Lenguajes de Programación
Practica 3

Karla Ramírez Pulido   José Eliseo Ortiz Montaño

Semestre 2024-2   **Fecha de inicio:** 15 de marzo 2024
Facultad de Ciencias UNAM   **Fecha de entrega:** 5 de abril 2024

Instrucciones

La realización y entrega de la práctica deberá realizarse en equipos de a lo más 3 personas[1]. La entrega debe respetar el orden de las funciones que es especificado en este documento. El uso de funciones auxiliares está totalmente permitido, sin embargo, deben ser declaradas justo después de la función principal que hace una de ellas.

```
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
;; Ejercicio 1
;; [Documentación de la función]
(define (my-fun a b c d) ... )
;; [Documentación de la función auxiliar - incluir una breve descripción de
;; la motivación de la función]
(define (an-aux-fun lst1 lst2) ... )
...
...
...
;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;;
```

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.

Se deberá subir la versión final de su práctica al apartado del classroom correspondiente antes de la fecha límite. Esto solo 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.

El archivo a entregar debe de ser un comprimido `zip`, con el nombre `practica03.zip`, en éste se deberán incluir los archivos `interp.rkt`, `parser.rkt`, `grammars.rkt` y `readme.md`, en el cual se deberán incluir los datos de cada miembro del equipo.

[1] 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.

Estructura

La práctica está conformada por los siguientes archivos:

- `grammars.rkt`: archivo en el que se encuentra la definición de tipos que representan el **ASA**[2] del lenguaje `WAE`, el cual corresponde a la siguiente sintaxis concreta:

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

<num> ::= ... | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | ...
<id> ::= a | b | c | ... | z | A | B | ... | Z | aa | ab | ac | ...
<bool> ::= true | false
<op> ::= +
       | -
       | *
       | /
```

- `parser.rkt`: archivo donde se encuentra la definición de la función `parse`, encargada de realizar el análisis sintáctico correspondiente.
- `interp.rkt`: archivo en el que se encuentra la definición de la función `interp`, la cual estará encargada de realizar el análisis semántico de una expresión del lenguaje correspondiente, y a su vez contendrá la definición de la función `subst` que aplicará el algoritmo de sustitución sobre una expresión del lenguaje.
- `test.rkt`: archivo que contendrá una serie de pruebas unitarias para probar el trabajo realizado en la práctica.

[2] Árbol de Sintaxis Abstracta, ASA.

Ejercicios

1. (1pt) Extender la gramática del lenguaje `WAE` siguiendo la siguiente gramática:

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

<num> ::= ... | -4 | -3 | -2 | -1 | 0 | 1 | 2 | 3 | 4 | ...
<id> ::= a | b | c | ... | z | A | B | ... | Z | aa | ab | ac | ...
<string> ::= "a" | "b" | "c" | ... | "z" | "A" | "B" | ... | "Z" | "aa" | "ab" |
...
<bool> ::= true | false
<op> ::= +
       | -
       | *
       | /
       | modulo
       | min
       | max
       | expt
       | sqrt
       | sub1
       | add1
       | <
       | >
       | <=
       | >=
       | =
       | not
       | and
       | or
       | zero?
       | num?
       | str?
       | bool?
       | str-length
       | str-first
       | str-last
```

   Para el caso de las expresiones `<string>` se deberá agregar la variante `str` al tipo que representa el ASA de esta nueva sintaxis concreta.

2. (3pt) Completar el cuerpo de la función `parse` del archivo `parser.rkt`, la cual es la encargada de realizar el análisis sintáctico. Toma en cuenta lo siguiente:

   - Para las funciones `sub1, add1, sqrt, not, zero?, num?, str?, bool?, str-length, str-first, str-last` la aridad tiene que ser 1.
   - Para las funciones `modulo y expt` la aridad tiene que ser 2.
   - Para el resto de funciones simplemente se tiene que verificar que los argumentos que reciben sea mayor a 0.
   - En todos los casos anteriores, se deberá arrojar un error en caso de que la aridad sea incorrecta; especificando la cantidad de argumentos dados y la cantidad de argumentos esperados.

- En el caso de `with` no se puede declarar dos variables de ligado (*bindings*) con el mismo identificador, y si esto sucede se debe mandar un error.
- Se debe verificar la estructura de las variables de ligado (*bindings*) correspondientes a las expresiones `with` y `with*`, esto es, un *binding* similar a `{x 10 10}` debe de mandar un error al momento de intentar parsearlo.

Ejemplos:

```
> (parse 'x)
(id 'x)
> (parse 2)
(num 2)
> (parse 'true)
(bool #t)
> (parse "Hola")
(str "Hola")
> (parse '{sub1 2})
(op sub1 (list (num 2)))
```

3. (3pt) Completar el cuerpo de la función `subst` del archivo `interp.rkt`, la cual recibe un identificador, un valor y una expresión perteneciente al lenguaje en forma de ASA, y sustituye las apariciones del identificador por el valor dentro de la expresión tomando en cuenta lo siguiente:

   - Si se encuentra un identificador se verifica si es el que se busca reemplazar, si este es el caso se reemplaza por el valor, en caso contrario se deja el mismo identificador.
   - Si se encuentra una operación se aplica recursión sobre todos sus argumentos.
   - Si se encuentra una expresión `with`, primero se verifica que el identificador que se busca reemplazar no este siendo declarado dentro de los `bindings` de la expresión, en este caso se hace la sustitución en cada uno de los valores de los `bindings` y en el cuerpo de la expresión, en caso de que si este siendo declarado solo se debe hacer la sustituciones en los valores de cada `binding`.
   - Si se encuentra una expresión `with*` primero se verifica que el identificador que se busca reemplazar no este siendo declarado dentro de los `bindings` de la expresión, en este caso se hace la sustitución en cada uno de los valores de los `bindings` y en el cuerpo de la expresión, en caso de que si este siendo declarado solo se debe hacer la sustituciones en los valores de cada `binding` exceptuando los que estén después de la declaración del identificador. Por ejemplo, si se busca hacer la sustitución de `[x ::= 10]` en la siguiente expresión:

     ```
     {with* {{y 20} {z y} {x {+ z x}} {w x}}
         {+ x y z w}}
     ```

     el resultado sería la siguiente expresión:

     ```
     {with* {{y 20} {z y} {x {+ z 10}} {w x}}
         {+ x y z w}}
     ```

   Ejemplos:

   ```
   > (subst 'x (num 2) (id 'x))
   (num 2)
   > (subst 'x (num 10) (op + (list (id 'x) (num 10))))
   (op #<procedure:+> (list (num 10) (num 10)))
   > (subst 'x (num 1) (bool #t))
   (bool #t)
   > (subst 'x (num 10) (with (list (binding 'x (id 'x))) (id 'x)))
   (with (list (binding 'x (num 10))) (id 'x))
   ```

4. (3pt) Completar el cuerpo de la función `interp` del archivo `interp.rkt`, la cual corresponde al análisis semántico de la expresión recibida como parámetro. Se debe considerar lo siguiente:

   - Si se recibe un identificador significará que este está libre, por lo que se deberá enviar un error.
   - Los valores numéricos, constantes booleanas y cadenas se evalúan así mismas.
   - En este punto no se están verificando los tipos de los argumentos de las operaciones, por lo que si se le es pasado un argumento del tipo incorrecto se deja a libre elección la forma de manejar estos errores. Esto deberá ser justificado.
   - La expresión `with*` permiten definir identificadores en términos de otros definidos con anterioridad, por lo que su interpretación es similar a la que se realiza en las expresiones `with`, pero también se tiene que interpretar los identificadores. Por lo anterior se puede ver a las expresiones `with*` como expresiones `with` anidadas.

   Ejemplos:

```
> (interp (id 'x))
interp: Variable libre
> (interp (num 10))
10
> (interp (op + (list (num 10) (num 10) (num 10))))
30
> (interp (with (list (binding 'x (num 10)) (binding 'y (num 20))) (op > (list (id 'x)
    (id 'y)))))
#f
> (interp (with* (list (binding 'x (num 10)) (binding 'y (id 'x))) (op = (list (id 'x)
    (id 'y)))))
#t
```