Práctica 5 – Verificador de tipos estáticos

Profesora: Karla Ramírez Pulido
Ayudante teoría: Odín Miguel Escorza Soria
Ayudante laboratorio: Joshua Emmanuel Mendoza Mendieta

Fecha de entrega: **18 de noviembre de 2014**

Dado nuestro lenguaje *CFWAE* (Conditional Functions, With, Arithmetic Expression), lo extenderemos con dos tipos concretos que son *char* y *string* y con un verificador de tipos estáticos, por lo que nuestra sintaxis concreta cambiará para adecuar dicha verificación. Nuestra gramática en *EBNF* de la sintaxis es:

```
<CFWAE> ::= <num>
          | <bool>
          | <char>
          | <string>
          | <id>
          | {with {{<id> : <type> <CFWAE>}+} <CFWAE>}
          | {fun {{<id> : <type>}+} : <type> <CFWAE>}
          | {if <CFWAE> <CFWAE> <CFWAE>}
          | {equal? <CFWAE> <CFWAE>}
          | {<op> <CFWAE>}
          | {<binop> <CFWAE> <CFWAE>}
          | {<CFWAE> <CFWAE>*}
```

```
<binop> ::= +
          | -
          | *
          | /
          | <
          | >
          | <=
          | >=
          | and
          | or
          | string-append
          | equal?
```

```
<op> ::= inc
       | dec
       | zero?
       | neg
       | string-length
```

```
<type> ::= number
         | boolean
         | char
         | string
         | ([<type> ->]+ <type>)
```

Observación, los caractéres `[` y `]` de la derivación de `<type>` no son parte de la sintaxis concreta, estos son ejemplos de su uso: `(number)`, `(string -> number)`, `(number -> number -> number)` para tipos de funciones.

Evaluación.

1. (**3pts) Chars y Strings**. Implementa el parseo e interpretación en *CFWAE* y *CFWAE-Value* de los tipos concretos *char* y *string*. La sintaxis concreta para ambos tipos son `c%CHAR` y `s%STRING` respectivamente, donde `CHAR` es un valor alfabético y `STRING` son uno o más valores alfanuméricos. La distinción de valores del lenguajes tendrás que hacerla en la función *parse*. Adicional, implementa las funciones unarias y binarias que involucran dichos tipos de datos.

2. (**2pts) Sintaxis de tipos**. Adecua tu parser y tu intérprete para que puedan aceptar programas con anotaciones de tipo (todas las ocurrencias de `<type>`).

3. (**5pts) typeof** Dada una expresión *CFWAE* y un ambiente de tipos, regresar el tipo de dicha expresión. Ejemplos de su uso son:

```
> (type-of (parse '{+ 1 2}) preloaded-type-env)
(tnum)
> (type-of (parse '{and true false}) preloaded-type-env)
(tbool)
> (type-of (parse 'c%A) preloaded-type-env)
(tchr)
> (type-of (parse '{string-append s%hola s%adios}) preloaded-type-env)
(tstr)
> (type-of (parse '{fun {{x : number}} : number (inc y) }) preloaded-type-env)
(tarrow (tnum) (tnum))
> (type-of (parse '{fun {{s : string}} : number (string-length s) }) preloaded-type-env)
(tarrow (tstr) (tnum))
> (type-of (parse '{fun {{x : number} {y : number}} : number (+ x y) }) preloaded-type-env)
(tarrow (tnum) (tarrow (tnum) (tnum)))
> (type-of (parse '{{fun {{x : number} {y : number}} : number (+ x y) }
                    {+ 1 2} {string-length s%racket}}) preloaded-type-env)
(tnum)
> (type-of (parse 'string-length) preloaded-type-env)
(tarrow (tstr) (tnum))
> (type-of (parse '<) preloaded-type-env)
(tarrow (tnum) (tarrow (tnum) (tbool)))
> (type-of (parse 'equal?) preloaded-type-env)
(tarrow (tnum) (tarrow (tnum) (tbool)))
```

4. (**+2pts) Polimorfismo**. La función `equal?` es una función que puede recibir distintos tipos de valores. Por lo que decirmos que el tipo de esta función tiene una *variable de tipo*. El tipo de esta función es `(a -> a -> boolean)` donde `a` puede ser cualquier tipo, aunque para fines de esta práctica lo restringiremos a tipos terminales como `number`, `boolean`, `char` y `string`. Ejemplos de su uso son:

```
> (type-of (parse '{equal? 1 2}) preloaded-type-env)
(tbool)
> (type-of (parse '{equal? true false}) preloaded-type-env)
(tbool)
> (type-of (parse '{equal? c%A c%B}) preloaded-type-env)
(tbool)
> (type-of (parse '{equal? s%hola s%adios}) preloaded-type-env)
(tbool)
> (type-of (parse 'equal?) preloaded-type-env)
(tarrow (tvar 'a) (tarrow (tvar 'a) (tbool))
```

Notas.

- Esta práctica no requiere de la implementación de *interp*.
- El ambiente de tipos sirve para asociar un identificador con su tipo asociado, es un análogo del ambiente de expresiones del lenguaje que usamos para el *interp*.
- Para el ambiente de tipos debes cargar los tipos de las funciones predeterminadas de nuestro lenguaje. Aquí puedes consultar la tabla de tipos para cada operación:

Operación	Tipo
inc	(number -> number)
dec	(number -> number)
zero?	(number -> boolean)
neg	(boolean -> boolean)
string-length	(string -> number)
+	(number -> number -> number)
-	(number -> number -> number)
*	(number -> number -> number)
/	(number -> number -> number)
<	(number -> number -> boolean)
>	(number -> number -> boolean)
<=	(number -> number -> boolean)
>=	(number -> number -> boolean)
and	(boolean -> boolean -> boolean)
or	(boolean -> boolean -> boolean)
string-append	(string -> string -> string)
equal?	(number -> number -> boolean)

Operación	Tipo con polimorfismo
equal?	(a -> a -> boolean)