Brady Dean

10/29/18

- Created by Alonzo Church in the 1930s.

Lambda Calculus is made of expressions that are defined recursively:

$$< expression > := < name > \mid < function > \mid < application >$$
$$< funciton > :=\ \lambda < name >. < expression >$$
$$< application > := < expression > < expression >$$

- The left hand side of an application is the function and the right hand side is the argument.
- This is untyped lambda calclulus.
- Everything is a function, even numbers.
    - Numbers are encoded by the number of times a function is applied to itself.
    - 0 = λs.λz. z the function s is applied to the argument z zero times
    - 1 = λs.λz. s z the function s is applied once
    - 2 = λs.λz. s (s z) the function s is applied twice
- A lambda is also known as an anonymous function.
- The only two keywords are lambda and dot.
- Function application associates to the left.
- Identifiers that do not appear in the head are free variables.
    - $(\lambda x. xy)$ – y is a free variable
    - An identifier is free if it is unbound from an expression.
- All identifiers are local to their function.
    - $(\lambda x. x)(\lambda x. xy)$ – These are two distinct x's.
- If substituting E brings an unbound into a bound expression, the variable is renamed.
- One way of making this distinction properly is to rename bound variables during substitution, making sure to always give them unique names. This is called α-conversion and expressions that only differ in bound variable names are considered α-equivalent or even completely equivalent.
- $(\lambda x. M)N = M[x \coloneqq N]$
- Application associates to the left
- Abstraction associates to the right