IB015 Neimperativní programování

Programování a data

Jiří Barnat
Libor Škarvada

Programování a data

Pozorování

- Programy pro své fungování potřebují různé informace – **data**.
- Data jsou vstupní hodnoty, výstupní hodnoty, mezivýsledky výpočtů, parametry funkcí, atd.

Programování a data

- Data je třeba uchovávat tak, aby je bylo možné zpracovat mechanicky/strojově.
- Tvorba jednoznačného popisu struktury a **způsobu uložení dat je nedílná součást procesu programování**.

Strukturovaná data

Dekompozice dat

- Veškerá data použitá v programu je třeba vystavět ze základních datových elementů podle definovaných pravidel.
- Existují striktní pravidla pro dekompozici dat, my si však v rámci IB015 vystačíme s intuicí.

Základní datové elementy

- Čísla, Znaky, Pravdivostní hodnoty

Základní způsoby kompozice dat

- **Uspořádané n-tice**
- **Seznamy**
- Stromy

Uspořádané n-tice

Co to je

- Pevně daný počet nějakých hodnot v pevně daném pořadí.
- Prvek kartézského součinu nosných množin.

Příklady

- Datum: `(11, "březen", 1977)` .
- Přihlašovací údaje: `("xbarnat", "mam-IQ-tykve")`
- Pozice pixelu v rastrovém obrázku: `(x, y)` ,
  všimněme si, že `(12,43)` $\neq$ `(43,12)` .

Kdy se má použít

- **Počet prvků v n-tici je znám předem**,
  tj. v okamžiku psaní zdrojového kódu.
- Počet prvků v n-tici je malý (hodnota n je malá).

Seznamy

Co to je

- Posloupnost hodnot stejného charakteru (stejného typu).
- Posloupnost může být prázdná, konečná i nekonečná.
- Každý prvek v seznamu je na nějaké (unikátní) pozici.

Příklady

- Seznam čísel: `[12,43,-3,15,29]`
- Nekonečný seznam přirozených čísel: `[1,2..]`
- Seznam uspořádaných dvojic:
  `[("Fero",12), ("Nero",7), ("Pero",5)]`
- Prázdný seznam: `[]`

Kdy se má použít

- **Data vznikají nebo se zpracovávají postupně.**
- Počet prvků použitých programem není předem znám.

Příklad dekompozice dat

Aplikace – Diář squashových partnerů.

- Program pro správu kontaktů na různé squashové hráče.
- Hlavní datová struktura je seznam kontaktů.

Datová dekompozice

Příklad dekompozice dat

Aplikace – Diář squashových partnerů.

- Program pro správu kontaktů na různé squashové hráče.
- Hlavní datová struktura je seznam kontaktů.

Datová dekompozice

- Seznam kontaktů – `[Kontakt]`

  `[kontakt1, kontakt2, kontakt3, ..., kontakt315]`

Příklad dekompozice dat

Aplikace – Diář squashových partnerů.

- Program pro správu kontaktů na různé squashové hráče.
- Hlavní datová struktura je seznam kontaktů.

Datová dekompozice

- Seznam kontaktů – `[Kontakt]`

  `[kontakt1, kontakt2, kontakt3, ..., kontakt315]`

- Kontakt je uspořádaná trojice

  `(Prezdivka, Telefon, Adresa)`

Příklad dekompozice dat

Aplikace – Diář squashových partnerů.

- Program pro správu kontaktů na různé squashové hráče.
- Hlavní datová struktura je seznam kontaktů.

Datová dekompozice

- Seznam kontaktů – `[Kontakt]`

  `[kontakt1, kontakt2, kontakt3, ..., kontakt315]`

- Kontakt je uspořádaná trojice

  `(Prezdivka, Telefon, Adresa)`

- Adresa je uspořádaná pětice

  `(Jmeno, Prijmeni, Ulice, Cislo Popisne, Mesto)`

Příklad dekompozice dat

Aplikace – Diář squashových partnerů.

- Program pro správu kontaktů na různé squashové hráče.
- Hlavní datová struktura je seznam kontaktů.

Datová dekompozice

- Seznam kontaktů – `[Kontakt]`

  `[kontakt1, kontakt2, kontakt3, ..., kontakt315]`
- Kontakt je uspořádaná trojice

  `(Prezdivka, Telefon, Adresa)`
- Adresa je uspořádaná pětice

  `(Jmeno, Prijmeni, Ulice, Cislo Popisne, Mesto)`
- `Prezdivka, Jmeno, Prijmeno, Ulice, Mesto` jsou seznamy znaků
- `Telefon, Cislo Popisne` jsou čísla

Hodnoty a Typy

Hodnoty a typy

Co je to typ a k čemu je

Co je to typ

- Označení množiny všech hodnot dané kvality.
- Komunikační prostředek napomáhající správnému skládání programů z jednotlivých funkcí.

K čemu se používají typy

- Každá hodnota, nebo výraz má svůj typ.
- Definice typové signatury funkcí.
- Kontrola logické konzistence programu v době překladu.
- Popis způsobu kompozice složených datových struktur, protože **typy se komponují podobně jako data.**

Konstrukce typů

Základní datové typy

- `Int, Integer, Float, Char, Bool`

Složené typy

- Uspořádané n-tice:

  `(Bool,Int)`

- Seznamy:

  `[Int], [Char], [[Char]]`

  `[Char]` ≡ `String`

Funkcionální typy

- `Integer -> Bool, Float -> Float -> Float`

Funkcionální typy

Konstrukce

- Jsou-li σ a τ nějaké typy, tak σ `->` τ je typ všech funkcí s parametrem typu σ a funkční hodnotou typu τ.

Typ n-árních funkcí

- Jsou-li σ_1, σ_2, σ_3 $\ldots$ σ_n a τ nějaké typy, tak

  $$\sigma_1 \texttt{ -> } \sigma_2 \texttt{ -> } \sigma_3 \texttt{ -> } \ldots \texttt{ -> } \sigma_n \texttt{ -> } \tau$$

  je typ všech funkcí s prvním parametrem typu σ_1, druhým parametrem typu σ_2, ... a funkční hodnotou typu τ.

Terminologie

- **Arita funkce** označuje počet parametrů funkce.
- Konstanty, unární, binární, ternární funkce.
- Nulární funkce (n=0) jsou konstanty daného typu.

Funkcionální typ a aplikace

Pozorování

- Typ výrazu, který je úplná aplikace funkce na parametry, lze odvodit z typu použité funkce **bez nutnosti výpočtu výsledné hodnoty**.

Příklad

```
odd :: Integer -> Bool
27 :: Integer
odd 27 :: Bool
```

Polymorfní typy

Pozorování

- Některé funkce nepotřebují znát konkrétní typy formálních parametrů, pouze jejich strukturu.
- Místo konkrétního typu se použije **typová proměnná**.
- Při aplikaci funkce na konkrétní parametry, se za typovou proměnnou dosadí typ, který odpovídá použitému parametru. (Typová proměnná se specializuje.)
- **POZOR! Typová proměnná zastupuje i složené typy.**

Příklad

```
fst :: (a,b) -> a
(not,"Coze?") :: (Bool -> Bool,[Char])
fst (not ,"Coze?") :: Bool -> Bool
```

Typové třídy

Pozorování

- Některé funkce nevyžadují konkrétní typ, ale zároveň nedovolují použití libovolného typu, proto je třeba specializaci typové proměnné omezit na vybranou podtřídu typů.

Základní typové třídy

- `Integral` – celočíselné
- `Num` – numerické
- `Ord` – uspořadatelné
- `Eq` – porovnatelné na rovnost

Příklady typů s omezením specializace typové proměnné

```
odd :: Integral a => a -> Bool
(+) :: Num a => a -> a -> a
```

Co není typ

Typ není váš nepřítel

Co není typ

Typ není váš nepřítel

Uspořádané n-tice a seznamy v Haskellu

Uspořádané n-tice v Haskellu

Zápis uspořádaných n-tic

- Přirozený, pomocí závorek a čárek.
- Příklady zápisu uspořádaných n-tic v Haskellu:

```
(12,15)
(2,3,'a',5,6)
("Fiii","jo", 350, "tisíc", '!')
((1,1),(2,2),(3,3))
```

Krajní případy

- Jednotice se nepoužívají.
- Nultice: `()`

Předdefinované funkce pro uspořádané dvojice

Hodnotové konstruktory

- `(,, ... ,)` – hodnotový konstruktor uspořádané n-tice
- `(,)` – hodnotový konstruktor uspořádané dvojice

```
(,) :: a -> b -> (a,b)
(,) x y = (x,y)
```

Projekce

- `fst` , `snd` – projekce na první a druhou složku

```
fst :: (a,b) -> a
fst (x,y) = x

snd :: (a,b) -> b
snd (x,y) = y
```

Seznamy

Zápis seznamů

- V hranatých závorkách uzavřená posloupnost prvků oddělených čárkou.
- Seznam znaků též jako řetězec (text v uvozovkách).

Příklady

```
[3,3,3,3]
[ [1], [1,2], [1,2,3] ]
[]
"ahoj" = ['a','h','o','j']
"toto je také seznam"
[ or, or, or, and ]
```

Konstrukce seznamů

Hodnotové konstruktory

- Prázdný seznam: `[]`

  ```
  [] :: [a]
  ```

- Operátor připojení prvku na začátek seznamu: `(:)`

  ```
  (:) :: a -> [a] -> [a]
  ```

Příklady

- Správné použití

  ```
  (:) 3 [3,3,3]    ⇝   [3,3,3,3]
  1:2:3:[]    ⇝   [1,2,3]
  4:[4,4,4,4]    ⇝   [4,4,4,4,4]
  'A':"hoj"    ⇝   "Ahoj"
  ```

- Nesprávné použití

  ```
  [2] : [3,4,5]    ⇝ ERROR
  [2,3,4] : 5    ⇝ ERROR
  'A' : [1,2,3]    ⇝ ERROR
  ```

Checkpoint

Úkol

- Jaké je implicitní ozávorkování v následujícím výrazu a proč je takové, jaké je?

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

Řešení

Checkpoint

Úkol

- Jaké je implicitní ozávorkování v následujícím výrazu a proč je takové, jaké je?

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

Řešení

- `1:(2:(3:(4:(5:[])))`
- Jiné ozávorkování není možné, nevyhovělo by typové signatuře hodnotového konstruktoru `(:)`.
- Dvojtečka nejvíce vlevo je **nejvnějšnější**.

Konstrukce seznamů – pokračování

Funkce pro spojení seznamů

- Seznamy **stejného typu** lze spojit pomocí funkce `(++)`

```
(++) :: [a] -> [a] -> [a]
```

Příklady

- Správné použití

```
(++) "Ahoj " "světe!"  ⇝  "Ahoj světe!"
"Ahoj" ++ " " ++ "světe!"  ⇝  "Ahoj světe!"
[1,2,3] ++ [4,5,6]  ⇝  [1,2,3,4,5,6]
```

- Nesprávné použití

```
2 ++ [3,4,5]  ⇝ ERROR
[2,3,4] ++ 5  ⇝ ERROR
[2,3] ++ "text"  ⇝ ERROR
```

Význam hodnotových konstruktorů

Hodnotové konstruktory ve víceřádkových definicích

- Fungují jako vzory na levých stranách definice.
- Mapují se vždy na **nejvnějšnější** výskyt.

Příklady

- Funkce `null` aplikovaná na nějaký seznam, vrací `True` pokud je seznam prázdný a `False` pokud je neprázdný.

```
null :: [a] -> Bool
null (_:_) = False
null [] = True
```

- Funkce `snd` aplikovaná na uspořádanou dvojici, vrací druhý prvek dvojice.

```
snd :: (a,b) -> b
snd (_,y) = y
```

Pojmenovaný vzor – symbol @

Použití symbolu @

- Pokud `vzor` je korektně vytvořený datový vzor, pak zápisem `jmeno@vzor` získáme proměnnou `jmeno`, která bude po úspěšném použití vzoru odkazovat na celý mapovaný obsah.
- Nejčastěji používané ve spojení se seznamy, ale funguje všeobecně pro jakékoliv hodnoty konstruované s využitím hodnotových konstruktorů.

Příklady

- Při mapování seznamu `[1,2,3]` na vzor `a@(x:t)`, bude

  `a = [1,2,3],` `x = 1,` `t = [2,3]` .

- `f (a@(x:y)) = x:a++y`

  `f [1,2,3]` $\rightsquigarrow^*$ `[1,1,2,3,2,3]`

  `f []` $\rightsquigarrow^*$ **ERROR**

Checkpoint

Úkoly

- Napište funkci, která vrátí druhý prvek seznamu, pokud je aplikována na seznam délky alespoň dva.