Tutorials 02:
Lambda Calculus

Exercise 01

Define the set of free and bound variables for each of the following expressions:

- `λx.λy.(λx.y λy.x)`
- `λx.x (λy.(λx.x y) x)`
- `λa.(λb.a λb.(λa.a b))`
- `λp.λq.(λr.(p (λq.(λp.(r q)))) (q p))`

Exercise 02

Evaluate the following expressions

- `((λx.λy.(y x) λp.λq.p) λi.i)`
- `(((λx.λy.λz.((x y) z) λf.λa.(f a)) λi.i) λj.j)`
- `(λh.((λa.λf.(f a) h) h) λf.(f f))`
- `((λp.λq.(p q) (λx.x λa.λb.a)) λk.k)`
- `(((λf.λg.λx.(f (g x)) λs.(s s)) λa.λb.b) λx.λy.x)`

Exercise 03

We remind that:

- `add = λmnfx.mf(nfx)`
- `mult = λmnf.m(nf)`

Perform the following calculations:

- 0 + 2
- 2 + 3
- 2 * 2
- 2 * 0

Exercise 04

Evaluate the following expressions on numbers (Church Numerals) and try to identify the implemented function:

1. `λxyz.(y(xyz))` (a single parameter of type integer)
2. `λxyz.xyz` (three parameters: a boolean and two integers)
3. `λfsb.(bfs)` (two integers)
4. `λp.p True` (p is obtained from expression 3 above)
5. `λp.p False` (p is obtained from expression 3 above)

Combinator

A combinator is a closed lambda expression (with no free variables). Combinators are used to evacuate the notion of variable and simplify the reduction of λ-expressions. They were proposed by Haskell Curry in an attempt to define a reduced set of combinators that allows computable functions to be defined. In the following, we assume the following list of combinators: True, False, not, ifzero, succ, pred, add, diff, mult, if, less, eq, the fix point (fix).

Exercise 05

Write the following programs in λ-expression form:

- The program that calculates the double of a number
- The program that takes two numbers and returns the larger one,
- The program that calculates the power (x to the power of n)