

Vlastní datové typy a aliasy; Map, Set; podpůrné nástroje Cabal a HLint

IB016 Seminář z funkcionálního programování

Vladimír Štill, Martin Ukrop

Fakulta informatiky, Masarykova univerzita

Jaro 2017

Vlastní datové typy: opakování

```
data BinTree a = BNode a (BinTree a) (BinTree a)
               | BEmpty
               deriving (Eq, Read, Show)
```

- definuje parametrizovaný typ binárního stromu

Vlastní datové typy: opakování

```
data BinTree a = BNode a (BinTree a) (BinTree a)
                | BEmpty
                deriving (Eq, Read, Show)
```

- definuje parametrizovaný typ binárního stromu
- na získávání hodnot používáme vzory

```
isEmpty BEmpty = True
isEmpty _      = False
```

```
btValue (BNode v _ _) = v
btLeft  (BNode _ l _) = l
btRight (BNode _ _ r) = r
```

Vlastní datové typy: záznamy

```
data BinTree a = BNode { btValue :: a
                        , btLeft  :: BinTree a
                        , btRight :: BinTree a
                        }
  | BEmpty
  deriving (Eq, Read, Show)
```

- v definici typu pojmenováváme jednotlivé hodnoty

Vlastní datové typy: záznamy

```
data BinTree a = BNode { btValue :: a
                        , btLeft  :: BinTree a
                        , btRight :: BinTree a
                        }
  | BEmpty
  deriving (Eq, Read, Show)
```

- v definici typu pojmenováváme jednotlivé hodnoty
- názvy hodnot lze použít ve vzorech i jako funkce

```
foo1 (BNode { btValue = val, btLeft = l }) = ...
foo2 node = btValue node + ...
```

- je možné modifikovat i podmnožinu atributů záznamu

```
const1tree node = node { btValue = 1 }
```

Typové aliasy

```
type String = [Char]
type Matrix a = [[a]]
```

jen nové pojmenování, lze zaměňovat s původním typem

- až na drobné výjimky: instance typových tříd
- funguje například `map (map (+4)) matrix`, kde `matrix :: Matrix Int`
- bez `deriving`

```
newtype Matrix a = M { unM :: [[a]] } deriving Show
```

v podstatě definice nového typu (jako `data`), ale musí mít právě jeden unární datový konstruktor

- typově rozlišitelné
- `M (map (map (+4)) (unM matrix))`
- rychlejší než `data`, další rozdíly s rozšířeními GHC¹

¹nad rámec kurzu: `-XGeneralizedNewtypeDeriving`

Typové díry (Typed Holes)

- otypování přes `:t` již znáte
- co když ale chceme vědět typ nějakého podvýrazu ve složitějším výrazu?

Typové díry (Typed Holes)

- otypování přes `:t` již znáte
- co když ale chceme vědět typ nějakého podvýrazu ve složitějším výrazu?
- typed holes v GHC (> 7.8) jsou výrazy začínající podtržítkem, které vygenerují typovou chybu obsahující popis typu, který by výraz na tomto místě měl mít

```
> [1, 2, _]
```

```
<interactive>:3:8: error:
```

```
* Found hole: _ :: t
```

```
  Where: 't' is a rigid type variable bound by
```

```
         the inferred type of
```

```
         it :: Num t => [t]
```

```
         at <interactive>:3:1
```

```
...
```

- v GHC 7.10 chybí kontext u typů

Typové díry II

- lze využít při prototypování programu namísto `undefined`
- typové chyby z typových děr lze odložit až do okamžiku kdy se pokusíme díru zavolat pomocí přepínače `GHC/GHCi -fdefer-typed-holes`
- pozor, i stejně pojmenované typové díry jsou *různé* typové díry
- více v [dokumentaci GHC](#) a na [GHC wiki](#)

asociativní mapy a množiny

- moduly `Data.Map` a `Data.Set` součástí běžné distribuce GHC (balík `containers`)
- `Map k v`
 - `k` je typ klíče, musí být `Ord`
 - `v` je typ hodnoty
- `Set a`
 - `a` je typ hodnoty, musí být `Ord`
- logaritmické vkládání, odstraňování, zjišťování minima a maxima
- jsou instancemi `Foldable`, `Traversable`

Datové typy Map a Set – ukázka

- kvalifikovaný import

```
import Data.Map (Map)
import qualified Data.Map as Map
```

```
foo :: Integral i => Map k i -> Map k i
foo = Map.filter even
```

- Set, Map definují mnoho funkcí se stejným názvem jako Prelude
 - tečka musí být bez mezery (přístup k funkci v modulu)
- prázdná mapa

```
let mapa1 = empty :: Map k a
```

- přidání prvků

```
let mapa2 = insert 5 "Karel" mapa1
```

nástroj pro správu balíčků v Haskellu

- součást Haskell Platform, nebo samostatný balík
- aktualizace databáze balíčků: `cabal update`
- instalace balíku z **Hackage**: `cabal install <package>`
- Pozor, `cabal uninstall` neexistuje!
 - možnost instalovat do sandboxů od verze 1.18
 - `cabal sandbox init`, `cabal sandbox repl`
- všechno ukládá do `$HOME/.cabal/`
 - konfiguraci a instalace je možno snadno smazat

Cabal: vytváření balíčků

- 1 vytvořit moduly, včetně všech potřebných importů (kvůli závislostem)
- 2 `cabal init` v kořenovém adresáři projektu
- 3 vyplnit dotazované, speciálně, zda se jedná o knihovnu nebo o binárku
- 4 vytvoří soubory `jmeno_projektu.cabal`, `Setup.hs` a `LICENSE`
 - `.cabal` obsahuje definici projektu, možné měnit závislosti, obsažené moduly a další...
 - zbytek většinou nepodstatný
- 5 případně relaxovat požadované závislosti

více na <https://www.haskell.org/cabal/users-guide/developing-packages.html>

nástroj hlásící návrhy na zlepšení kódu

- samostatný balík z Hackage, nutno doinstalovat
 - často dostupný přímo v repozitářích distribuce
 - FI PC: nainstalovaný z repozitářů Ubuntu
 - více podrobností v samostatném návodu v ISu
- závisí na generátoru parsrů Happy
- `cabal install happy hlint`
- `hlint [--hint <extra-definice>] <zdrojový-kód>`
- možno integrovat přímo do GHCi, viz návod v ISu
- soubor s extra definicemi v ISu

- GHC při kompilaci může vypisovat kromě chyb i varování
- varování se zapíná argumentem při volání GHC, resp. `:set <flag>` v GHCi
 - `-W` zapíná větší sadu varování
 - `-Wall` zapíná většinu varování
 - `-w` vypíná všechna varování
 - `-Werror` vyhodnocuje varování jako chyby
- úplný přehled varování v sekci dokumentace [4.8 Warnings and sanity checking](#)

- GHC při kompilaci může vypisovat kromě chyb i varování
- varování se zapíná argumentem při volání GHC, resp. `:set <flag>` v GHCi
 - `-W` zapíná větší sadu varování
 - `-Wall` zapíná většinu varování
 - `-w` vypíná všechna varování
 - `-Werror` vyhodnocuje varování jako chyby
- úplný přehled varování v sekci dokumentace [4.8 Warnings and sanity checking](#)

Standard pro domácí úkoly IB016 je čistý překlad s `-Wall`!

- možno přidat `:set -Wall` do `~/ghc/ghci.conf`

Samostatný úkol 1: cabal

- 1 s pomocí Hayoo zjistěte v jakém balíku se nachází modul `Data.Default`
- 2 tento balík si nainstalujte `cabalem`
- 3 implementujte nějakou instanci typové třídy `Default` pro datový typ

```
data BinTree a = BNode a (BinTree a) (BinTree a)
               | BEmpty
               deriving (Eq, Show, Read)
```

Samostatný úkol 2: telefonní databáze

- 1 vytvořte vhodný typ pro telefonní databázi
pro každého člověka si chcete pamatovat jméno, telefonní číslo a město
- 2 zdefinujte základní funkce pro práci s databází:
zobrazení, vyhledávání, vkládání, mazání
praktické jsou také funkce, které načtou/vypíší celou databázi ze/do seznamu
- 3 kód komentujte a kontrolujte `hlint`-em