Haskell Unit 9: Datatypes

Antoni Diller

26 July 2011

Introduction

The following definitions, relating to recursive datatypes, can be found in the file:

~ard/teaching/workshop/public/types.hs

First, I will consider enumerated types in Haskell.

```
data Day = Sun | Mon | Tue | Wed | Thu | Fri | Sat
           deriving (Eq,Ord,Enum,Show)

fromDay :: Day -> Int
fromDay = fromEnum

toDay :: Int -> Day
toDay = toEnum

weekend :: Day -> Bool
weekend Sun = True
weekend Sat = True
weekend _   = False
```

Second, I will consider composite types in Haskell.

```
data Tag = Tagi Int | Tagb Bool
           deriving (Eq,Ord,Show)

isInt :: Tag -> Bool
isInt (Tagi _ ) = True
isInt (Tagb _ ) = False
```

Third, I will consider recursive types in Haskell, in particular, binary trees.

```
data Btree a = Tip a | Bin (Btree a) (Btree a)
               deriving (Eq,Ord,Show)

tree1 = Bin (Tip 1) (Bin (Tip 6) (Tip 5))

size :: Btree a -> Int
size (Tip _)   = 1
size (Bin s t) = size s + size t

depth :: Btree a -> Int
depth (Tip _)   = 0
depth (Bin s t) = 1 + max (depth s) (depth t)

flatten :: Btree a -> [a]
flatten (Tip x)     = [x]
flatten (Bin xt yt) = flatten xt ++ flatten yt

inodes :: Btree a -> Int
inodes (Tip _)     = 0
inodes (Bin xt yt) = 1 + inodes xt + inodes yt

maxBtree :: Ord a => Btree a -> a
maxBtree (Tip x)     = x
maxBtree (Bin xt yt) = max (maxBtree xt) ( maxBtree yt)
```

It is possible to defines functions on binary trees analogous to the `map` and `fold` functions defined on lists.

```
mapBtree :: (a -> b) -> Btree a -> Btree b
mapBtree f (Tip x) = Tip (f x)
mapBtree f (Bin xt yt)
  = Bin (mapBtree f xt) (mapBtree f yt)

foldBtree :: (a -> b) -> (b -> b -> b) -> Btree a -> b
foldBtree f g (Tip x) = f x
foldBtree f g (Bin xt yt)
  = g (foldBtree f g xt) (foldBtree f g yt)
```

```
size1 :: Btree a -> Integer
size1 = foldBtree (const 1) (+) where const x y = x

depth1 :: Btree a -> Integer
depth1 = foldBtree (const 0) op
         where op m n = 1 + max m n; const x y = x

flatten1 :: Btree a -> [a]
flatten1 = foldBtree wrap (++) where wrap x = [x]

maxBtree1 :: Ord a => Btree a -> a
maxBtree1 = foldBtree id max

mapBtree1 :: (a -> b) -> Btree a -> Btree b
mapBtree1 f = foldBtree (Tip . f) Bin
```