Universidad Nacional Autónoma de México
Facultad de Ciencias
Lenguajes de Programación

Práctica 5

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: 21 de octubre de 2017
Fecha de término: 30 de octubre de 2017
Semestre 2018-1

Objetivos

- Agregar condicionales al lenguaje de la Práctica 4.
- Modificar el intérprete implementado en la Práctica 4 para que utilice evaluación perezosa mediante el uso de cerraduras de expresión (*expression closure*) y a su vez identificar los puntos estrictos del lenguaje.

Descripción general

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

```
<expr> ::= <id>
         | <num>
         | <bool>
         | {<op> <expr>+}
         | {if <expr> <expr> <expr>}
         | {cond {<expr> <expr>}+ {else <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 `CFWBAE/L`.
- `parser.rkt` archivo en el cual se debe implementar el analizador sintáctico para el lenguaje `CFWBAE/L`.
- `interp.rkt` archivo donde se debe implementar el analizador semántico para el lenguaje `CFWBAE/L`.
- `practica5.rkt` archivo con la lógica necesaria para ejecutar el intérprete final de `CFWBAE/L`.
- `test-practica5.rkt` archivo en el cual se deben agregar las pruebas unitarias de la práctica.

Desarrollo de la práctica

Ejercicio 5.1 (1 pt.) En el archivo `grammars.rkt` se encuentra el TDA que define los constructores para realizar el análisis sintáctico del lenguaje `CFWBAE/L`.

```
;; TDA para representar los árboles de sintaxis abstracta del
;; lenguaje CFWBAE/L.
;; Este TDA es una versión con azúcar sintáctica.
(define-type CFWBAE/L
   [idS (i symbol?)]
   [numS (n number?)]
   [boolS (b boolean?)]
   [opS (f procedure?) (args (listof CFWBAE/L?))]
   [ifS (expr CFWBAE/L?) (then-expr CFWBAE/L?) (else-expr CFWBAE/L?)]
   [condS (cases (listof Condition?))]
   [withS (bindings (listof binding?)) (body CFWBAE/L?)]
   [withS* (bindings (listof binding?)) (body CFWBAE/L?)]
```

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

```
    [funS (params (listof symbol?)) (body CFWBAE/L?)]
    [appS (fun-expr CFWBAE/L?) (args (listof CFWBAE/L?))])

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

;; TDA para representar condiciones.
(define-type Condition
    [condition (expr CFWBAE/L?) (then-expr CFWBAE/L?)]
    [else-cond (else-expr CFWBAE/L?)])
```

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-practica5.rkt`[2].

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

```
> (parse '{if {< 2 3} 4 5})
(ifS (opS < (list (numS 2) (numS 3))) (num 4) (num 5))
```

Ejercicio 5.2 (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 `CFWBAE/L`, es decir, las convierte en expresiones de `CFBAE/L`:

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

[2]Se deben agregar pruebas significativas.

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

Se deben agregar al menos cinco pruebas unitarias dentro del archivo `test-practica5.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 de función. Por ejemplo:

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

  se transforma en

  ```
  {{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}
  ```

- `cond` este tipo de expresiones son una versión endulzada de expresiones `if`. Por ejemplo:

  ```
  {cond
     {{< 2 3} 1}
     {{> 10 2} 2}
     {else 3}}
  ```

  se transforma en

  ```
  {if {< 2 3}
      1
      {if {> 10 2}
          2
          3}}
  ```

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

```
> (desugar (parse '{cond {{< 2 3} 1} {{> 10 2} 2} {else 3}}))
(iF
    (op < (list (num 2) (num 3))
    (num 1)
    (iF (op > (list (num 10) (num 2))) (num 2) (num 3)))
```

Ejercicio 5.3 (7 pts.) El analizador semántico de esta práctica debe realizar evaluación mediante ambientes y cerraduras para implementar alcance estático. Para implementar evaluación perezosa se agregan cerraduras de expresiones. 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 CFBAE/L-Value?) (env Env?)])
```

Los resultados devueltos por el analizador semántico, que almacena el ambiente deben ser del tipo de dato `CFBAE/L-Value`:

```
;; TDA para representar los resultados devueltos por el intérprete.
(define-type CFBAE/L-Value
   [numV (n number?)]
   [boolV (b boolean?)]
   [closureV (params (listof symbol?)) (body CFBAE/L?) (env Env?)]
   [exprV (expr CFBAE/L?) (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 `CFBAE/L`. 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 regresa o bien reporta un error, en caso de que no se haya encontrado. Ejemplo:

  ```
  > (interp (desugar (parse 'foo)) (mtSub))
  lookup: 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. En este tipo de expresiones se encuentra un punto estricto. Se deben evaluar cada uno de sus operandos para aplicar el operador mediante la definición de una función `strict`. Ejemplo:

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

- Las expresiones `if` deben evaluar su condición y regresar el valor correspondiente en caso de que la condición se evalúe a verdadero o falso, según sea el caso. En este tipo de expresiones se encuentra un punto estricto, por lo que se debe evaluar la condición para regresar el valor correcto mediante la definición de una función `strict`. Ejemplo:

  ```
  > (interp (desugar (parse '{if true true false})) (mtSub))
  (boolV #t)
  ```

- Las expresiones `cond` deben evaluar cada condición y regresar el valor correspondiente en caso de que alguna sea verdadera, en caso contrario, se deben seguir evaluando el resto de condiciones o ejecutar el caso `else` si ninguna condición fue verdadera. Al ser azúcar sintáctica de expresiones `if`, cada condición de este tipo de expresiones también es un punto estricto. Ejemplo:

  ```
  > (interp (desugar (parse '{cond {true 1} {true 2} {else 3}})) (mtSub))
  (numV 1)
  ```

- 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))
  (exprV
      (op + (list (id 'a) (id 'a)))
      (aSub 'a (exprV (num 2) (mtSub)) (mtSub)))
  ```

- 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))
  ```

- En las aplicaciones de función se encuentra un punto estricto, se debe evaluar la función para poder aplicarla con los argumentos correspondientes mediante la definición de una función `strict`. Este tipo de expresiones deben de evaluar el cuerpo de la función correspondiente considerando el ambiente donde ésta fue definida y agregando las expresiones al ambiente sin evaluar. Por ejemplo, en la

siguiente llamada, el ambiente de evaluación es:

```
(aSub 'a (exprV (op + (list (num 2) (num 3))) (mtSub)) (mtSub))
> (interp (desugar (parse '{{fun {x} {+ x 2}} {+ 2 3}})) (mtSub))
(numV 7)
```

Además, se deben agregar al menos cinco pruebas unitarias dentro del archivo `test-practica5.rkt` que correspondan con las pruebas agregadas en los ejercicios anteriores.

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

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

Ejercicio 5.4 (0 pts.) Una vez finalizados los ejercicios anteriores, ejecutar el archivo `practica5.rkt`. Realizar pruebas significativas para verificar que la práctica se completó con éxito.

```
Bienvenido a DrRacket, versión 6.10.1 [3m].
Lenguaje: plai, with debugging; memory limit: 128 MB.
(λ) {if {> 2 3} 1 2}
2
(λ) {cond {{> 2 3} 1} {{> 2 10} 2} {else 3}}
3
```

Figura 1: *Ejecución de* `CFWBAE/L`

Puntaje total: *10 puntos*

Entrega

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

- `parser.rkt`
- `interp.rkt`
- `test-practica5.rkt`

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

Referencias

Algunas referencias de consulta:

[1] Karla Ramírez, Manuel Soto, *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.