Análisis Sintáctico

Cázares Rodríguez Víctor Manuel
Limones Moscoso Ulises
Ojeda Ávila Diego Antonio
Rivera Arellanes Josué David

Grupo 2
Compiladores
Semestre 2020-2

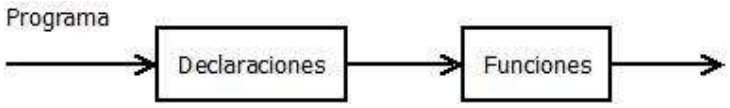
Análisis del problema

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.

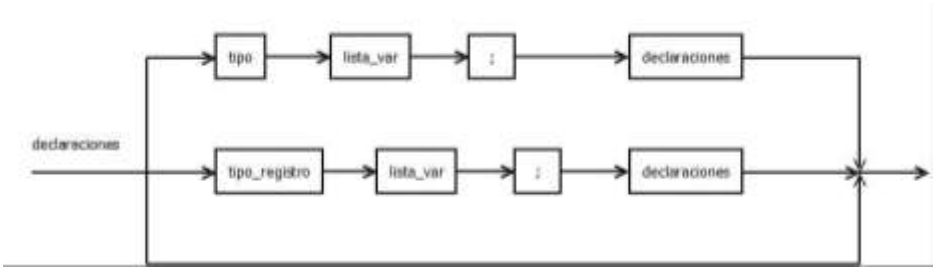
Diseño de la solución

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

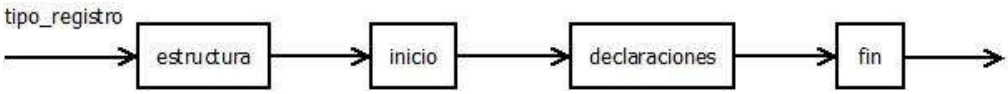
1. programa → declaraciones funciones



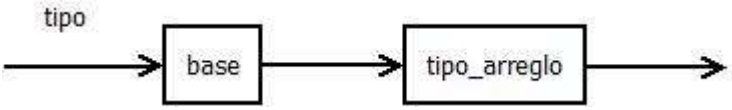
2. declaraciones → tipo lista_var **;** declaraciones
|tipo_registro lista_var **;** declaraciones
| ε



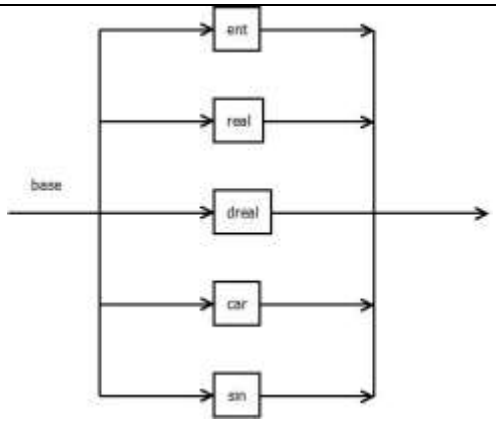
3. tipo_registro → **estructura inicio** declaraciones **fin**



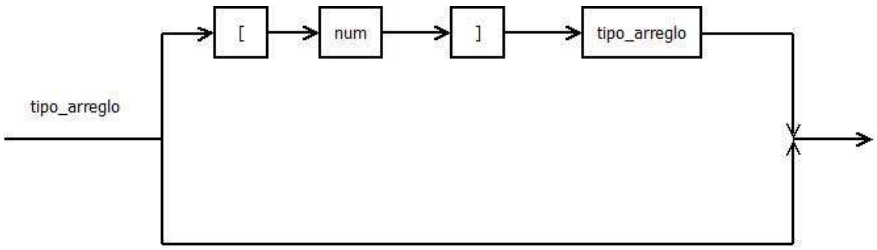
4. tipo→ base tipo_arreglo



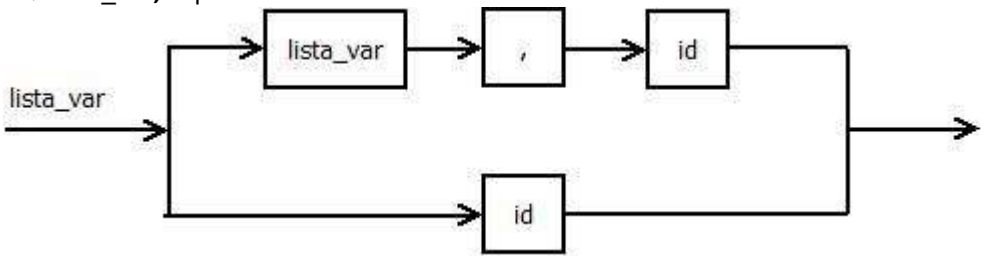
5. base→ **ent**|**real**|**dreal**|**car**|**sin**



6. tipo_arreglo→**[num]** tipo_arreglo| ε



7. lista_var→ lista_var**, id**|**id**



**************Eliminamos recursividad******************

Identificamos alfa, beta y A

Alfa=,id Beta=id A= lista_var

A→ beta A'

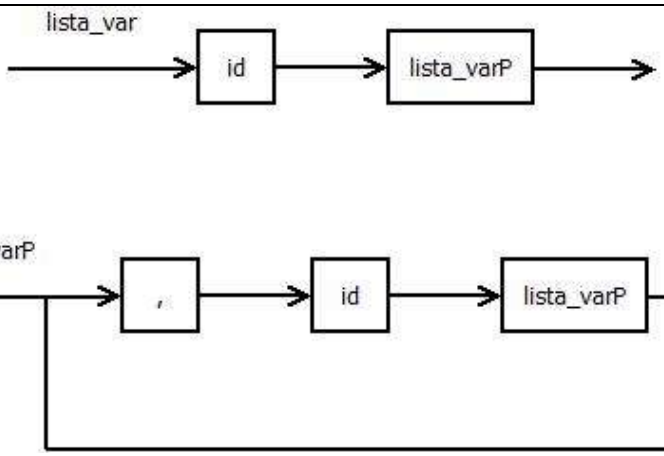
A'→ alfa A' | ε

****

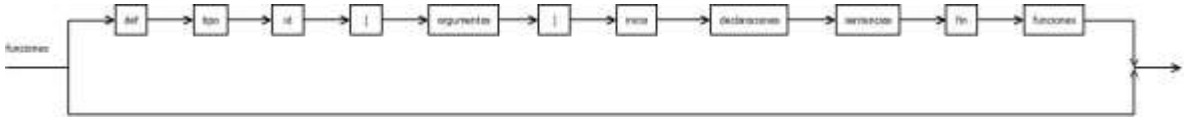
Nuevas producciones:

7. lista_var --> **id** lista_varP

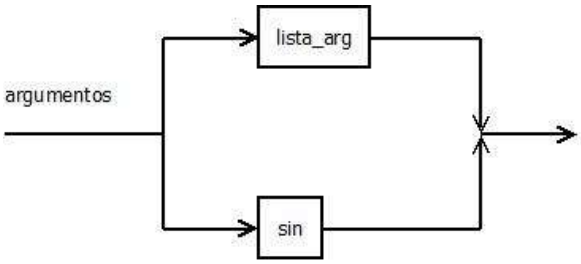
7'. lista_varP --> **,id** lista_varP | ε



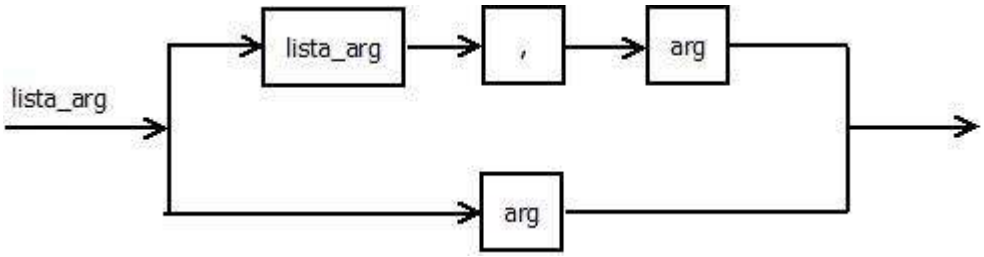
8. funciones→ **def** tipo **id(**argumentos**) inicio** declaraciones sentencias **fin** funciones | ε



9. argumentos→ listar_arg | **sin**



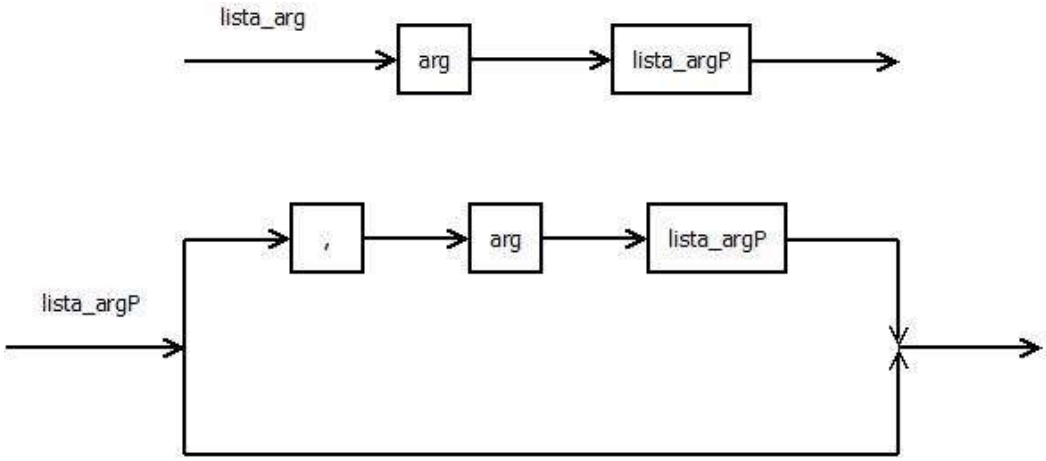
10. lista_arg→ lista_arg, arg | arg



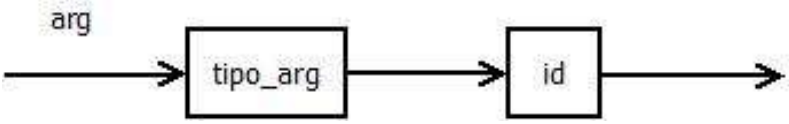
**************Eliminamos recursividad******************
Identificamos alfa, beta y A
Alfa=,arg Beta=arg A= lista_arg
A→ beta A'
A'→ alfa A' | ε
****

Nuevas producciones:
10. lista_arg→arg lista_argP

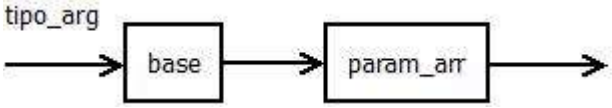
10’. lista_argP→**,**arg lista_argP | ε



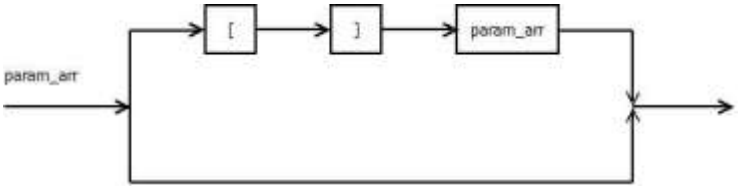
11. arg→ tipo_arg **id**



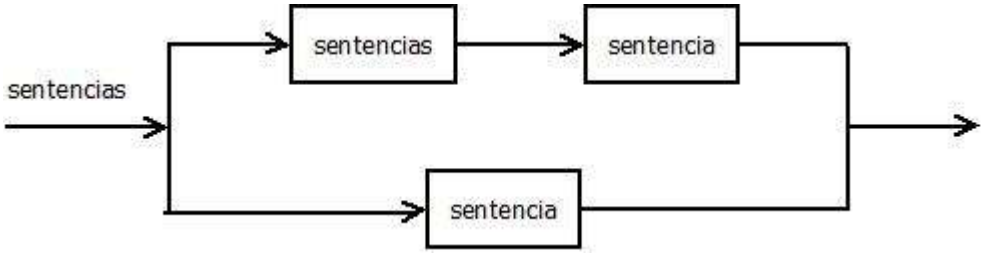
12. tipo_arg→ base param_arr



13. param_arr→ **[ ]** param_arr| ε



14. sentencias→ sentencias sentencia | sentencia

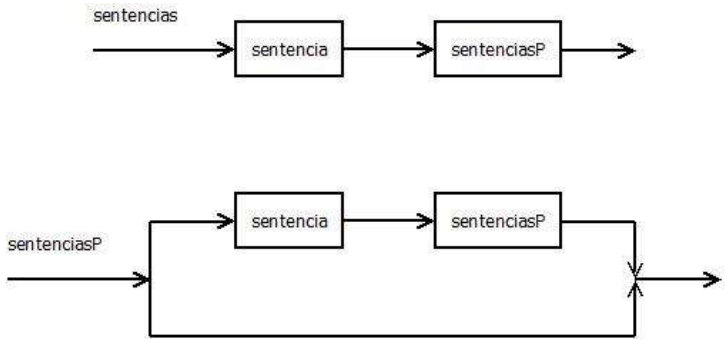


**************Eliminamos recursividad*****************

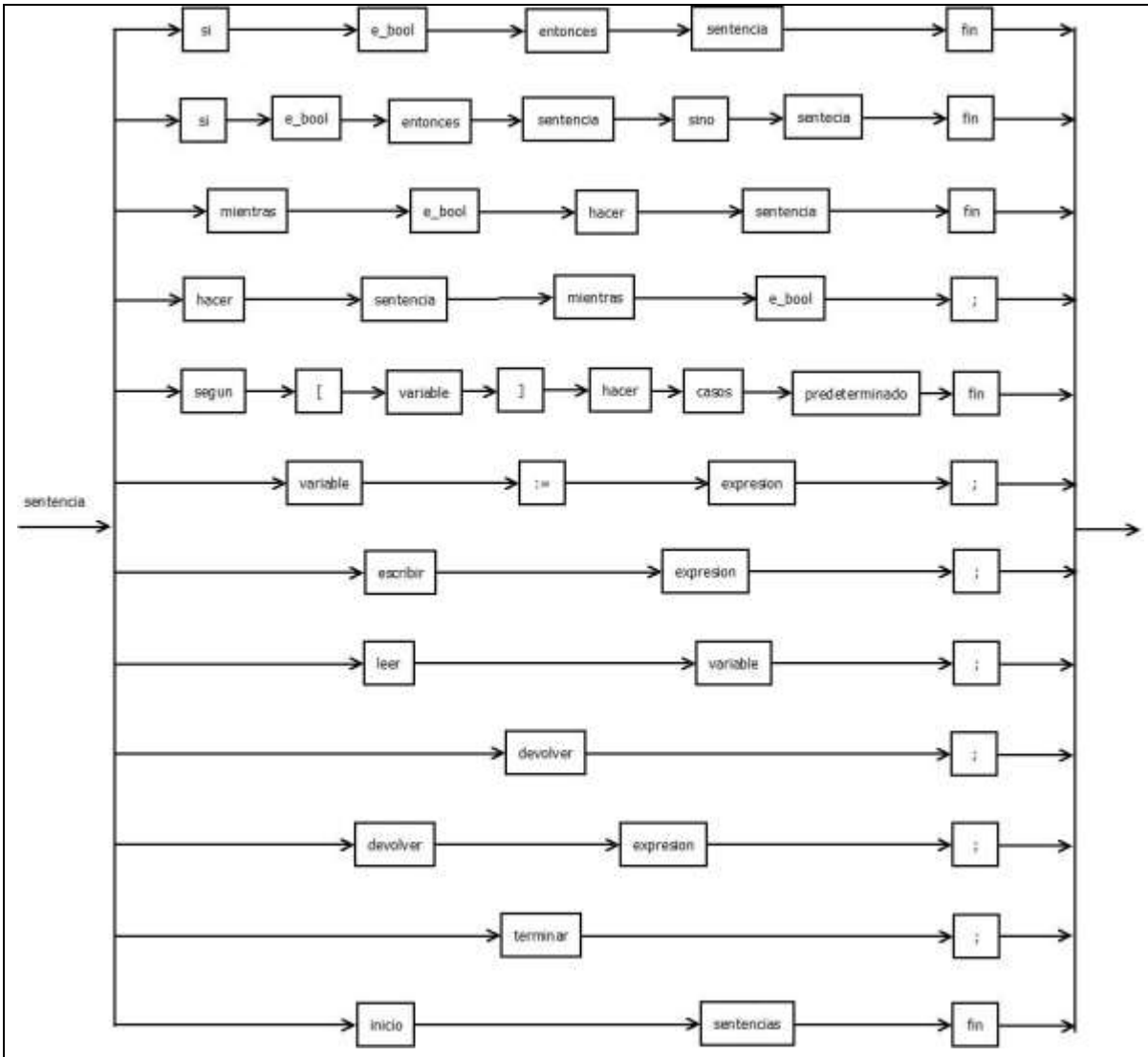
Identificamos alfa, beta y A

Alfa=sentencia Beta=sentencia A= sentencias
A→ beta A'
A'→ alfa A' | ε
****

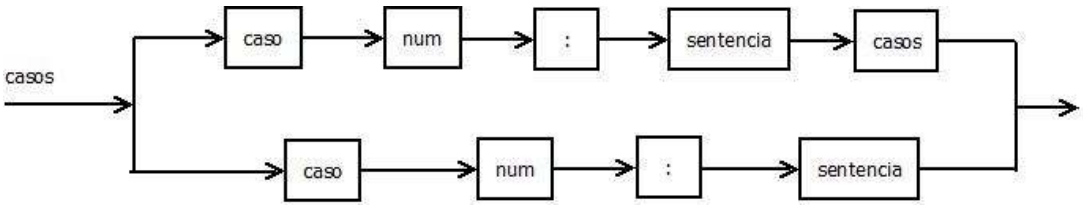
Nuevas producciones:
14. sentencias→ sentencia sentenciasP
14'. sentenciasP→ sentencia sentenciasP | ε



15. sentencia→ **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→**caso num:** sentencia casos|**caso num:** sentencia



**************Factorizamos*****************

Identificamos alfa, beta,gama y A

Alfa= caso num: sentencia **Beta(s)= casos | ε gama= ε A= casos**

A→ alfa A'

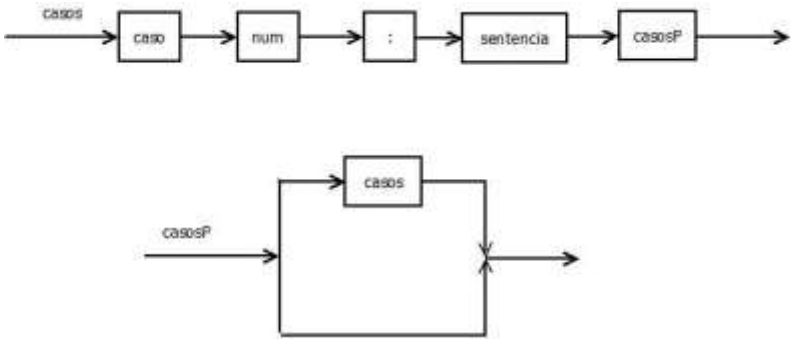
A'→ beta1 | … | betaN

****

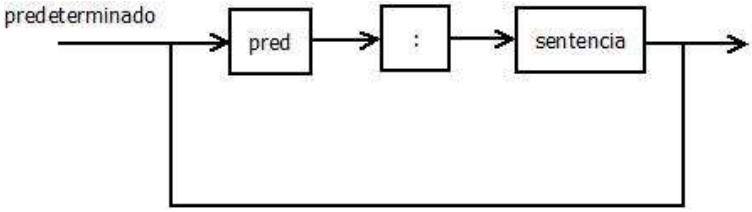
Nuevas producciones:

16. casos→ **caso num:** sentencia casosP

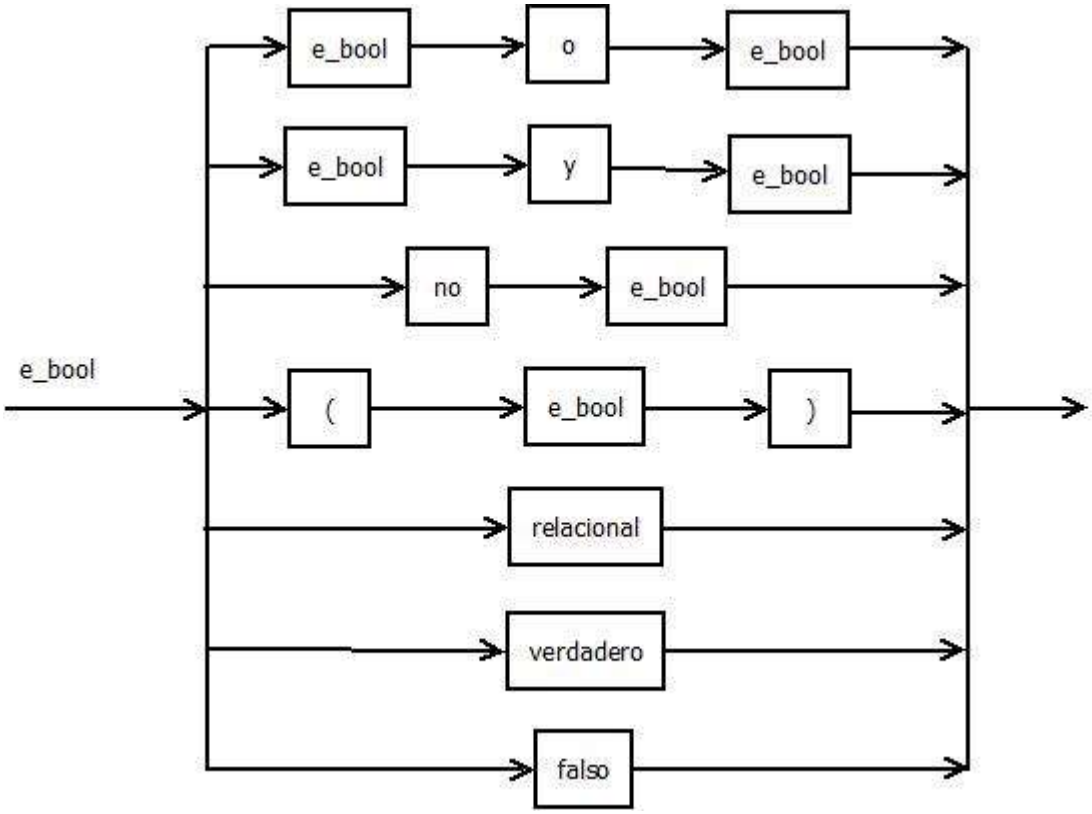
16'. casosP→ casos | ε



17. predeterminado→ **pred:** sentencia| ε



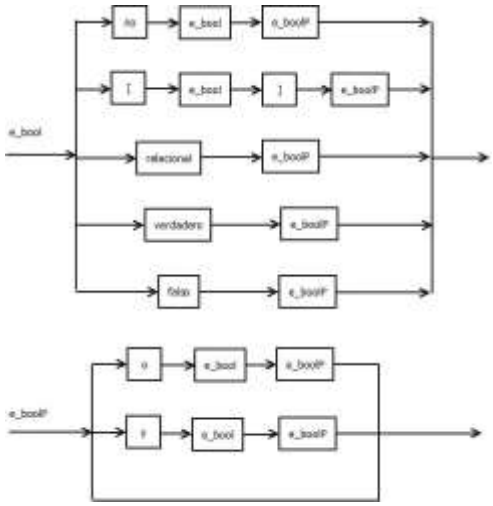
18. e_bool→ e_bool **o** e_bool|e_bool **y** e_bool |**no** e_bool|**(** e_bool **)** |relacional |**verdadero** |**falso**



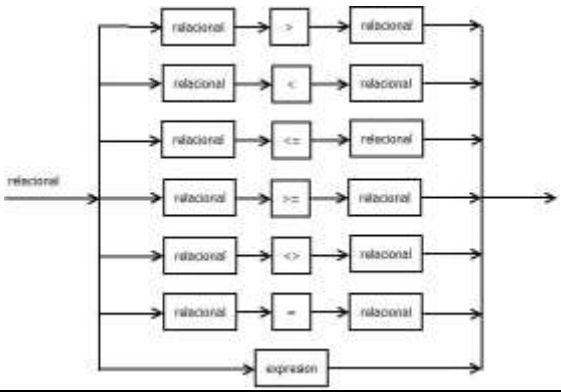
**************Eliminamos recursividad*****************

Identificamos alfa, beta y A
Alfa=o e_bool |y e_bool
Beta= no e_bool|(e_bool) |relacional |verdadero |falso
A= e_bool
A→ beta A'
A'→ alfa A' | ε
****

Nuevas Producciones:
18. e_bool→ **no** e_bool e_boolP|**(** e_bool **)** e_boolP |relacional e_boolP |**verdadero** e_boolP |**falso** e_boolP
18'. e_boolP→ **o** e_bool e_boolP| **y** e_bool e_boolP| ε

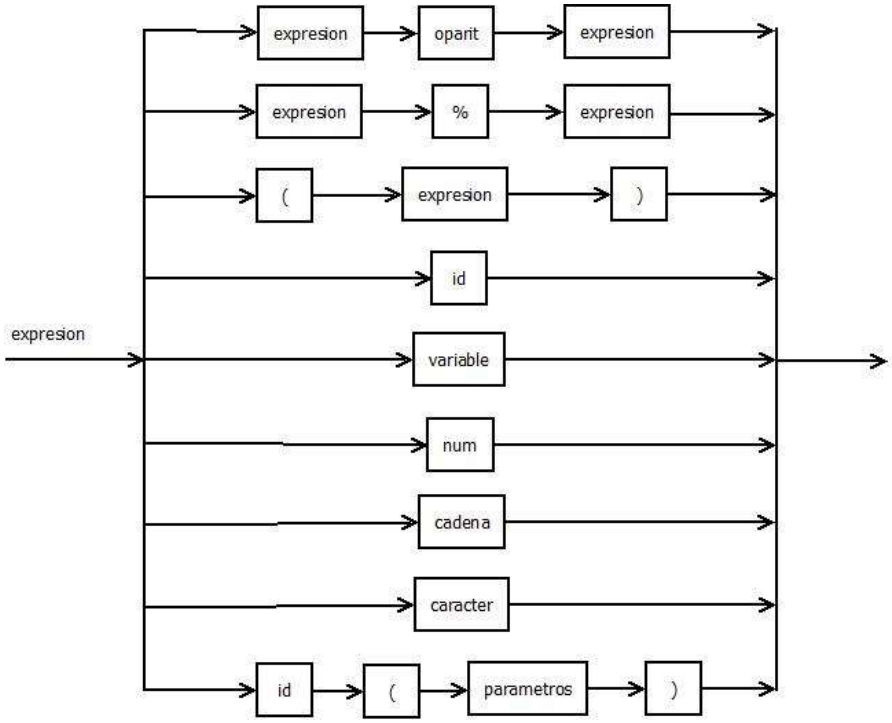


19. relacional→relacional > relacional
|relacional < relacional
|relacional <= relacional
|relacional >= relacional
|relacional <> relacional
|relacional = relacional
|expresion



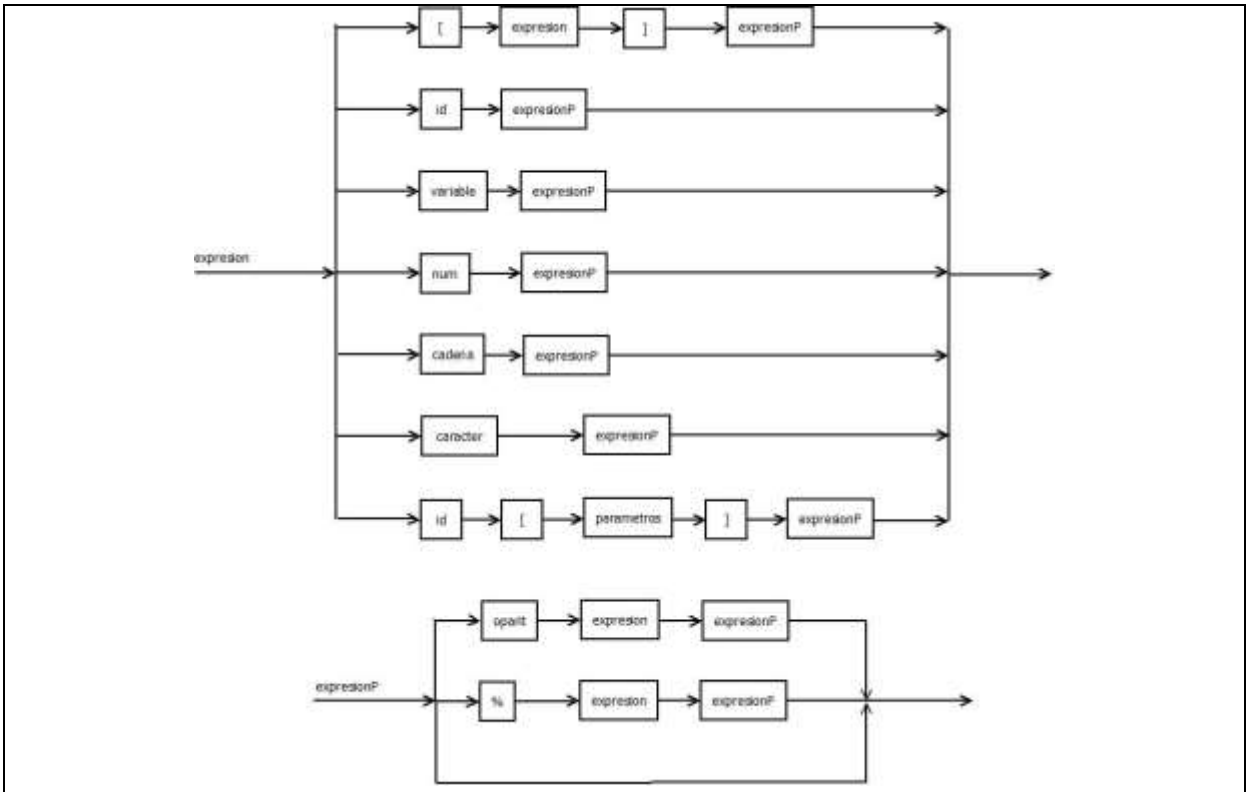
20. expresion→ expresion oparit expresion |expresion **%** expresion |**(**expresion **)**|**id** |variable

|**num**|**cadena**|**caracter**|**id(**parametros**)**

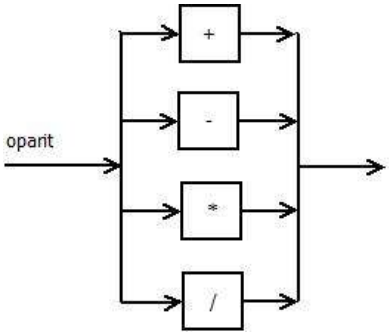


*************Eliminamos recursividad*****************
Identificamos alfa, beta y A
Alfa= oparit expresion | % expresion

Beta= (expresion **)**|**id** |variable|**num**|**cadena**|**caracter**|**id(**parametros**)**
A= expresion
A→ beta A'
A'→ alfa A' | ε
****
Nuevas producciones:
20.expresion→**(**expresion **)** expresionP |**id** expresionP |variable expresionP |**num** expresionP |**cadena** expresionP |**caracter** expresionP |**id(**parametros**)** expresionP
20'.expresionP→ oparit expresion expresionP | % expresion expresionP | ε

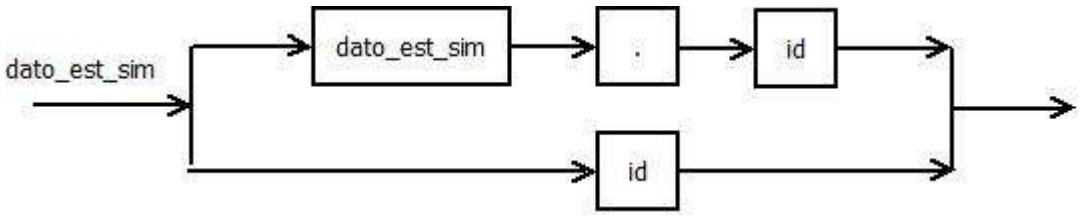


21. oparit→ **+** |**−**|**∗**| **/**



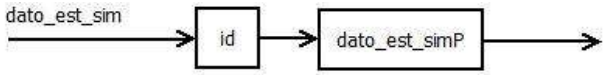
22. variable→ dato_est_sim | arreglo

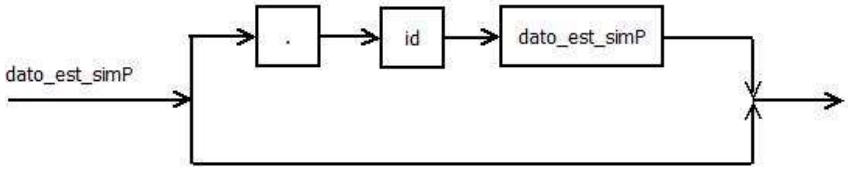
23. dato_est_sim→ dato_est_sim **.id**|**id**



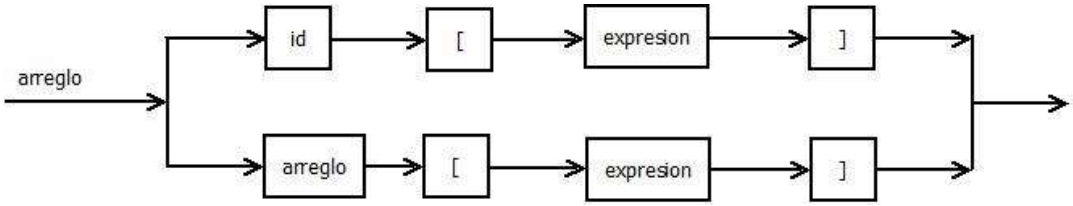
**************Eliminamos recursividad******************
Identificamos alfa, beta y A
Alfa= .id
Beta= id
A= dato_est_sim
A→ beta A'
A'→ alfa A' | ε
****

23. dato_est_sim→ **id** dato_est_simP
23'. dato_est_simP→ **.id** dato_est_simP | ε





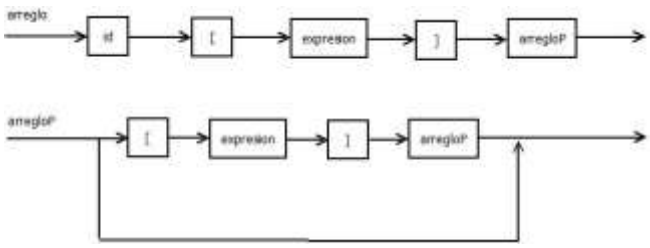
24. arreglo→ **id [** expresion **]**|arreglo **[** expresion **]**



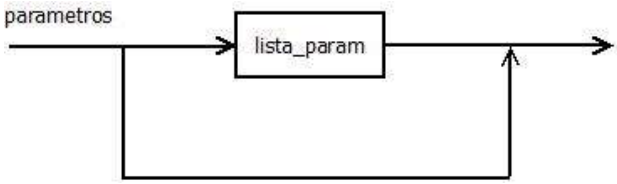
**************Eliminamos recursividad******************
Identificamos alfa, beta y A
Alfa= [expresion]
Beta= id[expresion]
A= arreglo
A→ beta A'
A'→ alfa A' | ε

Nuevas Producciones:
24. arreglo→id[expresion] arregloP

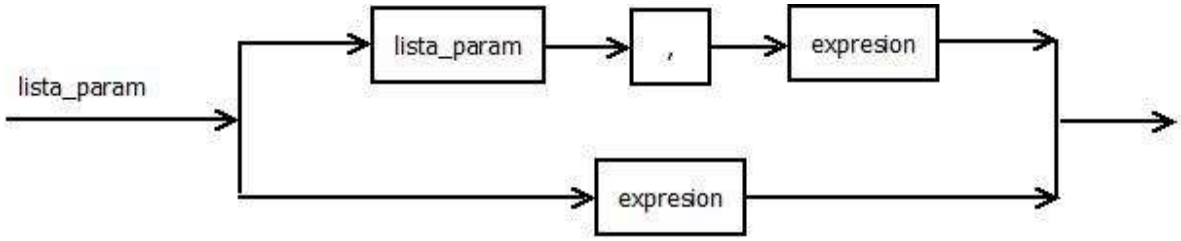
24'. arregloP→[expresion] arregloP | ε



25. parametros→ lista_param| ε



26. lista_param→ lista_param **,** expresion|expresión



**************Eliminamos recursividad******************

Identificamos alfa, beta y A

Alfa= , expresion

Beta= expresion

A= lista_param

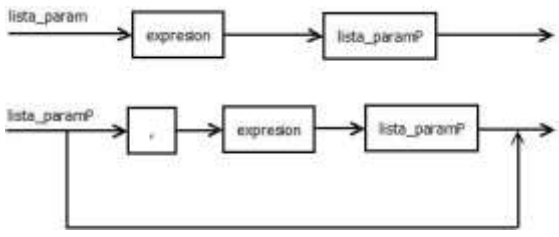
A→ beta A'

A'→ alfa A' | ε

Nuevas producciones:

26. lista_param → expresion lista_paramP

26'. lista_paramP →, expresion lista_paramP | ε



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".**