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: $(\lambda\ x.B)\ E \ {->}_{\beta}\ B[x:E]$

- η-redukcia: $\lambda\ x.(B\ x)\ {->}_{\eta}\ B$ ak $x \notin Free(B)$

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

- Ÿ $_{\beta\eta}$ je uzáver Ÿ $_{\beta}$ ∪ Ÿ $_{\eta}$ vzhľadom na podtermy, čo znamená:
  - ak M Ÿ $_{\beta}$ N alebo M Ÿ $_{\eta}$ N, potom M Ÿ $_{\beta\eta}$ N,
  - ak M Ÿ $_{\beta\eta}$ N, potom (P M) Ÿ $_{\beta\eta}$ (P N) aj (M Q) Ÿ $_{\beta\eta}$ (N Q),
  - ak M Ÿ $_{\beta\eta}$ N, potom λx.M Ÿ $_{\beta\eta}$ λx.N.

Vlastnosti β-redukcie

- známe λ-termy
  - $\omega = \lambda x.xx$
  - $\Omega = \omega\omega$
  - $\omega_3 = \lambda x.xxx$
- existuje nekonečná sekvencia
  - Ω Ÿ $_{\beta}$ Ω Ÿ $_{\beta}$ Ω Ÿ $_{\beta}$ …
- existuje neobmedzene puchnúca sekvencia
  - $\omega_3\ \omega_3$ Ÿ $_{\beta}$ $\omega_3\ \omega_3\ \omega_3$ Ÿ $_{\beta}$ $\omega_3\ \omega_3\ \omega_3\ \omega_3$
- nejednoznačný výsledok – existuje term s konečným a nekonečným odvodením
  - $KI\Omega$ Ÿ $_{\beta}$ I ale aj
  - $KI\Omega$ Ÿ $_{\beta}$ $KI\Omega$ Ÿ $_{\beta}$ $KI\Omega$ Ÿ $_{\beta}$ …
- 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

  (λx.(λy.y) x) (λz.z) Ÿ $_\beta$ (λx.x) (λz.z) Ÿ $_\beta$ (λz.z)

- leftmost-outermost – najľavejší redex neobsiahnutý v inom redexe

  (λx.(λy.y) x) (λz.z) Ÿ $_\beta$ ((λy.y) (λz.z)) Ÿ $_\beta$ (λz.z)

- leftmost – vyhodnotí funkciu skôr ako argumenty

  (λx.x)(λy.(λz.z) y) Ÿ $_\beta$ (λy.(λz.z) y) Ÿ $_\beta$ (λy.y)

- rightmost – vyhodnotí argumenty skôr ako funkciu

  (λx.x)(λy.(λz.z) y) Ÿ $_\beta$ (λx.x)(λy.y) Ÿ $_\beta$ (λy.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
  - $K I \Omega \to_\beta K I \Omega \to_\beta K I \Omega \to_\beta \dots$
  - $(\lambda x.x\ x)\ \underline{(\lambda x.x\ y)} \to_\beta \underline{(\lambda x.x\ x) y} \to_\beta yy$
- **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 ☹
  - $K I \Omega \to_\beta I$
  - $(\lambda x.x\ x)\ \underline{(\lambda x.x\ y)} \to_\beta \underline{(\lambda x.x\ y)}\ \underline{(\lambda x.x\ y)} \to_\beta y\ \underline{(\lambda x.x\ y)} \to_\beta y\ y$

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

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

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

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 ≡

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

RIGHT (PAIR TRUE A) Ÿ $_{\beta}$ A

Zoznamy

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)))

- 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) 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	$= \lambda g.g\ TRUE\ (\lambda x.x)\ (\lambda x.x)\ (\lambda x.x)$
Node	$= \lambda x.\lambda y.\lambda z.\lambda g.g\ FALSE\ x\ y\ z$
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) 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   -> $(\lambda v.M)\ N$

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

$v_2 = N_2 \ldots$

$v_n = N_n$

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

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

Operátor Y

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.

Weak head normal form

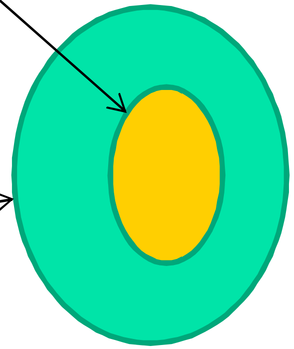
(slabo hlavová normálna forma)

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 neredukovateľným 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)

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

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

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

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

M Ÿ $_{\beta}^{*}M_1$ a Ÿ $_{\beta}^{*}M_2$

existuje R, že

M_1 Ÿ $_{\beta}^{*}R$ a M_2 Ÿ $_{\beta}^{*}R$

Inak:

(Ŷ $_{\beta}^{*}$ ∘ Ÿ $_{\beta}^{*}$) ⊆ (Ÿ $_{\beta}^{*}$ ∘ Ŷ $_{\beta}^{*}$)

teda ak M1 Ŷ $_{\beta}^{*}$M Ÿ $_{\beta}^{*}$M2, potom existuje R, že M1 Ÿ $_{\beta}^{*}$R Ŷ $_{\beta}^{*}$ M2

Veta: β-redukcia spĺňa Church-Rosserovu vlastnosť

Dôkazy sú technicky náročné:

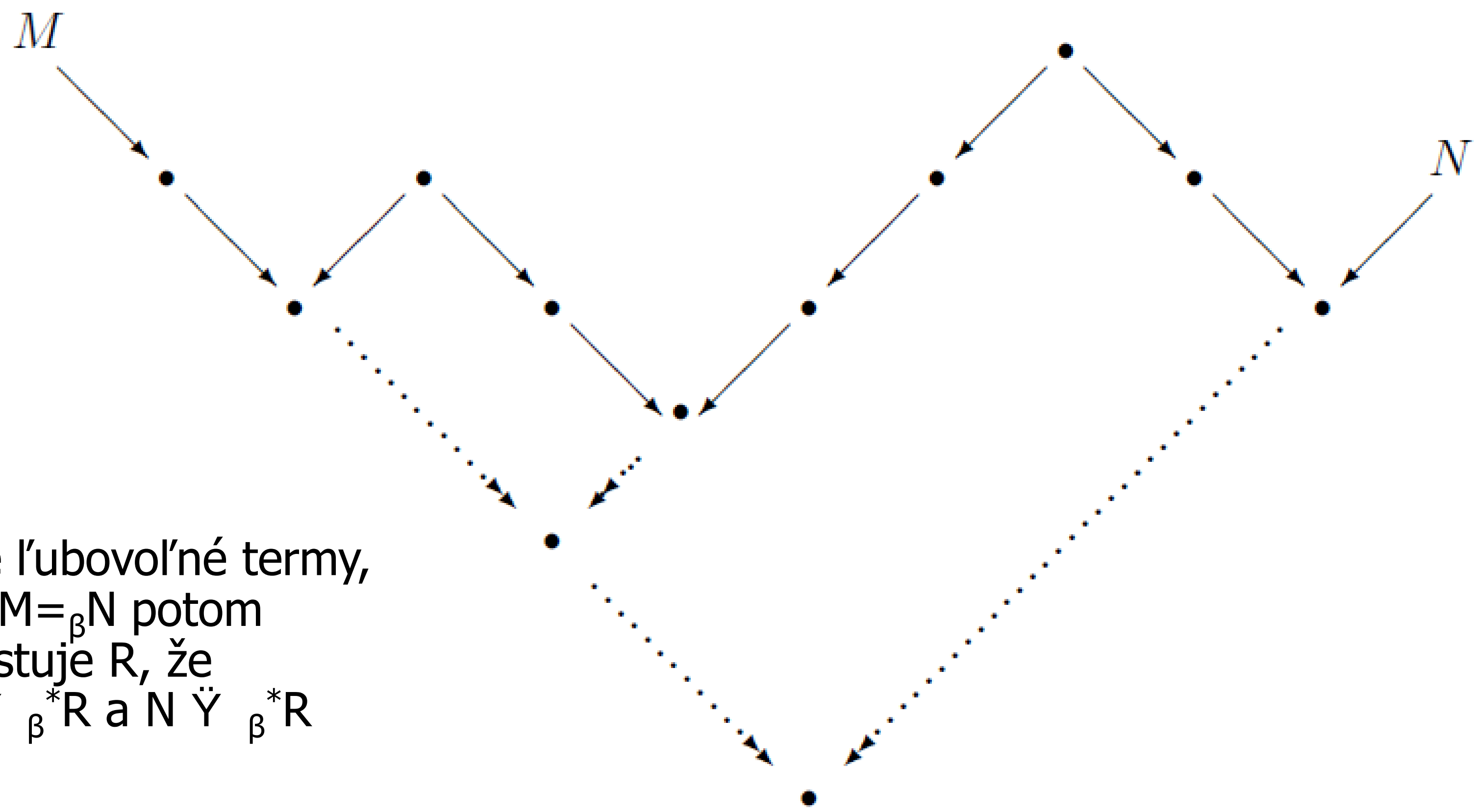
- 1936 Church, Rosser: Some properties of conversion
- 1981 Barendregt
- 1981 Löf, Tait

Dôsledok:

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

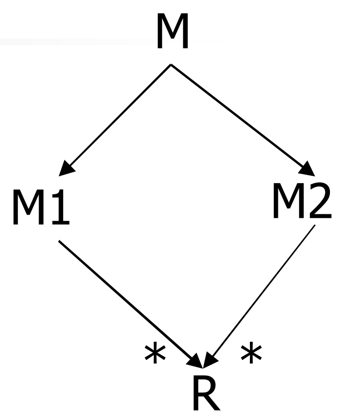
β ekvivalenica

- $=_\beta \equiv$ (Ÿ $_\beta \cup$ Ŷ $_\beta)^*$ Príklad: KIΩ $=_\beta$ II



pre ľubovoľné termy,
ak M$=_\beta$N potom
existuje R, že
MŸ $_\beta{}^*$R a N Ÿ $_\beta{}^*$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:

(Ŷ ∘ Ÿ) ⊆ (Ÿ * ∘ Ŷ *)

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 ?

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)

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)