

CURSO DE PROGRAMACION SCALA

Sesión 11

Sergio Couto Catoira

Índice

- › Fold sobre listas no-estrictas (streams)
- › Procesamiento de listas no-estrictas
- › Streams infinitos

Función fold

- > La función `exists` de la clase anterior hace la recursión explícitamente. Se puede generar un `fold` para implementar recursión generalizada.
- > Ejercicio: Implementa la función `foldRight` de forma lazy
 - `def foldRight[B](z: => B)(f:(A, =>B) => B): B`
- > Ejercicio: implementa `foldLeft` del mismo modo

Función fold

- > Ejercicio: Define la función `exists` usando `foldRight` y `foldLeft`
- > Ejercicio: Define la función `forAll` usando `fold` (el que quieras)

Función fold

- > Ejercicio: Define la función headOption usando fold
- > Ejercicio: Define las siguientes funciones usando fold
 - takeWhile
 - map
 - filter
 - append (el argumento de append debe ser no estricto)
 - flatMap

Procesamiento de streams

Listing 5.3 Program trace for Stream

```
Stream(1,2,3,4).map(_ + 10).filter(_ % 2 == 0).toList
cons(11, Stream(2,3,4).map(_ + 10)).filter(_ % 2 == 0).toList
Stream(2,3,4).map(_ + 10).filter(_ % 2 == 0).toList
cons(12, Stream(3,4).map(_ + 10)).filter(_ % 2 == 0).toList
12 :: Stream(3,4).map(_ + 10).filter(_ % 2 == 0).toList
12 :: cons(13, Stream(4).map(_ + 10)).filter(_ % 2 == 0).toList
12 :: Stream(4).map(_ + 10).filter(_ % 2 == 0).toList
12 :: cons(14, Stream().map(_ + 10)).filter(_ % 2 == 0).toList
12 :: 14 :: Stream().map(_ + 10).filter(_ % 2 == 0).toList
12 :: 14 :: List()
```

Apply filter to the first element.

Apply map to the first element.

Apply map to the second element.

Apply filter to the second element. Produce the first element of the result.

Apply filter to the fourth element and produce the final element of the result.

map and filter have no more work to do, and the empty stream becomes the empty list.

Ejercicio

- > Define una función `find` que devuelva un `option` con el primer elemento que cumpla un predicado

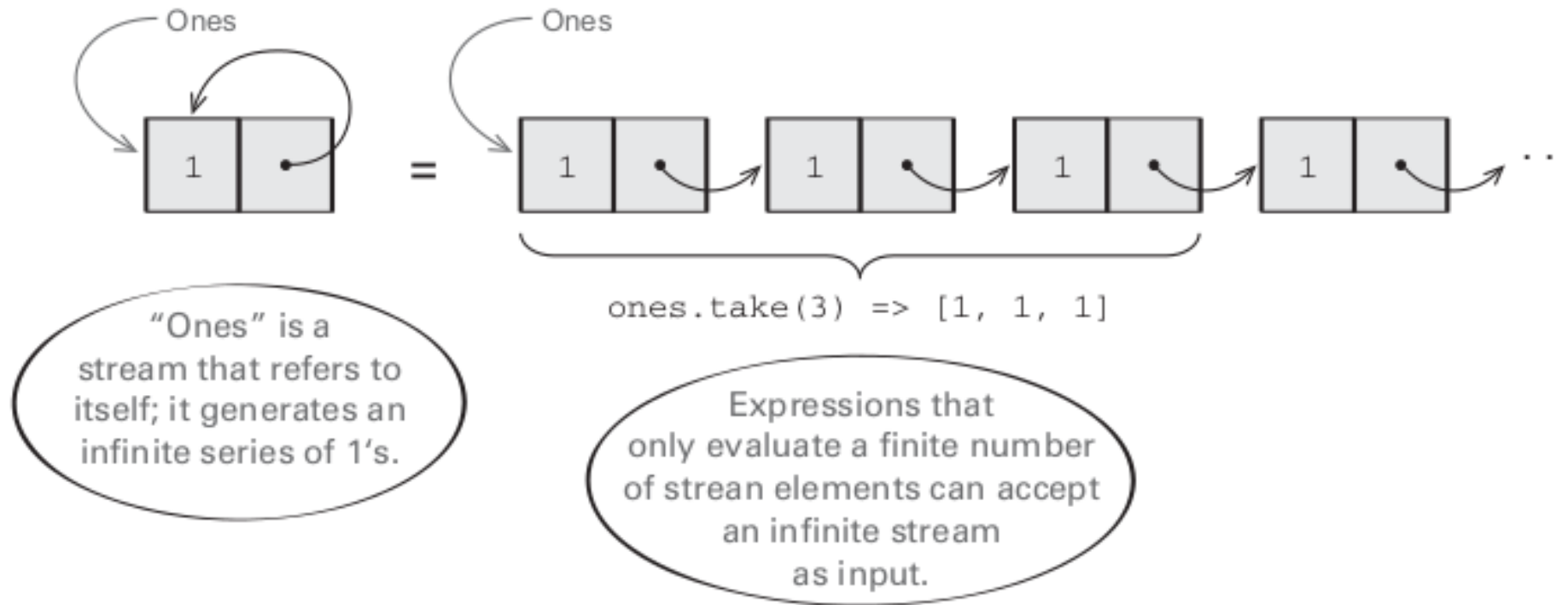
Pista: usa `filter` & `headOption`

Streams infinitos

- > Gracias a que no se computa la cola del Stream hasta que es necesario, podríamos definir Streams infinitos
- > Aunque sean infinitos, las funciones evalúan únicamente la parte que necesitan:
 - `ones.take(5)`
 - `ones.exists(_ % 2 != 0)`
 - `ones.map(_ + 1).exists(_ % 2 == 0)`
 - `ones.takeWhile(_ == 1)`
 - `ones.forAll(_ != 1)`

Streams infinitos

An infinite stream



Many functions can be evaluated using finite resources even if their inputs generate infinite sequences.

Streams infinitos

- > Generaliza ones en una función constant que devuelva un Stream infinito del valor indicado
- > Define una función from que genere un stream infinito 1 incremental de enteros, empezando en n. (n,n+1,n+2,n+3...)
- > Define una función fibs que genere un stream infinito con la secuencia de fibonacci
 - 0,1,1,2,3,5,8...