Práce se vstupem a výstupem (do-notace, operátor »=)

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Opakování

Je definován vlastní datový typ popisující počet bonbónů v balíčku:

```
data BonPari = BonPari Int deriving Show
```

Napište funkci
`applyOnBonPari :: BonPari -> (Int -> Int) -> BonPari`,
která bere jako argumenty balíček bonbonů a funkci, která s počtem bonbonů manipuluje (například odebere 5 bonbonů, přidá 2, atd.). Zkuste si v interpretru zřetězit za sebe pár `applyOnBonPari` funkcí.

```
applyOnBonPari (applyOnBonPari (BonPari 2) (+1))
(subtract 4) ⇝* BonPari (-1)
```

Užitečné funkce: `show`, `read`

```
show :: Show a => a -> String
```

- funkce, která převede svůj argument na řetězec
- možno pouze pro typy ze třídy `Show`
  - standardní instance pro čísla, seznamy, n-tice, ...

Užitečné funkce: `show`, `read`

```
show :: Show a => a -> String
```

- funkce, která převede svůj argument na řetězec
- možno pouze pro typy ze třídy `Show`
  - standardní instance pro čísla, seznamy, n-tice, ...

```
read :: Read a => String -> a
```

- funkce, která „přečte“ něco z řetězce (parsuje řetězec)
- možno pouze pro typy ze třídy `Read`
  - standardní instance pro čísla, seznamy, n-tice, ...
- při selhání shodí program s výjimkou
  `Prelude.read: no parse`
- může být potřeba explicitně uvést typ, který chceme přečíst
  - typ odvozen z kontextu: `read "True" || False`
  - `(read "1")` vs `(read "1" :: Int)`

Vstup a výstup v Haskellu

Běžné funkce v Haskellu nemají vedlejší efekty ani vnitřní stav:

- nesmí modifikovat soubory, vypisovat na obrazovku, ...

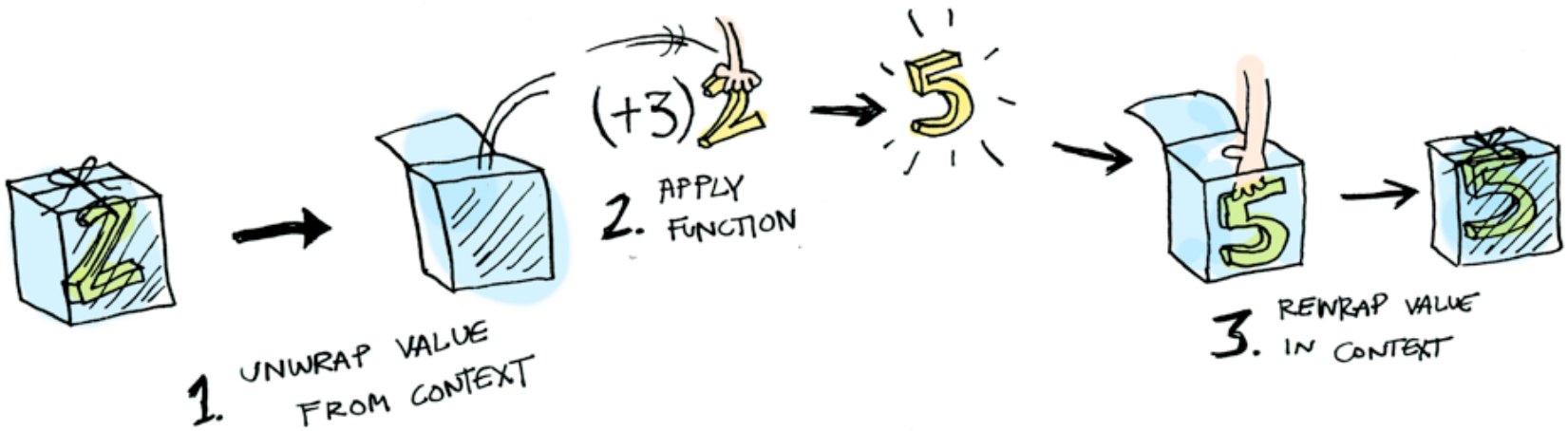
Jak komunikovat se světem (např. načíst vstup)?

Vstup a výstup v Haskellu

- speciální `IO` funkce, mají povoleny vedlejší efekty
  - `putStrLn :: String -> IO ()`
  - `getLine :: IO String`
  - ...
- hodnota typu `IO a`
  - (vstupně-výstupní) **akce**, má **vnitřní výsledek** typu `a`
- dva různé způsoby zápisu práce s `IO` funkcemi
  - `do`-notace
  - bind-operátor (funkce >>= a >>)

IO jako krabice

- k vnitřnímu výsledku IO akcí nelze přistupovat přímo, používáme speciální jazykové konstrukce
- z IO nelze „utéct“



© 2013 Aditya Bhargava, *Functors, Applicatives, And Monads In Pictures*

IO: do-notace

```
echo :: IO ()
echo = do
    putStrLn "Write something."
    line <- getLine
    let out = "You wrote: " ++ line
    putStrLn out
```

- návratovou hodnotu akce `getLine` extrahujeme pomocí `<-`
- blok musí končit akcí, její návratová hodnota je návratovou hodnotou bloku
- `let` uvnitř `do` platí od své definice až po konec bloku
- pozor na zarovnání

IO: do-notace

Příklad 8.2.1: Naprogramujte funkci `compareInputs :: IO ()`, která od uživatele načte dva řetězce a pak vypíše na obrazovku delší z nich.

Přehled užitečných IO funkcí

- `putStr :: String -> IO ()`
  vypíše zadaný řetězec na obrazovku
- `putStrLn :: String -> IO ()`
  vypíše zadaný řetězec na obrazovku a zalomí řádek
- `print :: Show a => a -> IO ()`
  vypíše na výstup hodnotu zobrazitelného typu (typu, který je instancí `Show`)
- `getLine :: IO String`
  načte ze vstupu řádek (řetězec)
- `pure :: a -> IO a`
  zabalí hodnotu do `IO` kontextu (nevykoná žádnou akci)

do-notace: příklady

Příklad 8.2.2: Definujte akci `getInt :: IO Int`, která ze standardního vstupu načte celé číslo. Využijte knihovní funkci `read :: (Read a) => String -> a`.

Můžete využít funkce `pure :: a -> IO a`, která zabalí hodnotu do `IO` akce, která vrací tuto hodnotu.

do-notace: příklady

Příklad 8.2.3: Upravte a doplňte následující zdrojový kód tak, aby program vyžadoval a načetl postupně tři celá čísla a o nich určoval, zda mohou být délkami hran trojúhelníku.

```
main :: IO ()
main = do putStrLn "Enter one number:"
          x <- getInt
          putStrLn (show (1 + x))
```

IO pomocí operátoru >>=

do-notace je syntaktickým cukrem pro tzv. operátory *bind*:[1]

- `(>>) :: IO a -> IO b -> IO b`
  pro řetězení akcí (bez použití výsledku první)
- `(>>=) :: IO a -> (a -> IO b) -> IO b`
  umožňuje přistoupit k vnitřnímu výsledku akce z funkce, která vrací akci

```
echo =
    putStrLn "Write something: " >>
    getLine >>= putStrLn . ("You wrote: " ++)
```

[1] Uvedené operátory jsou ve skutečnosti obecnější, jsou definované pro všechny *monády*. Pro kurz IB015 však můžete jakoukoliv `Monad m` považovat za `IO`. Více o *monádách* například v IB016 Seminář z funkcionálního programování.

Převod mezi notacemi

do-notace	***bind*-notace**
`do f` `  g`	`f >> g`
`do x <- f` `  g`	`f >>= \x -> g`
`do let x = y` `  f`	`let x = y in f`

Příklad převodu:

```
do putStr ">>> "
   s <- getLine
   let t = reverse s
   putStrLn t
```

```
putStr ">>> " >>
getLine >>= \s ->
let t = reverse s in
putStrLn t
```

IO pomocí >>=: příklady

Příklad 8.3.1: Převeďte následující program v do-notaci na notaci s použitím >>=.

```
main = do
    f <- getLine
    s <- getLine
    appendFile f (s ++ "\n")
```

IO pomocí >>=: příklady

Příklad 8.3.2: Napište program, který načte jeden řádek textu od uživatele, ze kterého pak odstraní všechny znaky, které nejsou znaky abecedy. Výsledek následně vypíše na výstup. V úkolu použijte funkci `isAlpha` z modulu `Data.Char`.

IO: Samostatně spustitelné programy

S pomocí `IO` lze vytvářet spustitelné programy v Haskellu:

- stačí definovat funkci `main :: IO ()`
- tato funkce se použije jako vstupní bod programu

Spouštění a kompilace:

- kompilace: `ghc File.hs`, vytvoří binárku `File/File.exe`
- program lze spustit pomocí `./File`

IO se soubory

- `readFile :: FilePath -> IO String`
  načte obsah souboru do řetězce
- `writeFile :: FilePath -> String -> IO ()`
  zapíše řetězec do souboru (původní obsah se přepíše)
- `appendFile :: FilePath -> String -> IO ()`
  zapíše řetězec na konec souboru

`FilePath` je typové synonymum (typový alias) pro `String`

IO se soubory: příklady

Příklad 8.2.4: Napište program, který vyzve uživatele, aby zadal jméno souboru, poté ověří, že zadaný soubor existuje, a pokud ano, vypíše jeho obsah na obrazovku, pokud ne, informuje o tom uživatele. Úkol řešte s využitím `doesFileExist` z modulu `System.Directory`

IO se soubory: příklady

Příklad 8.3.7: Vymyslete a naprogramujte několik triviálních prográmků manipulujících s textovými soubory:

- počítání řádků
- výpis konkrétního řádku podle zadaného indexu
- vypsání obsahu pozpátku
- seřazení řádků, ...

Definice alternativně přepište s a bez pomoci syntaktické konstrukce `do`.