Unidad III: Evaluación perezosa

3.1. La estrategia de evaluación perezosa

La operación que realizamos en funcional es aplicar funciones, la idea del tema que vamos a tratar a continuación es saber qué se tiene que tener en cuenta para determinar el orden en en que aplicarán las funciones de una expresión.

Primer ejemplo

```
masUno x = x + 1
```

La expresión masUno (2*3) puede ser evaluada de la siguiente forma

```
masUno (2*3)
```

- aplicamos *

```
masUno 6
```

- aplicamos masUno

```
6 + 1
```

- aplicamos +

```
7
```

Alternativamente podemos evaluar la misma expresión pero aplicando las funciones en el orden inverso

```
masUno (2*3)
```

- aplicamos masUno

```
(2*3) + 1
```

- aplicamos *

```
6 + 1
```

- aplicamos +

```
7
```

No importa el orden en que apliquemos las funciones vamos a llegar al mismo resultado final. Esto no solo vale para ejemplos sencillos sino que se cumple siempre en Haskell.

Esta propiedad no se cumple en la mayoría de los "lenguajes imperativos" (Pascal, C, Smalltalk, Java, C#, etc.), veamos un ejemplo en Smalltalk:

Si tenemos la expresión n + (n := 1) y n empieza apuntando a 0.

Si empezamos a evaluar de izquierda a derecha

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

- aplicamos n

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

- aplicamos :=

```
0 + 1
```

- aplicamos +

```
1
```

Si empezamos a evaluar de derecha a izquierda

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

- aplicamos :=

```
n + 1
```

- aplicamos n

```
1 + 1
```

- aplicamos +

```
2
```

Como se puede observar, si evaluamos las expresiones con distintas estrategias obtenemos resultados distintos; esto sucede porque las operaciones involucradas no tienen transparencia referencial en este caso particular debido a la introducción de una asignación destructiva (más sobre esto en la próxima clase teórica).

3.2. Técnicas de programación funcional perezosa

Los beneficios de la evaluación perezosa son:

- El incremento en el rendimiento al evitar cálculos innecesarios, y en tratar condiciones de error al evaluar expresiones compuestas.
- La capacidad de construir estructuras de datos potencialmente infinitas.
- La capacidad de definir estructuras de control como abstracciones, en lugar de operaciones primitivas.

La evaluación perezosa puede también reducir el consumo de memoria de una aplicación, ya que los valores se crean solo cuando se necesitan. Sin embargo, es difícil de combinar con la operaciones típicas de programación imperativa, como el manejo de excepciones o las operaciones de entrada/salida, porque el orden de las operaciones puede quedar indeterminado. Además, la evaluación perezosa puede conducir a fragmentar la memoria.

Lo contrario de la evaluación perezosa sería la evaluación acaparadora, o evaluación estricta, que es el modo de evaluación por defecto en la mayoría de los lenguajes de programación.