# Lambda Calculus II

4190.310
Programming Languages
Spring 2014

Lecture 03

Reading assignments: Chapter 10

# Call-by-value

- Given an application ($\lambda$x.t) s, s is a value before calling the function
  - Reduce leftmost-outermost redex where argument is a value
  - Given an application st, we first evaluate s until it is a value then we evaluate t until it is a value
    - Then, apply the function to the value
  - Value
    - $\lambda$-abstraction (pure lambda calculus)
    - An expression that cannot be reduced/simplified any further

$$
\begin{aligned}
&(\lambda x.\lambda y.y\ x)(5+4)(\lambda x.\ x + 1) \\
&\quad = (\lambda x.\lambda y.y\ x)\ 9\ (\lambda x.\ x + 1) \\
&\quad = (\lambda y.y\ 9)\ (\lambda x.\ x + 1) \\
&\quad = (\lambda x.\ x + 1)\ 9 \\
&\quad = 9 + 1 \\
&\quad = 10
\end{aligned}
$$

# Call-by-name

- Apply the function as soon as possible
  - Reduce the leftmost-outermost redex, but not inside abstractions

$$
\begin{aligned}
&(\lambda x.\lambda y.y\ x)(5+4)(\lambda x.\ x + 1) \\
&\quad = (\lambda y.y\ (5+4))(\lambda x.\ x + 1) \\
&\quad = (\lambda x.\ x + 1)\ (5+4) \\
&\quad = (5+4) + 1 \\
&\quad = 9 + 1 \\
&\quad = 10
\end{aligned}
$$

# Call-by-value vs. Call-by-name

- When answers are the same, the order of evaluation may be different

$$
\begin{aligned}
&(\lambda x.x\ 6)((\lambda y.y\ y)(\lambda x.x)) \\
&\quad = (\lambda x.x\ 6)((\lambda x.x)\ (\lambda x.x)) \\
&\quad = (\lambda x.x\ 6)(\lambda x.x) \\
&\quad = (\lambda x.x)\ 6 \\
&\quad = 6
\end{aligned}
$$

$$
\begin{aligned}
&(\lambda x.x\ 6)((\lambda y.y\ y)(\lambda x.x)) \\
&\quad = ((\lambda y.y\ y)(\lambda x.x))\ 6 \\
&\quad = ((\lambda x.x)(\lambda x.x))\ 6 \\
&\quad = (\lambda x.x)\ 6 \\
&\quad = 6
\end{aligned}
$$

# η-equivalence

- Two functions are the same iff they give the same result for all arguments
- $\lambda x.Mx = M$, where x is not free in M

# Polymorphic Functions

- $\lambda x.x$
- Identity function
  - $(\lambda x.x)\ M = M$ for any lambda expression M

- Functions that allow arguments of many types are known as polymorphic functions
  - $\lambda x.x$ acts as an identity function on the set of integers, on a set of functions of some type, or on any other kind of object

# Functions of Several Arguments

- Functions with two arguments x and y may be represented by
  - $\lambda x.(\lambda y.M)$

- $f(g, x) = g(x)$

- $f_{curry} = \lambda g.(\lambda x.\ g\ x)$
  - Invented by Schönfinkel, but named after Haskell Curry

# Numbers

- Numbers are defined as functions
  - The number of times a function parameter is applied
  - $0 = \lambda s.\lambda z.z$
  - $1 = \lambda s.\lambda z.sz$
  - $2 = \lambda s.\lambda z.s(sz)$
  - $3 = \lambda s.\lambda z.s(s(sz))$
  - …

# Successor Function

- Successor function
  - $S = \lambda w.\lambda y.\lambda x.y(wyx)$

$$
\begin{aligned}
S(0) &= (\lambda w.\lambda y.\lambda x.y(wyx))0 \\
&= (\lambda w.\lambda y.\lambda x.y(wyx))(\lambda s.\lambda z.z) \\
&= \lambda y.\lambda x.y((\lambda s.\lambda z.z)yx) \\
&= \lambda y.\lambda x.y((\lambda z.z)x) \\
&= \lambda y.\lambda x.yx \\
&= \lambda s.\lambda z.sz
\end{aligned}
$$

# Addition

- M + N = M S N
  - Apply the successor function M times to N
- 2 + 3

$$
\begin{aligned}
2S3 &= (\lambda s.\lambda z.s(sz))(\lambda w.\lambda y.\lambda x.y(wyx))3 \\
&= (\lambda z.(\lambda w.\lambda y.\lambda x.y(wyx))((\lambda w.\lambda y.\lambda x.y(wyx))\ z)3 \\
&= (\lambda w.\lambda y.\lambda x.y(wyx))((\lambda w.\lambda y.\lambda x.y(wyx))\ 3) \\
&= S(S3) \\
&= S4 \\
&= 5
\end{aligned}
$$

# Multiplication

- $M = \lambda xyz.x(yz)$

$$
\begin{aligned}
M22 &= (\lambda xyz.x(yz))\ 2\ 2 \\
&= \lambda yz.2(yz)\ 2 \\
&= \lambda z.2(2\ z) \\
&= \lambda z.2((\lambda v.\lambda w.v(vw))\ z) \\
&= \lambda z.2(\lambda w.z(zw)) \\
&= \lambda z.(\lambda v.\lambda w.v(vw))N \\
&\qquad (\text{let } N = (\lambda w.z(zw))) \\
&= \lambda z.(\lambda w.N(Nw)) \\
&= \lambda z.(\lambda w.N((\lambda w.z(zw))\ w)) \\
&= \lambda z.(\lambda w.N(z(zw))) \\
&= \lambda z.(\lambda w.(\lambda w.z(zw))(z(zw))) \\
&= \lambda z.(\lambda w.(\lambda x.z(zx))(z(zw))) \\
&= \lambda z.(\lambda w.(z(z(z(zw))))) \\
&= 4
\end{aligned}
$$

# Conditionals and Logical Operations

- $T = \lambda xy.x$
- $F = \lambda xy.y$
  - $Fz = \lambda y.y$ (identity function)
- $\wedge = \lambda xy.xyF$
- $\vee = \lambda xy.xTy$
- $\sim = \lambda x.xFT$

$$
\begin{aligned}
\sim T &= (\lambda x.xFT)T \\
&= TFT \\
&= (\lambda xy.x)FT \\
&= F
\end{aligned}
$$

# A Conditional Test

- A function that is true if a number is 0 and false otherwise
- Z = λx.xF~F λs.λz.z

$$
\begin{aligned}
Z0 &= (\lambda x.xF{\sim}F)0 \\
&= (\lambda s.\lambda z.z)F{\sim}F \\
&= (\lambda z.z){\sim}F \\
&= (\lambda z.z)((\lambda x.xFT))F \\
&= (\lambda x.xFT)F \\
&= FFT \\
&= T
\end{aligned}
$$

$$
\begin{aligned}
Z3 &= (\lambda x.xF{\sim}F)3 \\
&= 3F{\sim}F \\
&= (\lambda s.\lambda z.s(s(s(z))))F{\sim}F \\
&= F(F(F{\sim}))F \\
&= IF \quad (I = \lambda x.x) \\
&= F
\end{aligned}
$$

# Predecessor Function

- A pair (a, b) can be represented using the following function
  - PAIR = $\lambda a.\lambda b.\lambda x.xab$
  - FIRST = $\lambda p.pT$
  - SECOND = $\lambda p.pF$

# Predecessor Function

- The following function generates the pair (n+1, n) from the pair (n, m)
  - $\Phi = \lambda p.\text{PAIR}\ (\text{S}(\text{FIRST P}))\ (\text{FIRST}\ p)$
  - A new pair is formed using pT, then it is incremented in the first position and just copied for the second position

- The predecessor of a number n is obtained by applying n times the function Φ to the pair (0, 0) and then selecting the second member of the new pair
  - $\text{P} = \lambda n.\text{SECOND}(n\ \Phi\ (\text{PAIR}\ 0\ 0))$
  - $\text{P(P1)} = \ \text{P0} = \text{SECOND}(0\ \Phi\ (\text{PAIR}\ 0\ 0)) = 0$

# Equality

- A function that tests if a number x is greater than or equal to a number y
  - $GE = \lambda x.\lambda y.Z(xPy)$
- Equality test
  - If $x \geq y$ and $x \leq y$, then $x = y$
  - $EQ = \lambda x.\lambda y. \wedge (Z(xPy))\ (Z(xPy))$

$$\begin{aligned} GE\ 3\ 4 &= (\lambda x.\lambda y.Z(xPy))\ 3\ 4 \\ &= Z(3P4) \\ &= Z\ 1 \\ &= F \end{aligned}$$

$$\begin{aligned} GE\ 4\ 2 &= (\lambda x.\lambda y.Z(xPy))\ 4\ 2 \\ &= Z(4P2) \\ &= Z\ 0 \\ &= T \end{aligned}$$

$$\begin{aligned} EQ\ 2\ 2 &= (\lambda x.\lambda y. \wedge (Z(xPy))\ (Z(xPy)))\ 2\ 2 \\ &= \wedge\ (Z(2P2))\ (Z(2P2)) \\ &= \wedge\ T\ T \\ &= T \end{aligned}$$

# Recursion

- A fixed point of a function G is a value f such that f = G(f)

- A function that calls a function y and then regenerates itself
  - Fixed-point operator
  - $Y = \lambda y.(\lambda x.y(xx))(\lambda x.y(xx))$
  - Yf is a fixed point of f

$$
\begin{aligned}
Yf &= (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))\ f \\
&= (\lambda x.f(xx))(\lambda x.f(xx)) \\
&= f((\lambda x.f(xx))(\lambda x.f(xx))) \\
&= f(Yf)
\end{aligned}
$$

# Recursive Summation

- A function that adds up the first n natural numbers
  - f(0) = 0
  - f(n) = n + f(n-1)
  - If n is 0 the result is 0
  - Otherwise, the successor function is applied n times to the recursive call of the function applied to the predecessor of n

- $f = \lambda r.\lambda n.\ Z\ n\ 0\ (n\ S\ (r(Pn)))$

# Recursive Summation

$$
\begin{aligned}
Y\,f\,3 &= (\lambda y.(\lambda x.y(xx))(\lambda x.y(xx)))\ f\ 3 \\
&= f\ (Yf)\ 3 \\
&= (\lambda r.\lambda n.\ Z\ n\ 0\ (n\ S\ (r(Pn))))\ (Yf)\ 3 \\
&= (\lambda n.\ Z\ n\ 0\ (n\ S\ ((Yf)(Pn))))\ \ 3 \\
&= Z\ 3\ 0\ (3\ S\ ((Yf)(P\ 3))) \\
&= F\ 0\ (3\ S\ ((Yf)(P\ 3))) \\
&= 3\ S\ ((Yf)(P\ 3)) \\
&= 3\ S\ ((Yf)\ 2) \\
&= 3\ S\ (2\ S\ ((Yf)\ 1)) \\
&= 3\ S\ (2\ S\ (1\ S\ ((Yf)\ 0))) \\
&= 3\ S\ (2\ S\ (1\ S\ (Z\ 0\ 0\ (0\ S\ ((Yf)(P\ 0)))))) \\
&= 3\ S\ (2\ S\ (1\ S\ 0)) \\
&= 6
\end{aligned}
$$