Facultad de Ciencias, UNAM
Lenguajes de Programación
Características de la evaluación perezosa y glotona

Rubí Rojas Tania Michelle

07 de diciembre de 2020

Evaluación perezosa	Evaluación glotona
Los argumentos de una función no son evaluados hasta que es estrictamente necesario.	Los argumentos de una función siempre deben ser reducidos a un valor.

Ejemplo: Definimos la función `suma` como

```
(define (suma l)
  (if (empty? l)
      0
      (+ (car l) (suma (cdr l)))))
```

La evaluación de esta función con el parámetro `'(30 26)` utilizándo evaluación perezosa sería de la siguiente forma:

```
(suma '(30 26)) = (+ (car '(30 26)) (suma (cdr '(30 26))))
                = (+ (car '(30 26)) (suma '(26)))
                = (+ (car '(30 26))
                     (+ (car '(26)) (suma (cdr '(26)))))
                = (+ (car '(30 26))
                     (+ (car '(26)) (suma '())))
                = (+ (car '(30 26))
                     (+ (car '(26)) 0))
                = (+ (car '(30 26))
                        (+ 26 0))
                = (+ (car '(30 26)) 26)
                = (+ 30 26)
                = 56
```

Mientras que la evaluación de la función `suma` con el parámetro `'(30 26)` utilizándo evaluación perezosa sería de la siguiente forma:

```
(suma '(30 26)) = (+ (car '(30 26)) (suma (cdr '(30 26))))
                = (+ 30 (suma '(26)))
                = (+ 30 (+ (car '(26)) (suma (cdr '(26)))))
                = (+ 30 (+ 26 (suma '())))
                = (+ 30 (+ 26 0))
                = (+ 30 26)
                = 56
```

Evaluación perezosa	Evaluación glotona
Es más eficiente cuando hay variables que nunca usamos	Es menos eficiente cuando hay variables que nunca usamos.

Ejemplo: Definimos las funciones `fact` y `goo` como sigue

```
(define (fact n)
  (if (zero? n)
      1
      (* n (fact (sub1 n)))))

(define (goo n)
  (let ([x (fact 10000)])
    (let ([y (fact 20000)])
      (let ([z (fact 30000)])
        n))))
```

Si llamamos a `(goo 100000)`, utilizándo evaluación perezosa, como nunca utilizamos ninguna aparición de `fact`, entonces en nuestra pila de ejecución no tendremos la evaluación de cada una de las expresiones `fact`. Por otro lado, utilizándo evaluación glotona tendremos que evaluar cada una de las apariciones de la función `fact`, lo que hará que esta evaluación sea más lenta y ocupe más espacio en memoria, en comparación con la evaluación perezosa, la cual será más rápida y ocupará menos espacio en memoria.

Evaluación perezosa	Evaluación glotona
Puede que no detecte errores en semántica (en la expresión no evaluada)	Siempre detecta errores de semántica

Ejemplo: Definimos la función `foo` como sigue

```
(define (foo n)
  (let ([x (/ 7 0)])
    n))
```

Notemos que la expresión `(/ 7 0)` ocasiona un error, pues no es posible dividir entre cero. Utilizándo evaluación perezosa, como la expresión `(/ 7 0)` no es usada, entonces esta asignación es ignorada (se realiza la asignación, pero no la aplicación de función), por lo que simplemente regresaremos el valor que le pasamos como parámetro a la función `foo`. Por otro lado, utilizándo evaluación glotona, la n ni siquiera llega a evaluarse, pues detectaría el error al intentar dividir entre cero.

Evaluación perezosa	Evaluación glotona
En promedio, tiene peor uso de espacio	En promedio, tiene mejor uso de espacio

Ejemplo: Definimos la función `hoo` como sigue:

```
(define (hoo l)
  (if (empty? l)
      '()
      (cons (+ (car l) 1) (hoo (cdr l)))))
```

La evaluación de esta función con el argumento `'(5)` utilizándo evaluación perezosa sería de la forma:

```
(hoo '(5)) = (cons (+ (car '(5)) 1) (hoo (cdr '(5))))
           = (cons (+ (car '(5)) 1) (hoo '()))
           = (cons (+ (car '(5)) 1) '())
           = (cons (+ 5 1) '())
           = (cons 6 '())
           = '(6)
```

Mientras que la evaluación de la función con el mismo argumento utilizándo evaluación glotona sería de la forma:

```
(hoo '(5)) = (cons (+ (car '(5)) 1) (hoo (cdr '(5))))
           = (cons (+ 5 1) (hoo '()))
           = (cons 6 '())
           = '(6)
```

De esta forma, podemos notar que usando evaluación perezosa estamos cargando en memoria con las expresiones que no aún no evaluamos, mientras que la evaluación glotona no lo hace (pues evalúa las expresiones terminales inmediatamente). Por esta razón, la evaluación glotona tiene menor complejidad en espacio que la evaluación perezosa.

Evaluación perezosa
Es posible definir estructuras de datos infinitas.

Ejemplo: Como Haskell utiliza evaluación perezosa, entonces soporta listas infinitas. Si una función no tiene casos base, entonces podemos seguir calculando algo infinitamente (o bien, produciéndo una estructura infinita). Sin embargo, lo bueno de estas listas es que podemos *cortarlas* por donde queramos. La función `repeat` toma un elemento y regresa una lista infinita que simplemente tiene ese elemento. Una implementación recursiva podría ser como sigue:

```
repeat :: a -> [a]
repeat x = x : repeat x
```

Si llamamos a `repeat 5` entonces obtendríamos una lista que tiene un 5 en su cabeza y luego tendría una lista infinita de cincos en su cola, es decir, una lista de la forma `[5 5 5 5 ...]` que nunca terminaría su evaluación. Pero, si usamos la función `take` entonces podríamos cortar la lista. De esta forma, al llamar `take 3 (repeat 5)` obtenemos una lista con tres cincos, es decir, una lista de la forma `[5 5 5]`.

Evaluación glotona
Se pierde el contexto o referencia del origen de las variables.

Ejemplo: Recórdando el ejemplo número 1 (usándo evaluación glotona) podemos notar cómo vamos perdiéndo las referencias acerca de dónde obtenemos cada valor.

```
(suma '(30 26)) = (+ (car '(30 26)) (suma (cdr '(30 26))))
                = (+ 30 (suma '(26)))
                = (+ 30 (+ (car '(26)) (suma (cdr '(26)))))
                = (+ 30 (+ 26 (suma '())))
                = (+ 30 (+ 26 0))
                = (+ 30 26)
                = 56
```

Por ejemplo, al final, el número 30 no sabémos de dónde lo obtenemos, pues hemos perdido su contexto de origen.