Lambda calculus

1) What is a function in Lambda calculus?
A function in the lambda calculus has the formal notation λx.M which is also called Lambda abstraction.

2) Present the Lambda notation.
Lambda expression:
- variables: x, y, z, ...
- lambda abstraction: λx.M
- application: M N

Lambda abstraction:
- x is function argument
- M is function expression ->Receipt that specifies how function is »computed«

Application M N:
- If M = λx.M' then all occurences of x in M' are replaced with N
- Mechanical definition of parameter passing(?)

3) Present the syntax of Lambda calculus.
Definition: The set of λ-expressions Λ is constructed from infinite set of variables {v,v',v'',v''', ...} by using application and λ-abstraction:
- $x \in V \Rightarrow x \in \Lambda$
- $M,N \in \Lambda \Rightarrow (M\ N) \in \Lambda$
- $X \in V,\ M \in \Lambda \Rightarrow \lambda x.M \in \Lambda$

Syntax rules:
• Application is left-associative: M N L ≡ (M N) L
• Λ-abstraction is right-associative: λx.λy.λz.M N L ≡ λx.(λy.(λz.((M N) L)))
• We often use the following abbreviation: λxyz.M ≡ λx.λy.λz.M

4) Describe the concepts of the operation substitution, the Alpha conversion, and the Beta reduction.

1. Substitution

Substitute all instances of a variable x in λexpression M with N:

$$[N/x]M$$

Definition:
Let $M,N \in \Lambda$ and $x,z \in V$. Substitution rules:
- $[N/x]x = N$
- $[N/x]z = z$, if $z \neq x$
- $[N/x](L\ M) = ([N/x]L)([N/x]M)$
- $[N/x](\lambda z.M) = \lambda z.([N/x]M)$, if $z \neq x\ z \wedge \notin FV(N)$

Example:
- [y(λv.v)/x]λz.(λu.u) z x ≡ λz.(λu.u) z (y (λv.v))
- Check evaluation of substitution rules !

2. Alpha conversion

- Renaming bound variables in λ-expression yields equivalent λ-expression
- Example: λx.x ≡ λy.y
- Alpha conversion rule: $\lambda x.M \equiv \lambda y.([y/x]M)$, if $y \notin FV(M)$.

  Example:
- Λ-expression: (λf.λx.f (f x)) (λy.y + x)
- Blind substitution gives: λx.((λy.y + x) ((λy.y + x)x)) = λx.x + x + x
- Correct substitution: λz.((λy.y + x) ((λy.y + x) z)) = λz.z + x + x

3. Beta reduction

- β-reduction is the only rule used for evaluation of pure λ-calculus (aside from renaming)
- Expression (λx.M) N stands for operator (λx.M) applied to parameter N
- Intuitive interpretation of (λx.M) N is substitution of x in M for N

Definition: Let λx.M be λ-expression. Application of (λx.M) on parameter N is implemented with βreduction: $(\lambda x.M)\ N \rightarrow [N/x]M$

- Expression (λx.M) N is called redex (reducable expression)
- Expression [N/x]M is called contractum
- P includes redex (λx.M) N that is substituted with [N/x]M and we obtain P'
- We say that P β-reduces to P': $P \rightarrow_\beta P'$

Definition: β-derivation is composed of one or more β-reductions. β-derivation from M to N: $M \twoheadrightarrow_\beta N$

5) How to evaluate a Lambda expression?

- Λ-calculus is very expressive language equivalent to Turing machine
- Evaluation of λ-expressions is based on:
  1. α-coversion and
  2. substitution
- Evaluation is often called reduction
- Λ-expressions are reduced to value
  - Values are normal forms of λ-expressions i.e. λexpressions that can not be further reduced

6) How can we represent and compute with Boolean values in Lambda calculus?

- true ≡ λt.λf.t | function returning first argument of two
- false ≡ λt.λf.f | function returning second argument of two

6) How can we represent and compute with integer numbers in Lambda calculus?

Church numbers:

- Number n is represented with C n
  - n = 0+1+...+1 | n times successor of 0
  - z stands for zero and s represents successor function
- Arithmetic operations
  - Plus = λm.λn.λz.λs.m (n z s) s
  - Times = λm.λn.m C 0 (Plus n)

7) What are combinators? Present the essential combinators.

- Combinators are primitive functions
  - Expressing basic operations of computation
  - Functions: identity, composition, choice, etc.
  - Higher-order functions: apply, map, fold, filter, etc
- Identity function: I = λx.x
- Choosing one argument of two: K = λx.(λy.x)
- Passing argument to two functions: S = λx.λy.λz.(x z)(y z)
- Function that repeats itself: Ω = (λx.x x)(λx.x x)
- Function composition: B = λf.λg.λx.f(g x)
- Inverse function composition: B' = λf.λg.λx.g(f x)
- Duplication of function argument: W = λf.λx.f x x
- Recursive function: Y = λf.(λx.f (x x))(λx.f (x x))

8) How can we model the recursion in Lambda calculus?

- Recursion can be expressed using combinator Y
  - – $Y = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))$
- Important property of Y
  - – $Y\ F =_\beta F\ (Y\ F\ )$
  - – Proof:
    - $Y\ F = \lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x))\ F \rightarrow (\lambda x.F\ (x\ x))(\lambda x.F\ (x\ x)) \rightarrow F\ ((\lambda x.F\ (x\ x))(\lambda x.F\ (x\ x))) \leftarrow F\ ((\lambda f.(\lambda x.f\ (x\ x))(\lambda x.f\ (x\ x)))\ F\ ) = F\ (Y\ F\ )$

9) Present some important properties of Lambda calculus.

- LC is consistent
- LC is equivalent to TM (Turing machine)
  - LC is r.e.
  - LC is partially computable (not total !)
- LC with types is total function
  - Very limited class of languages
- The characterisation of total TM is not known