COMP 302 Lecture 16 24 October 2016

For closures → syntax and implementation strategy.
Syntax will be built on what we have, almost no changes. Whole idea of returning closures and passing them around in limited way.
Only generate closures at definition time. Will make things not quite as flexible, but eliminates complexity *see 16c302*.
Works for anonymous and named functions.
To recognize this, inspect the name (or the evaluated name) and if it begins with a back quote, we know that we will return it as a closure.
See 16c302-1

At this point, we've made a fairly complete functional language.
We will look at an abstract formalism.
The language we'll look at is actually quite old (1930s): *λ-calculus*. Alonzo Church (1932-36).
It's a model of computation.
Very simple syntax, there isn't much to this language, only a few symbols.

(λx.x)z -> lambda means we create a function. The first x is its argument. The second x is its body. The argument z is its function application. This reduces to z.
Pieces we will need:

- Variables/identifiers (single-letter usually).
- λ
- .
- () -> grouping behaviours. Sometimes drop these due to getting a lot of brackets, when unambiguous.

λ-terms are defined recursively, (capitals will be λ-terms)

1) variables are λ-terms
2) if x is a variable, and M is a λ-term, then I can create a new function (λx.M)
3) if M is a λ-term, and N is another λ-term, I can create another λ-term (M N)

((λx.y)z)
That's all there is to it, syntactically.
Entirely self-contained as a language. We won't add numbers, multiply/divide, etc, we will build everything from scratch.

Main activity (evaluation) is function-application.
(λx.M)N

Some concepts:

- Free vs. bound variables. A free variable is just floating around. A bound variable is one that was declared as an argument to a function.
    - In x, x is free.
    - If x is free in M, then all such occurrences are bound to λx.M.
    - If y ≠ x, and y is free in M as well, then y is still free in λx.M.
    - If x is free in M or N, then x is still free in (M N).
    - i.e. λx: x is bound to the parameter.
    - (λx.x)x -> last x is a different x.

Evaluation will require a process of substitution *see 16c302-2*
Other rules as well for more subtle cases. There is a way of avoiding this capturing. We could replace variable names. Take (λx.x) and (λy.y), they are both identity functions.
Process called α-substitution: replace variables when it doesn't change the meaning.
Main process of application: β-reduction: (λx.y)z -> y.