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)

- (λ c . c)
- (λ c . ((a b) c))
- (λ c . (((a b) c) b))
- (λ c . (((a b) c) b)) ((a b) b)
- (λ a . (λ c . (((a b) c) b))) ((a b) b)
- (λ b . (λ a . (λ c . (((a b) c) b)))) ((a b) b)
- …

Konvence. Vzhledem k neúnosnému množství závorek zavádíme několik konvencí

- **uzávorkování vícenásobné aplikace**
  - opakovanou aplikaci **(... (((A$_1$ A$_2$) A$_3$) A$_4$) ... A$_n$)** lze zjednodušeně zapsat bez závorek jako **A$_1$ A$_2$ A$_3$ … 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 **(A$_1$ A$_2$)** lze psát **A$_1$ A$_2$**
  - místo **(λ x . (A$_1$ A$_2$ A$_3$ … A$_n$))** lze psát **λ x . A$_1$ A$_2$ A$_3$ … A$_n$**, tedy uzávorkované nemusí být ani tělo lambda abstrakce, pokud jde o lambda abstrakci na nejvyšší úrovni
- **zjednodušený zápis vícenásobné abstrakce**
  - místo **λ x$_1$. (λ x$_2$. (λ x$_3$. (… (λ x$_n$. E) …)))** lze psát **(λ x$_1$ x$_2$ x$_3$ … 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 ∈ 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 ∈ 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
- λ x . **x**
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou
- λ x . **x** **y**
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou, zatímco červeně vyznačená proměnná **y** je **volnou** proměnnou
- (λ x . **x**) **x**
  - v tomto lambda výrazu je modře vyznačená proměnná **x** **vázanou** proměnnou, zatímco červeně vyznačená proměnná **x** je **volnou** proměnnou
- (λ **x** . **x** (λ **x** . **x**) **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 **FV : Λ → 2^X**, které zobrazuje lambda výraz na množinu volných proměnných v tomto lambda výrazu, takto:

- **proměnná x ∈ X**
    - FV(x) = {x}
    - v lambda výrazu *x* je jediná volná proměnná, kterou je právě *x*
- **aplikace (A B), A, B ∈ Λ**
    - FV(A B) = FV(A) ∪ FV(B)
    - funkci počítající volné proměnné propagujeme dovnitř aplikace, postupně na obě složky aplikace
- **abstrakce (λ x . A), x ∈ X, A ∈ Λ**
    - FV(λ x . A) = FV(A) \ {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 ∈ 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 ∈ Λ**
    - **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 ∈ X, A ∈ Λ**
    - **substituce (λ x . A)[B/x]**

        - (λ x . A)[B/x] = (λ x . A)
        - v lambda výrazu (λ 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 (λ x . A)[B/y]**
        - (λ x . A)[B/y] = (λ x . A[B/y])
        - musí ovšem platit, že **x ≠ 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
- (λ x . x)[y/x] = **(λ x . x)**
    - x je v rámci této lambda abstrakce vázanou proměnnou, efektivně k ničemu nedošlo
- (λ x . y)[(z z)/y] = **λ x . (z z)**
    - nahradili jsme **y** za **(z z)**, neboť **y** je zde volná proměnná
- **(λ 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**
- (λ x . y (λ z . z z) x y)[(λ z . z z)/y] = **λ x . (λ z . z z) (λ z . z z) x (λ z . z z)**

Ať **λ x . A** je lambda výraz nad množinou proměnných X ve tvaru abstrakce a ať y ∈ X je proměnná. **Alfa konverzí** (α-konverzí) rozumíme operaci přejmenování proměnné v hlavičce lambda abstrakce, kterou zapisujeme následovně: **λ x . A →$_\alpha$ λ 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:

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

- **λ x . y x y y →$_\alpha$ λ x . z x z z**
    - **nelze provést**, alfa konverze slouží výhradně pro přejmenování proměnné v hlavičce lambda abstrakce, nikoliv k přejmenování volných proměnných
- **x →$_\alpha$ y**
    - **nelze provést**, toto není alfa konverze, ale obyčejná substituce x[y/x] = y
- **λ x . (λ y . x) →$_\alpha$ λ y . (λ y . y)**
    - **nelze provést**, neboť *x* bylo původně vázano na vnější hlavičku a po konverzi bylo zachyceno vnitřní hlavičkou
- λ x . (λ y . y x) x →$_\alpha$ λ z . (λ y . y z) z →$_\alpha$ λ z . (λ x . x z) z →$_\alpha$ λ y . (λ 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 →$_\beta$ 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:

- (λ x . x) y **→$_\beta$** y
    - spotřebovali jsme **x** v hlavičce lambda abstrakce a volné výskyty **x** v těle lambda abstrakce jsme nahradili za **y**
- (λ x . x (λ x . x)) y **→$_\beta$** y (λ x . x)
    - stejný případ, všimněme si ale, že jsme nahradili pouze volné výskyty **x**
- (λ x . x x x) (λ x . x x x) **→$_\beta$** (λ x . x x x) (λ x . x x x) (λ x . x x x)
- (λ x y z . x z y x) (λ x . x x) **→$_\beta$** λ y z . (λ x . x x) z y (λ x . x x)
- **(λ x y . x y y) (y y) →$_\beta$ λ y . (y y) y y**
    - **nelze provést**, neboť volná proměnná **y** se stala vázanou
- (λ x y . x y y) (y y) →$_\alpha$ (λ x z . x z z) (y y) →$_\beta$ λ 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:

λ x . A x →$_\eta$ 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 **λ x . A x**
- intuice je taková, že pokud bychom se potýkali s aplikací **(λ x . A x) B**, pomocí beta redukce bychom tak jako tak obdrželi **(λ x . A x) B →β 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:

- λ x . y y x →η y y
- λ x . (λ y . (λ z . z) y) x →η λ y . (λ z . z) y →η λ z . z
  - u lambda výrazu (λ 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
- **λ x . x y y →η y y**
  - **nelze provést**, proměnná **x** musí být na konci těla lambda abstrakce
- **λ x . y (y x) →η y y**
  - **nelze provést,** zde je na konci těla aplikace **(y x)**, nikoliv **x**
- **λ z . z y z z →η z y z**
  - **nelze provést,** proměnná **z** je volná v lambda výrazu **z y z z**, z vázané proměnné se stala volná proměnná
- **λ x y z . z y x →η λ 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
- λ x y z . x y z →η λ x y . x y →η λ x . x
  - lze provést, neboť λ x y z . x y z = λ x . (λ y . (λ 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)
  - λ x . x y **≡** λ 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 ≡ E_1, B ≡ E_n a dále ∀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
    - (λ y x . y y x) (λ w . w) = (λ y z . (λ x . x) z) (λ x . x x), neboť
        - (λ y x . y y x) (λ w . w) $\rightarrow_\beta$ λ x . (λ w . w) (λ w . w) x $\rightarrow_\eta$ (λ w . w) (λ w . w) $\rightarrow_\beta$ **λ w . w**
        - **λ w . w** $\leftarrow_\alpha$ λ x . x $\leftarrow_\eta$ λ z . (λ x . x) z $\leftarrow_\beta$ (λ y z . (λ x . x) z) (λ 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 ≡ E_1, B ≡ E_n a dále ∀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
    - (λ y x . y y x) (λ w . w) → λ w . w, neboť
        - (λ y x . y y x) (λ w . w) $\rightarrow_\beta$ λ x . (λ w . w) (λ w . w) x $\rightarrow_\eta$ (λ w . w) (λ 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:

- $(\lambda x\ y\ .\ x\ x)\ (\lambda z\ .\ z\ x)\ (\lambda z\ .\ x) \rightarrow_\beta\ (\lambda y\ .\ (\lambda z\ .\ z\ x)\ (\lambda z\ .\ z\ x))\ (\lambda z\ .\ x) \rightarrow_\beta (\lambda z\ .\ z\ x)\ (\lambda z\ .\ z\ x) \rightarrow_\beta (\lambda z\ .\ z\ x)\ x \rightarrow_\beta x\ x$
    - provádění redukcí **v normálním pořadí**
    - došli jsme k normální formě
- $(\lambda x\ y\ .\ x\ x)\ (\lambda z\ .\ z\ x)\ (\lambda z\ .\ x) \rightarrow_\beta (\lambda x\ y\ .\ x\ x)\ ((\lambda z\ .\ x)\ x) \rightarrow_\beta (\lambda x\ y\ .\ x\ x)\ x \rightarrow_\beta \lambda y\ .\ x\ x$
    - 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 $(\lambda x\ .\ x\ x\ x)\ (\lambda x\ .\ x\ x\ x) \rightarrow_\beta (\lambda x\ .\ x\ x\ x)\ (\lambda x\ .\ x\ x\ x)\ (\lambda x\ .\ x\ x\ x) \rightarrow_\beta \ldots$
        - 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.