

Tema 3: Paradigma Funcional (II)

Lenguajes, Tecnologías y Paradigmas de
Programación

Índice

Introducción a la Programación Funcional

PARTE I: Tipos en Programación Funcional

1. Tipos funcionales. Tipos algebraicos.
2. Tipos predefinidos.
3. Polimorfismo: genericidad, sobrecarga y coerción. Herencia en Haskell.

PARTE II: Modelos de computación funcional.

4. **Modelo operacional.**

PARTE III: Características avanzadas

5. Funciones anónimas y composición de funciones.
6. Iteradores y compresores (foldl, foldr).

Modelo operacional

- Un **programa funcional** consta de:
 - Una lista de **ecuaciones de definición de función** (con sus definiciones de tipo correspondientes)
 - Una **expresión inicial** (sin variables libres)
- La **ejecución** de un programa funcional se efectúa evaluando la **expresión inicial** asociada
- Dicha evaluación consiste en el encadenamiento de pasos de **reducción**

Modelo operacional

- Utilizamos el concepto de **sustitución** para formalizar el **paso de parámetros** como un emparejamiento (matching) entre la expresión a evaluar y la parte izquierda de la ecuación $I=r$ empleada en el paso de ejecución.
- Una sustitución σ es una función de variables en expresiones tal que $\sigma(x) \neq x$ sólo para un conjunto finito de variables.
- Representaremos la sustitución indicando únicamente los enlaces no triviales $\{x_1 \rightarrow t_1, \dots, x_n \rightarrow t_n\}$ con $x_i \neq t_i$.
Ejemplo: $\sigma = \{x \rightarrow 1, y \rightarrow 0\}$ es una sustitución
- La identidad o sustitución vacía se representa como ε

Modelo operacional

- La aplicación de una sustitución σ a una expresión e se denomina **instanciación**, $\sigma(e)$

Ejemplo 1:

$$\sigma = \{x \rightarrow 1, y \rightarrow 0\}$$

$$e = f(x, g(y))$$

$$\sigma(e) = f(1, g(0))$$

Ejemplo 2

$$\sigma = \{x \rightarrow s(y), y \rightarrow 0\}$$

$$e = f(x, y)$$

$$\sigma(e) = f(s(y), 0)$$

Reducción

- Un **redex** es una instancia $\sigma(l)$ de la parte izquierda l de una ecuación $l = r$
(o, si es condicional, $l \mid c = r$)
- Una expresión e se **reduce** a e' si:
 - contiene un redex $\sigma(l)$ de una ecuación $l \mid c = r$
 - se satisface (es decir, se reduce a True) la condición c después que se le aplica σ
 - e' se obtiene reemplazando en e el redex $\sigma(l)$ por $\sigma(r)$
- Las expresiones que no pueden reducirse se llaman **formas normales**.

Reducción

Ejemplo:

seisveces 1 → doble (triple 1)



Redex

Ecuación:

seisveces x = doble (triple x)

Sustitución:

{ $x \rightarrow 1$ }

Reducción

Ejemplo:

Redex

seisveces 1 → doble (triple 1)
→ doble (3^*1)

Ecuación:

$$\text{triple } y = 3 * y$$

Sustitución:

$$\{y \rightarrow 1\}$$

Reducción

Ejemplo:

seisveces 1 → doble (triple 1)

→ doble (3^*1)

→ doble 3

↗ Redex

Ecuación:

predefinida: producto

Reducción

Ejemplo:

seisveces 1 → doble (triple 1)
→ doble ($3*1$)
→ doble 3 ↗
→ $3+3$

Redex

Ecuación:

$$\text{doble } x = x+x$$

Sustitución:

$$\{x \rightarrow 3\}$$

Reducción

Ejemplo:

seisveces 1 → doble (triple 1)
→ doble ($3*1$)
→ doble 3
→ $3+3$ ↴
→ 6

Redex

Ecuación:

predefinida: suma

Reducción

Ejemplo:

seisveces 1 → doble (triple 1)
→ doble ($3*1$)
→ doble 3
→ $3+3$
→ **6**

Forma normal

Resumen:

Programa funcional

Lista de Ecuaciones:

suma 0 x = x

suma (S x) y = S (suma x y)

Expresión inicial:

suma (suma 0 0) 0

No puede tener
variables libres

Resumen:

Programa funcional

Lista de Ecuaciones:

suma 0 x = x

suma (S x) y = S suma x y)

Expresión inicial:

suma (suma 0 0) 0

No puede tener
variables libres

redex seleccionado
por la estrategia de
reducción

Resumen:

Programa funcional

Lista de Ecuaciones:

$$\text{suma } 0 \ x = x$$

$$\text{suma } (S \ x) \ y = S \ (\text{suma } x \ y)$$

Expresión inicial:

$$\text{suma } (\text{suma } 0 \ 0) \ 0$$

No puede tener variables libres

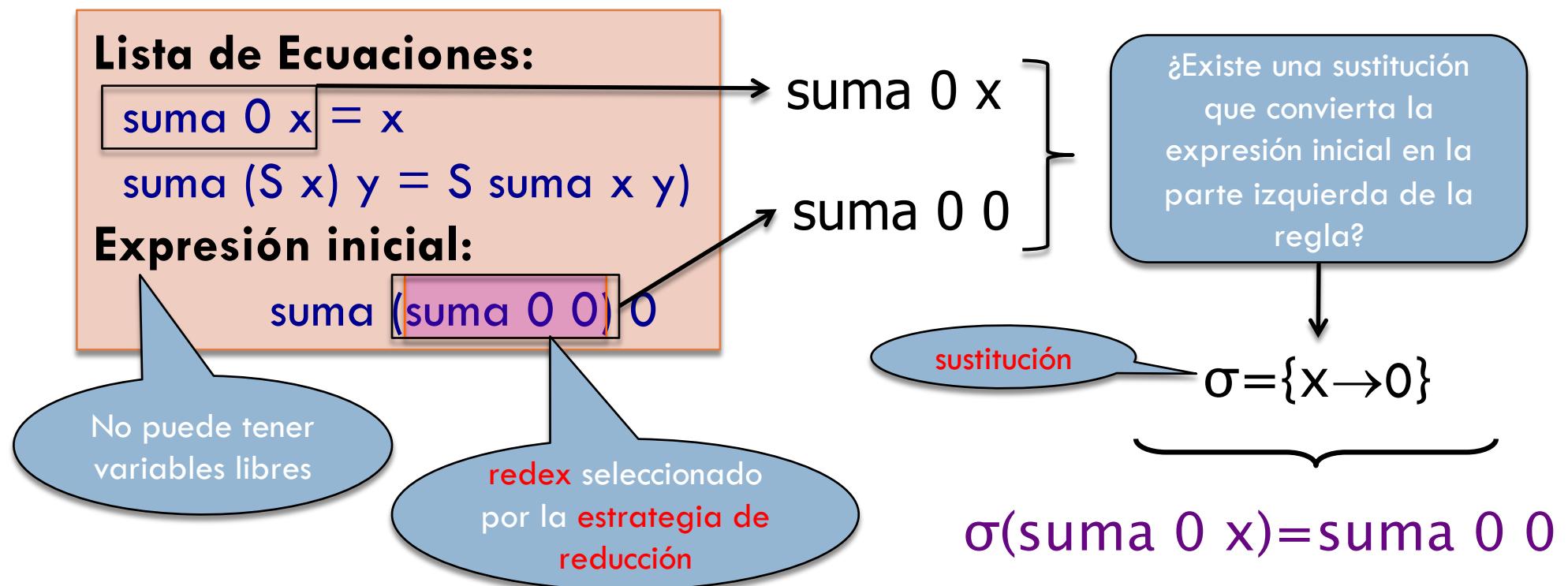
redex seleccionado por la estrategia de reducción

$$\begin{array}{l} \xrightarrow{\quad} \text{suma } 0 \ x \\ \xrightarrow{\quad} \text{suma } 0 \ 0 \end{array}$$

¿Existe una sustitución que convierta la expresión inicial en la parte izquierda de la regla?

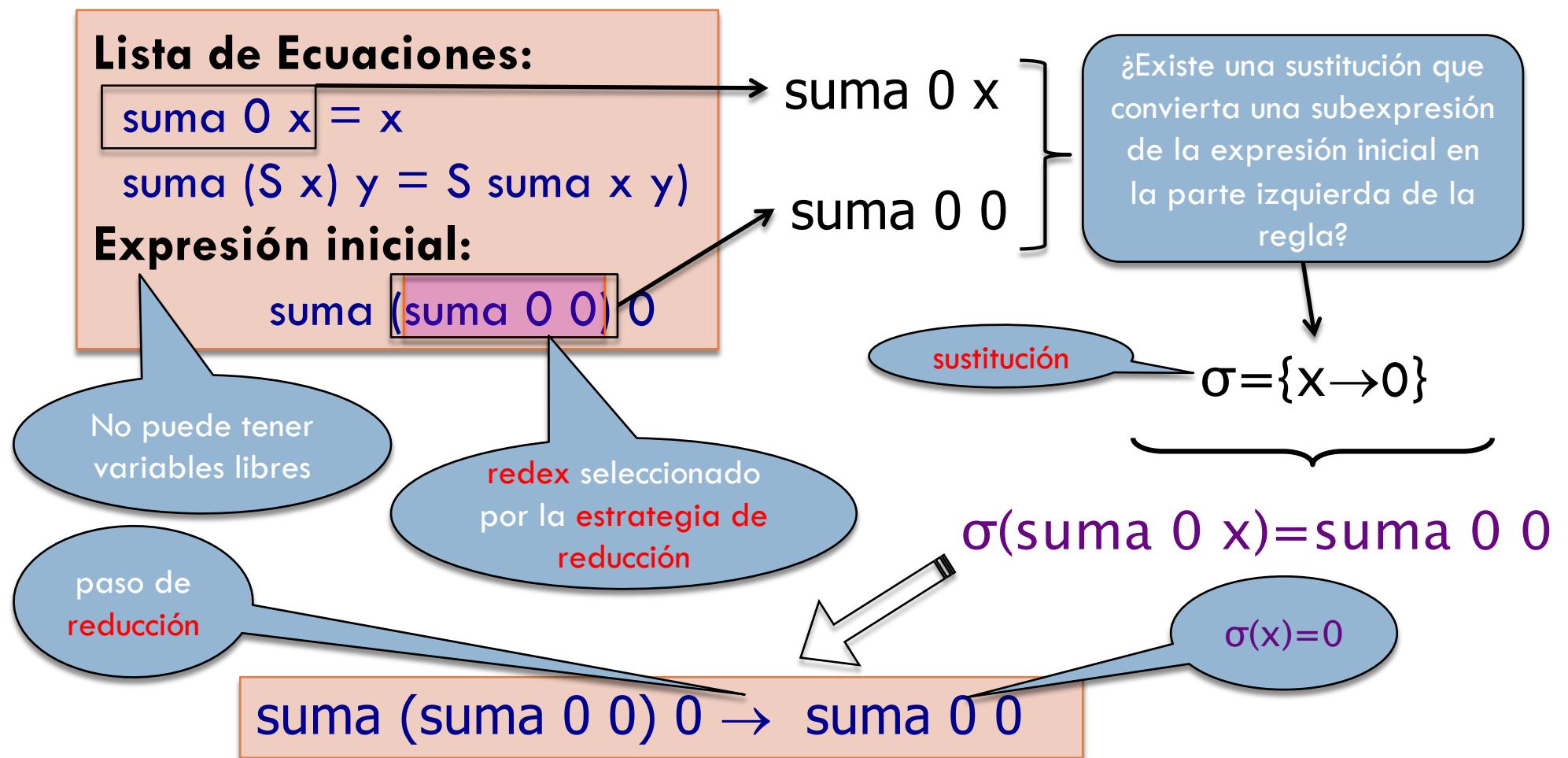
Resumen:

Programa funcional



Resumen:

Programa funcional



Evaluación

- La **evaluación** de una expresión consiste en aplicar sucesivos **pasos de reducción** hasta obtener una **forma normal**

Evaluación

- La **evaluación** de una expresión consiste en aplicar sucesivos pasos de **reducción** hasta obtener una **forma normal**
- Según la **estrategia de reducción** escogida, el resultado puede cambiar

Modos de evaluación

- Consideremos una llamada a función:

$$f \ e_1 \dots e_k$$

Podemos distinguir **dos modos de evaluación**:

- Evaluación **impaciente** (*eager*)
- Evaluación **perezosa** (*lazy*)

Modos de evaluación

- Evaluación **impaciente** (*call-by-value*): primero evalúa los argumentos y luego usa una ecuación de la definición de la función

```
seisveces 1 → doble (triple 1)
                    → doble (3*1)
                    → doble 3
                    → 3+3
                    → 6
```

Modos de evaluación

- Evaluación **perezosa** (*call-by-name*): sólo evalúa los argumentos si es necesario para aplicar alguna de las ecuaciones que definen la función

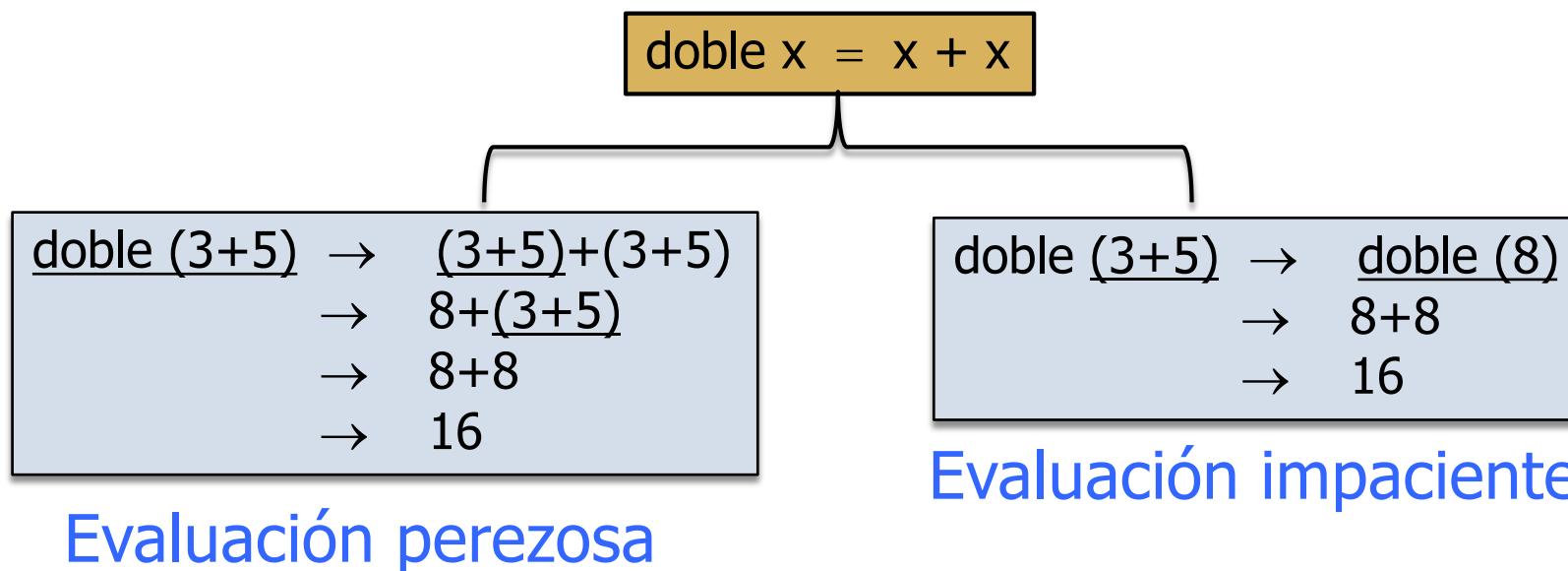
```
seisveces 1 → doble (triple 1)
                  → (triple 1)+(triple 1)
                  → (3*1)+(triple 1)
                  → 3+(triple 1)
                  → 3+(3*1)
                  → 3+3
                  → 6
```

Modos de evaluación

- ¿Qué estrategia de evaluación es más eficiente?

iDepende del programa!

Ejemplo en el que la evaluación impaciente es más eficiente que la perezosa

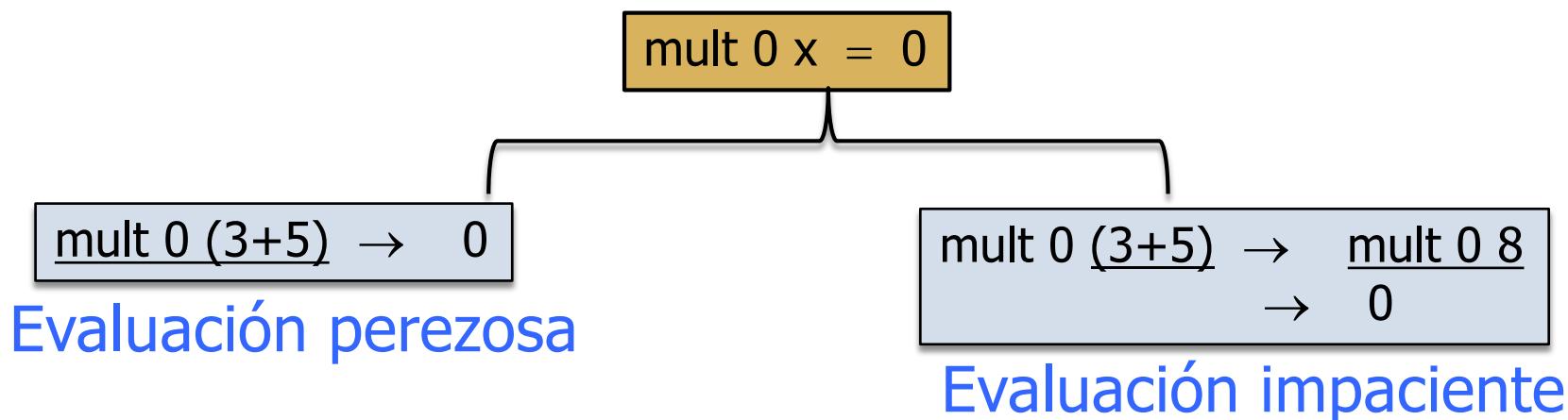


Modos de evaluación

- ¿Qué estrategia de evaluación es más eficiente?

¡Depende del programa!

Ejemplo en el que la evaluación perezosa es más eficiente que la impaciente

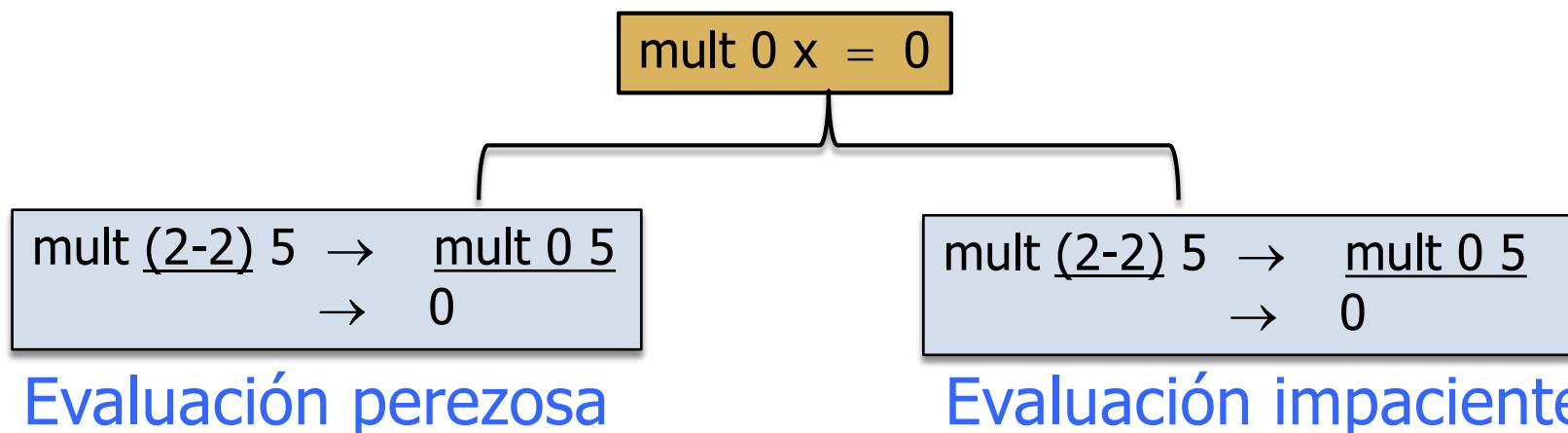


Modos de evaluación

- ¿Qué estrategia de evaluación es más eficiente?

iDepende del programa!

Ejemplo en el que la evaluación perezosa es igual de eficiente que la impaciente



Tipos de evaluación



□ Tipos de evaluación:

Tipos de evaluación

□ Tipos de evaluación:

- Con éxito: termina y obtiene un valor

seisveces 1 →* 6

Tipos de evaluación

□ Tipos de evaluación:

- **Con éxito:** termina y obtiene un valor
- **Con fallo:** termina pero no obtiene un valor

`tail (x:xs) = xs`

La expresión

`tail []`

ya está evaluada pues ya está en **forma normal**, pero no es un valor

Tipos de evaluación

□ Tipos de evaluación:

- ❑ **Con éxito:** termina y obtiene un valor
- ❑ **Con fallo:** termina pero no obtiene un valor
- ❑ **Incompleta:** no termina

```
loop =loop  
mult 0 x = 0
```

Ejemplo de secuencia de evaluación incompleta: mult loop 3 → mult loop 3
→ . . .

Evaluación perezosa

- Con la evaluación perezosa tenemos la oportunidad de **evitar la *no terminación***

loop = loop

mult 0 x = 0

mult 0 loop → 0

Evaluación perezosa

- Con la evaluación perezosa podemos operar con **estructuras de datos infinitas**

`from n = n:from (n+1)`

`sel 0 (x:xs) = x`

`sel n (x:xs) = sel (n-1) xs`

La expresión `from 0` denota una lista infinita que contiene todos los números naturales

Evaluación perezosa

sel 1 (from 0)

La evaluación perezosa es capaz de evaluar expresiones que involucran valores infinitos

Ejercicio

Indique la secuencia de reducción de la expresión:

inorder [2,6,1]

con la evaluación perezosa y con la evaluación impaciente

insertar x [] = [x]

insertar x (y:ys)

| $x \leq y = (x:y:ys)$

| otherwise = y : (insertar x ys)

inorder [] = []

inorder (x:xs) = insertar x (inorder xs)