



## **Análisis Sintáctico**

Cázares Rodríguez Víctor Manuel

Limones Moscoso Ulises

Ojeda Ávila Diego Antonio

Rivera Arellanes Josué David

Grupo 2

Compiladores

Semestre 2020-2

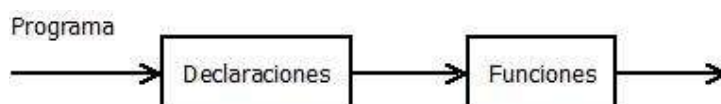
## Análisis del problema

Ahora se debe diseñar el analizador sintáctico para la gramática presentada en el problema. El primer paso a realizar es declarar todos los tokens que devuelve el análisis léxico dentro de nuestro analizador sintáctico, y se pone su asociatividad correspondiente para los que la necesitan. Tomando como base la gramática se deben ir declarando los símbolos dentro del programa, donde el símbolo inicial será el programa, como está indicado en la gramática.

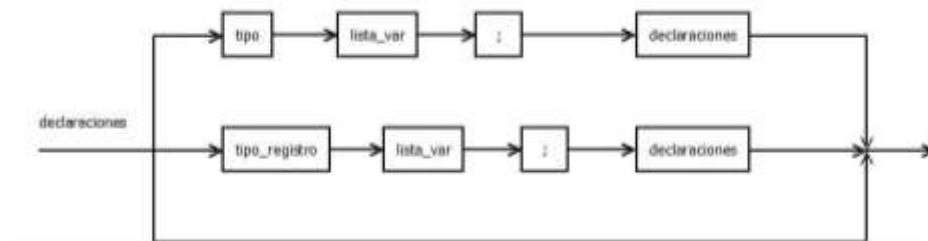
## Diseño de la solución

Revisando producciones para quitar ambigüedad, eliminar recursividad y factorizar de ser necesario.

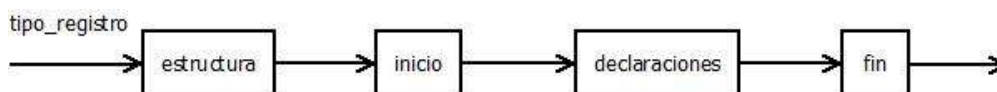
1. programa  $\rightarrow$  declaraciones funciones



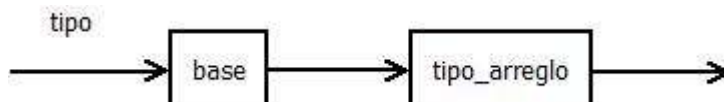
2. declaraciones  $\rightarrow$  tipo lista\_var ; declaraciones  
| tipo\_registro lista\_var ; declaraciones  
|  $\epsilon$



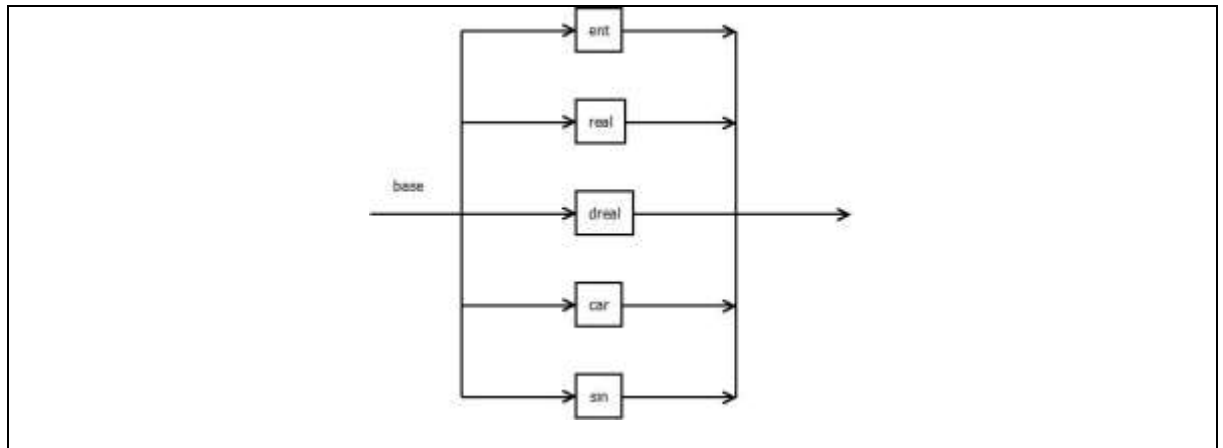
3. tipo\_registro  $\rightarrow$  estructura inicio declaraciones fin



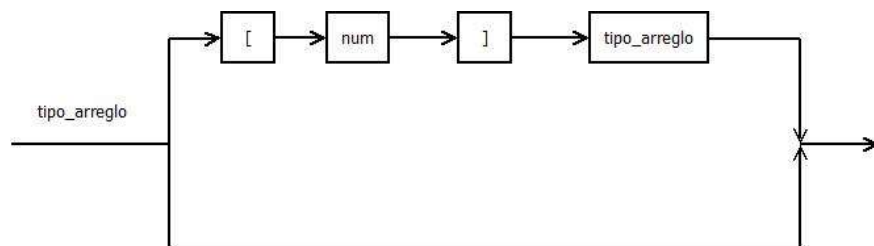
4. tipo  $\rightarrow$  base tipo\_arreglo



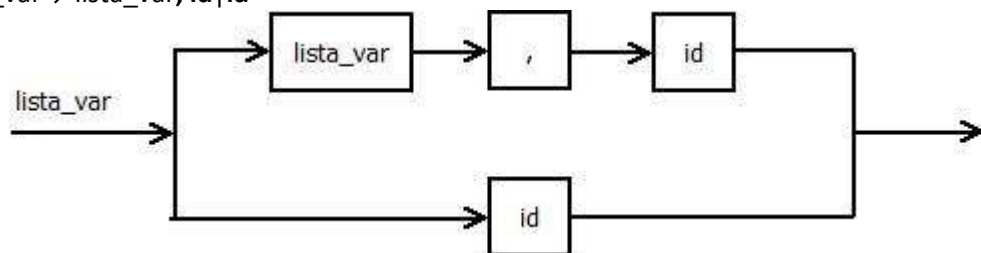
5. base  $\rightarrow$  ent|real|dreal|car|sin



6.  $\text{tipo\_arreglo} \rightarrow [\text{num}] \text{tipo\_arreglo} \mid \epsilon$



7.  $\text{lista\_var} \rightarrow \text{lista\_var}, \text{id} \mid \text{id}$



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa=id    Beta=id    A= lista\_var

$A \rightarrow \text{beta } A'$

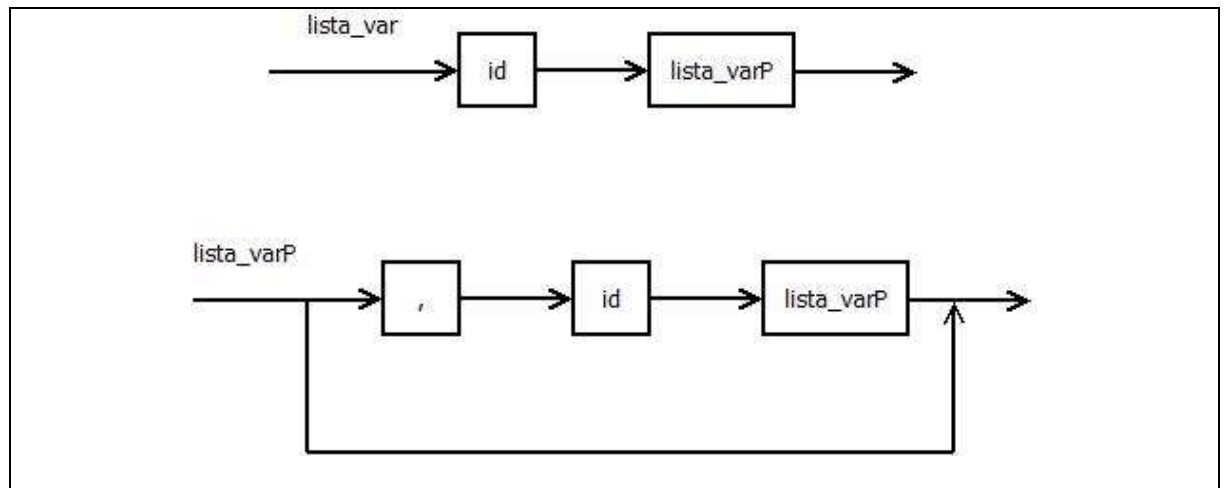
$A' \rightarrow \text{alfa } A' \mid \epsilon$

\*\*\*\*\*

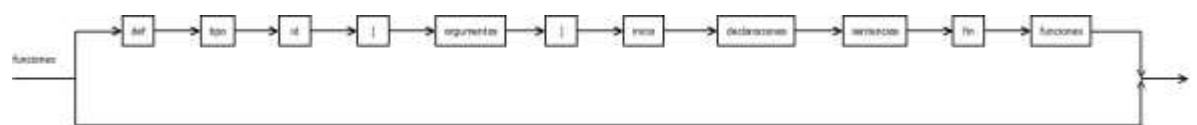
Nuevas producciones:

7.  $\text{lista\_var} \rightarrow \text{id } \text{lista\_varP}$

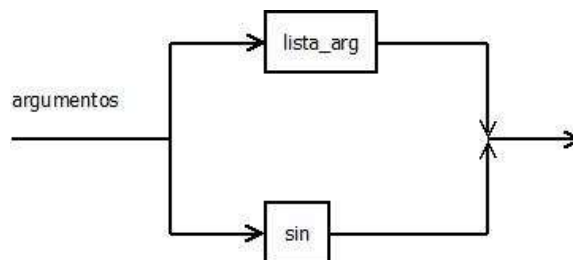
7'.  $\text{lista\_varP} \rightarrow ,\text{id } \text{lista\_varP} \mid \epsilon$



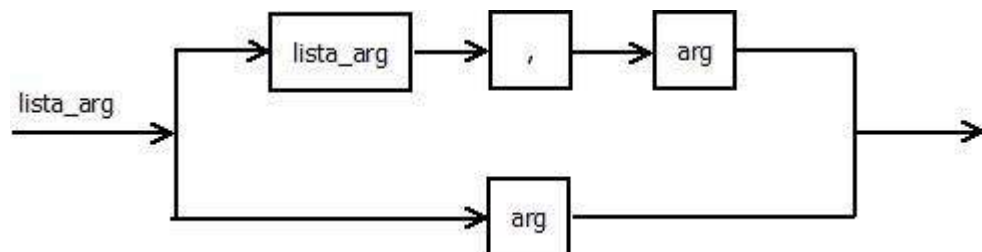
8. funciones  $\rightarrow$  **def** tipo **id**(argumentos) **inicio** declaraciones sentencias **fin** funciones |  $\epsilon$



9. argumentos  $\rightarrow$  listar\_arg | **sin**



10. lista\_arg  $\rightarrow$  lista\_arg, arg | arg



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa=,arg    Beta=arg    A= lista\_arg

$A \rightarrow \text{beta } A'$

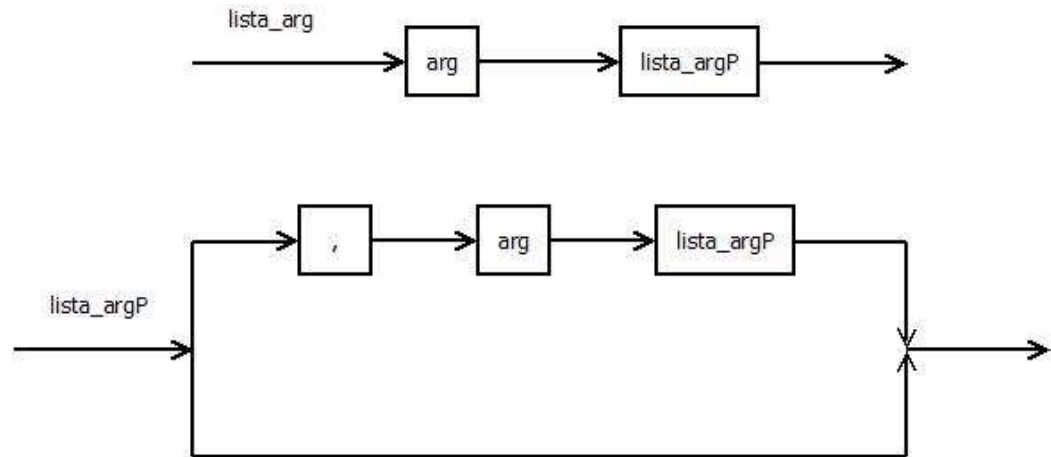
$A' \rightarrow \text{alfa } A' \mid \epsilon$

\*\*\*\*\*

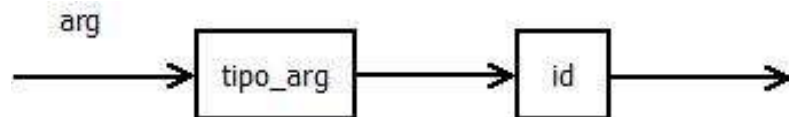
Nuevas producciones:

10. lista\_arg  $\rightarrow$  arg lista\_argP

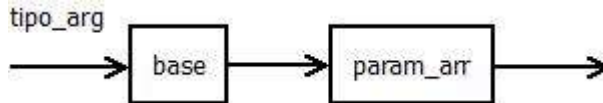
10'. lista\_argP  $\rightarrow$  ,arg lista\_argP |  $\epsilon$



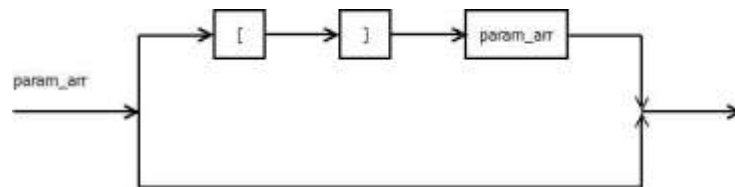
11. arg  $\rightarrow$  tipo\_arg id



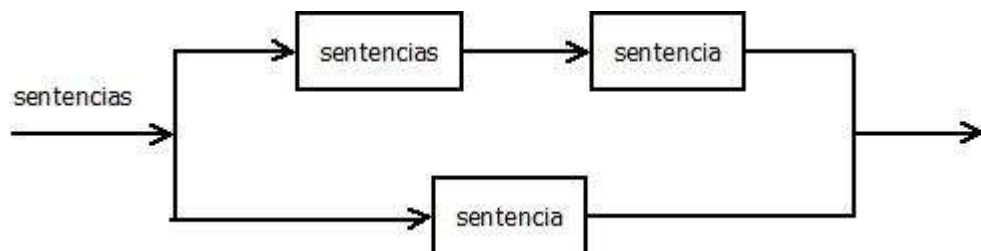
12. tipo\_arg  $\rightarrow$  base param\_arr



13. param\_arr  $\rightarrow$  [ ] param\_arr |  $\epsilon$



14. sentencias  $\rightarrow$  sentencias sentencia | sentencia



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa=sentencia    Beta=sentencia    A= sentencias

$A \rightarrow \text{beta } A'$

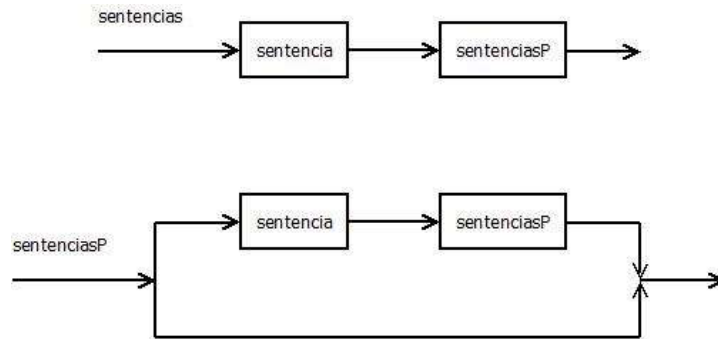
$A' \rightarrow \text{alfa } A' \mid \epsilon$

\*\*\*\*\*

Nuevas producciones:

14. sentencias  $\rightarrow$  sentencia sentenciasP

14'. sentenciasP  $\rightarrow$  sentencia sentenciasP  $\mid \epsilon$



15. sentencia  $\rightarrow$  **si** e\_bool **entonces** sentencia **fin**

| **si** e\_bool **entonces** sentencia **sino** sentencia **fin**

| **mientras** e\_bool **hacer** sentencia **fin**

| **hacer** sentencia **mientras** e\_bool;

| **segun** (variable) **hacer** casos predeterminado **fin**

| variable := expresion ;

| **escribir** expresion ;

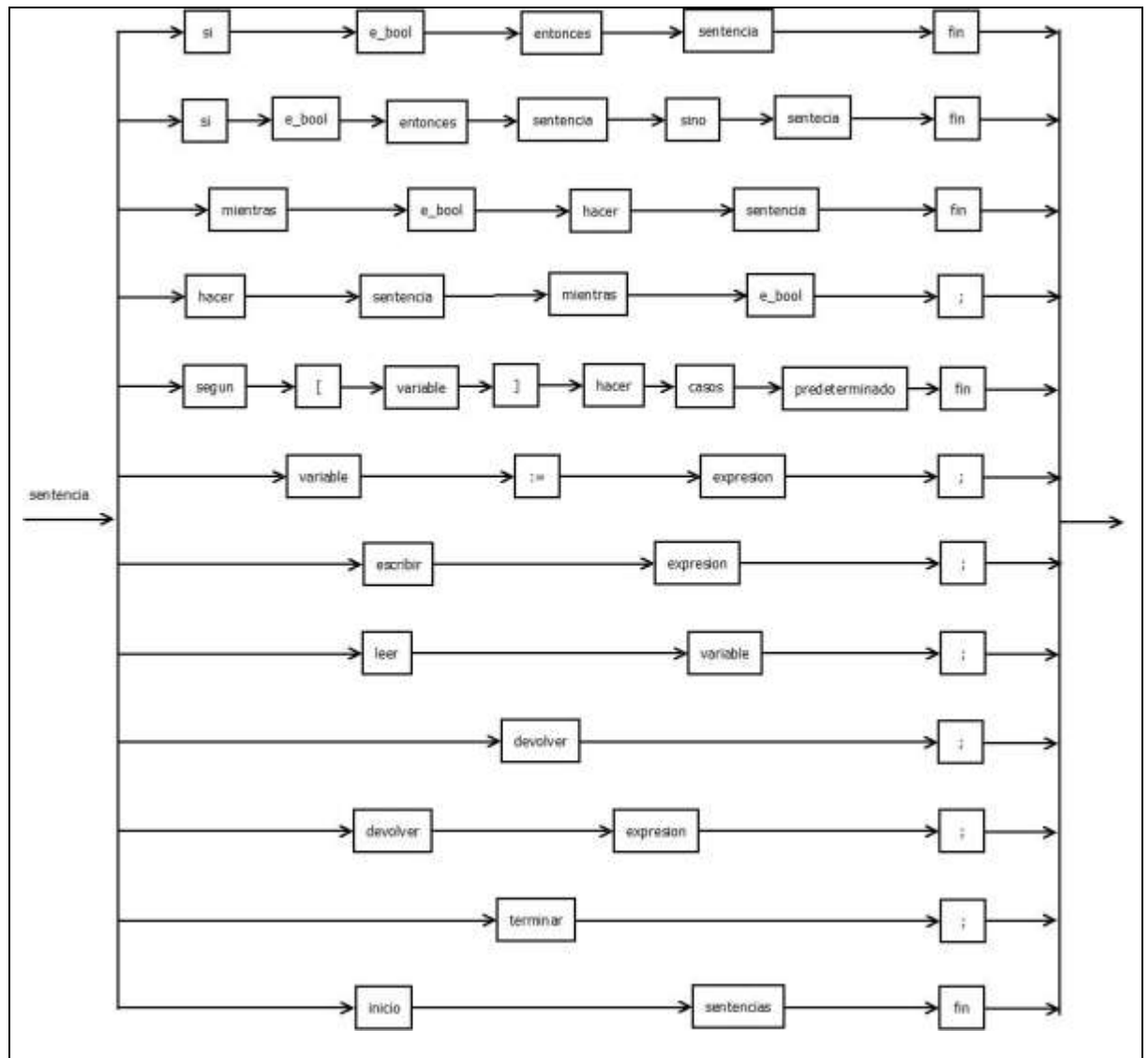
| **leer** variable ;

| **devolver**;

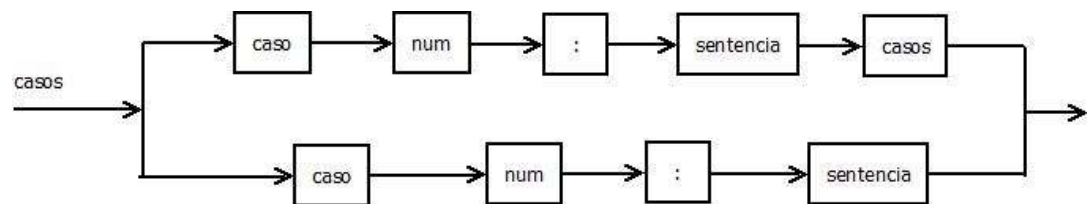
| **devolver** expresion;

| **terminar**;

| **inicio** sentencias **fin**



16. casos  $\rightarrow$  caso num: sentencia casos | caso num: sentencia



\*\*\*\*\*Factorizamos\*\*\*\*\*

Identificamos alfa, beta, gama y A

Alfa= caso num: sentencia    Beta(s)= casos |  $\epsilon$     gama=  $\epsilon$     A= casos

$A \rightarrow$  alfa A'

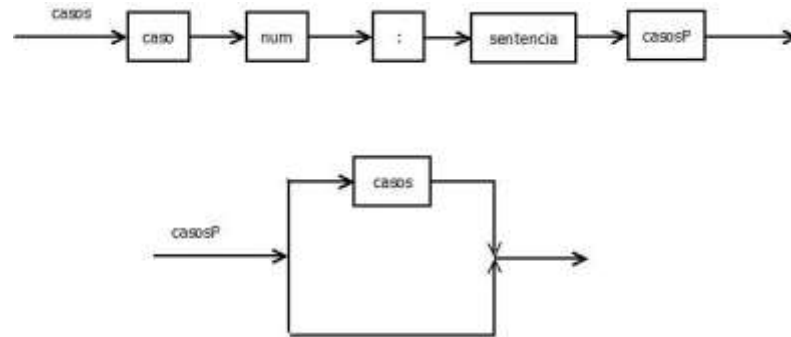
$A' \rightarrow$  beta1 | ... | betaN

\*\*\*\*\*

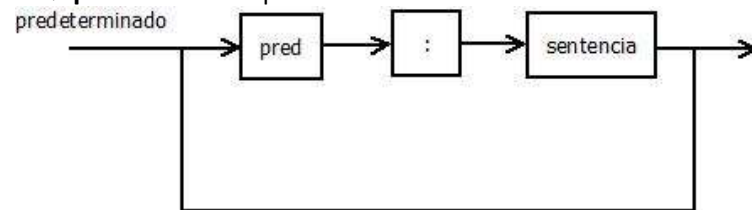
Nuevas producciones:

16. casos  $\rightarrow$  caso num: sentencia casosP

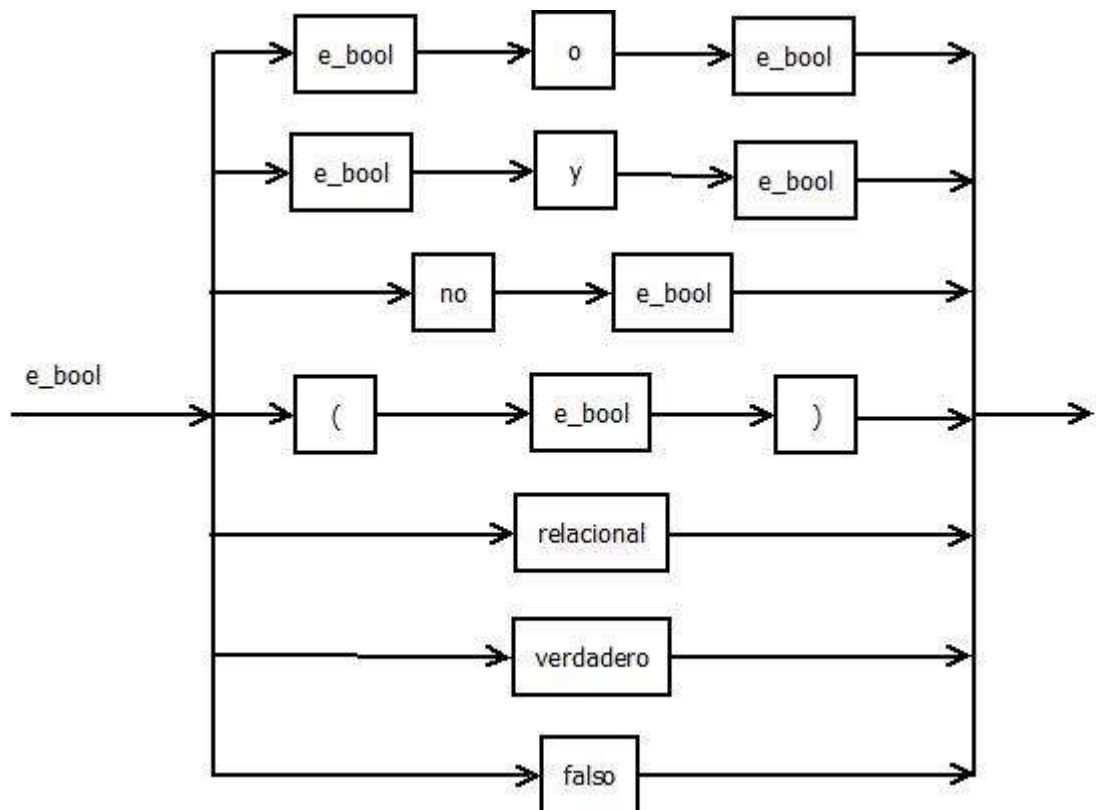
16'.  $\text{casosP} \rightarrow \text{casos} \mid \epsilon$



17.  $\text{predeterminado} \rightarrow \text{pred: sentencia} \mid \epsilon$



18.  $\text{e\_bool} \rightarrow \text{e\_bool o e\_bool} \mid \text{e\_bool y e\_bool} \mid \text{no e\_bool} \mid (\text{e\_bool}) \mid \text{relacional} \mid \text{verdadero} \mid \text{falso}$





\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa=  $\text{no } e\_bool \mid y \ e\_bool$

Beta=  $\text{no } e\_bool \mid ( e\_bool ) \mid \text{relacional} \mid \text{verdadero} \mid \text{falso}$

A=  $e\_bool$

$A \rightarrow \text{beta } A'$

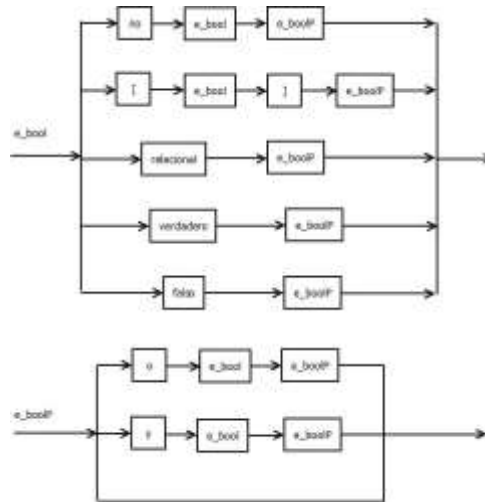
$A' \rightarrow \text{alfa } A' \mid \epsilon$

\*\*\*\*\*

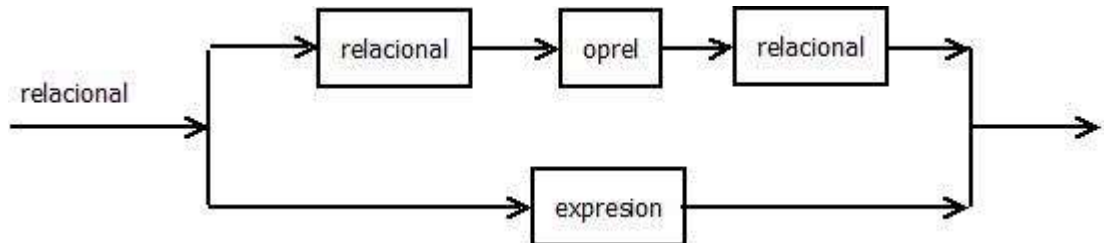
Nuevas Producciones:

18.  $e\_bool \rightarrow \text{no } e\_bool \ e\_boolP \mid ( e\_bool ) \ e\_boolP \mid \text{relacional } e\_boolP \mid \text{verdadero } e\_boolP \mid \text{falso } e\_boolP$

18'.  $e\_boolP \rightarrow \text{no } e\_bool \ e\_boolP \mid y \ e\_bool \ e\_boolP \mid \epsilon$



19.  $\text{relacional} \rightarrow \text{relacional } \text{oprel } \text{relacional} \mid \text{expresion}$



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa=  $\text{oprel } \text{relacional}$

Beta=  $\text{expresion}$

A=  $\text{relacional}$

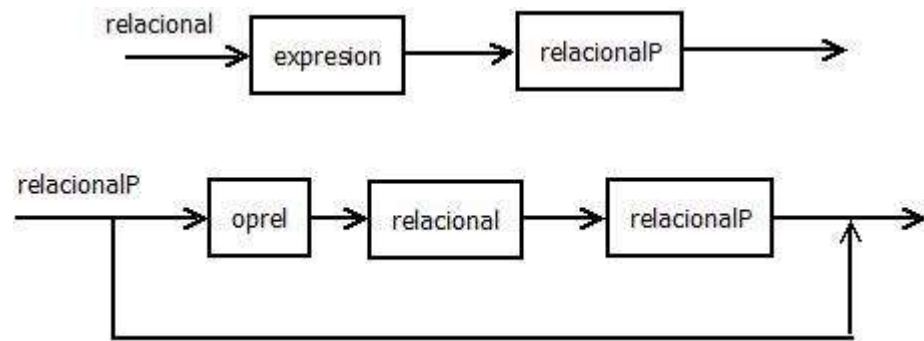
$A \rightarrow \text{beta } A'$

$A' \rightarrow \text{alfa } A' \mid \epsilon$

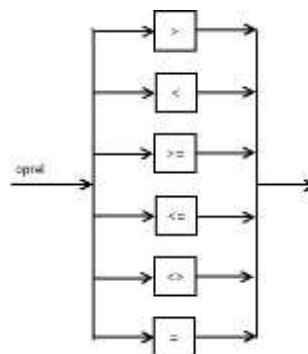
\*\*\*\*\*

19.  $\text{relacional} \rightarrow \text{expresion } \text{relacionalP}$

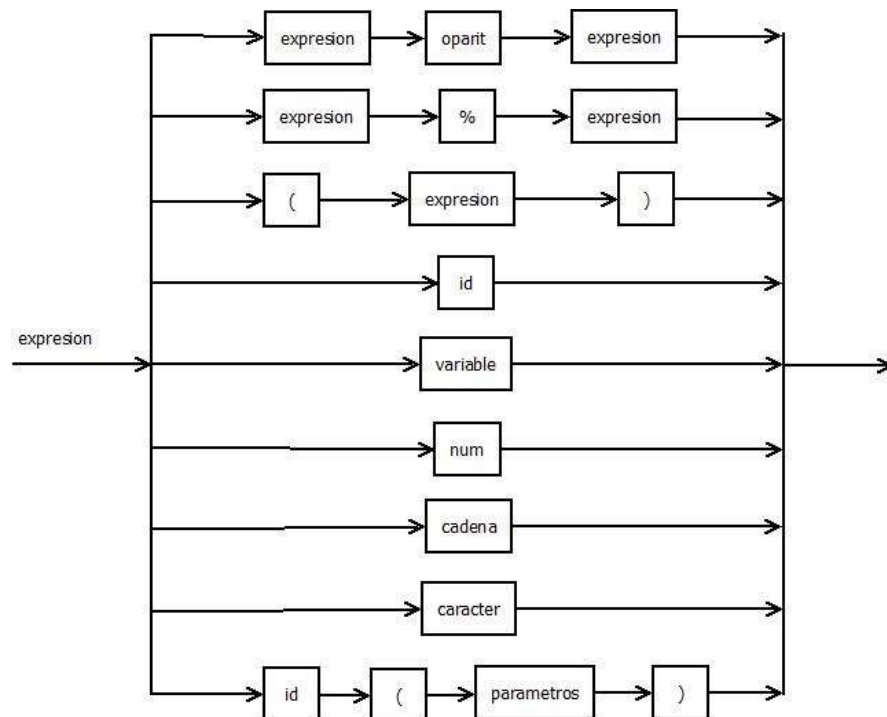
19'.  $\text{relacionalP} \rightarrow \text{oprel } \text{relacional } \text{relacionalP} \mid \epsilon$



20. oprel → > | < | >= | <= | <> | =

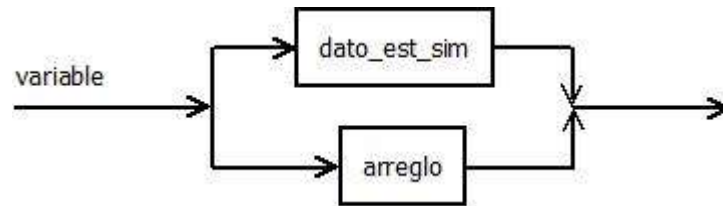


21. expresion → expresion oparit expresion | expresion % expresion | (expresion ) | id | variable | num | cadena | caracter | id(parametros)

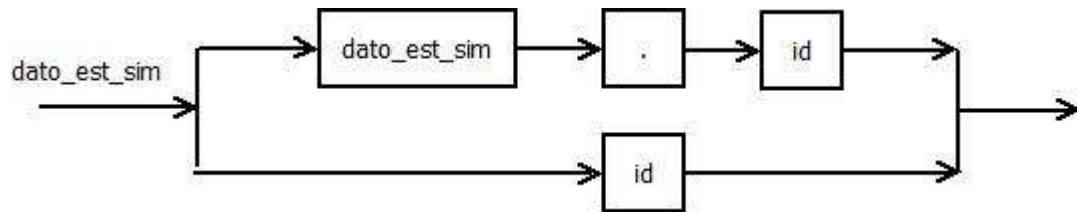




23.  $\text{variable} \rightarrow \text{dato\_est\_sim} \mid \text{arreglo}$



24.  $\text{dato\_est\_sim} \rightarrow \text{dato\_est\_sim}.\text{id} \mid \text{id}$



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa= .id

Beta= id

A= dato\_est\_sim

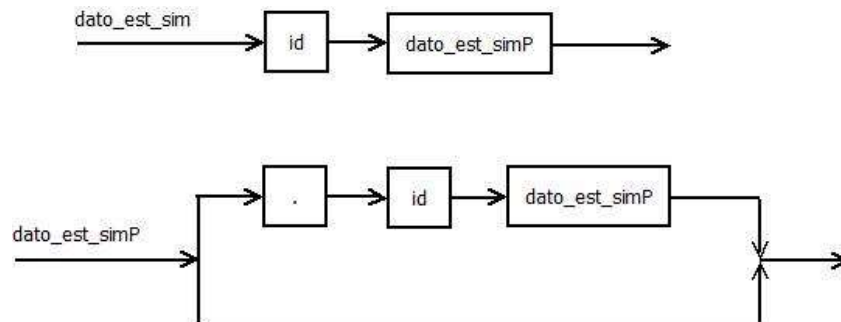
$A \rightarrow \text{beta } A'$

$A' \rightarrow \text{alfa } A' \mid \epsilon$

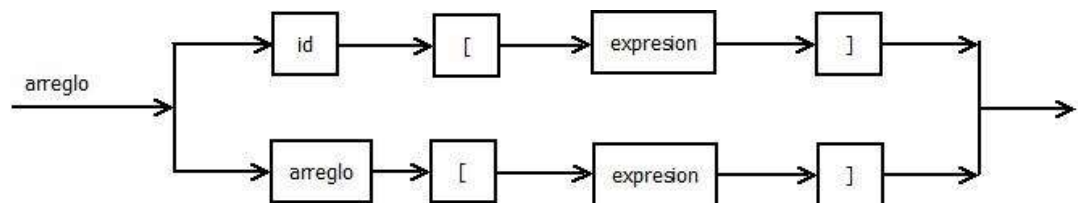
\*\*\*\*\*

24.  $\text{dato\_est\_sim} \rightarrow \text{id } \text{dato\_est\_simP}$

24'.  $\text{dato\_est\_simP} \rightarrow \text{id } \text{dato\_est\_simP} \mid \epsilon$



25.  $\text{arreglo} \rightarrow \text{id } [ \text{expresion } ] \mid \text{arreglo } [ \text{expresion } ]$



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa= [expresion]

Beta= id[expresion]

A= arreglo

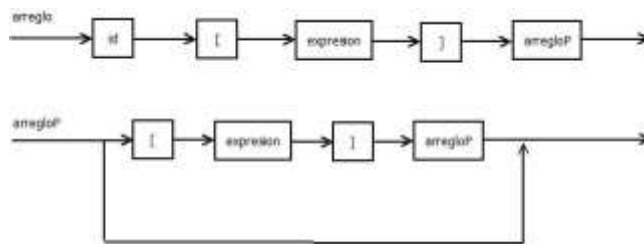
$A \rightarrow \text{beta } A'$

$A' \rightarrow \text{alfa } A' \mid \epsilon$

Nuevas Producciones:

25. arreglo  $\rightarrow$  id[expresion] arregloP

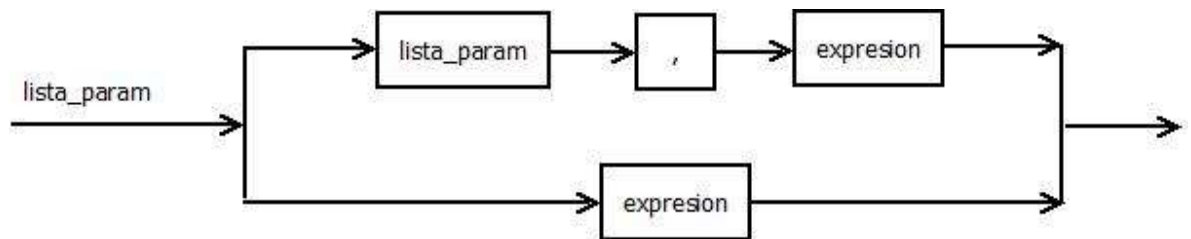
25'. arregloP  $\rightarrow$  [expresion] arregloP  $\mid \epsilon$



26. parametros  $\rightarrow$  lista\_param  $\mid \epsilon$



27. lista\_param  $\rightarrow$  lista\_param , expresion | expresión



\*\*\*\*\*Eliminamos recursividad\*\*\*\*\*

Identificamos alfa, beta y A

Alfa= , expresion

Beta= expresion

A= lista\_param

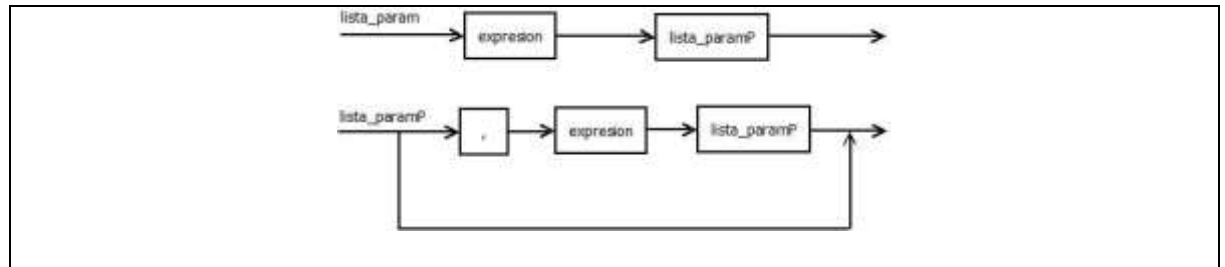
$A \rightarrow \text{beta } A'$

$A' \rightarrow \text{alfa } A' \mid \epsilon$

Nuevas producciones:

27. lista\_param  $\rightarrow$  expresion lista\_paramP

27'. lista\_paramP  $\rightarrow$  , expresion lista\_paramP  $\mid \epsilon$



## Implementación

Como se mencionó en el análisis del problema, debemos tener listo el analizador léxico, ya que será la base de nuestro programa. Lo primero a realizar es declarar la función `yylex()` y la función de error. Dado que tenemos identificadores, cadenas y números, necesitamos declarar una variable que tome los valores correspondientes a cada uno, cuando se usen. Se permitirán identificadores de hasta 32 caracteres y cadenas de hasta 100. Una vez que se declaran los tokens, se indica cuál es el símbolo inicial, que para esta gramática es "programa". Se escriben las diferentes producciones que componen a la gramática; en los terminales se escribe el valor que regresó el análisis léxico, y están indicados con letras mayúsculas. Al final del programa se tiene el código del usuario, y hay una función que indica si se presenta un error en la sintaxis, porque no se cumplen las reglas establecidas por la gramática.

## Forma de ejecutar el programa

Para realizar el análisis sintáctico se requiere tener una función principal para leer el archivo, así como un analizador sintáctico. Para su ejecución se debe poner la siguiente instrucción en la consola: **`./nombre_programa entrada`**.