftarrow N] \ldots M_n[X \leftarrow N])$$
primitives do not introduce names

ISWIM semantics.

- No eta rule $(\lambda X.\, M\ X) \not\eta M$
  $(\lambda X.\, (5\ X)) \neq 5$
  There's more than just functions
- No alpha (we don't need it)
- Beta is different

λ :	$(\lambda X.\, M)\ N$	β	$M[X \leftarrow N]$
ISWIM :	$(\lambda X.\, M)\ V$	β_V	$M[X \leftarrow V]$

new category of "values"

$$V = b \mid (\lambda X.\, M)$$

$((\lambda x.\ add\ x\ x)\ (add\ 5\ 10))$

$add\ (add\ 5\ 10)\ (add\ 5\ 10)$

$add\ 15\ (add\ 5\ 10)$

ISWIM

$((\lambda x.\ add\ x\ x)\ 15)$

$add\ 15\ 15$

30

4-3/

```
int f (int x) {
    return x+x;
}
int main () {
  int x = 1;
ret f (++x);
}
```

$\Rightarrow$ ret (++x) + (++x) $\Rightarrow$ 5

(++x) $\Rightarrow$ 2 ; (++x) $\Rightarrow$ 3

ISWIM semantics
- No rule to turn b into anything
- A rule for primitives

Δ

$(o^n \; V_1 \; \ldots \; V_n) \xrightarrow{\Delta} \delta(o^n, (V_1, \ldots, V_n))$

δ : Operation $\times$ $\overrightarrow{\text{Values}} \rightarrow M$

δ(~~neg~~, <true>) = ~~false~~ false)
δ(not, <false>) = true
δ(add, < n∈Nat, m∈Nat >) = n+m
δ(zero?, <0>) = $\lambda t.\lambda f.\, t$
δ(zero?, < n∈Nat s.t. n≠0>) = $\lambda t.\lambda f.\, f$

δ is partial function
δ(add, < $\lambda x.x$, 10 >) = ⊗

$v = \beta_v \cup \Delta$
$\rightarrow_v$ = compatible closure
$\rightarrow\!\!>_v$ = refl + trans closure
$=_v$ = symmetric closure

4-4/

$eval_v(M) \Rightarrow answers$

$$= \begin{cases} b & \text{if } M =_v b \\ \text{'fun} & \text{if } M =_v \lambda X.N \end{cases}$$

$answers = B \cup \{\text{'fun}\}$

Is $eval_v$ a function? (is it the case that
forall M, forall A, B
$eval_v(M) = A$
$\wedge\ eval_v(M) = B$
$\rightarrow\ A = B$)

Is $eval_v$ partial or total?
some M have no A s.t. $eval_v(M) = A$ → all of them do

- $M = X$ — free variable
- $M = (5\ 6)$
- $(b\ N)$ — called a constant as a function
- $M = (add\ (\lambda x.x)\ 10)$ — illegal primitive (i.e. δ is partial)
- "stuck terms" (code for "error")

or never stop running ("diverge")
$M = \Omega$

Recursive functions are still possible but different details (pg 50)

β theorem: If $M =_v V$ for some V, then
$(\lambda X.N)\ M =_v N[X \leftarrow M]$