Lenguajes de Programación

Práctica 7 - Sistema verificador de tipos

Semestre 2017-1

Fecha de inicio: 7 de noviembre de 2016

Fecha de término: 18 de noviembre de 2016

Instrucciones

- Resolver los siguientes ejercicios de manera clara y ordenada.
- Se deben modificar y/o enviar los archivos `parser.rkt` y `verificador.rkt`.
- La entrega es en equipos de máximo 3 integrantes. Seguir los lineamientos especificados en: `http://sites.ciencias.unam.mx/ldp171/formato`.

Descripción general

Dado el lenguaje objetivo de la **práctica 6**, se implementará un sistema verificador de tipos estáticos, por lo que la sintaxis concreta cambiará para adecuar dicha verificación.

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

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

<op> ::= + | - | * | / | % | min | max | pow | number?
       | and | or | not | < | > | <= | >= | = | != | zero?
       | head | tail | empty? | list?
```

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

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

Ejercicios

1. **Análisis sintáctico**

   Definir una función `(parse sexp)` que consume la sintaxis concreta de un programa y regresa el árbol de sintaxis concreta correspondiente.

2. **Sistema verificador de tipos**

   Nota: Este ejercicio se resuelve en un archivo `verificador.rkt`.

   Definir una función `(type-of exp)` que recibe la sintaxis abstracta de un programa (es decir, lo que regresa la función `parse`) y regresará el 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 `typeof` son:

   - **op**
     Dependiendo de la función n-aria en cuestión, se debe verificar que los parámetros sean del tipo adecuado.
   - **iF**
     La primitiva `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 a un tipo lógico.
   - **fun**
     A diferencia de las prácticas anteriores, ahora `fun` recibe los nombre de los parámetros formales seguidos del separador "`:`" y por último el tipo que tendrán asociados además de el valor de regreso. Las información del tipo de cada parámetro debe ser guardada por si se llegase a aplicar la función.
   - **rec**
     Se debe verificar que cada parámetro se evalúe con el mismo tipo con el que fue declarado, en otro caso se debe mandar error.
   - **Aplicación de función**
     Lo primero que se debe verificar es 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 del tipo declarado por el parámetro formal en la definición de la función y que se regresa el tipo correspondiente.

Dado que los valores de regreso del verificador de tipos son los tipos en sí (o errores) y no los valores de las expresiones, se debe agregar el siguiente tipo de dato que representa los tipos de esta gramática en el archivo `grammars.rkt`.

```
(define-type Tipo
     [tnumber]
     [tboolean]
     [tlistof (t Tipo?)]
     [tarrow (a Tipo?) (b Tipo?)])
```

Ejemplos:

```
> (type-of (parse '{+ 1 3}))
(tnumber)
> (type-of (parse '{+ 1 true}))
``Error: En + se esperaban argumentos de tipo number''
> (type-of (parse '{cons 1 {cons 2 {cons 3 empty}}}))
(tlistof tnumber)
> (type-of (parse '{cons {fun {{a : number}} : number {+ a a}} {cons 2 empty}}))
``Error: Se esperaba que todos los elementos fueran de tipo (number -> number)''
> (type-of (parse '{{fun {{a : number} {b : number}} : number {+ a b}} 3 true}))
``Error: Se esperaba que el parámetro b fuera de tipo number''
```