Práctica 5

Functores

1. Demostrar que los siguientes tipos de datos son functores. Es decir, dar su instancia de la clase Functor correspondiente y probar que se cumplen las leyes de los functores.

a) **data** Pair a = P (a, a)

b) **data** Tree a = Empty | Branch a (Tree a) (Tree a)

c) **data** GenTree a = Gen a [GenTree a]

d) **data** Cont a = C $((a \to \mathsf{Int}) \to \mathsf{Int})$

2. Probar que las siguientes instancias no son correctas (no cumplen las leyes de los functores).

a) **data** Func a = Func $(a \to a)$

instance Functor Func **where**
  fmap g (Func h) = Func id

b) **data** Br b a = B b (a, a)

instance Functor (Br b) **where**
  fmap f (B x (y, z)) = B x $(f\ z, f\ y)$

Functores aplicativos

3. Dar la instancia de Applicative para:

a) Either e, con e fijo.

b) $(\to)$ r, con r fijo.

4. Las funciones liftAx aplican una función a x argumentos contenidos en una estructura. Usando los operadores de la clase Applicative, dar las definiciones de:

a) liftA2 :: Applicative $f \Rightarrow (a \to b \to c) \to f\ a \to f\ b \to f\ c$

Por ejemplo, liftA2 (,) (Just 3) (Just 5) evalúa a Just $(3, 5)$.

b) liftA5 :: Applicative $f \Rightarrow (a \to b \to c \to d \to e \to k) \to f\ a \to f\ b \to f\ c \to f\ d \to f\ e \to f\ k$

5. Definir una función sequenceA :: Applicative $f \Rightarrow [f\ a] \to f\ [a]$, que dada una lista de acciones de tipo $f\ a$, siendo f un funtor aplicativo, transforme la lista una acción de tipo $f\ [a]$.

Uso de Mónadas y notación **do**

6. Probar que toda mónada es un functor, es decir, proveer una instancia

```
instance Monad m ⇒ Functor m where
  fmap ...
```

y probar que las leyes de los functores se cumplen para su definición de fmap.

7. Dados los siguientes tipos de datos:

```
newtype Id a     = Id a
data Either e a = Left e | Just a
```

Es decir,

a) Dar la instancia de Monad para Id y Either e.

Nota: Dado que toda instancia de Monad debe tener una instancia en la clase Applicative, y que todo Applicative debe tener instancia en Functor, podemos dar las siguientes instancias para cualquier mónada m y ocuparnos sólo de la instancia de Monad:

```
instance Functor M where
  fmap = liftM
instance Applicative M where
  pure = return
  (<*>) = ap
```

donde las operaciones ap y liftM se exportan de los siguientes módulos:

```
import Control.Applicative (Applicative (..))
import Control.Monad (liftM, ap)
```

b) Demostrar que para cada instancia valen las leyes de las mónadas.

8. Demostrar que el constructor de tipo [ ] (lista) es una mónada.

9. Dado el siguiente tipo de datos para representar expresiones matemáticas:

```
data Expr a = Var a | Num Int | Add (Expr a) (Expr a)
```

´

a) Dar la instancia de Monad para Expr y probar que es una mónada.

b) Dar el tipo y la definición de la función g de manera que Add (Var "y") (Var "x") ≫= g evalúe a la expresión Add (Mul (Var 1) (Num 2)) (Var 1).

c) Explicar qué representa el operador ≫= para este tipo de datos.

10. Escribir el siguiente fragmento de programa en términos de ≫= y return.

```
do x ← (do z ← y
           w ← f z
           return (g w z))
   y ← h x 3
   if y then return 7
        else do z ← h x 2
                return (k z)
```

11. Escribir el siguiente fragmento de programa monádico usando notación **do**.

$$(m \gg\!= \lambda x \to h\ x) \gg\!= \lambda y \to f\ y \gg\!= \lambda z \to \mathsf{return}\ (g\ z)$$

12. Escribir las leyes de las mónadas usando la notación **do**.

13. La clase Monoid clasifica los tipos que son monoides y está definida de la siguiente manera

```
class Monoid m where
  mempty  :: m
  mappend :: m → m → m
```

Se requiere que las instancias hagan cumplir que mappend sea asociativa, y que mempty sea un elemento neutro de mappend por izquierda y por derecha.

a) Probar que String es un monoide.

b) Probar que el siguiente constructor de tipos es una mónada, (asumiendo que el parámetro w es un monoide).

```
newtype Output w a = Out (a, w)
```

c) Dar una instancia diferente de Monad para el mismo tipo. Esto prueba que un mismo tipo de datos puede tener diferentes instancias de mónadas.

d) Definir una operación write :: Monoid $w \Rightarrow w \to$ Output w ().

e) Usando Output String, modificar el evaluador monádico básico de la teoría para agregar una traza de cada operación. Por ejemplo:

```
> eval (Div (Con 14) (Con 2))
El término (Con 14) tiene valor 14
El término (Con 2) tiene valor 2
El término (Div (Con 14) (Con 2)) tiene valor 7
7
```

Funciones monádicas genéricas

14. Sea M una mónada. Dados los operadores:

$$(\gg)\ :: \mathsf{M}\ a \to \mathsf{M}\ b \to \mathsf{M}\ b$$
$$(\gg\!=) :: \mathsf{M}\ a \to (a \to \mathsf{M}\ b) \to \mathsf{M}\ b$$

a) De ser posible, escribir $(\gg)$ en función de $(\gg\!=)$.

b) De ser posible, escribir $(\gg\!=)$ en función de $(\gg)$.

15. Definir las siguientes funciones:

a) mapM :: Monad $m \Rightarrow (a \to m\ b) \to [a] \to m\ [b]$, tal que mapM $f\ xs$ aplique la función monádica f a cada elemento de la lista xs, retornando la lista de resultados encapsulada en la mónada.

b) foldM :: Monad $m \Rightarrow (a \to b \to m\ a) \to a \to [b] \to m\ a$, análogamente a foldl para listas, pero con su resultado encapsulado en la mónada. *Ejemplo:*

```
foldM f e1 [x1, x2, x3] = do e2 ← f e1 x1
                             e3 ← f e2 x2
                             f e3 x3
```