

Кортежи и списъци

Трифон Трифонов

Функционално програмиране, 2018/19 г.

21 ноември 2018 г.

Кортежи (tuples)

Кортежите са наредени n -торки от данни от произволен тип.

- **Примери:** $(1, 2)$, $(3.5, 'A', \text{False})$,
 $(("square", (^2)), 1.0)$
- Тип кортеж от n елемента: (t_1, t_2, \dots, t_n)
- Стойности: наредени n -торки от вида (x_1, x_2, \dots, x_n) , където x_i е от тип t_i
- Позволяват “пакетиране” на няколко стойности в една
- Операции за наредени двойки:
 - $(,)$ $:: a \rightarrow b \rightarrow (a, b)$ — конструиране на наредена двойка
 - **fst** $:: (a, b) \rightarrow a$ — първа компонента на наредена двойка
 - **snd** $:: (a, b) \rightarrow b$ — втора компонента на наредена двойка

Потребителски типове

- Типът (`String`, `Int`) може да означава:
 - име и ЕГН на човек
 - продукт с описание и количество
 - сонет на Шекспир и поредният му номер
- Удобно е да именуваме типовете, за да означаваме смисъла им
- `type` <конструктор> = <тип>
 - конструкторите са идентификатори, започващи с главна буква
- **Примери:**
 - `type Student = (String, Int, Double)`
 - `type Point = (Double, Double)`
 - `type Triangle = (Point, Point, Point)`
 - `type Transformation = Point -> Point`
 - `type Vector = Point`
 - `addVectors :: Vector -> Vector -> Vector`
 - `addVectors v1 v2 = (fst v1 + fst v2, snd v1 + snd v2)`

Особености на кортежите

- `fst (1,2,3) → Грешка!`
 - `fst` и `snd` работят само над наредени двойки!
- $((a,b),c) \neq (a,(b,c)) \neq (a,b,c)$
- Няма специален тип кортеж от един елемент...
- ...но има тип “празен кортеж” `()` с единствен елемент `()`
 - в други езици такъв тип се нарича `unit`
 - използва се за означаване на липса на информация

Образци на кортежи

Образец на кортеж е конструкция от вида (p_1, p_2, \dots, p_n) .

Пасва на всеки кортеж от точно n елемента (x_1, x_2, \dots, x_n) , за който образецът p_i пасва на елемента x_i .

- `addVectors (x1, y1) (x2, y2) = (x1 + x2, y1 + y2)`
- `fst (x, _) = x`
- `snd (_, y) = y`
- `getFN :: Student -> Int`
- `getFN (_, fn, _) = fn`
- образците на кортежи могат да се използват за “разглобяване” на кортежи при дефиниция
- `(x,y) = (3.5, 7.8)`
- `let (_, fn, grade) = student in (fn, min (grade + 1) 6)`

Именувани образци

- намиране на студент с по-висока оценка

```
betterStudent (name1, fn1, grade1) (name2, fn2, grade2)
  | grade1 > grade2 = (name1, fn1, grade1)
  | otherwise      = (name2, fn2, grade2)
```

- ами ако имахме 10 полета?
- удобно е да използваме **именувани образци**
- <име>@<образец>

```
betterStudent s1@(_, _, grade1) s2@(_, _, grade2)
  | grade1 > grade2 = s1
  | otherwise      = s2
```

Списъци

Дефиниция

- 1 Празният списък $[]$ е списък от тип $[a]$
 - 2 Ако h е елемент от тип a и t е списък от тип $[a]$ то $(h : t)$ е списък от тип $[a]$
 - h — глава на списъка
 - t — опашка на списъка
- списъкът е последователност с **произволна дължина** от елементи от **еднакъв тип**
 - $(:)$ $:: a \rightarrow [a] \rightarrow [a]$ е **дясноасоциативна** двуместна операция
 - $(1:(2:(3:(4:[])))) = 1:2:3:4:[] \neq (((1:2):3):4):[]$
 - $[a_1, a_2, \dots, a_n]$ е по-удобен запис за $a_1:(a_2:\dots(a_n:[])\dots)$
 - $[1,2,3,4] = 1:[2,3,4] = 1:2:[3,4] = 1:2:3:[4] = 1:2:3:4:[]$

Примери

- $[False] :: [Bool]$
- $["Иван", 4.5] :: \perp$
- $[1]:2 :: \perp$
- $[[1,2],[3],[4,5,6]] :: [[Int]]$
- $([1,2],[3],[4,5,6]) :: ([Int],[Int],[Int])$
- $[(1,2),(3),(4,5,6)] :: \perp$
- $((1,2),(3),(4,5,6)) :: ((Int,Int),Int,(Int,Int,Int))$
- $[[]] :: [[a]]$
- $[]:[] :: [[a]]$
- $[1]:[[]] :: [[Int]]$
- $[:[1]] :: \perp$
- $[[1,2,3],[[]]] :: [[Int]]$
- $[[1,2,3],[[]]] :: \perp$
- $[1,2,3]:[4,5,6]:[[]] :: [[Int]]$

Низове

- В Haskell низовете са представени като списъци от символи
- `type String = [Char]`
- Всички операции над списъци важат и над низове
- **Примери:**

- `['H', 'e', 'l', 'l', 'o'] == "Hello"`
- `'H':'e':'l':'l':'o':[] == "Hello"`
- `'H':'e':"llo" == "Hello"`
- `"" == [] :: [Char]`
- `[[1,2,3], ""] :: ⊥`
- `["12", ['3'], []] :: [String]`

Основни функции за списъци

- `head :: [a] -> a` — връща главата на (непразен) списък
 - `head [[1,2],[3,4]] -> [1,2]`
 - `head [] -> Грешка!`
- `tail :: [a] -> [a]` — връща опашката на (непразен) списък
 - `tail [[1,2],[3,4]] -> [[3,4]]`
 - `tail [] -> Грешка!`
- `null :: [a] -> Bool` — проверява дали списък е празен
- `length :: [a] -> Int` — дължина на списък

Генератори на списъци

Можем да генерираме списъци от последователни елементи

- $[a..b] \rightarrow [a, a+1, a+2, \dots, b]$
- **Пример:** $[1..5] \rightarrow [1, 2, 3, 4, 5]$
- **Пример:** $['a'..'e'] \rightarrow "abcde"$
- Синтактична захар за `enumFromTo from to`

- $[a, a + \Delta x .. b] \rightarrow [a, a + \Delta x, a + 2\Delta x, \dots, b']$, където b' е най-голямото число $\leq b$, за което $b' = a + k\Delta x$
- **Пример:** $[1, 4..15] \rightarrow [1, 4, 7, 10, 13]$
- **Пример:** $['a', 'e'..'z'] \rightarrow "aeimquy"$
- Синтактична захар за `enumFromThenTo from then to`

Рекурсивни функции над списъци

- $(++) :: [a] \rightarrow [a] \rightarrow [a]$ — слепва два списъка
 - $[1..3] ++ [5..7] \rightarrow [1,2,3,5,6,7]$
- $a ++ b = \text{if null } a \text{ then } b \text{ else head } a : \text{tail } a ++ b$
- $\text{reverse} :: [a] \rightarrow [a]$ — обръща списък
 - $\text{reverse } [1..5] \rightarrow [5,4,3,2,1]$

```
reverse a
| null a      = a
| otherwise  = reverse (tail a) ++ [head a]
```