Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Práctica 4

Karla Ramírez Pulido
karla@ciencias.unam.mx

J. Ricardo Rodríguez Abreu
ricardo_rodab@ciencias.unam.mx

Manuel Soto Romero
manu@ciencias.unam.mx

Luis A. Patlani Aguilar
ayudantefc@gmail.com

Fecha de inicio: 10 de octubre de 2017
Fecha de término: 20 de octubre de 2017
Semestre 2018-1

Objetivos

- Agregar elementos de tipo booleano, nuevos operadores aritméticos y relacionales, además de funciones al lenguaje de la Práctica 3.
- Implementar un programa que realice transformaciones al código para eliminar el azúcar sintáctica de algunas expresiones.
- Modificar el intérprete implementado en la Práctica 3 para que utilice alcance estático mediante el uso de cerraduras y ambientes, en lugar de emplear el algoritmo de sustitución textual.

Descripción general

La práctica consiste en extender el intérprete de la práctica anterior. La gramática en EBNF para las expresiones del lenguaje `FWBAE` (*Functions With Boolean Arithmetic Expressions*), que se debe implementar en esta práctica es la siguiente:

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

<id> ::= a |... | z | A | ... | Z | aa | ab | ... | aaa | ...
         (Cualquier combinación de caracteres alfanuméricos
```

```
          con al menos uno alfabético)

<num> ::= ... | -2 | -1 | 0 | 1 | 2 | ...
        (Cualquier número admitido por Racket)

<bool> ::= false | true

<op> ::= + | - | * | / | % | min | max | pow
       | not | and | or | < | > | <= | >= | = | /=
```

Archivos requeridos

El material de esta práctica consta de los siguientes archivos[1]:

- `grammars.rkt` archivo con la definición de los tipos de datos abstractos que definen cada uno de los ASA para el lenguaje `FWBAE`.
- `parser.rkt` archivo en el cual se debe implementar el analizador sintáctico para el lenguaje `FWBAE`.
- `interp.rkt` archivo donde se debe implementar el analizador semántico para el lenguaje `FWBAE`.
- `practica4.rkt` archivo con la lógica necesaria para ejecutar el intérprete final de `FWBAE`.
- `test-practica4.rkt` archivo en el cual se deben agregar las pruebas unitarias de la práctica.

Desarrollo de la práctica

Ejercicio 4.1 (2 pts.) En el archivo `grammars.rkt` se encuentra el TDA que define los constructores para realizar el análisis sintáctico del lenguaje `FWBAE`.

```
;; TDA para representar los árboles de sintaxis abstracta
;; del lenguaje FWBAE. Este TDA es una versión con azúcar sintáctica.
(define-type FWBAE
   [idS (i symbol?)]
   [numS (n number?)]
   [boolS (b boolean?)]
   [opS (f procedure?) (args (listof FWBAE?))]
   [withS (bindings (listof binding?)) (body FWBAE?)]
   [withS* (bindings (listof binding?)) (body FWBAE?)]
   [funS (params (listof symbol?)) (body FWBAE?)]
   [appS (fun-expr FWBAE?) (args (listof FWBAE?))])
```

[1]Los archivos pueden descargarse desde la página del curso `http://lenguajesfc.com`.

```
;; TDA para asociar identificadores con valores.
(define-type Binding
   [binding (name symbol?) (value FWBAE?)])
```

El primer ejercicio a implementar en el intérprete de esta práctica, consiste en completar el cuerpo de la función `parse` (incluida en el archivo `parser.rkt`) para que realice el análisis sintáctico correspondiente. Además, se deben agregar al menos cinco pruebas unitarias dentro del archivo `test-practica4.rkt`[2].

```
;; parse: s-expression -> FWBAE
(define (parse sexp)
   (error 'parse "Función no implementada."))
```

```
> (parse '{fun {x} {+ x 2}})
(funS '(x) (opS + (list (idS 'x) (numS 2))))
```

Ejercicio 4.2 (2 pts.) Una vez que se complete el cuerpo de la función `parse`, implementar el cuerpo de la función `desugar` incluida en el archivo `parser.rkt`, el cual elimina el azúcar sintáctica[3] de las expresiones de `FWBAE`, es decir, las convierte en expresiones de `FBAE`:

```
;; TDA para representar los árboles de sintaxis abstracta
;; del lenguaje FBAE. Este TDA es una versión sin azúcar sintáctica.
(define-type FBAE
   [id (i symbol?)]
   [num (n number?)]
   [bool (b boolean?)]
   [op (f procedure?) (args (listof FBAE?))]
   [fun (params (listof symbol?)) (body FBAE?)]
   [app (fun-expr FBAE?) (args (listof FBAE?))])
```

Se deben agregar al menos cinco pruebas unitarias dentro del archivo `test-practica4.rkt` que correspondan con las pruebas agregadas en el ejercicio anterior.

Las únicas expresiones que tienen azúcar sintáctica, en esta práctica, son:

- `with` estas expresiones son una versión endulzada de aplicaciones a función. Por ejemplo:

  ```
  {with {{a 3}}
     {+ a 4}}
  ```

  se transforma en

[2] Se deben agregar pruebas significativas.

[3] Tipo de sintaxis que hace que un programa sea más "dulce" o fácil de escribir.

```
{{fun {a} {+ a 4}} 3}
```

- `with*` este tipo de expresiones son una versión endulzada de expresiones `with` anidadas que a su vez tienen azúcar sintáctica. Por ejemplo:

  ```
  {with* {{a 2} {b {+ a a}}}
     b}
  ```

  se transforma en

  ```
  {with {{a 2}}
     {with {{b {+ a a}}}
        b}}
  ```

  que a su vez se convierte en

  ```
  {{fun {a} {{fun {b} b} {+ a a}}} 2}
  ```

```
;; desugar: FWBAE -> FBAE
(define (desugar expr)
   (error 'desugar "Función no implementada."))
```

```
> (desugar (parse '{with {{a 3}} {+ a 4}}))
(app (fun '(a) (op + (id 'a) (num 4))) (list (num 3)))
```

Ejercicio 4.3 (6 pts.) El analizador semántico de esta práctica debe realizar evaluación mediante ambientes y cerraduras para implementar alcance estático. De esta forma se recibe un ASA (sin azúcar sintáctica) y un ambiente de evaluación definido por:

```
;; TDA para representar el ambiente de evaluación.
(define-type Env
   [mtSub]
   [aSub (name symbol?) (value FBAE-Value?) (env Env?)])
```

Los resultados devueltos por el analizador semántico, que almacena el ambiente deben ser de tipo `FBAE-Value`:

```
;; TDA para representar los resultados devueltos por el intérprete.
(define-type FBAE-Value
   [numV (n number?)]
   [boolV (b boolean?)]
   [closureV (params (listof symbol?)) (body FBAE?) (env Env?)])
```

El ejercicio consiste en completar el cuerpo de la función `(interp exp env)` (ver archivo `interp.rkt`) para que realice el análisis semántico correspondiente, es decir, evaluar expresiones de `FBAE`. Tomar los siguientes puntos a consideración:

- El valor de los identificadores debe ser buscando en el ambiente de evaluación mediante la definición de una función `lookup`, la cual encuentra el valor asociado en el ambiente y lo regrese o bien reporte un error, en caso de que no se haya encontrado. Ejemplo:

  ```
  > (interp (desugar (parse 'foo)) (mtSub))
  error: Free identifier
  ```

- Los números se evalúan a valores de tipo `numV`. Ejemplo:

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

- Las expresiones booleanas se evalúan a valores de tipo `boolV`. Ejemplo:

  ```
  > (interp (desugar (parse 'true)) (mtSub))
  (boolV #t)
  ```

- Los operadores son n-arios, por lo que esta versión del intérprete tiene un constructor `op` que recibe una función con la cual aplica la operación definida a cada uno de sus parámetros. Ejemplo:

  ```
  > (interp (desugar (parse '{/= 1 2 3 4 5})) (mtSub))
  (boolV #t)
  ```

- Las expresiones `with` son multiparamétricas, es decir, tienen más de un identificador. Ejemplo:

  ```
  > (interp (desugar (parse '{with {{a 2} {b 3}} {+ a b}}))) (mtSub))
  (numV 5)
  ```

- Las expresiones `with*` presentan un comportamiento parecido al de la primitiva `with`, sin embargo, estas expresiones permiten definir identificadores en términos de otros definidos anteriormente. Por ejemplo: `{with* {{a 2} {b {+ a a}}} b}`. Se interpreta similar a `with`, la única diferencia es que también se deben procesar los identificadores. Ejemplo:

  ```
  > (interp (desugar (parse '{with* {{a 2} {b {+ a a}}} b})) (mtSub))
  (numV 4)
  ```

- Las funciones deben evaluarse a una cerradura que incluya: los parámetros de la función, el cuerpo de la función y el ambiente dónde fue definida con el fin de evaluarlas usando alcance estático. Ejemplo:

  ```
  > (interp (desugar (parse '{fun {x} {+ x 2}})) (mtSub))
  (closureV '(x) (op + (list (id 'x) (num 2)) (mtSub))
  ```

- Las aplicaciones de función deben de evaluar el cuerpo de la función correspondiente considerando el ambiente donde ésta fue definida. Por ejemplo, en la siguiente llamada, el ambiente de evaluación es `(aSub 'b (numV 3) (aSub 'a (numV 2) (mtSub)))`:

  ```
  > (interp (desugar (parse '{{fun '{a b} {+ a b}} 2 3})) (mtSub))
  (numV 5)
  ```

Además, se deben agregar al menos cinco pruebas unitarias dentro del archivo `test-practica4.rkt` que correspondan con las pruebas agregadas en los ejercicios anteriores.

```
;; interp: FBAE -> FBAE-Value
(define (interp expr env)
   (error 'interp "Función no implementada."))
```

```
> (interp (desugar (parse '{or {not true} false})) (mtSub))
(boolV #f)
```

Ejercicio 4.4 (0 pts.) Una vez finalizados los ejercicios anteriores, ejecutar el archivo `practica4.rkt`. Realizar pruebas significativas para verificar que la práctica se completó con éxito.

```
Bienvenido a DrRacket, versión 6.8 [3m].
Lenguaje: plai, with debugging; memory limit: 128 MB.
(λ) {{fun {x} {+ x 2}} 3}
5
(λ) {with {{p true} {q false}} {or p q}}
true
(λ) {with {{f {fun {x} {pow x 2}}} {a 3}} {f a}}
9
(λ) {exit}
```

Figura 1: *Ejecución de* `FWBAE`

Puntaje total: *10 puntos*

Entrega

Los archivos que se deben enviar a los ayudantes de laboratorio son:

- `parser.rkt`
- `interp.rkt`
- `test-practica4.rkt`

Recordando seguir los lineamientos de entrega de prácticas especificados en la sección correspondiente de la página del curso: `http://lenguajesfc.com/lineamientos.html`.

Puntos extra

Escoger alguno de los siguientes ejercicios y resolverlo **individualmente**[4]:

[4]Pueden enviarse ambos ejercicios para ser revisados, sin embargo, sólo uno será tomado en cuenta.

Punto extra 4.1 (1 pt.) Investigar y dar un ejemplo de un software o programa que haya sido escrito en un lenguaje de programación que tenga alcance dinámico. Mencionar tres ventajas y tres desventajas de usar este tipo de alcance (tomar como referencia a los autores/creadores del lenguaje y del software) y explicar con palabras propias cómo el alcance estático hubiera mejorado el desempeño de dicho software. La investigación debe ser de al menos una cuartilla y debe incluir: Título, Introducción, Desarrollo, Conclusiones y Bibliografía. No se tomará en cuenta ninguna investigación que no esté debidamente citada e incluya las referencias correspondientes. La investigación deberá entregarse de forma impresa a más tardar el 20 de octubre de 2017 durante la sesión de laboratorio.

Punto extra 4.2 (1 pt.) En la tarea 2 del curso se incluyó el siguiente enunciado:

> *En clase se ha visto que que la definición de sustitución resulta ser ineficiente ya que en el peor caso es de orden cuadrático en relación al tamaño del programa (considerando el tamaño del programa como el numero de nodos en el árbol de sintaxis abstracta), por otro lado se analizó la alternativa de diferir la sustitución por medio ambientes. Sin embargo, implementar un ambiente usando un stack no parece ser mucho mas eficiente.*

El punto extra consiste en implementar el ambiente usando una estructura de datos que un intérprete pudiera usar para mejorar su complejidad. Se debe justificar el diseño e implementación en un archivo PDF por separado. El ejercicio debe enviarse a más tardar el 20 de octubre de 2017 en un archivo `cuenta_p4.tar.gz` con los archivos requeridos, dónde `cuenta` es el número de cuenta del alumno, al correo `manu+ldp@ciencias.unam.mx` con el asunto `[LDP-Extra P4]`.

Referencias

Algunas referencias de consulta:

[1] Karla Ramírez, Manuel Soto, Ricardo Rodríguez, *Notas de laboratorio del curso de Lenguajes de Programación*, Semestre 2018-1, Facultad de Ciencias, UNAM. Disponibles en: [`http://lenguajesfc.com/notas.html`].

[2] Rodrigo Ruiz Murgía, *Manual de prácticas para la asignatura de Lenguajes de Programación*, Reporte de actividad docente, Facultad de Ciencias, 2016.

[3] Shriram Krishnamurthi, *Programming Languages: Application and Interpretation*, Primera edición, Brown University, 2007.