Data Declarations to Class Definitions

Daniel Díaz

October 2010

First version

Contents

1 Summary 2

2 Construction 2
   2.1 Construction notes ........................................... 3

3 Examples 3
   3.1 Maybe example ................................................... 3
   3.2 Record example ................................................... 4
   3.3 Recursive example ............................................... 5
   3.4 Mixed example .................................................... 6

4 Final notes from the author 7
1 Summary

These notes explain how a data declaration can be transformed to a class
definition, preserving the meaning of the original type. The method here
explained is implemented in the DTC (Data To Class) package, which you can
found in Hackage.

2 Construction

Given a data declaration:

\[
data T \ v_1 \ldots \ v_n = C_1 \ a_{i_1}^1 \ldots a_{n_1}^1 | \ldots | C_m a_{i_m}^m \ldots a_{n_m}^m
\]

we can construct the following class definition:

```haskell
class T t where
  c_1 :: \ a_{i_1}^1 -> \ldots -> a_{n_1}^1 -> t \ v_1 \ldots \ v_n
  \ldots
  c_m :: \ a_{i_m}^m -> \ldots -> a_{n_m}^m -> t \ v_1 \ldots \ v_n
  d_1 :: t \ v_1 \ldots \ v_n -> (a_{i_1}^1, \ldots, a_{n_1}^1)
  \ldots
  d_m :: t \ v_1 \ldots \ v_n -> (a_{i_m}^m, \ldots, a_{n_m}^m)
```

If T is a recursive type, one or more \( a_i^j \) are equal to T. When this happens,
each one is replaced by \( t \).
2.1 Construction notes

Since we have \( m \) data constructors in the data declaration of \( T \), we have \( m \) constructor functions in the \( T \) class definition, each one represented by \( c_i \), with \( i = 1, \ldots, m \). Deconstructors (represented by \( d_i \)) are only built if the correspondent data constructor have one or more arguments.

3 Examples

Using the DTC package we can see some examples.

3.1 Maybe example

Given the original source code:

```hs
module MaybeExample where

data Maybe a = Just a | Nothing
```

We obtain the following module:

```hs
module MaybeExample where

class Maybe m where

  just :: a -> m a

  fromJust :: m a -> a

  nothing :: m a
```
3.2 Record example

Given the original source code:

module RecordExample where

data Point = Point { pointX :: Int, pointY :: Int }

We obtain the following module:

module RecordExample where

class Point p where

    point :: Int -> Int -> p

    pointX :: p -> Int

    pointY :: p -> Int
3.3 Recursive example

Given the original source code:

```haskell
module RecursiveExample where

data Tree a b = Leaf b | Node (Tree a b) a (Tree a b)
```

We obtain the following module:

```haskell
module RecursiveExample where

class Tree t where

    leaf :: b -> t a b

    fromLeaf :: t a b -> b

    node :: t a b -> a -> t a b -> t a b

    fromNode :: t a b -> (t a b, a, t a b)
```
3.4 Mixed example

Given the original source code:

```haskell
module MixedExample where

data Mixed a b c = Null | Record { comp1 :: a , comp2 :: Int }
                         | One b | Rec c (Mixed a b c) (Mixed a c b)

We obtain the following module:

module MixedExample where

class Mixed t where

  null :: t a b c

  record :: a -> Int -> t a b c

  comp1 :: t a b c -> a

  comp2 :: t a b c -> Int

  one :: b -> t a b c

  fromOne :: t a b c -> b

  rec :: c -> t a b c -> t a c b -> t a b c

  fromRec :: t a b c -> (c, t a b c, t a c b)
```

6
4 Final notes from the author

The purpose of these notes\footnote{These notes was created with \TeX{} 2.1.2.} is to show a way to define a class from a data declaration, and to be a documentation complement to the \texttt{DTC} package. The interest of \texttt{DTC} is more theoretical than practical. But, if you have a practical usage in mind, I will be interested in know it. As usually, I’m open to suggestions of any type.

Greetings,
Daniel Díaz