Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 – Traductores e Interpretadores
Enero – Marzo 2016

Etapa II
Análisis Sintáctico

Esta segunda etapa del proyecto corresponde al módulo de análisis sintáctico de programas escritos en `BOT`. Esto incluye la definición de la gramática y la construcción del árbol sintáctico abstracto corresponiente.

Usted deberá escribir una gramática para `BOT` que será luego pasada a una herramienta generadora de analizadores sintácticos, la cual generará un analizador que debe ser capaz de reconocer cualquier cadena válida en `BOT`. Al ser suministrada tal cadena, su programa deberá imprimir una representación legible del árbol sintáctico abstracto creado.

Esto incluye la implementación de distintas clases (Java, Ruby & Python) o tipos de datos (Haskell) que representen el significado de las expresiones e instrucciones en `BOT`. Por ejemplo, las expresiones conformadas por dos subexpresiones y un operador binario, podría ser representando por la clase/tipo de datos `BIN_EXPRESION`, que pudiera tener como información las dos subexpresiones, el operador utilizado y el tipo resultante de la expresión. También pudiera tenerse diferentes árboles de expresión binarios para los casos aritméticos, booleanos y relacionales; lo cual simplifica considerablemente el trabajo de verificación de tipos. Por otra parte, una instrucción de repetición indeterminada podría ser representada por la clase/tipo de datos `REPETICION_INDET`, que pudiera tener como información la condición de la iteración y la subinstrucción sobre la cual se está iterando.

Una posible estrategia (salvando Haskell) es la de crear una *clase abstracta* que represente a todos los árboles de expresión y otra para los árboles de instrucción de forma de aprovechar la herencia y unificar las características que sean comunes (En el caso de Haskell, puede realizarse un tratamiento similar utilizando *clases de tipos*).

De encontrar algún error sintáctico, deberá reportar dicho error y abortar la ejecución del analizador. Usted NO debe imprimir todos los errores sintácticos, solamente el primero encontrado. Sin embargo, aún deberá imprimir todos los errores léxicos.

En esta etapa, la gramática que utilice y los nombres que coloque a los símbolos *terminales* y *no terminales* presentes en la misma, son enteramente de su elección. Sin embargo, dicha gramática debe cumplir con algunas condiciones:

- Su gramática debe generar todas las posibles cadenas válidas en `BOT` y únicamente dichas cadenas (No puede generar cadenas que no pertenezcan al lenguaje).
- Los nombres escogidos para los símbolos *terminales* y *no terminales* deben ser apropiados y fácilmente reconocibles como parte del lenguaje. (Por ejemplo: $EXPR$, $EXPRESION$, $EXPRES$ son nombres apropiados para las expresiones, pero $UBUNTU$, $ROCKNROLL$ o $TRADUCTORES$ no lo son).
- Se sugiere fuertemente que diseñe su gramática manteniéndola ambigua y recursiva por la izquierda. Los conflictos existentes deberán ser tratados entonces por los mecanismos de resolución de conflictos (precedencia y asociatividad) que incluye cada herramienta.

Como ejemplo, si es suministrado el siguiente programa válido en `BOT`:

```
create
    int bot bar
        on activation:
            store 3.
        end
    end

    int bot contador
        on default:
            store 35.
        end
    end

execute
    activate contador.
    $- Asignar al contador
    el valor 35, si bar > 2 -$
    if bar > 2:
        advance contador.
    end
end
```

El resultado de la ejecución de su analizador deberá tener un formato similar al siguiente:

```
SECUENCIACION
    ACTIVACION
        - var: contador
    CONDICIONAL
        - guardia: BIN_RELACIONAL
            - operacion: 'Mayor que'
            - operador izquierdo: bar
            - operador derecho: 2
        - exito: AVANCE
            - var: contador
```

Note que la declaración y tipos de las variables (los robots) no se ve reflejado en el árbol sintáctico abstracto. Esta funcionalidad corresponde a una estructura llamada *tabla de símbolos* y será construida en la siguiente entrega. Aunque las instrucciones en las acciones de los robots no serán mostradas como salida del programa, de todas formas deben analizarse y reportar cualquier error presente en las mismas. Adicionalmente, se recomienda construir el árbol sintáctico abstracto para estas expresiones igualmente, ya que será utilizado en la próxima entrega.

Revisión Teórico-Práctica

1. Sea $G1$ la gramática $(\{ Expr \}, \{ +, NUM \}, P1, Expr)$, con $P1$ compuesto por las siguientes producciones:

   $Expr \rightarrow Expr + Expr$
   $Expr \rightarrow NUM$

(a) Muestre que la frase $NUM + NUM + NUM$ de $G1$ es ambigua.

(b) Dé una gramática no ambigua $Izq(G1)$ que genere el mismo lenguaje que $G1$ y que asocie las expresiones aritméticas generadas hacia la izquierda. Dé también una gramática $Der(G1)$ con las mismas características pero que asocie hacia la derecha.

(c) ¿Importa si se asocian las expresiones hacia la izquierda o hacia la derecha? Considere el caso del operador "$-$" o el caso del operador "$/$".

2. En `BOT` la secuenciación, o composición secuencial, es un operador binario infijo sobre instrucciones. Este operador es en realidad virtual, ya que no es representado por ninguna secuencia de símbolos en particular. Sin embargo, a efectos de esta pregunta, denotaremos la secuenciación por el símbolo "$;$". Suponga que para el manejo de este operador se decide utilizar la gramática $G2$ definida como $(\{ Instr \}, \{ ; , IS \}, P2 , Instr)$, con $P2$ compuesto por las siguientes producciones:

$$\begin{aligned} Instr &\rightarrow Instr \ ; \ Instr \\ Instr &\rightarrow IS \end{aligned}$$

Por conveniencia, momentáneamente se ignora al resto de los constructores de instrucciones compuestas de `BOT`, y se simplifica a las instrucciones simples con el símbolo terminal IS.

(a) ¿Presenta $G2$ los mismos problemas de ambigüedad que $G1$? ¿Cuáles son las únicas frases no ambiguas de $G2$?

(b) ¿Importa si la ambigüedad se resuelve con asociación hacia la izquierda o hacia la derecha?

Nota: Para entender mejor la semántica del operador de secuenciación "$;$" se recomienda leer el documento *"Un Monoide de Instrucciones"*, el cual puede encontrar aquí: monoide.pdf.

(c) Dé una derivación "más a la izquierda" (*leftmost*) y una derivación "más a la derecha" (*rightmost*) de $G2$ para la frase $IS; IS; IS$.

3. Consideremos ahora los constructores de instrucciones condicionales de `BOT`, pero ignorando momentáneamente el uso que la sintaxis de éstos hace de los "`end`" y manteniendo el operador "$;$" propuesto en la pregunta anterior.

Sea $G3$ la gramática $(\{ Instr \}, \{ ; , IF , : , ELSE , Bool , IS \}, P3 , Instr)$, con $P3$ compuesto por:

$$\begin{aligned} Instr &\rightarrow Instr \ ; \ Instr \\ Instr &\rightarrow IF \ Bool : Instr \\ Instr &\rightarrow IF \ Bool : Instr \ ELSE : Instr \\ Instr &\rightarrow IS \end{aligned}$$

(a) Note que $G3$ mantiene el mismo problema de $G2$, i.e. frases con más de una ocurrencia de "$;$" son ambiguas. Más aún, $G3$ tiene otros problemas pues frases con ninguna o sólo una ocurrencia de "$;$" también pueden ser ambiguas. Sea f la frase "$IF \ Bool : IS \ ; IS$". Muestre que f es una frase ambigua de $G3$.

(b) Dé una frase g de $G3$ sin ocurrencias de "$;$" que sea ambigua, y muestre que lo es.

(c) En lenguajes como Java, C y C++ se hace uso de los símbolos "{" y "}" ("*begin*" y "*end*" en Pascal) para diferenciar las dos posibles interpretaciones de la frase f. Lo mismo ocurre con la frase g. ¿Cómo se escribirían las dos interpretaciones de f y las dos interpretaciones de g usando las llaves como separadores?

(d) En `BOT`, que utiliza los terminadores "`end`" en la sintaxis de sus condicionales, ¿cómo se escribirían las dos interpretaciones de f y las dos interpretaciones de g?

Implementación En esta etapa Ud. deberá desarrollar el analizador sintáctico de `BOT` como se indica a continuación:

1. Diseñe una gramática cuyo lenguaje generado sea `BOT` y escriba la misma en el lenguaje de especificación de la herramienta ofrecida para el lenguaje de su elección.

2. Escriba las reglas para construir e imprimir el árbol sintáctico abstracto correspondiente a una frase reconocida.

3. Escriba un programa que lea un programa escrito en `BOT` (potencialmente incorrecto) y genere el árbol sintáctico correspondiente, de ser correcto. Si el programa no es correcto por razones puramente léxicas, debe imprimir únicamente todos los errores encontrados. Si el programa tiene al menos un error sintáctico, debe imprimir únicamente el primero de tales errores.

Entrega de la Implementación Ud. debe entregar un correo electrónico 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 "`./SintBot` ⟨*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 `BOT` que se analizará sintácticamente. Este archivo tendrá extensión ‘‘`.bot`’’, sin embargo no tiene que verificar esto.

- Un <u>*breve*</u> informe explicando la formulación/implementación de su analizador sintáctico y justificando todo aquello que Ud. considere necesario. Además el informe debe contener sus respuestas a la revisión teórico–práctica.

Nota: Es importante que su código pueda ejecutarse en las máquinas del LDC, pues es ahí y únicamente ahí donde se realizará su corrección.

Fecha de Entrega: Viernes, 12 de Febrero (Semana 8), hasta las 11:59 pm.
Valor: 7%.

S. Ancelmo, C. Ferreira y R. Monascal / Enero 2016