Funkcionální programování

Kristýna Kaslová

Historie

- Alonzo Church (30. léta)
  - Netypovaný lambda kalkul
    - Základ prvních funkcionálních jazyků
    - Jeho konstrukce i v mnoha současných programovacích jazycích (Python)
- Haskell Brooks Curry
  - Kombinatorická logika jako variace lambda kalkulu
    - Lambda výrazy nahrazeny omezenou sadou primitivních funkcí

- John McCarthy
  - Vytvořil programovací jazyk LISP
    - Později velmi populární v oblasti umělé inteligence
    - Common Lisp – scriptovací jazyk pro některé aplikace (Autocad)
- David Turner
  - Jazyk Miranda
    - První čistě funkcionální jazyk určený především pro komerční využití
- Jazyk Haskell
  - Vytvořen zvláštním výborem v 80. letech
  - Standard pro oblast funkcionálního programování

Lambda kalkul

- Teorie funkcí založená na velmi jednoduchém jazyce
- Záladní prvky tohoto jazyka (lanbda výrazy)
  - Proměnná, aplikace, abstrakce
    - Proměnná – blíže nespecifikovaná hodnota (x,y,z)
    - Aplikace – volání funkce s jedním parametrem
      - Funkce i argument jsou zadány jako lambda výrazy ve tvaru (e1 e2)
    - Abstrakce – reprezentace funkce tvořené proměnnou označující parametr a tělem ve tvaru lambda výrazu
      - \ x → e

Koncepty

- Higher-order funkce
- Čisté funkce
- Striktní, nestriktní a líné vyhodnocení

Higher-order funkce

- Můžou převzít nějakou funkci jako argument nebo navrátit funkci jako výsledek
  - Derivace, neurčitý integrál, součin funkcí,...
- Důsledek toho, že funkcemi jsou entity první kategorie (First class)
  - Funkce může vzniknout za běhu programu, být uložena v proměnné a předána jako argument jiné funkci

Čisté funkce

- Funkce bez vedlejších efektů
  - Chování jednodušší na pochopení
  - Pokud se výstup funkce nepoužívá, nemusí se volat
  - Funkce lze vyhodnocovat v libovolném pořadí, nebo i paralelně
  - Pokud mají dva výrazy stejné hodnoty, lze jeden dosadit za druhý

y = f(x) * f(x)

→ Pokud je funkce f(x) čistá

z = f(x)

y = z*z

→ Eliminuje se druhé vyhodnocení f(x)

Striktní vyhodnocení

- Argumenty funkce vyhodnoceny před jejím voláním
- Efektivní – výpočet se provádí jednou

f:=x^2+x+1

g:=x+y

f(g(1,4))

f(g(1,4)) → f(1+4) → f(5) → 5^2+5+1 → 31

Nestriktní vyhodnocení

- Méně efektivní
- Argumenty přenehány funkci nevyhodnocené
- Podporuje nekonečné datové struktury

f:=x^2+x+1

g:=x+y

f(g(1,4))

f(g(1,4)) → g(1,4)^2+g(1,4)+1 → (1+4)^2+(1+4)+1 → 5^2+5+1 → 31

Líné vyhodnocení

- Typ nestriktního vyhodnocení
- Argument není spočítán nikdy více než jednou
- Používán moderními čistě funkcionálními jazyky
  - Miranda, Clean, Haskell

Seznamy

- Základní datové typy mnoha funkcionálních i logických jazyků
- Můžeme pomocí nich reprezentovat text jako seznam řádků a řádky jako seznamy slov nebo znaků

Haskell

Základní rysy

- Syntaxe napodobující matematický zápis
  - Roli hraje i odsazení textu
- Statický typový systém
  - Každý symbol má již v době překladu přidělen datový typ
- Automatické líné vyhodnocování

Implementace

- Existuje jen relativně malé množství překladačů
  - Odkazy na ně na www.haskel.org
  - Hugs
    - Vhodný jako vývojový nastroj i pro začátečníky
    - Běh aplikací je extrémně pomalý
  - GHC
    - Vhodný pro produkční prostředí
    - Relativně rychlý

Operátory a funkce

- Funkce mají alfanumerické operátory a používají se prefixově
  - Infixový zápis jen pokud jsou binární
    - Musí být ve zpětných apostrofech

mod 5 2 (prefix.)

5 `mod` 2 (infix.)

- Operátory obsahují pouze speciální znaky
  - Používají se infixově – prefixově nebo samostatně v závorkách

2+3 (infix.)

(+) 2 3 (prefix.)

Priorita

- Při použití operátorů zohledněna priorita a asociativita
- Volání funkce má vyšší prioritu než jakýkoliv operátor
- Parametry funkcí se neuzavírají do závorek
  - Závorky však mnohdy nutné pro zajistění správného pořadí vyhodnocení

sin 1 + 3 = (sin 1) + 3

→ funkce má vyšší prioritu než sčítání

sin (1 + 3) = sin 4

→ zajištění priority vyhodnocení

cos (sin 1) = cos sin 1 ~ (cos sin) 1

→ typová chyba

Case controlled language

- Nejen že rozeznává velikost písmen, ale dokonce má velikost písmen syntaktickou roli
  - Identifikátory funkcí začínají malým písmenem
  - Identifiátory typů začínají velkým písmenem
  - Konstruktory nových hodnot začínají velkým písmenem (včetně nulárních konstant např. True)

Seznamy

- Homogenní (=položky stejného typu)
- Velikost není omezena
- Pořadí prvků je významné [1,2] != [2,1]
- Speciální případ je seznam hodnot typu Char, který je ekvivalentem typu String
- Prázdný seznam [ ]
- Přidání prvku do seznamu (:)
  - Připojí prvek zleva

Přidání prků do seznamu

1:2:3:4:5:[ ]

→ [1,2,3,4,5]

Zápis řetězce

Buď ['l','a','m','b','d','a']

Nebo "lambda"

Struktura programu

- Na nejvyšší úrovni se program skládá z modulů
  - Poskytují možnost opakovaného použití v nových projektech
  - Jeden z modulů se musí jmenovat Main (funkce main)
    - Určení, kde má výpočet začít
  - Hlavička modulu může vypadat takto:

```
module Sorting (quickSort, bubbleSort, mergeSort) where


import Maybe

import SortUtils hiding (merge)
```

- Není nutné (ani obvyklé) uvádět středníky ukončující výraz
  - Jejich funkci přebírá formátování vlastního kódu
    - Kód který patří do jednoho bloku má stejné odsazení od začátku řádky
    - Vnořený blok má odsazení větší

```
otoc xs = obrat xs []
    where
        obrat [] ys     = ys
        obrat (x:xs) ys = obrat xs (x:ys)
```

  - Pro odpůrce odsazování – možno místo odsazení použít { }

```
qsort [ ]     = [ ]
```

→ setříděním prázdného seznamu dostaneme prázdný seznam

```
qsort (x:xs) = qsort mensi ++ [x] ++ qsort vetsi_rovno
```

→ výsledkem třídění seznamu, jehož první prvek je x a zbytek xs, je seznam, který vznikne spojením setříděného seznamu mensi, prvku x a setříděného seznamu vetsi_rovno

where

```
mensi  = [ y | y <- xs, y < x]
```

→ Seznam mensi získáme ze seznamu xs výběrem takových prvků y, které jsou menší než x

```
vetsi_rovno  = [ y | y <- xs, y >= x]
```

→ Seznam vetsi_rovno získáme obdobně výběrem prvků větších nebo rovných x

Použité zdroje

- http://www.cs.vsb.cz/navrat/vyuka/flp/texty/fp/index.html
- http://makovice.nipax.cz/FEL/OBP/%23prednasky/X36OBP-functional.pdf
- http://ithil.ujep.cz/workspace/haskell.pdf
- http://cs.wikipedia.org/wiki/Funkcion%C3%A1ln%C3%AD_programov%C3
- http://www.root.cz/clanky/haskell-a-funkcionalni-programovani/

Děkuji za pozornost