9° Aquelarre Matemático

Datos infinitos con el mínimo esfuerzo

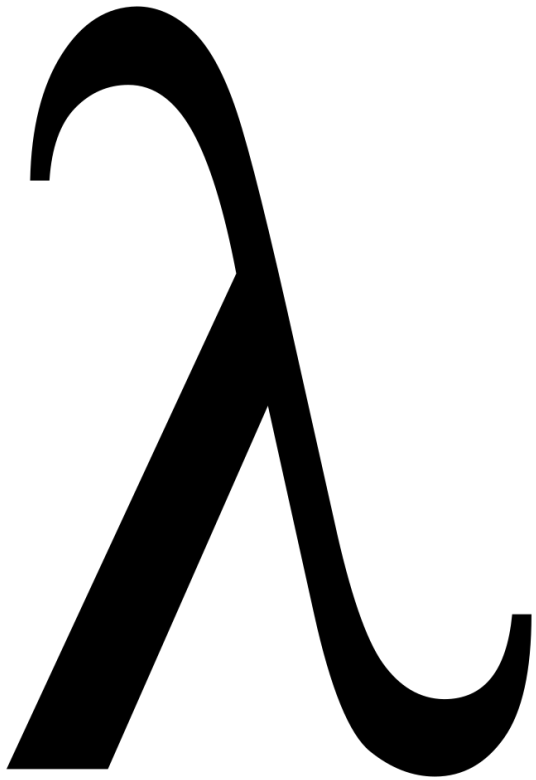
Manuel Soto Romero

manu@ciencias.unam.mx

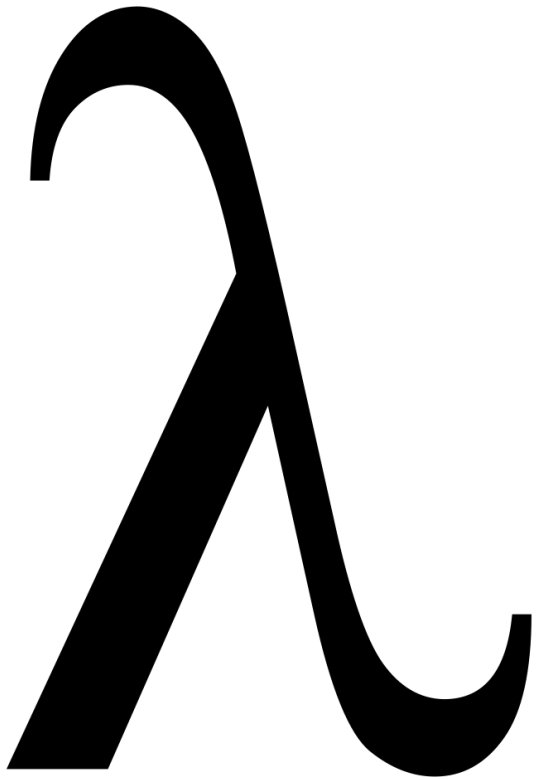
Facultad de Ciencias, UNAM

15 de octubre de 2018

Lenguajes de Programación



Lenguajes de Programación



Área de las Ciencias de la Computación que estudia modelos de lenguajes:

- Diseño
- Implementación
- Verificación formal

Evaluación glotona y perezosa

JAVA

```
public int foo(int n) {
    int nunca = fact(1000000);
    return n + 5;
}
```

Evaluación glotona y perezosa

JAVA

```
public int foo(int n) {
    int nunca = fact(1000000);
    return n + 5;
}
```

- El código usa evaluación glotona (*eager evaluation*)
- Se genera un desborde en la pila de ejecución del programa (*stack overflow*)

Evaluación glotona y perezosa

JAVA

```
public int foo(int n) {
    int nunca = fact(1000000);
    return n + 5;
}
```

HASKELL

```
foo :: Int -> Int
foo n =
    let nunca = fact 1000000 in
        n + 5
```

- El código usa evaluación glotona (*eager evaluation*)
- Se genera un desborde en la pila de ejecución del programa (*stack overflow*)

Evaluación glotona y perezosa

JAVA

```
public int foo(int n) {
    int nunca = fact(1000000);
    return n + 5;
}
```

- El código usa evaluación glotona (*eager evaluation*)
- Se genera un desborde en la pila de ejecución del programa (*stack overflow*)

HASKELL

```
foo :: Int -> Int
foo n =
    let nunca = fact 1000000 in
        n + 5
```

- El código usa evaluación perezosa (*lazy evaluation*)
- El programa termina correctamente pues no se evalúan expresiones que no son necesarias o que no se usan

Listas

Definición

Una ***lista*** es alguna de las siguientes:

- La lista vacía es una lista y se representa por `[]`.
- Si `x`$\in A$ y `xs` es una lista con elementos en A, entonces `x:xs` lo es también. Llamamos a `x` la cabeza de la lista y a `xs` el resto.
- Son todas.

Listas

Definición

Una ***lista*** es alguna de las siguientes:

- La lista vacía es una lista y se representa por `[]`.
- Si `x`$\in A$ y `xs` es una lista con elementos en A, entonces `x:xs` lo es también. Llamamos a `x` la cabeza de la lista y a `xs` el resto.
- Son todas.

```
> 1:2:3:4:5:[]
```

Listas

Definición

Una ***lista*** es alguna de las siguientes:

- La lista vacía es una lista y se representa por `[]`.
- Si `x`$\in A$ y `xs` es una lista con elementos en A, entonces `x:xs` lo es también. Llamamos a `x` la cabeza de la lista y a `xs` el resto.
- Son todas.

```
> 1:2:3:4:5:[]
[1,2,3,4,5]
```

Listas

Definición

Una ***lista*** es alguna de las siguientes:

- La lista vacía es una lista y se representa por `[]`.
- Si `x`$\in A$ y `xs` es una lista con elementos en A, entonces `x:xs` lo es también. Llamamos a `x` la cabeza de la lista y a `xs` el resto.
- Son todas.

```
> 1:2:3:4:5:[]
[1,2,3,4,5]

> [1,2,3,4,5]
```

Listas

Definición

Una ***lista*** es alguna de las siguientes:

- La lista vacía es una lista y se representa por `[]`.
- Si `x`$\in A$ y `xs` es una lista con elementos en A, entonces `x:xs` lo es también. Llamamos a `x` la cabeza de la lista y a `xs` el resto.
- Son todas.

```
> 1:2:3:4:5:[]
[1,2,3,4,5]

> [1,2,3,4,5]
[1,2,3,4,5]
```

Definiciones por comprensión

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \land x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
> [2*x | x <- [0..100000]]
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
> [2*x | x <- [0..100000]]
[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,...]
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
> [2*x | x <- [0..100000]]
[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,...]
> let pares = [2*x | x <- [0..100000]]
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
> [2*x | x <- [0..100000]]
[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,...]
> let pares = [2*x | x <- [0..100000]]
> pares !! 3
```

Definiciones por comprensión

Al igual que los conjuntos en matemáticas, es posible defiinir listas por comprensión.

$$\{2x \mid x \in \mathbb{N} \wedge x \leq 5\}$$

$$\{2x \mid x \in \{0, 1, 2, 3, 4, 5\}\}$$

```
> [2*x | x <- [0..5]]
[0,2,4,6,8,10]
> [2*x | x <- [0..100000]]
[0,2,4,6,8,10,12,14,16,18,20,22,24,26,28,30,32,34,36,38,...]
> let pares = [2*x | x <- [0..100000]]
> pares !! 3
> 6
```

¿Cómo funciona?

¿Cómo funciona?

Este comportamiento se logra gracias a la evaluación perezosa del lenguaje, pero...

¿Cómo funciona?

Este comportamiento se logra gracias a la evaluación perezosa del lenguaje, pero...

¿Cómo se logra en otros lenguajes de programación?

¿Cómo funciona?

Este comportamiento se logra gracias a la evaluación perezosa del lenguaje, pero...

¿Cómo se logra en otros lenguajes de programación?

Podemos simular evaluación perezosa usando *thunks.* Un *thunk* es una función que no recibe parámetros. La idea consiste en construir una lista cuyo resto sea encapsulado dentro de un *thunk*, una vez que éste sea evaluado, se generará el resto de la lista, en caso contrario, se mantiene la función sin evaluar.

Listas infinitas en Racket

Listas infinitas en Racket

En Racket, un *thunk* tiene la siguiente forma:

Listas infinitas en Racket

En Racket, un *thunk* tiene la siguiente forma:

```
(λ () <cuerpo>)
```

Listas infinitas en Racket

En Racket, un *thunk* tiene la siguiente forma:

$$(\lambda\ ()\ \texttt{<cuerpo>})$$

Este lenguaje provee una primitiva *thunk* que permite definir funciones de este tipo de manera sencilla. Para construir *listas infinitas*, se crea un nuevo tipo de dato a partir de la definición de lista, con algunas modificaciones.

```
(define-type ListaInf
   [sempty]
   [scons (cabeza any?) (resto thunk?)])
```

Listas infinitas en Racket

En Racket, un *thunk* tiene la siguiente forma:

$$(\lambda\ ()\ \texttt{<cuerpo>})$$

Este lenguaje provee una primitiva *thunk* que permite definir funciones de este tipo de manera sencilla. Para construir *listas infinitas*, se crea un nuevo tipo de dato a partir de la definición de lista, con algunas modificaciones.

```
(define-type ListaInf
   [sempty]
   [scons (cabeza any?) (resto thunk?)])
```

Ejemplo

Ejemplo

Una lista infinita que genera números naturales. Se provee el primer elemento y el *thunk* encierra una función que continúa generando el resto de la lista.

```
;; Función que genera una lista infinita con números
;; naturales a partir de n.
(define (genera-naturales n)
   (scons n (thunk (genera-naturales (+ n 1)))))

;; Lista infinita que genera números naturales
(scons 0 (thunk (genera-naturales 1)))
```

Ejemplo

Una lista infinita que genera números naturales. Se provee el primer elemento y el *thunk* encierra una función que continúa generando el resto de la lista.

```
;; Función que genera una lista infinita con números
;; naturales a partir de n.
(define (genera-naturales n)
   (scons n (thunk (genera-naturales (+ n 1)))))

;; Lista infinita que genera números naturales
(scons 0 (thunk (genera-naturales 1)))
```

¿Cómo se accede a los elementos de estas listas?

Obteniendo elementos de listas infinitas

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Ejemplo:

```
> (define naturales (scons 0 (thunk (genera-naturales 1))))
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Ejemplo:

```
> (define naturales (scons 0 (thunk (genera-naturales 1))))
> (shead naturales)
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Ejemplo:

```
> (define naturales (scons 0 (thunk (genera-naturales 1))))
> (shead naturales)
> 0
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Ejemplo:

```
> (define naturales (scons 0 (thunk (genera-naturales 1))))
> (shead naturales)
> 0
> (stail naturales)
```

Obteniendo elementos de listas infinitas

Primer elemento

```
(define (shead lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) h))
```

Resto de la lista

```
(define (stail lst)
   (match lst
      [(sempty) (error "Lista vacía")]
      [(scons h t) (t)]))
```

Ejemplo:

```
> (define naturales (scons 0 (thunk (genera-naturales 1))))
> (shead naturales)
> 0
> (stail naturales)
> (scons 1 (thunk (genera-naturales 2)))
```

Otras funciones útiles

Otras funciones útiles

N-ésimo elemento

```
(define (snth n lst)
    (match n
        [0 (shead lst)]
        [n (snth (stail lst) (- n 1))]))
```

Otras funciones útiles

N-ésimo elemento

```
(define (snth n lst)
   (match n
      [0 (shead lst)]
      [n (snth (stail lst) (- n 1))]))
```

Primeros n-elementos (en forma de lista finita)

```
(define (stake n lst)
   (match* (n lst)
      [(0 _) empty]
      [(_ (sempty)) empty)]
      [(n (scons h t)) (cons h (stake (stail lst) (- n 1)))]))
```