Proyecto final - Fundamentos de Lenguajes Programación

Carlos Andres Delgado S, Ing [*]

Mayo de 2017

1. Introducción

El presente proyecto tiene por objeto enfrentar a los estudiantes del curso:

- a la comprensión de todos los conceptos vistos en clase
- a la implementación de un interpretador de un lenguaje de programación
- al análisis de estructuras sintácticas, de datos y de control de un lenguaje de programación para la implementación d en interpretador

2. Lenguaje OBLIQ

Obliq[1] es un lenguaje de programación interpretado que soporta programación orientada a objetos y tiene una estructuras de paso de parámetros por valor. La sintaxis y semántica de Obliq es descrita a continuación:

2.1. Estructuras sintácticas

- **Identificadores:** Son secuencias de caracteres alfanuméricos que comienzan con una letra
- **Definiciones:** Comienzan con la*let*, *let rec* o *var* seguido de una lista de ligaduras a expresiones, separados por comas
- **Expresiones:** En Obliq, casi todas las estructuras sintácticas son una expresión y todas las expresiones producen un valor. Las expresiones pueden ser calificadas en *Identificadores, datos, constructores u operaciones*
- **Comentarios:** Inician con (* y terminan con *)

2.2. Estructuras de datos:

- Constantes:

`true,false`	booleanos
`0,1, 1`	enteros
'a'	caracteres
"abc"	cadenas
`ok`	valor vacío

- Operadores:

```
bool:       not, and, or
 int:   +,-,*,%,<,<=,>,>=, is
text:             &
```

La primitiva **is** retorna verdadero si dos expresiones son iguales (es lo mismo que == de C++ o Java). La primitiva **&** concatena dos textos

- **Objetos:** Los objetos son una colección de **campos** x_i =>a_i donde x_i es un *nombre de campo* y a_i es una expresión:

```
object{x1 => a, x2 => a2, ..., xn => an}
```

Para seleccionar el campo de un objecto se escriba `a.x` donde a es un objeto y x es el campo.

La invocación de un método se realiza con **send a.m(b1,b2,...,bn**donde a es el objeto, m es el método y b1,b2,...,bn son los argumentos

La clonación de un objeto se hace con: `clone(a1,...,an)`. Es implica herencia múltiple, ya que un clone retorna un objeto que recoge los campos y métodos de a1,...,an, deben realizar algunas validaciones para tratar cuando se tienen campos y métodos con **el mismo nombre**

Internamente en un objeto se utiliza:

- **update:** Para cambiar el valor de un campo (es un set básicamente)
- **get:** Para obtener el valor de un campo.

Estos operadores **únicamente** son válidos dentro de objetos

2.3. Estructuras de control

- **Definiciones:**

```
var x1 = a1,...,xn = an in <expresion> end
let x1 = a1,...,xn = an in <expresion> end
let rec x1=p1,...,xn= pn in <expresion> end
```

- Una definición **var** introduce una colección de identificadores actualizables y sus valores iniciales

[*] carlos.andres.delgado@correounivalle.edu.co

- Una definición **let** introduce una colección de identificadores **no actualizables** y sus valores iniciales
- Una definición **let rec** introducción una colección de procedimientos (no actualizables) recursivos

- **Asignación:**

```
set x := a
```

Donde x es un identificador y a es una expresión. Por ejemplo:

```
set x := +(x,1)
```

Tenga presente que la asignación no puede realizarse en definiciones tipo let ni let rec.

- **Secuenciación:**

```
a1;a2;...;an
```

Un conjunto de expresiones puede ser evaluado secuencialmente, separándolos con "punto y coma". Ejemplo:

```
var x = 3 in begin set x :=+(x 2); set  x:=+(
    ↪ x 1); x end end
```

Debe retornar 6. El cual es la ultima expresión en la secuenciación.

- **Procedimientos y métodos:**

```
proc(x1,...,xn) b end
meth(s,x1,...,xn) b end
```

En donde b es una expresión y xi son los argumentos y s es el self (del objeto invocado). El **s** es obligatorio.

- **Condicionales:**

```
if a1 then a2 elsif a3 then a4 . . . else a n
    ↪  end
```

- **Iteraciones:**

```
for x = a  to a2 do a3 end
```

Donde a1 y a2 son expresiones que son números y a3 es una expresión.

3. Evaluación

El proyecto podrá ser realizado en grupos de hasta 3 personas utilizando la librería SLLGEN de Dr Racket. Este debe sustentado y cada persona del grupo obtendrá una nota entre 0 y 1 (por sustentación), la cual se multiplicará por la nota obtenida en el proyecto.

1. **(5 puntos)** Definiciones léxicas y gramaticales
2. **(5 puntos)** Funcionamiento de las constantes
3. **(10 puntos)** Funcionamiento de los operadores de texto, booleanos y enteros
4. **(15 puntos)** Planteamiento del ambiente y funcionamiento de las estructuras de control: var, let y letrec. En este punto se tendrá en consideración la abstracción del ambiente para funcionar con var y let, utilizando un sólo Datatype
5. **(15 puntos)** Procedimientos y su evaluación: Funcionamiento del proc.
6. **(10 puntos)** Objetos: Representación y creación.
7. **(10 puntos)** Objetos: Invocación de métodos y selección de campos
8. **(15 puntos)** Objetos: Clonación.
9. **(5 puntos)** Secuenciación.
10. **(5 puntos)** Condicionales: if.
11. **(5 puntos)** Iteradores: for.

El informe hace parte de cada uno de los puntos (es nota transversal) debe estar en formato **PDF**, bien escrito y con buena presentación y debe contener:

- Sintaxis del lenguaje
- Descripciones de las funciones auxiliares que crearon
- Descripción de la representación de ambientes que utilizaron
- Descripción de la representación de objetos que utilizaron
- Analisis y aplicación de al menos 10 pruebas (que usted se invente) al interpretador
- Conclusiones: No es una conclusión *Dr Racket permite implementar ...* o cosas de la teoría. Estas representan lo que se obtiene con este ejercicio.

Referencias

[1] Cardelli, L. A language with distributed scope. In *Proceedings of the 22Nd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages* (New York, NY, USA, 1995), POPL '95, ACM, pp. 286–297.

Anexos

Factorial

```
let rec
    Fact(n)= if is(n 0) then 1 else *(n apply Fact(-(n 1))) end
in
        apply Fact(5)
end
```

Números primos

```
let primo =
    object { m =>
            meth(s, n)
                begin
                    n;
                    let
                        s0 = clone(s)
                    in
                        update s.m :=
                        meth(s1,n1)
                            if is(%(n1 n) 0) then
                                ok
                            else
                                send s0.m(n1)
                            end
                        end
                    end
                end
            end
}
(* muestra los primos < 100 *)
in
    for i = 2 to 100 do send primo.m(i) end
end
```

Calculadora

```
let calc =
    object{
        arg => 0,
        acc => 0,
        (* entra un nuevo argumento *)
        enter =>
            meth(s, n)
                begin
                    update s.arg := n;
                    s
                end
            end,
            (* la suma *)
            add =>
                meth(s)
                    begin
                        update s.acc := send s.equals;
                        update s.equals := meth(s) +(get s.acc get s.arg) end;
                        s
                    end
                end,
            (* la resta *)
            sub =>
                meth(s)
                    begin
                        update s.acc := send s.equals;
                        update s.equals := meth(s) -(get s.acc get s.arg) end;
                        s
                    end
                end,
            (* el resultado *)
            equals =>
                meth(s)
                    get s.arg end,
            (* inicializar *)
            reset =>
                meth(s)
                    begin
                        update s.arg := 0;
                        update s.acc := 0;
                        update s.equals := meth (s) get s.arg end; s end
                    end
    }
in
    begin
        send calc.reset();
        send calc.enter(3);
        send calc.equals(); (* 3 *)
        send calc.reset();
        send calc.enter(3);
        send calc.sub();
        send calc.enter(2);
        send calc.equals(); (* 1 *)
        send calc.reset();
        send calc.enter(3);
        send calc.add();
        send calc.equals(); (* 6 *)
        send calc.reset();
        send calc.enter(3);
        send calc.add();
        send calc.add();
        send calc.equals; (* 9 *);
        ok
    end
end
```