Escuela Técnica Superior de Ingeniería Informática

Departamento de Lenguajes y Ciencias de la Computación

Apellidos Nombre

PROGRAMACION DECLARATIVA (y LAB. V)
EXTRAORDINARIO DE SETIEMBRE (03/9/02)

PUNTUACION: **1**(2.5) **2**(1+1) **3(**2.5**)** **4**(1.5+1.5)

---

1 Sabiendo que *id = foldr* (:) [ ], demuestra la siguiente propiedad para listas

$\forall$ *as, bs* :: [*a*] . *foldr* (:) *as bs* $\equiv$ *foldr* (:) [ ] (*bs* ++ *as*)

SOLUCION. Recordemos la función foldr:

foldr :: (a -> b -> b) -> b -> [a] -> b

foldr f z [] = z

foldr f z (x:xs) = f x (foldr f z xs)

Probaremos la propiedad por inducción sobre la lista *bs*. Si usamos la propiedad *id = foldr* (:) [ ], lo que tenemos que probar es:

$\forall$ bs :: [a] . ($\forall$ *as* . *foldr* (:) *as bs* $\equiv$ *bs* ++ *as*)

El caso base será

$\forall$ *as*:: [*a*] . *foldr* (:) *as* [] $\equiv$ [] ++ *as*

= ! primera ecuación de *foldr*; propiedades de ++

$\forall$ *as*:: [*a*] . *as* $\equiv$ *as*

=!

Cierto

La prueba del paso inductivo sería:

$\forall$ *as*:: [*a*] . *foldr* (:) *as (b:bs)* $\equiv$ *(b:bs)* ++ *as*

= ! segunda ecuación de *foldr*; definición de ++

$\forall$ *as*:: [*a*] . b : (*foldr* (:) *as* bs) $\equiv$ b : (bs ++ *as)*

= ! hipótesis de inducción

Cierto

2.A Escribe la expresión *map f (filter p xs)* usando listas por comprensión

SOLUCION. [f x | x <- xs, p x]

2.B Escribe la expresión *[p (f x) | x <-xs, p x]* usando las funciones *filter* y *map*

SOLUCION. map (p . f) (filter p xs)

3 Consideremos las siguientes funciones:

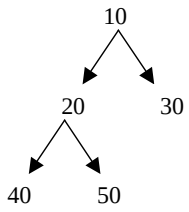
```
unfold p h t x
  | p x          = [ ]
  | otherwise = h x : unfold p h t (t x)
desdeHasta n m = [n .. m]
```

Da un definición de la función *desdeHasta* usando *unfold*.

SOLUCION. desdeHasta n m = unfold (> m) id (+1) n

---

4 Una rama es una lista con los nodos desde la raíz de un árbol hasta una de sus hojas. Para el árbol que aparece a continuación las distintas ramas son [10,20,40], [10,20,50] y [10,30]:



4.A Para el tipo

```
data ÁrbolB a = VacíoB | NodoB (ÁrbolB a) a (ÁrbolB a) deriving Show
```

define una función *ramas :: ÁrbolB a -> [[a]]* que devuelva todas las ramas de un árbol binario.

SOLUCION

```
ramas  VacíoB                    = [[]]
ramas (NodoB VacíoB x VacíoB)  = [[x]]
ramas (NodoB i a d)              = map (a:) (ramas i ++ ramas d)
```

4.B Resuelve el mismo problema pero para un árbol general:

```
data Árbol a = Vacío | Nodo a [Árbol a] deriving Show
```

SOLUCION

```
ramasG :: Árbol a -> [[a]]
ramasG Vacío          = []
ramasG (Nodo x [])    = [[x]]
ramasG (Nodo x rs)    = map (x:) $ concat $ map ramasG $ rs
```