Programación Funcional

Trabajo Práctico Nro. 4

Temas: Tipos algebraicos. Pattern matching. Listas.

Bibliografía relacionada:

- Simon Thompson. The craft of Functional Programming. Addison Wesley, 1996. Cap. 4.
- L.C. Paulson. ML for the working programmer. Cambridge University Press, 1996. Cap. 3.
- Bird, Richard. Introduction to funtional programming using Haskell. Prentice Hall, 1998 (Second Edition). Cap. 4.

1. Indicar cuáles de los siguientes son patterns válidos. Justificar.

   a) `(x, y)`
   b) `(1, 2)`
   c) `(n+1)`
   d) `(10)`
   e) `('a',('a',b))`
   f) `(a,(a,b))`
   g) `(x,(x,y))`
   h) `([]:[4])`
   i) `(x:y:[])`
   j) `[xs]`
   k) `([]:[])`

2. Definir recursivamente las siguientes funciones y dar sus tipos:

`sum`,	suma los elementos de una lista de números.
`any`,	devuelve `True`si algún elemento de una lista es `True`.
`all`,	devuelve `True`si todos los elementos de una lista son `True`.
`codes`,	dada una lista de caracteres, devuelve la lista de sus códigos.
`remainders`,	dada una lista de números, devuelve los restos de su división por un número.
`squares`,	dada una lista de números, devuelve la lista de sus cuadrados.
`lengths`,	dada una lista de listas, devuelve la lista de sus longitudes.
`order`,	dada una lista de pares ordenados de números, devuelve la lista de aquellos cuya primer componente es menor que el triple de la segunda.
`pairs`,	dada una lista de números, devuelve la lista con aquellos que son pares.
`chars`,	dada una lista de caracteres, devuelve la lista de los que son letras.
`moreThan`,	dada una lista de listas `xss` y un número `n`, devuelve la lista de aquellas listas de `xss` que tienen longitud mayor que `n`.

3. Indicar bajo qué suposiciones pueden evaluarse las siguientes ecuaciones y determinar cuáles de ellas resultan ser verdaderas y cuáles falsas. Justificar las respuestas. Para las que sean falsas, indicar cuál debería ser el resultado de cada expresión.

   a) `[[]] ++ xs = xs`
   b) `[[]] ++ [xs] = [[],xs]`
   c) `[[]] ++ xs = [xs]`
   d) `[]:xs = xs`
   e) `[[]] ++ [xs] = [xs]`
   f) `[[]] ++ xs = []:xs`
   g) `[xs] ++ [xs] = [xs,xs]`
   h) `[] ++ xs = []:xs`
   i) `[[]] ++ xs = [[],xs]`
   j) `[xs] ++ [] = [xs]`

4. Dadas las siguientes definiciones que representan a los booleanos mediante funciones,

```
ifThenElse_Lam = \x -> x
true_Lam = \x -> \y -> x
false_Lam = \x -> \y -> y
not_Lam = \x ->ifThenElse_Lam x false_Lam true_Lam
```

   definir las operaciones `or_Lam`, `and_Lam`, `xor_Lam`, `iff_Lam` que se comporten como sus contrapartes booleanas.

5. Dadas la definición de `pair_Lam` que representan a los pares mediante funciones,
   `pair_Lam = \x ->\y ->\b ->ifThenElse_Lam b x y`

   definir las operaciones `fst_Lam`, `snd_Lam` que se comporten como las proyecciones de las componentes del par.

6. Definir las operaciones de unión e intersección de dos conjuntos y el predicado de pertenencia para conjuntos de elementos de tipo `a`, los cuales se representan:

   a) por extensión (como lista de elementos de `a`).
   b) por comprensión (como predicado `a ->Bool`);

7. ⊛ Implementar una función que reciba una lista de listas de números y devuelva una lista de los impares que aparecen en las mismas.

Ejercicios complementarios

8. *a*) Definir las operaciones de suma y producto módulo 2 para el tipo
`data DigBin = Cero | Uno`

   b) Definir las operaciones de suma binaria, producto por dos y cociente y resto de la división por dos para el tipo `type NumBin = [DigBin]` donde el primer elemento de la lista de dígitos es el dígito menos significativo del número representado. Ej: `sumabin [Uno, Cero, Uno] [Uno, Uno] = [Cero, Cero, Cero, Uno]`

   c) Redefinir las funciones del ítem anterior, observando una convención opuesta (en este caso, el primer elemento es el dígito más significativo).

   d) Definir funciones para convertir valores de tipo `DigBin` a `Int` y viceversa.