Practica 5 - Interprete RCFAEL

Profesora: Karla Ramírez Pulido Ayudante: Héctor Enrique Gómez Morales

Fecha de inicio: 14 de Octubre de 2015
Fecha de entrega: 4 de Noviembre de 2015

1. Instrucciones

Para esta práctica, requerirás tomar como base el interprete de la practica anterior e implementar las características que se solicitan.

El objetivo de esta práctica es hacer un intérprete del lenguaje `RCFAEL` (Recursive, Conditional, Function, Arithmetic Expression and List) con *cajas*.

Se tendrán dos sintaxis, la primera `RCFAELS` es una sintaxis que se define explícitamente las expresiones `with`, y la sintaxis `RCFAEL`.

En la anterior practica vimos que usar la aplicación de funciones para implementar la funcionalidad del `with`.

- **Recursive** El intérprete de nuestro lenguaje debe poder evaluar correctamente variables con expresiones que hacen autorreferencia en el mismo ambiente por medio de *cajas*.
- **Conditional** El intérprete evaluara correctamente el control de flujo `if` que toma expresiones que se evalúan a booleanos.
- **FAE** El intérprete debe evaluar correctamente expresiones de la práctica pasada como son funciones, aplicaciones y operaciones binarias predefinidas así como su extensión a operaciones unarias predefinidas y la inclusión de valores booleanos.
- **List** El interprete debe evaluar correctamente expresiones para la construcción de listas y sus operaciones unarias correspondientes.

```
<RCFAEL>::= <id>
          | <num>
          | <bool>
          | <MList>
          | {with {{<id> <RCFAEL>}+} <RCFAEL>}
          | {rec {{<id> <RCFAEL>}+} <RCFAEL>}
          | {fun {<id>*} <RCFAEL>}
          | {if <RCFAEL> <RCFAEL> <RCFAEL>}
          | {equal? <RCFAEL> <RCFAEL>}
          | {<op> <RCFAEL>}
          | {<binop> <RCFAEL> <RCFAEL>}
          | {<RCFAEL> <RCFAEL>*}

<id>::= a|...|z|A|...|Z|aa|ab|...|aaa|...
        (Cualquier combinación de caracteres alfanuméricos
         con al menos uno alfabético)

<num>::= ...|-2|-1|0|1|2|...

<bool>::= true | false

<MList>::= {empty}
         | {cons <RCFAEL> <RCFAEL>}
```

```
<op>::= inc
      | dec
      | zero?
      | num?
      | neg
      | bool?
      | first
      | rest
      | empty?
      | list?

<binop>::= +
         | -
         | *
         | /
         | <
         | >
         | <=
         | >=
         | and
         | or
```

Esta práctica debe ser implementada con la variante plai, es decir su archivo con terminación *.rkt* debe tener como primer linea lo siguiente: `#lang plai`.

Todos los ejercicios requieren contar con pruebas mediante el uso de la función `test`.

2. Ejercicios

1. (**1pts**) **Booleanos** Agrega al tipo RCFAEL y RCFAEL-Value la variante booleana. Esto es que en sintaxis concreta puedas reconocer símbolos `true` y `false` y al momento de procesar la lista de símbolos de una expresión que los incluya con *parse*, en el árbol de sintaxis abstracta los manejes por medio de la variante de tipo `bool` (análogo a `num`), adicional a la variante `boolV` para el intérprete.

2. (**1pts**) **Listas** Agrega al tipo RCFAEL y RCFAEL-Value los constructores específicos para la creación y manejo de listas. En un README aparte explica por qué es necesario tener los varientes de tipo de lista para los dos tipos de datos RCFWAEL y RCFWAEL-Value por separado.

3. (**1pts**) **if** Agrega al parser y al intérprete la variante `if`, que recibe tres argumentos que son expresiones RCFAEL. Al interpretarse, si la evaluación del primer argumento es `(boolV #t)`, la evaluación del `if` es de la evaluación de la primera expresión RCFAEL recibida. En caso de que sea `(boolV #f)`, la evaluación del `if` es la evaluación de la segunda expresión RCFAEL recibida.

4. (**1pts**) **equal?** Agrega al parser y al intérprete la variante `equal?`, que recibe dos expresiones RCFAEL. Si las dos expresiones son `numV`, compara si son los mismos números regresando `(boolV #t)` o `(boolV #f)` según sea el caso. Si son expresiones booleanas, sólo si son iguales valores `boolV`. Al implementar las listas, `equal?` aplicado a dos listas debe comparar elemento por elemento si son iguales, en caso de que no lo sean o sean de tamaños distintos, regresar `(boolV #f)`, de lo contrario `(boolV #t)`. Cualquier otro caso de uso de `isequal?` no es válido y debes regresar un mensaje de error que diga "La aplicación de equal? no es adecuada". En tu README explica qué inconvenientes tendría implementar **equal?** como parte de la variante de tipo `binop`.

5. (**1pts**) **op, binop** Terminar de implementar todas las operaciones unarias y binarias indicadas en la definición de la sintaxis de RCFAEL.

6. (**3pts**) **RCFAEL** Implementar los ambientes recursivos por medio de cajas para la variante de tipo *rec*

7. (**2pts**) **Procedurales** Implementar los ambientes recursivos por medio de *procedimientos* para la variante de tipo *rec*