Programación Funcional

Trabajo Práctico Nro. 5

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. ¿Puede asegurar que las funciones que definió en el ejercicio anterior terminan? ¿Por qué? ¿Qué diferencia encuentra con las demostraciones de la práctica 4?

4. 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]`

5. Demostrar:

   a) `pairs . squares = squares . pairs`, donde `squares` eleva al cuadrado los elementos de una lista, y `pairs` devuelve sólo los elementos pares de una lista de números.

   b) `((‘mod‘ n) . sum) (remainders n xs) = (sum xs) ‘mod‘ n`, para todo $n > 0$

6. Demostrar la siguiente propiedad: `sum` $x_s \leq$ `len` x_s $*$ `maxl` x_s, siendo `maxl` la función

```
maxl [] = 0
maxl (x:xs) = x ‘max‘ maxl xs
```

   y considerando que x_s es una lista finita de números naturales.

7. 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.

8. 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.

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

   a) Implementar en Pascal.

   b) Implementar en Haskell.

Ejercicios complementarios

10. *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.

11. ⍟ Dadas las funciones `reverse` y `rev` definidas de la siguiente manera:

```
reverse, rev :: [a] -> [a]
reverse  []    = []
reverse (x:xs) = reverse xs ++ [x]
rev xs = fastrev [] xs
         where fastrev xs []     = xs
               fastrev xs (y:ys) = fastrev (y:xs) ys
```

   a) Demuestre que se cumple:

   Lema 1 `fastrev ys xs = fastrev [] xs ++ ys`, para todas listas `xs,ys`

   b) Demuestre que `reverse` = `rev`.

Propiedades útiles:

1. $\forall$ n, m ϵ N, n $\leq$ max (n,m)
2. x es par $\Leftrightarrow$ $x * x$ es par
3. $(x \bmod n) \bmod n = x \bmod n$
4. (A + B) mod n = (A mod n + B mod n) mod n