Episode III

Recursion of the Sith

Vad har vi lärt oss?

- Funktioner
  - `plusOne x = x + 1`
- Typer
  - `plusOne :: Num a => a → a`
- Listor
  - `(0 : [1,2,45]) ++ [10 .. 15]`
- Mönstermatchning och vakter
  - ```
    first (x:xs) | x > 10 = “First element >10”
    first []             = “No elements!”
    ```
    ```

Vad har vi lärt oss? (forts.)

- `case` och `where`
  -
```
numberToString x =
      “Talet är “ ++ num
    where
      num = case plusOne x of
              0 → “noll“
              n → show n
```
- Hembakta datatyper
  - `data Pet = Cat | Dog | Rock`
  - `type Age = Int`

Rekursiva typer och funktioner

- Funktioner kan referera till sig själva – rekursion
  - ```
    sum (x:xs) = x + sum xs
    sum []     = 0
    ```
- Typer också – rekursiva typer
  - ```
    data List a
       = Empty
       | OneMore a (List a)
    ```

Övning: modellera ett släktträd

- Fundera: hur ser ett släktträd ut?
- Rita gärna diagram!
- Uttryck strukturen som en egen datatyp
- Försök använda både `type`, `data`, typvariabler och records!
  - Hint för typvariabler: olika tillfällen kräver olika info om personer – ibland bara ett namn, ibland namn + ålder, etc.
- Försök uttrycka ditt (eller någon annans) släktträd med hjälp av din typ – fungerar det?

Högre ordningens funktioner

- Första ordningens funktioner
  - Opererar på ickefunktioner
  - `addOne :: Int → Int`
- Andra ordningens funktioner
  - Opererar på första ordningens funktioner
  - `forEach :: [a] → (a → b) → [b]`
- Tredje ordningens funktioner
  - Opererar på andra ordningens funktioner
  - `stuff :: ((a -> b) -> c) -> (a -> b) -> c`

Bygga funktioner

- Funktioner utan namn - “anonyma funktioner”
  - `forEach [1 .. 10] (\x → x + 5)`
  - `pairwise [1 .. 10] (\x y → x * y)`
- Partiell applikation
  - `plus x y = x + y`
  - `forEach [1 .. 10] (plus 5)`
- Komposition
  - `compose :: (a → b) → (b → c) → (a → c)`
  - `compose f g = g . f`

Standardbiblioteket

- `map :: (a → b) → [a] → [b]`
- `zip :: [a] → [b] → [(a, b)]`
- `zipWith :: (a → b → c) → [a] → [b] → [c]`
- `foldl :: (a → b → a) → a → [b] → a`
- ...och mycket mer!
  - http://www.haskell.org/hoogle/

Interagera med omvärlden

- Haskell har orena funktioner också!
  - ...men typsystemet begränsar deras användning
- `putStrLn :: String → IO ()`
  - Accepterar en String, utför I/O och lämnar tillbaka ()
- `getLine :: IO String`
  - Utför I/O, och lämnar tillbaka `String`

Kombinera I/O

```
greet :: String → IO ()
greet person = putStrLn (“Hello, “ ++ person)


conversation :: IO ()
conversation = do
  putStrLn “What is your name?”
  name ← getLine
  greet name
```

Exempel: 20 frågor

- Tänk på ett djur, svara på datorns frågor
  - Vem vinner denna giganternas kamp?!
- Använder (nästan) allt vi lärt oss hittills
  - Datatyper
  - Rekursion
  - Rekursiva datatyper
  - Högre ordningens funktioner
  - I/O

Så mycket mer!

- Autogenerera testfall med QuickCheck
- Parallellisera din kod genom att ändra en rad
- Ersätt `if` och `case` med do-notation
- Kör Haskell i webbläsaren
- Skriv en Java-kompilator