

# (27) HASKELL

- silnie typowany funkcionalny programovací jazyk
- je case-sensitive a typy, typy typicky, datove hodnoty odd. mimi paimat
- celým písmem a vše ostatní malým
- nejsem zde reference → vše jsou funkce

## Datové typy

### 1) Primitivní datové typy

- Int a Integer - oba to jsou celé čísla, Int má omezený rozsah ale je efektivnější
- Word - celé čísla bez znaménka → přirozená čísla
- Float - desetinná čísla se single precision
- Double - - " - s double precision
- Char - znaky
- Bool - logické hodnoty True a False

⇒ mimi list s  
celým  
písmem  
a odd.

⇒ POZOR! silnie typovaný jazyk - všechno je typováno předem a všechno se striktně deklaruje, datový typi operandy odd.

### 2) Strukturované datové typy

- seznamy - homogenní datová struktura

- konstruktorem jsou: a [] kde [] je prázdný seznam a : pouze připojuje do seznamu

[1,2,3] ⇒ 1:2:3:[] - umístění to může být 1:2:3 ale mimi by  
chvilku připojil první k seznamu - ale pak k přidání

(x:xs) <sup>↑</sup> kóde seznamu

↳ hlavička seznamu

tail (x:xs) ⇒ xs - kde xs je vždy  
seznam (přijímá prázdný)

head (x:xs) ⇒ x je první

head [] - exception

head (x:\_) = x

- n-tice - heterogenní datová struktura
- konstruktorem jsou: a ()


data + máce s velkými  
písmem + definice = ...

objekt: data Color = Blue | Red | Green

parametrické: data RGBColor a = RGB a a a

rekursivní: data Tree a = LF | Nd a (Tree a) (Tree a)

data Stack a = Top a (Stack a) | Bottom

nechť seznam je list a  
mimo to co má dle příkazu  
↓  
list je to všechno, co má list  
dne -  ①



- Haskell je jazyk s níže uvedenými typy datových typů a výstupů se speciální následující

$:: \text{Int} \rightarrow [\text{Int}] \rightarrow [\text{Int}]$

$:: \text{Float} \rightarrow [\text{Float}] \rightarrow [\text{Float}]$

$+ :: \text{Int} \rightarrow \text{Int} \rightarrow \text{Int}$

} nizkoprima!  
funkce

- abychom mohli pracovat s datovými typy pomocí operátorů: nemuseli jsme definovat nové vstupní a výstupní datové typy - protože můžeme získat o pole Int nebo o pole Float nebo i Bool ať už se s nimi pracuje s Int, Double a Float tak se s nimi pracuje následující

$:: a \rightarrow [a] \rightarrow [a]$  - a to říká, že můžeme mít pole nějakého typu a pole pole nějakého typu a na výstup můžeme mít pole pole nějakého pole nějakého typu

- se a může být Int nebo Double nebo cokoli

### 3) Typové třídy

- typové třídy jsou podstatou abstrakce, které sdílejí metody, které mají nad sebou implementaci ~~typu~~ typ, který je instancí této třídy

- typ A je typem typové třídy B, protože má definované operace, které třída B vyžaduje

- každá typová třída má definované operace pro jednotlivé datové typy, které mohou být její instance

- například třída Eq říká, že všechny typy, které jsou její instancí, musí mít implementovanou metodu porovnání ( $=$ ) a to se používá na datovém typu, který ten typ je - tedy porovnání Charu, Intu, Floatu, ...

- datová třída tedy definuje minimální operace nad typem, který je instancí té datové třídy

- například typový třídní Eq - typ musí mít definované  $=$  a  $\neq$  (ROVNOST)  
- Ord - typ musí mít definované  $\leq$ ,  $<$ ,  $>$ ,  $\geq$  (USPOŘÁDÁNÍ)  
- Show - typ musí mít definovaný způsob, jakým se přetvoří na řetězec - používá se symboly

- můžeme mít funkci, která přijímá třídu a restrikuje, která se hlásí, že funkce se může přizpůsobit nějakému typu a ale musí to být typ, který patří do třídy Eq

- viz  $=$  operace je definována nad typem a který je třídní Eq

$= :: (\text{Eq } a) \Rightarrow a \rightarrow a \rightarrow \text{Bool}$

Př:

$\text{member} :: (\text{Eq } a) \Rightarrow a \rightarrow [a] \rightarrow \text{Bool}$   
Ověří, že je/je nečlenem?

to je to, o čem - přese se třídu funkce  $\Rightarrow$

### Rekurze

1) definiční - je návratem z rekurze nově vypočtené hodnoty  
 $f x = f(x+1)$

2) ofsetní - při návratu z rekurze se ještě něco přičte  
 $f x = (f x) + 1$

3) lineární - ve výpočtu funkce je jím jich rekurzivní volání sama sobě  $\rightarrow$  lze přeměnit na cyklus ②

Ord  $\Rightarrow$

Show  $\Rightarrow$

$\vdots$

to je přechod  
jazyka  
 $f x = 1 + f x$   
 $+ f(x+1)$   
rekurze



## Funkce

- Haskellu majú' vstúpené funkcie dávajú predpis daktorých typov parametrov a návratové hodnoty. Sú to daktoré typy, je dávajú buď konštantu (Int, ...) alebo obecnú ( $a, b, \dots$ ) a musia byť uvedené také podmienky na nájdení toho typu a do nájdení triedy (Eq, Ord, Show, ...)

$f :: [a] \rightarrow a$       Se a zion shjane' beq parametere xi pole tipin nejjallbe a  
 $f :: Int \rightarrow Int \rightarrow Int$       a marratoreu' loduolu xi rose tipin a

$f :: a \rightarrow [a] \rightarrow b$      $\text{let } a \text{ is } b$  gives machine type

$f :: \text{Ord } a \Rightarrow a \rightarrow a \rightarrow a$     Sece pe marea' nictel malarintol typp a  
do typpu' tirdy Ord a la f pe tirdone

Re:

length :: [a] → Int

length [] = 0

$\text{length } (x:xs) = 1 + (\text{length } xs)$  — rekursiv lineární řešení

- poznamenání části roste - třeba farmaceut

merge [ ] [1] = [1]

merge  $[2] = 2$

$$\text{merge } L2@(x:xs) \text{ } L1@(y:ys) = \text{if } x < y \text{ then } x:(\text{merge } xs \text{ } L1) \text{ else } y:(\text{merge } ys \text{ } L2)$$

c) Hashell num' if then else !

- anonymní pracovníci - každý má své jméno co to je a může to rozpoznat

$$\text{tail}(x:-) = x$$

(3) omogućiti promišljanje

- localm' funkce - polno potrebyne pravit' nuzitel' lokaln' funkci is nuzn'mi z'dm' funkci a nepotrebyne sm't' z'j' platno nilete z'mde tak z' pravit' a tak z' po WHERE definicije

$$f_a = g_a$$

WHERE

WHERE

duleñit  $\rightarrow$  ga = a + 1  
vj h obavezi

Pf.: fibonacci  $m = \text{fib} !! m$   
WHERE

WHERE

$$fib = 0:1: zipWith (+) fib (tail fib)$$

(6) *loa'ibm' fumbre hlan' not'isa' neho'weme' pole*



Lazy evaluation

- Haskell považí každou 'line' chodu 'ať' a každý depláit 'nájde' hodnoty až keď je spin potrebný

- Pr.: take 5 [1..]  $\Rightarrow$  [1,2,3,4,5]

⇒ LE se vyvíjí pro výpočet  $n$ -tého prvku Fibonacciho posloupnosti:

fib 0 = 0

$$f(1) = 1$$
$$fib\ n = fib(n-1) + fib(n-2)$$

— je lu redura (repetma' a melinca'm)  
a lui je lu pomeli' a neefelion'  
(m-2) — model & shonist p'etion pro  
melodulic p'etion - p'etion pro  
m = 999999999999  
(lu d'et me')

Fibonacci  $n = \text{fib} !! n \leftarrow \text{naar}' m \text{ 'y' fms rekursies pols lre'gi}$   
WHERE  
define' nre jals lre'sm' fms

$\text{fib} = 0:1: \text{zip With } (+) \text{ fib (tail fib)}$

zip with (4)  $[1, 2, 3] [3, 4, 5] \Rightarrow [1+3, 2+4, 3+5] \Rightarrow [4, 6, 8]$

zip with (\*)  $[1, 2, 3] [3, 4] \Rightarrow [4, 6]$  - so how 4 then 6

$\text{tail}[0, 1, 2] \Rightarrow [1, 2]$        $\text{tail}[1..10] \Rightarrow [2..10]$

- for  $m=2$  no  $du'$

zip with (4)  $\begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} 1 \end{bmatrix}$   
 $= \begin{bmatrix} 1 \end{bmatrix}$

zipWith (4) [0, 1, 1] [1, 1]  
= [1, 2]

ohne Prüfung!  
0:1: [1, 2]

**Akce** - abe křesťan C noví 'leč' program - jistě se sebou a přivádí deponování, vyřizování, měření.  
 - abe jistě se Haskellu vyřizování IO operací křesťan IO Integer je abe  
 - se Haskellu měření se přivádí přivádění funkce ale  
 se abe 'm' doměření (křesťan IO) a křesťan noví 'leč'  
 se do křesťan abe se přivádí přivádění  
 noví 'leč' se noví 'leč'

main :: IO ()  
 main = do  
 c <- getChar  
 putChar c

getChar :: IO Char  
 - jistě se noví 'leč' (4)

Scanned with CamScanner