PRÁCTICA 5

INTRODUCCIÓN WINHUGS

Universidad de Huelva

5.1. REPASO

5.1. Repaso

TUPLAS

Tuplas: 2-tupla 3-tupa

Funciones sobre tuplas

fst, snd, splitAt, span, break

Empaquetamiento

zip, zip3, zipWith, zipWith3, unzip, unzip3

5.1. Repaso

FUNCIONES PROPIAS

Orden de prioridad

Infix, Infixl, Infixr, (), “.”

x1 x2 · · · xn−1 xn => (. . . (x1 x2) · · · xn−1) xn

x1 x2 · · ·xn−1 xn => x1 (x2 · · ·(xn−1 xn) . . .)

Evaluación perezosa

Notación (x:xs)

```
3   suma :: [Int] -> Int
4   suma (x:xs) = x + suma xs
```

Formas de definir una función: varias ecuaciones, guardas, if then else, case, definiciones locales

Índice

¿Qué vamos a ver?

Concepto de Currificación

Concepto de principio de inducción y recursividad

Listas intensionales

Ejercicios

Práctica 2 hasta final de la clase

5.2.1. Currificación

Currificación de funciones

Consiste en realizar una llamada a una función con los parámetros que están más a la izquierda.

Una función puede recibir un número variable de argumentos, dando valores de forma parcial de izquierda a derecha.

Si la función tiene 3 argumentos puedo dar el primero y dejar los otros dos…

También puedo poner los dos primero fijos y dar el último

```
Main> map (suma 1 1) [1,2,3]
[3,4,5] :: [Int]
```

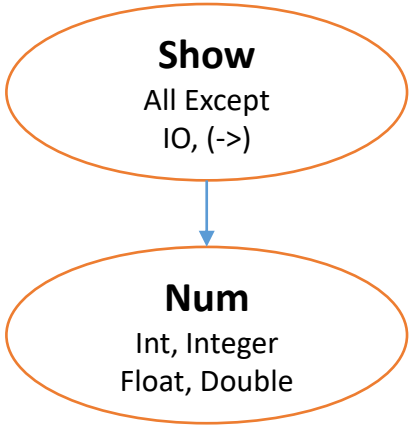
```
{-- CURRIFICACIÓN DE FUNCIONES --}

--suma :: Num a => a -> a -> a
--suma x y = x + y
--sumar :: Integer -> Integer
--sumar = suma(4)
```

5.2.1. Currificación

Currificación de funciones: suma x y z = x + y + z

```
Main> map (suma 2 4) [1..4]
[7,8,9,10]
Main> map (suma 2) [1..4]
ERROR - Cannot find "show" function for:
*** Expression : map (suma 2) (enumFromTo 1 4)
*** Of type    : [Int -> Int]
```



¿Qué devuelve? ¿por qué da el error? ¿es un error?

Devuelve una lista de funciones y **no se puede mostrar por pantalla -> [(+3),(+4),(+3)], lista de funciones**

`(a->b) -> (a->a->a)`

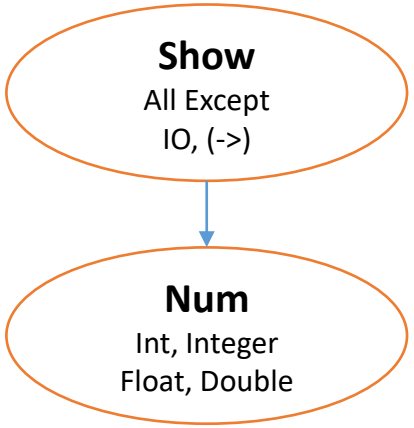
5.2.1. Currificación

Currificación de funciones: lista de funciones

Veamos un ejemplo. ¿Está completa?

```
miFuncion::[(a->b)] -> [a] -> [b]
miFuncion (f:fs) (x:xs) = (f x):(miFuncion fs xs)
```

```
miFuncion::[(a->b)] -> [a] -> [b]
miFuncion [] _ = []
miFuncion (f:fs) (x:xs) = (f x):(miFuncion fs xs)
```



Hay que añadir el caso base

5.2.1. Currificación

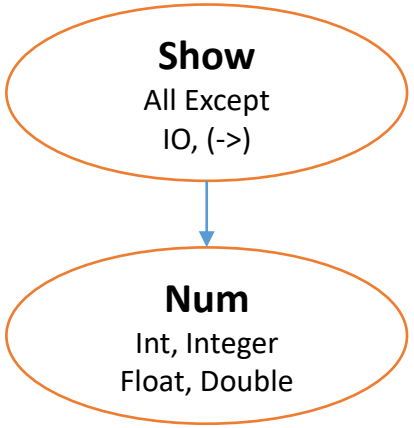
Currificación de funciones: lista de funciones

Veamos un ejemplo

```
Main> miFuncion [(+1), (+2), (+3)] [1..3]
[2,4,6]

¿y si dividimos y restamos?

Main> miFuncion [(/4), (**4), (3-)] [1..3]
[0.25,16.0,0.0]
Main> miFuncion [(div 4), (^4), (3-)] [1..4]
[4,16,0]
```

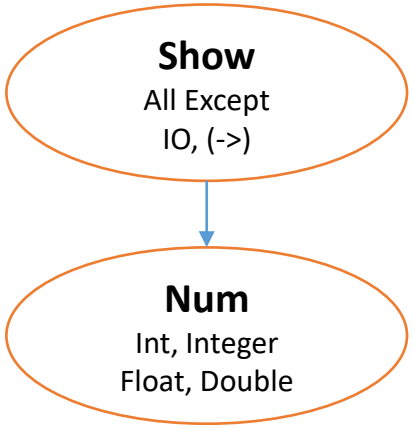


5.2.1. Currificación

Currificación de funciones: función map aplicando currifucación

```
Main> funcionMapCurrificada (map (suma3 3) [1,2,3]) [1,2,3]
[5,7,9] :: [Int]
```

¿Cuál sería su implementación? 5 minutos




```haskell
suma3 ::  Int -> Int-> Int -> Int
suma3 x y z = x + y + z

funcionMapCurrificada :: [a -> b] -> [a] -> [b]
funcionMapCurrificada [] _ = []
funcionMapCurrificada (f:fs) (x:xs) = f x : (funcionMapCurrificada fs xs)
```

5.2.2. Principio de inducción

Principio de inducción

Sea x que pertenece al conjunto S, **S tiene que ser ordenable**

Queremos probar si la propiedad P es cierta para todo elemento del conjunto S

1) **P es cierta para el n0 (el elemento mas pequeño)**

   Ejemplo 1: naturales 1, 2, 3, 4 …

   Ejemplo 2: listas [], [_],[_,_], …

   Ejemplo 3: arboles binarios: nil, a(_, nil, nil), …

2) **Si P es cierta para n-1 entonces puedo afirmar que puede ser cierta para n: P(n-1) -> P(n)**

5.2.2. Principio de inducción

Principio de inducción

Suma todos los elementos de una lista aplicando el principio de inducción matemático.

1) P(n0)

2) P(n-1) -> P(n)

3) ¿n-1 en listas? Operador ":", que separa la cabeza del resto.

4) Suma es cierto si devuelve la suma de los elementos de la lista que se pasa por parámetros como arg. 1

suma :: Num a => [a] -> a

suma [] = 0 | suma (cabeza:resto) = cabeza + (suma resto). Pensemos la función como algo estático.

5.2.2. Principio de inducción

Principio de inducción

Suma todos los elementos de una lista aplicando el principio de inducción matemático.

1) suma [1,2,3] -> suma [2, 3] -> suma [3] -> suma []

2) 1+5 2+3 3+0 0

Y si quisiéramos demostrar los siguiente, ¿Cuándo sería cierto?

```
miFuncionLista::[(a->b)] -> [a] -> [[b]]
```

```
Ejemplo: map (+1) (+2) (+3) [1,2,3] -> [[2,3,4],[3,4,5][4,5,6]]
```

5.2.2. Principio de inducción

Principio de inducción

miFuncionLista es cierto si devuelve una lista de listas, resultado de aplicar las funciones de la lista de funciones del primer argumento a todos los valores de la lista de valores del segundo argumento.

```
miFuncionLista::[(a->b)] -> [a] -> [[b]]
miFuncionLista [] _ = []
miFuncionLista (f:fs) lista = (map f lista) : (miFuncionLista fs lista)
```

```
Main> miFuncionLista [(+2), (+3)] [1,2,3]
[[3,4,5],[4,5,6]] :: [[Integer]]
```

5.2.2. Principio de inducción

Principio de inducción: definimos la función map de otra forma

```
miFuncion2::[(a->b)] -> [a] -> [[b]]
miFuncion2 [] _ = []
miFuncion2 (f:fs) x = miFuncion3 f x : miFuncion2 fs x
```

¿Como sería miFuncion3?

```
Main> miFuncion2 [(+2), (+3)] [1,2,3]
[[3,4,5],[4,5,6]] :: [[Integer]]
```

5.3. LISTAS INTENSIONALES

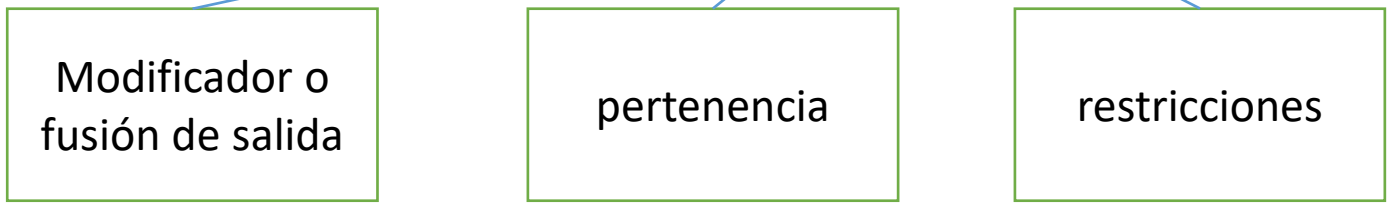
5.3.1. Concepto

Listas intensionales

Haskell incluye una sintaxis especial para representar de forma más compacta operaciones sobre listas.

En matemáticas existe una forma “intensiva de definir conjuntos”

$$S = \{2 \cdot x \mid x \in \mathbb{N},\ x \leq 10\}$$



Esto significa que el conjunto contiene todos los dobles de los número naturales que cumplen el predicado

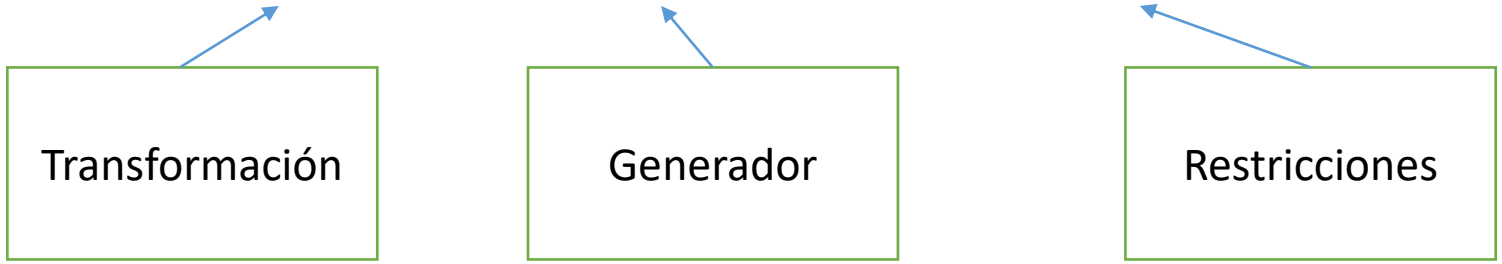
5.3.1. Concepto

Listas intensionales

En haskell las listas intensionales son similares a los conjuntos definidos intensionalmente.

¿Cómo definiríamos el caso anterior? Consta de 3 partes

```
[(x,True) | x <- [1..20], even x, x < 15]
```



¿Y si quisiéramos todos los números del 50 al 100 cuyo resto al dividir por 7 fuera 3?

```
[ x | x <- [50..100], x `mod` 7 == 3]
```

5.3.1. Concepto

Listas intensionales

Son muy utilizadas en haskell, ya que permite generar una lista sin tener que utilizar una función con recursividad.

Veamos algunos ejemplos

```
Main> [(1,2*x)|x<-[1..5]]
[(1,2),(1,4),(1,6),(1,8),(1,10)] :: [(Integer,Integer)]
Main> [x==1|x<-[1..5]]
[True,False,False,False,False] :: [Bool]
Main> [f 1|f<-[(+1), (+2), (+3)]]
[2,3,4] :: [Integer]
```

5.3.1. Concepto

Listas intensionales

Puedo utilizar más de un generador.

```
Main>
[2,3,4,3,4,5,4,5,6] :: [Integer]
```

¿qué ocurre si cambiamos el orden? Nota: debemos utilizar una función diferente.

Se fija un generador y se aplica a todos los del segundo.

```
Main> [f x | f<-[(==1), (==2), (==3)],x <- [3,4,5]]
[False,False,False,False,False,False,True,False,False] :: [Bool]
Main> [f x | x <- [3,4,5], f<-[(==1), (==2), (==3)]]
[False,False,True,False,False,False,False,False,False] :: [Bool]
```

5.3.2. Ejercicios

Listas intensionales

Implementar la función filter2: filter2 :: (a -> Bool) -> [a] -> [a]

```
filter2 :: (a -> Bool) -> [a] -> [a]
filter2 p xs =  [x | x <- xs, p x]
```

```
Main> filter2 (==1) [1,2,3]
[1] :: [Integer]
```

5.3.2. Ejercicios

Listas intensionales

Implementar sumaCuadradosPares xs es la suma de los cuadrados de los números pares de la lista xs**.**

Por ejemplo: sumaCuadradosPares [1..5] == 20

```haskell
sumaCuadradosPares :: [Int] -> Int
sumaCuadradosPares xs = sum [x^2 | x <- xs, even x]
```

```
Main> sumaCuadradosPares [1..5]
20 :: Int
```

5.3.2. Ejercicios

Listas intensionales

Generar cada una de las siguientes listas utilizando la notación extendida de listas, con la lista [1..10] como generador. Es decir, cada solución debe tener la siguiente forma, donde se deben completar los blancos y sólo los blancos.

```
[ ____________________ | x <- [1 .. 10] ____________________]
```

De forma más explícita, la respuesta debe utilizar el generador x<- [1.. 10], y no debe añadir a esta definición ninguna llamada a función: por ejemplo, no utilizar reverse [x | x <- [1 .. 10]] para crear la lista [10,9,8,7,6,5,4,3,2,1]. De la misma forma, modificaciones del tipo [x|x <- [10,9..1]] y [x|x <- reverse[1 ..10]] también están prohibidas.

5.3.2. Ejercicios

Listas intensionales

1. `[11,12,13,14,15,16,17,18,19,20]`
2. `[[2],[4],[6],[8],[10]]`
3. `[[10],[9],[8],[7],[6],[5],[4],[3],[2],[1]]`
4. `[True,False,True,False,True, False,True,False,True,False]`
5. `[(3,True),(6,True),(9,True),(12,False),(15,False), (18,False)]`
6. `[(5,False),(10,True),(15,False),(40,False)]`
7. `[(11,12),(13,14),(15,16),(17,18),(19,20)]`
8. `[[5,6,7],[5,6,7,8,9],[5,6,7,8,9,10, 11],[5,6,7,8,9,10,11,12,13]]`
9. `[21,16,11,6,1]`
10. `[[4],[6,4],[8,6,4],[10,8,6,4],[12,10,8,6,4]]`