

Para hacer la traza subrayamos la sub-expresión que se evalúa/reduce en cada paso. La evaluación finaliza cuando la expresión es irreducible/canónica.

i. (equal? '((b a) . (a b)) (append '(b) '(a . ( ))) '(a b)))

(equal? '((b a) a b) (append '(b) '(a . ( )) '(a b)))

(equal? '((b a) a b) (append '(b) '(a) '(a b)))

(equal? '((b a) a b) '(b a a b))

#f

ii. (equal? '((b a) . (a b))  
(append '((b a) a b) '()))

(equal? '((b a) . (a b))  
'((b a) a b))

(equal? '((b a) a b)) '((b a) a b))

#t

iii. (map (lambda(x y z) (if (> y x) z y))  
'(40 30 20) '(28 25 21) '((a) a (NO (a))))  
( ((lambda...) 40 28 '(a)) ((lambda...) 30 25 'a) ((lambda...) 20 21 '(NO (a))))  
( (if (> 28 40) '(a) 28)) (if (> 25 30) 'a 25)) (if (> 21 20) '(NO (a)) 21 )  
( 28 25 '(NO (a)))

iv. (map (lambda(x y z) (z x y))  
(list list cons) (list list cons) (list cons list))

(map (lambda(x y z) (z x y))  
(list cons) (list cons) (cons list))

( ((lambda(...)) list list cons)  
((lambda(...)) cons cons list) )

( (cons list list) (list cons cons) )

( (list . list) (cons cons) )

v. ((lambda(x) ((cdr x) (car x))) (cons 5 null?) )

((lambda(x) ((cdr x) (car x))) '(5 . null?) )

((cdr '(5 . null?)) (car '(5 . null?)) )

(null? 5)

#f

vi. (let\* ((x 1) (y 2))

(let ((y 8) (x (+ 3 y)) (z (+ x y))) (list x y z)))

;x=1 y=2

(let ((y 8) (x (+ 3 2)) (z (+ 1 2))) (list x y z)))

;x=5 y=8 z=3

(list 5 8 3)

(5 8 3)

vii. (take-while positive? `(-1 -2 3 4))

; recoge elementos mientras sean positivos

()

viii. (let ((x list) (y 2))

(let\* ((y 8) (z (+ y 5))) (x y z)))

(let ((x list) (y 2)) ; x=list y=2

(let\* ((y 8) (z (+ y 5))) (x y z)))

(let ((x list) (y 2)) ; x=list y=2

(let\* ((y 8) (z (+ y 5))) ; y=8 z =13

(x y z)))

(list 8 13)

(8 13)

ix. ((lambda x (cons x '(a b))) 2 4 )

(cons '(2 4) '(a b))

((2 4) a b)

x. ((curry apply -) '(-1 3))

; Lambda(L) (apply - L)

((lambda(x) (apply - x)) '(-1 3))

(apply - '(-1 3))

(- -1 3)