Rozšíření typových tříd

```
class Collection col elem where
  empty::col elem
  contains::elem->col elem->Bool
```
Víceparametrické třídy

Funguje dobře, dokud nebudeme chtít mít množinu čísel reprezentovanou v Integeru pomocí bitů.
Pak totiž v typu `empty` nedává `Integer Int` smysl.

```
class Collection colelem elem where
  empty::colelem
  contains::elem->colelem->Bool
```
Tohle způsobí kompilační chybu při prvním použití

Řešením jsou funkční závislosti...
```
lass Collection colelem elem | colelem->elem where...
instance Collection [a] a where...   instance Collection Integer Int where...
```

... nebo typové třídy, GHC 6.10
```
class Collection col where
  type ColElem col
  empty :: col
  contains :: col -> ColElem col -> Bool
instance Collection Integer where
  type ColElem Integer = Int
  ...
```

Datové struktury

```
module Data.Set
  data Set a = ...
  empty :: Set a
  singleton :: a -> Set a
  null :: Set a -> Bool
  size :: Set a -> Int
  member :: Ord a => a -> Set a -> Bool
  insert :: Ord a => a -> Set a -> Set a
  delete :: Ord a => a -> Set a -> Set a
  find{Min,Max} :: Set a -> a
  delete{Min,Max} :: Set a -> Set a
  deleteFind{Min,Max} :: Set a -> (a, Set a)
  toList, toAscList :: Set a -> [a]
  fromList :: Ord a => [a] -> Set a
  fromAscList :: Eq a => [a] -> Set a
  fromDistinctAscList :: [a] -> Set

module Data.Map
  data Map k a
  empty :: Map k a
  singleton :: k -> a -> Map k a
  null, size, member ::
  lookup :: Ord k => k -> Map k a -> Maybe a
  insert :: Ord k => k -> a -> Map k a -> Map k a
  delete :: Ord k => k -> Map k a -> Map k a
  adjust :: Ord k => (a -> a) -> k -> Map k a -> Map k a
  find{Min,Max} :: Map k a -> (k, a)
  delete{Min,Max} :: Map k a -> Map k a
  deleteFind{Min,Max} :: Map k a -> ((k, a), Map k a)
  update{Min,Max} :: (a -> Maybe a) -> Map k a -> Map k a
  toList, toAscList :: Map k a -> [(k, a)]
  fromList :: Ord k => [(k, a)] -> Map k a
  fromAscList :: Eq k => [(k, a)] -> Map k a
  fromDistinctAscList :: [(k, a)] -> Map k a

module Data.IntSet

module Data.IntMap
```

```
                              Fronty
                              ------
module Queue (Queue(..)) where
class Queue q where
    none::q a
    empty::q a->Bool

    head::q a->a
    tail::q a->q a
    snoc::a->q a->q a

                          Klasická fronta
                          ---------------
module Classic (Classic,none) where
import Queue(Queue); import qualified Queue

data Classic a = Classic [a] [a] deriving Show
none::Classic a; none=Queue.none

check (Classic [] t) = Classic (reverse t) []
check otherwise = otherwise

instance Queue Classic where
    none = Classic [] []
    empty (Classic h _) = null h

    head (Classic (h:_)  _) = h
    tail (Classic (_:hs) t) = check $ Classic hs t
    snoc x (Classic h t)    = check $ Classic h (x:t)
```