Lambda kalkul
λ-kalkul

Dušan Kolář

Reference

- Češka, M., Motyčková, L., Hruška, T.: Vyčíslitelnost a složitost, 1992, VUT v Brně, ISBN 80-214-0441-8
- Foundations of functional programming, Matthew Parkinson, 2 Lectures (Lent 2009)
- Wikipedia
- Google
- A mnoho dalších zdrojů

Úvod

- Alonso Church
  - 20. léta 20. století
- Haskell Curry
  - V roce 1930 ukázal, že λ-kalkul je ekvivalentní teorii kombinátorů (Moses Schönfinkel, teorie funkcí)
- Kleene
  - 1930, λ-kalkul je univerzální výpočetní systém
- McCarthy
  - 1950, inspirován při tvorbě jazyka LISP

Syntaxe

- Nechť E je výraz λ-kalkulu
  - E := V *proměnná*

    $(E_1\ E_2)$ *aplikace*

    $(\lambda V.\ E)$ *abstrakce*
  - hlavička tělo

Konvence

- $((\dots ((E_1\ E_2)\ E_3) \dots)\ E_n) \sim E_1\ E_2\ E_3 \dots E_n$
- $(\lambda V.(E_1 \dots E_n)) \sim \lambda V.E_1 \dots E_n$
- $(\lambda V_1.(\dots (\lambda V_n . E) \dots)) \sim \lambda V_1 \dots V_n . E$

Volné a vázané proměnné

- λ vázaná . volná vázaná
- λ x . x y (λ x . x) y

Substituce

- E[E‘/V]
  - Volné výskyty V jsou nahrazeny E‘ v E.

- Platnost substituce
  - Žádná volná proměnná v E‘ se nesmí stát vázanou v E[E‘/V]

α - konverze

- $\lambda V . E \to_\alpha \lambda V' . E[V'/V]$
- $\xcancel{\lambda x . x y \to_\alpha \lambda y . y y}$
- $\lambda x . x y \to_\alpha \lambda z . z y$
- $\xcancel{\lambda x . x y \to_\alpha \lambda x . x z}$

β - konverze

- $(\lambda\ V\ .\ E_1)\ E_2\ \rightarrow_\beta\ E_1\,[E_2/V]$
- $\cancel{(\lambda\ x\ y\ .\ x\ y)\ (x\ y)\ \rightarrow_\beta\ \lambda\ y\ .\ (x\ y)\ y}$
  - volná vázaná
- $(\lambda\ x\ y\ .\ x\ y)(x\ y) \rightarrow_\alpha (\lambda\ x\ z\ .\ x\ z)\ (x\ y) \rightarrow_\beta (\lambda\ z\ .\ (x\ y)\ z)$ $\rightarrow_{\text{konvence}}\ \lambda\ z\ .\ x\ y\ z$
- $\cancel{(\lambda\ x\ y\ .\ x\ y)\ (x\ y) \rightarrow_\alpha (\lambda\ x\ z\ .\ x\ z)\ (u\ v) \rightarrow_\beta \lambda\ y\ .\ u\ v\ y}$

η - konverze

- $\lambda\, V\, .\, (E\ V) \;\to_\eta\; E$
  - *V není volné v E*
- $\cancel{\lambda\, x\, .\, (x\ y)\ x \;\to_\eta\; (x\ y)}$
- $\lambda\, x\, .\, (u\ v)\ x \;\to_\eta\; u\ v$
- $\cancel{\lambda\, x\, .\, x\ y \;\leftarrow_\eta\; \lambda\, f\, x\, .\, (x\ y)\ f}$
  - $\lambda\, f\, x\, .\, (x\ y)\ f = \lambda\, f\, .\, (\lambda\, x\, .\, ((x\ y)\ f))$; ale! $\lambda\, f\, .\, (\lambda\, x\, .\, (x\ y))\ f$
- Haskell
  - sqrlist l = map sqr l
  - sqrlist‘ = map sqr

Identita, rovnost, relace $\rightarrow$

- Identita: $E_1 \equiv E_2$
- Rovnost: $E_1 = E_2$
- Relace $\rightarrow$: $E_1 \rightarrow E_2$

- $E_1 \equiv E^0 * E^1 * \ldots * E^n \equiv E_2$
  - Rovnost: $* \sim \rightarrow_\alpha, \rightarrow_\beta, \rightarrow_\eta, \leftarrow_\alpha, \leftarrow_\beta, \leftarrow_\eta$
  - Relace $\rightarrow$: $* \sim \rightarrow_\alpha, \rightarrow_\beta, \rightarrow_\eta$

Pojmenování výrazu

- LET True $= \lambda$ x y. x
  LET False $= \lambda$ x y. y
  LET Not $= \lambda$ t. t False True
  - *Jedná se o textovou zkratku, nikoliv rovnost z matematického pohledu, LET FF = … FF … je tedy nesmysl*
- *Příklad*
- Not True = (λt. t False True) True
  = True False True
  = (λ x y. x) False True
  = (λ y. False) True
  = False

Další logické spojky

- Analýza
  - True: vezmi dva argumenty, vrať první
  - False: vezmi dva argumenty, vrať druhý
- AND – short evaluation
  - Je-li první False, výsledek je False
  - Je-li první True, je výsledek druhý výraz
  - LET AND = λ u v. u v False
- OR
  - LET OR = λ u v. u True v

Domácí úkol 1

- Definujte
  - XOR
  - Implikaci
- Ukažte
  - Pravdivost definice dle pravdivostní tabulky

- Pro zdatné
  - Definujte True a False jiným způsobem (ovšem nikoliv prohozením vlastností True a False) a k tomu logické spojky, ukažte vlastnosti

Peanova čísla/aritmetika

- LET 0 = λ f n. n
- 1 = succ 0
- 2 = succ 1 = succ (succ 0)
- …
- Další funkce: iszero, add, prev, sub, mul, div
- Pomocné fuknce: if-then-else/(? :), pár, …

Mocniny v aplikaci

- LET 0 = λ f n. n
- LET 1 = λ f n. f n
- LET 2 = λ f n. f (f n) = λ f n. f^2 n
- LET 3 = λ f n. f (f (f n)) = λ f n. f^3 n

- LET n = λ f n. f^n n

Následník – succ

- LET succ = λ x g m. x g (g m)

- *Příklad*
- succ 0 = (λ x g m. x g (g m)) 0
  = (λ g m. 0 g (g m))
  = (λ g m. (λ f n. n) g (g m))
  = (λ g m. (λ n. n) (g m))
  = λ g m. g m
- succ 2 = λ g m. 2 g (g m)
  = λ g m. (λ f n. f (f n)) g (g m)
  = λ g m. (λ n. g (g n)) (g m)
  = λ g m. g (g (g m))

Test na 0

- LET iszero = λ m. m (λ v. False) True

- *Příklad*
- iszero 0 = (λ m. m (λ v. False) True) 0
  = 0 (λ v. False) True
  = (λ f n. n) (λ v. False) True
  = True
- iszero 2 = 2 (λ v. False) True
  = (λ f n. f (f n)) (λ v. False) True
  = (λ v. False) ((λ v. False) True)
  = False

Předchůdce – prev

- prev 0 = 0
  prev 1 = 0
  prev 2 = 1
  …

- LET (? :) = λ c t f. c t f
- LET (_,_) = λ f s e. e f s
- LET fst = λ p. p True
- LET snd = λ p. p False

Domácí úkol 2

- Ukažte, že:

  - (True ? 1 : 0) = 1
  - (False ? 0 : 3) = 3
  - fst (1,2) = 1
  - snd (True, False) = False

Definice prev

- LET prefn = λ f p. (fst p ?
  (False, snd p) :
  (False, f (snd p)))

- LET prev = λ x g m. snd (x (prefn g) (True, m))

- prev 0 = (λ x g m. snd (x (prefn g) (True, m))) 0
  = λ g m. snd (0 (prefn g) (True, m))
  = λ g m. snd ((λ f n. n) (prefn g) (True, m))
  = λ g m. snd (True, m)
  = λ g m. m

Užití prev

- prev 2 = (λ x g m. snd (x (prefn g) (True, m))) 2
  = λ g m. snd (2 (prefn g) (True, m))
  = λ g m. snd ((λ f n.f (f n)) (prefn g) (True, m))
  = λ g m. snd (prefn g (prefn g (True, m)))
  = λ g m. snd (prefn g ((λ f p. (fst p ?
  (False, snd p) :
  (False, f (snd p)))) g (True, m)))
  = λ g m. snd (prefn g (False,m))
  = λ g m. snd ((λ p. (fst p ? (False, snd p) :
  (False, g (snd p)))) (False,m))
  = λ g m. snd (False, g m)
  = λ g m. g m

Součet – add

- LET add = λ a b g m. a g (b g m)

- *Příklad*
- add 2 3 $\quad$ = (λ a b g m. a g (b g m)) 2 3
  = λ g m. 2 g (3 g m)
  = λ g m. (λ f n.f(f n)) g ((λ f n.f(f(f n))) g m)
  = λ g m. (λ n. g (g n)) (g (g (g m)))
  = λ g m. g (g (g (g (g m))))

Rozdíl – sub

- LET sub = λ a b. b prev a

- *Příklad*
- sub 3 2 = (λ a b. b prev a) 3 2
  = (λ f n. f (f n)) prev 3
  = prev (prev 3)
  = prev 2
  = 1

Domácí úkol 3

- Ukažte (detailně) správnost
  - iszero (succ 0)
  - sub 2 4
  - iszero (add 0 0)
- Definujte
  - eq tak, že eq bere dvě čísla a vrací True, pokud jsou si rovny, jinak False
  - Ukažte správnou funkci eq

Násobení – mult

- Rozbor:
  - Je-li jedno z čísel 0, potom výsledek je 0
  - Jinak je to, pro čísla *m* a *n*, *m*-krát sečteno číslo *n*
- Tedy, pracovně:
  - LET mult = λ m n. (OR (iszero m) (iszero n) ? 0 :
    multfn m n)
  - LET multfn = λ m n. m (add n) 0
- A vidíme, že
  - multfn 2 3 = (λ m n. m (add n) 0) 2 3 = 2 (add 3) 0
    = (λ f n. f (f n)) (add 3) 0 = (add 3) ((add 3) 0)
    = add 3 (add 3 0) = add 3 3 = 6
  - multfn 0 2 = (λ m n. m (add n) 0) 0 2 = 0 (add 2) 0
    = (λ f n. n) (add 2) 0 = 0

Násobení – mult (pokračování)

- No a také
  - multfn 2 0 = (λ m n. m (add n) 0) 2 0 = 2 (add 0) 0
    = (λ f n. f (f n)) (add 0) 0
    = add 0 (add 0 0) = add 0 0
    = 0
- Takže:
- LET mult = λ m n. m (add n) 0

Domácí úkol 4

- Definujte funkci sqr, která vrací druhou mocninu parametru a ukažte její správnou funkci
- Definujte fukci gt, která bere dvě čísla jako parametr a vrací True, pokud je první ostře větší, jak druhé, jinak vrací False
- Ukažte, že gt (sqr 3) (sqr 2) = True

Celočíselné dělení – div

- Uvažme div m n
  - Je-li *n* rovno 0, potom výsledek není definován
  - Jinak odčítáme od *m* tak dlouho *n*, dokud výsledek není menší, jak *n*, u toho počítáme počet odečtení
- Co to znamená, že není definován v rovině λ-kalkulu?
  - Z pohledu programátora
    - Výjimka
    - **Nekonečná smyčka**
  - Definujme výraz, který bere neomezené množství argumentů do nekonečna…

Bottom

- Výraz, který bude neustále na výstup produkovat sebe nazveme bottom (anglicky, nepřekládáme)
  - Tento modeluje do jisté míry nekonečnou smyčku v programu.
- Využijeme pro jeho definici **operátor pevného bodu**
  - Pevný bod výrazu E: Y E
    - Operátor je Y
  - Nechť pro výraz E je pevný bod k_E = Y E
  - Z definice vlastnosti: E k_E = k_E = Y E = E (Y E)
- Tedy pro bottom definujeme
  - LET ⊥ = Y (λ f x. f)

Příklad

- $\perp$ 1 True AND $= (Y\ (\lambda\ f\ x.\ f))\ 1\ True\ AND$
  $= ((\lambda\ f\ x.\ f)\ (Y\ (\lambda\ f\ x.\ f)))\ 1\ True\ AND$
  $= ((\lambda\ f\ x.\ f)\ \perp)\ 1\ True\ AND$
  $= (\lambda\ x.\ \perp)\ 1\ True\ AND$
  $= \perp\ True\ AND$
  $= (Y\ (\lambda\ f\ x.\ f))\ True\ AND$
  $= ((\lambda\ f\ x.\ f)\ \perp)\ True\ AND$
  $= (\lambda\ x.\ \perp)\ True\ AND$
  $= \perp\ AND$
  $= (Y\ (\lambda\ f\ x.\ f))\ AND$
  $= ((\lambda\ f\ x.\ f)\ \perp)\ AND$
  $= \perp$

Operátor pevného bodu Y

- LET Y = λ f . (λ x. f (x x)) (λ x. f (x x))
- Ukázka vlastnosti
  - (Y E) = (λ f . (λ x. f (x x)) (λ x. f (x x))) E
    = (λ x. E (x x)) (λ x. E (x x))
    = E ((λ x. E (x x)) (λ x. E (x x)))
    = E (YE)

Celočíselné dělení – div (pokračování)

- Tedy
  - LET div = λ m n. (iszero n ? ⊥ : divfn m n 0)
  - *LET <u>divfn</u> = λ m n r.(gt n m ? r : <u>divfn</u> (sub m n) n (succ r))*
- Jak z toho ven?
  - Operátorem pevného bodu
  - LET divf = (λf m n r.(gt n m ? r : f (sub m n) n (succ r)))
  - LET divfn = Y divf
- Příklad
  - div 5 2 = (λ m n. (iszero n ? ⊥ : divfn m n 0)) 5 2
    = divfn 5 2 0 = (Y divf) 5 2 0 = divf (Y divf) 5 2 0
    = (gt 2 5 ? 0 : divfn (sub 5 2) 2 (succ 0))
    = divfn 3 2 1 = divf (Y divf) 3 2 1
    = (gt 2 3 ? 1 : divfn (sub 3 2) 2 (succ 1)) = divfn 1 2 2 = 2

Domácí úkol 5

- Definujte operace sub a mult pomocí operátoru pevného bodu
- Ukažte jejich správnou funkčnost

Seznamy

- Prázdný seznam
  - LET [] = (False, False)
- Konstruktor seznamu
  - LET (:) = λ h t. (True,(h,t))
- Hlavička (první prvek) neprázdného seznamu
  - LET head = λ l. (fst l ? fst (snd l) : $\perp$)
- Zbytek seznamu
  - LET tail = λ l. (fst l ? snd (snd l) : $\perp$)
- Test na prázdný seznam
  - LET null = λ l. not (fst l)

Domácí úkol 6

- Ukažte správnou funkčnost funkcí pro seznamy
  - head (1:2:3:[])
  - tail (2:[])
  - null (1:2:[])
- Definujte funkci vracející délku seznamu vloženého jako parametr
- Definujte funkci vracející n-tý prvek seznamu délky alespoň n

Vybrané věty o λ-kalkulu 1.

- $E_1 \rightarrow E_2$
  - E_2 jsme získali „vyhodnocením“ E_1
  - Pokud E_2 neobsahuje žádný β-, η- redex, je úplně vyhodnocen
- Definice NF
  - Výraz E je v **normální formě**, pokud neobsahuje žádné jiné redexy krom α-redexu.
- Definice HNF
  - Výraz E je v **hlavičkové normální formě**, pokud to je proměnná, hodnota, vestavěná funkce aplikovaná na příliš málo argumentů, nebo λ-abstrakce jejíž tělo není redukovatelné (oproti NF může obsahovat redexy na pozicích argumentů)
- Definice WHNF
  - Výraz E je ve **slabé hlavičkové normální formě** pokud je ve HNF nebo je to λ-abstrakce

Vybrané věty o λ-kalkulu 2.

- Church-Rosserův teorém pro λ-kalkul
  - Pokud $E_1 = E_2$, pak existuje E takové, že $E_1 \to E$ a $E_2 \to E$
- Důsledky Church-Rosserova teorému
  - Pokud E má normální formu, potom $E \to E'$ pro nějaké E' v normální formě
  - Pokud E má normální formu a $E = E'$, potom E' má normální formu
  - *Pokud $E = E'$ a E i E' jsou oba v normální formě, potom E a E' jsou identické až na přejmenování vázaných proměnných* (α-konverzi)

Vybrané věty o λ-kalkulu 3.

- Normalizační teorém
  - Pokud E má normální formu, potom opakovaná redukce nejlevějšího β-, η- redexu (po možné α-konverzi pro zabránění neplatné náhradě) bude končit ve výrazu v normální formě.
- Definice
  - Posloupnost redukcí, v nichž je vždy redukován nejlevější redex, se nazývá **posloupnost redukcí v normálním pořadí**.

Kombinátory

- Teorie funkcí
- Lze odvodit z λ-kalkulu
- Lze zavést jako zcela oddělenou teorii

- V 80. letech 20. století „strojový kód“ pro překlad funkcionálních jazyků – Turner

Kombinátory S, K, I

- LET K = λ x y. x
- LET S = λ f g x. (f x) (g x)
- LET I = λ x. x = S K K

- Úkol:
  - Ukažte, že S K K = I.

Kombinátory B, C, K, W

- LET B = λ x y z. x (y z)
- LET C = λ x y z. x z y
- LET K = λ x y. x
- LET W = λ x y. x y y

- Úkol, ukažte, že:
  - **B** = **S (K S) K**
  - **C** = **S (S (K (S (K S) K)) S) (K K)**
  - **W** = **S S (K (S K K))**
  - **S** = **B (B (B W) C) (B B)**

Převod kalkulu na kombinátory

- Definice
  - Pro libovolnou proměnnou V a kombinatorický výraz E existuje jiný kombinatorický výraz λ*V.E, který simuluje λV.E tak, že λ*V.E = λV.E

- Induktivně pro S, K, I, když
  - P ::= S | K | I | P P | x

  - λ*x.x = I
  - λ*x.P = K P , $x \notin FV(P)$
  - λ*x.P P‘ = S (λ*x.P) (λ*x.P‘)

Ukázka převodu

- $\lambda^*x.\ \lambda^*y.\ x$
  $= \lambda^*x.\ K\ x$
  $= S\ (\lambda^*x.\ K)\ (\lambda^*x.\ x)$
  $= S\ (\lambda^*x.\ K)\ I$
  $= S\ (K\ K)\ I$

- $(\lambda^*\ x.\ \lambda^*\ y.\ y\ x)$
  $= \lambda^*\ x.\ S\ (\lambda^*\ y.\ y)\ (\lambda^*\ y.\ x)$
  $= \lambda^*\ x.\ (S\ I)\ (K\ x)$
  $= S\ (\lambda^*\ x.\ (S\ I))\ (\lambda^*\ x.\ K\ x)$
  $= S\ (K\ (S\ I))\ (S\ (\lambda^*\ x.\ K)\ (\lambda^*\ x.\ x))$
  $= S\ (K\ (S\ I))\ (S\ (K\ K)\ I)$

Redukce

- $I\ E \to E$
- $K\ E_1\ E_2 \to E_1$
- $S\ E_1\ E_2\ E_3 \to (E_1\ E_3)\ (E_2\ E_3)$

- $S\ (K\ K)\ I\ P\ Q\ R$

  $\to ((K\ K)\ P)\ (I\ P)\ Q\ R$

  $\to K\ P\ Q\ R$

  $\to P\ R$

Optimalizace

- S (K E) I = E
- S (K E) (K E‘) = K (E E‘)

- ?
- S (K K) I = K

- Domácí úkol 7
  - Ukažte!