"Programming" w/ the λ-calculus

Multiple Arguments

$\lambda x.\ \lambda y.\ \lambda z.\ x$ (fun x → fun y → fun z → x = fun x y z → x)

This approach is called currying after Haskell Curry (though he didn't invent it)

Booleans (Church Booleans)

if b then e_1 else e_2
if true then e_1 else $e_2 \equiv e_1$
if false then e_1 else $e_2 \equiv e_2$

Can only use application
Try: if b then e_1 else $e_2 \triangleq b\ e_1\ e_2$
What are true and false?
true e_1 e_2 must $\equiv e_1$ $\Rightarrow$ true $\triangleq \lambda t.\ \lambda f.\ t$
false e_1 e_2 must $\equiv e_2$ $\Rightarrow$ false $\triangleq \lambda t.\ \lambda f.\ f$

Pairs

fst $(x, y) \equiv x$ snd $(x, y) \equiv y$

$(x, y) \triangleq \lambda s.\ s\ x\ y$

↑ Which one do you want?

fst $\triangleq \lambda x.\ \lambda y.\ x$
snd $\triangleq \lambda x.\ \lambda y.\ y$

Recursion

$\lambda x. xx$ - Applies x to itself.
Interesting.

$(\lambda x. xx)(\lambda x. xx)$
$\mapsto (\lambda x. xx)(\lambda x. xx)$
$\mapsto (\lambda x. xx)(\lambda x. xx)$
$\mapsto \ldots$

Recursion, Part 2

Let's say we have numbers (yeah, those can be programmed in λ too)

$fact \triangleq \lambda n.$ if $n=0$ then 1 else $n *$ fact $(n-1)$

oops, not defined

no "let rec" in λ-calculus

Let's take another fact function as an argument

$fact' \triangleq \lambda f\, \lambda n.$ if $n=0$ then 1 else $n * f\,(n-1)$

$fact \triangleq fact'\, fact$

oops, same problem

Fixed point of a function f = value x such that $fx = x$

Fixed point combinator: A function "fix"

such that $fix\, f \equiv f\,(fix\, f)$

Let's say we have a "fix"

$fix\, fact' \equiv fact'\,(fix\, fact')$

$\equiv fact'\,(fact'\,(fix\, fact'))$

Is this good enough?

$fact'\,(fact'\,(fix\, fact'))$

$\equiv \lambda n.$ if $n=0$ then 1 else $n * \underbrace{fact'\,(fix\, fact')}_{\equiv fact'}\,(n-1)$

Looks good

$Y = \lambda f.(\lambda x.\, f(x\,x))(\lambda x.\, f(x\,x))$ - most famous fixed pt. comb.

$Y\, f =_\beta (\lambda x.\, f(x\,x))(\lambda x.\, f(x\,x))$

$=_\beta f((\lambda x.\, f(x\,x))(\lambda x.\, f(x\,x)))$

$= f\,(Y\, f)$ ✓