Lenguajes de Programación, 2017-2

Práctica 7: Sistema Verificador de Tipos

Karla Ramírez Pulido José Ricardo Rodríguez Abreu
Manuel Soto Romero

Fecha de inicio: 6 de mayo de 2017
Fecha de término: 19 de mayo de 2017

1. Objetivos

- Implementar un sistema verificador de tipos para el lenguaje diseñado en la Práctica 6 del curso. Para implementar este verificador, se agregarán tipos a la gramática del lenguaje[1].

2. 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 sistema verificador de tipos.
- 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 `verificador.rkt` que contiene los procedimientos necesarios para obtener el tipo de las expresiones del lenguaje.
- Un archivo `practica7.rkt` que se encarga de ejecutar el intérprete y verificador final, usando todas las funciones anteriores.

3. Desarrollo de la práctica

Completar los siguientes ejercicios para lograr la correcta ejecución del archivo `practica7.rkt` que implementa el verificador e intérprete para el siguiente lenguaje descrito en notación EBNF[2]:

[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.

[2] Del inglés Extended Backus–Naur Form

```
<expr> ::= <id>
         | <num>
         | <bool>
         | <list>
         | {<op> <expr>+}
         | {if <expr> <expr> <expr>}
         | {cond {<expr> <expr>}+ {else <expr>}}
         | {with {{<id> : <type> <expr>}+} <expr>}
         | {with* {{<id> : <type> <expr>}+} <expr>}
         | {rec {{<id> : <type> <expr>}+} <expr>}
         | {fun {{<id> : <type>}*} : <type> <expr>}
         | {<expr> <expr>*}

<id> := b | .. | 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

<list> ::= empty
         | {cons <expr> <list>}

<op> ::= + | - | * | / | % | min | max | pow
       | neg | and | or | < | > | <= | >= | = | != | zero?
       | head | tail | empty?

<type> ::= number
         | boolean
         | {listof <type>}
         | {<type> -> <type>}
         | a
```

Ejercicios

1. (4 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 `Typed-RCFWBAEL`.

2. (1 pt.) Completar el cuerpo de la función `(desugar sexps)` contenida en el archivo `parser.rkt` que recibe una expresión dentro del lenguaje `Typed-RCFWBAEL` y eliminar el azúcar sintáctica, es decir, regresar el árbol de sintaxis abstracta dentro del lenguaje `RCFBAEL`.

   `RCFBAEL` es una versión *desendulzada* de `Typed-RCFWBAEL` que no cuenta con constructores para `with`, `with*, cond` y no tiene tipos. Para eliminar el azúcar sintáctica de este tipo de expresiones, se debe de considerar:

- `with` puede ser expresado como una aplicación de función endulzada.
- `with*` puede ser expresado como una cadena de expresiones `with` anidadas.
- `cond` puese ser expresado como una cadena de expresiones `if` anidadas.
- Ninguna expresión presenta tipos asociados.

3. (5 pts.) Completar el cuerpo de la función `(typeof expr env)` contenida en el archivo `verificador.rkt` que recibe la sintaxis abstracta de un programa (es decir, lo que regresa la función `parse`), un ambiente de tipos y regresaráel tipo al que se evalúa el programa, en caso de no haber ningún error. Si lo hubiera, se debe informar del error del tipo encontrado.

   Las verificaciones que debe hacer esta función son:

   - El tipo de un **identificador** debe buscarse en el ambiente. Esto es equivalente a la interpretación de una expresión mediante la función `lookup`, la diferencia es que obtenemos un tipo en lugar de un valor.
   - El tipo de un **número** es `(tnumber)`.
   - El tipo de un valor **booleano** es `(tboolean)`.
   - El tipo de una **lista** dependerá del tipo de sus elementos. El verificador debe asegurarse de que todos los elementos tengan el mismo tipo en cuyo caso regresará el tipo `(listof <tipo>)` donde `<tipo>` es el tipo de los elementos, en caso contrario se debe reportar un error.
   - Para las **operaciones n-arias**, se debe verificar que todos los parámetros sean del tipo adecuado y reportar el tipo que se debe obtener después de aplicar la operación. Si algún parámetro no corresponde con el tipo necesario, reportar un error.
   - El **condicional if** consta de tres componentes, la condición, la rama a ejecutar cuando se cumple la condición, y la rama a ejecutar cuando no se cumple la condición. Se debe verificar que la condición se evalúe al tipo lógico y que el valor de sus ramas sea el mismo.
   - Para el **condicional cond**, se debe verificar lo mismo que en el condicional anterior para cada caso posible.
   - Para las **asignaciones locales with, with* y rec** se debe verificar que cada identificador se evalúe con el mismo tipo con el que fue declarado, en otro caso se debe mandar un error.
   - En el caso de las **funciones** se debe regresar el tipo `(tarrow <tipo> <tipo>)` que representa el valor de los paŕametros y el valor de regreso de la función. La información del tipo de cada parámetro debe guardarse en el ambiente por si se aplica la función. Ejemplo:

     El tipo de `{fun {{n : number} {m : number}} : number {+ n m}}` después de pasar por las funciones parse y typeof (en ese orden) debe regresar el tipo `(tarrow (tnumber) (tarrow (tnumber) (tnumber)))` que representa `(number -> number -> number)`.
   - Finalmente, para el caso de la **aplicación de función**, primero se debe verificar que el primer valor sea de tipo función, si no lo fuera entonces se debe enviar un error. Una vez que se sabe que es una función, se debe verificar que el tipo del parámetro real,

efectivamente sea el del tipo declarado por el parámetro formal en la definición de la función y que se regresa el tipo correspondiente.

- Para el caso de la **lista vacía**, se tiene que es de tipo polimórfico, pues puede tomar el tipo del resto de los elementos de la lista. En caso de que se llame a la función `type-of` con la lista vacía, regresar el tipo `(listof (a))`. Si se encuentra contenida al final de otra lista, reportar el tipo del resto de los elementos.

Nota: El archivo `interp.rkt` debe reemplazarse por el correspondiente de la Práctica 6 para la correcta ejecución del archivo `practica7.rkt`.

Referencias

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

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