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

Fecha de inicio: 20 de marzo de 2018
Fecha de término: 6 de abril de 2018
Semestre 2018-2

Objetivo

Agregar expresiones booleanas y funciones al lenguaje implementado en la Práctica 3 y repasar los conceptos de azúcar sintáctica, alcance estático y dinámico y su implementación mediante cerraduras y ambientes de evaluación, como una optimización al algoritmo de sustitución textual.

Antecedentes

En las sesiones previas de laboratorio se revisaron las diferencias entre alcance estático y dinámico y se realizaron actividades para implementar un pequeño intérprete para una versión simplificada del lenguaje de programación `FWAE` que hace uso de cerraduras y ambientes de evaluación y que elimina azúcar sintáctica. Se recomienda revisar dichas actividades junto con las Notas 3 del curso.

Repositorio

El material necesario para completar esta práctica se encuentra en el repositorio de *GitHub Classroom* del curso: `https://classroom.github.com/g/6Ack4L_T`.

Desarrollo de la práctica

La gramática en EBNF para las expresiones del lenguaje `FWBAE` (*Functions, With, Booleans and Arithmetic Expressions)* que se implementará en esta práctica es la siguiente:

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

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

 <num> ::= ... | -1 | 0 | 1 | 2 | ...

<bool> ::= true | false

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

En equipos de **tres integrantes** se deben completar las funciones faltantes de los archivos `grammars.rkt`, `parser.rkt`, `desugar.rkt` e `interp.rkt` hasta que logren pasar todas las pruebas unitarias que se incluyen en el archivo `pruebas_practica4.rkt` y se ejecute correctamente el archivo `pratica4.rkt`[1].

Ejercicio 4.1 (1 pts) Completar el cuerpo de las siguientes funciones, del archivo `grammars.rkt`.

1. La función `(mand . args)` que realiza la conjunción multiparamétrica.

```
;; mand: any any ... -> boolean
(define (mand . args) ...)
```

```
> (mand #t #t #t #f)
#f
```

2. La función `(mor . args)` que realiza la disyunción multiparamétrica.

```
;; mor: any any ... -> boolean
(define (mor . args) ...)
```

```
> (mor #t #t #t #f)
#t
```

3. La función `(mequal? . args)` que verifica si los elementos que recibe son iguales.

[1]Para tener derecho a calificación, los archivos deben ejecutarse sin errores.

```
;; mequal?: any any ... -> boolean
(define (mequal? . args) ...)
```

```
> (mequal? 'a 'a 'a)
#t
```

4. La función `(not-equal? . args)` que verifica si los elementos que recibe son distintos.

```
;; not-equal?: any any ... -> boolean
(define (not-equal? . args) ...)
```

```
> (not-equal? 'a 'a 'a)
#f
```

Ejercicio 4.2 (2 pts.) Completar el cuerpo de la función `(parse sexp)` del archivo `parser.rkt` que realiza el análisis sintáctico correspondiente, es decir, construye expresiones del TDA `FWBAE` incluido en el archivo `grammars.rkt`.

```
;; parse: s-expression -> FWBAE
(define (parse sexp) ...)
```

```
> (parse '{+ 1 2})
(opS + (list (numS 1) (numS 2))
```

Ejercicio 4.3 (2 pts.) Completar el cuerpo de la función `(desugar expr)` del archivo `desugar.rkt` que elimina azúcar sintáctica[2] de las expresiones de `FWBAE`, es decir, las convierte en expresiones del TDA `FBAE` incluido en el archivo `grammars.rkt`.

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

```
{{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:

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

```
{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) ...)
```

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

Ejercicio 4.4 (5 pts.) Completar el cuerpo de la función `(interp expr env)` del archivo `interp.rkt` que realiza el análisis semántico correspondiente, es decir, evalúa expresiones de `FBAE`. Tomar los siguientes puntos a consideración:

- El valor de los identificadores debe ser buscado en el ambiente de evaluación mediante la función `lookup`, la cual si encuentra el identificador, entonces regresa su valor asociado o bien reporta el error `Identificador libre` en caso de que no se haya encontrado. Ejemplo:

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

- 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 realiza la operación definida a cada uno de sus operandos. Ejemplo:

```
> (interp (desugar (parse '{+ 1 2 3 4 5})) (mtSub))
(numV 15)
```

- Las expresiones `with` son multiparamétricas, lo cual quiere decir que 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 en el mismo ambiente de definiciones del `with*`. Por ejemplo `{with* {{a 2} {b {+ a a}}} b}`. Se interpreta similar a `with`, la única diferencia es que también se deben procesar cada uno de los identificadores ahí definidos. 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 ésta 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 a función deben 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)
```

```
;; interp: WAE -> number
(define (interp expr env) ...)
```

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

Referencias

Algunas referencias de consulta:

[1] Karla Ramírez, Manuel Soto, *Notas de laboratorio del curso de Lenguajes de Programación*, Semestre 2018-2, 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.