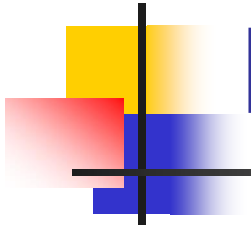


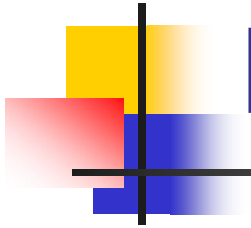
Lambda calculus 2

Štruktúra prednášok:

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

domáca úloha: interpreter λ-kalkulu, ...

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



Domáca úloha (nepovinná)

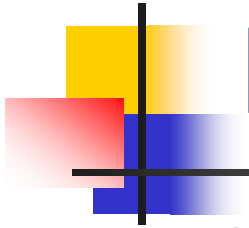
Pri práci s vašim interpretrom vám bude chýbať:

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

  fromString "\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

- výstup λ termu – funkcia toString :: LExp -> String, ktorá vám vytvorí textovú (čitateľnú) reprezentáciu pre λ term.

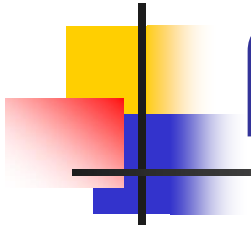


Fold na termoch

```
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 == (APL 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)
```



β a η-redukcia

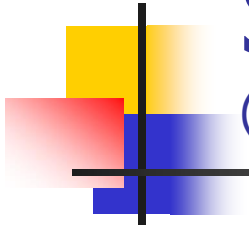
- β-redukcia: $(\lambda\, x.B)\ E \rightarrow_{\beta} B[x:E]$
- η-redukcia: $\lambda\, x.(B\ x) \rightarrow_{\eta} B$ ak $x \notin Free(B)$

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

- $\ddot{Y}_{\beta\eta}$ je uzáver $\ddot{Y}_{\beta} \cup \ddot{Y}_{\eta}$ vzhľadom na podtermy, čo znamená:
  - ak $M\ \ddot{Y}_{\beta}\ N$ alebo $M\ \ddot{Y}_{\eta}\ N$, potom $M\ \ddot{Y}_{\beta\eta}\ N$,
  - ak $M\ \ddot{Y}_{\beta\eta}\ N$, potom $(P\ M)\ \ddot{Y}_{\beta\eta}\ (P\ N)$ aj $(M\ Q)\ \ddot{Y}_{\beta\eta}\ (N\ Q)$,
  - ak $M\ \ddot{Y}_{\beta\eta}\ N$, potom $\lambda x.M\ \ddot{Y}_{\beta\eta}\ \lambda x.N$.

Vlastnosti β-redukcie

- známe λ-termy
  - $\omega = \lambda x.xx$
  - $\Omega = \omega\omega$
  - $\omega_3 = \lambda x.xxx$
- existuje nekonečná sekvencia
  - $\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \ldots$
- 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
  - $KI\Omega \to_\beta KI\Omega \to_\beta KI\Omega \to_\beta \ldots$
- existuje term s dvomi rôznymi normálnymi formami ?



Stratégia redukcie

(na výber 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 Ÿ $_{\beta\eta}$ μ(M)
- μ výpočet je postupnosť M, μ(M), …, μ^i(M), … a môže byť (ne)konečná

Najznámejšie stratégie

- leftmost-innermost – najľavejší redex neobsahuje iný redex

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

- leftmost-outermost – najľavejší redex neobsiahnutý v inom redexe

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

- leftmost – 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 – 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
nf t   = if t == t' then t else nf t' where t'=oneStepβ t
-----------------------------------------------
oneStepβ (APL (LAMBDA x m) n) = substitute m x n

oneStepβ (APL m n)    = if m == m' then
                              (APL m (oneStepβ n))
                        else
                              (APL m' n)
                        where m' = oneStepβ m

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

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

Stratégie β-redukcie

- kdekoľvek, až kým nie je v n.f.
- **leftmost-innermost** (nie je normalizujúca stratégia)
  - argumenty funkcie sú zredukované skôr ako telo funkcie
  - $KI\Omega \rightarrow_\beta KI\Omega \rightarrow_\beta KI\Omega \rightarrow_\beta \ldots$
  - $(\lambda x.x\ x)\ \underline{(\lambda x.x\ y)} \rightarrow_\beta \underline{(\lambda x.x\ x)y} \rightarrow_\beta yy$
- **leftmost-outermost** (je normalizujúca stratégia)
  - ak je možné dosadiť argumenty do tela funkcie, urobí sa tak ešte pred ich vyhodnotením, ale tým aj kopíruje redexy ☹
  - $KI\Omega \rightarrow_\beta I$
  - $(\lambda x.x\ x)\ \underline{(\lambda x.x\ y)} \rightarrow_\beta\ \underline{(\lambda x.x\ y)}\ \underline{(\lambda x.x\ y)} \rightarrow_\beta y\ \underline{(\lambda x.x\ y)} \rightarrow_\beta y\ y$

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

Domáca úloha
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 = m n f = ((n m) f) = $(m^n f)$

•definujte n-1 (na rozmýšľanie)

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

$n\,f\;x = f^n\;x$

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)

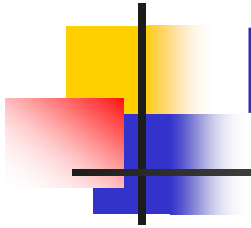
- succ := λn.λf.λx.f (n f x) = λn.λf.λx.(f ((n f) x))
- plus := λm.λn.λf.λx. m f (n f x) = λm.λn.λf.λx. ((m f) ((n f) x))
  -- idea: $f^m(f^n\;x) = f^{m+n}\;x$

Zadáme tieto dve konštrukcie:

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

potom vypočítame:

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



Logika a predikáty

TRUE	:= λx.λy. x := λxy.x	(vráti 1.argument)
FALSE	:= λx.λy. y := λxy.y	(vráti 2.argument)
AND	:= λx.λy. x y FALSE := λxy.x y FALSE	
OR	:= λx.λy. x TRUE y := λxy.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 Ÿ $_{\beta}$ TRUE FALSE FALSE
≡ (λ x y. x) FALSE FALSE Ÿ $_{\beta}$ FALSE

definujte XOR

Kartézsky súčin typov (pár)

PAIR := λx.λy.(λc. c x y) := λxyc. c x y
LEFT := λx.x TRUE
RIGHT := λx.x FALSE

TRUE := λx.λy.x := λxy.x
FALSE := λx.λy.y := λxy.y

LEFT (PAIR A B) ≡
LEFT ((λxyc. c x y) A B) Ÿ $_\beta$
LEFT (λc. c A B) Ÿ $_\beta$
(λx.x TRUE) (λc. c A B) Ÿ $_\beta$
(λc. c A B) (λxy.x) Ÿ $_\beta$
((λxy.x) A B) Ÿ $_\beta$ 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:

λ(x,y).M vs. (λx.λy.M)

λ(x,y).M -> λp. (λx.λy.M) (LEFT p) (RIGHT p)

PAIR := λx.λy.(λc. c x y) := λxyc. c x y

Súčet typov (disjunkcia)

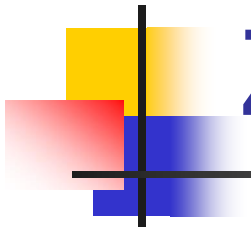
A+B reprezentujeme ako dvojicu Bool x (A|B)

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

$1^{st-1}\ 1^{st}\ A \equiv$

(λz.RIGHT z) (λx.PAIR TRUE x) A Ÿ $_{\beta}$

RIGHT (PAIR TRUE A) Ÿ $_{\beta}$ A



Zoznamy

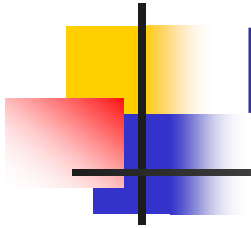
- List t = Nil | Cons t (List t)

Domáca úloha (vašim interpretrom):
Ånull? (cons a Nil) Ÿ $_{\beta}^{*}$
Åhead (cons a Nil) Ÿ $_{\beta}^{*}$
Åtail (cons a Nil) Ÿ $_{\beta}^{*}$
Åhead (tail (cons a (cons b Nil)))

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.:

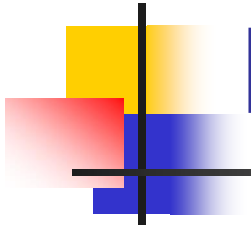
isNil Nil = (λp.p (λx.λy.λz.x)) (λz.z TRUE FALSE FALSE) Ÿ $_{\beta}$
((λz.z TRUE FALSE FALSE) (λx.λy.λz.x)) Ÿ $_{\beta}$
((λx.λy.λz.x) TRUE FALSE FALSE) Ÿ $_{\beta}$
TRUE



Binárne stromy

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

Empty	= λg.g TRUE (λx.x) (λx.x) (λx.x)
Node	= λx.λy.λz.λg.g FALSE x y z
isEmpty	= λt.t (λu.λx.λy.λz.u)
root	= λt.t (λu.λx.λy.λz.x)
left	= λt.t (λu.λx.λy.λz.y)
right	= λt.t (λu.λx.λy.λz.z)



Binárne stromy

Odvoďme, napr.:

root (Node a Empty Empty) Ÿ $_{\beta}$

(λt.t (λu.λx.λy.λz.x)) (Node a Empty Empty) Ÿ $_{\beta}$

((Node a Empty Empty) (λu.λx.λy.λz.x)) Ÿ $_{\beta}$

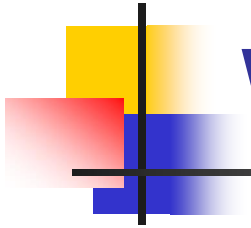
(((λx.λy.λz.λg.g FALSE x y z) a Empty Empty) (λu.λx.λy.λz.x)) Ÿ $_{\beta}$

((λg.g FALSE a Empty Empty) (λu.λx.λy.λz.x)) Ÿ $_{\beta}$

((λu.λx.λy.λz.x) FALSE a Empty Empty) (λu.λx.λy.λz.x)) Ÿ $_{\beta}$

((λu.λx.λy.λz.x) FALSE a Empty Empty)) Ÿ $_{\beta}$

a



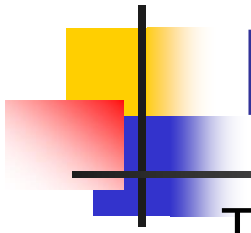
where

M where v = N -> (λv.M) N

M where $v_1 = N_1$ -> (λ(v_1, v_2, …, v_n).M) (N_1 , …, N_n)
$v_2 = N_2$ …
$v_n = N_n$

zložený where

n*(x+n) where -> (λn. (λx.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.

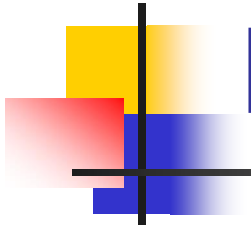
Príklad:
FAC := λn.(if (= n 0) 1 (* n (FAC (- n 1))))

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)
hľadaná funkcia FAC je *pevný bod* funkcie H

- Aký je typ Y ??



Pevný bod

Potrebujeme trochu teórie:

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

Dar nebies (operátor pevného bodu):

Y = λf.(λx. f(x x)) (λx. f(x x))

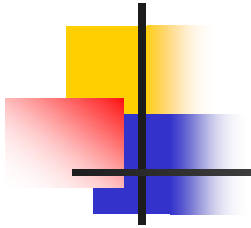
potom

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

Skúsme to (aspoň) overiť:

Y F = (λf.(λx. f(x x)) (λx. f(x x))) F = (λx. F(x x)) (λx. F(x x)) Ÿ $_{\beta}$

- F(x x)[x:λx. F(x x)] Ÿ $_{\beta}$
- F(λx.F(x x) λx.F(x x)) =
- F (Y F) preto (Y F) je naozaj pevný bod F



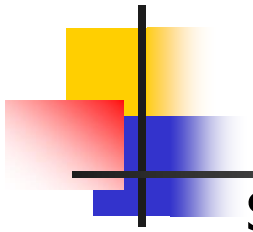
Operátor Y

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
        = 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
```

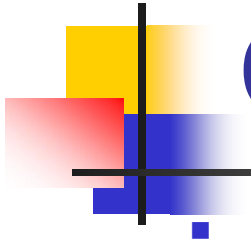


1+2+3+…+n

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



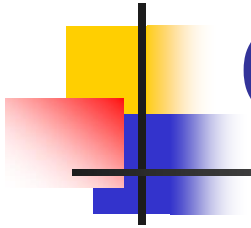
Cvičenie

- (na zamyslenie) nájdite príklady funkcií
  - s nekonečným počtom pevných bodov
  - s práve jedným pevným bodom

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



Cvičenie

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

Vyhodnoťte LExp [LExp [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

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

$X_1 = F_1\, X_1\, X_2 \ldots X_n$

$X_2 = F_2\, X_1\, X_2 \ldots X_n$

… …

$X_n = F_n\, X_1\, X_2 \ldots X_n.$

vektorovo:

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

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

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

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

Primitívne rekurzívna funkcia je:

- nulová funkcia $N^nŸ N$,
- succ: $NŸ N$,
- projekcia p_i: $N^nŸ N$, $p_i\, x_1\, x_2 \ldots x_n = x_i$
- 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 $g : N^nŸ N$, $h : N^{n+2}Ÿ N$, potom $f : N^{n+1}Ÿ N$

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

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

Čiastočne vyčíslitelná (nemusí byť totálna):

- μ-rekurzia $r : N^{n+1}Ÿ N$, potom $f : N^{n+1}Ÿ N$

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

λ-vypočítateľná funkcia

Parciálna funkcia $f : N^n \ddot{Y} N$ je λ-vypočítateľná, ak existuje λ-term F taký, že

F $\underline{x_1}$ $\underline{x_2}$ … $\underline{x_n}$ sa zredukuje na $\underline{f\ x_1\ x_2\ \ldots\ x_n}$, ak n-tica $x_1\ x_2\ \ldots\ x_n$ patrí do def.oboru f

F $\underline{x_1}$ $\underline{x_2}$ … $\underline{x_n}$ nemá normálnu, ak n-tica $x_1\ x_2\ \ldots\ x_n$ nepatrí do def.oboru f

Veta: Každá parciálne vyčíslitelná funkcia je λ-vypočítateľná.

Dôkaz:

- nulová fcia, succ, projekcie p_i, kompozícia priamočiaro
- primitívna rekurzia $g : N^n \ddot{Y} N$, $h : N^{n+2} \ddot{Y} N$, potom $f : N^{n+1} \ddot{Y} N$

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

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

  $F = \mathbf{Y}\ (\lambda f.\lambda y.\lambda x_1.\lambda x_2\ \ldots\ \lambda x_n\ .$ (if (isZero y) $G(x_1\ x_2\ \ldots\ x_n)$ then

  else $H(f((\text{pred } y)\ x_1\ x_2\ \ldots\ x_n)\ (\text{pred } y)\ x_1\ x_2\ \ldots\ x_n)))$
- μ-rekurzia $r : N^{n+1} \ddot{Y} N$

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

Veta: Každá λ-vypočítateľná je parcialne vyčíslitelná funkcia.

Weak head normal form

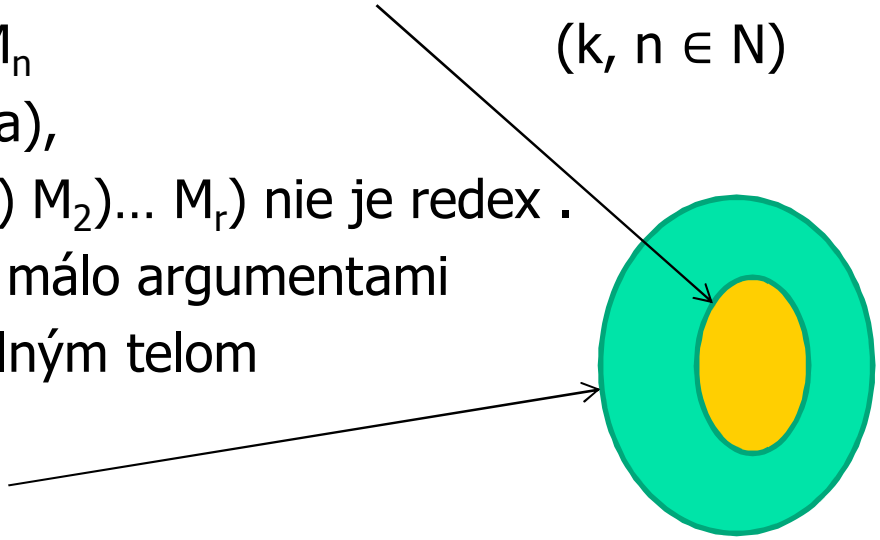
(slabo hlavová normálna forma)

Head normal form (h.n.f)

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

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

Konštanta, premenná alebo λ-abstrakcia s málo argumentami.

λx.((λy.y) z) nie je h.n.f. (až po red. ((λy.y) z) Ÿ $_\beta$ 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)

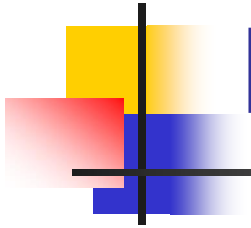
  $\underline{(\lambda x.\ \lambda y.(x\ y))}\ (\lambda z.z) \to_\beta \lambda y.(\underline{(\lambda z.z)\ y}) \to_\beta \lambda y.y$ n.f.

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

  normalizuje úplne...

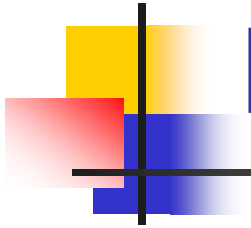
- eager - argumenty najprv (call by value/data driven/strict)

  nenormalizuje...



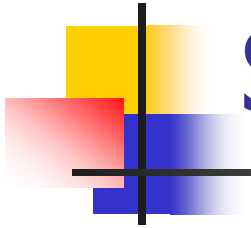
Lazy

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



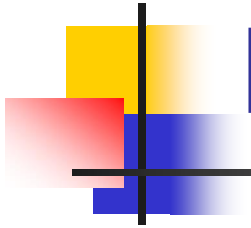
Full lazy

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



Strict

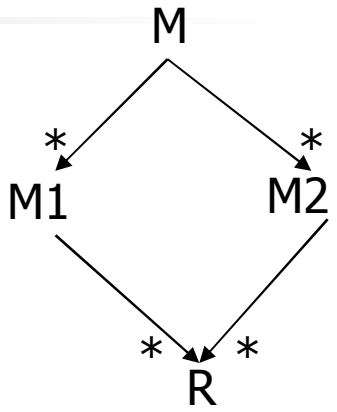
- (λx. λy.(* (+ x x) y) ((λx.x)(* 3 4))) (λx.(+2 x) 6) Ÿ $_{\beta}$
- (λx. λy.(* (+ x x) y) ((λx.x)(* 3 4))) (+2 6) Ÿ $_{\beta}$
- (λx. λy.(* (+ x x) y) ((λx.x)(* 3 4))) 8 Ÿ $_{\beta}$
- (λx. λy.(* (+ x x) y) (* 3 4)) 8 Ÿ $_{\beta}$
- (λx. λy.(* (+ x x) y) 12) 8 Ÿ $_{\beta}$
- (λy.(* (+ 12 12) y)) 8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



Eager

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

Church-Rosser vlastnosť
(konzistentnosť λ-kaklulu)



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

$M \twoheadrightarrow_\beta^* M_1$ a $M \twoheadrightarrow_\beta^* M_2$

existuje R, že

$M_1 \twoheadrightarrow_\beta^* R$ a $M_2 \twoheadrightarrow_\beta^* R$

Inak:

$$(\twoheadrightarrow_\beta^* \circ \twoheadrightarrow_\beta^*) \subseteq (\twoheadrightarrow_\beta^* \circ \twoheadrightarrow_\beta^*)$$

teda ak $M1 \twoheadrightarrow_\beta^* M \twoheadrightarrow_\beta^* M2$, potom existuje R, že $M1 \twoheadrightarrow_\beta^* R \twoheadrightarrow_\beta^* M2$

Veta: β-redukcia spĺňa Church-Rosserovu vlastnosť

Dôkazy sú technicky náročné:

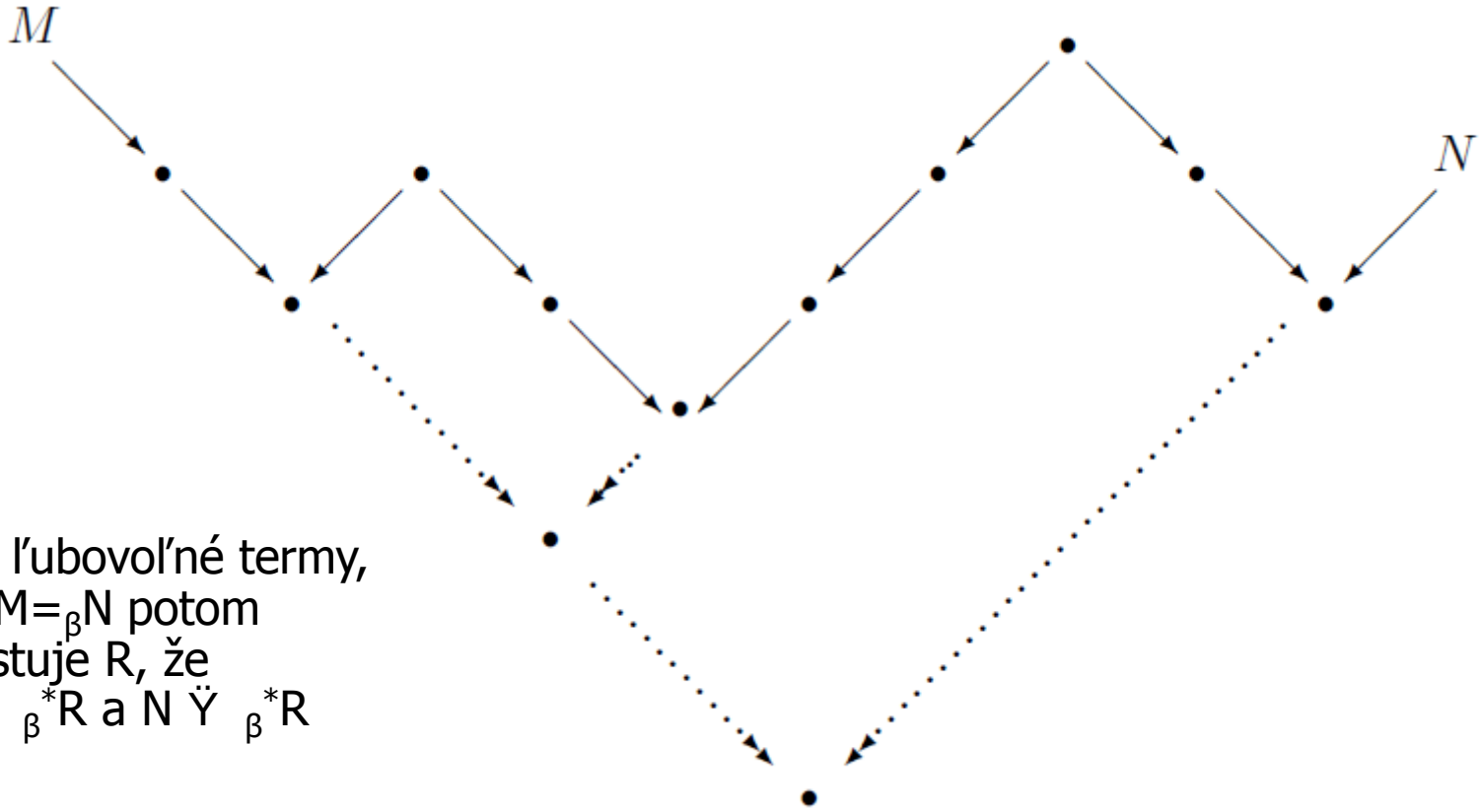
- 1936 Church, Rosser: Some properties of conversion
- 1981 Barendregt
- 1981 Löf, Tait

Dôsledok:

ak term má normálnu formu vzhľadom na $\twoheadrightarrow_\beta$, potom je jednoznačne určená

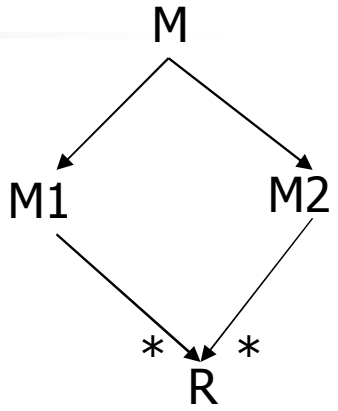
β ekvivalenica

- $=_\beta \equiv (\ddot{Y}_\beta \cup \hat{Y}_\beta)^*$ Príklad: $KI\Omega =_\beta II$



pre l'ubovoľné termy,
ak $M=_\beta N$ potom
existuje R, že
$M\ddot{Y}_\beta{}^*R$ a $N\ddot{Y}_\beta{}^*R$

Slabá Church-Rosser vlastnosť



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

M Ÿ M_1 a M Ÿ M_2

existuje R, že

M_1 Ÿ *R a M_2 Ÿ *R

Inak:

$(\hat{Y} \circ \ddot{Y}) \subseteq (\ddot{Y}^* \circ \hat{Y}^*)$

teda ak M1Ŷ M Ÿ M2, potom existuje R, že M1 Ÿ *RŶ * M2

Veta: **Nech Ÿ je noetherovská/silne normalizujúca/terminujúca relácia.**

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

Dôkaz sporom (SCR implikuje CR, spor: SCR and ¬CR):

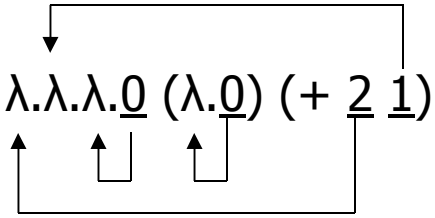
- Nech M má dve normálne formy, M1<>M2, t.j. M Ÿ * M_1 a M Ÿ * M_2.
- M nie je v normálnej forme (ak by bolo, M=M1=M2 a pritom M1<>M2),
- potom existuje M', že M Ÿ M',
- M' má tiež dve normálne formy, ak by nie, spor s lokálnou konfluentosťou,
- M'', M''', M'''', a t.ď (noetherovskosť relácie vyrobí 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

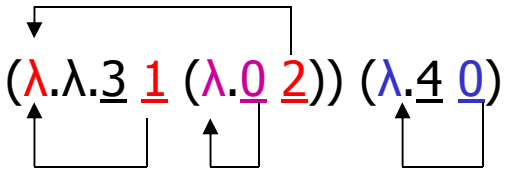
- λx.(+ x 1)
  - λ.(+ 0 1)



- 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 > hľbku λ-vnorení

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



β-redukcia s de Bruijn-indexami

Príklady:

- K=λx.λy.x
  - λ.λ.1
- S=λx.λy.λz.((x z) (y z))
  - λ.λ.λ.((2 0) (1 0))

- λx.(+ x 1) 5 — hypotéza, ako by to mohlo fungovať
  - λ.(+ 0 1) 5 = (+ 5 1) — β-redukcia (λ.M) N = M[0:N] ??? ale nefunguje...
- K a b =(λx.λy.x) a b
  - (λ.λ.1 a) b = λ.a b = a — skúsme intuitívne

- (λx.λy.((z x) (λu.(u x)))) (λx.(w x))
  - (λ.λ.((3 1) (λ.(0 2)))) (λ.(4 0))
  - (λ.λ.((3 ĭ) (λ.(0 ĭ)))) (λ.(4 0)) — označíme si miesta, kam sa substituuje nahrať, ale pozor na voľné premenné
  - (λ.(2 (λ.5 0)) (λ.(0 (λ.6 0))))

(λy.z (λx.w x) (λu.u (λx.w′ x)))

β s de Bruijn.indexami

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

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

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:

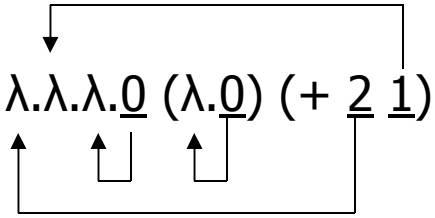
- $(\lambda.\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2})))\ (\lambda.(\underline{4}\ \underline{0})) =$
  - $\lambda.((\underline{3}\ \underline{1})\ (\lambda.(\underline{0}\ \underline{2}))\ [(\lambda.(\underline{4}\ \underline{0})),\underline{0},\underline{1},\underline{2},\ldots]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

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

de Bruijn index

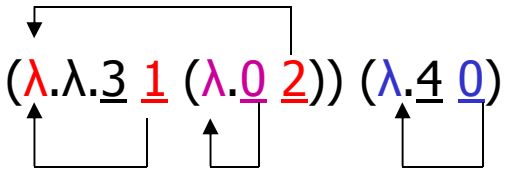
čo robilo problémy pri substitúcii, sú mená premenných
idea (pána de Brujin): premenné nahradíme indexami

- λx.(+ x 1)
  - λ.(+ 0 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 > hľbku λ-vnorení



β-redukcia s de Bruijn-indexami

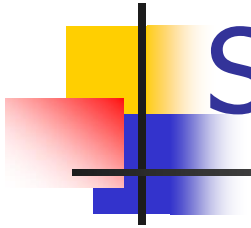
Príklady:

- K=λx.λy.x
  - λ.λ.1
- S=λx.λy.λz.((x z) (y z))
  - λ.λ.λ.((2 0) (1 0))

- λx.(+ x 1) 5 — hypotéza, ako by to mohlo fungovať
  - λ.(+ 0 1) 5 = (+ 5 1) — β-redukcia (λ.M) N = M[0:N] ??? ale nefunguje...
- K a b =(λx.λy.x) a b
  - (λ.λ.1 a) b = λ.a b = a — skúsme intuitívne

- (λx.λy.((z x) (λu.(u x)))) (λx.(w x))
  - (λ.λ.((3 1) (λ.(0 2)))) (λ.(4 0))
  - (λ.λ.((3 ĭ) (λ.(0 ĭ)))) (λ.(4 0)) — označíme si miesta, kam sa substituuje nahrať, ale pozor na voľné premenné
  - (λ.(2 (λ.5 0)) (λ.(0 (λ.6 0))))

(λy.z (λx.w x) (λu.u (λx.w′ x)))

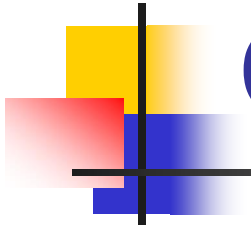


SKK

- K=λx.λy.x
  - λ.λ.1
- S=λx.λy.λz.x z (y z)
  - λ.λ.λ.2 0 (1 0)

Ďalší testovací príklad

- S K K = ((λ.λ.λ.2 0 (1 0)) λ.λ.1) λ.λ.1 =
  - (λ.λ.2 0 (1 0) [λ.λ.1, 0,1,2,…]) λ.λ.1 =
  - (λ.λ.(2 0 (1 0)) [0,1,λ.λ.1, 0,1,2,…]) λ.λ.1 =
  - (λ.λ.((λ.λ.1) 0 (1 0))) λ.λ.1 =
  - (λ.((λ.λ.1) 0 (1 0))) [λ.λ.1,0,1,2,…] =
  - λ.(((λ.λ.1) 0 (1 0)) [0,λ.λ.1,0,1,2,…]) =
  - λ.((λ.λ.1) 0 (λ.λ.1 0)) =
  - λ.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