Tema 8. Listas

8.1 Secuencias aritméticas

✓ Sintaxis para definir listas, siempre que los elementos de la lista sean instancia de $Enum$.

✓ Todos los tipos simples predefinidos son instancias de $Enum$.

Ejemplos:

✓ Lista con enteros entre 1 y 10 (de uno en uno)

? [1 .. 10]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10] :: [*Integer*]

✓ De dos en dos (se especifican los dos primeros elementos)

? [1, 3 .. 11]
[1, 3, 5, 7, 9, 11] :: [*Integer*]

✓ En orden decreciente.

? [10, 9 .. 1]
[10, 9, 8, 7, 6, 5, 4, 3, 2, 1] :: [*Integer*]

✓ Si no se especifica el elemento final, se pueden obtener listas infinitas:

? [1 ..]
[1, 2, 3, 4, 5, 6, 7, 8, 9, 10, 11, 12, 13, 14, 15 . . .

8.2 Algunas funciones predefinidas

✓ Selectores básicos

```
? head [1 .. 5]                 -- head :: [a] → a
1  :: Integer                   -- cabeza

? tail [1 .. 5]                 -- tail :: [a] → [a]
[2,3,4,5]  :: [Integer]         -- cola

? last [1 .. 5]                 -- last :: [a] → a
5  :: Integer                   -- último

? init [1 .. 5]                 -- init :: [a] → [a]
[1,2,3,4]  :: [Integer]         -- inicio
```

✓ Más selectores

```
? take 3 [1 .. 5]               -- take :: Int → [a] → [a]
[1,2,3]  :: [Integer]           -- toma

? drop 3 [1 .. 5]               -- drop :: Int → [a] → [a]
[4,5]  :: [Integer]             -- quita

? [1 .. 5] !! 3                 -- (!!) :: [a] → Int → a
4  :: Integer                   -- selecciona
```

✓ *map* y filtros

```
? map (*10) [2,3,4,6]           -- map :: (a → b) → [a] → [b]
[20,30,40,60]  :: [Integer]     -- aplicar a todos

? filter even [2,4,8,9,10,11,12]  -- filter :: (a → Bool) → [a] → [a]
[2,4,8,10,12]  :: [Integer]       -- filtrar

? takeWhile even [2,4,8,9,10,11,12]  -- takeWhile :: (a → Bool) → [a] → [a]
[2,4,8]  :: [Integer]                -- mayor segmento inicial
```

Algunas funciones predefinidas (2)

✓ Concatenación

```
? [1 .. 5] ++ [10 .. 13]                    -- (++) :: [a] → [a] → [a]
[1, 2, 3, 4, 5, 10, 11, 12, 13] :: [Integer]   -- de dos listas

? concat [ [1, 2, 3], [6, 7], [9, 10, 11, 12] ]   -- concat :: [[a]] → [a]
[1, 2, 3, 6, 7, 9, 10, 11, 12] :: [Integer]       -- de lista de listas
```

✓ Numéricas

```
? sum [1 .. 5]                  -- sum :: Num a ⇒ [a] → a
15 :: Integer                   -- sumar elementos

? product [1 .. 5]              -- product :: Num a ⇒ [a] → a
120 :: Integer                  -- multiplicar elementos
```

✓ Orden

```
? maximum [10, 4, 15, 2]        -- maximum :: Ord a ⇒ [a] → a
15 :: Integer                   -- máximo

? minimum [10, 4, 15, 2]        -- minimum :: Ord a ⇒ [a] → a
2 :: Integer                    -- mínimo
```

✓ Emparejamiento

```
? zip [1, 2, 3, 4] ['a', 'b', 'c']          -- zip :: [a]→[b]→[(a, b)]
[(1, 'a'), (2, 'b'), (3, 'c')] :: [ (Integer, Char) ]   -- emparejar

? unzip [ (1, 'a'), (2, 'b'), (3, 'c') ]    -- unzip :: [(a, b)]→([a], [b])
( [1, 2, 3], ['a', 'b', 'c'] ) :: ( [Integer], [Char] )   -- desemparejar

? zipWith (+) [1, 2, 3] [10, 20, 30]        -- zipWith :: (a → b → c) → [a]→[b]→[c]
[11, 22, 33] :: [Integer]                   -- emparejar con
```

8.3 Listas por comprensión

✓ Similar a los *conjuntos por comprensión* en *matemáticas*

✓ Sintaxis:

$$[\ expr \mid qual_1,\ qual_2,\ \ldots\ ,\ qual_n\ ]$$

✓ Un cualificador puede ser un:

◇ Un *generador* ($patrón \leftarrow expr$) con $expr$ de tipo lista:

? $[\ x \hat{}\ 2 \mid x \leftarrow [1\ ..\ 5]\ ]$
$[1,4,9,16,25] \ :: \ [Integer]$

◇ Un *filtro* o *guarda* (expresión de tipo $Bool$):

? $[\ x \mid x \leftarrow [1\ ..\ 10],\ even\ x\ ]$
$[2,4,6,8,10] \ :: \ [Integer]$

◇ Una *definición local* ($\mathbf{let}\ patrón\ =\ expr$):

? $[\ (x,y) \mid x \leftarrow [1\ ..\ 5],\ \mathbf{let}\ y\ =\ 2 * x\ ]$
$[\ (1,2),\ (2,4),\ (3,6),\ (4,8),\ (5,10)\ ] \ :: \ [(Integer, Integer)]$

✓ Varios *generadores* (los últimos cambian más rápido)

? $[\ (x,y) \mid x \leftarrow [1\ ..\ 3],\ y \leftarrow [10,20]\ ]$
$[\ (1,10),\ (1,20),\ (2,10),\ (2,20),\ (3,10),\ (3,20)\ ] \ :: \ [(Integer, Integer)]$

? $[\ (x,y) \mid y \leftarrow [10,20],\ x \leftarrow [1\ ..\ 3]\ ]$
$[\ (1,10),\ (2,10),\ (3,10),\ (1,20),\ (2,20),\ (3,20)\ ] \ :: \ [(Integer, Integer)]$

✓ Un *generador* o *def. local* puede depender de otro previo:

? $[\ (x,y) \mid x \leftarrow [1,2],\ y \leftarrow [x\ ..\ 3]\ ]$
$[\ (1,1),\ (1,2),\ (1,3),\ (2,2),\ (2,3)\ ] \ :: \ [(Integer, Integer)]$

? $[\ (x,y) \mid x \leftarrow [y,2],\ y \leftarrow [1\ ..\ 3]\ ]$
$ERROR\ -\ Undefined\ variable\ "y"$

Listas por comprensión (2)

Algunos ejemplos:

✓ La función map:

```
map      :: (a → b) → [a] → [b]
map f xs = [ f x | x ← xs ]
```

✓ La función $filter$:

```
filter      :: (a → Bool) → [a] → [a]
filter p xs = [ x | x ← xs, p x ]
```

✓ Divisores de un número natural:

```
divideA      :: Integer → Integer → Bool
d `divideA` n = (n `mod` d == 0)

divisores    :: Integer → [Integer]
divisores n = [ x | x ← [1 .. n], x `divideA` n ]
```

✓ Máximo común divisor de dos números:

```
mcd     :: Integer → Integer → Integer
mcd x y = maximum [ n | n ← divisores x, n `divideA` y ]
```

✓ Posiciones de un dato en una lista:

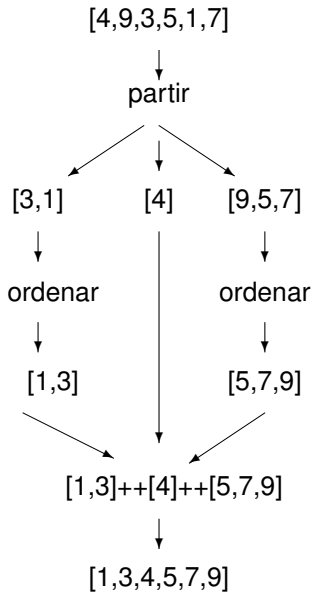
```
posiciones      :: Eq a ⇒ a → [a] → [Integer]
posiciones x xs = [ p | (p, y) ← zip [0 .. ] xs, x == y ]
```

```
? posiciones 'a' "la casa"
[1, 4, 6] :: [Integer]
```

8.4 Ejemplo: QuickSort

✓ Método

- ◇ Tomar el primer elemento de la lista (*pivote*).
- ◇ Partir la cola de la lista en dos: los elementos menores al pivote y los demás.
- ◇ Ordenar cada una de estas listas.
- ◇ A partir de las dos listas ordenadas, obtener la lista original ordenada concatenando la primera con el pivote y la segunda.



✓ En Haskell:

$$
\begin{array}{lcl}
qSort & :: & Ord\ a \Rightarrow [a] \rightarrow [a] \\
qSort\ [\,] & = & [\,] \\
qSort\ (p : xs) & = & qSort\ menores +\!\!+ [p] +\!\!+ qSort\ mayores \\
\quad \textbf{where} & & \\
\quad menores & = & [\ x \mid x \leftarrow xs, x < p\ ] \\
\quad mayores & = & [\ x \mid x \leftarrow xs, x >= p\ ]
\end{array}
$$

8.5 Funciones de plegado

✓ $foldr$ captura un patrón recursivo habitual sobre listas

✓ Consideremos

$$\begin{array}{lcl} suma & :: & [Integer] \to Integer \\ suma\ [\,] & = & 0 \\ suma\ (x : xs) & = & (+)\ x\ (suma\ xs) \end{array}$$

$$\begin{array}{lcl} producto & :: & [Integer] \to Integer \\ producto\ [\,] & = & 1 \\ producto\ (x : xs) & = & (*)\ x\ (producto\ xs) \end{array}$$

✓ Ambas siguen el mismo patrón:

$$\begin{array}{lcl} fun & :: & [a] \to b \\ fun\ [\,] & = & \boxed{e} \\ fun\ (x : xs) & = & \boxed{f}\ x\ (fun\ xs) \end{array}$$

✓ Una función de orden superior para este esquema

$$\begin{array}{l} foldr\ f\ e\ =\ fun \\ \quad \textbf{where} \\ \quad\quad fun\ [\,] = e \\ \quad\quad fun\ (x : xs) = f\ x\ (fun\ xs) \end{array} \qquad foldr :: (a \to b \to b) \to b \to ([a] \to b)$$

✓ O equivalentemente (ya que $fun \equiv foldr\ f\ e$)

$$\begin{array}{lcl} foldr & :: & (a \to b \to b) \to b \to [a] \to b \\ foldr\ f\ e\ [\,] & = & e \\ foldr\ f\ e\ (x : xs) & = & f\ x\ (foldr\ f\ e\ xs) \end{array}$$

✓ Las funciones originales como concreción de $foldr$:

$$\begin{array}{lcl} suma & :: & [Integer] \to Integer \\ suma & = & foldr\ (+)\ 0 \end{array}$$

$$\begin{array}{lcl} producto & :: & [Integer] \to Integer \\ producto & = & foldr\ (*)\ 1 \end{array}$$

Funciones de plegado (2)

✓ Es más fácil ver el comportamiento de $foldr$ del siguiente modo:

Comportamiento de $foldr$

$$foldr\ (\otimes)\ z\ [x_1, x_2, \ldots, x_n] \Longrightarrow x_1 \otimes (x_2 \otimes (\ldots \otimes (x_n \otimes z)))$$

Por ejemplo

$\underline{suma}\ [1,2,3]$
$\Longrightarrow$ {definición de $suma$}
$\underline{foldr\ (+)\ 0\ [1,2,3]}$
$\Longrightarrow$ {comportamiento de $foldr$}
$\ldots$
$1 + (2 + (3 + 0))$
$\Longrightarrow$ {por $(+)$}
$\ldots$
6

✓ Más ejemplos:

```
and :: [Bool] -> Bool   -- Conjunción de booleanos
and = foldr (&&) True

or :: [Bool] -> Bool    -- Disyunción de booleanos
or = foldr (||) False

concat :: [[a]] -> [a]  -- Concatenación de lista de listas
concat = foldr (++) []
```

Funciones de plegado (3)

✓ *foldl* pliega la lista de izquierda a derecha

$$\begin{array}{lcl} foldl & :: & (b \to a \to b) \to b \to [a] \to b \\ foldl\ f\ e\ [\,] & = & e \\ foldl\ f\ e\ (x : xs) & = & foldl\ f\ (f\ e\ x)\ xs \end{array}$$

Comportamiento de *foldl*

$$foldl\ (\otimes)\ z\ [x_1, x_2, \ldots, x_n] \Longrightarrow (((z \otimes x_1) \otimes x_2) \ldots \otimes x_{n-1}) \otimes x_n$$

✓ Por ejemplo:

$$\begin{array}{lcl} suma & :: & [Integer] \to Integer \\ suma & = & foldl\ (+)\ 0 \end{array}$$

$$\begin{array}{l} \quad suma\ [1,2,3] \\ \Longrightarrow \{\text{definición de } suma\} \\ \quad \underline{foldl\ (+)\ 0\ [1,2,3]} \\ \Longrightarrow \{\text{comportamiento de } foldl\} \\ \quad \ldots \\ \quad (0+1)+2)+3 \\ \Longrightarrow \{\text{por } (+)\} \\ \quad \ldots \\ \quad 6 \end{array}$$

Funciones de plegado (4)

- ✓ No todas las funciones se definen igual usando $foldr$ y $foldl$
- ✓ Para resolver un problema usando $foldr\ f\ z$
  - ◇ z será la solución para la lista vacía
  - ◇ f tomará
    - ○ como primer argumento la cabeza de la lista y
    - ○ como segundo argumento la solución del problema para la cola

$$\begin{array}{lcl} reverse & :: & [a] \to [a] \\ reverse & = & foldr\ (\lambda\ x\ xs \to xs +\!\!+ [x])\ [\,] \end{array}$$

- ✓ Para resolver un problema usando $foldl\ f\ z$
  - ◇ z será la solución para la lista vacía
  - ◇ f tomará
    - ○ como primer argumento la solución para el inicio de la lista y
    - ○ como segundo argumento el último elemento de la lista

$$\begin{array}{lcl} reverse & :: & [a] \to [a] \\ reverse & = & foldl\ (\lambda\ xs\ x \to x : xs)\ [\,] \end{array}$$

Objetivos del tema

El alumno debe:

- ✓ Conocer la notación de secuencias aritméticas para definir listas
- ✓ Conocer las funciones predefinidas para listas comentadas en el tema
- ✓ Conocer la notación de listas por comprensión. Debe saber calcular el resultado de este tipo de expresiones y debe saber definir funciones usando esta notación
- ✓ Conocer las funciones de plegado predefinidas $foldr$ y $foldl$
- ✓ Saber reducir expresiones en las que aparezcan funciones de plegado
- ✓ Saber definir funciones sobre listas como concreciones de las funciones de plegado