26. LAMBDA KALKUL λ

- je formální systém a výpočetní model který se používá v teoretické informatice a matematice pro studium funkcí a rekurze
- teoretický základ pro funkcionální programování a funkcionální programovací jazyky
- je možné jej chápat i jako jednoduchý univerzální programovací jazyk
- každou rekurzivně spočetnou funkci lze vyjádřit pomocí λ-kalkulu

⇒ silou ekvivalentní Turingovým strojům a parciálně rekurzivním funkcím !

(vyjadřovací)

- každý výraz je funkcí jednoho parametru, pokud má mít funkce více parametrů pak se to zapíše takto λa λb λc . abc nebo zjednodušeně λabc . abc ale úplně formálně to asi není funkce o třech parametrech ale tři vnořené funkce o ~~třech~~ jednom parametru
- výrazy můžeme ukládat i do nějakých proměnných, respektive to nejsou proměnné ale pojmenované výrazy LET K = λx.xy

3 typy výrazů !

- zjednodušeně se všechny mohou označit jako E

1) Proměnné – klasické proměnné jak všude jinde a, b, c, x, y, R, ... $E \to X$ gramatika

2) Abstrakce – v podstatě to je definice funkce včetně hlavičky i těla

$\lambda xy . x+y+z$ → tělo / definice funkce (λxy = hlavička funkce a zároveň vstupní parametry funkce)

$E \to \lambda V.E$ gramatika

- Volné a vázané proměnné – proměnné jsou vázané jestliže jsou parametrem funkce – vyskytují se jak v těle (definici) tak v hlavičce
  - ostatní jsou volné
  - zde x a y vázané a z volná

3) Aplikace – v podstatě aplikace funkce nebo také volání funkce s nějakým argumentem
- můžeme volat s více argumenty postupně ale jak víme tak každá funkce přijímá jen jeden, takže jestli jich přijímá více tak je to více vnořených funkcí a argumenty se postupně předávají každé z nich

$(\lambda xy . x+y+2)\ 3\ 4$ – tak je to aplikace
- potom se pomocí β-konverze provede substituce argumentů 3 a 4 za proměnné x a y

Identita a rovnost λ-výrazů

≡ – identické jsou jestliže se shodují přesně jejich zápis strcmp(A,B) == 0

= – rovnost platí jestliže je možné pomocí redukcí jeden výraz převést na druhý

→ – redukce (simplifikace) $E_1 \to E_2$ – E1 lze přímo zredukovat na E_2

Redukce

– redukují / konvertují se λ-výrazy na jiné – původní výraz a zredukovaný se rovnají (viz předchozí strana)

α-redukce (přejmenování)

(nad [a/x]: a → to se tam dá, x → to tam bylo)

– jedná se o substituci proměnných za jiné, např. a za x $[a/x]$

– substituce musí být však platná, potažmo proveditelná

⟹ z žádné volné proměnné se nesmí stát vázaná

– máme-li tedy výraz E tak substituce $[a/x]$ lze pokud tu x nemá nějaký výskyt v oboru parametru a, a navíc a nemá volný výskyt v tom výrazu

- $\lambda xy.xxy \xrightarrow{\alpha} \lambda ay.aay\ [a/x]$ LZE
- $\lambda xa.x \xrightarrow{\alpha} \lambda aa.a\ [a/x]$ NELZE
- $\lambda x.ax \xrightarrow{\alpha} \lambda a.aa\ [a/x]$ NELZE
- $\lambda x.xy \xrightarrow{\alpha} \lambda x.xa\ [a/y]$ NELZE – jde jen vázané proměnné

β-redukce (dosazení)

– jedná se o substituci (dosazení) argumentu za vstupní parametry funkce

– zase to musí být proveditelné – nesmí se z volné proměnné stát vázaná

- $\lambda xy.xy(xy) \xrightarrow{\beta} \lambda y.xyy\ [(xy)/x]$ NELZE – z volného y se stalo vázané

  – lze to řešit se nejdříve udělá α-redukce

- $\lambda xy.xy(xy) \xrightarrow{\alpha} \lambda xz.xz(xy)\ [z/y] \xrightarrow{\beta} \lambda z.xyz\ [xy/x]$ LZE

  – ty kumulativní závorky se ani psát nemusí za to

η-redukce (odstranění abstrakce)

(η = eta)

– myšlenka ekvivalence dvou funkcí, respektive odstranění abstrakce

$\lambda V.(EV)$ lze převést na E pokud V nemá volný výskyt (V je proměnná v E, E je výraz) – odebere se hlavička i to s tím spojené v těle

- $\lambda x.(uv)x \xrightarrow{\eta} (uv)$ LZE
- $\lambda x.(yx)x \xrightarrow{\eta} (yx)$ NELZE x má volný výskyt v (yx)

Normální forma

– výraz je v normální formě jestliže nad ním již nelze provést žádné β či η redukce ⟹ nejsou tam β či η redexy

– lze provádět pouze α redukce

– každý výraz lze β a η redukcemi (s využitím α redukcí) převést na výraz v normální formě

LOGIKA

LET True = $\lambda xy.x$ – vezme 2 parametry a vrátí 1.
LET False = $\lambda xy.y$ – vezme 2 parametry a vrátí 2.
} nad nimi je definován ten výběr ↓

IF/THEN/ELSE – booleovský operátor:

$\lambda xyz.xyz$ – vezme 1. parametr jako podmínku a když je true tak vrátí y (první) a když false tak z (druhý)

AND:

$\lambda xy.xy$ False – zkrácené vyhodnocení – když je první false tak rovnou vrátím to false (druhé v pořadí) jinak vrátím druhý parametr a ten určí jestli to bude true nebo false ⇒ short evaluation

OR: $\lambda xy.x$ True y – zase short evaluation
– když je první true tak rovnou vrátím true

↳ $(\lambda xy.x\ \text{true}\ y)(\text{True}) \xrightarrow{\beta} (\lambda y.\text{True}\ \text{True}\ y)\overset{(\text{False})}{[\text{True}/x]}$
$\xrightarrow{\beta}$ True True False $[\text{False}/y] \xrightarrow{\text{úprava}} (\lambda xy.x)$ True False
$\xrightarrow{\beta} (\lambda y.\text{True})[\text{False}][\text{True}/x] \xrightarrow{\beta}$ True $[\text{False}/y]$

NOT: $\lambda x.x$ False True

NAND: $\lambda xy.x$ (NOT y) True – když je x false tak víme že NAND je true protože chceme aby to byl negovaný and a když je true tak záleží na y a to musí být false aby výsledek byl true

XOR: $\lambda xy.x$ (NOT y) y – musí se lišit a tak když je x true tak y musí být false a když je x false tak y musí být true

NOR: $\lambda xy.x$ False (Not y) – musí být obojí false aby to byla true

IMPLIKACE: $\lambda xy.x\ y$ True – pokud je x true tak musí být i y true aby to celé platilo
– pokud je x false tak to je celé automaticky true

– nebo taky
$\lambda xy.$ (Not x) True y – jako OR ale s jiným negovaným prvkem

Peanova čísla:

– pro reprezentaci čísel v λ-kalkulu

$0: \lambda fn.n$

$1: \lambda fn.fn$

$2: \lambda fn.f(fn)$

$3: \lambda fn.f(f(fn))$ atd...

– před 0 je 0

sub x y: $\lambda xy.y(\text{prev } x)$ – y krát se udělá předchůdce x

Př.: $(\lambda xy.y(\text{prev } x))(3\,2) \xrightarrow{\beta} 2(\text{prev } 3)$

$\rightarrow (\lambda fn.f(fn))\,\text{prev } 3 \xrightarrow{\beta}$

$(\lambda n.\text{prev}(\text{prev } n))\,3 \xrightarrow{\beta} \text{prev}(\text{prev } 3)$

$\rightarrow \text{prev } 2 \rightarrow \underline{1}$

add x y: $\lambda xy.x(\text{succ } y)$ – x krát se udělá následník y nebo taky y krát se udělá následník x

Př.: $(\lambda xy.x(\text{succ } y))\,3\,2 \xrightarrow{\beta,\beta} 3(\text{succ } 2)$

$\xrightarrow{\text{nápravo}} (\lambda fn.fffn)(\text{succ } 2) \xrightarrow{\beta} (\lambda n.\text{succ}(\text{succ}(\text{succ } n)))\,2$

$\xrightarrow{\beta} \text{succ}(\text{succ}(\text{succ } 2))) \rightarrow \text{succ}(\text{succ } 3) \rightarrow \text{succ } 4 \rightarrow \underline{5}$

succ x: $\lambda xgm.x\,g\,(gm)$

Př.: $(\lambda xgm.x\,g\,(gm))\,2 \xrightarrow{\beta} \lambda gm.2\,g\,(gm)$

$\rightarrow \lambda gm.(\lambda fn.ffn)\,g\,(gm) \xrightarrow{\beta} \lambda gm.\lambda n.ggn\,(gm)$

$\xrightarrow{\beta} \lambda gm.gggm \Leftrightarrow \lambda fn.fffn \Leftrightarrow \underline{3}$

iszero: $\lambda m.m(\lambda v.\text{False})\,\text{True}$

Př.: $(\lambda m.m(\lambda v.\text{False})\,\text{True})\,2 \xrightarrow{\beta} (2(\lambda v.\text{False}))\,\text{True}$

$\rightarrow (\lambda fn.ffn\,(\lambda v.\text{False}))\,\text{True} \rightarrow (\lambda n.(\lambda v.\text{False})(\lambda v.\text{False})n)\,\text{True} \rightarrow (\lambda v.\text{False})(\lambda v.\text{False})\,\text{True}$

$\rightarrow (\lambda v.\text{False})(\lambda v.\text{False}) \rightarrow \underline{\text{False}}$

Rekurze a operátor pevného bodu

- operátor pevného bodu umožňuje mít v λ výrazech rekurzi
- operátor pevného bodu je nějaký výraz Y pro který platí že pro libovolný výraz E:

$$Y\,E = E\,(Y\,E) \qquad E \text{ je } \lambda \text{ výraz}$$

Například:

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

- v λ-kalkulu není klasická rekurze ale šlo udělat

$$\text{LET } a = \lambda x.f\,a$$

a tak se místo toho musí použít operátor pevného bodu