Curso: Lenguajes de programación
Práctica 4: Generación de código ejecutable - Análisis semántico.

Karla Ramírez Pulido Alan Alexis Martínez Lopez José Ricardo Desales Santos

José Eliseo Ortíz Montaño

Fecha de inicio: Martes 2 de Mayo del 2023.
Fecha de entrega: Jueves 11 de Mayo del 2023.

Objetivo

Reforzar los conceptos relacionados con los tipos de análisis por los que pasa el código fuente para generar código ejecutable, así como implementar un analizador semántico (*parser*) para el lenguaje `WBAE`.

Estructura

La práctica está conformada por los siguientes archivos:

- `grammars.rkt`. En este archivo se encuentra definida la gramática del lenguaje `WBAE`, la cual fue parte del trabajo de la Práctica 3 y que en su EBNF se representa de la siguiente manera:

```
<expr> ::= <id>
         | <num>
         | <bool>
         | <char>
         | <string>
         | <list>
         | {<op> <expr>+}
         | {with {{<id> <expr>}+} <expr>}
         | {with* {{<id> <expr>}+} <expr>}

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

<bool>  ::= true | false
<char>  ::= 'a' | 'b' | 'c' | ...
<string>::= "a" | "aa" | "ab" | ...
<list>  ::= empty | {lst <expr>*}
<num>   ::= ... | -1 | 0 | 1 | 2 | ...
<op>    ::= + | - | * | / | modulo | min | max | expt | sqrt | sub1 | add1
            | < | <= | = | > | >= | not | and | or | zero?
            | num? | bool? | char? | string? | list?
            | car | cdr   | length | empty? | string-length
```

particularmente, recordemos que contiene las definiciones de los ADT `Binding` y `WBAE`.

- `parser.rkt`. En este archivo se encuentra la definición de la función `(parse sexp)`. Dicha función realiza el análisis sintáctico correspondiente a la `sexp` pasada como parámetro, contruyendo expresiones del TDA `WBAE` que se encuentra definido en el archivo `grammmars.rkt`.
- `interp.rkt`. En este archivo se encuentra la definición de la función `(interp exp)`. Dicha función realiza el análisis semántico de las expresiones del lenguaje `WBAE`; esto es, una vez habiendo realizado el proceso de análisis sintáctico (*parsing*) de una s-expression (`sexp`) se procede a la evaluación semántica de esta, a partir de la cual obtenemos el resultado final del programa escrito.
- `test.rkt`. Aquí se encuentran definidas un conjunto de pruebas unitarias para probar el trabajo realizando en la práctica. En el intérprete, se deberá dar click en el botón de *Run* para ejecutar dichas pruebas. El pasar todas las pruebas satisfactoriamente no garantiza obtener la máxima calificación en la práctica, ya que esto depende también de la manera en cómo se desarrolle esta (trabajo colaborativo, buenas prácticas de programación, etc...).[1]

Ejercicios

1. **(1 pt)** Modificar el cuerpo de la función `(parse sexp)` para que existan listas vacías dentro del lenguaje `WBAE`. Recordaremos que la especificación de la Práctica 3 indica que:

   "... [se define la expresión] `<list>`, de manera que su identificador sea `lst`, y como único parámetro tenga una lista, llamada `l`, de elementos de tipo `WBAE`."

   por lo que hasta ahora no se cuenta con alguna definición para este tipo de listas. Con esto `(lst empty)` y `(lst '())` son parte del lenguaje.

```
;; Ejemplo del proceso de parsing de las listas vacías.

>(parse empty)
(lst empty)

>(parse '())
(lst '())
```

2. **(1 pt)** Modificar el cuerpo de la función `(parse sexp)` de tal manera que los símbolos `'true` y `'false` sean analizados sintácticamente a sus correspondientes constantes del ADT `WBAE`. Es importante recalcar que se está haciendo énfasis en el proceso de parseo de **símbolos**[2]

```
;; Ejemplo del proceso de parsing de los símbolos 'true y 'false respectivamente
>(parse 'true)
(bool true)

>(parse 'false)
(bool false)
```

3. **(2 pts)** Completar la firma y cuerpo de las funciones `anD` y `oR`[3] siguiendo la siguiente especificación:
   - Ambas funciones reciben un ejemplar de `lst`. Esto implica que a pesar de que la firma corresponda a una función unaria, la interpretación semántica corresponde a la de una función n-aria. Esto quiere decir que se tiene que hacer un ajuste con respecto a la verificación de la aridad de dichas funciones en el proceso de análisis sintáctico con respecto a lo establecido en la práctica anterior.
   - `(anD l)` regresa `(bool true)` si y sólo si todos los elementos de `lst` se evalúan a `(bool true)` o si `l` es igual a la lista vacía. En algún otro caso regresa `(bool false)`.

[1] Es importante almacenar todos los archivos dentro de un mismo directorio.
[2] Recordemos el operador `quote` de Racket.
[3] Definidas en la práctica anterior

- `(oR l)` regresa `(bool true)` si y sólo si existe algún elemento en `lst` que se evalúe a `(bool true)`, En algún otro caso regresa `(bool false)`.

Hint: investiga las funciones `andmap` y `ormap` nativas de Racket.

4. **(2 pts)** Completar el cuerpo de la función `(subst expr id value)`, la cual recibe una expresión perteneciente al lenguaje `WBAE` (`expr`), un identificador y un valor (`id` y `value` respectivamente), y sustituye las apariciones de `id` por `value` en `expr`. [4]

5. **(4 pts)** Completar el cuerpo de la función `(interp exp)`, la cual, como ya se ha mencionado, realiza el análisis semántico de la expresión que recibe como parámetro. Para esto, considera las siguientes especificaciones:

   - Los identificadores no pueden ser evaluados -observa que no tienen un valor asignado-, por lo que en este caso se debe regresar un error indicando que el identificador se encuentra libre.
   - Los valores numéricos, constantes booleanas, los caracteres y las cadenas se evalúan asimismas.
   - La interpretación de un ejemplar de `lst` corresponderá a la interpretación de cada uno de sus elementos, o en su defecto, al valor constante de la lista vacía.
   - Los operadores se evalúan al resultado de aplicar la función tomando cada uno de sus operandos. Hasta ahora **no** estamos verificando los tipos de los argumentos de los procedimientos, por lo que si en este punto los argumentos no son del tipo correcto, se deja a elección de las y los programadores la forma en la que se manejarán dichos errores. [5] Este punto deberá ser plenamente comentado y justificado.
   - Las expresiones `with` son multiparamétricas, lo cual significa que pueden recibir más de un identificador.
   - Las expresiones `with*` permiten definir identificadores en términos de otros definidos con anterioridad, por lo que su interpretación es muy similar a la que se realiza en las expresiones `with`, pero también se tienen que interpretar los identificadores. En otras palabras, podemos ver a las expresiones `with*` como expresiones `with` anidadas.
     Considera la siguiente expresión:

     ```
     '(with*
         ([a 2]
           [b {+ a a}])
           b)
     ```

     la cual es equivalente a[6][7]:

     ```
     '(with ([a 2])
         (with ([b {+ a a}])
             b))
     ```

     de donde, si realizamos el proceso de sustitución tal y como se nos indica:

     ```
     '(with ([b {+ 2 2}])
            b)
     ```

     y finalmente, la expresión a interpretar (sin ninguna sustitución pendiente) es:

     ```
     (+ 2 2)
     ```

     Ahora, como segundo ejemplo consideremos la siguiente expresión:

     ```
     '(with* ([x 1]
         [y 2])
         (+ y x))
     ```

     la cual es equivalente a:

[4] Esta función corresponde a la implementación del *algoritmo de sustitución* visto en clase.
[5] La verificación de tipos es un trabajo que se realizará en prácticas posteriores.
[6] Esta equivalencia semántica nos indica que `with*` es *azúcar sintáctica* la cual será eliminada como parte del *desugar* aplicado al lenguaje en prácticas posteriores.
[7] De hecho, las expresiones `with` también son azúcar sintáctica de *aplicaciones de funciones.*

```
'(with ([x 1])
    (with ([y 2])
        (+ y x)))
```

y realizando el proceso de sustitución correspondiente se obtiene:

```
'(with ([y 2])
    (+ y 1))
```

y realizando la última sustitución obtenemos la expresión a interpretar:

```
(+ 2 1)
```

```
;; interp: WBAE -> WBAE
(define (interp expr) ... )

> (interp (parse 'foo))
error: Identificador libre

> (interp (parse 12345))
(num 12345)

> (interp (parse true))
(bool true)

> (interp (anD (list (bool true) (bool true))))
(bool true)

> (interp (oR (list (bool false) (bool false))))
(bool false)

> (interp (parse '{+ 1 2 3 4 5}))
(num 15)

> (interp (parse '{with {{a 2} {b 3}} {+ a b}}))
(num 5)

;; Es improtante observar el caso de with*, el cual corresponde a asignaciones with
;; anidadas.
> (interp (parse '{with* {{a 2} {b {+ a a}}} b}))
(num 4)

> (interp (op + (list (num 1) (num 2))))
(num 3)
```

Entrega

Acerca de la entrega de la práctica:

- La realización y entrega de la práctica deberá realizarse en equipos de a lo más 4 personas.[8]
- Su entrega debe consistir en los archivos `grammars.rkt`, `interp.rkt` y `parser.rkt`, los cuales se proporcionan junto con este archivo. Recuerden agregar, en cada uno de estos, los datos de los integrantes del equipo.
- El uso de funciones auxiliares está totalmente permitido, sin embargo, deben ser declaradas justo después de la función principal que hace uno de ellas.
- Las funciones deben incluir comentarios, tanto de documentación como del proceso de desarrollo. Estos deben ser claros, concisos y descriptivos.
- Queda estrictamente prohibido utilizar funciones primitivas del lenguaje que resuelvan directamente los ejercicios.[9]
- Se deberá subir la versión final de su práctica al apartado del classroom correspondiente antes de la fecha límite. Esto sólo debe realizarlo un integrante del equipo; el resto deberá marcar como entregada la actividad y en un comentario privado especificar quienes son los miembros del equipo.
- Para la administración de la práctica en GitHub, selecciona el siguiente enlace.

[8] Cualquier situación con respecto a este punto será tratada de acuerdo a las particularidades del caso. Para esto, acercarse al ayudante del rubro del laboratorio a la brevedad.

[9] Cualquier duda con respecto a este punto, por favor manda un correo para atenderla.