G22.3110 Programming Languages (Honors)
Lecture 11

The Lambda Calculus

11.1. The Syntax of the Lambda Calculus

- exp ::=   *constant*   Built-in constant
  - | *var*   Variable name
  - | *exp exp*   Application
  - | λ *var*. *exp*   Lambda Abstraction

11.2. Terminology: bound and free occurences of variables

- x is <u>free</u> in x, but not in any other variable or constant.

- x is <u>free</u> in (E F) iff x is free in E or x is free in F

- x is <u>free</u> in (λy. E) iff x and y are different variables and x is free in E

- x is <u>bound</u> in (E F) iff x is bound in E or x is bound in F

- x is <u>bound</u> in (λy. E) iff   x and y are the same variable and x occurs free in E,
     <u>or</u> x occurs bound in E.

Intuitively, an occurrence of a variable in an expression is free if it refers to a variable introduced (by a λ-abstraction) outside of the expression.

11.3. Substitution

The definition of subsitution in the lambda calculus

- x [M/x] = M

- c [M/x]   where c is any variable or constant other than x
       = c

- (E F) [M/x] = (E [M/x]) (F [M/x])

- (λx. E) [M/x] = λx. E

- (λy. E) [M/x]   where y is any variable other than x
           = λy. (E [M/x])   if x does not occur free in E or y does not occur free in M

           = λz. (E[z/y]) [M/x]   otherwise, where z is a new variable name which does not occur free in E or M.

11.4. Conversion Rules

11.4.1 α-conversion

- If y is not free in E then

    (λx. E) ⇔_α *(λy. E[y/x])*

11.4.2 β-conversion

$(\lambda x.\ E)\ M \underset{\beta}{\Leftrightarrow}\ E[M/x]$

11.4.3 η-conversion

- If x is not free in E and E denotes a function

$(\lambda x.\ E\ x) \underset{\eta}{\Leftrightarrow}\ E$

11.4.4 β- and η-Reduction

When used left to right, the β-conversion and η-conversion rules are called β-reduction and η-reduction, respectively, and the arrow is written as $\Rightarrow$.

An expression of the form $(\lambda x.\ E)\ M$ is a candidate for β-reduction and is called a "redex" (reducible-expression).

An expression in which there are no redex's is said to be in "normal form". Notice that not all expressions can be reduced to a normal form, for example:

$(\lambda x.\ x\ x)\ (\lambda x.\ x\ x)$

11.4.5 δ-reduction

As a convenience for providing built-in constant functions, such as +, -, if, etc., an additional reduction rule, called δ-reduction, is defined for those constant functions.

11.5. Recursion: The Y combinator

Recursion is provided through the fixpoint combinator Y, which is defined as

$Y f = f\ (Y f)$

In other words, given a function f, Y returns the fixpoint (also called the "fixed point") of f. Y can be expressed in the lambda calculus as

$Y = (\lambda h.\ (\lambda x.\ h\ (x\ x))\ (\lambda x.\ h\ (x\ x)))$

The factorial function would thus be written as

$FAC = Y\ (\lambda fac.\ \lambda n.\ if\ (= n\ 0)\ 1\ (*\ n\ (fac\ (-\ n\ 1))))$

11.6. Reduction Order

There may be several redex's in an expression and the choice of which redex to reduce next is determined by the chose reduction order. Some common reduction orders are:

- Applicative Order Reduction: Leftmost, innermost redex first

- Normal Order Evaluation: Leftmost, outermost redex first.

Other possibilities are parallel innermost, rightmost innermost, etc.

11.7. The Church-Rosser Theorems

How will the reduction order affect the normal form reached, if any?

11.7.1 Church-Rosser Theorem I

Theorem If $E_1 \Leftrightarrow E_2$, then there exists an expression E such that $E_1 \Rightarrow E$ and $E_2 \Rightarrow E$.

Corollary No expression can be converted to two distinct normal forms (i.e. normal forms that are not

α-interconvertible).

- Does this mean all reduction orders are equivalent?

11.7.2 Church-Rosser Theorem II

<u>Theorem</u> If $E_1 \Rightarrow E_2$ and E_2 is in normal form, then there exists a <u>normal order</u> reduction sequence from E_1 to E_2.