Listas

Taller de Álgebra I

Verano 2018

Un nuevo tipo

Tipos

- ¿De qué tipo es la función `identidad i`, que dado un número devuelve al mismo número?
- ¿De qué tipo es la función `numeroYsiguiente i`, que dado el número `i` devuelve al mismo número y al número siguiente?
- ¿De qué tipo es la función `numerosHasta i`, que dado el número `i` devuelve todos los números hasta `i`

Tipo Lista

La lista es una secuencia de valores del mismo tipo. Ejemplos:

- `[1, 2] :: [Integer]`.
- `[1.0, 2] :: [Float]`.
- `[[1], [2,3], [], [1,1000,2,0]] :: [[Integer]]`.
- `[1, True]` NO ES UNA LISTA VÁLIDA
- `[1.0, div 1 1]` NO ES UNA LISTA VÁLIDA
- `[] :: [a]`.

Construcción

Toda lista es:

- La **lista vacía**: `[]`
- El resultado de **agregar un elemento** al principio de **otra lista**: `(:)`

Ejemplos:

- `1:2:3:[]`
- `[1,2]:[10,20,30]:[]`
- `[]:[]:[]`
- `True:2:3:[]`

Operaciones

Operaciones principales

- `(:) :: a -> [a] -> [a]`
- `(++) :: [a] -> [a] -> [a]`
- `head :: [a] -> a`
- `tail :: [a] -> [a]`
- `length :: [a] -> Integer`
- `reverse :: [a] -> [a]`

Tipar y evaluar las siguientes expresiones

- `head [(1,2), (3,4), (5,2)]`
- `tail [1,2,3,4,4,3,2,1]`
- `head []`
- `head [1,2,3] : [2,3]`
- `[True, True] ++ [False, False]`
- `[1,2] : []`

Listas

Formas rápidas para crear listas

Prueben las siguientes expresiones en GHCI

- `[1,2,3,4,5]`
- `[1..100]`
- `[1,3..100]`
- `[100..1]`

Ejercicios

- Definir la función `listar :: a -> a -> a -> [a]` que toma 3 elementos y los convierte en una lista.
- Escribir una expresión que denote la lista estrictamente decreciente de enteros que comienza con el número 1 y termina con el número -100.

Pattern matching en listas

Ya vimos cómo hacer *pattern matching* sobre distintos tipos (`Bool`, `Integer`, tuplas).
¿Se puede hacer *pattern mathing* en listas?

¿Cuáles son las dos formas de crear una lista?

Las listas tienen dos "pintas":

- `[]` (lista vacía)
- `elemento : lista` (lista no vacía)

¿Cómo escribir la función `sumatoria :: [Integer] -> Integer` usando *pattern matching*?

```
sumatoria [] = 0
sumatoria (x:xs) = sumatoria xs + x
```

Recursión sobre listas

Implementar las siguientes funciones

- `pertenece :: Integer -> [Integer] -> Bool`
  que indica si un elemento aparece en la lista. Por ejemplo:
  `pertenece 9 []` ⇝ `False`
  `pertenece 9 [1,2,3]` ⇝ `False`
  `pertenece 9 [1,2,9,9,-1,0]` ⇝ `True`
- `maximo :: [Integer] -> Integer`
  que devuelve el número más grande de una lista no vacía.

Pattern matching avanzado

Puedo utilizar valores también para los elementos:

```
tieneUnDiez [] = False
tieneUnDiez (10:xs) = True
tieneUnDiez (x:xs) = tieneUnDiez xs
```

Puedo hacer pattern matching sobre más de un elemento (y también utilizar el patrón _). Por ejemplo:

```
tieneDosElementos :: [a] -> Bool
tieneDosElementos (_:_:[]) = True
tieneDosElementos _  = False
```

Ejercicios

- `productoria :: [Integer] -> Integer` que devuelve la productoria de los elementos.
- `sumarN :: Integer -> [Integer] -> [Integer]` que dado un número *N* y una lista *xs*, suma N a cada elemento de *xs*.
- `sumarElUltimo :: [Integer] -> [Integer]` que dada una lista no vacía *xs*, suma el último elemento a cada elemento de *xs*. Ejemplo `sumarElUltimo [1,2,3]` $\rightsquigarrow$ `[4,5,6]`
- `sumarElPrimero :: [Integer] -> [Integer]` que dada una lista no vacía *xs*, suma el primer elemento a cada elemento de *xs*. Ejemplo `sumarElPrimero [1,2,3]` $\rightsquigarrow$ `[2,3,4]`
- `pares :: [Integer] -> [Integer]` que devuelve una lista con los elementos pares de la lista original. Ejemplo `pares [1,2,3,8]` $\rightsquigarrow$ `[2,8]`
- `multiplosDeN :: Integer -> [Integer] -> [Integer]` que dado un número *N* y una lista *xs*, devuelve una lista con los elementos multiplos N de *xs*.
- `quitar :: Integer -> [Integer] -> [Integer]` que elimina la primera aparición del elemento en la lista (de haberla).
- `hayRepetidos :: [Integer] -> Bool` que indica si una lista tiene elementos repetidos.
- `eliminarRepetidos :: [Integer] -> [Integer]` que deja en la lista una única aparición de cada elemento, eliminando las repeticiones adicionales.
- `maximo :: [Integer] -> Integer` que calcula el máximo elemento de una lista no vacía.
- `ordenar :: [Integer] -> [Integer]` que ordena los elementos de forma creciente.