# Sintaxis y Semántica de los Lenguajes

*Trabajo Práctico N°2: Parser*

FRD.UTN

**Carrera:** Ingeniería en Sistemas de Información

**Docentes:**
- Miranda Hernán
- Santos Juan Miguel

**Año:** 2022

**Alumnos:**
- Dominguez Tomás
- Gonzalez Tomás Nahuel
- Pastor Hanna Sofia
- Zaracho Julieta Mariel

## *Introducción*

En este trabajo se realizó un analizador sintáctico (Parser) descendente predictivo con tabla, este se encarga de recibir los tokens generados por el Lexer a partir de una cierta cadena de entrada (el código fuente) y, apoyándose en la tabla de derivaciones, verificar si dicha cadena es o no aceptada por la gramática. Este tipo de analizadores sintácticos es mucho más eficiente que aquellos que no son predictivos, como por ejemplo aquellos que utilizan backtracking. El algoritmo con el que trabajamos nos permite obtener tanto si la cadena ingresada pertenece al lenguaje como las derivaciones utilizadas para obtenerla desde el símbolo distinguido “en el primer intento” es decir sin probar combinaciones hasta que una coincida.

## *Gramática Asignada*

```
VT = {eq, id, num, *, +, op, clp, si, entonces, sino, mostrar,
aceptar, mientras, esMenorQue, hacer, (, )}
VN = {Programa, Asignacion, Estructura, Expresion, Valor,
ListaExpresiones, Termino, Factor}
S = Program
P = {

Program → Estructura Program
         | λ

Estructura →“mientras” “id” “esMenorQue” Valor “hacer” “op“ Program
        “clp”
            | “si” Expresion “entonces” “op“ Program “clp” “sino”
        “op“ Program “clp”
            | “mostrar” Expresion
            | “aceptar” “id”
            | “id” “eq” Expresion

Valor → “id”
        |”num”

Expresion → Termino Expresion2
Expresion2 → “+” Termino Expresion2
           | λ

Termino → Factor Termino2
Termino2 → “*” Factor Termino2
          | λ

Factor → “(“  Expresion  “)”
        | Valor

}
```

# *Explicaciones y Observaciones del Trabajo Realizado*

## *Cálculo de símbolos directrices*

Antes que nada, para poder utilizar un analizador sintáctico predictivo por tabla necesitamos contar con una gramática **LL(1)**, esto se puede verificar calculando los símbolos directrices de cada una de sus producciones y verificando que la intersección de los conjuntos pertenecientes a una derivación del mismo no terminal sea igual a vacío.
A continuación se encuentran los cálculos:

$$SD(Program \ -> \ Estructura\ Program) \ =$$
$$= \ Primeros(Estructura\ Program) \ =$$
$$= \{ mientras,\ si,\ mostrar,\ aceptar,\ id \}$$
$$SD(Program \ -> \ \lambda) =$$
$$= \ Primeros(\lambda) \ \cup \ Siguientes(Program) \ =$$
$$= \ \{ clp \}$$

} Intersección = Ø

$$SD(Estructura \ -> \ mientras\ id\ esMenorQue\ Valor\ hacer\ op\ Program\ clp) \ =$$
$$= \ Primeros(mientras\ id\ esMenorQue\ Valor\ hacer\ op\ Program\ clp) \ =$$
$$= \{ mientras \}$$
$$SD(Estructura \ -> \ si\ Expresion\ entonces\ op\ Program\ clp\ sino\ op\ Program\ clp) \ =$$
$$= \ Primeros(si\ Expresion\ entonces\ op\ Program\ clp\ sino\ op\ Program\ clp) \ =$$
$$= \ \{ si \}$$
$$SD(Estructura \ -> \ aceptar\ id) \ =$$
$$= \ Primeros(aceptar\ id) =$$
$$= \{ aceptar \}$$
$$SD(Estructura \ -> \ id\ eq\ Expresion) \ =$$
$$= \ Primeros(id\ eq\ Expresion) =$$
$$= \{ id \}$$

} Intersección = Ø

$$SD(Valor \ -> \ id) \ =$$
$$= \ Primeros(id) =$$
$$= \{id\}$$
$$SD(Valor \ -> \ num) \ =$$
$$= \ Primeros(num) =$$
$$= \{ num \}$$

} Intersección = Ø

$$SD(Expresion \ -> \ Termino\ Expresion2) \ =$$
$$= \ Primeros(Termino\ Expresion2) =$$
$$= \{ (,\ id,\ num \}$$

$SD(Expresion2 \rightarrow \ + Termino\ Expresion2) =$
$= Primeros(+\ Termino\ Expresion2) =$
$= \{+\}$
$SD(Expresion2 \rightarrow \lambda) =$
$= Primeros(\lambda) \cup Siguientes(Expresion2) =$
$= \{mientras, si, mostrar, aceptar, id, clp, entonces, )\}$

} Intersección = Ø

$SD(Termino \rightarrow Factor\ Termino2) =$
$= Primeros(Factor\ Termino2) =$
$= \{ (, id, num \}$

$SD(Termino2 \rightarrow \ * Factor\ Termino2) =$
$= Primeros(*\ Factor\ Termino2) =$
$= \{ \ * \ \}$
$SD(Termino2 \rightarrow \lambda) =$
$= Primeros(\lambda) \cup Siguientes(Termino2) =$
$= \{mientras.\ si, mostrar, aceptar, id, clp, entonces, ), \ +\}$

} Intersección = Ø

$SD(Factor \rightarrow (Expresion)\} =$
$= Primeros(\ (Expresion)\ ) =$
$= \{ (\ \}$
$SD(Factor \rightarrow Valor\} =$
$= Primeros(Valor) =$
$= \{ id, num \}$

} Intersección = Ø

Con esto verificamos que la gramática es de hecho LL(1) y por lo tanto podemos proceder a la realización del analizador sintáctico predictivo por tabla.

*Explicación del Algoritmo (Archivo parser_TP2.py)*

Como ya sabemos la función del parser es la de analizar los tokens generados por el lexer a partir de una cadena en específico para verificar si esta pertenece a la gramática o no, es decir, si puede ser derivada desde el símbolo distinguido.

Antes de comenzar con la función principal, es necesario convertir la salida del lexer a una entrada que este pueda comprender, aquí es donde entra en juego la función ***'traduccionParser'*** esta convierte la lista de tuplas que sale del lexer a una lista compuesta de solamente el primer elemento de cada tupla y le añade el símbolo de fin de cadena (#) al final para que el parser pueda funcionar correctamente..

Una vez solucionado el tema de la conexión entre el lexer y el parser toca definir el elemento más importante para un analizador sintáctico predictivo por tabla, **la tabla**, en nuestro caso elegimos realizarla utilizando diccionarios, como estos son solo unidimensionales implementamos un diccionario de diccionarios para poder obtener una estructura de datos similar a una matriz pero con la capacidad de referenciar los elementos utilizando los nombres de los no terminales y terminales.

Con todos estos elementos implementados ahora podemos realizar el algoritmo principal:

**Algoritmo de análisis sintáctico descendente predictivo con tabla.**

Repetir
Si *tope* $\in V_T$
    si *tope* == t
        extraer el tope de la pila
        avanzar 1 en w
    sino
        error.
sino
    si M( t, *tope* ) == *tope* $\rightarrow Y_1\ Y_2\ ..\ Y_k$
        Extraer *tope* de la pila
        Poner $Y_k, Y_{k-1}, \ldots, Y_1$ en la pila.
    sino
        error.
hasta que tope == "#" $\wedge$ t == "#"

De este algoritmo surge la función **‘parser’** que es una adaptación a python del algoritmo mostrado arriba, con la única salvedad de que nuestra implementación además se encarga de guardar las derivaciones utilizadas para llegar a la cadena ingresada.

Las derivaciones se obtienen mediante la función **‘generarDerivacion’** esta función recibe el tope de la pila, la producción a derivar y la derivación que se va a utilizar y devuelve como resultado la producción una vez derivada.

Para extraer el tope de la pila utilizamos la función **‘.pop()’** que elimina el último elemento de una lista dada.

A la hora de verificar si $M(t, tope) == tope \ -> \ Y_1 Y_2 ... Y_k$ utilizamos un bloque de **‘try’** y **‘except’** que directamente intenta asignarle a la variable **‘produccionTabla’** el elemento de la tabla sin verificar antes si este existe o no, si el valor existe, la ejecución continuará sin problemas, en caso de que no exista obtendremos un error y se pasará a la ejecución de las líneas de código en el bloque **‘except’** donde modificaremos la variable **‘continuar’** para detener la ejecución puesto que la cadena no pertenece a la gramática.

## *Cadenas de Prueba*

1. *“mientras var esMenorQue 69 hacer op clp”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['mientras', 'id', 'esMenorQue', 'Valor', 'hacer', 'op', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'clp']

2. *“mostrar var”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['mostrar', 'Expresion', 'Program']
   ['mostrar', 'Termino', 'Expresion2', 'Program']
   ['mostrar', 'Factor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'Valor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', 'Expresion2', 'Program']
   ['mostrar', 'id', 'Program']
   ['mostrar', 'id']

3. *“si 2 entonces op clp sino op clp”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['si', 'Expresion', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'Termino', 'Expresion2', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'Factor', 'Termino2', 'Expresion2', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'Valor', 'Termino2', 'Expresion2', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'num', 'Termino2', 'Expresion2', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'num', 'Expresion2', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'num', 'entonces', 'op', 'Program', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'num', 'entonces', 'op', 'clp', 'sino', 'op', 'Program', 'clp', 'Program']
   ['si', 'num', 'entonces', 'op', 'clp', 'sino', 'op', 'clp', 'Program']
   ['si', 'num', 'entonces', 'op', 'clp', 'sino', 'op', 'clp']

4. *“var1 eq var2”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['id', 'eq', 'Expresion', 'Program']
   ['id', 'eq', 'Termino', 'Expresion2', 'Program']
   ['id', 'eq', 'Factor', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'Valor', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'id', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'id', 'Expresion2', 'Program']
   ['id', 'eq', 'id', 'Program']
   ['id', 'eq', 'id']

5. *"mientras 69 esMenorQue (contador + 45)"*
   Obtenemos que es una **cadena no aceptada** por la gramática.

6. *“mientras x esMenorQue 4 hacer op aceptar var clp”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['mientras', 'id', 'esMenorQue', 'Valor', 'hacer', 'op', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'Estructura', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'aceptar', 'id', 'Program', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'aceptar', 'id', 'clp', 'Program']
   ['mientras', 'id', 'esMenorQue', 'num', 'hacer', 'op', 'aceptar', 'id', 'clp']

7. *“aceptar var”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['aceptar', 'id', 'Program']
   ['aceptar', 'id']

8. *“ x  eq 69”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['id', 'eq', 'Expresion', 'Program']
   ['id', 'eq', 'Termino', 'Expresion2', 'Program']
   ['id', 'eq', 'Factor', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'Valor', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'num', 'Termino2', 'Expresion2', 'Program']
   ['id', 'eq', 'num', 'Expresion2', 'Program']
   ['id', 'eq', 'num', 'Program']
   ['id', 'eq', 'num']

9. *“mostrar x * 9”*
   Obtenemos que es una **cadena aceptada** por la gramática con derivaciones:

   ['Estructura', 'Program']
   ['mostrar', 'Expresion', 'Program']
   ['mostrar', 'Termino', 'Expresion2', 'Program']
   ['mostrar', 'Factor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'Valor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', '*', 'Factor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', '*', 'Valor', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', '*', 'num', 'Termino2', 'Expresion2', 'Program']
   ['mostrar', 'id', '*', 'num', 'Expresion2', 'Program']
   ['mostrar', 'id', '*', 'num', 'Program']
   ['mostrar', 'id', '*', 'num']

10. *“Si (num * id) * id + num entonces op aceptar var clp sino op mostrar var clp”*
    Obtenemos una **cadena no aceptada** por la gramática.