INFORME FINAL - INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN - OBLIQ

Juan Pablo Ospina, Diana Marcela Oviedo, Juan Camilo Saenz, Jean Carlos Rojas
2411023, 2459375, 2259416, 2259305
Ingeniería de Sistemas

Aspectos importantes de la solución:

Ambiente extendido recursivo es uno de los aspectos más importantes de este trabajo para solucionar problemas que tuvieran recursividad como su nombre lo especifica, para darle solución a este problema en El entorno extendido se construye con:

1. **vectores-clausuras**: Este vector almacena las clausuras (procedimientos que tienen un entorno asociado).
2. **letrec:** Crea un entorno recursivo que permite referencias mutuas entre las definiciones (nombres-procedimientos) y sus clausuras.
3. **ambiente:** Es un entorno extendido generado con las clausuras, donde las definiciones pueden acceder tanto a las variables del entorno extendido como del antiguo (old-env).
4. **set:** Si el entorno implementa las variables como mutables (p. ej., en vector-set!), puedes modificarlas directamente.

Otro aspecto importante fue la construcción del **var** ya que esta aunque tenía una gramática similar al let su comportamiento es un poco diferente ya que en este las variables son mutables (se pueden realizar cambios en la variables, ejemplo con set). La llamamos var-expresion y esta recibe: **lista-ids** es una lista de identificadores (nombres de variables), **lista-expresiones** es una lista de expresiones que se evaluarán, y **exp** es una expresión que se evaluará con las variables ya definidas.

El flujo que se planteó para **var-expresion** es el siguiente:

1. Las expresiones de **lista-expresiones** se evalúan en el ambiente actual (con **evaluar-expresion**), y los resultados se almacenan en valores.
2. Se crea un nuevo entorno (a través de **ambiente-extendido**) que contiene las asociaciones entre las variables de **lista-ids** y los **valores** enviados o calculados..

INFORME FINAL - INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN - OBLIQ

Juan Pablo Ospina, Diana Marcela Oviedo, Juan Camilo Saenz, Jean Carlos Rojas
2411023, 2459375, 2259416, 2259305
Ingeniería de Sistemas

Universidad del Valle

3. La expresión **exp** se evalúa dentro de este nuevo ambiente extendido.
4. El resultado final es el valor de la evaluación de exp.

Y como realiza este flujo?

1. **evaluar-expresion** es una función que evalúa una expresión en un ambiente dado.
2. **ambiente-extendido** es una función que genera un nuevo entorno a partir de un entorno existente, pero añadiendo nuevas asociaciones de variable-valor.
3. **Map** permite evaluar las expresiones en paralelo, garantizando que las variables se asignen con los valores correspondientes antes de la evaluación de exp.

```
interpretador> var x = 1, y = var x = 2 in +(x,1) end in begin set x := 3; +(x,y) end end
6
interpretador>
```

Ahora bien, siguiendo los aspectos importantes también tenemos a LET a la cual le dimos el nombre de **let-expresion** y recibe los siguientes parametros **(lista-ids lista-expresiones sub-exp)** nombres de variables, lista de expresiones que se evaluarán y la expresión que se evaluará en el nuevo entorno. El flujo que se planteó para esta es el siguiente

1. **Validación de set**: Se verifica si la expresión **sub-exp** contiene un **set**. Si lo tiene, se lanza un error y se detiene la ejecución.
2. **Evaluación de las expresiones**: Si **sub-exp** es válido, se evalúan todas las expresiones en **lista-expresiones** en el contexto del ambiente actual (**ambi**).
3. **Creación de un nuevo entorno**: Se crea un nuevo **ambiente extendido**, asociando las variables de **lista-ids** a los valores enviados..
4. **Evaluación de la expresión final**: Se evalúa **sub-exp** dentro de este nuevo **entorno extendido**, lo que incluye las nuevas variables del let.

¿Cómo hace esto?

1. **Evaluación de expresiones**: Se evalúan las expresiones en **lista-expresiones** de forma

INFORME FINAL - INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN - OBLIQ

Juan Pablo Ospina, Diana Marcela Oviedo, Juan Camilo Saenz, Jean Carlos Rojas
2411023, 2459375, 2259416, 2259305
Ingeniería de Sistemas

Universidad del Valle

secuencial, en el contexto del ambiente actual.

2. **Verificación de set**: Antes de crear el nuevo entorno, se asegura que **sub-exp** no contenga un **set**, para mantener la inmutabilidad de las variables locales definidas por **let**
3. **Creación del ambiente extendido**: Después de verificar, se crea un nuevo entorno que extiende el ambiente actual y asocia las variables de **lista-ids** a los valores de **lista-expresiones.**
4. **Evaluación en el entorno extendido**: Finalmente, se evalúa **sub-exp** en este nuevo ambiente extendido, donde las variables locales definidas en **let** son accesibles.

```
interpretador> let x = 1, y = let x = 2 in +(x,1) end in +(x,y) end
4
interpretador>
```

Un componente en el lenguajes es el letrec, el cual nos permite tener un comportamiento recursivo en nuestro lenguajes. Esto lo conseguimos mediante el reconocimiento inmediato de los valores, es decir, en la sección de declaración podemos utilizar el mismo valor que estamos definiendo. Para esto se implementó el ambiente recursivo en el cual almacenamos las declaraciones de manera inmediata para que estén listas en el uso recursivo.

```
interpretador> letrec f(x) = if is(x,0) then 0 elseif is(x,1) then
 1 else +(apply f(-(x, 1)), apply f(-(x, 2))) end in apply f(5) en
d
5
interpretador>
```

proc: Aquí cuando un procedimiento se evalúa, se crea una clausura(closure) donde asocia los parámetros con el cuerpo del procedimiento y el ambiente actual. Lo anterior lo reflejamos como <proc-expresion>.

```
interpretador> proc(p) p end
#(struct:closure (p) #(struct:variable-expresion p) #(struct:ambiente-extendido-referenc
ia (var1 var2 var3) #(2 "Hola mundo" "hola") #(struct:ambiente-vacio)))
```

apply: Al usarlo para invocar un procedimiento, evaluamos primero los argumentos y luego ejecutamos el procedimiento con ellos dentro del ambiente. Lo anterior, lo manejamos dentro de la función <aplicar-procedimiento>

INFORME FINAL - INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN - OBLIQ

Juan Pablo Ospina, Diana Marcela Oviedo, Juan Camilo Saenz, Jean Carlos Rojas
2411023, 2459375, 2259416, 2259305
Ingeniería de Sistemas

Universidad del Valle

```
interpretador> letrec pikachu(w) = if is(w,0) then 1 else *(w, apply pika
chu(-(w,1))) end in apply pikachu(6) end
720
```

For: Es una construcción iterativa que permite repetir la evaluación de un conjunto de expresiones dentro de un rango numérico especificado.

```
interpretador> for i=0 to 3 do
       for j=0 to 3 do +(i,j) end end
((0 1 2 3) (1 2 3 4) (2 3 4 5) (3 4 5 6))
```

Object: La estructura object representa un objeto que contiene un conjunto de propiedades y métodos definidos. Cada propiedad está formada por un identificador (la clave) y una expresión asociada (el valor o el cuerpo del método)

```
interpretador> object { x=>3 y=>meth(s) +(2,3) end}
```

meth: La estructura meth permite definir métodos dentro de un objeto. Un método es una función asociada que puede utilizar tanto los datos del propio objeto (accedidos mediante el identificador especial self) como parámetros adicionales proporcionados al momento de su invocación.

```
interpretador> let prueba = object { x=> meth(s,m) +(m,2) end } in for i=0 to 10 do
      send prueba.x(i) end end
(2 3 4 5 6 7 8 9 10 11 12)
```

send: La estructura send es una construcción que permite la invocación de métodos en un objeto, proporcionando un mecanismo para ejecutar comportamientos asociados a claves específicas dentro de un objeto

```
interpretador> let prueba = object{x=>5 y=>6 z=>meth(s) +( get s.x, get s.y) end}
       in send prueba.z() end
11
```

get: La estructura get es una construcción que permite acceder a los valores almacenados en las propiedades de un objeto. Esta operación no modifica el objeto ni las propiedades; únicamente evalúa el identificador del objeto y extrae el valor asociado al identificador de la propiedad especificada.

update: La estructura update se utiliza para modificar el valor asociado a una clave específica dentro de un objeto.

```
interpretador> let prueba = object{x=>5 y=>6 z=>meth(s,p) begin update
s.x := p; get s.x end end}
       in send prueba.z(1) end
1
```

INFORME FINAL - INTERPRETACIÓN Y COMPILACIÓN DE LENGUAJES DE PROGRAMACIÓN - OBLIQ

Juan Pablo Ospina, Diana Marcela Oviedo, Juan Camilo Saenz, Jean Carlos Rojas
2411023, 2459375, 2259416, 2259305
Ingeniería de Sistemas

Universidad del Valle

clone: La construcción clone permite crear una copia de un objeto existente

```
interpretador> let prueba = object{y=>6 z => meth(s) get s.y end
        w => meth(s) update s.y := 8 end x => meth(s) let adentro
one(s) in
        begin send adentro.w(); send adentro.z() end end end }
        in
        begin
        send prueba.w();
        send prueba.z()
        end
        end
8
```

Pruebas

```
interpretador> true
#t
interpretador> false
#f
interpretador> ok
ok
interpretador> "pruebita"
"pruebita"
interpretador> 'Q'
|'Q'|
```

```
interpretador> &("Lenguaje ", "OBLIQ")
"Lenguaje OBLIQ"
interpretador> +(1,2,3,4)
10
interpretador> -(5,2,1)
4
interpretador> *(2,4,2)
16
interpretador> /(12,2)
```

```
interpretador> %(2,3)
2
interpretador> %(4,2)
0
interpretador> <(5,2)
#f
interpretador> >(20,1)
#t
interpretador> <=(3,3)
#t
interpretador> >=(9,9)
#t
```

```
interpretador> is(3,4)
#f
interpretador> is(9,9)
#t
interpretador> if <(1, 10) then 1 else 0 end
1
interpretador> let x = 1 in let y = 2 in +(x,y) end end
3
interpretador> letrec f(x) = if is(x,0) then 0 elseif is(x,1) then 1 else +(apply f(-(x, 1)), apply f
(-(x, 2))) end in apply f(5) end
5
```