TALLER #2

JUAN CAMILO GUTIERREZ VIVEROS - 2159874
CARLOS ALBERTO PADILLA MESA - 2059962
ANDRÉS FELIPE ROJAS - 2160328

DOCENTE CARLOS ANDRÉS DELGADO SAAVEDRA

FUNDAMENTOS DE INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN

UNIVERSIDAD DEL VALLE
TULUÁ, VALLE DEL CAUCA
30 DE NOVIEMBRE DE 2024

Modificación a la gramatica

En el Interpretador se incluyen las siguientes expresiones de listas:
“(expresion ("empty") list-empty-exp)
   (expresion ("cons" "(" expresion expresion ")") list-exp)
   (expresion ("length" "(" expresion ")") list-length-exp)
   (expresion ("first" "(" expresion ")") list-first-exp)
   (expresion ("rest" "(" expresion ")") list-rest-exp)
   (expresion ("nth" "(" expresion expresion ")") list-nth-exp)
  ))”
Siendo “empty” una expresión que representa una lista vacia, luego sigue “cons” que se encarga de construir una lista de la forma (a . b). Luego encontramos la gramatica de la expresión “length” que nos retorna el tamales de una lista l, luego la gramatica de “first” retornarno el primer elemento de una lista y de una misma forma “rest” nos retorna el resto y por último vemos la expresión “nth” que recibe un número n y una lista n y nos retorna el en-ésimo elemento de la lista.

Concepto de cond

El cond en Racket es una estructura condicional que permite evaluar múltiples expresiones basadas en condiciones. La sintaxis básica de cond es la siguiente:

```
(cond
  [condición1 expresión1]
  [condición2 expresión2]
  ...
  [else expresión_else])
```

Cada condición es evaluada en orden, y si una es verdadera, se ejecuta la expresión asociada a esa condición. Si ninguna de las condiciones es verdadera, se ejecuta la expresión en la cláusula else, si esta está presente.

Estructura del Condicional cond

El primer paso fue establecer una estructura básica para evaluar la expresión cond en Racket. La forma más simple de evaluarlo es que cada "condición" y su correspondiente "expresión" sean listas que se evaluarán de manera secuencial.

Implementación Básica de Condicionales:

```
(define (evaluar-cond-exp cond-expresion x)
  (cond
    [(< x 10) (+ x 1)] ; Primera condición: Si x < 10, retorna x + 1
    [(> x 10) (* x 2)] ; Segunda condición: Si x > 10, retorna x * 2
```

```
   [else (- x 1)] ; Si no se cumple ninguna condición, retorna x - 1
  ))
```

La evaluación de cond aquí funciona bajo las siguientes reglas:

Si (< x 10) es verdadero, se ejecuta (+ x 1).

Si (< x 10) es falso, pero (> x 10) es verdadero, se ejecuta (* x 2).

Si ninguna de las condiciones previas es verdadera, se ejecuta (- x 1) gracias al else.

Expresiones Anidadas y Múltiples Condiciones

A medida que la lógica de las pruebas se expandía, se necesitaba manejar múltiples condiciones y expresiones anidadas dentro del cond. Esto permitía que las condiciones pudieran estar dentro de otras estructuras cond, lo que facilita la creación de flujos de control más complejos.

Ejemplo con Condiciones Anidadas:

```
(define (evaluar-cond-exp cond-expresion x)
  (cond
    [(< x 10) (+ x 1)] ; Si x es menor que 10, evalúa x + 1
    [(> x 10) (* x 2)] ; Si x es mayor que 10, evalúa x * 2
    [else (cond ; Si no es ni menor ni mayor que 10, entra en un cond anidado
            [(= x 0) 5] ; Si x es igual a 0, retorna 5
            [else 9])])) ; En cualquier otro caso, retorna 9
```

Este ejemplo muestra cómo los condicionales se pueden anidar dentro de cond, lo que permite tomar decisiones más complejas dependiendo de los valores de entrada.

Estructura para Manejar Condicionales Completos

Para soportar las pruebas más complejas, se estructuraron varias expresiones dentro de un conjunto de pruebas con múltiples condiciones, el caso else, y expresiones anidadas.

Pruebas con Condiciones Complejas:

Aquí se crearon pruebas que evalúan condiciones verdaderas, falsas, múltiples condiciones y el uso de else.

Pruebas

Para asegurarnos de que nuestro código funciona correctamente y cubre todos los casos posibles, creamos pruebas unitarias con racketunit. Estas pruebas incluyen varios escenarios que deben ser cubiertos por la estructura de cond.

Casos de Prueba:

1. Condición Simple Verdadera:

Verifica si la expresión dentro de cond se evalúa correctamente cuando la primera condición es verdadera.

2. Condición Simple Falsa con else:

Verifica el comportamiento cuando la primera condición es falsa y se evalúa el else.

3. Múltiples Condiciones, Primera Verdadera:

Prueba un cond con múltiples condiciones donde la primera condición es verdadera.

4. Múltiples Condiciones, Segunda Verdadera:

Evalúa un cond con varias condiciones, pero en este caso, la segunda condición es la que es verdadera.

5. Múltiples Condiciones Ninguna Verdadera con else:

Verifica cómo el sistema maneja la evaluación de múltiples condiciones donde ninguna de ellas es verdadera, y se debe ejecutar la expresión del else.

6. Condicional Anidado:

Verifica que las expresiones cond anidadas se evalúen correctamente.

7. Condicional Vacío con else:

Verifica que el else sea ejecutado cuando no hay ninguna condición y solo existe el else.

Pruebas Específicas:

```
(check-equal? (evaluar-cond-exp '(((< x 10) (+ x 1))) '(- x 1) '((x . 5))) 6)
(check-equal? (evaluar-cond-exp '(((< x 10) (+ x 1))) '(- x 1) '((x . 15))) 14)
(check-equal? (evaluar-cond-exp '(((< x 10) (+ x 1)) ((> x 10) (* x 2))) '(- x 1) '((x . 5))) 6)
(check-equal? (evaluar-cond-exp '(((< x 10) (+ x 1)) ((> x 10) (* x 2))) '(- x 1) '((x . 15))) 30)
(check-equal? (evaluar-cond-exp '(((< x 10) (+ x 1)) ((> x 10) (* x 2))) '(- x 1) '((x . 15))) 14)
(check-equal? (evaluar-cond-exp '(((cond ((< x 10) (+ x 1)) (else 3))) (else 0))) 3)
(check-equal? (evaluar-cond-exp '() '(+ 1 1) '()) 2)
```

6. Errores Encontrados y Solución

Durante el desarrollo, surgieron varios errores relacionados con cómo se trataban las expresiones dentro de cond. El principal problema fue que las expresiones estaban siendo tratadas como listas literales debido al uso incorrecto de las comillas. La solución consistió en asegurarse de que las expresiones pasadas dentro de las condiciones fueran evaluables y no literales.

Conclusión

Se creó una estructura robusta para evaluar expresiones cond en Racket, incluyendo:

1. Condiciones simples.

2. Múltiples condiciones y expresiones anidadas.

3. Manejo del caso else.

También se implementaron pruebas exhaustivas que cubren todos los posibles casos de uso, asegurando que las expresiones dentro del cond se manejen correctamente.

Las pruebas fueron diseñadas para verificar el comportamiento tanto en casos sencillos como complejos, incluyendo condiciones verdaderas y falsas, así como el manejo adecuado del else.