Patrones

Cecilia Manzino

16 de mayo de 2023

Buscando similitudes

Si observamos la práctica 5.1 encontraremos muchas similitudes entre las funciones de una misma sección.

- Sección 2: las funciones reciben una lista y se quedan con los elementos que cumplen determinada propiedad.
- Sección 3: las funciones reciben una lista y aplican una función a cada elemento de la misma.
- Sección 4: Dada una lista aplican una operación sobre sus elementos.

Patrones

- Para evitar la repetición de código veremos un mecanismo de abstracción que consiste en buscar patrones comunes entre los programas.
- Ventajas de usar abstracciones:
  - Simplifica la escritura de los programas.
  - Se logran programas más legibles.
  - Previenen errores, ya que nos concentramos en lo esencial.

Patrón filter

```
; positivos : List (Number) -> List (Number)
 (define   (positivos l)
           (cond [(empty? l) '()]
                 [(positive? (first l))
                             (cons (first l) (positivos (rest l)))]
                 [else (positivos (rest l))]))

; pares : List (Number) -> List (Number)
 (define   (paress l)
           (cond [(empty? l) '()]
                 [(pares? (first l))
                          (cons (first l) (pares (rest l)))]
                 [else (pares (rest l))]))
```

Patrón filter

```
; positivos : List (Number) -> List (Number)
 (define   (positivos l)
           (cond [(empty? l) '()]
                 [(positive? (first l))
                             (cons (first l) (positivos (rest l)))]
                 [else (positivos (rest l))]))

; pares : List (Number) -> List (Number)
 (define   (pares l)
           (cond [(empty? l) '()]
                 [(pares? (first l))
                           (cons (first l) (pares (rest l)))]
                 [else (pares (rest l))]))
```

Diseño de filter

```
; filter : (X -> Boolean) List (X) -> List (X)
; Dado un predicado p y una lista l construye una
; lista con los elementos de l que satisfacen p.

(check-expect (filter even? (list 1 2 3 4)) (list 2 4))
(check-expect (filter odd? '()) '())

(define   (filter p l)
          (cond [(empty? l) '()]
                [(p (first l)) (cons (first l) (filter p (rest l)))]
                [else (filter p (rest l))]))
```

Usando de filter

```
; pares : List (Number) -> List (Number)
(define (pares l) (filter even? l))

; positivos : List (Number) -> List (Number)
(define (positivos l) (filter positive? l))
```

Patrón map

Dada una función f, la función map transforma una lista:

$$[a_0, \ldots, a_n]$$

en

$$[f(a_0), \ldots, f(a_n)]$$

Diseño de map

```
; map : (X -> Y) List (X) -> List (Y)
; dada una función f que transforma elementos de X
; en elementos de Y y una lista de elementos en X
; devuelve el resultado de aplicar f sobre
; cada elemento de la lista.

(check-expect (map sqr (list 1 2 3 4)) (list 1 4 9 16))
(check-expect (map sqr '()) '())

(define   (map f l)
          (cond [(empty? l) '()]
                [else (cons (f (first l)) (map f (rest l)))]))
```

Usando map

```
; cuadrados : List (Number) -> List (Number)
; dada una lista de números devuelve
; una lista con los cuadrados de los números.

(define (cuadrados l) (map sqr l))


; raíces : List (Number) -> List (Number)
; dada una lista de números devuelve
; una lista con las raíces de los mismo.

(define (raices l) (map sqrt l))
```

Patrón foldr

Dado un operador $\oplus$ y un valor c la función foldr convierte la lista:

$$[a_0, \ldots, a_n]$$

en

$$a_0 \oplus (a_1 \oplus \cdots \oplus (a_n \oplus c))$$

Definición de foldr

```
;foldr : (X Y -> Y) Y List (X) -> Y

(define   (foldr f c l)
          (cond [(empty? l) c]
                [else (f (first l) (foldr f c (rest l)))]))
```

Usando de foldr

```
; suma: List (Number) -> Number
; dada una lista de números devuelve la suma de los mismos.

(define (suma l) (foldr + 0 l))


; prod: List (Number) -> Number
; dada una lista de números devuelve el producto de los
mismos.

(define (prod l) (foldr * 1 l))
```