Proyecto Final Construcción interpretador ObliQ

Sofia Castillo Giraldo

Merly Velásquez Cortez

Faculta de Ingeniera, Universidad del Valle

Fundamentos de Interpretación y Compilación de Lenguajes de Programación

Carlos Andrés Delgado S.

17 de diciembre del 2024

1. Descripción del Problema

Se tiene como objetivo la creación del lenguaje Obliq por medio de un interpretador con la herramienta rakect. Por medio de implementaciones orientado a objetos y estructuras de paso por valor. Todos estos conceptos vistos en clases se tratarán de llevar a cabo en la implementación de este nuevo lenguaje.

2. Diseño de la Solución

Por medio de la gramática, se realizó, primeramente, la especificación léxica y la gramatical, que fue la base del proyecto para construir todas las funciones necesarias y generar las soluciones de los diferentes problemas que se ingresen al interpretador. Se tuvo que analizar la gramática, para luego definir la estructura de cada una de las variantes. Además, se tuvo en cuenta que en este lenguaje Obliq era necesario agregar nuevas variantes.

3. Implementación de la Solución

Para la implementación del lenguaje, se intentó cumplir con cada módulo que conlleva a la solución, se realizó:

- Se definió las nuevas variantes de la gramática tanto en la especificación léxica como en la gramatical.
- la evaluación de todos los ítems/variantes de la gramática.
- Se añadieron y evaluaron procedimientos.
- La creación de objetos, su manipulación de campos y clonación.
- La definición de métodos, clases, ambientes, data-types.

3.1 Implementación Objetos

```
;;Implementacion object-lookup (buscar atributo del objeto)
(define object-lookup
  (lambda (obj attr-id)
    ;; Verifica si el objeto tiene el atributo
    (let ((fields (object->fields obj)))  ;; Obtener los campos del objeto
      (cond
        ;; Si el atributo está en los campos del objeto, devuelve su valor
        ((assoc attr-id fields) => cdr)  ;; Si la clave 'attr-id' está en los campos, devuelve el valor.
        ;; Si no se encuentra el atributo, genera un error
        (else (eopl:error "Atributo no encontrado en el objeto" attr-id))))))
```

```
;; Implementación de object-update! para actualizar campos de objetos
(define object-update!
  (lambda (obj field-id new-value)
    (cases object obj
      (an-object (class-name fields)
        (let ((pos (find-field-position field-id class-name)))
          (if pos
              (begin
                (vector-set! fields pos new-value)
                new-value)
              (eopl:error "Campo no encontrado en el objeto:" field-id)))))))
```

3.2 Evaluación de expresiones

```
;;Evaluar Expresiones Booleanas

(define evaluar-bool-expresion
  (lambda (exp amb)
    (cases bool-expresion exp
      (true-bool-exp () #t)

      (false-bool-exp () #f)

      (bool-prim-exp (prim args)
       (evaluar-bool-primitiva prim
                               (map (lambda (x) (evaluar-expresion x amb)) args)))

      (bool-oper-exp (op args)
       (evaluar-bool-operacion op
                               (map (lambda (x) (evaluar-bool-expresion x amb)) args)))

      (else (eopl:error "No es una expresión booleana válida:" exp)))))
```

```
;;Evaluar Operaciones Booleanas

(define evaluar-bool-operacion
  (lambda (op args)
    (cases bool-oper op
      (not-bool () (operacion-bool-oper args args not))  ;; Negación (not)
      (and-bool () (operacion-bool-oper args (lambda (x y) (and x y)))) ;; C
      (or-bool () (operacion-bool-oper args (lambda (x y) (or x y))))  ;; Di
)))

(define operacion-bool-oper
  (lambda (lval op)
    (cond
      [(null? lval) #t]  ;; Si la lista está vacía, devolvemos true para 'ar
      [(= (length lval) 1) (car lval)]  ;; Si solo hay un elemento, devolvem
      [else
       (op
        (car lval)
        (operacion-bool-oper (cdr lval) op))]  ;; Recursión: aplica la opera
    )
```

```
;;Evaluar Primitivas Booleanas

(define evaluar-bool-primitiva
  (lambda(prim lval)
    (cases bool-primitiva prim
      (mayor-prim ()(operacion-bool-prim lval > #f))
      (mayorigual-prim () (>= (car lval) (cadr lval)))
      (menor-prim () (operacion-bool-prim lval < #f))
      (menorigual-prim () (operacion-bool-prim lval <= #f))
      (is-prim () (operacion-bool-prim lval = #f))
      [else (eopl:error "Operación booleana primitiva no reconocida" prim)]
      )))

(define operacion-bool-prim
  (lambda (lval op term)
    (cond
      [(null? lval) term]
      [else
       (op
        (car lval)
        (operacion-bool-prim (cdr lval) op term))
```

4. Pruebas Realizadas

No se pudo realizar ninguna prueba por conflictos con el parser que no se le pudieron dar una solución adecuada.

5. Conclusiones

- Aunque hubo muchos errores y problemas durante la implementación, se trató de llegar una buena solución, que cubriera cada caso que se presentara en un ejercicio.
- Fue una gran practica para poner a prueba todo lo aprendido en clase, parciales y talleres.
- En general, nos da una idea de cómo funcionan por detrás los lenguajes y todo lo que implica crear uno, fuera de lo que siempre vemos mientras programamos.