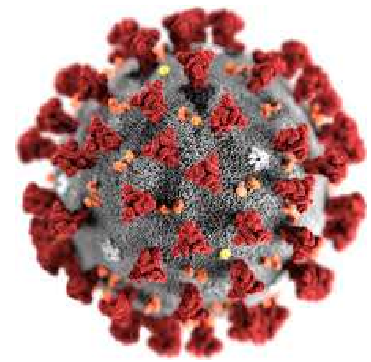
Úvod do λ-kalkulu

Peter Borovanský

I-18

http://dai.fmph.uniba.sk/courses/FPRO/



- skončili sme intro do Haskellu, aspoň tak, aby každý v kurze bol ako-tak komfortný, vedel, kde hľadať radu, ak ju potrebuje...
- meškáme 1 týždeň, ale nevadí... Pýtal som si F-B, ako s kurzom ďalej, a prišli dva, úplne protichodné ☺
- 4.prednáška bola formou samoštúdia z knihy F-Perly

  http://dai.fmph.uniba.sk/courses/FPRO/bird_pearls.pdf
- FPRO je teraz dobré na to, prísť na iné myšlienky, preto ponúkam
  - možnosť samoštúdia z knihy, plus konzultácie, dohodnutá kapitola/týždeň
  - pokračovanie klasicky, prednášky/cviko/DÚ
  - Čaká nás midterm, pred Veľkou nocou...

Lambda calculus

Štruktúra prednášok:

- pár slov z histórie (niečo už bolo 1.týždeň)
- úvod do syntaxe, netypovaný λ-kalkul (gramatika + konvencie)
- sémantika (redukčné pravidlá)
- programovací jazyk nad λ-kalkulom

domáca úloha: interpreter λ-kalkulu, ...

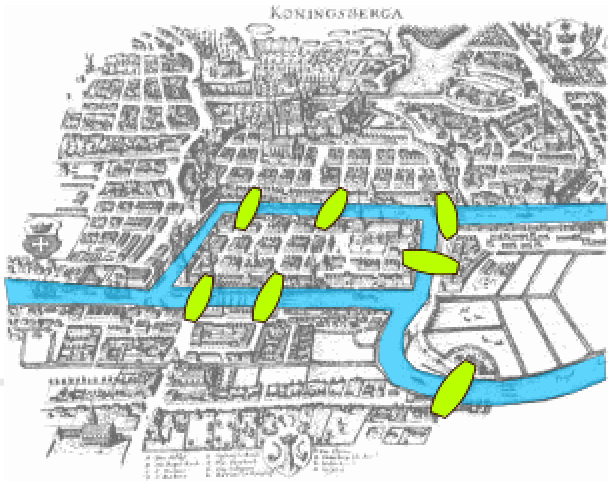
Určite vystačíte s https://en.wikipedia.org/wiki/Lambda_calculus

dnes nebude (ale príde):

- rekurzia (pevný bod)
- vlastnosti teórie
- de Bruijn-ova notácia
- typovaný λ-kalkul

domáca úloha: typovač λ-kalkulu, ...

Trochu z histórie



- 1900, David Hilbert, formalizácia matematiky

  formuloval 23 vtedy neriešiteľných problémov v matematike

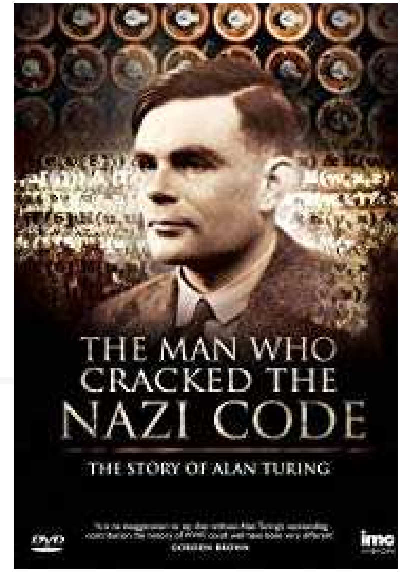
  https://en.wikipedia.org/wiki/Hilbert%27s_problems#Table_of_problems

- 10.problém: algoritmus na riešenie polynomiálnych Diofantických rovníc
  - existencia rozhodovacieho algoritmu, či existuje celočíselné riešenie celočíselnej DR

  Entscheidung der Losbarkeit einer diophantischen Gleichung.
  Rozhodnuteľnosť
  Decidability

  Diofantov epitaf:
  Diofantova mladosť trvala 1/6 jeho života. Fúzy mu narástly o ďalšiu 1/12 jeho života. O nasledujúcu 1/7 života sa Diofantos oženil. Po piatich rokoch sa mu narodil syn. Syn žil presne 1/2 dĺžky života svojho otca. Diofantos zomrel 4 roky po smrti svojho syna.
  Koľko rokov sa dožil ?

Trochu z histórie



- 1936, Alan Turing, On Computable Numbers with an Application to **Entscheidungsproblems**

  formuloval pojem počítania a čísla, ktoré vieme 'mechanicky' vypočítať

  prišiel k záveru, že 'Turingov stroj' nevie vypočítať ľubovoľné reálne číslo lebo Cantorov dôkaz

- 1936, Alonzo Church, A note on the **Entscheidungsproblems**
- 1936, An Unsolvable Problem of Elementary Number Theory

- základ lambda calculus (effective calculability)
  - teoretický základ FP
  - kalkul funkcií: abstrakcia, aplikácia, kompozícia
  - Princeton: A.Church, A.Turing, J. von Neumann, K.Gödel - skúmajú formálne modely výpočtov

Trochu z histórie FP

- éra: WWII, prvý von Neumanovský počítač: Mark I (IBM), balistické tabuľky

- 1958, Haskell B.Curry, logika kombinátorov
  - alternatívny pohľad na funkcie, menej známy a populárny
  - „premenné vôbec nepotrebujeme“

- 1958, LISP, John McCarthy
  - implementácia lambda kalkulu na „von Neumanovskom HW“

- 1960, SECD (**S**tack-**E**nvironment-**C**ontrol-**D**ump) Machine, Landin
  - predchodca p-code, rôznych stack-orientovaných bajt-kódov, virtuálnych mašín.
  - SECD použili pri implementácii
    - Algol 60, PL/1 – predchodcu Pascalu
    - LISP – prvého funkcionálneho jazyka založenom na λ-kalkule

Od Haskellu k λ-kalkulu

- length []   = 0
- length (x:xs)   = 1+length xs

> length [1,2,3,4,5]

let length xs = if (null xs) then 0 else (1+length (tail xs)) **in** length [1,2,3,4,5]

let length xs = (if (null xs) 0 ((+) 1 (length (tail xs))))
in (length ((:) 1 ((:) 2 ((:) 3 ((:) 4 ((:) 5 [])))))))

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

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

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

Syntax

Celý program v jazyku pozostáva z jedného λ-termu.

L je λ-term:

- x je premenná (spočítateľná množina premenných)

$$L ::= x \mid (L\ L) \mid (\lambda x\ L)$$

$$L ::= x \mid L\ L \mid \lambda x.L$$

- (L L) je aplikácia (funkcie)
- (λx L) je λ-abstrakcia definujúca funkciu s argumentom x a telom L

Cvičenie (na zamyslenie): syntax jazyka je veľmi jednoduchá, neporovnateľná napr. s Javou. Zamyslite sa nad tým, či existuje viac programov v Jave alebo λ-termov.

Príklady λ-termov

- (λx x)
- (λx y)
- (λx (x x))
- ((λx (x x)) (λx (x x)))
- (λy (λx (x x)))

z týchto príkladov zatiaľ nie je evidentné, že to bude programovací jazyk.

Syntaktické konvencie

- malé písmená označujú premenné: x, y, x_1, x_2, …
- veľké písmená označujú λ-termy: M, N, …
- vonkajšie zátvorky nepíšeme

- symbol . nahradzuje (, zodpovedajúca) chýba
  - (λx x) -> λx.x
  - (λx (x x)) -> λx.xx   ale nie (λx.x)x
  - ((λx (x x)) (λx (x x))) -> (λx.xx)(λx.xx)

- vnorené abstrakcie majú asociativitu vpravo
  - λy.λx.x -> (λy (λx x))
  - (λy (λx (x x))) -> λy.λx.xx -> λyx.xx

- vnorené aplikácie majú asociativitu vľavo
  - f a b c -> (((f a) b) c)
  - (((λxyz.yz) a) b) c) -> (λxyz.yz)abc

Dôležité príklady termov

O ich dôležitosti sa dozvieme neskôr, keď budeme vedieť, ako sa v tejto teórii počíta

- K = λxy.x = λx.λy.x (funkcia s dvomi argumentami, výsledkom je 1.)
- I = λx.x (identita)
- S = λxyz.(xz)(yz) = λxyz.((x z) (y z)) (S a b c = (a c) (b c))

- ω= λx.xx = λx (x x)
- Ω = ωω = (λx.x x)(λx.x x) = ((λx (x x)) (λx (x x)))
- ω_3 = λx.xxx = λx.(xx)x = (λx ((x x) x))

Výpočet

- na to, aby sme vedeli počítať v tejto teórii, potrebujeme definovať redukčné pravidlo(á), ktoré simuluje krok výpočtu,
- redukčné pravidlo je založené na pojme substitúcie, ktorú, ak pochopíme len intuitívne, dostaneme intuitívne zlé výsledky,
- preto sa vybudovaniu substitúcie treba chvíľku venovať s pomocnými pojmami, ako je voľná premenná, ...
- musíme sa presvedčiť, že redukčné pravidlo má rozumné vlastnosti, zamyslíme sa nad tým, čo je rozumné...
- výpočet je opakované aplikovanie redukčného pravidla, a to kdekoľvek to v terme ide. Keď to už nejde, máme výsledok (tzv. normálna forma)
- môže sa stať, že rôznym aplikovaním red.pravidla prídeme k rôznym výsledkom, resp. rôznym výpočtom ???

Voľná premenná, podterm

- voľná premenná λ-termu
  - Free(x) = x
  - Free(λx.M) = Free(M) – {x}
  - Free(M N) = Free(M) ∪ Free(N)

viazaná premenná λ-termu
Bound(x) = {}
Bound(λx.M) = Bound(M) U {x}
Bound(M N) = Bound(M)∪Bound(N)

viazaná premenná a voľná premenná:
- λx.xy – y je voľná, x je viazaná

Bound(M) ∩ Free(M) =??? ∅

- podtermy λ-termu
  - Subt(x) = x
  - Subt(λx.M) = Subt(M) ∪ {λx.M}
  - Subt(M N) = Subt(M) ∪ Subt(N) ∪ { (M N) }

Príklady

- $\lambda x.\lambda y.(x\ z)$
  - x je viazaná,
  - z voľná,
  - y sa nenachádza v Subt($\lambda x.\lambda y.(x\ z)$)

- $\lambda x.((\lambda y.y)\ (x\ (\lambda y.y)))$
  - má dva výskyty podtermu ($\lambda y.y$)

- $(x\ (y\ z)) \in \text{Subt}(\ (w\ (x\ (y\ z)))\ )$
  - ale $(x\ (y\ z)) \notin \text{Subt}(\ w\ x\ (y\ z)\ ) = \text{Subt}(\ ((w\ x)\ (y\ z))\ )$,
  lebo
  - Subt(w x (y z)) obsahuje tieto podtermy:
    - ((w x) (y z)), (w x), (y z), w, x, z, y

  - teda w x (y z) = (w x)(y z)

Substitúcia

- ak sa na to ide naivne (alebo “textovo”):
  - (λx.zx)[z:y] -> λx.yx
  - (λy.zy)[z:y] -> λy.yy

Problém: ‘rovnaké’ vstupy po aplikovaní rovnakej operácie dali `rôzne’ výsledky – výrazy (λx.zx) a (λy.zy) *intuitívne* predstavujú rovnaké funkcie a po substitúcii [z:y] sú výsledky λx.yx a λy.yy *intuitívne* rôzne

- substitúcia N[x:M]

	x[x:M]	= M	-- x za x
	y[x:M]	= y	-- x za y
	(A B)[x:M]	= (A[x:M] B[x:M])	-- aplikácia
	(λx.B)[x:M]	= (λx.B)	-- viazaná λ
	(λy.B)[x:M]	= λ z.(B[y:z][x:M]) ak **x∈ Free(B), y∈ Free(M)**	
		ak z nie je voľné v B alebo M, z∉Free((B M)),x≠y	
inak	(λy.B)[x:M]	= λy.(B[x:M])	x≠y

- správne:(λy.zy)[z:y]->(λw.(zy)[y:w])[z:y]->(λw.(z w))[z:y] ->λw.yw

naivne
(λx.zx)[z:y] ->
λx.yx
(λy.zy)[z:y] ->
λy.yy

Príklady

inak
(λy.B)[x:M] = λy.(B[x:M])
x ∉ Free(B) || y ∉ Free(M)

- (λx.zx)[z:y] =
  - λx.((zx)[z:y]) =
  - λx.(z[z:y]x[z:y]) =
  - λx.(yx)

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

- (λy.zy)[z:y] =
  - (λw.(zy)[y:w])[z:y] = – treba si vymyslieť novú premennú
  - (λw.(z[y:w]y[y:w]))[z:y] =
  - (λw.(zw))[z:y] =
  - λw.(zw)[z:y] =
  - λw.(z[z:y]w[z:y]) =
  - λw.(yw)

Vlastnosti substitúcie 1

Ak premenná x nie je voľná v M, $x \notin Free(M)$, potom M[x:N] = M.

Dôkaz indukciou...

- M = x, neplatí predpoklad...
- M = y, potom y[x:N]=y=M
- M = (A B), potom z x nie je voľná v (A B) vyplýva, že x nie je voľná v A aj v B, (A B)[x:N] = (A[x:N] B[x:N]) = ind.predp. (A B) = M
- M = (λx.B), potom (λx.B)[x:N] = (λx.B) = M
- M = (λy.B), potom x nie je voľná B, (λy.B)[x:N] = (λx.B[x:N]) = M

Ak Free(M) = ∅, M nazývame uzavretý výraz.

Dôsledok: Uzavretý výraz sa aplikáciou substitúcie nezmení.

Vlastnosti substitúcie 2

Lemma:

Predopoklady:

- $x \neq y$ sú rôzne premenné,
- x nie je voľná v L, $x \notin Free(L)$,
- ak každá viazaná premenná v M nie je voľná v (N L)
  $v \in Bound(M) \Rightarrow v \notin Free((N\ L))$,

potom

1. M[x:N] [y:L] = M[y:L][x:N[y:L]] -- superpozícia substitúcií
2. M[x:N] [y:L] = M[x:N[y:L]] -- skladanie substitúcií

1) $M[x:N]\,[y:L] \quad = M[y:L][x:N[y:L]]$

superpozícia substitúcií

Predpoklady:
$x \neq y$
$x \notin Free(L)$,
$v \in Bound(M) \Rightarrow v \notin Free((N\ L))$

Indukciou vzhľadom na M:

- M je premenná
  - M = x, obe strany sú N[y:L]
  - M = y, obe strany sú L, lebo x nie je voľná v L,
  - M = z, rôzna premenná od x,y, potom obe strany sú z.
- M =(λz.Q)
  - z = x, L.S.=(λx.Q)[x:N] [y:L]= (λx.Q)[y:L]=<u>λx.(Q[y:L])</u>,
    P.S.=(λx.Q) [y:L][x:N[y:L]]= (λx.Q[y:L])[x:N[y:L]]= <u>λx.(Q[y:L])=L.S.</u>
  - z = y, obe strany sú (λy.(Q[x:N])), lebo y je viazaná v M, preto nie je voľná v (N L), takže ani N
  - z je rôzne od x,y, potom, podľa predpokladu, z nie je voľná v N ani L
    (λz.Q)[x:N] [y:L] = λz.(Q[x:N]) [y:L] =
    λz.(Q[x:N][y:L]) = … indukcia
    λz.(Q[y:L][x:N[y:L]]) = (λz.Q)[y:L][x:N[y:L]].
- M =(Q R), no problem, indukciou na Q a R...

2) M[y:N] [y:L] = M[y:N[y:L]]

skladanie substitúcií

Domáca úloha:

podobne dokážte tvrdenie 2) predchádzajúcej lemmy.

Domáca úloha:

za akých podmienok (najslabších) platí, navrhnite a zdôvodnite, dokážte...

- M[x:N] [y:L] = M[y:L][x:N]
- M[x:y] [y:N] = M[x:N]
- M[x:y] [y:x] = M

α-konverzia

$$\lambda x.M =_\alpha \lambda y.M[x:y]$$

- $\lambda x.M$ je premenovaním viazanej premennej $\lambda y.M[x:y]$, ak y nie je voľná v M

- $=_\alpha$ je relácia ekvivalencie
- $=_\alpha$ kongruencia na λ termoch

- intuícia: výrazy, ktoré sa odlišujú menom viazanej premennej predstavujú rovnaké funkcie
- Príklad:

  $\lambda x.x =_\alpha \lambda y.y$

  $\lambda x.(x\ y)\ !=_\alpha \lambda y.(y\ y)$ ale $\lambda x.(x\ y) =_\alpha \lambda z.(z\ y)$

$K = \lambda xy.x$
$I = \lambda x.x$
$S = \lambda xyz.xz(yz)$

β-redukcia

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

Príklad:

- $I\ M = x$
  - $(\lambda x.x)\ M \to_\beta x[x:M] = M$
- $K\ M\ N = M$
  - $(\lambda xy.x)MN \to_\beta (\lambda y.M)N \to_\beta M$
- $S\ M\ N\ P = M\ P\ (N\ P)$
  - $\lambda xyz.xz(yz)\ MNP \to^3_\beta MP(NP)$

- $S\ K\ S = ?$
- $\lambda xyz.\ ((xz)(yz))\quad (\lambda xy.x)\quad (\ \lambda xyz.\ ((xz)(yz))\ ) \to_\beta$

- $S\ K\ K = ?$
  - $\lambda xyz.\ ((xz)(yz))\quad (\lambda xy.x)\quad (\lambda xy.x) \to_\beta$
  - ???

$K = \lambda xy.x$
$I = \lambda x.x$
$S = \lambda xyz.xz(yz)$

β-redukcia

$$(\lambda\ x.B)\ E \to_\beta B[x{:}E]$$

Príklad:

- $I\ M = x$
  - $(\lambda x.x)\ M \to_\beta x[x{:}M] = M$
- $K\ M\ N = M$
  - $(\lambda xy.x)MN \to_\beta (\lambda y.M)N \to_\beta M$
- $S\ M\ N\ P = M\ P\ (N\ P)$
  - $\lambda xyz.xz(yz)\ MNP \to^3_\beta MP(NP)$
- $S\ K\ K = I$
  - $\lambda xyz.\ ((xz)(yz))\ (\lambda xy.x)\ (\lambda xy.x) \to_\beta$
  - $\lambda yz.\ (\ ((\lambda xy.x)z)\ (yz)\ )\ (\lambda xy.x) \to_\beta$
  - $\lambda z.\ ((\lambda xy.x)z((\lambda xy.x)z)\ ) \to_\beta$
  - $\lambda z.\ ((\lambda y.z)((\lambda xy.x)z)\ ) \to_\beta$
  - $\lambda z.\ ((\lambda y.z)(\lambda y.z)) \to_\beta$
  - $\lambda z.z = I$

Vlastnosti β-redukcie

- $\omega = \lambda x.xx = \lambda x.(x\ x)$
- $\Omega = \omega\omega$
- $\omega_3 = \lambda x.((x\ x)\ x)$

- nekonečná sekvencia
  - $\Omega \to_\beta \Omega \to_\beta \Omega \to_\beta \ldots$
- 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 pre dva rôzne výpočty ($K = \lambda xy.x$)
  - $KI\Omega \to_\beta I$ ale aj
  - $KI\Omega \to_\beta KI\Omega \to_\beta KI\Omega \to_\beta \ldots$

Cvičenie: overte si tieto tvrdenia,
Pokúste sa nájsť λ term, ktorý vedie k rôznym výsledkom ☺

Cvičenie

- $\Omega = \omega\omega = ((\lambda x.xx)\ (\lambda x.xx)) \to_\alpha (\lambda z.zz)\ (\lambda x.xx) \to_\beta$
  $((\lambda x.xx)\ (\lambda x.xx)) \to_\beta \ldots$

- $\omega_3\ \omega_3 \to_\beta (\lambda x.((x\ x)\ x)\ \lambda x.((x\ x)\ x)\ ) \to_\alpha$
  $(\lambda z.((z\ z)\ z)\ \lambda x.((x\ x)\ x)\ ) \to_\beta$
  $((\omega_3\ \omega_3)\ \omega_3) \to_\beta \ldots (((\omega_3\ \omega_3)\ \omega_3)\ \omega_3)\ ) \to_\beta \ldots$

- $KI\Omega$
  $(\lambda xy.x)I\ \Omega \to_\beta I$
  $(\lambda xy.x)I\ \Omega \to_\beta (\lambda xy.x)I\ \Omega \to_\beta (\lambda xy.x)I\ \Omega \to_\beta (\lambda xy.x)I\ \Omega$

η-redukcia

- $\lambda x.(B\ x) \to_\eta B$ ak $x \notin Free(B)$
  - Dôvod:
  - L.S. = $(\lambda x.(B\ x))M \to_\beta (B[x:M]\ M) \to_\beta (B\ M)$
  - P.S. = $(B\ M)$

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

- $\to_{\beta\eta}$ je uzáver $\to_\beta \cup \to_\eta$ vzhľadom na podtermy, čo znamená
  - 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 $\lambda x.M \to_{\beta\eta} \lambda x.N$.

Domáca úloha

Definujte základné funkcie pre interpreter λ-kalkulu:

- free - zistí, či premenná je voľná
- subterm - vráti zoznam podtermov
- substitute - korektne implementuje substitúciu
- oneStepBetaReduce
- normalForm - opakuje redukciu, kým sa dá

navrhovaná reprezentácia (kľudne si zvoľte inú):

```
data LExp = LAMBDA String LExp |      – abstrakcia
            ID String |               – premenná
            LExp [LExp] |             – aplikácia, zovšeobecnená
            App LExp LExp |           – aplikácia
            CON String |              – konštanta, built-in fcia
            CN Integer                – int.konštanta
        deriving(Show, Read, Eq)
```

Cvičenie (použite váš tool)

1) určite voľné a viazané premenné:

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

2) redukujte:

- (λx.λy.x (λz.y z)) (((λx. λy.y) 8) (λx.(λy.y) x))
- (λh.(λx.h (x x)) (λx.h (x x))) ((λa.λb.a) (+ 1 5))

3) Nech F = (λt.t t) (λf.λx.f (f x)).
Vyhodnoťte F succ 0, succ = λx. (+ x 1)

Riešenie (zle)

- (λx.λy.x (λz.y z)) (((λx. λy.y) 8) (λx.(λy.y) x)) $->_\beta$
  - (λx.λy.x (λz.y z)) ((λy.y) (λx.(λy.y) x)) $->_\beta$
  - (λx.λy.x (λz.y z)) ((λy.y) (λy.y)) $->_\beta$
  - **(λx.λy.x (λz.y z))** (λy.y) $->_\beta$
  - …(**λy.x)[x:(λz.y z)]** … **y**∈**Free(λz.y z)** , x:Free(x)
  - **λy.(λz.y z)** (λy.y) $->_\beta$
  - (λz.(λy.y) z) $->_\beta$
  - (λz.z) $->_\beta$
  - I

Riešenie (dobre)

Nájdite pomocou vášho nástroja pre vyhodnocovanie λ-výrazov

- (λx.λy.(x ((λz.y) z))) (((λx. λv.v) 8) (λx.(λw.w) x)) $->_{\beta}$
  - (λx.λy.(x ((λz.y) z))) ((λv.v) (λx.(λw.w) x)) $->_{\beta}$
  - (λx.λy.(x ((λz.y) z))) ((λv.v) (λw.w)) $->_{\beta}$
  - (λx.λy.(x ((λz.y) z))) (λv.v) $->_{\beta}$
  - λy.((λv.v) ((λz.y) z)) $->_{\beta}$
  - λy.(((λz.y) z)) $->_{\beta}$
  - λy.y

Riešenie

- $(\lambda h.(\lambda x.h\ (x\ x))\ (\lambda x.h\ (x\ x)))\ ((\lambda a.\lambda b.a)\ (+\ 1\ 5)) \to_\beta$
  - $(\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x))\ (\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x)) \to_\beta$
  - $((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ ($
    $(\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x))\ (\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x))) \to_\beta$
  - $(\lambda b.(+\ 1\ 5)\ ($
    $(\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x))\ (\lambda x.((\lambda a.\lambda b.a)\ (+\ 1\ 5))\ (x\ x))) \to_\beta$
  - $(+\ 1\ 5) \to_\beta$
  - 6

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.

čo by Vás mohlo inšpirovať

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)
```

Od λ-termu k programu

Na to, aby sme vedeli v tomto jazyku programovať, potrebujeme:

- mať v ňom nejaké hodnoty, napr. aspoň int, bool, ...
- základné dátové typy, záznam (record, record-case), zoznam, ...
- if-then-else
- let, letrec, či where
- rekurziu

V ďalšom obohatíme λ-kalkul syntaktickými cukrovinkami tak, aby sme sa presvedčili, že sa v tom programovať naozaj dá.

Rekurzia pomocou operátora pevného bodu bude najnáročnejším klincom v tejto línii.

Churchove čísla

- $\mathit{0} := \lambda f.\lambda x.x$
- $\mathit{1} := \lambda f.\lambda x.f\ x$
- $\mathit{2} := \lambda f.\lambda x.f\ (f\ x)$
- $\mathit{3} := \lambda f.\lambda x.f\ (f\ (f\ x))$

. . . . .

$C\,(+1)\ 0 = c$

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

Domáca úloha (povinná):
•definujte mult,
•definujte 2^n, m^n ,
•definujte n-1

Logické hodnoty a operátory

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

AND := $\lambda x.\lambda.y.\ x\ y$ FALSE := $\lambda xy.x\ y$ FALSE
OR := $\lambda x.\lambda y.\ x$ TRUE y := $\lambda xy.x$ TRUE y
NOT := $\lambda x.\ x$ FALSE TRUE
IFTHENELSE := $\lambda pxy.\ p\ x\ y$

AND TRUE FALSE
$\equiv$ (λ p q. p q FALSE) TRUE FALSE $\rightarrow_\beta$ TRUE FALSE FALSE
$\equiv$ (λ x y. x) FALSE FALSE $\rightarrow_\beta$ FALSE

Cvičenie: 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

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

LEFT (PAIR A B) ≡

- LEFT (($\lambda xyc.\ c\ x\ y$) A B) $\rightarrow_\beta$
- LEFT ($\lambda c.\ c$ A B) $\rightarrow_\beta$
- ($\lambda x.x$ TRUE) ($\lambda c.\ c$ A B) $\rightarrow_\beta$
- ($\lambda c.\ c$ A B) ($\lambda xy.x$) $\rightarrow_\beta$
- (($\lambda xy.x$) A B) $\rightarrow_\beta$ A

Cvičenie: definujte n-ticu

Curry
$\lambda(x,y).M \rightarrow \lambda p.\ (\lambda x\lambda y.M)$ (LEFT p) (RIGHT p)

Súčet typov (disjunkcia)

A+B reprezentujeme ako pár [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 $\to_\beta$
RIGHT (PAIR TRUE A) $\to_\beta$ A

Cvičenie: reprezentujte zoznam s konštruktormi Nil, Cons a funkciami isEmpty, head a tail

where (let, letrec)

M where v = N -> (λv.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