Procedimientos y recursión

José A. Alonso y María J. Hidalgo

Ciencias de la Computación e Inteligencia Artificial

Universidad de Sevilla

Expresiones lambda

- Ejemplos:

```
> (lambda (x y) (/ (+ x y) 2))
#<CLOSURE (x y) (/ (+ x y) 2)>
> ((lambda (x y) (/ (+ x y) 2)) 3 7)
5
> (define a 3)
#<unspecified>
> ((lambda (x y) (/ (+ x y) 2)) a 7)
5
```

- Comentarios:
  - Sintaxis: `(lambda <parámetros> <cuerpo>)`
  - Evaluación con expresiones lambda

Definición de procedimientos

- Ejemplo

```
> (define  media2
    (lambda (x y)
      (/ (+ x y) 2)))
#<unspecified>
> (media2 3 7)
5
> (define x 9)
#<unspecified>
> (media2 3 7)
5
> x
9
```

- Comentarios:
  - **Def. de procedimientos:** `(define <nombre> <expresión-lambda>)`
  - **Evaluación:** `(<nombre> <expresión-1> ... <expresión-n>)`
  - **Entorno local y global**

Descomposición de procedimientos

```
;;; (agrupa '(a b c d))  =>  ((a b) (c d))
(define agrupa
  (lambda (l)
    (haz-lista-de-2 (primer-grupo l)
                    (segundo-grupo l))))

;;; (primer-grupo '(a b c d))  =>  (a b)
(define primer-grupo
  (lambda (l)
    (haz-lista-de-2  (car l) (cadr l))))

;;; (haz-lista-de-2 'a 'b)  =>  (a b)
(define haz-lista-de-2
  (lambda (x y)
    (cons x (haz-lista-de-1 y))))
```

Descomposición de procedimientos

```
;;; (haz-lista-de-1 'b)  =>  (b)
(define haz-lista-de-1
  (lambda (x)
    (cons x ())))

;;; (segundo-grupo '(a b c d))  =>  (c d)
(define segundo-grupo
  (lambda (l)
    (cddr l)))
```

Construcción de listas con `list` y `append`

- Ejemplos:

```
(list 'a 'b 'c 'd)          =>  (a b c d)
(list)                      =>  ()
(list '(a b) '(c d))        =>  ((a b) (c d))
(append '(a b) '(c d))      =>  (a b c d)
(cons '(a b) '(c d))        =>  ((a b) c d)
(append '(a (b)) '((c)))    =>  (a (b) (c))
(append '(a) '(b) '(c d))   =>  (a b c d)
(append '(a) ())            =>  (a)
(append)                    =>  ()
```

- Comentarios:
  - Sintaxis: `(list <expresión-1> ... <expresión-n>)`
  - Sintaxis: `(append <expresión-1> ... <expresión-n>)`

Expresiones condicionales: `if`

- Ejemplos:

```
(if #f 2 3)  =>  3
(if 1 2 3)   =>  2
(if 'a 2 3)  =>  2
(if () 2 3)  =>  2
(if #t 2)    =>  2
(if #f 2)    =>  #<unspecified>
(if (if #f 2) 3 4)  =>  3
```

- Comentarios:
  - Condicional: `(if <test> <consecuencia> [<alternativa>])`
  - Verdadero y falso

Expresiones condicionales

- Ejemplo:

```
;;; (semaforo-1 'rojo)   =>  parar
;;; (semaforo-1 'ambar)  =>  continuar
;;; (semaforo-1 'verde)  =>  continuar
(define semaforo-1
  (lambda (color)
    (if (eq? color 'rojo)
        'parar
        'continuar)))
```

Expresiones condicionales

```
;;; (semaforo-2 'rojo)   =>  parar
;;; (semaforo-2 'ambar)  =>  continuar-con-precaucion
;;; (semaforo-2 'verde)  =>  continuar
;;; (semaforo-2 'azul)   =>  color-imposible
(define semaforo-2
  (lambda (color)
    (if (eq? color 'rojo)
        'parar
        (if (eq? color 'ambar)
            'continuar-con-precaucion
            (if (eq? color 'verde)
                'continuar
                'color-imposible)))))
```

Expresiones condicionales

- Ejemplo:

```
;;; (semaforo-2 'rojo)   =>  parar
;;; (semaforo-2 'ambar)  =>  continuar-con-precaucion
;;; (semaforo-2 'verde)  =>  continuar
;;; (semaforo-2 'azul)   =>  color-imposible
(define semaforo-2
  (lambda (color)
    (cond
     ((eq? color 'rojo)  'parar)
     ((eq? color 'ambar) 'continuar-con-precaucion)
     ((eq? color 'verde) 'continuar)
     (else 'color-imposible))))
```

- Sintaxis:

```
(cond
 (<condicion-1> <consecuencia-1>)
 (<condicion-2> <consecuencia-2>)
 ..............................
 (<condicion-n> <consecuencia-n>)
 (else          <alternativa>))
```

Operadores lógicos

- Ejemplos:

```
(not #f)                                    =>  #t
(not #t)                                    =>  #f
(not (list? '(1 2 3)))                      =>  #f
(or (eq? 1 2) (eq? 'a 'A))                  =>  #t
(or (eq? 1 2) (eq? 'a '(A)))                =>  #f
(or (eq? 1 2) (eq? 'a '(A)) (symbol? 'a1))  =>  #t
(and (eq? 1 2) (eq? 'a 'A))                 =>  #f
(and (eq? 1 1) (eq? 'a 'A))                 =>  #t
(and (number? 3) (boolean? (symbol? 23)))   =>  #t
(and (= (+ 2 2) 4) 6)                       =>  6
(or 6 (= (+ 2 2) 4))                        =>  6
(or)                                        =>  #f
(and)                                       =>  #t
```

Operadores lógicos

```
(define lista-simple?
  (lambda (l)
    (and (pair? l)
         (null? (cdr l)))))                    =>  #<unspecified>
(lista-simple? '(a b c))                       =>  #f
(lista-simple? '(a))                           =>  #t
(lista-simple? ())                             =>  #f
```

- Comentarios:
  - **Negación**: `(not <expresión>)`
  - **Conjunción**: `(and <expresión-1> ... <expresión-n>)`
  - **Disyunción**: `(or <expresión-1> ... <expresión-n>)`

Recursión numérica

- Definición de factorial

```
0! = 1                      (caso base)
n! = n * (n - 1)!           (relación de recurrencia)

3! = 3 * 2! =
   = 3 * 2 * 1! =
   = 3 * 2 * 1 * 0! =
   = 3 * 2 * 1 * 1 =
   = 6
```

- Procedimiento factorial

```
(define fact
  (lambda (n)
    (if (= n 0)
        1
        (* n (fact (- n 1))))))
```

Recursión numérica

- Evaluación con traza

```
> (fact 3)
6
> (trace fact)
#<unspecified>
> (fact 3)
"CALLED" fact 3
 "CALLED" fact 2
  "CALLED" fact 1
   "CALLED" fact 0
   "RETURNED" fact 1
  "RETURNED" fact 1
 "RETURNED" fact 2
"RETURNED" fact 6
6
> (untrace fact)
#<unspecified>
> (fact 3)
6
```

Recursión sobre listas: `ultimo`

- Ejemplos:

```
(ultimo '(a b c))           =>  c
(ultimo '((1 2) 3 (4 5)))   =>  (4 5)
(ultimo '(a))               =>  a
```

- Esquema:

```
(a) ---> a
    car

(a b c) ---> (b c) ------> c
        cdr        ultimo
```

Recursión sobre listas: `ultimo`

- Procedimiento:

```
(define ultimo
  (lambda (l)
    (if (null? (cdr l))
        (car l)
        (ultimo (cdr l)))))
```

- Cálculo con traza:

```
> (trace ultimo)
#<unspecified>
> (ultimo ´(a b c))
"CALLED" ultimo (a b c)
 "CALLED" ultimo (b c)
  "CALLED" ultimo (c)
  "RETURNED" ultimo c
 "RETURNED" ultimo c
"RETURNED" ultimo c
c
> (untrace)
(ultimo)
```

Recursión sobre listas: pertenece?

- Ejemplos:

```
(pertenece? 2 '(1 3 2 5))    =>  #t
(pertenece? 2 '(1 (3 2) 5))  =>  #f
(pertenece? 'a '(b c d))     =>  #f
```

- Esquema:

```
Bases:     (pertenece? 'a ())        =>  #f
           (pertenece? 'a '(a b c))  =>  #t
Reducción: (pertenece? 'b '(a b c))  =>  (pertenece? 'b '(b c))  =>  #t
           (pertenece? 'd '(a b c))  =>  (pertenece? 'd '(b c))  =>  #f
```

Recursión sobre listas: pertenece?

- Procedimiento:

```
(define pertenece?
  (lambda (x l)
    (cond
     ((null? l) #f)                        ; Base 1
     ((equal? x (car l)) #t)               ; Base 2
     (else (pertenece? x (cdr l))))))      ; Reducción
```

- Definición con conectivas:

```
(define pertenece?
  (lambda (x l)
    (and (not (null? l))
         (or (equal? x (car l))
             (pertenece? x (cdr l))))))
```

Predicados de pertenencia

- Ejemplos:

```
(memq 'a '(a b c))              =>  (a b c)
(memq 'b '(a b c))              =>  (b c)
(memq 'a '(b c d))              =>  #f
(memq 101 '(100 101 102))       =>  (101 102)
(memq 1.1 '(100 1.1 102))       =>  #f
(memv 1.1 '(100 1.1 102))       =>  (1.1 102)
(memq   (list 'a) '(b (a) c))   =>  #f
(member (list 'a) '(b (a) c))   =>  ((a) c)
```

- Comentarios:
  - **Pertenencia con** `eq?`: `(memq <expresión> <lista>)`
  - **Pertenencia con** `eqv?`: `(memv <expresión> <lista>)`
  - **Pertenencia con** `equal?`: `(member <expresión> <lista>)`

Recursión sobre listas: quita-primera

- Ejemplos:

```
(quita-primera 'a '(c d a b a e))  =>  (c d b a e)
(quita-primera 'a '(c (a b) a d))  =>  (c (a b) d)
(quita-primera 'a ())              =>  ()
(quita-primera 'a '(1 2 3))        =>  (1 2 3)
```

- Esquema:

```
Bases:     (quita-primera 'a ())                      =>  ()
           (quita-primera 'a '(a b c a))              =>  (b c a)
Reduccion: (quita-primera 'c '(a b c a))              =>
           => (cons 'a (quita-primera 'c '(b c a)))   =>  (a b a)
```

Recursión sobre listas: quita-primera

- Procedimiento:

```
(define quita-primera
  (lambda (x l)
    (cond
     ((null? l) ())
     ((equal? (car l) x) (cdr l))
     (else (cons (car l) (quita-primera x (cdr l)))))))
```

Recursión sobre listas: `intercambia`

- Ejemplos:

```
(intercambia 'a 'b '(c d a b a c f))      =>  (c d b a b c f)
(intercambia 'a 'b '(c d b a b (a c) f))  =>  (c d b a b (a c) f)
```

- Esquema:

```
1 (intercambia 'a 'b ())
    =>  ()
2 (intercambia 'a 'b '(a c b a))  =>
    => (cons 'b (intercambia 'a 'b '(c b a)))
    => (b c a b)
3 (intercambia 'a 'b '(b c b a))  =>
    => (cons 'a (intercambia 'a 'b '(c b a)))
    => (a c a b)
4 (intercambia 'a 'b '(d c b a))  =>
    => (cons 'd (intercambia 'a 'b '(c b a)))
    => (d c a b)
```

Recursión sobre listas: `intercambia`

- Procedimiento:

```
(define intercambia
  (lambda (x y l)
    (cond
     ((null? l) ())                                          ; 1
     ((equal? (car l) x)                                     ; 2
      (cons y (intercambia x y (cdr l))))
     ((equal? (car l) y)                                     ; 3
      (cons x (intercambia x y (cdr l))))
     (else (cons (car l) (intercambia x y (cdr l)))))))      ; 4
```

Bibliografía

- [Abelson–96]
  Cap. 1: "Building abstractions with procedures".
- [Springer–94]
  Cap. 2: "Procedures and recursion".