β-redukce

(λx.M)N ≈ [N/x] M

η-redukce

(λx.Mx) ≈ M

Definice

True = λxλy.x

False = λxλy.y

not = λy.y False True

if = λxλyλz.xyz

and=λuλv.u v False

or=λuλv.u True v

(M,N)=λz.z MN

fst = λp.p True

snd = λp.p False

[]= λz.z T F F

(:)=λxyz.z F x y

zeroes=Y(λz.(:) 0 s)

Churchovy čísl.

0 = λs.z z

1 = λs.z sz

n = λs.z s^nz

KOMBINÁTORY S,K,I,B,C

S :: (α→β→γ)→(α→β)→α→γ

S gfx → fx(gx), S ≈ λfλgλx.fx(gx)

K :: α→β→α, K xy → x, K ≈ λxλy.x

I :: α→α, I x → x, I ≈ λx.x

IM → M, I ≈ SKK	S(K*P*)*Q* → B*PQ*
SMNP → MP(NP)	*SP*(K*Q*) → C*PQ*
KMN → M	S(K*P*)(K*Q*) → K(*PQ*)
BMNP → M(NP)	S(K*P*)I → *P*
CMNP → MPN	

λx.M → KM, λx.Mx → M, λx.x → I

λx.y → Ky, λx.E_1E_2 → S(λx.E_1)(λx.E_2)

λx.PQ → S(λx.P)(λx.Q) →* S(KP)(KQ)

λx.Mx → S(λx.M)(λx.x) →* S(KM)I

Příklad: Θ xyz → zxy

λxλyλz.zxy

λxλy.S(λz.zx)(λz.y)

λxλy.S(S(λz.z)(λz.x))(Ky)

λxλy.S(S(I)(Kx))(Ky)

λxλy.S(CIx)(Ky)

λxλy.C(CIx)y η-redukce

λx.C(CIx)

S(λx.C)(λx.CIx)

S(KC)(CI)

BC(CI)

SUPERKOMBINÁTORY, λ-lifting

Příklad: Převod termu do superkomb. termu

λxλy.F(λz.Fxz)(λz.Fzzy)(λz.Fzzx)

1. α x z = F x z

λxλy.F(αx)(λz.Fzzy)(λz.Fzzx)

2. β y z = F z z y

λxλy.F(αx)(βy)(λz.Fzzx)

3. γ x z = F z z x

pouhým přejmenováním x na y získáme β ≈ γ = β

λxλy.F(αx)(βy)(βx)

4. δ x y = F(αx)(βy)(βx)

λx.δx $\to_\eta$ δ

výsledné superkombinátory: α, β, δ

KOMBINÁTOR PEVNÉHO BODU

Pro každý term *F* existuje term *X* takový, že *F X* ≈ *X*.

YF ≈ F(YF), YF → F(YF) → F(F(YF)) ...

z ≈ *F z*, *z* = **Y***F* = **Y**(*λz.M*)

Příklad 1:

from :: Int → [Int]

from n = n : from(succ n)

zafixovat funkci a její proměnné

from = Y(λfλn.n : f (succ n))

Příklad 2:

data Tree = Empty | Node [Int] Tree Tree

t = f [] where f s = Node s (f(0:s)) (f(1:s))

nejprve vezmeme funkci *f*

f = Y(λφλs.Node s (φ(0:s)) (φ(1:s)))

pak pomocí *f* vyjádříme funkci *t*

t = Y(λφλs.Node s (φ(0:s)) (φ(1:s))) []

Příklad 3:

Y(λx."Pes...slov:„"++x++"“")

Vícenásobná rekurze

X = Y(λz.(F_1(p_1 z)...(p_n z), ..., F_n(p_1 z)...(p_n z))) p ... selektor

Příklad 1:

even = λn. if (iszero n) then True else (odd(pred n))

odd = λn. if (iszero n) then False else (even(pred n))

nejprve vezmeme novou funkci z, která bude tvořena dvojicí (podle počtu vzájemně rekurzivních funkcí)

z = (even, odd) = (λn. if (iszero n) then True else (odd(pred n)),
λn. if (iszero n) then False else (even(pred n)))

nahradíme funkce odd a even selektory na typu z

z = (λn. if (iszero n) then True else (snd z (pred n)),
λn. if (iszero n) then False else (fst z (pred n)))

z = Y(λz.(λn. if (iszero n) then True else (snd z (pred n)),
λn. if (iszero n) then False else (fst z (pred n))))

Příklad 4:

cycl :: [a]→[a], cycl s = s ++ cycl s

chceme ziskat definici funkce cycl ve tvaru: cycl =Y(_________),

v definici ‘cycl s = s ++ cycl s‘ převedeme s doprava a zavedeme novou funkci f, kterou nahradíme funkci cycl.

λfλs.s ++ f s, cycl = Y(λfλs.s ++ f s)

IMPREDIKATIVNÍ TYPOVÝ SYSTÉM I

Příklad 1:

data Bitlist = Nil | ConsOn Bitlist | ConsOff Bitlist

zvolíme si nějaký typ, který bude reprezentovat typ Bitlist, např: τ

Bitlist = ∀τ.τ→(τ→τ)→(τ→τ)→τ

Příklad 2:

data Either a b = Left a | Right b

Either α β = ∀τ.(α→τ)→(β→τ)→τ

Příklad 3:

data Trojice a b c = T a b c

Trojice α β γ = ∀τ.(α→β→γ→τ)→τ

IMPREDIKATIVNÍ TYPOVÝ SYSTÉM II

Odvozovací pravidla pro predikativní typový systém a termy s let.

Specializace (SPEC) $\dfrac{\Gamma \rhd M :: \forall\alpha.\sigma}{\Gamma \rhd M :: [\tau/\alpha]\sigma}$ (τ typ)

Abstrakce (ABS) $\dfrac{\Gamma, x :: \sigma \rhd M :: \tau}{\Gamma \rhd \lambda x.M :: \sigma \to \tau}$

Let-výraz (LET) $\dfrac{\Gamma \rhd N :: \sigma \quad \Gamma, x :: \sigma \rhd M :: \tau}{\Gamma \rhd \text{let } x = N \text{ in } M :: \tau}$

Axiom proměnné (VAR) $\dfrac{}{\Gamma, x :: \sigma \rhd x :: \sigma}$

Zeslabení (W) $\dfrac{\Gamma \rhd M :: \sigma}{\Gamma, x :: \tau \rhd M :: \sigma}$ $x \neq FV(M)$

Aplikace (APP) $\dfrac{\Gamma \rhd N :: \sigma \to \tau \quad \Gamma \rhd M :: \sigma}{\Gamma \rhd NM :: \tau}$

Generalizace (GEN) $\dfrac{\Gamma \rhd M :: \sigma}{\Gamma \rhd M :: \forall\alpha.\sigma}$ $\alpha \neq FV(\Gamma)$

Axiomy konstant (CON) $\dfrac{}{\rhd 0 :: Nat}$ $\dfrac{}{\rhd False :: Bool}$

Příklad 1:

$0 :: Nat$, $(+) :: Nat \to Nat \to Nat$. Výraz: let $f = \lambda x \lambda y.x\ y\ 0$ in $f\ f\ (+)$, $\Delta = \{f :: \forall\alpha\forall\beta.(\alpha \to Nat \to \beta) \to \alpha \to \beta\}$

$[*1] = [Nat \to Nat \to Nat/\alpha, Nat/\beta]$ $\qquad$ $[*2] = [Nat/\alpha, Nat/\beta]$

$$\dfrac{\Delta \rhd f :: \forall\alpha\forall\beta.(\alpha \to Nat \to \beta) \to \alpha \to \beta}{\Delta \rhd f :: [*1]((Nat \to Nat \to Nat) \to Nat \to Nat) \to (Nat \to Nat \to Nat) \to Nat}\ \text{SPEC}$$

$$\dfrac{\Delta \rhd f :: \forall\alpha\forall\beta.(\alpha \to Nat \to \beta) \to \alpha \to \beta}{\Delta \rhd f :: [*2](Nat \to Nat \to Nat) \to Nat \to Nat}\ \text{SPEC}$$

$$\dfrac{\Delta \rhd f :: [*1]\ldots \quad \Delta \rhd f :: [*2]\ldots}{\Delta \rhd f\ f :: (Nat \to Nat \to Nat) \to Nat}\ \text{APP}$$

$$\dfrac{\Delta \rhd f\ f :: (Nat \to Nat \to Nat) \to Nat \quad \dfrac{}{\Delta \rhd (+) :: Nat \to Nat \to Nat}\ \text{CON}}{\Delta \rhd f\ f\ (+) :: Nat}\ \text{APP}$$

$$\dfrac{\Gamma \rhd \lambda x \lambda y.x\ y\ 0 :: \forall\alpha\forall\beta.(\alpha \to Nat \to \beta) \to \alpha \to \beta \qquad \Delta \rhd f\ f\ (+) :: Nat}{\text{let } f = \lambda x \lambda y.x\ y\ 0 \text{ in } f\ f\ (+) :: Nat}\ \text{LET}$$

Odvození základního typu: $f\ f\ (+) \approx (\lambda x \lambda y.x\ y\ 0)\ \underline{(\lambda x \lambda y.x\ y\ 0)}\ \underline{(+)} \approx_{2\beta} (\lambda x \lambda y.x\ y\ 0)\ \underline{(+)}\ \underline{0} \approx_{2\beta} (+)\ 0\ 0 :: Nat$

Příklad 2:

$0 :: Nat$, $succ :: Nat \to Nat$. Výraz: let $f = \lambda x \lambda y.x(x\ y)$ in $f\ f\ succ\ 0$, $\Delta = \{f :: \forall\alpha.(\alpha \to \alpha) \to \alpha \to \alpha\}$

Odvození základního typu: $f\ f\ succ\ 0 \approx (\lambda x \lambda y.x(x\ y))\ \underline{(\lambda x \lambda y.x(x\ y))}\ \underline{succ}\ 0 \approx_{2\beta} (\lambda x \lambda y.x(x\ y))\underline{((\lambda x \lambda y.x(x\ y))\ succ)}\ \underline{0} \approx_{2\beta}$ $((\lambda x \lambda y.x(x\ y))\ \underline{succ})(((\lambda x \lambda y.x(x\ y))\ \underline{succ})\ 0) \approx_{1\beta} (\lambda y.succ(succ\ y))\underline{((\lambda y.succ(succ\ y))\ 0)} \approx_{1\beta} (\lambda y.succ(succ\ y))\underline{(succ(succ\ 0))} \approx_{1\beta}$ $(succ(succ(succ(succ\ 0)))) :: Nat$

PARSER - SYNTAKTICKÁ ANALÝZA

Příklad 1: V následujícím programu na syntaktickou analýzu lambda-termů parser ***term*** analyzuje termy složené z (jednopísmenných) proměnných, aplikací a z abstrakcí tvaru ***\v.M*** . Vícenásobné abstrakce vyžaduje ve tvaru např. ***\x.\y.\z.M*** . Předefinujte dvě lokální hodnoty ***variables*** a ***mklam*** v definici parseru ***lam*** tak, aby parser ***term*** akceptoval vicenasobné abstrakce i ve tvaru ***\x y z.M*** .

```
data Term = Var Char | App Term Term | Lam Char Term

lam = do symbol "\\"
         s ← many1(token letter) \\přičtení 1 písmenka a mezery za ním
         symbol "."
         t ← term
         return (foldr Lam t s) \\Lam ‘x’(Lam ‘y’ (Lam ‘z’ t)) - funkce foldr nám to takto uspořádá
```

$foldr\ l\ t\ [x_1, x_2, \ldots, x_n] = l\ x_1\ (l\ x_2\ (\ldots\ (l\ x_n\ t)\ldots))$

```
term :: Parser Term
term = atom ‘chainl1’ return App

atom :: Parser Term
atom = var +++ lam +++ paren

var :: Parser Term
var = token letter >>= return . Var

lam :: Parser Term
lam = symbol "\\" >> variables >>= \v -> symbol "." >>
         term >>= return . mklam v
           where variables = token letter
                 mklam     = Lam

parent :: Parser Term
parent = bracket (symbol "(") term (symbol ")")
```

(\x.x y) y (\y.z)

```
term :: Parser Term
term = atom ‘chainl1’ return App  \\z minulého příkladu

atom :: Parser Term
atom = var +++ lam +++ paren

var :: Parser Term
var = token letter >>= return . Var

lam :: Parser Term
lam = symbol „\\“ >> variable >>= \x -> symbol „.“ >>
         term >>= return . Lam x

parent :: Parser Term
parent = bracket (symbol „(„) term (symbol „)“)

variable :: Parser Char
variable = token ident >>= return . head
```