FLP - λ-kalkul (definice všech pojmů, operací, ...)

Syntaxe

- nechť E je výraz λ-kalkulu

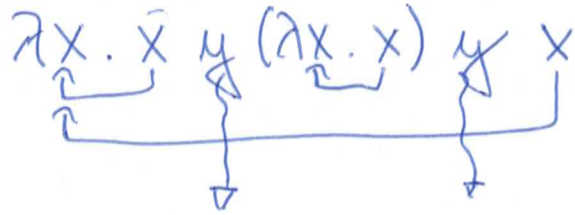
$E ::= V$ (proměnná)

$(E_1\ E_2)$ (aplikace)

$(\lambda V.\ E)$ (abstrakce)

Volné a vázané proměnné

- proměnná je vázaná pokud je uvnitř λ abstrakce a je v její hlavičce



Substituce

$E[E'/V]$

čím — kterými nahrazuji — co nahrazuji

- volné výskyty V jsou nahrazeny E' ve výrazu E

Platnost substituce

- žádná volná proměnná v E' se nesmí stát vázanou v $E[E'/V]$

$(\lambda x\ y.\ x)[x/x] = (\lambda x\ y.\ x)$ – nic se nezmění, protože x není volné

$(\lambda x\ y.\ x)[y/x] = (\lambda x\ y.\ y)$

α-konverze "přejmenování"

$\lambda V.E \to_\alpha \lambda V'.E[V'/V]$ - může přejmenovat pouze proměnné v hlavičce

Př:

$\lambda x.xy \to_\alpha \lambda z.zy$

β-konverze - není závislost λ výrazu

$(\lambda V.E_1)E_2 \to_\beta E_1[E_2/V]$

$(\lambda x\,y.x\,y)(x\,y) \to_\alpha (\lambda x\,z.x\,z)(x\,y) \to_\beta (\lambda z.(x\,y)\,z)$

η-konverze (eta-konverze)

$\lambda V.(E\,V) \to_\eta E$

V není volné v E

$(\lambda x.E\,x)Q \to_\beta E\,Q$

$\downarrow_\eta$

$E\;Q$

Identita

$E_1 \equiv E_2$ $\quad \lambda x.x \equiv \lambda x.x \not\equiv \lambda y.y$

Rovnost

$E_1 = E_2$ $\quad * \to_{\alpha_1} \to_{\beta_1} \to_{\mu_1} \leftarrow_{\alpha_1} \leftarrow_{\beta_1} \leftarrow_\mu$

$\lambda x.x \to_\alpha \lambda y.y \equiv \lambda y.y$

$\lambda x.x = \lambda y.y$

Relace →

$E_a \to E_b$

$E_a \equiv E_0 * E_1 * E_2 \ldots E_n \equiv E_b$

$* \sim \to_\alpha \to_\beta \to_\eta$

Bottom

= výraz, který bude neustále na výstup produkovat sebe navzemě bottoma

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

Operátor pevného bodu Y

Pevný bod výrazu E je r_E t.ž $Er_E = r_E$

$YE = r_E = Er_E = E(YE)$

$YE = E(YE)$

Normální forma

Výraz E je v normální formě, pokud neobsahuje žádné jiné redexy krom α-redexu.

např. $(\lambda x y . x)$, $\nrightarrow y$

$(\lambda x . yx) \to_\eta y$ ← tohle už je v NF

Normalizační teorém

- pokud má výraz E normální formu, potom opakovaná redukce nejlevějšího β,η-redexu bude končit ve výrazu v NF.

- Pozn. posloupnost redukcí, v níchž je vždy redukován nejlevější redex, se nazývá posloupnost redukcí v normálním pořádí

FLP - Práce v λ-kalkulu (reprezentace čísel, pravdivostních hodnot a práce nad nimi)

Reprezentace pravdivostních hodnot

LET True = λ t f. t

LET False = λ t f. f

LET Not = λ t. t False True

XOR	a	b	R
	0	0	0
	0	1	1
	1	0	1
	1	1	0

LET XOR = λ a b. a (Not b) b

AND	a	b	R
	0	0	0
	0	1	0
	1	0	0
	1	1	1

LET AND = λ a b. a b False

Reprezentace čísel v λ-kalkulu

LET $0 = \lambda f\, n.\, n$

LET $1 = \lambda f\, n.\, f\, n$

LET $2 = \lambda f\, n.\, f\,(f\, n)$

LET succ $= \lambda x\, g\, m.\, x\, g\,(g\, m)$

LET $(?:) = \lambda c\, t\, f.\, c\, t\, f$

LET $(\_,\_) = \lambda f\, s\, e.\, e\, f\, s$

LET first $= \lambda x.\, x$ True

LET second $= \lambda x.\, x$ False

LET add $= \lambda x\, y\, g\, m.\, x\, g\,(y\, g\, m)$

LET sub $= \lambda x\, y.\, y$ pred a

LET predn $= \lambda f\, p.\,($first p ? (False, second p) : (False, f (second p)))

LET pred $= \lambda x\, g\, m.$ second $(x$ (predn g) (True, m))

Multiplikace

mult n $0 = 0$

mult n $y+1 = n +$ mult (n, y)

;

LET mult $= Y(\lambda f\, x\, y.$ iszero y ? 0 : add x (f x (pred y)))

Dělení v λ-kalkulu

$$\text{div } x\ 0 = \bot$$

$$\text{div } x\ y = \text{divfn } x\ y\ 0$$

$$\text{divfn } x\ y\ r = x < y\ ?\ r : \text{divfn } (x-y)\ y\ (r+1)$$

$$\text{LET div} = \lambda x\,y.\ \text{iszero } y\ ?\ \bot : \text{divfn } x\ y\ 0$$

$$\text{LET divfn} = Y(\lambda f\,x\,y\,r.\ gt\ y\ x\ ?\ r : f\ (\text{sub } x\ y)\ y\ (\text{add } r\ 1))$$

λ-kalkul – Základní pojmy

1. λ-výraz, volné a vázané proměnné, substituce
2. α-konverze
3. β-konverze
4. η-konverze
5. normální forma
6. Bottom
7. Operátor pevného bodu
8. Posloupnost redukcí v normálním pořadí

λ-kalkul – Práce v λ-kalkulu

1. Reprezentace TRUE, FALSE
2. Ternární operátor
3. NOT, XOR
4. Definuj (–,–), fst, snd
5. Reprezentace přirozených čísel 0, 1, 2, ..., pred, succ
6. Definuj +, −, *, div pomocí rekurze

Operátor pevného bodu Y

$$Y E = k_E$$

k_E je pevný bod E, tedy $E\, k_E = k_E$

$$Y E = k_E = E\, k_E = E\,(Y E)$$

Bottom $\bot$

- výraz, který na výstupu neustále produkuje sám sebe

$$\text{LET } \bot = Y(\lambda f x.\, f)$$

α-konverze "přejmenování"

$$\lambda V.\, E \to_\alpha \lambda V'.\, E[V'/V]$$

β-konverze

$$(\lambda V.\, E)\, E' \to_\beta E[E'/V]$$

např. $(\lambda a b.\, a)\, x \to_\beta \lambda b.\, x$

η-konverze (eta konverze)

$\lambda V.\, E V \to_\eta E$, pokud V není volné v E

např. $(\lambda x.\, f x) \to_\eta f$

1

LET TRUE = $\lambda x\, y.\, x$

LET FALSE = $\lambda x\, y.\, y$

LET NOT = $\lambda v.\, v$ FALSE TRUE

LET ?: = $\lambda v\, t\, f.\, v\, t\, f$

LET XOR = $\lambda a\, b.\, a$ (NOT b) b

XOR

a	b	
0	0	0
0	1	1
1	0	1
1	1	0

Definuj $(\_,\_)$, fst, snd

LET $(\_,\_)$ = $\lambda f\, s\, a.\, a\, f\, s$

LET fst = $\lambda x.\, x$ TRUE

LET snd = $\lambda x.\, x$ FALSE

Přirozená čísla

LET 0 = $\lambda f\, z.\, z$

LET 1 = $\lambda f\, z.\, f\, z$

LET 2 = $\lambda f\, z.\, f\,(f\, z)$

LET 3 = $\lambda f\, z.\, f\,(f\,(f\, z))$

LET succ = $\lambda x\, g\, m.\, g\,(x\, g\, m)$

LET succ = $\lambda x\, g\, m.\, x\, g\,(g\, m)$

Definuj pred - potřebuju obebrat jedno f

pred 1 = $\lambda f\, n.\ n = 0$

pred $(\lambda f\, n.\ f\ \hat{n}) = \lambda f\, n.\ n = 0$

pred $(\lambda f\, n.\ f\ (f\ n)) = \lambda f\, n.\ f\ n = 1$

(pár, čísla)

predfn = $\lambda x\, g.$ snd x ? (fst x, False) : (g (fst x), False)

pred = $\lambda x\, g\, m.$ x (predfn g) (m, True) fst)

Definuj +

0 + 1 = 1

LET plus = $\lambda x\, y\, g\, m.$ x g (y g m)

LET sub = $\lambda x\, y\, g\, m.$ y pred (x g m)

nebo sub = $\lambda x\, y.$ y pred x

Multiplikace pomocí rekurze

DEF is_zero = $\lambda x.$ x $(\lambda y.$ False) True

mul x 0 = 0

mul x 1 = x

mul x (y+1) = x + mul x y

neboli

mul x 0 = 0

mul x y = x + mul x (y-1)

LET mul = $\lambda x\, y.$ is_zero y ? 0 : add x (f x (prev y))

Opakování - kombinační kalkul

Lambda výrazy definujeme rekurzivně takto

$E := x \quad x \in \mathbb{X}$, kde $\mathbb{X}$ je množina proměnných

$:= (A\ B)$, kde A, B jsou lambda výrazy (aplikace)

$:= (\lambda x.\ E')$, kde $x \in \mathbb{X}$ a E' je lambda výraz (abstrakce)

- jsou to řetězce - nemají sémantiku

Lambda kalkul je pak výpočetní systém nad λ-výrazy, který je silou ekvivalentní k TS

Vyjadřování

λ-konverze (přejmenování)

$\lambda V.\ E \to_\alpha \lambda V'.\ E[V'/V]$

např. $\lambda x.\ x \to_\alpha \lambda y.\ y$

β-konverze

$(\lambda V.\ E_1)\ E_2 \to_\beta E_1[E_2/V]$

např. $(\lambda x.\ x)\ y \to_\beta y$ - musí jít o platnou substituci

η-konverze

$(\lambda V.\ E\ V) \to_\eta E$ pokud V není volné v E

např. $(\lambda x.\ f\ x)\ \text{[illegible]} \to_\eta f$ [illegible]

Normální forma λ-výrazu

λ-výraz je v normální formě, pokud neobsahuje žádný β nebo η redex (α-obsahovat může)

Bottom

- je to výraz, který pro jakýkoliv argument vrací vždy sám sebe

$$\text{LET } \bot = Y(\lambda f x . f)$$

Operátor pevného bodu Y

Pevný bod výrazu E je k_E právě, že $E\, k_E = k_E$, $YE = k_E$.

Pak platí, že

$$YE = k_E = E\, k_E = E(YE)$$

$$\underbrace{YE = E(YE)}$$

- použití pro rekurzivní fce

Práce v λ-kalkulu

① Definuj TRUE, False, NOT, XOR, ?:

LET True = λx y . x

LET False = λx y . y

LET NOT = λv. v False True

LET XOR = λx y . x (NOT y) y

LET ? := λc l f . c l f

XOR

a	b	result
0	0	0
0	1	1
1	0	1
1	1	0

② Definuj (_,_) a fst, snd

LET (_,_) = λf s a . a f s

LET fst = λx . x True

LET snd = λx . x False

③ Definuj reprezentaci čísel

LET 0 = λf n . n

LET 1 = λf n . f n

LET 2 = λf n . f (f n)

⋮

LET succ = λx g m . x g (g m)

LET add = λx y . x succ y

LET sub = λx y . y pred x

LET mul = Y (λf x y . iszero y ? 0 : add x (f x (pred y)))

LET iszero = λx . x (λa . False) True

mul x 0 = 0

mul x y = x + mul x (

Defining pred:

LET predfn = λf x. snd x ? (fst x, False) : (f(fst x), False)

LET pred = λX g n. fst (X (predfn g) ((n, True)))

42. Lambda kalkul (definice všech pojmů, operací...).

Odkaz na video: SZZ : Lambda kalkul - úvod

Video pokrývá témata: *Lambda kalkul. Lambda výraz. Volné a vázané proměnné. Substituce. Alfa konverze. Beta redukce. Eta redukce. Relace identity, relace rovnosti, relace šipky. Přejmenování lambda výrazů. Normální forma lambda výrazu. Posloupnost redukcí v normálním pořadí. Vztah Turingových strojů a lambda kalkulu. Nerozhodnutelné problémy lambda kalkulu.*

Základní pojmy

Lambda kalkul (λ-kalkul) je výpočetně úplný přepisovací systém. Je formální bází některých funkcionálních jazyků, jako je Haskell.

Mějme libovolnou spočetnou množinu proměnných X. Pomocí ní rekurzivně definujeme množinu všech lambda výrazů Λ nad množinou X následovně:

- $x \in X \Rightarrow x \in \Lambda$
  - tedy samotná **proměnná** je lambda výrazem
- $A, B \in \Lambda \Rightarrow (A\ B) \in \Lambda$
  - lambda výrazu (A B) budeme říkat **aplikace**
  - lambda výraz *A* je aplikovaný na lambda výraz *B*
- $x \in X \wedge A \in \Lambda \Rightarrow (\lambda\ x\ .\ A) \in \Lambda$
  - lambda výrazu (λ x . A) budeme říkat **abstrakce**
  - v lambda abstrakci (λ x . A) část *x* nazveme hlavičkou lambda abstrakce, část *A* tělem lambda abstrakce

Každý lambda výraz nad množinou proměnných X je tedy textový řetězec, který ze skládá z elementů množiny X, z kulatých závorek, teček, mezer a symbolů λ. Řetězec takových symbolů je lambda výrazem tehdy, pokud je možné jej zkonstruovat pomocí výše uvedených pravidel.

Pro množinu proměnných X = {a, b, c} můžeme zkonstruovat například následující lambda výrazy:

- a
- b
- c
- (a b)
- ((a b) b)

- $(\lambda\ c\ .\ c)$
- $(\lambda\ c\ .\ ((a\ b)\ c))$
- $(\lambda\ c\ .\ (((a\ b)\ c)\ b))$
- $(\lambda\ c\ .\ (((a\ b)\ c)\ b))\ ((a\ b)\ b)$
- $(\lambda\ a\ .\ (\lambda\ c\ .\ (((a\ b)\ c)\ b)))\ ((a\ b)\ b)$
- $(\lambda\ b\ .\ (\lambda\ a\ .\ (\lambda\ c\ .\ (((a\ b)\ c)\ b))))\ ((a\ b)\ b)$
- ...

Konvence. Vzhledem k neúnosnému množství závorek zavádíme několik konvencí

- **uzávorkování vícenásobné aplikace**
  - opakovanou aplikaci $\mathbf{(\ldots\ (((A_1\ A_2)\ A_3)\ A_4)\ \ldots\ A_n)}$ lze zjednodušeně zapsat bez závorek jako $\mathbf{A_1\ A_2\ A_3\ \ldots\ A_n}$
  - jednotlivé aplikace musí být původně uzávorkované zleva tak, jak je naznačeno výše
  - neplatí tedy například $((A_1\ A_2)\ (A_3\ A_4)) = A_1\ A_2\ A_3\ A_4$, neboť $A_1\ A_2\ A_3\ A_4 = (((A_1\ A_2)\ A_3)\ A_4)$
- **odstranění vnějších závorek na nejvyšší úrovni lambda výrazu**
  - místo $\mathbf{(A_1\ A_2)}$ lze psát $\mathbf{A_1\ A_2}$
  - místo $\mathbf{(\lambda\ x\ .\ (A_1\ A_2\ A_3\ \ldots\ A_n))}$ lze psát $\mathbf{\lambda\ x\ .\ A_1\ A_2\ A_3\ \ldots\ A_n}$, tedy uzávorkované nemusí být ani tělo lambda abstrakce, pokud jde o lambda abstrakci na nejvyšší úrovni
- **zjednodušený zápis vícenásobné abstrakce**
  - místo $\mathbf{\lambda\ x_1\ .\ (\lambda\ x_2\ .\ (\lambda\ x_3\ .\ (\ \ldots\ (\lambda\ x_n\ .\ E)\ \ldots\ )))}$ lze psát $\mathbf{(\lambda\ x_1\ x_2\ x_3\ \ldots\ x_n\ .\ E)}$, tedy takto zjednodušeně vyjáříme lambda abstrakci s vyšším množstvím proměnných

Ať E je nějaký lambda výraz nad množinou proměnných X. Ať $x \in X$ je proměnná, která se v tomto lambda výrazu *E* vyskytuje. Proměnnou *x* v tomto jejím konkrétním výskytu označíme za **vázanou proměnnou**, pokud se nachází v těle nějaké lambda abstrakce, v jejíž hlavičce se rovněž vyskytuje. Jinak ji označíme za **volnou proměnnou**. Pokud je proměnná vázaná, pak se váže k nejbližší hlavičce, v níž se vyskytuje.

V jednom lambda výrazu se může tato proměnná $x \in X$ vyskytovat vícekrát, v rámci různých výskytů může být volnou proměnnou, jinde vázanou.

- **x**
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **volnou** proměnnou
- $\lambda\ x\ .\ \mathbf{x}$
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou
- $\lambda\ x\ .\ \mathbf{x\ y}$
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou, zatímco červeně vyznačená proměnná **y** je **volnou** proměnnou
- $(\lambda\ x\ .\ \mathbf{x})\ \mathbf{x}$
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou, zatímco červeně vyznačená proměnná **x** je **volnou** proměnnou
- $(\lambda\ x\ .\ \mathbf{x}\ (\lambda\ x\ .\ \mathbf{x})\ \mathbf{x})\ x$

- v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou, přičemž je vázaná k zelenému výskytu proměnné **x** v hlavičce
- v tomto lambda výrazu je červeně vyznačená proměnná **x** **vázanou** proměnnou, přičemž je vázaná k žlutému výskytu proměnné **x** v hlavičce

Mějme libovolnou spočetnou množinu proměnných X. Ať dále Λ je množina všech lambda výrazů nad množinou proměnných X. Definujeme zobrazení $\mathbf{FV} : \Lambda \to 2^X$, které zobrazuje lambda výraz na množinu volných proměnných v tomto lambda výrazu, takto:

- **proměnná $x \in X$**
  - $FV(x) = \{x\}$
  - v lambda výrazu *x* je jediná volná proměnná, kterou je právě *x*
- **aplikace (A B), $A, B \in \Lambda$**
  - $FV(A\ B) = FV(A) \cup FV(B)$
  - funkci počítající volné proměnné propagujeme dovnitř aplikace, postupně na obě složky aplikace
- **abstrakce (λ x . A), $x \in X, A \in \Lambda$**
  - $FV(\lambda x . A) = FV(A) \setminus \{x\}$
  - funkci počítající volné proměnné propagujeme dovnitř těla abstrakce, ovšem z výsledku odstraníme proměnnou x, která zde není volná

Ať A, B jsou lambda výrazy a ať x je proměnná. **Substitucí** budeme rozumět operaci zapsanou jako **A[B/x]**, která v lambda výrazu A nahradí všechny volné výskyty proměnné x za lambda výraz B. Provedená substituce musí být **platná**, tedy žádná volná proměnná v lambda výrazu B se nesmí stát vázanou proměnnou v lambda výrazu A[B/x].

Při provádění substituce může dojít k následujícím případům:

- **proměnná $x \in X$**
  - **substituce x[A/x]**
    - x[A/x] = A
    - v lambda výrazu *x* je *x* volná proměnná, tedy ji nahradíme za lambda výraz *A*
  - **substituce x[A/z]**
    - x[A/z] = x
    - v lambda výrazu *x* není žádná volná proměnná *z*, tedy efektivně k ničemu nedojde, substituce je ovšem platná
- **aplikace (A B), $A, B \in \Lambda$**
  - **substituce (A B)[C/x]**
    - (A B)[C/x] = (A[C/x] B[C/x])
    - substituci propagujeme dovnitř lambda aplikace
- **abstrakce (λ x . A), $x \in X, A \in \Lambda$**
  - **substituce (λ x . A)[B/x]**

        - $(\lambda x . A)[B/x] = (\lambda x . A)$
        - v lambda výrazu $(\lambda x . A)$ nikdy neexistuje volný výskyt proměnné *x*, neboť *x* je v hlavičce této lambda abstrakce, tedy se efektivně nic nestane, ačkoliv je substituce platná
    - **substituce $(\lambda x . A)[B/y]$**
        - $(\lambda x . A)[B/y] = (\lambda x . A[B/y])$
        - musí ovšem platit, že $x \neq y$ a navíc **x není volné v B**
        - substituce se propaguje dovnitř těla lambda abstrakce, ovšem musí být platná

To si demonstrujeme na příkladech:

- $(x\ x)[y/x] = x[y/x]\ x[y/x] =$ **y y**
    - v lambda výrazu **x x** jsme nahradili volné výskyty proměnné **x** za **y**, tedy výsledkem je lambda výraz **y y**
- $x[y/z] =$ **x**
    - v lambda výrazu **x** jsme nahradili volné výskyty proměnné **z** za **y**, tedy výsledkem je lambda výraz **x**, efektivně k ničemu nedošlo
- $(\lambda x . x)[y/x] =$ **$(\lambda x . x)$**
    - x je v rámci této lambda abstrakce vázanou proměnnou, efektivně k ničemu nedošlo
- $(\lambda x . y)[(z\ z)/y] =$ **$\lambda x . (z\ z)$**
    - nahradili jsme **y** za **(z z)**, neboť **y** je zde volná proměnná
- **$(\lambda x . y)[(x\ x)/y]$**
    - **nelze provést**, protože volné proměnné **x** v lambda výrazu **(x x)** by se staly vázanými proměnnými a substituce by **nebyla platná**
- $(\lambda x . y\ (\lambda z . z\ z)\ x\ y)[(\lambda z . z\ z)/y] =$ **$\lambda x . (\lambda z . z\ z)\ (\lambda z . z\ z)\ x\ (\lambda z . z\ z)$**

Ať **$\lambda x . A$** je lambda výraz nad množinou proměnných X ve tvaru abstrakce a ať $y \in X$ je proměnná. **Alfa konverzí** (α-konverzí) rozumíme operaci přejmenování proměnné v hlavičce lambda abstrakce, kterou zapisujeme následovně: $\lambda x . A \rightarrow_\alpha \lambda y . A[y/x]$, tedy:

- **zaměníme proměnnou v hlavičce lambda abstrakce**
- **provedeme substituci A[y/x] v těle abstrakce**
    - veškeré volné výskyty proměnné **x** ve výrazu **A** zaměníme za **y**
    - musí jít o platnou substituci

Lambda výraz, na nějž je možné aplikovat alfa konverzi, nazveme **alfa redex**.

Provedení alfa konverze ukažme na příkladech:

- $\lambda x . x \rightarrow_\alpha \lambda y . y$
- $\lambda x . (\lambda y . y)\ x \rightarrow_\alpha \lambda y . (\lambda y . y)\ y$
- **$\lambda x . y\ x \rightarrow_\alpha \lambda y . y\ y$**
    - **nelze provést**, volná proměnná **y** v těle lambda abstrakce by se stala vázanou a substituce by **nebyla platná**

- $\lambda x . y x y y \rightarrow_\alpha \lambda x . z x z z$
  - **nelze provést**, alfa konverze slouží výhradně pro přejmenování proměnné v hlavičce lambda abstrakce, nikoliv k přejmenování volných proměnných
- $x \rightarrow_\alpha y$
  - **nelze provést**, toto není alfa konverze, ale obyčejná substituce x[y/x] = y
- $\lambda x . (\lambda y . x) \rightarrow_\alpha \lambda y . (\lambda y . y)$
  - **nelze provést**, neboť *x* bylo původně vázano na vnější hlavičku a po konverzi bylo zachyceno vnitřní hlavičkou
- $\lambda x . (\lambda y . y x) x \rightarrow_\alpha \lambda z . (\lambda y . y z) z \rightarrow_\alpha \lambda z . (\lambda x . x z) z \rightarrow_\alpha \lambda y . (\lambda x . x y) y$
  - takto jsme elegantně prohodili názvy proměnných **x** a **y** prostřednictvím třetí pomocné proměnné, aniž bychom provedli neplatnou substituci

Ať **(λ x . A) B** je lambda výraz ve tvaru aplikace, kde první ze složek aplikace je lambda abstrakcí. **Beta redukcí** (β-redukcí) rozumíme operaci aplikace lambda výrazu **B** na abstrakci, kterou lze chápat jako zavolání funkce pro nějaký argument. Zapisujeme ji následovně: **(λ x . A)** $B \rightarrow_\beta$ **A[B/x]**, tedy:

- **spotřebujeme proměnnou v hlavičce lambda abstrakce**
- **v těle lambda abstrakce provedeme substituci A[B/x]**
  - veškeré volné výskyty proměnné **x** ve výrazu **A** zaměníme za **B**
  - musí jít o platnou substituci

Lambda výraz, na nějž je možné aplikovat beta redukci, nazveme **beta redex**.

Provedení beta redukce ukažme na příkladech:

- $(\lambda x . x) y \rightarrow_\beta y$
  - spotřebovali jsme **x** v hlavičce lambda abstrakce a volné výskyty **x** v těle lambda abstrakce jsme nahradili za **y**
- $(\lambda x . x (\lambda x . x)) y \rightarrow_\beta y (\lambda x . x)$
  - stejný případ, všimněme si ale, že jsme nahradili pouze volné výskyty **x**
- $(\lambda x . x x x) (\lambda x . x x x) \rightarrow_\beta (\lambda x . x x x) (\lambda x . x x x) (\lambda x . x x x)$
- $(\lambda x y z . x z y x) (\lambda x . x x) \rightarrow_\beta \lambda y z . (\lambda x . x x) z y (\lambda x . x x)$
- $(\lambda x y . x y y) (y y) \rightarrow_\beta \lambda y . (y y) y y$
  - **nelze provést**, neboť volná proměnná **y** se stala vázanou
- $(\lambda x y . x y y) (y y) \rightarrow_\alpha (\lambda x z . x z z) (y y) \rightarrow_\beta \lambda z . (y y) z z$
  - pomohli jsme si alfa konverzí, abychom mohli provést beta redukci

Ať **λ x . A x** je lambda výraz ve tvaru abstrakce, jejíž tělo je aplikací, kde na posledním místě stojí právě proměnná daná hlavičkou dané lambda abstrakce. **Eta redukcí** (η-redukcí)

rozumíme operaci, kterou lze chápat jako odstranění přebytečného argumentu lambda abstrakce. Zapisujeme ji následovně:

$\lambda x . A x \rightarrow_\eta A$, tedy:

- **odstraníme hlavičku lambda abstrakce i odpovídající proměnnou z konce těla lambda abstrakce**
- **x nesmí být volná proměnná v samotném lambda výrazu A**
- jde o speciální operaci pro zjednodušení lambda výrazu, kterou lze skutečně použít jen v případě, kdy se potýkáme s lambda výrazem ve tvaru $\lambda x . A x$
- intuice je taková, že pokud bychom se potýkali s aplikací $(\lambda x . A x) B$, pomocí beta redukce bychom tak jako tak obdrželi $(\lambda x . A x) B \rightarrow_\beta A B$, tedy nemá význam proměnnou **x** použít, ponecháme pouze **A**, rovnou lze vytvářet aplikace typu **A B**

Lambda výraz, na nějž je možné aplikovat eta redukci, nazveme **eta redex**.

Provedení eta redukce ukažme na příkladech:

- $\lambda x . y y x \rightarrow_\eta y y$
- $\lambda x . (\lambda y . (\lambda z . z) y) x \rightarrow_\eta \lambda y . (\lambda z . z) y \rightarrow_\eta \lambda z . z$
  - u lambda výrazu $(\lambda z . z)$ nelze eta redukci provést, ačkoliv by se mohlo zdát, že ano, ovšem nemáme definováno nic jako prázdný lambda výraz, který by byl výsledkem
- $\lambda x . x y y \rightarrow_\eta y y$
  - **nelze provést**, proměnná **x** musí být na konci těla lambda abstrakce
- $\lambda x . y (y x) \rightarrow_\eta y y$
  - **nelze provést,** zde je na konci těla aplikace **(y x)**, nikoliv **x**
- $\lambda z . z y z z \rightarrow_\eta z y z$
  - **nelze provést,** proměnná **z** je volná v lambda výrazu **z y z z**, z vázané proměnné se stala volná proměnná
- $\lambda x y z . z y x \rightarrow_\eta \lambda y z . z y$
  - **nelze provést**, protože **λ x y z . z y x** je zkratka pro lambda výraz **λ x . (λ y . (λ z . z y x))**, tedy tělo této lambda abstrakce nemá tvar nějaké aplikace, kde poslední element je proměnná **x**, tělo této lambda abstrakce je jiná lambda abstrakce
- $\lambda x y z . x y z \rightarrow_\eta \lambda x y . x y \rightarrow_\eta \lambda x . x$
  - lze provést, neboť $\lambda x y z . x y z = \lambda x . (\lambda y . (\lambda z . x y z))$, takže zde provádíme η redukce zevnitř

Ať A, B jsou dva lambda výrazy. Tyto lambda výrazy mohou být v relaci

- **identity ≡**
  - **A, B** jsou dva zcela identické textové řetězce (čistě na syntaktické úrovni)
  - $\lambda x . x y \equiv \lambda x . x y$
- **rovnosti =**

- lambda výrazy A, B se rovnají (jsou v relaci =) tehdy, pokud existují lambda výrazy E_0, E_1, E_2, …, E_{n-1}, E_n takové, že $A \equiv E_1$, $B \equiv E_n$ a dále $\forall 0 < k <= n$: $E_{k-1} \sim_k E_k$, kde $\sim_k$ je vždy některá z následujících relací:
  - $\rightarrow_\alpha, \rightarrow_\beta, \rightarrow_\eta, \leftarrow_\alpha, \leftarrow_\beta, \leftarrow_\eta$
- tedy postupným prováděním alfa, beta, eta redukcí v různých směrech je možné dospět z lambda výrazu A k lambda výrazu B
- $(\lambda\ y\ x\ .\ y\ y\ x)\ (\lambda\ w\ .\ w) = (\lambda\ y\ z\ .\ (\lambda\ x\ .\ x)\ z)\ (\lambda\ x\ .\ x\ x)$, neboť
  - $(\lambda\ y\ x\ .\ y\ y\ x)\ (\lambda\ w\ .\ w) \rightarrow_\beta \lambda\ x\ .\ (\lambda\ w\ .\ w)\ (\lambda\ w\ .\ w)\ x \rightarrow_\eta (\lambda\ w\ .\ w)\ (\lambda\ w\ .\ w) \rightarrow_\beta$ **λ w . w**
  - **λ w . w** $\leftarrow_\alpha \lambda\ x\ .\ x \leftarrow_\eta \lambda\ z\ .\ (\lambda\ x\ .\ x)\ z \leftarrow_\beta (\lambda\ y\ z\ .\ (\lambda\ x\ .\ x)\ z)\ (\lambda\ x\ .\ x\ x)$

- →
  - lambda výrazy A, B jsou v relaci → tehdy, pokud existují lambda výrazy E_0, E_1, E_2, …, E_{n-1}, E_n takové, že $A \equiv E_1$, $B \equiv E_n$ a dále $\forall 1 <= k <= n$: $E_{k-1} \sim_k E_k$, kde $\sim_k$ je vždy jedna z následujících relací:
    - $\rightarrow_\alpha, \rightarrow_\beta, \rightarrow_\eta$
  - podobný případ jako relace =, jen s tím rozdílem, že povolujeme pouze jednotlivé konverze směrem doprava
  - $(\lambda\ y\ x\ .\ y\ y\ x)\ (\lambda\ w\ .\ w) \rightarrow \lambda\ w\ .\ w$, neboť
    - $(\lambda\ y\ x\ .\ y\ y\ x)\ (\lambda\ w\ .\ w) \rightarrow_\beta \lambda\ x\ .\ (\lambda\ w\ .\ w)\ (\lambda\ w\ .\ w)\ x \rightarrow_\eta (\lambda\ w\ .\ w)\ (\lambda\ w\ .\ w) \rightarrow_\beta$ **λ w . w**

Nad rámec definice lambda kalkulu povolujeme pojmenovávání lambda výrazů pro jejich snazší použití. Toho docílíme pomocí zápisu **LET N = E**, kde **N** je nově zavedené jméno lambda výrazu **E**.

Pozn.: Definice pravdivostních hodnot a logických spojek jsou součástí další otázky, zde jde jen o demonstraci použití klíčového slova LET.

Příklad.

- LET TRUE = (λ x y . x y)
- LET FALSE = (λ x y . y x)
- LET OR = (λ a b . a (λ s . (λ x y . x y)) (λ t . b))

Pak lze použít:

- OR TRUE FALSE = TRUE
  - místo (λ a b . a (λ s . (λ x y . x y)) (λ t . b)) (λ x y . x y) (λ x y . y x) = (λ x y . x y)
- OR FALSE FALSE = FALSE
  - místo (λ a b . a (λ s . (λ x y . x y)) (λ t . b)) (λ x y . y x) (λ x y . y x) = (λ x y . y x)

Jde pouze o textovou definici, ne o rovnost, definované jméno se **nesmí** vyskytnout v rámci definice (a to ani nepřímo skrze jiné definice). Nelze tedy napsat nic jako:

- LET A = λ x . A x x
  - **nelze,** není povoleno
- LET B = λ x y . C y y, LET C = λ x y . x x B
  - **nelze,** není povoleno

Ať **E** je lambda výraz. Řekneme, že lambda výraz E je v **normální formě**, pokud neobsahuje žádné β redexy ani η redexy, tedy pokud není možné v něm provést žádnou β redukci ani η redukci.

- výraz **λ x y . x x** je v normální formě
- výraz **(λ x y . x x) (λ z . z)** není v normální formě
  - lze provést β redukci
- výraz **λ x y . x y** není v normální formě
  - lze provést η redukci

Ať **E** je lambda výraz. Provádíme-li v něm postupně nejlevější a nejvnějšnější β redukce a η redukce, pak provádíme **redukce v normálním pořadí**. Pokud je možné posloupností β redukcí a η redukcí transformovat E do normální formy, pak je možné toto provést posloupností redukcí v normálním pořadí:

- (λ x y . x x) (λ z . z x) (λ z . x) $\rightarrow_\beta$ (λ y . (λ z . z x) (λ z . z x)) (λ z . x) $\rightarrow_\beta$ (λ z . z x) (λ z . z x) $\rightarrow_\beta$ (λ z . z x) x $\rightarrow_\beta$ x x
  - provádění redukcí **v normálním pořadí**
  - došli jsme k normální formě
- (λ x y . x x) (λ z . z x) (λ z . x) $\rightarrow_\beta$ (λ x y . x x) ((λ z . x) x) $\rightarrow_\beta$ (λ x y . x x) x $\rightarrow_\beta$ λ y . x x
  - provádění redukcí **v jiném než v normálním pořadí**
  - také jsme došli k normální formě, ale k jinému lambda výrazu

Typicky používáme pouze redukce v normálním pořadí.

Lambda kalkul je model s výpočetní silou ekvivalentní Turingovým strojům. Turingovy stroje mohou vyhodnocovat libovolné lambda výrazy a pomocí lambda výrazů lze simulovat běh libovolného Turingova stroje. Z toho plynou následující dvě vlastnosti:

- **lambda kalkul má vysokou vyjadřovací sílu**
  - vše, co lze algoritmicky řešit, je možné řešit pomocí lambda kalkulu
  - pomocí lambda kalkulu je možné definovat obvykle používané struktury, jde tedy o dostatečně silnou formální bázi pro některé funkcionální jazyky
- **pohybujeme se na hranicích rozhodnutelnosti**
  - mnohé jednoduše definovatelné rozhodovací problémy týkající se lambda kalkulu jsou nerozhodnutelné
  - problém, zda daný lambda výraz má **normální formu**, je **nerozhodnutelný**
    - mnohé lambda výrazy nemají normální formu, provádění β redukcí nikam nevede, například (λ x . x x x) (λ x . x x x) $\rightarrow_\beta$ (λ x . x x x) (λ x . x x x) (λ x . x x x) $\rightarrow_\beta$ ...

        - při vyhodnocování β redexů tedy lze "cyklit", podobně jako může cyklit běh Turingova stroje
        - obecně není rozhodnutelné, zda nějaká konečná posloupnost β redukcí povede k normální formě
    - problém, zda jsou dva lambda výrazy **β-ekvivalentní**, je **nerozhodnutelný**
        - ať A, B jsou dva lambda výrazy
        - obecně je nerozhodnutelné, zda existují lambda výrazy E_0, E_1, E_2, …, E_{n-1}, E_n takové, že $A \equiv E_1$, $B \equiv E_n$ a dále $\forall 1 <= k <= n$: $E_{k-1} \sim_k E_k$, kde $\sim_k$ je vždy jedna z následujících relací:
            - $\rightarrow_\beta, \leftarrow_\beta$
    - …

Pokud v textu najdete chybu, nebudete něčemu rozumět nebo budete mít dojem, že by bylo vhodné něco doplnit, kontaktujte na discordu uživatele kocotom.