Programovací jazyk Haskell

Ing. Lumír Návrat
katedra informatiky, D 403
☎ 59 732 3252

Historie

- září 1991 – Gofer
  - experimentální jazyk
  - Mark P. Jones
- únor 1995 – Hugs
- Hugs98
  - téměř úplná implementace jazyka Haskell 98
  - některá rozšíření navíc

Instalace + dokumentace

- Základní zdroje
  - http://haskell.org
    - popis jazyka a knihoven
  - http://haskell.org/hugs
    - instalace (Win / Unix)
    - uživatelská příručka (je součástí instalace)
- Další součásti
  - Scholl of Expression (SOE)
  - Hugs Graphics Library

Použití

- Princip výpočtu: kalkulátor

```
$ hugs
Prelude> 2*(3+5)
16
Prelude> cos 0
1.0
Prelude>
```

- Skript: definice uživatelských funkcí

```
$ hugs priklad.hs
```

Řídicí příkazy

- Editace souboru
  :e*dit [*soubor.hs*]* :e
- Načtení skriptu
  :l*oad [*soubor.hs*]* :r*eload*
- Ukončení
  :q*uit*
- Nápověda
  :?
- Unix – nastavení editoru v souboru ~/.profile
  export HUGSFLAGS="-E\"vi +%d %s\" +t +s -u"

Skript

- priklad.hs

```
module Priklad where
-- funkce, která vrací součet dvou čísel
soucet x y = x + y
```

- priklad.lhs

```
> module Priklad where

Funkce, která vrací faktoriál čísla

> f n = if n == 0 then 1 else n * f (n-1)
```


Datové typy

- Základní datové typy
  - 1::Int
  - ‘a’::Char
  - True,False::Bool
  - 3.14::Float
- Seznamy `[a]`
  - prázdný seznam `[]`
  - neprázdný seznam `(x:xs)`
  - `1:2:3:[] :: [Int]`
  - `[1,2,3]  :: [Int]`

Datové typy

- Uspořádané n-tice `(a,b,c,...)`
  - `(1,2) :: (Int,Int)`
  - `(1,['a','b']):(Int, [Char])`
  - `() :: ()`
- Funkce `a->b`
  - `faktorial :: Int -> Int`
  - `soucet :: Int -> Int -> Int`
  - `plus :: (Int, Int) -> Int`

Datové typy

- Uživatelské datové typy
  - `data Barva  = Cerna`
    `| Bila`
  - `data Tree a = Leaf a`
    `| Node a (Tree a) (Tree a)`
  - `type String = [Char]`
  - `type Tabulka a = [(String, a)]`

Definice funkcí

- *Rovnice* a *unifikace vzorů* (pattern matching):
  - `f  pat11  pat12  ...  = rhs1`
  - `f  pat21  pat22  ...  = rhs2`
  - `...`
- Vybere se první rovnice vyhovující parametrům
- Pokud se nenajde → chyba

Vzory

- proměnná
  - `inc x = x + 1`
- konstanta
  - `not True = False`
  - `not False = True`
- seznam
  - `length []   = 0`
  - `length (x:xs) = 1 + length xs`

Vzory

- n-tice
  - `plus (x,y) = x+y`
- konstruktor uživatelského typu
  - `nl (Leaf _) = 1`
  - `nl (Tree _ l r) = (nl l) + (nl r)`
- pojmenování části vzoru
  - `duphd p@(x:xs) = x:p`

Vzory

- anonymní proměnná _
  - `hd (x:_) = x`
- vzor typu *n+k*
  - `fact 0 = 1`
  - `fact (n+1) = (n+1)*fact n`
- strážené rovnice
  - ```
    fact n | n == 0    = 1
           | otherwise = n * fact(n-1)
    ```


Příklady

- Faktoriál
  - ```
    fakt1 n     = if n == 0 then 1
                  else n * fakt1 (n-1)
    ```
  - ```
    fakt2 0     = 1
    fakt2 n     = n * fakt2 (n-1)
    ```
  - ```
    fakt3 0     = 1
    fakt3 (n+1) = (n+1) * fakt3 n
    ```
  - ```
    fakt4 n | n == 0    = 1
            | otherwise = n * fakt4 (n-1)
    ```


Příklady

- Fibonacciho čísla
  - ```
    fib :: Int -> Int
    fib 0     = 0
    fib 1     = 1
    fib (n+2) = fib n + fib (n+1)
    ```


Příklady

- Délka seznamu
  - ```
    length [] = 0
    length (x:xs) = 1 + length xs
    ```
- Poznámka: pozor na konflikt s předdefinovanými funkcemi!
  - ```
    module Pokus where
    import Prelude hiding(length)

    length []     = 0
    length (_:xs) = 1 + length xs
    ```

Lokální definice

- Konstrukce `let ... in`
  - ```
    f x y = let p = x + y
                q = x - y
            in p * q
    ```
- Konstrukce `where`
  - ```
    f x y = p * q
            where p = x + y
                  q = x - y
    ```


Částeční aplikace funkcí

- Curryho tvar funkce
  - ```
    add :: Int -> Int -> Int
    add x y = x + y
    ```
  - ```
    plus :: (Int, Int) -> Int
    plus (x,y) = x + y
    ```
  - `add = curry plus` `curry :: ?`
  - `plus = uncurry add` `uncurry :: ?`
- Řezy funkce
  - `inc x = 1 + x`
  - `inc x = add 1 x`
  - `inc = add 1`
  - `inc = (+1)  = (1+)`
  - `add = (+)`

Příklady

- Vytvoření seznamu druhých mocnin
  - ```
    dm [] = []
    dm (x:xs) = sq x : dm xs
                where sq x = x * x
    ```
- Seřazení seznamu (quicksort)
  - ```
    qs [] = []
    qs (x:xs) =
       let ls = filter (< x) xs
           rs = filter (>=x) xs
       in qs ls ++ [x] ++ qs rs
    ```


Funkce pro seznamy

- Přístup k prvkům seznamu
  - `head [1,2,3]  = 1`
  - `tail [1,2,3]  = [2,3]`
  - `last [1,2,3]  = 3`
  - `init [1,2,3]  = [1,2]`
  - `[1,2,3] !! 2  = 3`
  - `null []  = True`
  - `length [1,2,3] = 3`

Funkce pro seznamy

- Spojení seznamů
  - `[1,2,3] ++ [4,5]  = [1,2,3,4,5]`
  - `[[1,2],[3],[4,5]] = [1,2,3,4,5]`
  - `zip [1,2] [3,4,5] = [(1,3),(2,4)]`
  - `zipWith (+) [1,2] [3,4] = [4,6]`
- Agregační funkce
  - `sum [1,2,3,4]     = 10`
  - `product [1,2,3,4] = 24`
  - `minimum [1,2,3,4] = 1`
  - `maximum [1,2,3,4] = 4`

Funkce pro seznamy

- Výběr části seznamu
  - `take 3 [1,2,3,4,5] = [1,2,3]`
  - `drop 3 [1,2,3,4,5] = [4,5]`
  - `takeWhile (>0) [1,3,0,4] = [1,3]`
  - `dropwhile (> 0) [1,3,0,4] = [0,4]`
  - `filter (>0) [1,3,0,2,-1] = [1,3,2]`
- Transformace seznamu
  - `reverse [1,2,3,4] = [4,3,2,1]`
  - `map (*2) [1,2,3] = [2,4,6]`

Úkol

- Pokuste se uvedené funkce pro seznamy implementovat
  - co je triviální případ?
    - `length  []  = 0`
    - `maximum [x] = x`
  - funkci umím udělat s kratším seznamem -> jak tuto hodnotu zkombinuji s prvním prvkem seznamu, abych dostal výsledek?
    -
```
maximum (x:y:ys) | x > y = maximum (x:ys)
                 | otherwise = maximum
(y:ys)
```