

Taller 2: Semántica de lenguajes de programación

Fundamentos de interpretación y compilación de lenguajes de programación

Jesús Ediber Arenas - 2266066
Anderson Gomez Garcia - 2266242
Kevin Alexis Lorza - 2266098
Juan David Pérez - 2266289

Carlos Andres Delgado S.

Universidad del valle sede Tuluá
30/11/2024



En este proyecto se nos pide agregar expresiones de listas en el lenguaje con la siguiente gramática

```
<expresion> := cons ( <expresion> <expresion> )  
              list-exp(valor lista)  
              := empty  
              list-empty-exp
```

Entonces los agregamos a la especificación de gramática:

```
(expresion ("list" "(" (separated-list expression ",") ")") list-exp)  
(expresion ("cons" "(" expression expression ")") cons-exp)  
(expresion ("empty") list-empty-exp)  
(expresion ("length" "(" expression ")") length-exp)  
(expresion ("first" "(" expression ")") first-exp)  
(expresion ("rest" "(" expression ")") rest-exp)  
(expresion ("nth" "(" expression "," expression ")") nth-exp)
```

Con la estructura de (palabraReservada (“ cuerpoExpresion “) expresion-evaluar) procedemos con la creación de los métodos para evaluar cada expresión requerida:

```
(list-exp (elems)  
  (map (lambda (e) (evaluar-expresion e amb)) elems))  
  
(cons-exp (e1 e2)  
  (let* ([v1 (evaluar-expresion e1 amb)] ;; Evalúa el primer argumento  
         [v2 (evaluar-expresion e2 amb)] ;; Evalúa el segundo argumento  
        (if (list? v2) ;; Verifica que el segundo argumento sea una lista  
            (cons v1 v2) ;; Crea una nueva lista  
            (error "Error: el segundo argumento de cons no es una lista" v2))))  
  
(list-empty-exp () '())  
(length-exp (e)  
  (length (evaluar-expresion e amb)))  
(first-exp (e)  
  (car (evaluar-expresion e amb)))  
(rest-exp (e)  
  (cdr (evaluar-expresion e amb)))  
(nth-exp (e n)  
  (list-ref (evaluar-expresion e amb)  
    (evaluar-expresion n amb)))
```

luego ejecutamos algunas pruebas para evaluar la eficacia del código:

```
-->empty
()
-->cons(999 empty)
(999)
-->first(cons(3 cons(4 empty)))
3
-->length(cons(7 cons(5 empty)))
2
-->cons (let x = 7 in x cons (9 empty ))
(7 9)
-->cons (123 654)
Error: el segundo argumento de cons no es una lista 654
-->length(empty)
0
-->+(1,2)
3
-->list(1,2,3)
(1 2 3)
```

Ahora procedemos con la implementación de “cond”, lo añadimos a la especificación-gramatical =

```
(expresion ("cond" (arbno expresion "==>" expresion ) "else" "==>" expresion "end") cond-exp)
(expresion (primitiva "(" (separated-list expresion ",") ")") prim-exp)
```

Luego, creamos su método para evaluar la expresión:

```
(cond-exp (condiciones cuerpo-condiciones else-exp)
  (let loop
    ([condiciones condiciones]
     [cuerpo-condiciones cuerpo-condiciones]) ;; Cuerpo de las condiciones
    (cond
      [(null? condiciones) (evaluar-expresion else-exp amb)] ;; No se encontró ninguna condición verdadera
      [else
       (let ([valor-condicion (evaluar-expresion (car condiciones) amb)]) ;; Evaluar la condición
         (if (not(= valor-condicion 0)) ;; Verificar si la condición es verdadera
             (evaluar-expresion (car cuerpo-condiciones) amb) ;; Evaluar el cuerpo de la condición
             (loop (cdr condiciones) (cdr cuerpo-condiciones))) ;; Continuar con las siguientes condiciones
         ])))]
```

Por último, ejecutamos algunas pruebas para evaluar la funcionalidad del código:

```
-->cond -(9,1) ==> 1 else ==> 2 end
1
-->cond *(7,0) ==> 1 else ==> 2 end
2
```

```
-->cond +(0,1) ==> 1 else ==> 2 end
1
-->cond /(15,5) ==> 1 else ==> 2 end
1
```

Las pruebas se realizan mediante racketunit y scan&parse leer los strings como código:

```
#lang racket

(require rackunit "interpretador.rkt")

(define test-cond-list
  (test-suite "Test de cond"
    (test-case
      "Test restas"
      (check-equal? (evaluar-programa
        (scan&parse "cond
          -(1,2) ==> 1
          -(2,1) ==> 2
          -(3,3) ==> 3
          else ==> 4
          end"))
        2)))
```

Se ejecutan todas las pruebas:

```
PS C:\Users\Ander\Desktop\Taller2_FLP> racket c:/Users/Ander/Desktop/Taller2_FLP/flp_taller2/pruebas.rkt
-->
```