Lenguajes de Programación, 2017-2

Práctica 4: Funciones de primera clase

Manuel Soto Romero

Fecha de inicio: 17 de marzo de 2017

Fecha de término: 31 de marzo de 2017

1. Objetivos

Implementar una versión avanzada del intérprete para el lenguaje diseñado de la Práctica 3 del curso[1].

```
Bienvenido a FBAE v2.0.
(λ) 1729
1729
(λ) {with {{a 2} {b 3}} {+ a b}}
5
(λ) {with* {{a 2} {b {+ a a}}} b}
4
(λ) {{fun {x y} {pow x y}} 2 3}
8
(λ)
```

Figura 1: Intérprete en ejecución

La implementación deberá soportar identificadores, expresiones aritméticas, expresiones booleanas, asignaciones locales y funciones de primera clase. Para interpretar las aplicaciones de funciones se usará sustitución diferida mediante ambientes.

2. Antecedentes

2.1 Azúcar sintáctica

Azúcar sintáctica es cualquier aspecto de la sintaxis de un lenguaje de programación que facilita su lectura, escritura y legibilidad. El término fue acuñado por Peter Landin en 1964 para describir un lenguaje hipotético que expresara las operaciones del Cálculo Lambda, reemplazando el símbolo lambda "λ" por la palabra "`where`" [1].

Un clásico ejemplo de azúcar sintáctica son las expresiones con operadores combinados como `+=`. Por ejemplo, la expresión `x += y` es azúcar sintáctica de `x = x + y`.

2.2 Taxonomía de funciones

En el mundo de los lenguajes de programación, existe una clasificación o taxonomía que describe cómo se comportan las funciones de un determinado lenguaje [2]:

- **Funciones de primer orden**. Donde las funciones no son valores en el lenguaje.

[1] La práctica se entrega siguiendo los lineamientos especificados en la página del curso `http://lenguajesfc.com/lineamientos.html` y por equipos de tres integrantes.

- **Funciones de orden superior**. Donde las funciones pueden regresar otras funciones como valores.
- **Funciones de primera clase.** En donde las funciones son tratadas como cualquier otro valor dentro del lenguaje. Pueden pasarse como parámetro a otras funciones, pueden regresarse y almacenarse.

2.3 Sustitución diferida

El intérprete que se implementará en esta práctica debe procesar funciones de primera clase. Para llevar a cabo la evaluación de funciones, se hará uso de cerraduras (*closures*), los cuales son *una estructura de datos que contiene todo lo que la función necesita para ser aplicada* [2].

Para los propósitos de esta práctica, las cerraduras deben almacenar:

1. Los parámetros formales de la función.
2. El cuerpo de la función.
3. El ambiente donde fue definida la función.

Usando esta representación evaluaremos las aplicaciones de funciones sustituyendo el valor de los parámetros en el cuerpo de la función. Existen dos maneras de llevar a cabo la sustitución de parámetros. La primera, implementada en la Práctica 3, consiste en hacer sustitución textual, sin embargo, como se vio en clase, esta representación presenta un inconveniente (su implementación es de orden $O(n^2)$ tal que n hace referencia al número de variables a sustituir en un programa cualquiera).

La segunda manera de llevar a cabo la sustitución consiste en hacer uso de una estructura de datos llamada **ambiente**, la cual es un repositorio de sustituciones diferidas, es decir, se almacenará en esta estructura de datos cada una de las variables con su valor correspondiente. Para llevar a cabo la sustituciones, basta con buscar el identificador correspondiente en el ambiente y regresar el valor asociado.

En esta práctica se implementarán los ambientes usando una estructura de lista con algunas operaciones similares a las de una pila.

3. Archivos requeridos

Anexo a este archivo en formato PDF se encuentran los siguientes archivos, necesarios para desarrollar la práctica:

- Un archivo `grammars.rkt` que contiene los TDA necesarios para implementar el intérprete.
- Un archivo `parser.rkt` que contiene las funciones necesarias para convertir sintaxis concreta en la sintaxis abstracta correspondiente.

- Un archivo `interp.rkt` que contiene los procedimientos necesarios para evaluar las expresiones del lenguaje.
- Un archivo `practica4.rkt` que se encarga de ejecutar el intérprete final, usando todas las funciones anteriores.

4. Desarrollo de la práctica

Completar los siguientes ejercicios para lograr la correcta ejecución del archivo `practica4.rkt` que implementa un intérprete para el siguiente lenguaje descrito en notación EBNF[2]:

```
<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 | ...

<bool> ::= true | false

<op> ::= + | - | * | / | % | min | max | pow
       | neg | and | or | < | > | <= | >= | = | !=
```

Ejercicios

1. (2 pts.) Completar el cuerpo de la función `(parse sexp)` contenida en el archivo `parser.rkt` que recibe una expresión en sintaxis concreta y posteriormente construir el árbol de sintaxis abstracta del lenguaje `FWBAE`.

2. (3 pts.) Completar el cuerpo de la función `(desugar sexps)` contenida en el archivo `parser.rkt` que recibe una expresión dentro del lenguaje `FWBAE` y eliminar el azúcar sintáctica, es decir, regresar el árbol de sintaxis abstracta dentro del lenguaje `FBAE`.

   `FBAE` es una versión *desendulzada* de `FWBAE` que no cuenta con constructores para `with` ni `with*`. Para eliminar el azúcar sintáctica de este tipo de expresiones, considerar:

   - `with` puede ser expresado como una aplicación de función endulzada. Por ejemplo:

[2]Del inglés Extended Backus–Naur Form

```
{with {{a 2} {b 3}}
   {+ a b}}
```

es una versión endulzada de

```
{{fun {a b} {+ a b}} 2 3}
```

- `with*` puede ser expresado como una cadena de expresiones `with` anidadas. Por ejemplo:

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

es una versión endulzada de

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

3. (4 pts.) Completar el cuerpo de la función `(interp expr env)` contenida en el archivo `interp.rkt` que recibe un árbol de sintaxis abstracta `FBAE` y un ambiente de sustitución y a su vez regresa la interpretación del árbol como un valor de tipo `FBAE-Value`[3].

   El funcionamiento correcto del intérprete, dependerá (1) tanto del cómo se introducen las variables al ambiente al aplicar una función, (2) como de la implementación de una función `lookup` que recupere el valor de los identificadores.

5. Preguntas

(1 pt.) Contestar las siguientes preguntas dentro de un archivo `readme.txt`:

1. ¿Qué ventajas se obtuvieron de quitar el azúcar sintáctica de las expresiones `with` al momento de interpretar?
2. ¿Por qué fue necesario introducir cerraduras para evaluar expresiones en la función `interp`?
3. ¿Por qué esta versión del intérprete regresa valores de tipo `FBAE-Value`?
4. ¿Qué tipo de alcance tiene el lenguaje diseñado en esta práctica?
5. ¿Qué modificaciones tendrían que hacerse al código del procedimiento `interp` para que se usase el otro tipo de alcance?

[3]El ambiente usando una estructura de lista con comportamiento de pila está definido en el archivo `grammars.rkt`.

6. Explicar paso a paso, (incluyendo las llamadas a `parse`, `desugar` e `interp`), cómo se interpreta la siguiente expresión:

   ```
   {with* {{a 2} {b {+ a a}} {c {pow b 2}}} {% c 1729}}
   ```

6. Puntos extra

Los puntos extras son individuales y sólo se tomará en cuenta para la calificación final de la práctica, uno de estos[4].

- (1 pt.) **Teórico**

  Se entrega de forma impresa a más tardar el día 31 de marzo, durante la sesión de laboratorio.

  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 respuesta debe ser de al menos una cuartilla, recordar incluir todas las fuentes consultadas.

- (1 pt.) **Práctico**

  Se envía por correo electrónico a más tardar el 31 de marzo en un archivo con nombre `<no.cuenta>_P04.rkt` *con el asunto* `[LDP-EXP4]`*, incluir el nombre del autor en el cuerpo del mensaje.*

  En clase se vio 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 número de nodos en el árbol de sintaxis abstracta), por otro lado se analizó la alternativa de diferir la sustitución por medio de ambientes. Sin embargo utilizar un ambiente bajo una representación de pila (stack) no parece ser mucho más eficiente.

  Describir e implementar una estructura de datos para un ambiente que un intérprete pudiera usar para mejorar su complejidad. Justificar el diseño e implementación en un archivo con extensión `.pdf` por separado.

Referencias

[1] "Syntactic Sugar" en *Computer Hope: Free computer help and information* [En línea]. Accesado el 13 de marzo de 2017.

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

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

[4] Si se desean hacer los dos puntos extra ambos serán revisados, sin embargo, sólo uno será tomado en cuenta para la evaluación de esta práctica.