Programación Funcional en Haskell

Paradigmas de Lenguajes de Programación

Facultad de Ciencias Exactas y Naturales
Universidad de Buenos Aires

29 de agosto de 2017

Nos faltaba...

1. Funciones como estructuras de datos
2. Generación infinita

Ejercicio

Se cuenta con la siguiente representación de conjuntos, caracterizados por su función de pertenencia:

```
type Conj a = (a->Bool)
```

De este modo, si `conj1` es un conjunto y `e` un elemento, la expresión `conj1 e` devuelve `True` si `e` pertenece a `conj1`, y `False` en caso contrario.

Ejercicio

Se cuenta con la siguiente representación de conjuntos, caracterizados por su función de pertenencia:

```
type Conj a = (a->Bool)
```

De este modo, si `conj1` es un conjunto y `e` un elemento, la expresión `conj1 e` devuelve `True` si `e` pertenece a `conj1`, y `False` en caso contrario.

Operaciones sobre conjuntos

- Definir y dar el tipo de las siguientes funciones:
  - vacío
  - agregar
  - unión
  - intersección

Ejercicio

Se cuenta con la siguiente representación de conjuntos, caracterizados por su función de pertenencia:

```
type Conj a = (a->Bool)
```

De este modo, si `conj1` es un conjunto y `e` un elemento, la expresión `conj1 e` devuelve `True` si `e` pertenece a `conj1`, y `False` en caso contrario.

Operaciones sobre conjuntos

- Definir y dar el tipo de las siguientes funciones:
  - vacío
  - agregar
  - unión
  - intersección
- ¿Puede definirse la función `esVacio :: Conj a -> Bool`?

Ejercicio

Se cuenta con la siguiente representación de conjuntos, caracterizados por su función de pertenencia:

```
type Conj a = (a->Bool)
```

De este modo, si `conj1` es un conjunto y `e` un elemento, la expresión `conj1 e` devuelve `True` si `e` pertenece a `conj1`, y `False` en caso contrario.

Operaciones sobre conjuntos

- Definir y dar el tipo de las siguientes funciones:
  - vacío
  - agregar
  - unión
  - intersección
- ¿Puede definirse la función `esVacio :: Conj a -> Bool`?
  ¿Y `esVacio :: Conj Bool -> Bool`?
- Definir la función `primerOcurrencia :: a -> [Conj a] -> Int` que, dados un elemento e y una lista de conjuntos (que puede ser finita o infinita), devuelva la primera posición de la lista en la cual el conjunto correspondiente contiene al elemento e. Se asume que e pertenece al menos a un conjunto de la lista.

Generación Infinita

Ejercicio: Definir

`puntosDelCuadrante :: [Punto]`

Donde `Punto`, un renombre de tipos: `type Punto = (Int, Int)`

El resultado debe ser una lista (infinita) que contenga todos los puntos del cuadrante superior derecho del plano (sin repetir).

Generación Infinita

Ejercicio: Definir

`puntosDelCuadrante :: [Punto]`

Donde `Punto`, un renombre de tipos: `type Punto = (Int, Int)`

El resultado debe ser una lista (infinita) que contenga todos los puntos del cuadrante superior derecho del plano (sin repetir).

Ejercicio de tarea: Definir

`listasPositivas :: [[Int]]`

que contenga todas las listas finitas de enteros mayores o iguales que 1.

Generación Infinita

Ejercicio: Definir

`puntosDelCuadrante :: [Punto]`

Donde `Punto`, un renombre de tipos: `type Punto = (Int, Int)`

El resultado debe ser una lista (infinita) que contenga todos los puntos del cuadrante superior derecho del plano (sin repetir).

Ejercicio de tarea: Definir

`listasPositivas :: [[Int]]`

que contenga todas las listas finitas de enteros mayores o iguales que 1.

Ayuda: Definir primero

`listasQueSuman :: Int -> [[Int]]`

que, dado un número natural n, devuelve todas las listas de enteros mayores o iguales que 1 cuya suma sea n