Theory of Computation
λ-Calculus

λ-Calculus

- Consists of a set of objects called λ-terms and some rules to manipulate them.
- Designed to capture functional abstraction and functional application, and their interaction
- These principles used in programming languages like LISP, SCHEME and DYLAN

Expressions

- $\lambda x.f(x)$ denotes a function f that on input x computes $f(x)$.
- To apply this function to an input, substitute the input for the variable x in $f(x)$ and evaluate.
- Eg. Successor function: $\lambda x.(x+1)$ is the expression; $(\lambda x.(x+1))7 \rightarrow 7+1=8$.
- Functions can also be taken as inputs: An expression like $\lambda f.\lambda g.\lambda x.f(gx)$ takes input function f and computes the expression on variables g, x, then input function g and the expression on variable x and finally input x and evaluates the expression for the composition function fg.

Example of Functions as inputs

Take f and g to be successor functions:

$(\lambda f.\lambda g.\lambda x.(f(gx)))(\lambda y.(y+1))(\lambda z.(z+1))$

$\rightarrow (\lambda g.\lambda x.((\lambda y.(y+1))(gx)))(\lambda z.(z+1))$ when f is substituted

$\rightarrow \lambda x.((\lambda y.(y+1))((\lambda z.(z+1))x))$ when g is substituted

$\rightarrow \lambda x.((\lambda y.(y+1))(x+1))$ when x substitutes z

$\rightarrow \lambda x.((x+1)+1)$ when $x+1$ substitutes y

Example of Functions as inputs contd.

Take f and g to be successor functions:
Try the same by substituting y by gx in the second step
Same answer should come - the order of substitutions does not matter.

Currying

- Functions take one variable at a time.
- For two variable functions $f(x, y)$, we assume that first the variable x is being substituted to give a function on y and then the variable y is being substituted to evaluate $f(x, y)$.
- $\lambda x.\lambda y.f(x, y)$
- This is called Currying after the Haskell B. Curry.

Pure λ-calculus

- There are only variables $\{f, g, h, x, y, \ldots\}$ and operators for λ-abstraction and application.
- λ-terms (inductively defined): Any variable x is such a term;
  If M and N are λ-terms then so is MN - as if M is a function that is to be applied on input N,
  If M is a λ-term and x is a variable, then $\lambda x.M$ is also a λ-term - as if on input x the term M will be computed.

Pure λ-calculus contd.

- The operation of application is not associative: $(MN)P$ and $M(NP)$ could lead to very different answers.
- By convention, in unparenthesized expressions the parenthesis is given from the left. Eg. MNP should be $(MN)P$.
- λ-terms can be thought of as both functions as well as input data - shown in substitution rule in next slide.

Pure λ-calculus: Substitution rules

- Given λ-terms $\lambda x.M$ and N, $s_N^x(M)$ is the term where,
  (i) bound variables y of M are renamed so that they are not common with x or variables of N, and
  (ii) all occurrences of x are substituted by N.
- β-reduction: Suppose a λ-term has a subterm of the form $(\lambda x.M)N$, we can replace the subterm with $s_N^x(M)$.

Pure λ-calculus: Substitution rules

- The order of reductions does not matter - by Church-Rosser property.
- Normal form: if a term cannot be converted by any more β-reductions.
- not all terms have normal forms: $(\lambda x.xx)(\lambda x.xx)$.

Church Numerals and Equivalence with TMs

- Method to encode natural numbers as λ-terms.
- This method will help us show that all μ-recursive functions can be simulated in λ-calculus, and therefore all Turing computable functions.
- On the other hand, description of exhaustive application of substitution rules can be implemented in Turing machines.
- So λ-calculus and Turing Machines are equivalent.

Church Numerals contd.

- $\overline{0} := \lambda f.\lambda x.x$
- $\overline{1} := \lambda f.\lambda x.fx$
- $\overline{2} := \lambda f.\lambda x.f(fx)$
- $\overline{n} := \lambda f.\lambda x.f^n x$

Successor function by Church numerals

- $s := \lambda m.\lambda f.\lambda x.f(mfx)$.
- $s\overline{n} = (\lambda m.\lambda f.\lambda x.f(mfx))(\lambda f.\lambda x.f^n x)$
  $\rightarrow (\lambda m.\lambda g.\lambda y.g(mgy))(\lambda f.\lambda x.f^n x)$ as f, x needed renaming
  $\rightarrow \overline{n+1}$ by a series of β-reductions (please work it out).

Other μ-recursive functions

Similarly, all the μ-recursive functions can be defined in terms of λ-calculus and Church numerals.