Programación Funcional

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

Casos de uso de lazy evaluation

Operaciones no estrictas sobre booleanos

$$\begin{aligned} True\ \&\&\ b &= b \\ False\ \&\&\ b &= False \end{aligned}$$

$$\begin{aligned} True\ \|\ b &= True \\ False\ \|\ b &= b \end{aligned}$$

$$\begin{aligned} &\textbf{if}\ True\ \textbf{then}\ e\ \textbf{else}\ \_ = e \\ &\textbf{if}\ False\ \textbf{then}\ \_\ \textbf{else}\ e = e \end{aligned}$$

El **if** es estricto en la condición, pero no en las ramas.

$$\textbf{if}\ \perp\ \textbf{then}\ e\ \textbf{else}\ e' = \perp$$

Explotando lazy evaluation

Igualdad de listas.

$$
\begin{array}{llll}
[\,] & \texttt{==}\ [\,] & = & \mathit{True} \\
[\,] & \texttt{==}\ (\_ : \_) & = & \mathit{False} \\
(\_ : \_) & \texttt{==}\ [\,] & = & \mathit{False} \\
(x : xs) & \texttt{==}\ (y : ys) & = & (x \texttt{==} y)\ \&\&\ (xs \texttt{==} ys)
\end{array}
$$

Ejemplos:

$$[1,2] \texttt{==} [1,2] \xrightarrow{*} \mathit{True}$$

Explotando lazy evaluation

Igualdad de listas.

$$\begin{array}{llll} [\,] & == [\,] & = & \mathit{True} \\ [\,] & == (\_ : \_) & = & \mathit{False} \\ (\_ : \_) & == [\,] & = & \mathit{False} \\ (x : xs) & == (y : ys) & = & (x == y)\ \&\&\ (xs == ys) \end{array}$$

Ejemplos:

$$[1, 2] == [1, 2] \xrightarrow{*} \mathit{True}$$

$$[1, 2, \mathit{inf}\,] == [1, 2] \xrightarrow{*} \mathit{False}$$

Explotando lazy evaluation

Igualdad de listas.

$$
\begin{array}{lll}
[\,] & == [\,] & = \mathit{True} \\
[\,] & == (\_ : \_) & = \mathit{False} \\
(\_ : \_) & == [\,] & = \mathit{False} \\
(x : xs) & == (y : ys) & = (x == y)\ \&\&\ (xs == ys)
\end{array}
$$

Ejemplos:

$$[1, 2] == [1, 2] \xrightarrow{*} \mathit{True}$$

$$[1, 2, \mathit{inf}\,] == [1, 2] \xrightarrow{*} \mathit{False}$$

$$[1, \mathit{inf}\,, 2] == [1, 2, 3] \xrightarrow{*} \perp$$

Explotando lazy evaluation (2)

Determinar si dos árboles contienen los mismos valores en las hojas y en el mismo orden. Los árboles pueden tener formas distintas.

$$\textbf{data}\ Btree\ a = Leaf\ \ a \mid Fork\ (Btree\ a)\ (Btree\ a)$$

$$\begin{array}{l} eqleaves :: Ord\ a \Rightarrow Btree\ a \rightarrow Btree\ a \rightarrow Bool \\ eqleaves\ t\ t' = leaves\ t == leaves\ t' \end{array}$$

$$\begin{array}{ll} leaves\ (Leaf\ \ a) & = [a] \\ leaves\ (Fork\ l\ r) & = leaves\ l +\!\!+ leaves\ r \end{array}$$

Las listas *leaves* t y *leaves* t' se van construyendo a demanda y tanto como sea requerido por el operador de igualdad (==).

Ejemplo de evaluación

$$
\begin{array}{ll}
 & \mathit{eqleaves}\ (\mathit{Fork}\ (\mathit{Leaf}\ 2)\ (\mathit{Leaf}\ 3))\ (\mathit{Leaf}\ 2) \\
\longrightarrow & \mathit{leaves}\ (\mathit{Fork}\ (\mathit{Leaf}\ 2)\ (\mathit{Leaf}\ 3)) == \mathit{leaves}\ (\mathit{Leaf}\ 2) \\
\longrightarrow & \mathit{leaves}\ (\mathit{Leaf}\ 2) +\!\!+\ \mathit{leaves}\ (\mathit{Leaf}\ 3) == \mathit{leaves}\ (\mathit{Leaf}\ 2) \\
\xrightarrow{*} & 2 : ([\,] +\!\!+\ \mathit{leaves}\ (\mathit{Leaf}\ 3)) == 2 : [\,] \\
\longrightarrow & (2 == 2)\ \&\&\ ([\,] +\!\!+\ \mathit{leaves}\ (\mathit{Leaf}\ 3) == [\,]) \\
\xrightarrow{*} & \mathit{leaves}\ (\mathit{Leaf}\ 3) == [\,] \\
\longrightarrow & 3 : [\,] == [\,] \\
\longrightarrow & \mathit{False}
\end{array}
$$

Explotando lazy evaluation (3)

Determinar si algún elemento de una lista cumple un predicado.

$$
\begin{aligned}
&\mathit{any} :: (a \to \mathit{Bool}) \to [a] \to \mathit{Bool} \\
&\mathit{any}\ p\ [\,] && = \mathit{False} \\
&\mathit{any}\ p\ (x : xs) \mid p\ x && = \mathit{True} \\
&\phantom{\mathit{any}\ p\ (x : xs)} \mid \mathit{otherwise} && = \mathit{any}\ p\ xs
\end{aligned}
$$

Explotando lazy evaluation (3)

Determinar si algún elemento de una lista cumple un predicado.

$$
\begin{array}{lll}
\multicolumn{3}{l}{\mathit{any} :: (a \to \mathit{Bool}) \to [a] \to \mathit{Bool}} \\
\mathit{any}\ p\ [\,] & & = \mathit{False} \\
\mathit{any}\ p\ (x : xs) & \mid p\ x & = \mathit{True} \\
 & \mid \mathit{otherwise} & = \mathit{any}\ p\ xs
\end{array}
$$

Alternativamente,

$$\mathit{any}\ p = \mathit{not}\ .\ \mathit{null}\ .\ \mathit{filter}\ p$$

El procesamiento termina en cuanto un elemento pase el *filter*.

Explotando lazy evaluation (3)

Determinar si algún elemento de una lista cumple un predicado.

$$
\begin{aligned}
&\mathit{any} :: (a \to \mathit{Bool}) \to [a] \to \mathit{Bool} \\
&\mathit{any}\ p\ [\,] &&= \mathit{False} \\
&\mathit{any}\ p\ (x : xs) \mid p\ x &&= \mathit{True} \\
&\phantom{\mathit{any}\ p\ (x : xs)} \mid \mathit{otherwise} &&= \mathit{any}\ p\ xs
\end{aligned}
$$

Alternativamente,

$$\mathit{any}\ p = \mathit{not} \,.\, \mathit{null} \,.\, \mathit{filter}\ p$$

El procesamiento termina en cuanto un elemento pase el *filter*.

Otra forma

$$
\begin{aligned}
\mathit{any}\ p &= \mathit{foldr}\ (||)\ \mathit{False} \,.\, \mathit{map}\ p \\
&= \mathit{foldr}\ ((||) \,.\, p)\ \mathit{False}
\end{aligned}
$$

Estructuras infinitas

Lazy evaluation permite algo que en principio parece imposible: programar con estructuras infinitas.

Listas infinitas

- Un ejemplo clásico de listas infinitas es la lista *ones*:

$$\mathit{ones} = 1 : \mathit{ones}$$

Listas infinitas

- Un ejemplo clásico de listas infinitas es la lista *ones*:

$$ones = 1 : ones$$

- Evaluar *ones* produce una lista infinita de unos.

$$> ones$$
$$[1, 1, 1, 1, 1, 1, 1, 1, \ldots$$

Listas infinitas

- Un ejemplo clásico de listas infinitas es la lista *ones*:

$$ones = 1 : ones$$

- Evaluar *ones* produce una lista infinita de unos.

$$> ones$$
$$[1, 1, 1, 1, 1, 1, 1, 1, \ldots$$

- Gracias a lazy evaluation podemos procesar sólo lo necesario:

$$\begin{array}{l} head\ (tail\ ones) \\ \longrightarrow\ head\ (tail\ (1 : ones)) \\ \longrightarrow\ head\ ones \\ \longrightarrow\ head\ (1 : ones) \\ \longrightarrow\ 1 \end{array}$$

Listas infinitas (2)

Propiedad

Con lazy evaluation toda expresión se evalúa hasta lo que le requiera el contexto en que se encuentra.

- En este sentido, más que como una lista infinita, *ones* puede verse como una lista potencialmente infinita.
- Esta idea no está restringida a listas, se aplica a cualquier estructura de datos.

Programación modular

- Lazy evaluation permite separar control de datos.
- Por ejemplo, un lista de tres unos puede ser obtenida tomando los tres primeros elementos (control) de la lista *ones* (datos).

$$> \mathit{take}\ 3\ \mathit{ones}$$
$$[1,1,1]$$

Programación modular

- Lazy evaluation permite separar control de datos.

- Por ejemplo, un lista de tres unos puede ser obtenida tomando los tres primeros elementos (control) de la lista *ones* (datos).

  $> \mathit{take}\ 3\ \mathit{ones}$
  $[1, 1, 1]$

- Algún cuidado es igualmente necesario para evitar quedar en loop:

  $> \mathit{filter}\ (\leqslant 5)\ [1\,..]$
  $[1, 2, 3, 4, 5$

Programación modular

- Lazy evaluation permite separar control de datos.

- Por ejemplo, un lista de tres unos puede ser obtenida tomando los tres primeros elementos (control) de la lista *ones* (datos).

  $> \mathit{take}\ 3\ \mathit{ones}$
  $[1, 1, 1]$

- Algún cuidado es igualmente necesario para evitar quedar en loop:

  $> \mathit{filter}\ (\leqslant 5)\ [1\,..]$
  $[1, 2, 3, 4, 5$

- Mejor es algo de este estilo, que termina:

  $> \mathit{takeWhile}\ (\leqslant 5)\ [1\,..]$
  $[1, 2, 3, 4, 5]$

Funciones sobre listas infinitas: *iterate*

$$iterate :: (a \to a) \to a \to [a]$$
$$iterate\ f\ x = x : iterate\ f\ (f\ x)$$

$$> iterate\ f\ x$$
$$[x, f\ x, f\ (f\ x), f\ (f\ (f\ x)), \ldots$$

$$pot2 = iterate\ (*2)\ 1$$
$$> pot2$$
$$[1, 2, 4, 8, 16, \ldots$$

$$\begin{aligned} nats &= iterate\ (+1)\ 0 \\ [n\,..] &= iterate\ (+1)\ n \\ [m\,..\,n] &= takeWhile\ (\leqslant n)\ (iterate\ (+1)\ m) \end{aligned}$$

La Criba de Eratóstenes

La criba de Eratóstenes es un algoritmo que permite hallar todos los números primos.

La descripción del algoritmo es esencialmente la siguiente:

1. Liste todos los números naturales desde el 2.
2. Marque el primer elemento p de esta lista como primo.
3. Tache todos los elementos múltiplos de p de la lista.
4. Retorne al paso 2.

La Criba de Eratóstenes (2)

Una solución es:

$$primos = criba\ [2\,..]$$
$$criba\ xs = map\ head\ (iterate\ borrar\ xs)$$
$$borrar\ (p : xs) = [x \mid x \leftarrow xs, x\ \text{`}mod\text{`}\ p\ /\!\!= 0]$$

La Criba de Eratóstenes (2)

Una solución es:

$$primos = criba\ [2\,..]$$

$$criba\ xs = map\ head\ (iterate\ borrar\ xs)$$

$$borrar\ (p : xs) = [x \mid x \leftarrow xs, x\ `mod`\ p\ /\!=\ 0]$$

La definición de *criba* la podemos transformar:

$$\begin{aligned}
criba\ (p : xs) &= map\ head\ (iterate\ borrar\ (p : xs)) \\
&= map\ head\ ((p : xs) : iterate\ borrar\ (borrar\ (p : xs))) \\
&= p : map\ head\ (iterate\ borrar\ (borrar\ (p : xs))) \\
&= p : criba\ (borrar\ (p : xs))
\end{aligned}$$

La Criba de Eratóstenes (2)

Una solución es:

$$
\begin{aligned}
&\mathit{primos} = \mathit{criba}\ [2\,..] \\
&\mathit{criba}\ \mathit{xs} = \mathit{map}\ \mathit{head}\ (\mathit{iterate}\ \mathit{borrar}\ \mathit{xs}) \\
&\mathit{borrar}\ (p : \mathit{xs}) = [x \mid x \leftarrow \mathit{xs}, x\ \text{`}\mathit{mod}\text{`}\ p\ /\!= 0]
\end{aligned}
$$

La definición de *criba* la podemos transformar:

$$
\begin{aligned}
\mathit{criba}\ (p : \mathit{xs}) &= \mathit{map}\ \mathit{head}\ (\mathit{iterate}\ \mathit{borrar}\ (p : \mathit{xs})) \\
&= \mathit{map}\ \mathit{head}\ ((p : \mathit{xs}) : \mathit{iterate}\ \mathit{borrar}\ (\mathit{borrar}\ (p : \mathit{xs}))) \\
&= p : \mathit{map}\ \mathit{head}\ (\mathit{iterate}\ \mathit{borrar}\ (\mathit{borrar}\ (p : \mathit{xs}))) \\
&= p : \mathit{criba}\ (\mathit{borrar}\ (p : \mathit{xs}))
\end{aligned}
$$

llegando a la siguiente definición recursiva:

$$
\begin{aligned}
&\mathit{primos} = \mathit{criba}\ [2\,..] \\
&\mathit{criba}\ (p : \mathit{xs}) = p : \mathit{criba}\ [x \mid x \leftarrow \mathit{xs}, x\ \text{`}\mathit{mod}\text{`}\ p\ /\!= 0]
\end{aligned}
$$

Definiciones circulares

Lista de naturales en forma circular:

$$cnats = 0 : zipWith\ (+)\ cnats\ ones$$

Definiciones circulares

Lista de naturales en forma circular:

$$\mathit{cnats} = 0 : \mathit{zipWith}\ (+)\ \mathit{cnats}\ \mathit{ones}$$

Cómo funciona:

$$\begin{array}{lcl}
\mathit{cnats} & = & 0:1:2:3:4:5:6:7\ldots \\
\mathit{ones} & = & 1:1:1:1:1:1:1:1\ldots \\
\mathit{cnats} & = & 0:1:2:3:4:5:6:7:8\ldots
\end{array}$$

Definiciones circulares

Lista de naturales en forma circular:

$$cnats = 0 : zipWith\ (+)\ cnats\ ones$$

Cómo funciona:

$$\begin{array}{lcl}
cnats = & & 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 \ldots \\
ones \ \ = & & 1 : 1 : 1 : 1 : 1 : 1 : 1 : 1 \ldots \\
cnats = & 0 : 1 : 2 : 3 : 4 : 5 : 6 : 7 : 8 \ldots &
\end{array}$$

$$\begin{array}{lcl}
cnat & \longrightarrow & 0 : zipWith\ (+)\ cnats\ ones \\
& \longrightarrow & 0 : zipWith\ (+)\ (0 : \ldots)\ (1 : ones) \\
& \longrightarrow & 0 : 1 : zipWith\ (+)\ (1 : \ldots)\ ones \\
& \longrightarrow & 0 : 1 : zipWith\ (+)\ (1 : \ldots)\ (1 : ones) \\
& \longrightarrow & 0 : 1 : 2 : zipWith\ (+)\ (2 : \ldots)\ ones \\
& \longrightarrow & \ldots
\end{array}$$

Lista de factoriales

$$
\begin{aligned}
&\mathit{facts} = \mathit{map}\ \mathit{fact}\ [0\,..] \\
&\quad \textbf{where}\ \mathit{fact}\ 0 = 1 \\
&\qquad\qquad\ \ \mathit{fact}\ n = n * \mathit{fact}\ (n-1)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{facts}' = \mathit{map}\ \mathit{fst}\ \$\ \mathit{iterate}\ \mathit{next}\ (1,0) \\
&\quad \textbf{where}\ \mathit{next}\ (f, n) = (f * (n+1), n+1)
\end{aligned}
$$

$$
\mathit{cfacts} = 1 : \mathit{zipWith}\ (*)\ [1\,..]\ \mathit{cfacts}
$$

Lista de factoriales

$$\begin{array}{l} \mathit{facts} = \mathit{map}\ \mathit{fact}\ [0\,..] \\ \quad \textbf{where}\ \mathit{fact}\ 0 = 1 \\ \qquad\qquad \mathit{fact}\ n = n * \mathit{fact}\ (n - 1) \end{array}$$

$$\begin{array}{l} \mathit{facts}' = \mathit{map}\ \mathit{fst}\ \$\ \mathit{iterate}\ \mathit{next}\ (1, 0) \\ \quad \textbf{where}\ \mathit{next}\ (f, n) = (f * (n + 1), n + 1) \end{array}$$

$$\mathit{cfacts} = 1 : \mathit{zipWith}\ (*)\ [1\,..]\ \mathit{cfacts}$$

$$\begin{array}{lll} \mathit{cfacts} & \longrightarrow & 1 : \mathit{zipWith}\ (*)\ [1\,..]\ \mathit{cfacts} \\ & \longrightarrow & 1 : \mathit{zipWith}\ (*)\ [1\,..]\ (1 : ...) \\ & \longrightarrow & 1 : 1 : \mathit{zipWith}\ (*)\ [2\,..]\ (1 : ...) \\ & \longrightarrow & 1 : 1 : 2 : \mathit{zipWith}\ (*)\ [3\,..]\ (2 : ...) \\ & \longrightarrow & 1 : 1 : 2 : 6 : \mathit{zipWith}\ (*)\ [4\,..]\ (6 : ...) \\ & \longrightarrow & ... \end{array}$$

Secuencia de fibonacci

$$\begin{aligned}
&\mathit{fibos} = \mathit{map}\ \mathit{fib}\ [0\,..] \\
&\quad \textbf{where}\ \mathit{fib}\ 0 = 0 \\
&\quad\phantom{\textbf{where}}\ \mathit{fib}\ 1 = 1 \\
&\quad\phantom{\textbf{where}}\ \mathit{fib}\ n = \mathit{fib}\ (n-1) + \mathit{fib}\ (n-2)
\end{aligned}$$

$$\begin{aligned}
&\mathit{fibos}' = \mathit{map}\ \mathit{fst}\ \$\ \mathit{iterate}\ \mathit{next}\ (0,1) \\
&\quad \textbf{where}\ \mathit{next}\ (m,n) = (n, m+n)
\end{aligned}$$

$$\mathit{cfibos} = 0 : 1 : \mathit{zipWith}\ (+)\ \mathit{cfibos}\ (\mathit{tail}\ \mathit{cfibos})$$

Secuencia de fibonacci

$$
\begin{aligned}
&\mathit{fibos} = \mathit{map}\ \mathit{fib}\ [0\,..]\\
&\quad \textbf{where}\ \mathit{fib}\ 0 = 0\\
&\qquad\qquad \mathit{fib}\ 1 = 1\\
&\qquad\qquad \mathit{fib}\ n = \mathit{fib}\ (n-1) + \mathit{fib}\ (n-2)
\end{aligned}
$$

$$
\begin{aligned}
&\mathit{fibos}' = \mathit{map}\ \mathit{fst}\ \$\ \mathit{iterate}\ \mathit{next}\ (0,1)\\
&\quad \textbf{where}\ \mathit{next}\ (m,n) = (n, m+n)
\end{aligned}
$$

$$\mathit{cfibos} = 0 : 1 : \mathit{zipWith}\ (+)\ \mathit{cfibos}\ (\mathit{tail}\ \mathit{cfibos})$$

$$
\begin{array}{lll}
\mathit{cfibos} & = & \quad 0 : 1 : 1 : 2 : 3 : 5 :\ \ldots\\
\mathit{tail}\ \mathit{cfibos} & = & \quad 1 : 1 : 2 : 3 : 5 : 8 :\ \ldots\\
\color{blue}{\mathit{cfibos}} & = & \color{blue}{0 : 1 : 1 : 2 : 3 : 5 : 8 : 13 : \ldots}
\end{array}
$$

Árboles infinitos

$$\mathit{btree1} = \mathit{Fork}\ (\mathit{Leaf}\ 5)\ \mathit{btree1}$$
$$\mathit{btree2} = \mathit{Fork}\ (\mathit{Fork}\ \mathit{btree2}\ (\mathit{Leaf}\ 2))\ (\mathit{Fork}\ (\mathit{Leaf}\ 3)\ \mathit{btree1})$$

$$\begin{aligned} &\mathit{leftmost}\ (\mathit{Leaf}\ x) &&= x \\ &\mathit{leftmost}\ (\mathit{Fork}\ l\ r) &&= \mathit{leftmost}\ l \end{aligned}$$

Árboles infinitos

$$btree1 = Fork\ (Leaf\ 5)\ btree1$$
$$btree2 = Fork\ (Fork\ btree2\ (Leaf\ 2))\ (Fork\ (Leaf\ 3)\ btree1)$$

$$\begin{aligned} leftmost\ (Leaf\ x) &= x \\ leftmost\ (Fork\ l\ r) &= leftmost\ l \end{aligned}$$

$$ejemplo1 = leftmost\ btree1$$

Árboles infinitos

$$btree1 = Fork\ (Leaf\ 5)\ btree1$$
$$btree2 = Fork\ (Fork\ btree2\ (Leaf\ 2))\ (Fork\ (Leaf\ 3)\ btree1)$$

$$\begin{aligned} leftmost\ (Leaf\ x)\ &= x \\ leftmost\ (Fork\ l\ r) &= leftmost\ l \end{aligned}$$

$$ejemplo1 = leftmost\ btree1$$

$$ejemplo2 = leftmost\ btree2$$

Árboles infinitos

$$btree1 = Fork\ (Leaf\ 5)\ btree1$$
$$btree2 = Fork\ (Fork\ btree2\ (Leaf\ 2))\ (Fork\ (Leaf\ 3)\ btree1)$$

$$leftmost\ (Leaf\ x) \ = x$$
$$leftmost\ (Fork\ l\ r) = leftmost\ l$$

$$ejemplo1 = leftmost\ btree1$$

$$ejemplo2 = leftmost\ btree2$$

$$ejemplo3 = take\ 5\ \$\ map\ (*2)\ \$\ leaves\ btree1$$

Árboles infinitos

$$btree1 = Fork\ (Leaf\ 5)\ btree1$$
$$btree2 = Fork\ (Fork\ btree2\ (Leaf\ 2))\ (Fork\ (Leaf\ 3)\ btree1)$$

$$\begin{aligned} leftmost\ (Leaf\ x)\ &= x \\ leftmost\ (Fork\ l\ r) &= leftmost\ l \end{aligned}$$

$$ejemplo1 = leftmost\ btree1$$

$$ejemplo2 = leftmost\ btree2$$

$$ejemplo3 = take\ 5\ \$\ map\ (*2)\ \$\ leaves\ btree1$$

$$ejemplo4 = take\ 5\ \$\ map\ (*2)\ \$\ leaves\ btree2$$