Notes on the Lambda Calculus

(Abstracted from Stoy "Denotational Semantics", chapter 5.)

Functional notation: f x instead of f(x)
f x is the application of function f to argument x

Functions are written as *abstractions* with λ thus: λ x.M where x is the function's variable (parameter in programming languages) and M is its body.

Thus (λ x.M)a applies the function to its argument a. The λ-calculus gives this notation a formal syntax and transformation or conversion rules.

Syntax:

```
<expression> ::= <variable>
               | <expression> <expression>
               | λ <variable> . <expression>
               | ( <expression> )
```

Application associates to the left e.g. the expression `abc` means `(ab)c`, not `a(bc)`

The body of an abstraction extends to the first unmatched right parenthesis or the end of the expression. e.g. `(λ x.λ y.N)ab` means `(λ x.(λ y.N))ab` which `means ((λ x.( λ y.N))a)b`

Free and Bound

Considering the four cases in the syntax definition, *free* means:

1. x is free in x
2. x is free in XY if x is free in X or Y
3. x is free in λ y.X if x is not equal to y and x is free in X
4. x is free in (X) if x is free in X

For the same four cases, *bound* means:

1. no variable is bound in x for any x
2. x is bound in XY if x is bound in X or Y
3. x is bound in λ y.X if x is equal to y or if x is bound in X
4. x is bound in (X) if x is bound in X

e.g. consider (λ x.ax)x
x is free in ax
x is bound in (λ x.ax)
the third x is free in (λ x.ax)x

NOTE: not free is not the same as bound and not bound is not the same as free
e.g. x is not free in y, but is also not bound
y is not bound in λ x.x, but is also not free

However: free is the same as not bound and bound is the same as not free.

In an abstraction λx.M, x is the *bound* variable and M is the *body*.

Definition of Substitution

Let x be a variable; M, X be expressions.
X' is then the expression [M/x]X defined as follows over the four syntactic cases:

1. X is a variable

   1.1. If X ≡ x, X' ≡ M e.g. [M/x]x ⇒ M

   1.2. If X ¬≡ x, X' ≡ X e.g. [M/x]y ⇒ y

2. X is an application YZ
   X' ≡ ([M/x]Y)([M/x]Z)

3. X is an abstraction λy.Y

   3.1. If y ≡ x, X' ≡ X e.g. [M/x]λx.Y ⇒ λx.Y (can't substitute for a bound variable)

   3.2. If y ¬≡ x,

      3.2.1. If x is not free in Y or y is not free in M,
      X' ≡ λy.[M/x]Y e.g. [y/x]λy.z ⇒ λy.z (x is not free in Y, y is free in M)
      e.g. [z/x]λy.x ⇒ λy.z (x is free in Y, y is not free in M)

      3.2.2. If x is free in Y and y is free in M,
      X' ≡ λz.[M/x]([z/y]Y) where z is a new variable that is not free in M or Y
      e.g. [y/x]λy.x ⇒ λz.[y/x][z/y]x
      ⇒ λz.y

Conversion Rules

α rule: if y is not free in X, $\lambda x.X \Rightarrow^{\alpha} \lambda y.[y/x]X$

e.g. $\lambda x.x \Rightarrow^{\alpha} \lambda y.[y/x]x$

$\Rightarrow \lambda y.y$

but $\lambda x.y$ does not give $\lambda y.[y/x]y$ or $\lambda y.y$

β rule: $(\lambda x.M)N \Rightarrow^{\beta} [N/x]M$

e.g. $(\lambda x.y)y \Rightarrow^{\beta} [y/x]x$

$\Rightarrow y$

η rule: If x is not free in M, $\lambda x.Mx \Rightarrow^{\eta} M$

e.g. $\lambda x.yx \Rightarrow^{\eta} y$

why? because $(\lambda x.yx)N \Rightarrow^{\beta} yN$

but, $\lambda x.xx$ does not give x

because $(\lambda x.xx)M \Rightarrow^{\beta} MM$, not xM

Evaluation Order

1. Normal order - reduce the leftmost expression first and then apply it (like "call by name")
2. Applicative order - reduce two leftmost expressions (function and its argument) and apply one to the other (like "call by value")

Normal order terminates if any order does. Applicative order may not terminate.
e.g. $(\lambda x.\lambda y.y)((\lambda x.xx)(\lambda x.xx))b$
Evaluated by normal order this produces $(\lambda y.y)b$ or b, since x does not appear in the body of the function. Evaluated by applicative order this never terminates, since the argument to $((\lambda x.xx)\ (\lambda x.xx))$ reduces to itself.