Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
CI3725 - Traductores e Interpretadores
Abril - Julio 2012

# Etapa II
# Análisis Sintáctico

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

Usted deberá escribir una gramática para AsGArD 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 AsGArD. 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 AsGArD. 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, relacionales y de canvas; ahorrando así, particularmente en el caso de AsGArD, la necesidad de calcular el tipo resultante de la expresión en la mayoría de los casos. Sin embargo, el caso particular de los identificadores es problemático, por lo que se descarta esta alternativa. 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.

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 ser ambigua y recursiva por la izquierda. Los conflictos deberán ser tratados entonces por los mecanismos de resolución de conflictos que incluye cada herramienta.
- Su gramática debe generar todas las posibles cadenas válidas en AsGArD 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).

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

```
using foo, bar of type integer begin
  bar := 3 ;
  {- Asignar a foo
  el valor 35 + bar, si bar > 2. -}
  if bar > 2 then
    foo := 35 + bar
  done
end
```

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

```
SECUENCIACION
    ASIGNACION
        - var: VAR bar
        - val: INT_LITERAL 3
    CONDICIONAL
        - guardia: BIN_EXPRESION
            - operacion: 'Mayor que'
            - tipo: boolean
            - operando izquierdo: VAR bar
            - operando derecho: INT_LITERAL 2
        - exito: ASIGNACION
            - var: foo
            - val: BIN_EXPRESION
                - operacion: 'Suma'
                - tipo: integer
                - operando izquierdo: INT_lITERAL 35
                - operando derecho: VAR bar
```

Note que la declaración y tipos de las variables 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.

Para esta entrega Ud. no deberá calcular el tipo de las expresiones, pues sin el tipo de las variables tal tarea no es posible. La misma se pospondrá para la tercera fase del proyecto.

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

1. Diseñe una gramática cuyo lenguaje generado sea AsGArD 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 AsGArD (potencialmente incorrecto) y genere el árbol sintáctico correspondiente de ser correcto. Si el programa no es correcto por razones puramente léxicas, debe imprimir todos los errores encontrados. Si el programa tiene al menos un error sintáctico, debe imprimir únicamente el primero de tales errores.

**Lenguajes y Herramientas Permitidos** Para la implementación de este proyecto se cuenta para escoger con los mismos cinco (5) lenguajes de programación, de la entrega anterior. A continuación las herramientas generadoras de analizadores sintácticos para cada uno de ellos:

- *C++:* La herramienta generadora de analizadores sintácticos a utilizar se llama *Bison* y puede ser encontrada en la siguiente dirección Web: (http://www.gnu.org/software/bison/).
- *Java:* La herramienta generadora de analizadores sintácticos a utilizar se llama *CUP* y puede ser encontrada en la siguiente dirección Web: (http://www2.cs.tum.edu/projects/cup/).
- *Ruby:* La herramienta generadora de analizadores sintácticos a utilizar se llama *RACC* y puede ser encontrada en la siguiente dirección Web: (http://i.loveruby.net/en/projects/racc/).
- *Python:* La herramienta generadora de analizadores sintácticos a utilizar es a la vez la misma con la que generaron sus analizadores lexicográficos. Esta herramienta se llama *PLY* y puede ser encontrada desde la siguiente dirección Web: (http://www.dabeaz.com/ply/).
- *Haskell:* La herramienta generadora de analizadores sintácticos a utilizar se llama *Happy* y puede ser encontrada en la siguiente dirección Web: (http://www.haskell.org/happy/).

**Entrega** Ud. debe entregar:

- El código fuente en el lenguaje y la herramienta de su elección, entre los permitidos, de su analizador sintáctico. Todo el código debe estar debidamente documentado. El analizador deberá ser ejecutado con el comando "`./SintAsgard`", 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. Note que la entrada para su programa será a través de la entrada estándar del sistema de operación.
- Un <u>breve</u> informe explicando la formulación/implementación de su analizador sintáctico, los problemas encontrados y sus soluciones, detallando adicionalmente todo aquello que Ud. considere necesario. Además el informe debe contener sus respuestas a la revisión teórico–práctica, la cual será explicada más adelante.

**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 AsGArD la secuenciación, o composición secuencial, es un operador binario infijo sobre instrucciones denotado 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:

   *Instr* → *Instr* ; *Instr*
   *Instr* → *IS*

   Por conveniencia, momentáneamente se ignora al resto de los constructores de instrucciones compuestas de AsGArD, 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"* que se encuentra en el directorio `http://www.ldc.usb.ve/~jravelo/docencia/discretas/discrII02abr/`, en los archivos `monoide.ps` y `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* .

3. Consideremos ahora los constructores de instrucciones condicionales de AsGArD, pero ignorando momentáneamente el uso que la sintaxis de éstos hace de los "`done`". Sea *G3* la gramática ( { *Instr* } , { ; , *IF* , *THEN* , *ELSE* , *Bool* , *IS* } , *P3* , *Instr* ) , con *P3* compuesto por:

   *Instr* → *Instr* ; *Instr*
   *Instr* → *IF Bool THEN Instr*
   *Instr* → *IF Bool THEN Instr ELSE Instr*
   *Instr* → *IS*

   (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 THEN 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 los corchetes como separadores?

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

4. Se decidió agrupar todas las expresiones binarias bajo una sola categoría: `BIN_EXPRESION`, dejando el análisis de tipos como parte del análisis de contexto que se realizará a posteriori. La razón de no dividir las expresiones por su tipo, es que el caso de los identificadores sería problemático. Explique brevemente por qué dicho caso es problemático.

**Fecha de Entrega:** Domingo, 10 de Junio (Antes de semana 7), hasta las 11:00 pm.
**Dirección de Entrega:** Deberá entregar su proyecto tanto por Aula Virtual como por medio de correo electónico a los cuatro encargados del curso.
**Formato de Entrega:** Deberá entregar un archivo comprimido con el nombre `"E2GX.tar.gz"` (sin las comillas), donde deben sustituir `X` por el número de su grupo en el Aula Virtual. No deben crear carpetas adicionales, el proyecto deberá ser subido en el directorio raíz de la página de cada grupo. El correo que han de enviar como respaldo, a los cuatro encargados del curso, deberá tener como asunto `"[CI3725] Entrega 2"` (sin las comillas).
**Valor:** 9%.

---

C. Perez. R. Monascal, C. Gómez, M. Gómez, H. González y J. A. Goncalves / Abril 2012