Programación Funcional

Instituto de Computación, Facultad de Ingeniería
Universidad de la República, Uruguay

Operaciones sobre listas

Listas

- Históricamente, el tipo de las listas ha sido uno de los más usados en Programación Funcional.

- Su relevancia en el contexto de PF se compara a la de los conjuntos en la matemática.

- Las listas son un tipo polimórfico.
  Constructores:

  $[\,] :: [a]$

  $(:) :: a \rightarrow [a] \rightarrow [a]$

Generador de listas

- La expresión $[m \,..\, n]$ denota la lista de valores entre m y n.
- Por ejemplo, $[1 \,..\, 4]$ denota la lista $[1, 2, 3, 4]$.
- Si $m > n$ denota la lista vacía $[\,]$.

Divisores de un número

- Ser divisor:

$$m \text{ `}divide\text{` } n = n \text{ `}mod\text{` } m == 0$$

Divisores de un número

- Ser divisor:

$$m \text{ `}divide\text{` } n = n \text{ `}mod\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

Divisores de un número

- Ser divisor:

$$m \text{ `}\mathit{divide}\text{` } n = n \text{ `}\mathit{mod}\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

$$\mathit{divisores}\ n = [d \mid d \leftarrow [1\,..\,n], d \text{ `}\mathit{divide}\text{` } n]$$

Divisores de un número

- Ser divisor:

$$m \text{ `}divide\text{` } n = n \text{ `}mod\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

$$divisores\ n = [d \mid d \leftarrow [1\,..\,n], d \text{ `}divide\text{` } n]$$

  equivale a

$$divisores\ n = filter\ (\text{`}divide\text{`}n)\ [1\,..\,n]$$

Divisores de un número

- Ser divisor:

$$m \text{ `}\mathit{divide}\text{` } n = n \text{ `}\mathit{mod}\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

$$\mathit{divisores}\ n = [d \mid d \leftarrow [1\,..\,n], d \text{ `}\mathit{divide}\text{` } n]$$

  equivale a

$$\mathit{divisores}\ n = \mathit{filter}\ (\text{`}\mathit{divide}\text{`}n)\ [1\,..\,n]$$

- Máximo común divisor:

Divisores de un número

- Ser divisor:

$$m \text{ `}divide\text{` } n = n \text{ `}mod\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

$$divisores\ n = [d \mid d \leftarrow [1 \mathinner{..} n], d \text{ `}divide\text{` } n]$$

equivale a

$$divisores\ n = filter\ (\text{`}divide\text{`}n)\ [1 \mathinner{..} n]$$

- Máximo común divisor:

$$mcd\ x\ y = maximum\ [d \mid d \leftarrow divisores\ x, d \text{ `}divide\text{` } y]$$

Divisores de un número

- Ser divisor:

$$m \text{ `}divide\text{` } n = n \text{ `}mod\text{` } m == 0$$

- Divisores de un entero positivo (incluyéndolo):

$$divisores\ n = [d \mid d \leftarrow [1 \,..\, n], d \text{ `}divide\text{` } n]$$

equivale a

$$divisores\ n = filter\ (\text{`}divide\text{`}n)\ [1 \,..\, n]$$

- Máximo común divisor:

$$mcd\ x\ y = maximum\ [d \mid d \leftarrow divisores\ x, d \text{ `}divide\text{` } y]$$

equivale a

$$mcd\ x\ y = maximum\ .\ filter\ (\text{`}divide\text{`}y)\ .\ divisores\ \$\ x$$

donde $f\ \$\ x = f\ x$

Números primos

- Determinar si un número es primo:

Números primos

- Determinar si un número es primo:

$$\textit{primo}\ n = \textit{divisores}\ n == [1, n]$$

Números primos

- Determinar si un número es primo:

$$\mathit{primo}\ n = \mathit{divisores}\ n == [1, n]$$

- Esto puede computarse de forma mas eficiente:

Números primos

- Determinar si un número es primo:

$$\mathit{primo}\ n = \mathit{divisores}\ n == [1, n]$$

- Esto puede computarse de forma mas eficiente:

$$\begin{aligned}\mathit{primo}\ n = {} & (n > 1) \\ & \&\& \\ & ([d \mid d \leftarrow [2 \mathinner{..} \mathit{isqrt}\ n], d\ \text{`}\mathit{divide}\text{`}\ n] == [\,])\end{aligned}$$

donde *isqrt n* computa el mayor entero cuyo cuadrado es menor o igual a n, o sea, $\mathit{isqrt}\ n = \lfloor\sqrt{n}\rfloor$.

Números perfectos

- Factores de un número (divisores menores que el número):

$$\mathit{factores}\ n = \mathit{filter}\ (\text{`}\mathit{divide}\text{`}\, n)\ [1 \mathinner{\ldotp\ldotp} n\ \text{`}\mathit{div}\text{`}\ 2]$$

Números perfectos

- Factores de un número (divisores menores que el número):

$$\mathit{factores}\ n = \mathit{filter}\ (`\mathit{divide}`\, n)\ [1\mathinner{..} n\ `\mathit{div}`\ 2]$$

- Se dice que un número es perfecto si la suma de sus factores es igual al número:

$$\mathit{perfecto}\ n = \mathit{sum}\ (\mathit{factores}\ n) == n$$

Números perfectos

- Factores de un número (divisores menores que el número):

  $$\mathit{factores}\ n = \mathit{filter}\ (\text{`}\mathit{divide}\text{`}\, n)\ [1\,..\,n\ \text{`}\mathit{div}\text{`}\ 2]$$

- Se dice que un número es perfecto si la suma de sus factores es igual al número:

  $$\mathit{perfecto}\ n = \mathit{sum}\ (\mathit{factores}\ n) == n$$

- Números perfectos entre 1 y 100:
  $> \mathit{filter}\ \mathit{perfecto}\ [1\,..\,100]$

Números perfectos

- Factores de un número (divisores menores que el número):

$$factores\ n = filter\ (\text{`}divide\text{`}\, n)\ [1\,..\,n\ \text{`}div\text{`}\ 2]$$

- Se dice que un número es perfecto si la suma de sus factores es igual al número:

$$perfecto\ n = sum\ (factores\ n) == n$$

- Números perfectos entre 1 y 100:
  $>filter\ perfecto\ [1\,..\,100]$
  $[6, 28]$

take y *drop*

- *take n xs*
  - retorna el segmento inicial de *xs* de largo *n*
  - si el largo de *xs* es menor que *n* retorna toda *xs*
  - si $n \leqslant 0$ retorna la lista vacía
- *drop n xs*
  - retorna la lista luego de quitar los primeros *n* elementos de *xs*
  - si el largo de *xs* es menor que *n* retorna la lista vacía
  - si $n \leqslant 0$ retorna la lista *xs*
- Propiedad que satisfacen *take* y *drop*:

$$take\ n\ xs \mathbin{+\!\!+} drop\ n\ xs == xs$$

Ejemplos

- Último elemento de una lista.

$$
\begin{array}{l}
\mathit{last} :: [a] \to a \\
\mathit{last}\ \mathit{xs} = \mathit{head}\ \$\ \mathit{drop}\ (\mathit{length}\ \mathit{xs} - 1)\ \mathit{xs}
\end{array}
$$

Ejemplos

- Último elemento de una lista.

$$\begin{array}{l} last :: [a] \to a \\ last\ xs = head\ \$\ drop\ (length\ xs - 1)\ xs \end{array}$$

- Partir una lista en dos en un determinado lugar.

$$\begin{array}{l} splitAt :: Int \to [a] \to ([a],[a]) \\ splitAt\ n\ xs = (take\ n\ xs, drop\ n\ xs) \end{array}$$

Ejemplos

- Último elemento de una lista.

$$\mathit{last} :: [a] \rightarrow a$$
$$\mathit{last}\ xs = \mathit{head}\ \$\ \mathit{drop}\ (\mathit{length}\ xs - 1)\ xs$$

- Partir una lista en dos en un determinado lugar.

$$\mathit{splitAt} :: \mathit{Int} \rightarrow [a] \rightarrow ([a],[a])$$
$$\mathit{splitAt}\ n\ xs = (\mathit{take}\ n\ xs, \mathit{drop}\ n\ xs)$$

- Intercalar un elemento en una posición de una lista.

$$\mathit{intercalar} :: \mathit{Int} \rightarrow a \rightarrow [a] \rightarrow [a]$$
$$\mathit{intercalar}\ n\ x\ xs = ys +\!\!+ [x] +\!\!+ zs$$
$$\quad \textbf{where}\ (ys, zs) = \mathit{splitAt}\ n\ xs$$

Función *trail*

Este es un ejemplo de diseño composicional.

Deseamos quedarnos con las últimas n líneas de un texto (dado como un string).

$$
\begin{aligned}
&trail :: Int \to String \to String \\
&trail\ n = unlines\ .\ reverse\ .\ take\ n\ .\ reverse\ .\ lines
\end{aligned}
$$

La función *lines* separa un texto en líneas; *unlines* es su inversa.

$$
\begin{aligned}
&lines \quad\ :: String \to [String] \\
&unlines :: [String] \to String
\end{aligned}
$$

takeWhile y *dropWhile*

- *takeWhile p xs*
  Prefijo más largo de una lista que satisface un predicado.

  $$takeWhile :: (a \to Bool) \to [a] \to [a]$$

- *dropWhile p xs*
  Resto de una lista *xs* después de aplicar *takeWhile p xs*.

  $$dropWhile :: (a \to Bool) \to [a] \to [a]$$

- Cortar una lista en dos de acuerdo a un predicado.

  $$\begin{aligned} &span :: (a \to Bool) \to [a] \to ([a],[a]) \\ &span\ p\ xs = (takeWhile\ p\ xs, dropWhile\ p\ xs) \end{aligned}$$

$zip :: [a] \rightarrow [b] \rightarrow [(a, b)]$

$zip :: [a] \to [b] \to [(a, b)]$

- Producto escalar de dos vectores (dados como listas):

$zip :: [a] \rightarrow [b] \rightarrow [(a, b)]$

- Producto escalar de dos vectores (dados como listas):

$$prodEsc :: Num\ a \Rightarrow [a] \rightarrow [a] \rightarrow a$$
$$prodEsc\ xs\ ys = sum\ .\ map\ (uncurry\ (*))\ \$\ zip\ xs\ ys$$

donde

$$sum\ [x_1, ..., x_n] = x_1 + ... + x_n$$

$zip :: [a] \to [b] \to [(a, b)]$

- Producto escalar de dos vectores (dados como listas):

$$
\begin{aligned}
&prodEsc :: Num\ a \Rightarrow [a] \to [a] \to a \\
&prodEsc\ xs\ ys = sum\ .\ map\ (uncurry\ (*))\ \$\ zip\ xs\ ys
\end{aligned}
$$

donde

$$sum\ [x_1, ..., x_n] = x_1 + ... + x_n$$

- Determinar si una secuencia es no decreciente:

$zip :: [a] \to [b] \to [(a, b)]$

- Producto escalar de dos vectores (dados como listas):

$$prodEsc :: Num\ a \Rightarrow [a] \to [a] \to a$$
$$prodEsc\ xs\ ys = sum\ .\ map\ (uncurry\ (*))\ \$\ zip\ xs\ ys$$

donde

$$sum\ [x_1, ..., x_n] = x_1 + ... + x_n$$

- Determinar si una secuencia es no decreciente:

$$nondec :: Ord\ a \Rightarrow [a] \to Bool$$
$$nondec\ xs = and\ .\ map\ (uncurry\ (\leqslant))\ \$\ zip\ xs\ (tail\ xs)$$

donde

$$and\ [b_1, ..., b_n] = b_1\ \&\&\ ...\ \&\&\ b_n$$

zipWith

En los ejemplos anteriores usamos un patrón común:

$$zipWith :: (a \to b \to c) \to [a] \to [b] \to [c]$$
$$zipWith\ f\ xs\ ys = map\ (uncurry\ f)\ (zip\ xs\ ys)$$

zipWith

En los ejemplos anteriores usamos un patrón común:

$$
\begin{array}{l}
zipWith :: (a \to b \to c) \to [a] \to [b] \to [c] \\
zipWith\ f\ xs\ ys = map\ (uncurry\ f)\ (zip\ xs\ ys)
\end{array}
$$

Entonces:

$$prodEsc\ xs\ ys = sum\ \$\ zipWith\ (*)\ xs\ ys$$

$$nondec\ xs = and\ \$\ zipWith\ (\leqslant)\ xs\ (tail\ xs)$$

$foldr :: (a \to b \to b) \to b \to [a] \to b$

Dada una función **f** y un elemento **e**,

$$foldr\ \mathbf{f}\ \mathbf{e}\ [x_1, ..., x_n]$$

es decir,

$$foldr\ \mathbf{f}\ \mathbf{e}\ (x_1 \mathbin{\mathbf{:}} x_2 \mathbin{\mathbf{:}} ... \mathbin{\mathbf{:}} x_n \mathbin{\mathbf{:}} \mathbf{[]})$$

retorna,

$$x_1\ \text{`}\mathbf{f}\text{`}\ (x_2\ \text{`}\mathbf{f}\text{`}\ (...(x_n\ \text{`}\mathbf{f}\text{`}\ \mathbf{e})...))$$

o que es lo mismo,

$$\mathbf{f}\ x_1\ (\mathbf{f}\ x_2\ (...(\mathbf{f}\ x_n\ e)...))$$

En otras palabras, *foldr* sustituye (**:**) por **f** y **[]** por **e**.

Ejemplos

$$sum :: Num\ a \Rightarrow [a] \rightarrow a$$
$$sum = foldr\ (+)\ 0$$

$$and :: [Bool] \rightarrow Bool$$
$$and = foldr\ (\&\&)\ True$$

$$\begin{array}{l} map :: (a \rightarrow b) \rightarrow [a] \rightarrow [b] \\ map\ f = foldr\ fcons\ [\,] \\ \quad \textbf{where}\ fcons\ x\ r = f\ x : r \end{array}$$

$$\begin{array}{l} filter :: (a \rightarrow Bool) \rightarrow [a] \rightarrow [a] \\ filter\ p = foldr\ op\ [\,] \\ \quad \textbf{where}\ op\ a\ r \begin{array}{|ll} p\ a & = a : r \\ otherwise & = r \end{array} \end{array}$$