Universidad Simón Bolívar
Departamento de Computación y Tecnología de la Información
Traductores e Interpretadores
Trimestre Septiembre-Diciembre 2019
Diego Peña 15-11095
Manuel Gil 14-10397

# Informe
# Etapa 2: Análisis Sintáctico

Esta segunda etapa del proyecto corresponde al módulo de análisis sintáctico de programas
escritos en GuardedUSB.

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

## Archivos y ejecución del programa

En el presente se entregaran el archivo principal del programa “entrega2.py”, el archivo de gramática y del arbol ATS (ATS.py). Para ejecutar el programa es necesario tener la librería PLY y Python3. El comando a ejecutar será: “Python3 entrega2.py <InputArchivo>”, donde <InputArchivo> corresponde al archivo que tiene el programa en formato .gusb

## Formulación del Análisis Sintáctico

En esta fase nos centraremos en construir el parser del programa, dado que ya tenemos el lexer para reconocer los tokens, ahora con el parser definiremos una serie de reglas para construir nuestra gramática, y dado un programa en .gusb en especifico, crearemos el Árbol Sintáctico correspondiente. En el momento de definir las reglas buscamos que no existan ambigüedades, y una posible estrategia para evitar shift / reduce, es definir precedencias entre los tokens.

Cuando el programa en .gusb tiene un error, el programa imprimirá “Syntax error in input” y devolverá donde sucedió dicho error, luego terminará el programa

## Implementación del Análisis Sintáctico

### Árbol Sintáctico (ATS)

Creamos un objeto que será nuestro Árbol Sintáctico, en el cual tiene como atributos la categoría, el valor y los nodos hijos que posee. Además tiene una función llamado “PrintTree” el cual mediante una llamada recursiva imprimirá dicho árbol

### Reglas de la Gramática

En Python definimos unas clases que nos servirá para tener las reglas y así crear nuestra gramática.

Comenzamos con la regla de **ProgramBlock** el cual consiste en reconocer el token |[, luego el token TkDeclare, y dos reglas: DeclareLines e Instructions, finalmente cerramos con el token |]. La logica es: |[ TkDeclare **DeclareLines Instructions** ]|. También podemos tener que un ProgramBlock el cual solo tendra instrucciones y no declaración de variables.

En **DeclareLines** vamos a declarar las variables, para ello puede ser una línea con varias **VarDeclaration** o una sola. En VarDeclaration tenemos dos opciones: **MultipleTypeDeclaration** el cual consiste en para dado las variables a, b, c… a cada una le corresponderá un tipo de dato que pueden ser diferentes, por ejemplo: a, b, c… : int, bool, int… (La cantidad de variables debe corresponder con la misma cantidad de tipos de datos). Luego la siguiente opción es **SingleTypeDeclaration** el cual consiste en dado una lista de variables a, b, c… todas tendrán asignada un tipo de dato en común

Luego en **Instructions** podemos tener una instrucción o una secuencia de instrucciones (**InstSequence**), cada secuencia está separada por un punto y coma. Y en cada línea de instrucción (**InstructionLine**) tenemos como opciones: Asignar una expresión a una varialbe (**Asig**), Read, IfDo, Println, Print y For.

**Read** es la regla cuando se identifica el token TkRead y el TkId que se quiere leer.

**IfDo** es la regla en donde entre los tokens TkIf y Tkfi habrá un cuepro (**Body**, otra regla) el cual dado una expresión (ExpAux), hay unas instrucciones (Instructions), pero también existe una lista de guardias (**GuardList**) el cual nos permite crear las posibles guardias correspondientes (o crear más casos para el mismo If)

**Println** y **Print** tienen una lógica parecida, cuando reconoce el token de TkPrintln o TkPrint puede suceder como caso base que tengan una Expresion o un String, pero también existe la posibilidad de crear todas las posibles combinaciones entre expresiones y string's gracias a la regla de **Concat**

La regla de **For** se trata que entre los tokens de TkFor y TkRof, se ejecutará un ProgramBlock, pero antes se define una regla **In**, el cual consiste en dado una expresión in en otra expresión hasta (to) expresión, nos dará la cantidad de iteraciones que tiene que hacer el For

Las expresiones que viene siendo definida por **ExpAux** puede ser un valor en especifico, o expresiones con algún operador (igual, no igual, mayor que, menor que, disyunción, conjunción, suma, resta, multiplicación, división o operaciones de arreglos). En la regla de **Value** existe la posibilidad que sea una función que pueden ser las funciones: atoi, max, size y min