Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Abril - Julio 2016

Fase II
Análisis Sintáctico

Esta fase incluye la definición de gramática del lenguaje y también la construcción del árbol sintáctico abstracto. A ésto es lo que denominamos el análisis sintáctico del programa.

Ud. tiene que escribir una gramática para BasicTran. Esta gramática generara un analizador que debe ser capaz de reconocer cualquier cadena válida en el lenguaje. Entonces, se le pasa como entrada tal cadena y el programa debe ser capaz de hacer una representación legible del árbol sintáctico abstracto correspondiente.

Por ejemplo, las expresiones conformadas por dos sub-expresiones y un operador binario, podría ser representado por la clase de datos BIN-EXPRESION, que pudiera tener como información las dos sub-expresiones, el operador utilizado y el tipo resultante de la expresión. También pudieran tenerse diferentes árboles de expresión binarios para los casos aritméticos, booleanos y relacionales.

En lenguajes orientados a objetos, es posible crear una clase abstracta que representa a todos los árboles de expresión y otra para los árboles de instrucción. En Haskell pueden usar esto con *clases de tipos*.

Todos los errores léxicos deben ser reportados, de encontrar algún error sintáctico, deberá reportar dicho error y abortar (al conseguir el primer error debes abortar).

Lo siguiente debe cumplirse: la gramática debe generar todas las posibles cadenas válidas y sólo/únicamente dichas cadenas/no debe generar cosas adicionales; Los nombres de los símbolos terminales y no terminales deben ser apropiados; la gramática debe ser ambigua y de izquierda, los conflictos de ambigüedad deben ser tratados con precedencia y asociatividad.

Ejemplo:

```
with
      var contador, bar : int
begin
      bar <- 42;
      if bar > 2 ->
               contadorInterno <- 35;
      end
end
```

El resultado del analizador debe devolver un resultado con un formato similar al siguiente:

```
SECUENCIACION
      ASIGNACION
                  - contenedor: VARIABLE
                  - identificador: bar
                  - expresion: LITERAL ENTERO
                  - valor: 42
      CONDICIONAL
                  - guardia: BIN_RELACIONAL
                        -operacion: ‘Mayor que’
                        - operador izquierdo: VARIABLE
                              - identificador: bar
                        - operador derecho: LITERAL ENTERO
                              - valor: 2
                  - exito: ASIGNACION
                        - contenedor: VARIABLE
                              - identificador: contadorInterno
                     - expresion: LITERAL ENTERO
                              - valor: 35
```

Pasos de lo que quiero que hagas:

1. Diseñar una gramática cuyo lenguaje generado sea BasicTran y escríbala en el lenguaje de su selección.
2. Escribir las reglas para imprimir el árbol sintáctico abstracto correspondiente a una frase reconocida.
3. Escriba un programa que lea un programa escrito en BasicTran (que puede estar mal escrito, quien sabe), y genere el árbol sintáctico correspondiente ***de ser correcto***. En el caso de que el programa sólo tenga errores léxicos, sólo deben reportarse *todos* errores encontrados. Si el programa tiene al menos un error sintáctico, debe imprimir el primer error.

Los lenguajes y herramientas permitidas:

Python	**PLY**
Ruby	**RACC**
Haskell	**HAPPY**
Java	**CUP**

Entrega de la Implementación
Ud. debe entregar un correo electrónico a flovera1@gmail.com / flovera@usb.ve que contenga:

- **El código fuente en el lenguaje y la herramienta de su elección**, entre los permitidos, de su analizador lexicográfico. Todo el código debe estar debidamente documentado. El analizador deberá ser ejecutado con el comando "./BasicTran ⟨Archivo⟩, por lo que es posible que tenga que incorporar un script a su entrega que permita que la llamada a su programa se realice de esta forma. ⟨Archivo⟩ tendrá el programa escrito en Neo que se analizará sintácticamente. Este archivo tendrá extension ''.bt'', sin embargo, no tiene que verificar esto.

- Un ARCHIVO **readme.txt** explicando la formulación/implementación de su analizador sintáctico y justificando todo aquello que Ud. considere necesario.

Fecha de Entrega: Sábado 02 de Junio (Semana 6)
Valor: 7 puntos.

Ferderico Flaviani; Fernando Lovera; Genesis Kufatty; / Mayo 19 2018