Czech Technical University
Faculty of Informatics
Department of Theoretical Informatics

BI-PPA 2017 | **Cvi Introduction to the λ-calculus** | Date: 6. 10. 2017

We are going to cover Functions in λ-calculus, α-, β- a η-reductions, order of evaluation (normal vs. aplicative), the normal form, arithmetics in λ-calculus

Rules of the λ-calculus (from last exercise):

1. Variable is a valid expression in λ-calculus (any lowercase letter from english alphabet).
2. If M and N are valid λ-calculus expressions, then the following are also valid expressions:
   - (M) . . . enclosing an expression in parentheses,
   - $\lambda id.\ M$ . . . so called **abstraction**, where id is any variable,
   - MN . . . so called **application**, where N is applied to M.

$\{(,), ., \lambda, a, \ldots, z\}$, starting nonterminal is `<exp>` and the rules are

Simplifying the notation (from last exercise:

- Expressions in the form of (((((AB)C)D)E)F),
- (λx. (λy. (λz. ((x y) z)))) can be written as (λx.λy.λz. ((x y) z)) and then as (λxyz. (x y z)).
- Analogically we will use (λxyz. (x y z)) 1 2 3 in the meaning of (λx. (λy. (λz. ((x y) z)) 3) 2) 1.
- Discarding the inner parentheses, i.e. instead of (λxyz. (x y z)) we can use (λxyz. x y z) and instead of (λxyz. (+ x (− yz))) we will write (λxyz. + x (− y z)).

Example: (λx. (λy. (+ x y)) 4) 3 in simplified form (λxy. + x y) 3 4 will be transformed in the following way:
(λxy. + x y) 3 4 $\to$ (λy. + 3 y) 4 $\to$ (+ 3 4) $\to$ 7. **Ex. 1**. Transform the following.

a) (λx. + x 1)3

b) (λxy. − x y)3 5

c) (λx. (λy. − x y)3) 5, ! diffenent than the example above.

d) (λx. + x 1)((λy. + y 2)3)

e) (λfx.f x)(λy. + y 1)

f) (λfx.f x)(λy. + y 1) 5

g) (λf x. fx)(λy.y)

h) (λx.(+ x ((λx. + x 1)3)))2

i) ((λx.λy.x)y)z, be careful about free and bound variables.

j) (λs. λq. s q q) (λq.q) q 5

Reductions, normal form:

- **β-reduction** (Appplication): (λx. E) A, all bound uses of x in the expression E will be replaced with A. AS LONG AS A does not have other free variables that might collide.
  Example: (λxy. (x y)) (ay) CANNOT transform to (λy. ((ay) y)), because y would become bound variable in (ay) (the solution is to use α-reduction). But (λxy. (x y)) (az) is fine and will look as (λy. ((az) y)) after the application.

- **α-reduction** (Renaming): Based on the principle that (λx.x) and (λy.y) are identical functions, because variable names alone do not matter. It is therefore only renaming of all bound uses of x to y, but ONLY IF y was not free in E, in which case we must select different letter. **Example:** (λxy. (x y)) y IS NOT (λy. (y y)), because y would be bound. We'll rename first using (α-reduction) y to t: (λxt. (x t))y, and then can apply y to x with result: (λt. (y t)). This reduction is used before β-reduction in which free variables might be incorrectly bound.

- **η-reduction** (Optimalization): Special case of the β-reduction, where instead of application, we just delete the lambda. It is only useful for expression in the form of (λx.A x), where bound x is the rightmost element in the function definition and there are no uses of x in A. For example (λx.(A x))B replaces lambda with B and the result is AB. Using η-reduction x and λ will be deleted and connected to the rest of the form after the parentheses, i.e. we will remove x and λ from the function (λx.A x)B and end up with having AB as well.

Functions in **normal form** are those functions for which there is no further reduction possible using either β-rductions or η-reductions, only renamings (α-reductions).p. **Ex. 2**. Further examples: What is the result of the following expressions?

a) (λy. + 8 y)((λx. + x 1) 3)

b) (λx. (λx. (λy. * x y)3) ((λz.+ x z)2))1

c) (λx. x i)((λz. (λq. q) z) h)

d) (λx. x i)((λz. (λq. q z)) h)

e) (λx. x o j)((λy. (λz. z h)y)a)

f) (λx. (λx. (λx. xxx)(bx)x)(ax))c

g) try writing lambda function that "prints" expression.

h) (λw.(λx.(λy.w y a) (u w)) b) y

i) (λp. (λq. (λp. p (p q))(λr. + p r))(+ p 4))2 . . . normal vs applicative evaluation

There are many ways to simplify this example:
Normal and applicative evaluation are two differet ways how to tranform λexpressions. **Normal** evaluation proceeds always from the left (i.e. what we did before). It has only one rule, which is

to **strictly apply from the left** (i.e. substitute what is after the parentheses with λ) without any modifications. This means that the resulting expression might be more complicated than necessary. On the other hand, the **applicative** evaluation attempts to simplify an expression **before** ts application (substitution) on λ.

- An example of normal evaluation:
  (λx. + xx) ((λp. + p 4)3) leads to
  $\rightarrow$ + ((λp. + p 4)3)((λp. + p 4)3) …**only substitute**, no other updates for the expression ((λp. +p 4)3) are made
  $\rightarrow$ + (+ 3 4) ((λp. + p 4)3)… and again evaluate from left
  $\rightarrow$ + (+ 3 4) (+ 3 4) $\rightarrow$ + 7 7 $\rightarrow$ 14

- An example of applicative evaluation:
  (λx. + xx) ((λp. + p 4)3) leads to
  $\rightarrow$ (λx. + xx) (+ 3 4) …**ssimplify** second parenthesis before its application (because it is possible), and apply the result of the simplification
  $\rightarrow$ (λx. + xx) 7 … update more and the subsstitute x for 7
  $\rightarrow$ + 7 7 $\rightarrow$ 14

While the second example looks much more elegant, the disadvantage of applicative evaluation is that there are cases in which applicative transformations may not lead to a normal form (but if it does, it gives the same result as normal evaluation - proved in 1936 in the Church-Rosser theorem). This is shown in the next example: dávat stejný výsledek.

a) (λx. (λw.(λy. wyw)b))((λx. xxx)(λx. xxx)) ((λz.z)a) … transform using both normal and applicative evaluation order

b) (λxy. (+ x y))((λx. ((λx. λy. (+ y((λz. (* x z))3)))7 5)) x) 4
   $\Rightarrow$ (+ (+ 5 (* 7 3)) 4)

c) ((λx. xx)(λx. xx))

d) ((λx. R(xx))(λx. R(xx)))

Ex. 3. Transform the following expressions into normal forms using normal and then applicative order of evaluation.

a) (λs. (λq. s q q) (λq.q)) q

b) (λx. ((λx y. (* x y)) 2 (+ x y))) y

c) (λx. λy. (x y)) (λz. (z y))

d) (λx. y (λt. t t) (λx. x x))

e) (λx. y) ((λt. t t) (λx. x x))