Typové třídy, akumulační funkce na seznamech

IB015 Neimperativní programování

Kolektiv cvičících IB015

Fakulta informatiky, Masarykova univerzita

podzim 2018

Opakování

Mějme datový typ `Shape` definovaný z minulého cvičení.

```
data Shape = Circle Double
           | Rectangle Double Double
           | Point
           deriving Show
```

Naprogramujte následující funkce

- `isEqual :: Shape -> Shape -> Bool`, která vrátí True, právě tehdy, když jsou si oba argumenty rovny.
- `isGreater :: Shape -> Shape -> Bool`, která vrátí True, pokud je první argument větší než druhý (Shape je vetší než druhý, když má větší plochu).

Typové třídy

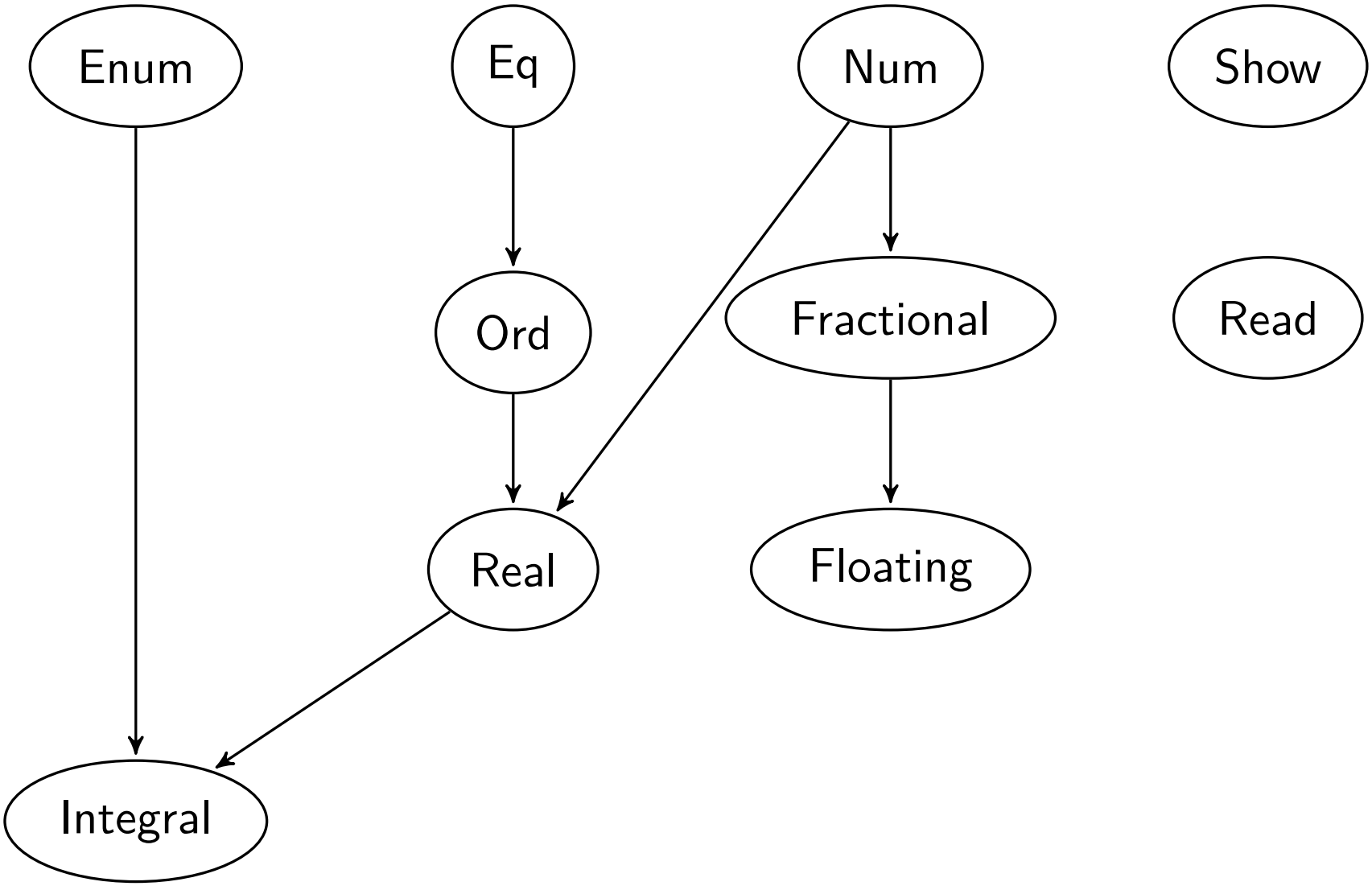
- vyjadřují nějakou vlastnost hodnot
  (porovnatelnost, převeditelnost na text, . . .)
- obsahují funkce, které implementují tuto vlastnost
  (porovnávání, . . .)
- typ patří do třídy $\sim$ instance typové třídy
  - automatické instance (`deriving`)
  - ruční instance (pozor na odsazení!)

```
data Shape = P | C Double | R Double Double
instance Eq Shape where
    P == P                   = True
    (C r1) == (C r2)         = r1 == r2
    (R x1 y1) == (R x2 y2) = x1 == x2 && y1 == y2
    _ == _                   = False
```

Typové třídy

- často lze některé funkce odvodit z jiných
  - `(/=)` z `(==)`
  - `(>)`, `(<)`, `(>=)` z `(<=)` a `(==)`
- *minimální kompletní implementace*: skupina funkcí, které musíme implementovat, aby typová třída fungovala
  - pro `Eq`: `(==)`
  - pro `Ord`: `(<=)`
- základní informace o typové třídě vypíše interpret při povelu `:info <nazev-tridy>`
- typové třídy jsou v hierarchii
  - instance `Ord` vyžaduje instanci `Eq`
  - část hierarchie následuje
- při použití přepínače `:set -Wincomplete-patterns` bude ghci hlásit vzory nepokryté případy

Hierarchie základních typových tříd



Typové třídy – příklad

Příklad 6.1.2: Uvažte datový typ představující semafor zadefinovaný níže.

```
data TrafficLight = Red | Orange | Green
                  deriving (Show)
```

Umožněte vzájemné porovnávání a řazení (zelená $<$ oranžová $<$ červená) hodnot tohoto typu. Řečeno jinak, napište instanci `TrafficLight` pro typové třídy `Eq` a `Ord`.

Typové třídy – příklad

Příklad 6.1.3: Zadefinujme vlastní typ uspořádaných dvojic s názvem `PairT`. Tento typ bude mít pouze jeden binární hodnotový konstruktor `PairV` (viz definice níže).

```
data PairT a b = PairV a b
```

Vytvořte instanci `PairT` pro typovou třídu `Show`. Zobrazování hodnot tohoto typu nechť je slovní (tedy namísto obligátního `(1,2)` vypište třeba `"pair of 1 and 2"`).

Typová třída Num

- sdružuje všechna čísla - proto je poněkud komplikovaná, ale pro nás jsou hlavní dvě její podtřídy
  - `Integral`, která v sobě zahrnuje celá čísla
    - obsahuje typy `Int` a `Integer`, které již dobře znáte
  - `Floating`, která zahrnuje čísla s plovoucí desetinou čárkou
    - patří sem typy `Float` a `Double`

Užitečná je také funkce
`fromInteger :: Num a => Integer -> a`[1] , která jako argument bere celé číslo typu `Integer` a vrací obecnější číslo typu `Num`.

[1] Haskell interně reprezentuje celočíselné literály jako `(fromInteger (??? :: Integer))`, což umožňuje tyto literály interpretovat v různých datových typech. Mají tedy typ `Num a => a`. Obdobně to platí pro funkci `fromRational :: Fractional a => Rational -> a`.

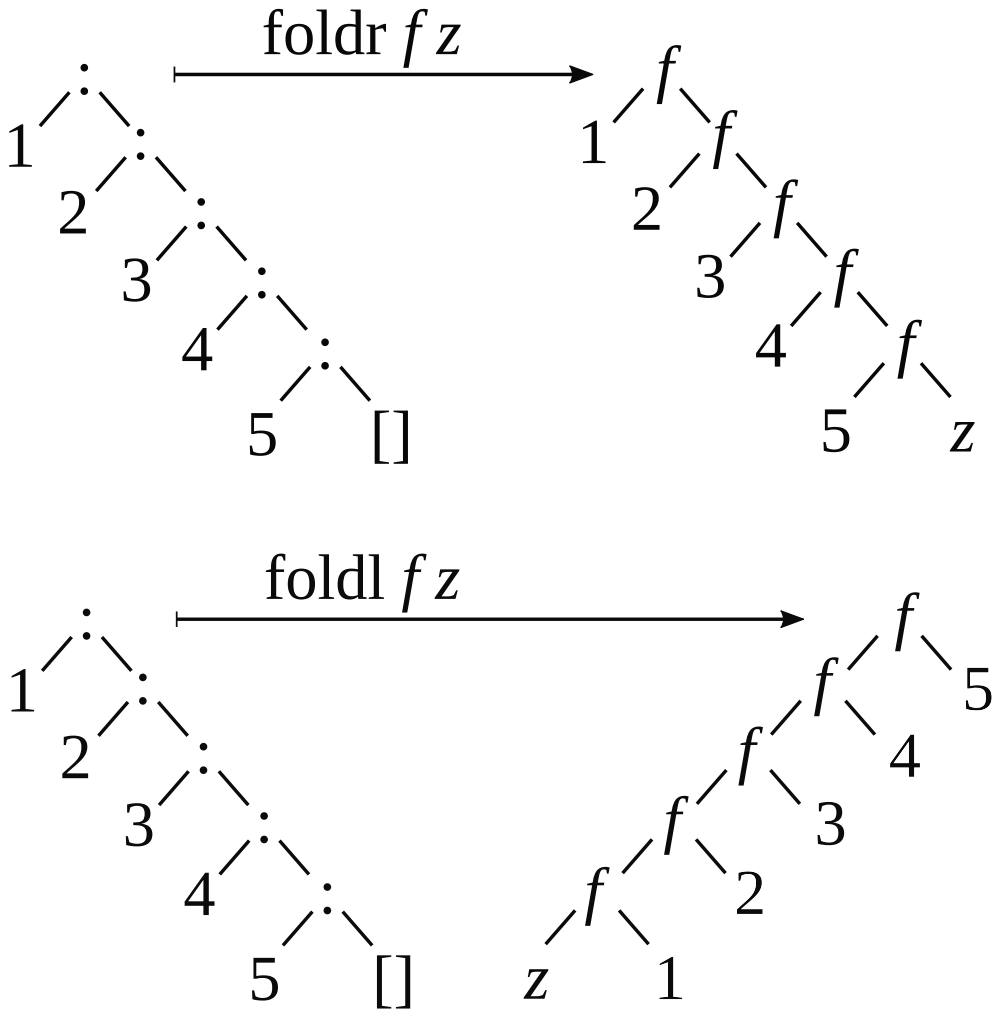
Akumulační funkce na seznamech – motivace

Příklad 6.2.1: Definujte následující funkce rekurzivně:

a) `product'` – součin prvků seznamu
b) `length'` – počet prvků seznamu
c) `map'` – funkci `map`

Co mají tyto definice společné? Jak by vypadalo jejich zobecnění?

Akumulační funkce na seznamech – ilustrace



Zdroj: Haskell wiki (`https://wiki.haskell.org/Fold`)

Akumulační funkce na seznamech

Seznamové akumulační funkce:[2]

- zpracují seznam na jednu hodnotu
- transformace seznamu dle struktury

- `foldr :: (a -> b -> b) -> b -> [a] -> b`
- `foldl :: (b -> a -> b) -> b -> [a] -> b`

- `foldr1 :: (a -> b -> b) -> [a] -> b`
- `foldl1 :: (b -> a -> b) -> [a] -> b`

„Praktická ukázka“ (vtip): `http://foldr.com/`

[2] Uvedené funkce jsou ve skutečnosti obecnější, jsou definované pro `Foldable t`. Pro kurz IB015 však můžete jakoukoliv `Foldable t` považovat za seznam (`[]`). Více o *foldable* například v IB016 Seminář z funkcionálního programování.

Akumulační funkce na seznamech

Zhluboka se nadechněte, ve sbírce si otevřete příklad 6.2.8 (implementace funkcí pomocí akumulačních funkcí) a řešte jednotlivé podpříklady.