Lambda calculus 2

Štruktúra prednášok:

- úvod do syntaxe (gramatika + konvencie)
- sémantika (redukčné pravidlá)
- programovací jazyk nad λ-kalkulom

domáca úloha: interpreter λ-kalkulu, ...

- opakované aplikovanie β-redukcie

Dnes: (alternatíva: https://en.wikipedia.org/wiki/Lambda_calculus)

- vlastnosti β-redukcie
- rekurzia a pevný bod
- de Bruijn indexy (miesto mien premenných)

Rekapitulácia

β-redukcia – substitúcia

- $(\lambda\ x.B)\ E \rightarrow_\beta B[x:E]$

λx.λy.(x y)

$(\lambda\ x\ (\lambda\ y\ (x\ y)\ ))\ \ (x\ y) \rightarrow_\beta$

$(\lambda\ y\ (x\ y)\ ))[x:(x\ y)] \rightarrow_\beta$ ZLE $(\lambda\ y\ ((x\ y)\ y)))$

(λy.B)[x:M] = λ z.(B[y:z][x:M])
ak **x∈Free(B) && y∈Free(M)**

$(\lambda\ y\ (x\ y)\ ))[x:(x\ y)] \rightarrow_\beta$ DOBRE

$(\lambda\ w\ (x\ w)\ ))[x:(x\ y)] \rightarrow_\beta$

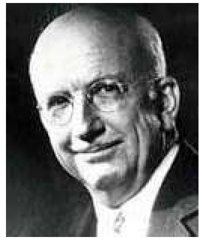
$(\lambda\ w\ ((x\ y)\ w)\ ))$

Terminácia – Noetherovská

Emmy Noether

- výpočet = opakované aplikácie β-redukcie nemusia skončiť
- jediná šanca, aby výpočtový model λ-kalkul mohol byť ekvivalentný Turingovmu stroju, a nie konečnému automatu
- nekonečná sekvencia
  - $\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \ldots$
- neobmedzene puchnúca sekvencia
  - $\omega_3\ \omega_3 \to_\beta \omega_3\ \omega_3\ \omega_3 \to_\beta \omega_3\ \omega_3\ \omega_3\ \omega_3$

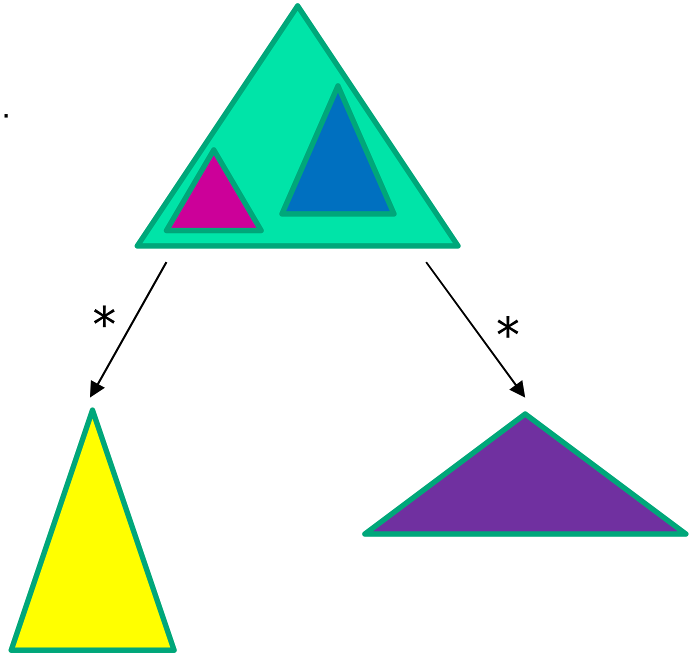
$\omega = \lambda x.xx = \lambda x.(x\ x)$
$\Omega = \omega\omega$

Konfluencia-Church-Roser

jednoznačnosť -problém Janka a Marienky

- normálna forma (vzhľadom na β-redukciu) je, keď sa už nedá použiť
- či existujú dve rôzne normálne formy (vzhľadom na β-redukciu) ?
- nejednoznačný "výsledok" pre dva rôzne výpočty ($K=\lambda xy.x$, $I=\lambda x.x$, $\Omega=\omega\omega$)
  - $K I \Omega \rightarrow_\beta I$ ale aj
  - $K I \Omega \rightarrow_\beta K I \Omega \rightarrow_\beta K I \Omega \rightarrow_\beta \ldots$



Syntaktická analýza λ-termu
(bolo na cvičení)

Pri práci s vašim interpretrom vám (ne)bude chýbať:

- **vstup λ termu** – funkcia fromString :: String -> LExp, ktorá vám vytvorí vnútornú reprezentáciu z textového reťazca, príklad:

  parser "\x.xx" = (LAMBDA "x" (LExp (ID "x"), (ID "x")))

  takejto funkcii sa hovorí syntaktický analyzátor a musíte sa vysporiadať s problémom, keď je vstupný reťazec nekorektný
- použili sme "techniku" rekurzívneho zostupu (Recursive descent), ale o analyzátoroch-parseroch v tejto prednáške bude reč neskôr
- **výstup λ termu** – funkcia show :: LExp -> String, ktorá vám vytvorí textovú (čitateľnú) reprezentáciu pre λ term.

čo by Vás mohlo inšpirovať

Fold na termoch
(bolo na cvičení)

```
foldLambda lambda var apl con cn lterm
    | lterm == (LAMBDA str exp) =
                lambda str (foldLambda lambda var apl con cn exp)
    | lterm == (VAR str) = var str
    | lterm == (APP exp1 exp2) =
                apl      (foldLambda lambda var apl con cn exp1)
                         (foldLambda lambda var apl con cn exp2)
    | lterm == (CON str) = con str
    | lterm == (CN int) = cn int

vars = foldLambda (\x y->y) (\x->[x]) (++) (\_->[]) (\_->[])

show :: LExp -> String
show = foldLambda (\x y->"(\\"++x++"->"++y++")")
          (\x->x) (\x->\y-> "("++x++" "++y++")" ) (\x->x) (\x->x)
```

čo by Vás mohlo inšpirovať

Fold na termoch

(bolo na cvičení)

```
foldLambda fcie@(lambda,var,apl,con,cn) lterm
    | lterm == (LAMBDA str exp) = lambda str (foldLambda fcie exp)
    | lterm == (VAR str)        = var str
    | lterm == (APP exp1 exp2)  = apl   (foldLambda fcie exp1)
                                        (foldLambda fcie exp2)
    | lterm == (CON str)        = con str
    | lterm == (CN int)         = cn int

vars = foldLambda ((\x y->y),(\x->[x]),(++),(\_->[]),(\_->[]))

show :: LExp -> String
show = foldLambda ((\x y->"(\\"++x++"->"++y++")"),
          (\x->x),(\x->\y-> "("++x++" "++y++")"),(\x->x),(\x->x))
```

η-redukcia

- β-redukcia: $(\lambda\ x.B)\ E \rightarrow_\beta B[x:E]$
- η-redukcia: $\lambda\ x\ (B\ x) \rightarrow_\eta B$ ak $x \notin Free(B)$

Príklad: $\lambda x\ (f\ x) \rightarrow_\eta x$, resp. v úvode minulej prednášky bolo

let length = λys.(if (null xs) 0 ((+) 1 (length (tail ys)))) in (length …

$anti_\eta$ $\quad {}_\eta$<-

let length = (λf.λys.(if (null xs) 0 ((+) 1 (f (tail ys))))) length in (length …

podmienka je podstatná, lebo ak napr. B=x, teda $x \in Free(B)$, $\lambda x.(x\ x) \rightarrow_\eta x$

β a η-redukcia

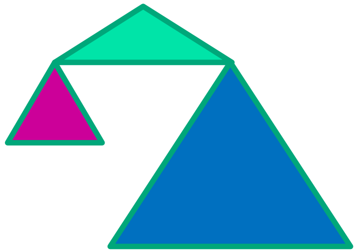
- $\to_{\beta\eta}$ je **uzáver** $\to_\beta \cup \to_\eta$ vzhľadom na podtermy, čo znamená:

matematická definícia:

- ak $M \to_\beta N$ alebo $M \to_\eta N$, potom $M \to_{\beta\eta} N$,
- ak $M \to_{\beta\eta} N$, potom $(P\ M) \to_{\beta\eta} (P\ N)$ aj $(M\ Q) \to_{\beta\eta} (N\ Q)$,
- ak $M \to_{\beta\eta} N$, potom $\lambda x.M \to_{\beta\eta} \lambda x.N$.

Ako to chápe informatik:

- rekurzívne prejdi celý λ-term
- aplikuj $\to_\beta$ resp. $\to_\eta$ kde sa len dá



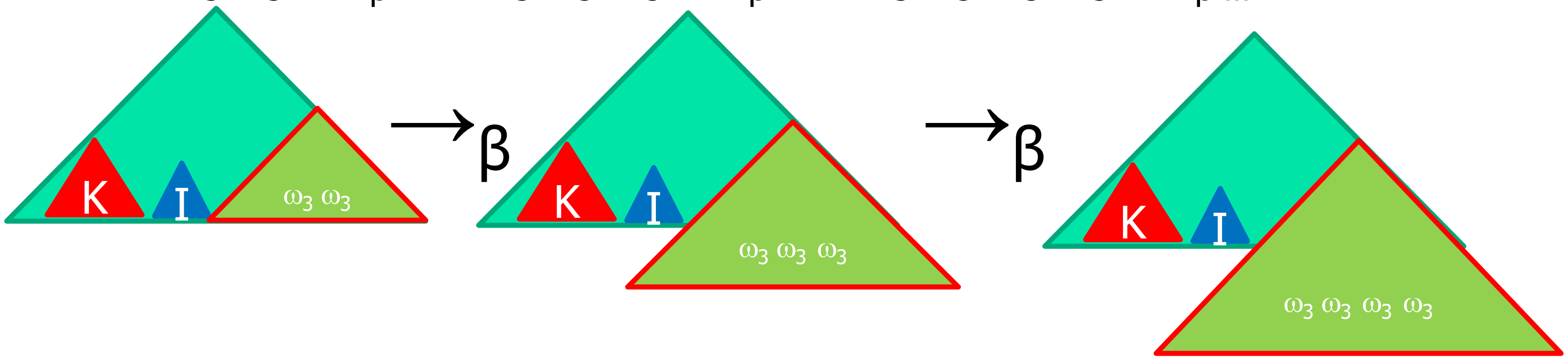
Otázku, ktorú by si mal položiť:

- v akom poradí, preorder, inorder, ...
- a záleží na poradí ?

$\omega = \lambda x.xx$
$\Omega = \omega\omega$
$\omega_3 = \lambda x.xxx$

Vlastnosti β-redukcie

- existuje nekonečná sekvencia
  - $\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \dots$
- existuje neobmedzene puchnúca sekvencia
  - $\omega_3\ \omega_3 \to_\beta \omega_3\ \omega_3\ \omega_3 \to_\beta \omega_3\ \omega_3\ \omega_3\ \omega_3$
- nejednoznačný výsledok – existuje term s konečným a nekonečným odvodením
  - $KI\Omega \to_\beta I$ ale aj
  - $KI\Omega \to_\beta KI\Omega \to_\beta KI\Omega \to_\beta \dots$
  - $KI(\omega_3\ \omega_3) \to_\beta KI\ (\omega_3\ \omega_3\ \omega_3) \to_\beta KI\ (\omega_3\ \omega_3\ \omega_3\ \omega_3) \to_\beta \dots$



Stratégia redukcie
(na výbere záleží)

- **βη-redex**
  je podterm λ-termu, ktorý môžeme prepísať β alebo η redukciou

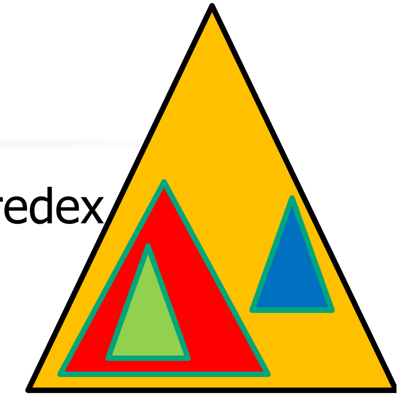
- **normálna forma** λ-termu nemá redex

- **reducibilný** λ-term nie je v normálnej forme

- **Stratégia redukcie** μ je čiastočné zobrazenie λ-termov, že $M \rightarrow_{\beta\eta} \mu(M)$

- μ výpočet je postupnosť $M, \mu(M), \dots, \mu^i(M), \dots$ a môže byť (ne)konečná

Najznámejšie stratégie



- **leftmost-innermost** – najľavejší redex neobsahuje iný redex

  $(\lambda x.(\underline{(\lambda y.y)}\ x))\ (\lambda z.z) \to_\beta \underline{(\lambda x.x)}\ (\lambda z.z) \to_\beta (\lambda z.z)$

  hint: $\underline{(\lambda y.y)}\ x \to_\beta x$

  <u>redex redukovaný stratégiou je podčiarknutý</u>

- **leftmost-outermost** – najľavejší redex neobsiahnutý v inom redexe

  $\underline{(\lambda x.((\lambda y.y)\ x))\ }(\lambda z.z) \to_\beta (\underline{(\lambda y.y)}\ (\lambda z.z)) \to_\beta (\lambda z.z)$

  hint: $\underline{(\lambda x.((\lambda y.y)\ x))\ }(\lambda z.z) \to_\beta \underline{((\lambda y.y)\ x)[x:}\lambda z.z] = (\underline{(\lambda y.y)\ (\lambda z.z)})$

- **leftmost stratégia** – vyhodnotí funkciu skôr ako argumenty

  $\underline{(\lambda x.x)}(\lambda y.((\lambda z.z)\ y)) \to_\beta (\lambda y.(\underline{(\lambda z.z)}\ y)) \to_\beta (\lambda y.y)$

- **rightmost stratégia** – vyhodnotí argumenty skôr ako funkciu

  $(\lambda x.x)(\lambda y.(\underline{(\lambda z.z)}\ y)) \to_\beta \underline{(\lambda x.x)}(\lambda y.y) \to_\beta (\lambda y.y)$

Ako to programujeme

- Extrémne drahé riešenie (prečo) ?

```
nf      :: LExp -> LExp        -- test na normálnu formu
nf t    = if t == t' then t else nf t' where t'=oneStepβ t
-----------------------------------------------------
```

((λx.M)N)

```
oneStepβ (APP (LAMBDA x m) n) = substitute m x n
```

(M N)

```
oneStepβ (APP m n)    = if m == m' then
                               (APP m (oneStepβ n))
                          else
                               (APP m' n)
                          where m' = oneStepβ m
```

(λx.M)

```
oneStepβ (LAMBDA x m) =  (LAMBDA x (oneStepβ m))
```

- je to innermost či outermost ? Ako vyzerá to druhé ??

Stratégie β-redukcie

Normalizujúca stratégia je taká, ktorá nájde normálnu formu, ak existuje

- **leftmost-innermost** (nie je normalizujúca stratégia)
  - argumenty funkcie sú zredukované skôr ako telo funkcie
  - $K I \Omega \to_\beta K I \Omega \to_\beta K I \Omega \to_\beta \ldots$ zacyklí sa pri leftmost-innermost
  - $(\lambda x.x\ x)\ (\underline{\lambda x.x}\ y) \to_\beta \underline{(\lambda x.x\ x)}y \to_\beta (y\ y)$
- **leftmost-outermost** (je normalizujúca stratégia, toto nie dôkaz !!!)
  - ak je možné dosadiť argumenty do tela funkcie, urobí sa tak ešte pred ich vyhodnotením, ale tým aj kopíruje redexy ☹
  - $K I \Omega \to_\beta I$
  - $\underline{(\lambda x.x\ x)}\ (\lambda x.x\ y) \to_\beta (\underline{\lambda x.x}\ y)\ (\lambda x.x\ y) \to_\beta y\ (\underline{\lambda x.x}\ y) \to_\beta y\ y$

    Call by need (lazy)
    - pri aplikácii funkcie sa do jej tela nedosadzuje argument, ale pointer na hodnotu argumentu, ktorý sa časom event. vyhodnotí

Churchove čísla

- $\underline{0}$:= λf.λx.x
- $\underline{1}$:= λf.λx.f x
- $\underline{2}$:= λf.λx.f (f x)
- …
- $\underline{n}$:= λf.λx.f^n x

- succ := λn.λf.λx.(f (n f x))= f(f^n x)
- plus := λm.λn.λf.λx. m f (n f x)

definujte mult

mult := λm.λn.λf.λx. n (m f) x

lebo (*m* f) = λx.(f^m x), potom (*n* (*m* f)) = λx.(($f^m)^n$ x) = λx.(f^{m*n} x)

•definujte m^n

exp := λm.λn. n m

exp m n f = n m f = ((n m) f) = $(m^n\ f)$

exp m n f x = $(m^n\ f)\ x = f^{(m\wedge n)}\ x$

•definujte <u>n-1</u> (na rozmýšľanie)

Churchove čísla
(pomalšie)

- $\underline{\mathit{0}} := \lambda f.\lambda x.x$
- $\underline{\mathit{1}} := \lambda f.\lambda x.f\ x$
- $\underline{\mathit{2}} := \lambda f.\lambda x.f\ (f\ x)$
- …
- $\underline{\mathit{n}} := \lambda f.\lambda x.f^n\ x$

- succ := $\lambda n.\lambda f.\lambda x.(f\ (n\ f\ x))$
- plus := $\lambda m.\lambda n.\lambda f.\lambda x.\ ((m\ f)\ (n\ f\ x))$

(succ $\underline{\mathit{2}}$) = $(\lambda n.\lambda f.\lambda x.(f\ (n\ f\ x))\ \lambda f.\lambda x.f\ (f\ x)\ ) = \lambda f.\lambda x.f((\lambda f'.\lambda x'.f'\ (f'\ x')\ )\ f\ x) = \lambda f.\lambda x.(f\ (f\ (f\ x))) = \underline{\mathit{3}}$

((plus $\underline{\mathit{2}}$) $\underline{\mathit{2}}$) $= ((\lambda m.\lambda n.\lambda f.\lambda x.\ (m\ f)\ (n\ f\ x)\ \underline{\mathit{2}}\ )\ \underline{\mathit{2}}\ ) =$
$((\lambda n.\lambda f.\lambda x.\ (\underline{\mathit{2}}\ f)\ (n\ f\ x))\ \underline{\mathit{2}}\ ) =$
$(\lambda f.\lambda x.\ (\underline{\mathit{2}}\ f)\ (\underline{\mathit{2}}\ f\ x)) =$
$(\lambda f.\lambda x.\ ((\lambda f'.\lambda x'.f'\ (f'\ x'))\ f)\ (\underline{\mathit{2}}\ f\ x)) =$
$(\lambda f.\lambda x.\ (\lambda x'.f\ (f\ x'))\ (\underline{\mathit{2}}\ f\ x)) =$
$(\lambda f.\lambda x.\ (f\ (f\ (\underline{\mathit{2}}\ f\ x)))) =$
$(\lambda f.\lambda x.\ (f\ (f\ (\lambda f.\lambda x.f\ (f\ x)\ f\ x)))) = \quad (\lambda f.\lambda x.\ (f\ (f\ (f\ (f\ x))))) = \underline{\mathit{4}}$

Už ste to raz videli

(prvá prednáška)

succ := λn.λf.λx.f(n f x)
plus := λm.λn.λf.λx. m f (n f x)
mult := λm.λn.λf.λx. n (m f) x

```
true  x y = x
false x y = y

ifte  c t e = c t e

two   f x = f (f x)
one   f x = f x
zero  f x = x

incr n f x = f (n f x)
incr = \n->\f->\x->(f (n f x))
add   m n f x = m f (n f x)
add = \m->\n->\f->\x->((m f) (n f x))
mul   m n f x = m (n f) x

isZero n =  n (\_ -> false) true

decr n = n (\m f x -> f (m incr zero))
           zero
           (\x -> x)
           zero
```

Pokúste sa dokázať decr <u>n</u> = <u>n-1</u>.

```
fact :: (forall a. (a->a)->a->a) -> (a->a) -> a -> a
fact n  =
     ifte (isZero n)
          one
          (mul n (fact (decr n)))

main =
  -- print $ (decr (add (mul two two) one)) (+1) 0
  -- print $ (fact (add (mul two two) one)) (+1) 0
  print $ (fact (add two
                (add (mul two two) (mul two two))))
         (+1) 0

-- 3628800
-- (4.75 secs, 2,598,673,208 bytes)
```

Aký je rozdiel medzi týmto interpretrom a tým, čo máte nakódiť na domácu úlohu ?

- Haskell realizuje β-redukciu
- Haskell je striktne typovaný

Súbor: Church.hs

$n\,(+1)\ 0 = n$

$n\ f\ x = f^n\ x$

Testovanie domácej úlohy

Potrebujeme prirodzené čísla, použijeme konštrukciu podľa A.Churcha:

- *0* := λf.λx.x
- *1* := λf.λx.f x
- *2* := λf.λx.f (f x)

- succ := λn.λf.λx.f (n f x) = λn.λf.λx.(f ((n f) x))
- plus := λm.λn.λf.λx. m f (n f x) = λm.λn.λf.λx. ((m f) ((n f) x))
  -- idea: $f^m(f^n\ x) = f^{m+n}\ x$

Zadáme tieto dve konštrukcie:

```
zero = (LAMBDA "f" (LAMBDA "x" (ID "x")))
succ = (LAMBDA "n" (LAMBDA "f" (LAMBDA "x"
                    (APP (ID "f") (APP (APP (ID "n") (ID "f")) (ID "x"))))))
```

potom vypočítame:

```
one = (APP succ zero)= LAMBDA "f" (LAMBDA "x" (APP (ID "f") (ID "x")))
two = (APP succ one) = LAMBDA "f" (LAMBDA "x" (APP (ID "f") (APP (ID "f") (ID "x"))))
three = (APP succ two)
    = LAMBDA "f" (LAMBDA "x" (APP (ID "f") (APP (ID "f") (APP (ID "f") (ID "x")))))
```

Logika a predikáty

TRUE	:= λx.λy. x := λxy.x = K	(vráti 1.argument)
FALSE	:= λx.λy. y := λxy.y	(vráti 2.argument)

AND	:= λx.λy. ((x y) FALSE) := λxy.x y FALSE
AND x y	= ((x y) FALSE)
OR	:= λx.λy. ((x TRUE) y) := λxy.x TRUE y
OR x y	= x TRUE y
NOT	:= λx. x FALSE TRUE
IFTHENELSE	:= λc.λx.λy.(c x y)

Príklad:
AND TRUE FALSE
≡ (λ x y. x y FALSE) TRUE FALSE $\to_\beta$ TRUE FALSE FALSE
≡ (λ x y. x) FALSE FALSE $\to_\beta$ FALSE

definujte XOR

Kartézsky súčin typov (pár)

PAIR := $\lambda x.\lambda y.(\lambda c.\ c\ x\ y)$:= $\lambda xyc.\ c\ x\ y$
LEFT := $\lambda x.(x\ TRUE)$
RIGHT := $\lambda x.(x\ FALSE)$

PAIR A B = $(\lambda c.\ ((c\ A)\ B))$

TRUE := $\lambda x.\lambda y.x$:= $\lambda xy.x$
FALSE := $\lambda x.\lambda y.y$:= $\lambda xy.y$

LEFT (PAIR A B) ≡

$$\mathrm{LEFT}\ ((\lambda xyc.\ c\ x\ y)\ A\ B) \to_\beta$$
$$\mathrm{LEFT}\ (\lambda c.\ c\ A\ B) \to_\beta$$
$$(\lambda x.x\ \mathrm{TRUE})\ (\lambda c.\ c\ A\ B) \to_\beta$$
$$(\lambda c.\ c\ A\ B)\ (\lambda xy.x) \to_\beta$$
$$((\lambda xy.x)\ A\ B) \to_\beta\ A$$

definujte 3-ticu.

Konštrukcia n-tice nás oprávňuje písať n-árne funkcie, t.j. funkcie, ktorých argumentom je n-tica – tzv. currying, na počesť pána Haskell Curry:

```
curry   :: ((a,b) -> c) -> a -> b -> c
uncurry :: (a -> b -> c) -> (a,b) -> c
```

$\lambda(x,y).M$ vs. $(\lambda x.\lambda y.M)$

$\lambda(x,y).M$ -> $\lambda p.\ (\lambda x.\lambda y.M)\ (\mathrm{LEFT}\ p)\ (\mathrm{RIGHT}\ p)$

Bolo kedysi

```
dvojica a b = pair
    where pair f = f a b

prvy p = p (\ a-> \b -> a)

druhy p = p (\ a-> \b -> b)
```

```
def cons(a, b):
    def pair(f):
        return f(a, b)
    return pair

def head(p):
    return p(lambda a,b :a)

def tail(p):
    return p(lambda a,b:b)

print(head(cons(4,5)))
print(tail(cons(4,5)))
```

```
def pair(a,b):
    return lambda f: f(a,b)

def fst(p):
    return p(lambda a,b: a)

def snd(p):
    return p(lambda a,b: a)

print(fst(pair(4,5)))
```

Všetko, čo by ste chceli vediet o Haskelli, ale báli ste sa spýtať

Otázka z interview

a ako na ňu

Haskell:

```
dvojica a b = pair
    where pair f = f a b
dvojica :: s->t->((s->t->v)->v)
-- inak
dvojica a b = \f -> f a b
dvojica a b f = f a b
prvy (dvojica a b) = a
druhy (dvojica a b) = b
prvy p = p (\x-> \y -> x)
druhy p = p (\x-> \y -> y)
print $ prvy (dvojica 4 5)
print $ druhy (dvojica 4 5)
prvy p = p true
   where true a b = a
druhy p = p false
    where false a b = b
```

Python, JS, ...:

```
def dvojica(a, b):
    def pair(f):
        return f(a, b)
    return pair
# inak
def dvojica(a,b):
    return lambda f: f(a,b)

def head(p):
    return p(lambda a,b :a)

def tail(p):
    return p(lambda a,b:b)

print(head(dvojica(4,5)))
print(tail(dvojica(4,5)))
```

pair.hs, pair.py

PAIR := $\lambda x.\lambda y.(\lambda c.\ c\ x\ y)$:= $\lambda xyc.\ c\ x\ y$

Súčet typov (disjunkcia)

A+B reprezentujeme ako dvojicu Bool x (A|B)
teda (TRUE, A) alebo (FALSE, B)

1^{st}	:= $\lambda x.$PAIR TRUE x	konštruktor pre A
2^{nd}	:= $\lambda y.$PAIR FALSE y	B
1^{st-1}	:= $\lambda z.$RIGHT z	deštruktor pre A
2^{nd-1}	:= $\lambda z.$RIGHT z	B
$?1^{st-1}$	:= $\lambda z.$LEFT z	test, či A ?

$1^{st-1}\ 1^{st}\ A \equiv$
$(\lambda z.\text{RIGHT } z)\ (\lambda x.\text{PAIR TRUE } x)\ A \rightarrow_\beta$
$\text{RIGHT (PAIR TRUE A)} \rightarrow_\beta A$

Zoznamy

Domáca úloha (vašim interpretrom):
•null? (cons a Nil) $\rightarrow_\beta^*$
•head (cons a Nil) $\rightarrow_\beta^*$
•tail (cons a Nil) $\rightarrow_\beta^*$
•head (tail (cons a (cons b Nil)))

- List t = Nil | Cons t (List t)

Nil = λz.z TRUE FALSE FALSE
Cons = λx.λy.λz.z FALSE x y

head = λp.p (λx.λy.λz.y)
tail = λp.p (λx.λy.λz.z)
isNil = λp.p (λx.λy.λz.x)
Odvoďme, napr.:
isNil Nil = (λp.p (λx.λy.λz.x)) (λz.z TRUE FALSE FALSE) $\rightarrow_\beta$
((λz.z TRUE FALSE FALSE) (λx.λy.λz.x)) $\rightarrow_\beta$
((λx.λy.λz.x) TRUE FALSE FALSE) $\rightarrow_\beta$
TRUE

Binárne stromy

- BinTree t = Empty | Node t (BinTree t) (BinTree t)

Empty	$= \lambda g.g\ \mathrm{TRUE}\ (\lambda x.x)\ (\lambda x.x)\ (\lambda x.x)$
Node	$= \lambda x.\lambda y.\lambda z.\lambda g.g\ \mathrm{FALSE}\ x\ y\ z$

isEmpty	$= \lambda t.t\ (\lambda u.\lambda x.\lambda y.\lambda z.u)$
root	$= \lambda t.t\ (\lambda u.\lambda x.\lambda y.\lambda z.x)$
left	$= \lambda t.t\ (\lambda u.\lambda x.\lambda y.\lambda z.y)$
right	$= \lambda t.t\ (\lambda u.\lambda x.\lambda y.\lambda z.z)$

Binárne stromy

Odvoďme, napr.:

root (Node a Empty Empty) $\rightarrow_\beta$

(λt.t (λu.λx.λy.λz.x)) (Node a Empty Empty) $\rightarrow_\beta$

((Node a Empty Empty) (λu.λx.λy.λz.x)) $\rightarrow_\beta$

(((λx.λy.λz.λg.g FALSE x y z) a Empty Empty) (λu.λx.λy.λz.x)) $\rightarrow_\beta$

((λg.g FALSE a Empty Empty) (λu.λx.λy.λz.x)) $\rightarrow_\beta$

((λu.λx.λy.λz.x) FALSE a Empty Empty) (λu.λx.λy.λz.x)) $\rightarrow_\beta$

((λu.λx.λy.λz.x) FALSE a Empty Empty)) $\rightarrow_\beta$

a

where/let

let v = N in M

vlastne to isté ako

M where v = N $\rightarrow$ $(\lambda v.M)\ N$

M where $v_1 = N_1$ $\rightarrow$ $(\lambda(v_1, v_2, \ldots, v_n).M)\ (N_1, \ldots, N_n)$

$v_2 = N_2$ …

$v_n = N_n$

zložený where

n*(x+n) where $\rightarrow$ $(\lambda n.\ (\lambda x.n*(x+n))\ (4*n+1))\ 3$

n = 3

x = 4*n+1

Rekurzia

To, čo stále nevieme, je definovať rekurzívnu funkciu, resp. cyklus.
Na to sa používa konštrukcia pomocou operátora pevného bodu.

Príklad:
FAC := λn.(if (= n 0) 1 (* n (FAC (- n 1))))

FAC := λn.if (n = 0) then 1 else (n * FAC (n - 1))
... trik: η-redukcia (λx.M x) = M, ak x nie je Free(M)
FAC := (λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1))))) FAC)

H := λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1)))))

hľadáme funkciu FAC, ktorá má túto vlastnosť:
FAC := (H FAC) f x = x
hľadaná funkcia FAC je *pevný bod* funkcie H

- Aký je typ Y ??

Pevný bod

Potrebujeme trochu teórie:

Veta:
Pre ľubovoľný λ-term F existuje pevný bod, t.j. X také, že X = F X.

Dar nebies (operátor pevného bodu):

$Y = \lambda f.(\lambda x. (f(x\ x)))\ (\lambda x. f(x\ x))$

potom

(Y F) je pevný bod F, t.j. (Y F) = F (Y F).

Skúsme to (aspoň) overiť:

$Y\ F = (\lambda f.(\lambda x. f(x\ x))\ (\lambda x. f(x\ x)))\ F \rightarrow_\beta (\lambda x. F(x\ x))\ (\lambda x. F(x\ x)) \rightarrow_\beta$

- $(\lambda x'. F(x'\ x'))\ (\lambda x. F(x\ x)) \rightarrow_\beta F(x'\ x')[x':\lambda x. F(x\ x)] \rightarrow_\beta$
- $F(\lambda x.F(x\ x)\ \lambda x.F(x\ x)) =$
- F (Y F)

preto (Y F) je naozaj pevný bod

a je jediný ?

Operátor Y

FAC := (H FAC)
FAC := Y H
H:=λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1)))))
Platí
Y H = H (Y H)

Presvedčíme sa, že Y nám pomôže definovať rekurzívnu funkciu:

FAC = Y H = Y (λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1))))))
(λf.(λx. f(x x)) (λx. f(x x))) (λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1))))))
– toto je faktoriál – verzia nevhodná pre slabšie povahy

FAC 1 = (Y H) 1 ... z vlastnosti pevného bodu Y H=H (Y H)
= (H (Y H)) 1
= λfac.(λn.(if (= n 0) 1 (* n (fac (- n 1))))) (Y H) 1
= λn.(if (= n 0) 1 (* n ((Y H)(- n 1)))) 1
= if (= 1 0) 1 (* 1 ((Y H) (- 1 1)))
= (* 1 ((Y H) (- 1 1)))
= (* 1 ((Y H) 0))
= (* 1 (H (Y H) 0)) ... trochu zrýchlene
= (* 1 1)
= 1

1+2+3+…+n

SUM = λs.λn.if (= n 0) 0 (+ n (s (- n 1)))

(Y SUM) 2 =

- Y (λs.λn.if (= n 0) 0 (+ n (s (- n 1)))) 2
- (λs.λn.if (= n 0) 0 (+ n (s (- n 1)))) (Y SUM) 2
- (λn.if (= n 0) 0 (+ n ((Y SUM) (- n 1)))) 2
- if (= 2 0) 0 (+ 2 ((Y SUM) (- 2 1)))
- (+ 2 ((Y SUM) 1))
- (+ 2 ((λs.λn.if (= n 0) 0 (+ n (s (- n 1)))) (Y SUM) 1))
- (+ 2 ((λn.if (= n 0) 0 (+ n ((Y SUM) (- n 1)))) 1))
- (+ 2 ((if (= 1 0) 0 (+ n ((Y SUM) (- 1 1))))))
- (+ 2 (+ 1 ((Y SUM) 0)))
- (+ 2 (+ 1 ((λs.λn.if (= n 0) 0 (+ n (s (- n 1)))) (Y SUM) 0)))
- (+ 2 (+ 1 ((λn.if (= n 0) 0 (+ n ((Y SUM) (- n 1)))) 0)))
- (+ 2 (+ 1 ((if (= 0 0) 0 (+ 0 ((Y SUM) (- 0 1)))))))
- (+ 2 (+ 1 0)) = 3

Cvičenie

- (na zamyslenie) nájdite príklady funkcií
  s nekonečným počtom pevných bodov
  s práve jedným pevným bodom, fix-point : f x = x

- realizujte interpreter λkalkulu, pokračujte v kóde z minulého cvičenia tak, aby počítal hodnoty rekurzívnych funkcii

--sucet = \s -> \n -> (if (= n 0) 0 (+ n (s (- n 1))))

```
sucet =     LAMBDA "s"
           (LAMBDA "n"
            (APP
             (APP
               (APP
                 (CON "IF")
                   (APP (APP (CON "=") (ID "n")) (CN 0))   -- condition
               )
               (CN 0) -- then
             )
             (APP (APP (CON "+")    -- else
                     (ID "n"))
                     (APP (ID "s")
                       (APP (APP (CON "-") (ID "n")) (CN 1))
                       )
                   )
             )
            )
           )
```

Cvičenie

Y = λf.(λx. f(x x)) (λx. f(x x))

```
-- plati Y f = f(Y f)
y =  LAMBDA "f"
     (APP (LAMBDA "x" (APP (ID "f") (APP (ID "x") (ID "x"))))
          (LAMBDA "x" (APP (ID "f") (APP (ID "x") (ID "x")))))
```

Vyhodnoťte (APP (APP y sucet) CN 4)
1+2+3+4 = 10 ?
A čo faktorial ?

Poznámka:
Obohaťte Váš interpreter o vstavané celé čísla so základnými operáciami (+1, -1, +, *), plus test (napr. na nulu). V opačnom prípade budete bojovať s Church.číslami a interpreter sa vám bude ťažšie ľadiť.

Viacnásobná rekurzia
(idea)

foo x y = … (goo x′ y′) … (foo x″ y″) …
goo x y = … (foo x′ y′) … (foo x″ y″) …

vektorovo:

foogoo (x,y) = (
… snd \$ (foogoo (x′,y′)) … fst \$ (foogoo (x″, y″)) …
,
… fst \$ (foogoo (x′, y′)) … snd \$ (foogoo (x″, y″)) …
)

$\underline{\mathbf{X}}$ **=** (x,y) = λ$\underline{\mathbf{z}}$**.**(foo (fst $\underline{\mathbf{z}}$, snd $\underline{\mathbf{z}}$), goo (fst $\underline{\mathbf{z}}$, snd $\underline{\mathbf{z}}$)) $\underline{\mathbf{X}}$

preto

$\underline{\mathbf{X}}$ **= Y (** λ$\underline{\mathbf{z}}$**.**(foo (fst $\underline{\mathbf{z}}$, snd $\underline{\mathbf{z}}$), goo (fst $\underline{\mathbf{z}}$, snd $\underline{\mathbf{z}}$)) **)**

Viacnásobná rekurzia

Veta o pevnom bode: Pre ľubovoľné $F_1, F_2, \ldots, F_n$ existujú $X_1, X_2, \ldots, X_n$, že

$$X_1 = F_1\, X_1\, X_2 \ldots X_n$$
$$X_2 = F_2\, X_1\, X_2 \ldots X_n$$
$$\ldots \qquad \ldots$$
$$X_n = F_n\, X_1\, X_2 \ldots X_n.$$

vektorovo:

$$(X_1, X_2, \ldots, X_n) = (F_1\, X_1\, X_2 \ldots X_n,\ F_2\, X_1\, X_2 \ldots X_n, \ldots,\ F_n\, X_1\, X_2 \ldots X_n)$$

$$\underline{\mathbf{X}} = (F_1\, (p_1\, \underline{\mathbf{X}})\, (p_2\, \underline{\mathbf{X}}) \ldots (p_n\, \underline{\mathbf{X}}), \ldots, F_n\, (p_1\, \underline{\mathbf{X}})\, (p_2\, \underline{\mathbf{X}}) \ldots (p_n\, \underline{\mathbf{X}}))$$

$$\underline{\mathbf{X}} = \boxed{\lambda \underline{\mathbf{z}}.(F_1\, (p_1\, \underline{\mathbf{z}})\, (p_2\, \underline{\mathbf{z}}) \ldots (p_n\, \underline{\mathbf{z}}), \ldots F_n\, (p_1\, \underline{\mathbf{z}})\, (p_2\, \underline{\mathbf{z}}) \ldots (p_n\, \underline{\mathbf{z}}))}\ \underline{\mathbf{X}}$$

p_i = i-ta projekcia vektora.

preto

$$\underline{\mathbf{X}} = \mathbf{Y}\, (\lambda \underline{\mathbf{z}}.(F_1\, (p_1\, \underline{\mathbf{z}})\, (p_2\, \underline{\mathbf{z}}) \ldots (p_n\, \underline{\mathbf{z}}), \ldots F_n\, (p_1\, \underline{\mathbf{z}})\, (p_2\, \underline{\mathbf{z}}) \ldots (p_n\, \underline{\mathbf{z}})\ )\ )$$

Primitívna rekurzia
(ideovo)

Rózsa Peter, founding mother of recursion theory

Due to the effects of the Great Depression, many university graduates could not find work and Péter began private tutoring. At this time, she also began her graduate studies.

https://en.wikipedia.org/wiki/R%C3%B3zsa_P%C3%A9ter

Primitívne rekurzívna funkcia je:

- 0, resp. nulová funkcia $N^n \rightarrow N$,
- +1, resp. succ: $N \rightarrow N$,
- $p_i\, x_1\, x_2 \dots x_n = x_i$ resp. projekcia p_i: $N^n \rightarrow N$, $p_i\, x_1\, x_2 \dots x_n = x_i$
- f . g resp. kompozícia
  $f\, x_1\, x_2 \dots x_n = g(h_1(x_1\, x_2 \dots x_n) \dots h_m(x_1\, x_2 \dots x_n))$
- číselná rekurzia z n+1 na n:

primitívna rekurzia $g : N^n \rightarrow N$, $h : N^{n+2} \rightarrow N$, potom $f : N^{n+1} \rightarrow N$

$f\, \mathbf{0}\, x_1\, x_2 \dots x_n = g(x_1\, x_2 \dots x_n)$

$f\, (\mathbf{n+1})\, x_1\, x_2 \dots x_n = h(\, f(\mathbf{n}\, x_1\, x_2 \dots x_n)\, \mathbf{n}\, x_1\, x_2 \dots x_n)$

Primitívna rekurzia

Primitívne rekurzívne funkcia je totálna, resp. všade definovaná

- je akákoľvek číselná totálna funkcia (N->N) primitívne rekurzívna ?
- je akákoľvek číselná "haskellovská" totálna funkcia primitívne rekurzívna ? (je predpoklad n+1 -> n vážne obmedzujúci ?)
- je, Ackermannova funkcia, je jednoduchým príkladom funkcie, ktorá nie je primitívne rekurzívna (1935):

$$
\begin{aligned}
\mathrm{A}(0,n) &= n+1 \\
\mathrm{A}(m+1,0) &= \mathrm{A}(m,1) \\
\mathrm{A}(m+1,n+1) &= \mathrm{A}(m,A(m+1,n))
\end{aligned}
$$

- Ackermannova funkcia rastie rýchlejšie ako akákoľvek primitívne rek.

https://en.wikipedia.org/wiki/Primitive_recursive_function

https://en.wikipedia.org/wiki/Fast-growing_hierarchy

Viac než primitívna rekurzia

Primitívne rekurzívna funkcia je:

- nulová funkcia $N^n \to N$,
- succ: $N \to N$,
- projekcia p_i: $N^n \to N$, $p_i\ x_1\ x_2 \ldots x_n = x_i$
- kompozícia $f\ x_1\ x_2 \ldots x_n = g(h_1(x_1\ x_2 \ldots x_n) \ldots h_m(x_1\ x_2 \ldots x_n))$
- primitívna rekurzia $g : N^n \to N$, $h : N^{n+2} \to N$, potom $f : N^{n+1} \to N$

  $f\ 0\ x_1\ x_2 \ldots x_n = g(x_1\ x_2 \ldots x_n)$

  $f\ (n+1)\ x_1\ x_2 \ldots x_n = h(f(n\ x_1\ x_2 \ldots x_n)\ n\ x_1\ x_2 \ldots x_n)$

Parciálne/Čiastočne vyčíslitelná/rekurzívna (nemusí byť totálna):

nech $r : N^{n+1} \to N$ je primitívne rekurzívna funkcia

- μ-rekurzia definuje $f : N^{n+1} \to N$

  $f\ y\ x_1\ x_2 \ldots x_n = \min_z.(r(z\ x_1\ x_2 \ldots x_n) = y)$

pohľad programátora:

$f\ y\ x_1\ x_2 \ldots x_n =$

for (int z = 0; ; z++)

if (r(z x_1 x_2 … x_n) == y) return z;

λ-vypočítateľná funkcia
(technické – na dlhé letné večery)

Parciálna funkcia $f : N^n \to N$ je **λ-vypočítateľná**, ak existuje λ-term F taký, že
$F\ \underline{x_1}\ \underline{x_2} \dots \underline{x_n}$ sa zredukuje na $\underline{f\ x_1\ x_2 \dots x_n}$, ak n-tica $x_1\ x_2 \dots x_n$ patrí do def.oboru f
$F\ \underline{x_1}\ \underline{x_2} \dots \underline{x_n}$ nemá normálnu, ak n-tica $x_1\ x_2 \dots x_n$ nepatrí do def.oboru f

Veta: Každá parciálne vyčíslitelná funkcia je λ-vypočítateľná.
Dôkaz:
- nulová fcia, succ, projekcie $p_{i,}$ kompozícia priamočiaro
- primitívna rekurzia $g : N^n \to N$, $h : N^{n+2} \to N$, potom $f : N^{n+1} \to N$

  $f\ 0\ \ x_1\ x_2 \dots x_n = g(x_1\ x_2 \dots x_n)$

  $f\ (n+1)\ x_1\ x_2 \dots x_n = h(f(n\ x_1\ x_2 \dots x_n)\ n\ x_1\ x_2 \dots x_n)$

  $F = \mathbf{Y}\ (\lambda f.\lambda y.\lambda x_1.\lambda x_2 \dots \lambda x_{n\,.}$ (if (isZero y) $G(x_1\ x_2 \dots x_n)$ then
  else $H(f((\text{pred } y)\ x_1\ x_2 \dots x_n)\ (\text{pred } y)\ x_1\ x_2 \dots x_n)))$
- μ-rekurzia $r : N^{n+1} \to N$

  $F = \lambda y \lambda x_1 \lambda x_2 \dots \lambda x_n.\mathbf{Y}\ (\lambda h.\lambda z._{}$ (if (eq y $G(z\ x_1\ x_2 \dots x_n)$) then z else h (succ z)))

Veta: Každá λ-vypočítateľná je parcialne vyčíslitelná funkcia.

Weak head normal form

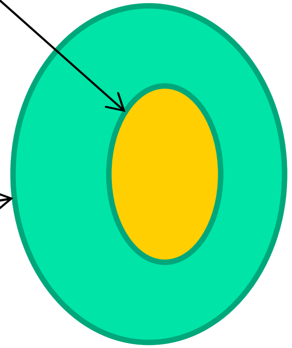
(slabo hlavová normálna forma)

Head normal form (h.n.f)

- $(\lambda x_1.\ \lambda x_2.\ \dots\ .\ \lambda x_k.\ v)\ M_1\ M_2 \dots\ M_n$ $(k, n \in N)$
- v je premenná (resp. konštanta),
- pre ľubovoľné $r \le n$, $(\dots((v\ M_1)\ M_2)\dots\ M_r)$ nie je redex .

Ak k=0, konštanta či premenná s málo argumentami

Ak k>0, λ-abstrakcia s nereducibilným telom



Weak head normal form (w.h.n.f)

- $v\ M_1\ M_2 \dots\ M_n$ $(n \in N)$
- v je premenná alebo λ-abstrakcia (resp. konštanta),
- pre ľubovoľné $r \le n$, $(\dots((v\ M_1)\ M_2)\dots\ M_r)$ nie je redex .

Konštanta, premenná alebo λ-abstrakcia s málo argumentami.

$\lambda x.((\lambda y.y)\ z)$ nie je h.n.f. (až po red. $((\lambda y.y)\ z) \to_\beta z$), ale je w.h.n.f.

Najznámejšie stratégie

- weak leftmost outermost (call by need/output driven/lazy/full lazy)

  $\underline{(\lambda x.\ \lambda y.(x\ y))}\ (\lambda z.z) \to_{\beta} \lambda y.((\lambda z.z)\ y)$ w.h.n.f.

  redukuje argumenty funkcie, len ak ich treba

  Keďže w.h.n.f. môže obsahovať redex, tak nenormalizuje úplne...

- strong leftmost outermost (call by name/demand driven)

  $\underline{(\lambda x.\ \lambda y.(x\ y))}\ (\lambda z.z) \to_{\beta} \lambda y.\underline{((\lambda z.z)\ y)} \to_{\beta} \lambda y.y$ n.f.

  redukuje argumenty funkcie, len ak ich treba, ale pokračuje v hľadaní redexov, kým nejaké sú

  normalizuje úplne...

- eager - argumenty najprv (call by value/data driven/strict)

  nenormalizuje...

Lazy

- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)(*\ 3\ 4))\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(\ \lambda y.(*\ (+\ ((\lambda x.x)(*\ 3\ 4))\ ((\lambda x.x)(*\ 3\ 4)))\ y)\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(*\ (+\ ((\lambda x.x)(*\ 3\ 4))\ ((\lambda x.x)(*\ 3\ 4)))\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ (+\ (*\ 3\ 4)\ ((\lambda x.x)(*\ 3\ 4)))\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ (+\ 12\ ((\lambda x.x)(*\ 3\ 4)))\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ (+\ 12\ (*\ 3\ 4))\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ (+\ 12\ 12)\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ 24\ (\lambda x.(+2\ x)\ 6)\ ) \rightarrow_\beta$
- $(*\ 24\ (+2\ 6)\ \ ) \rightarrow_\beta$
- $(*\ 24\ 8\ ) \rightarrow_\beta$
- 192

Full lazy

- (λx. λy.(* (+ x x) y) ((λx.x)(* 3 4))) (λx.(+2 x) 6) $\rightarrow_\beta$
- (λy.(* (+ ((λx.x)(* 3 4)) ((λx.x)(* 3 4))) y)) (λx.(+2 x) 6) $\rightarrow_\beta$
- (* (+ ((λx.x)(* 3 4)) ((λx.x)(* 3 4))) (λx.(+2 x) 6)) $\rightarrow_\beta$
- (* (+ (* 3 4) (* 3 4)) (λx.(+2 x) 6)) $\rightarrow_\beta$
- (* (+ 12 12) (λx.(+2 x) 6)) $\rightarrow_\beta$
- (* 24 (λx.(+2 x) 6)) $\rightarrow_\beta$
- (* 24 (+2 6)) $\rightarrow_\beta$
- (* 24 8) $\rightarrow_\beta$
- 192

Strict

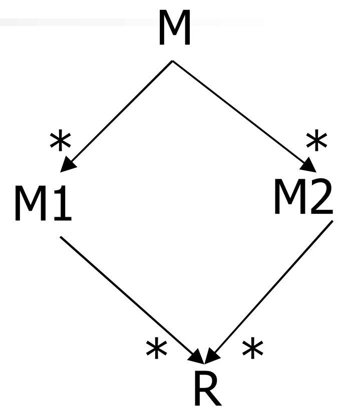
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)(*\ 3\ 4))\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)(*\ 3\ 4))\ )\ (+2\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)(*\ 3\ 4))\ )\ 8 \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ (*\ 3\ 4)\ )\ 8 \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ 12\ )\ 8 \rightarrow_\beta$
- $(\lambda y.(*\ (+\ 12\ 12)\ y))\ 8 \rightarrow_\beta$
- $(*\ (+\ 12\ 12)\ 8) \rightarrow_\beta$
- $(*\ 24\ 8) \rightarrow_\beta$
- 192

Eager

- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)(*\ 3\ 4))\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ ((\lambda x.x)\ 12)\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ 12\ )\ (\lambda x.(+2\ x)\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ 12\ )\ (+2\ 6) \rightarrow_\beta$
- $(\lambda x.\ \lambda y.(*\ (+\ x\ x)\ y)\ 12\ )\ 8 \rightarrow_\beta$
- $(\lambda y.(*\ (+\ 12\ 12)\ y))\ 8 \rightarrow_\beta$
- $(\lambda y.(*\ 24\ y))\ 8 \rightarrow_\beta$
- $(*\ 24\ 8) \rightarrow_\beta$
- 192

Church-Rosser vlastnosť

(konzistentnosť λ-kaklulu – *Janko/Marienka vlastnosť*)



pre ľubovoľnú trojicu termov M, M_1, M_2 takých, že

$M \rightarrow_{\beta}^{*} M_1$ a $\rightarrow_{\beta}^{*} M_2$

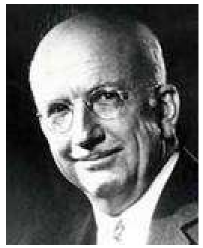
existuje R, že

$M_1 \rightarrow_{\beta}^{*} R$ a $M_2 \rightarrow_{\beta}^{*} R$

Inak:

$$(\leftarrow_{\beta}^{*} \circ \rightarrow_{\beta}^{*}) \subseteq (\rightarrow_{\beta}^{*} \circ \leftarrow_{\beta}^{*})$$

teda ak $M1 \leftarrow_{\beta}^{*} M \rightarrow_{\beta}^{*} M2$, potom existuje R, že $M1 \rightarrow_{\beta}^{*} R \leftarrow_{\beta}^{*} M2$

Veta: β-redukcia spĺňa Church-Rosserovu vlastnosť

Dôkazy sú technicky náročné:

- 1936 Church, Rosser: Some properties of conversion
- 1981 Barendregt
- 1981 Löf, Tait

Dôsledok:

ak term má normálnu formu vzhľadom na $\rightarrow_{\beta}$, potom je jednoznačne určená

Vysvetlenie

$$
\begin{tikzcd}
 & M \arrow[dl, "*"'] \arrow[dr, "*"] & \\
M1 \arrow[dr, "*"'] & & M2 \arrow[dl, "*"] \\
 & R &
\end{tikzcd}
$$

- $(\leftarrow_\beta^* \circ \rightarrow_\beta^*) \subseteq (\rightarrow_\beta^* \circ \leftarrow_\beta^*)$

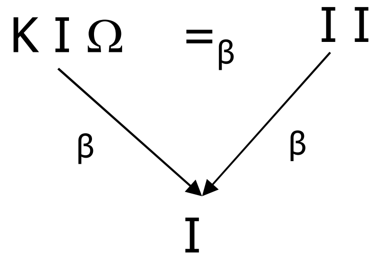
- $x\,(\leftarrow_\beta^* \circ \rightarrow_\beta^*)\,y \quad => \quad x\,(\rightarrow_\beta^* \circ \leftarrow_\beta^*)\,y$

- $\forall\, m, \exists\, r$:

  $x \leftarrow_\beta^* m \rightarrow_\beta^* y \quad => \quad x \rightarrow_\beta^* r \leftarrow_\beta^* y$

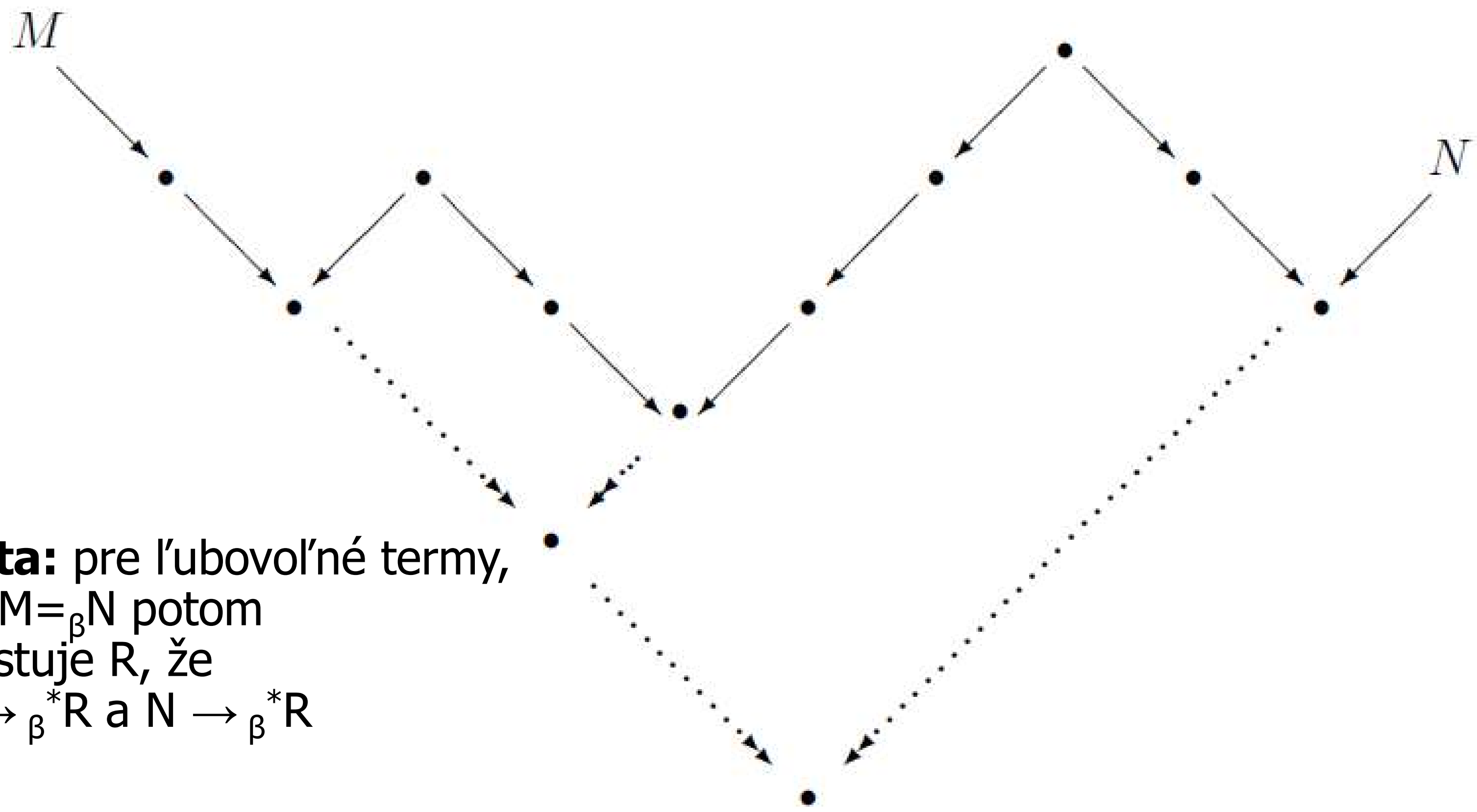
- $\forall\, m\ \exists\, r$:

  $x \leftarrow_\beta^* m \ \wedge\ m \rightarrow_\beta^* y \quad => \quad x \rightarrow_\beta^* r \ \wedge\ r \leftarrow_\beta^* y$



β ekvivalenica

- $=_\beta$ definujeme ako $(\rightarrow_\beta \cup \leftarrow_\beta)^*$ Príklad: $K I \Omega =_\beta I I$



Veta: pre ľubovoľné termy, ak $M =_\beta N$ potom existuje R, že $M \rightarrow_\beta^* R$ a $N \rightarrow_\beta^* R$

Slabá Church-Rosser vlastnosť

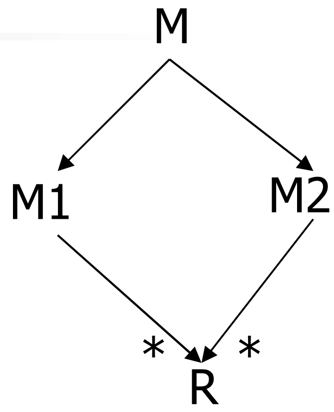
(slabá Janko/Marienka vlastnosť)

pre ľub.trojicu termov M, M_1, M_2 takých, že

$$M \rightarrow M_1 \text{ a } M \rightarrow M_2$$

existuje R, že

$$M_1 \rightarrow^* R \text{ a } M_2 \rightarrow^* R$$



Inak:

$$(\leftarrow \circ \rightarrow) \subseteq (\rightarrow^* \circ \leftarrow^*)$$

teda ak $M1 \leftarrow M \rightarrow M2$, potom existuje R, že $M1 \rightarrow^* R \leftarrow^* M2$

- $\forall m\ \exists r: x \leftarrow_\beta m \ \wedge\ m \rightarrow_\beta y \ \ => \ \ x \rightarrow_\beta^* r \ \wedge\ r \leftarrow_\beta^* y$

Veta: **Nech $\rightarrow$ je noetherovská/silne normalizujúca/terminujúca relácia.**

- **$\rightarrow$ má Church-Rosser vlastnosť (confluent) je ekvivalentné s**
- **$\rightarrow$ má slabú Church-Rosser vlastnosť (local confluent)**

Dôkaz: CR => SCR, to je jasné...
preto zostáva SCR =>?? CR

Zamyslenie: je noetherovská podmienka podstatná, neplatí veta aj bez nej ?

Slabá Church-Rosser vlastnosť

Veta: **Nech → je noetherovská/silne normalizujúca/terminujúca relácia**

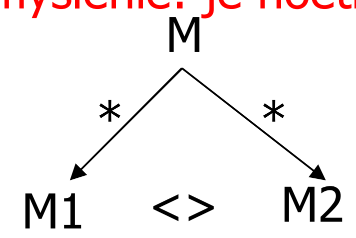
- **→ má Church-Rosser vlastnosť (confluent) je ekvivalentné s**
- **→ má slabú Church-Rosser vlastnosť (local confluent)**

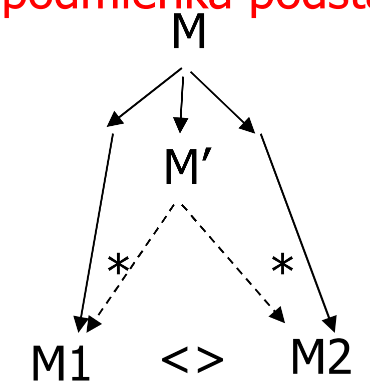
Dôkaz sporom:

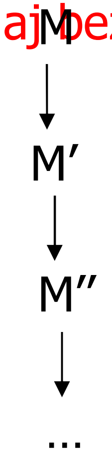
NOE & SCR implikuje CR, ukážeme **spor**: NOE & SCR & ¬CR:

- (¬CR): Nech M má dve normálne formy, M1<>M2, t.j. $M \rightarrow^{*} M_1$ a $M \rightarrow^{*} M_2$.
- M nie je v normálnej forme (ak by bolo, M=M1=M2 a pritom M1<>M2),
- potom existuje M′, že M → M′,
- M′ má tiež dve normálne formy, ak by nie, spor s lokálnou konfluentosťou,
- M″, M‴, M″″, a t.ď (noetherovskosť relácie vyrobí spor).

Zamyslenie: je noetherovská podmienka podstatná, neplatí veta aj bez nej ?







de Bruijn index

čo robilo problémy pri substitúcii, sú mená premenných
idea (pána de Brujin): premenné nahradíme indexami

- λx.(+ x 1)
  - λ.(+ 0 1)

λ.λ.λ.0 (λ.0) (+ 2 1)

- λx.λy.λf.f ((λx.x) (+ x y))
  - λ.λ.λ.0 ((λ.0) (+ 2 1))

index: neform.: cez koľko λ treba vyskákať, aby sme našli λ danej premennej

- α-konverzia neexistuje lebo premenné nemajú mená
- dôsledok: rôzne premenné môžu mať rovnaký index (v <> kontextoch)
- voľné premenné majú index > hĺbku λ-vnorení

- (λx.λy.((z x) (λu.(u x)))) (λx.(w x))
  - (λ.λ.((3 1) (λ.(0 2))) (λ.(4 0))

(λ.λ.3 1 (λ.0 2)) (λ.4 0)

β-redukcia s de Bruijn-indexami

Príklady:

- K=λx.λy.x
  - $\lambda.\lambda.\underline{1}$
- S=λx.λy.λz.((x z) (y z))
  - $\lambda.\lambda.\lambda.((\underline{2}\ \underline{0})\ (\underline{1}\ \underline{0}))$

- λx.(+ x 1) 5
  - $\lambda.(+\ \underline{0}\ 1)\ 5 = (+\ 5\ 1)$

hypotéza, ako by to mohlo fungovať
β-redukcia $(\lambda.M)\ N = M[\underline{0}{:}N]$???
ale nefunguje...

- K a b =(λx.λy.x) a b
  - $(\lambda.\lambda.\underline{1}\ a)\ b = \lambda.a\ b = a$

skúsme intuitívne

- (λx.λy.((z x) (λu.(u x)))) (λx.(w x))
  - $(\lambda.\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2}))))\ (\lambda.(\underline{4}\ \underline{0}))$
  - $(\lambda.\lambda.((\underline{3}\ \underline{\square})\ (\lambda.(\underline{0}\ \underline{\square}))))\ (\lambda.(\underline{4}\ \underline{0}))$
  - $(\lambda.(\underline{2}\ (\lambda.\underline{5}\ \underline{0}))\ (\lambda.(\underline{0}\ (\lambda.\underline{6}\ \underline{0}))))$

označíme si miesta, kam sa substituuje
nahrať, ale pozor na voľné premenné

$(\lambda y.\underline{z}\ (\lambda x.\underline{w}\ \underline{x})\ (\lambda u.\underline{u}\ (\lambda x.\underline{w'}\ \underline{x})))$

β s de Bruijn.indexami

Substitúcia $[t_0, t_1, \ldots, t_n] = [\underline{0}:t_0][\underline{1}: t_1]\ldots[\underline{n}: t_n]$

- $\underline{k}[t_0, t_1, \ldots, t_n] = t_k$, $k <= n$
- $(M\ N)\ [t_0, t_1, \ldots, t_n] = (M[t_0, t_1, \ldots, t_n]\ N[t_0, t_1, \ldots, t_n])$
- $(\lambda M)\ [t_0, t_1, \ldots, t_n] = (\lambda\ M[\underline{0}, t_0{}^\wedge 1, t_1{}^\wedge 1, \ldots , t_n{}^\wedge 1])$

t^1 – pripočítaj 1 k voľným premenným

β: $(\lambda M)\ N = M[N,\underline{0},\underline{1},\underline{2},\underline{3}, \ldots]$

- $(\lambda.\lambda.\underline{1}\ a)\ b=((\lambda.\underline{1})\ [a,\underline{0},\underline{1},\underline{2},\ldots])b= (\lambda.(\underline{1}\ [\underline{0}, a, \underline{1}, \underline{2},\ldots]))\ b=$
  $\lambda.a\ b=a$ - a, b sú konštanty neobs. premenné

Príklad z predošlého slajdu:

- $(\lambda.\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2}))))\ (\lambda.(\underline{4}\ \underline{0})) =$
  - $\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2})))\ [(\lambda.(\underline{4}\ \underline{0})),\underline{0},\underline{1},\underline{2},\ldots] =$
  - $\lambda.(((\underline{3}\ \underline{1})[\underline{0},(\lambda.(\underline{5}\ \underline{0})),\underline{1},\underline{2},\ldots])\quad ((\lambda.(\underline{0}\ \underline{2}))\ [\underline{0},(\lambda.(\underline{5}\ \underline{0})),\underline{1},\underline{2},\ldots])) =$
  - $\lambda.((\underline{2}\ (\lambda.(\underline{5}\ \underline{0})))\quad (\lambda.(\underline{0}\ \underline{2}))\ [\underline{0},(\lambda.(\underline{5}\ \underline{0})),\underline{1},\underline{2},\ldots])) =$
  - $\lambda.((\underline{2}\ (\lambda.(\underline{5}\ \underline{0})))\quad (\lambda.(\underline{0}\ \underline{2})[\underline{0},\underline{1},(\lambda.(\underline{6}\ \underline{0})),\underline{2},\underline{3},\underline{4},\ldots])))$
  - $\lambda.((\underline{2}\ (\lambda.(\underline{5}\ \underline{0})))\quad (\lambda.(\underline{0}\ (\lambda.(\underline{6}\ \underline{0})))))$

$= (\lambda y.(\underline{z}\ (\lambda x.(\underline{w}\ \underline{x})))\ (\lambda u.(\underline{u}\ (\lambda x.(\underline{w'}\ \underline{x})))))$

de Bruijn index

čo robilo problémy pri substitúcii, sú mená premenných
idea (pána de Brujin): premenné nahradíme indexami

$\lambda.\lambda.\lambda.\underline{0}\ (\lambda.\underline{0})\ (+\ \underline{2}\ \underline{1})$

- λx.(+ x 1)
  - $\lambda.(+\ \underline{0}\ 1)$
- λx.λy.λf.f ((λx.x) (+ x y))
  - $\lambda.\lambda.\lambda.\underline{0}\ ((\lambda.\underline{0})\ (+\ \underline{2}\ \underline{1}))$

index: neform.: cez koľko λ treba vyskákať, aby sme našli λ danej premennej
- α-konverzia neexistuje lebo premenné nemajú mená
- dôsledok: rôzne premenné môžu mať rovnaký index (v <> kontextoch)
- voľné premenné majú index > hľbku λ-vnorení

$(\lambda.\lambda.\underline{3}\ \underline{1}\ (\lambda.\underline{0}\ \underline{2}))\ (\lambda.\underline{4}\ \underline{0})$

β-redukcia s de Bruijn-indexami

Príklady:

- K=λx.λy.x
  - $\lambda.\lambda.\underline{1}$
- S=λx.λy.λz.((x z) (y z))
  - $\lambda.\lambda.\lambda.((\underline{2}\ \underline{0})\ (\underline{1}\ \underline{0}))$

- λx.(+ x 1) 5 — hypotéza, ako by to mohlo fungovať
  - $\lambda.(+\ \underline{0}\ 1)\ 5 = (+\ 5\ 1)$ — β-redukcia (λ.M) N = M[$\underline{0}$:N] ???
    ale nefunguje...

- K a b =(λx.λy.x) a b
  - $(\lambda.\lambda.\underline{1}\ a)\ b = \lambda.a\ b = a$ — skúsme intuitívne

- (λx.λy.((z x) (λu.(u x)))) (λx.(w x))
  - $(\lambda.\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2}))))\ (\lambda.(\underline{4}\ \underline{0}))$
  - $(\lambda.\lambda.((\underline{3}\ \underline{\square})\ (\lambda.(\underline{0}\ \underline{\square}))))\ (\lambda.(\underline{4}\ \underline{0}))$ — označíme si miesta, kam sa substituuje
  - $(\lambda.(\underline{2}\ (\lambda.\underline{5}\ \underline{0}))\ (\lambda.(\underline{0}\ (\lambda.\underline{6}\ \underline{0}))))$ — nahrať, ale pozor na voľné premenné

$(\lambda y.\underline{z}\ (\lambda x.\underline{w}\ \underline{x})\ (\lambda u.\underline{u}\ (\lambda x.\underline{w'}\ \underline{x})))$

SKK

- $K=\lambda x.\lambda y.x$
  - $\lambda.\lambda.\underline{1}$
- $S=\lambda x.\lambda y.\lambda z.x\ z\ (y\ z)$
  - $\lambda.\lambda.\lambda.\underline{2}\ \underline{0}\ (\underline{1}\ \underline{0})$

Ďalší testovací príklad

- $S\ K\ K = (\ (\ \lambda.\lambda.\lambda.\underline{2}\ \underline{0}\ (\underline{1}\ \underline{0})\ )\ \lambda.\lambda.\underline{1}\ )\ \lambda.\lambda.\underline{1}\ =$
  - $(\ \lambda.\lambda.\underline{2}\ \underline{0}\ (\underline{1}\ \underline{0})\ [\lambda.\lambda.\underline{1},\ \underline{0},\underline{1},\underline{2},\ldots]\ )\ \lambda.\lambda.\underline{1}\ =$
  - $(\ \lambda.\lambda.(\underline{2}\ \underline{0}\ (\underline{1}\ \underline{0}))\ [\underline{0},\underline{1},\lambda.\lambda.\underline{1},\ \underline{0},\underline{1},\underline{2},\ldots]\ )\ \lambda.\lambda.\underline{1}\ =$
  - $(\lambda.\lambda.((\lambda.\lambda.\underline{1})\ \underline{0}\ (\underline{1}\ \underline{0})\ ))\ \lambda.\lambda.\underline{1}\ =$
  - $(\lambda.((\lambda.\lambda.\underline{1})\ \underline{0}\ (\underline{1}\ \underline{0})))\ [\lambda.\lambda.\underline{1},\underline{0},\underline{1},\underline{2},\ldots]\ =$
  - $\lambda.(((\lambda.\lambda.\underline{1})\ \underline{0}\ (\underline{1}\ \underline{0}))\ [\underline{0},\lambda.\lambda.\underline{1},\underline{0},\underline{1},\underline{2},\ldots])\ =$
  - $\lambda.((\lambda.\lambda.\underline{1})\ \underline{0}\ (\lambda.\lambda.\underline{1}\ \underline{0}))\ =$
  - $\lambda.\underline{0} = I$

Cvičenie

Prepíšte do de Bruijn notácie

- λx.λy.y (λz.z x) x
- λx.(λx.x x) (λy.y (λz.x))
- (λx. + x ((λy.y) (- x (λz.3)(λy.y y)))

Definujte funkciu na prevod do de Bruijn notácie.

Implementujte β-redukciu s pomocnými funkciami