Lenguajes de Programación

Práctica 4

Semestre 2022-1
Facultad de Ciencias, UNAM

Profesora	Karla Ramírez Pulido	**Fecha de inicio:** 19 de Noviembre de 2021
Ayud. Lab	Silvia Díaz Gómez	**Fecha de entrega:** 01 de Diciembre de 2021
Ayud. Lab	Fhernanda Montserrat Romo Olea	

Descripción

La práctica consiste en implementar un intérprete sencillo para el lenguaje **FWAE**. Para esto, se debe completar el cuerpo de las funciones faltantes dentro de los archivos **grammars.rkt**, **parser.rkt**, **desugar.rkt** e **interp.rkt** hasta que pasen las pruebas unitarias incluidas en el archivo **test-practica4.rkt** y se ejecute correctamente el archivo **practica4.rkt**.

La gramática del lenguaje **FWAE** se presenta a continuación:

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

<id> ::= a | b | c | ...

<num> ::= 1 | 2 | 3 | ...

<op> ::= + | - | * | / | modulo | expt | add1 | sub1
```

Ejercicios

1. (2 pts.) Completar el cuerpo de la función **(parse sexp)** dentro del archivo **parser.rkt** el cual recibe una expresión simbólica[1], realiza el análisis sintáctico correspondiente es decir, construir el Árbol de Sintaxis Abstracta[2] (ASA).

[1] Del inglés, *s-expression*. Puede ser un número, un símbolo o una lista de expresiones simbólicas.
[2] Del inglés, *Abstract Syntax Tree (AST)*.

```
;; Definicion del tipo Binding.
(define-type Binding
   [binding (id symbol?) (value SAST?)])

;; Definicion del tipo SAST.
(define-type SAST
   [idS    (i symbol?)]
   [numS   (n number?)]
   [opS    (f procedure?) (args (listof SAST?))]
   [withS  (bindings (listof binding?)) (body SAST?)]
   [withS* (bindings (listof binding?)) (body SAST?)]
   [funS   (params (listof symbol?)) (body SAST?)]
   [appS   (fun-expr SAST?) (args (listof SAST?))])

;; parse: s-expression → AST
(define (parse sexp) ...)
```

2. (5 pts.) Completar el cuerpo de la función **`(desugar expr)`** dentro del archivo **`desugar.rkt`**. Esta función recibe una expresión en forma de Árbol de Sintaxis Abstracta Endulzado[3], eliminando el azúcar sintáctica de las expresiones correspondientes. Y regresará un nuevo árbol. Las expresiones con azúcar sintáctica son:

   - **Asignaciones locales**

     Las expresiones **`with`** pasan a ser aplicaciones de función. Expresiones de la forma:

     ```
     {with {{<id> <value>}}
        <body>}
     ```

     se transforma en una aplicación de función respectivamente. El **`<id>`** y el **`<body>`** del **`with`** forman el parámetro y cuerpo de la función, mientras que el campo **`<value>`** representa el argumento a aplicar.

     ```
     {{fun {<id>} <body>} <value>}
     ```

   - **Asignaciones locales anidadas**

     Las expresiones **`with*`** pasan a ser expresiones **`with`** anidadas. Expresiones de la forma:

     ```
     {with* {{<id> <value>} {<id> <value} ...}
        <body>}
     ```

     las cuales se transforman en expresiones **`with`** anidadas de tal forma que cada pareja de identificador y valor **`{<id> <value>}`** representa una nueva asignación local.

[3] *Sugared Abstract Syntax Tree.*

```
{with {{<id> <value>}}
   {with {{<id> <value>}}
      ...
      <body>}...}
```

- **Funciones**

  Las expresiones de tipo función **fun** con n parámetros se deberán transformar en funciones con un único parámetro, es decir, expresiones currificadas. Si se tiene una expresión:

  ```
  {fun {<param> <param> ...} <body>}
  ```

  se deberá transformar en una expresión **fun** anidada de tal forma que cada función regrese una nueva función como resultado.

  ```
  {fun {<param>} {fun {<param>} ... <body>} ...}
  ```

- **Aplicaciones de función**

  Dado que las funciones deben currificarse, las aplicaciones de función, deben simplificarse de tal forma que se vayan aplicando argumento por argumento. Recuerda que la aplicación de funciones asocia a la izquierda. Así, las expresiones de la forma:

  ```
  {<fun-expr> <arg> <arg> ...}
  ```

  se transformarán en aplicaciones de función argumento por argumento.

  ```
  {{<fun-expr> <arg>} <arg>} ...}...}
  ```

```
;; Definicion del tipo AST.
(define-type AST
   [id   (i symbol?)]
   [num  (n number?)]
   [op   (f procedure?) (args (listof AST?))]
   [fun  (param symbol?) (body AST?)]
   [app  (fun-expr AST?) (arg AST?)])

;; desugar: SAST → AST
(define (desugar expr) ...)
```

3. (3 pts.) Completar el cuerpo de la función **(interp expr env)** dentro del archivo **interp.rkt** que dada una expresión, regresa la evaluación correspondiente. Para la evaluación de expresiones debe usarse alcance estático. Tomar en consideración:

   - **Identificadores**

     Se debe lanzar un error indicando que se trata de una variable libre.

```
(interp (desugar (parse 'foo)) (mtSub)) => error: Variable libre
```

- **Números**

  Al ser un valor atómico, los números se evalúan así mismos.

```
(interp (desugar (parse 1729)) (mtSub)) => (numV 1729)
```

- **Operaciones aritméticas**

  Se debe aplicar el operador correspondiente a la lista de operandos indicada.

```
;; Operaciones unarias
(interp (desugar (parse '{add1 18})) (mtSub))     => (numV 19)
(interp (desugar (parse '{sub1 35})) (mtSub))     => (numV 34)

;; Operaciones binarias
(interp (desugar (parse '{modulo 10 2})) (mtSub)) => (numV 0)
(interp (desugar (parse '{expt 2 3})) (mtSub))    => (numV 8)

;; Operaciones n-arias
(interp (desugar (parse '{+ 1 2 3})) (mtSub))     => (numV 6)
(interp (desugar (parse '{- 3 2 1})) (mtSub))     => (numV 0)
(interp (desugar (parse '{* 1 2 3})) (mtSub))     => (numV 6)
(interp (desugar (parse '{/ 8 2 2})) (mtSub))     => (numV 2)
```

- **Funciones**

  Cuando una expresión es de tipo función, se genera una cerradura (*closure*) la cual almacena el ambiente donde fue definida la función, que permite implementar alcance estático.

```
(interp (desugar (parse '{fun {x} x}))) =>
   (closureV 'x (id 'x) (mtSub))
```

- **Aplicación de función**

  Cuando se tienen aplicaciones de función se deberá aplicar la sustitución del parámetro formal por el parámetro real en el cuerpo de la misma. Para esto debe buscarse el valor de cada variable en el ambiente correspondiente.

```
(interp (desugar (parse '{{fun {x} x} 2})) (mtSub)) => (numV 2)
```

```
;; interp: AST → number
(define (interp expr) ...)
```

Referencias

[1] Shriram Krishamurthi, *Programming Languages: Application and Interpretation,* First Edition, Brown University, 2007.