Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Traductores e Interpretadores
Trimestre Abril-Julio 2025

Etapa II: Análisis Sintáctico con Árbol Sintáctico Abstracto (AST)

En esta entrega usted deberá diseñar una gramática libre de contexto para el lenguaje imperativo del proyecto y utilizarla para implementar un reconocedor. Su reconocedor debe además construir el árbol sintáctico abstracto del programa reconocido e imprimirlo de forma legible por la pantalla.

Su gramática debe ser lo suficientemente completa tal que pueda reconocer cada instrucción y expresión del lenguaje, y combinarlos para hacer programas. Note que su gramática solo debe chequear la forma de la entrada, pero no su significado; es decir, no es necesario que detecte problemas de tipos ni de variables no declaradas.

Para construir el árbol sintáctico debe implementar las clases necesarias para representar las instrucciones y expresiones del lenguaje.

1. Formato

Su programa principal deberá llamarse `parse.py` y recibirá como único argumento el nombre del archivo a analizar. La ejecución del mismo mostrará en pantalla la representación del árbol de sintaxis (descrita más adelante).

2. Errores

Si el analizador lexicográfico consigue errores, deben reportarse todos y abortar la ejecución de la misma manera que se realizó en la primera entrega.

Basta que su analizador sintáctico aborte ante el primer error de sintáxis. Si bien las herramientas en uso le permitirían implantar una recuperación de errores sintácticos completa, no es el objetivo de este curso, así que no invierta tiempo en ello. Sin embargo, al encontrar el primer error sintáctico, el reconocedor debe abortar la ejecución indicando la línea y columna del error, así como el token que causó el problema. Por ejemplo

Suponiendo que el contenido del archivo `programa.imperat` es:

```
{
    a := ;
    b := ;
}
```

Al pasar este archivo por su programa debe obtenerse un resultado similar a:

```
$ Sintax error in row 2, column 10: unexpected token ';'.
```

3. Impresión de AST

Para la impresión del árbol sintáctico considere el siguiente ejemplo de código:

```
{
     int a, b, i;
     function[..3] x, y;

  // iterar entre a y b
  a := -1;
  b := 4;
  x := x(0:a)(1:b);
  i:= a;
  while i <= b-a -->
      print "Variable \"i\" es igual a: " + i;
      i:=i+1
  end;

  // iterar sobre la segunda funcion
  y := 1, -1, 2, -2;
  i:=0;
  while i <= 3 -->
      print i + ":" + y . i + " ";
      i:=i+1
  end;

  a := 3 + b;
  b := -4;

  if 2 <= b and b <= 5 and true--> print b
  [] y.2 < b and b < y.2 --> print a
  [] y.3 >= b or b >= y.3 -->
     {
        function[..1] z;
        print a + b;
        z := z(0:a)(1:b);
        print "function[" + z.2 + ".." + z.3 + "]"
     }
  fi;

  i := 3;
  while i < 10 -->
      print "Still here!";
      i := i+1
  end
}
```

El resultado del analizador al leer el programa anterior, debe devolver impreso el árbol AST con el siguiente formato

```
Block
```

```
-Declare
--Sequencing
---a, b, i : int
---x, y : function[..Literal: 3]
-Sequencing
--Sequencing
---Sequencing
----Sequencing
-----Sequencing
------Sequencing
-------Sequencing
--------Sequencing
---------Sequencing
----------Sequencing
-----------Sequencing
------------Sequencing
------------Asig
--------------Ident: a
--------------Minus
--------------Literal: 1
------------Asig
--------------Ident: b
--------------Literal: 4
-----------Asig
------------Ident: x
------------WriteFunction
--------------WriteFunction
--------------Ident: x
--------------TwoPoints
----------------Literal: 0
----------------Ident: a
--------------TwoPoints
---------------Literal: 1
---------------Ident: b
----------Asig
-----------Ident: i
-----------Ident: a
----------While
----------Then
-----------Leq
------------Ident: i
------------Minus
--------------Ident: b
--------------Ident: a
-----------Sequencing
------------Print
--------------Plus
---------------String: "Variable \"i\" es igual a: "
---------------Ident: i
------------Asig
--------------Ident: i
--------------Plus
```

```
---------------Ident: i
---------------Literal: 1
---------Asig
----------Ident: y
----------Comma
-----------Comma
------------Comma
-------------Literal: 1
-------------Minus
--------------Literal: 1
------------Literal: 2
-----------Minus
------------Literal: 2
--------Asig
---------Ident: i
---------Literal: 0
-------While
--------Then
---------Leq
----------Ident: i
----------Literal: 3
---------Sequencing
----------Print
-----------Plus
------------Plus
-------------Plus
--------------Ident: i
--------------String: ":"
-------------App
--------------Ident: y
--------------Ident: i
------------String: " "
----------Asig
-----------Ident: i
-----------Plus
------------Ident: i
------------Literal: 1
------Asig
-------Ident: a
-------Plus
--------Literal: 3
--------Ident: b
-----Asig
------Ident: b
------Minus
-------Literal: 4
----If
-----Guard
------Guard
-------Then
--------And
---------And
```

```
----------Leq
----------Literal: 2
----------Ident: b
----------Leq
----------Ident: b
----------Literal: 5
---------Literal: true
--------Print
---------Ident: b
-------Then
--------And
---------Less
----------App
----------Ident: y
----------Literal: 2
----------Ident: b
---------Less
----------Ident: b
----------App
----------Ident: y
----------Literal: 2
--------Print
---------Ident: a
------Then
-------Or
--------Geq
---------App
----------Ident: y
----------Literal: 3
---------Ident: b
--------Geq
---------Ident: b
---------App
----------Ident: y
----------Literal: 3
-------Block
--------Declare
---------z : function[..Literal: 1]
--------Sequencing
---------Sequencing
----------Print
-----------Plus
------------Ident: a
------------Ident: b
----------Asig
-----------Ident: z
-----------WriteFunction
------------WriteFunction
-------------Ident: z
-------------TwoPoints
--------------Literal: 0
--------------Ident: a
```

```
------------TwoPoints
-------------Literal: 1
-------------Ident: b
---------Print
----------Plus
-----------Plus
------------Plus
-------------Plus
--------------String: "function["
--------------App
---------------Ident: z
---------------Literal: 2
-------------String: ".."
------------App
-------------Ident: z
-------------Literal: 3
-----------String: "]"
---Asig
----Ident: i
----Literal: 3
--While
---Then
----Less
-----Ident: i
-----Literal: 10
----Sequencing
-----Print
------String: "Still here!"
-----Asig
------Ident: i
------Plus
-------Ident: i
-------Literal: 1
```

4. Lenguajes y Herramientas a usar

Para la implementación de este proyecto se permitirá el uso de las siguientes herramientas. Estas son:

- Python:
  - Interpretador *python*
  - Generador de analizadores lexicográficos y sintácticos PLY

5. Entrega de la Implementación

La entrega del proyecto es el domingo de la semana 7 al correo electrónico de su profesor. Su entrega debe incluir lo siguiente:

- Un archivo llamado `gramatica.txt` que contega la gramática propuesta por usted para reconocer GCL. Esta gramática debe coincidir con la implementada por su reconocedor y debe seguir el siguiente formato

```
S -> A B C
   | A B

A -> C
...
```

  indicando cual es el símbolo inicial de la gramática.

- Un archivo comprimido `tar.gz` con el código fuente de su proyecto, debidamente documentado con el archivo `gramatica.txt`. El nombre del archivo debe ser **Etapa2-XX-YY.tar.gz** donde **XX-YY** son los carné de los integrantes del grupo.

  El no cumplimiento de los requerimientos podría resultar en el rechazo de su entrega.

Federico Flaviani / fflaviani@gmail.com / Mayo 2025