TECNOLÓGICO NACIONAL DE MÉXICO

INSTITUTO TECNOLÓGICO DE HERMOSILLO

Definición del autómata finito de su caso de estudio

Angel Eduardo Gaxiola Javier

Angélica Anahy Figueroa Yánez

Lenguajes y autómatas 1

Ana Luisa Millan Castro

Hermosillo / Sonora / México

Viernes / 1 / Junio /2017

Características principales y aplicación

GO es un proyecto desarrollado por el equipo de Google y sus diseñadores principales son Robert Griesemer, Rob Pike y Ken Thompson.

Fue desarrollado por el equipo de Google.

El propósito de GO es que los programadores sean más productivos, es un proyecto de código abierto para la comunidad.

GO fue desarrollado en el año 2009.

Fue desarrollado para facilitar la escritura de programas ya que es conciso, limpio y eficiente, aprovecha las maquinas multinucleo y en red, siendo flexible y modular lo que permite la compilación rápida.

Las plataformas para las cuales se desarrolla en GO es arquitectura Web, Android e IOS.

Es un lenguaje compilado de alto nivel debido a que su curva de aprendizaje es muy suave y es la intención del lenguaje.

Algunas de sus principales características:

- Compilado.
- Estáticamente tipado.
- Uso poco usual de POO: GO no usa clases, no usa herencia y el uso de interfaces se realiza de manera implícita. Esto con el fin de mejorar el rendimiento al momento de diseñar tu software.
- Uso de paquetes: Se usan los paquetes para organizar el código. Un paquete puede tener varios archivos “GO” que permiten definir lo que va a realizar el paquete. Para usar un paquete en tu programa, debes importarlo.

Tabla de símbolos

ID	TOKEN	EXPRESION REGULAR
letra(letra|digito)*	100	<ID>::= L(L|D)*

ENTEROS	TOKEN	EXPRESION REGULAR
(digito)+	101	<ENTEROS>::= (D)+

DECIMALES	TOKEN	EXPRESION REGULAR
(digito)+(.digito+)?	102	<ENTEROS>::= (D)+(.D+)?

COMENTARIOS DE LINEA	TOKEN	EXPRESION REGULAR
//	No genera toquen	<COMENTARIO DE LINEA>::= //

COMENTARIOS DE PARRAFO	TOKEN	EXPRESION REGULAR
/**/	No genera toquen.	<COMENTARIO DE PARRAFO>::= /**/

CADENA DE AGRUPACIÓN	TOKEN	EXPRESION REGULAR
(	106	<CADENA DE AGRUPACIÓN>::= (
)	107	<CADENA DE AGRUPACIÓN>::=)

{	108	<CADENA DE AGRUPACIÓN>::= {
}	109	<CADENA DE AGRUPACIÓN>::= }
[	110	<CADENA DE AGRUPACIÓN>::= [
]	111	<CADENA DE AGRUPACIÓN>::=]

CARACTERES DE PUNTACIÓN	**TOKEN**	**EXPRESION REGULAR**
.	103	<CARACTERES DE PUNTUACIÓN>::= .
;	104	<CARACTERES DE PUNTUACIÓN>::= ;

OPERADORES ARITMETICOS	**TOKEN**	**EXPRESION REGULAR**
+	112	<OPERADORES ARITMETICOS>::= +
-	113	<OPERADORES ARITMETICOS>::= -
*	117	<OPERADORES ARITMETICOS>::= *
/	115	<OPERADORES ARITMETICOS>::= /

%	116	<OPERADORES ARITMETICOS>::= %

OPERADORES RELACIONALES	**TOKEN**	**EXPRESION REGULAR**
==	129	<OPERADORES RELACIONALES>::= ==
!=	118	<OPERADORES RELACIONALES>::= !=
<	119	<OPERADORES RELACIONALES>::= <
<=	121	<OPERADORES RELACIONALES>::= <=
120	>	<OPERADORES RELACIONALES>::= >
122	>=	<OPERADORES RELACIONALES>::= >=

OPERADOR DE ASIGNACIÓN	**TOKEN**	**EXPRESION REGULAR**
=	128	<OPERADOR DE ASIGNACIÓN>::= =
:=	123	<OPERADOR DE ASIGNACIÓN>::= :=

OPERADOR LOGICO	**TOKEN**	**EXPRESION REGULAR**
&&	125	<OPERADOR LOGICO>::= &&
||	126	<OPERADOR LOGICO>::= ||
!	123	<OPERADOR LOGICO>::= !

OPERADOR BOOLEANO	**TOKEN**	**EXPRESION REGULAR**
true	212	<OPERADOR BOOLEANO>::= true
false	213	False

PALABRAS RESERVADAS	**TOKEN**	**EXPRESION REGULAR**
package	200	<PALABRAS RESERVADAS>::= package
main	201	<PALABRAS RESERVADAS>::= main
func	202	<PALABRAS RESERVADAS>::= func
print	203	<PALABRAS RESERVADAS>::= print
scan	204	<PALABRAS RESERVADAS>::= scan
var	205	<PALABRAS RESERVADAS>::=

		var
int	206	<PALABRAS RESERVADAS>::= int
string	207	<PALABRAS RESERVADAS>::= string
bool	208	<PALABRAS RESERVADAS>::= bool
if	209	<PALABRAS RESERVADAS>::= if
else	210	<PALABRAS RESERVADAS>::= else
for	211	<PALABRAS RESERVADAS>::= for
true	212	<PALABRAS RESERVADAS>::= true
false	213	<PALABRAS RESERVADAS>::= false
break	214	<PALABRAS RESERVADAS>::= break
continue	215	<PALABRAS RESERVADAS>::= continue
const	216	<PALABRAS RESERVADAS>::= const
type	217	<PALABRAS RESERVADAS>::= type
float	218	<PALABRAS RESERVADAS>::=

		float

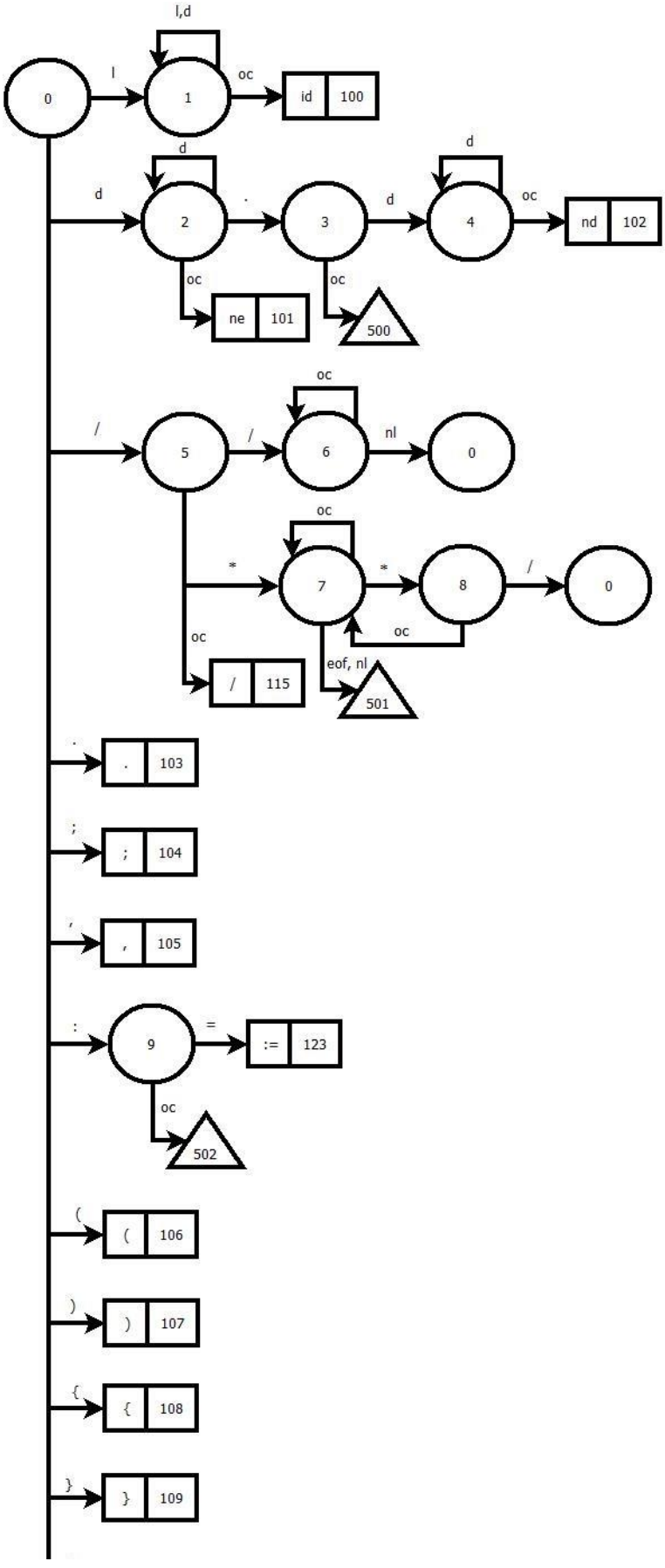
ERRORES	**TOKEN**
Se esperaba decimal	500
Se esperaba cierre del comentario	501
Se esperaba =	502
Se esperaba cierre de comilla	503
Se esperaba &	504
Se esperaba |	505
Elemeno no identificado	506

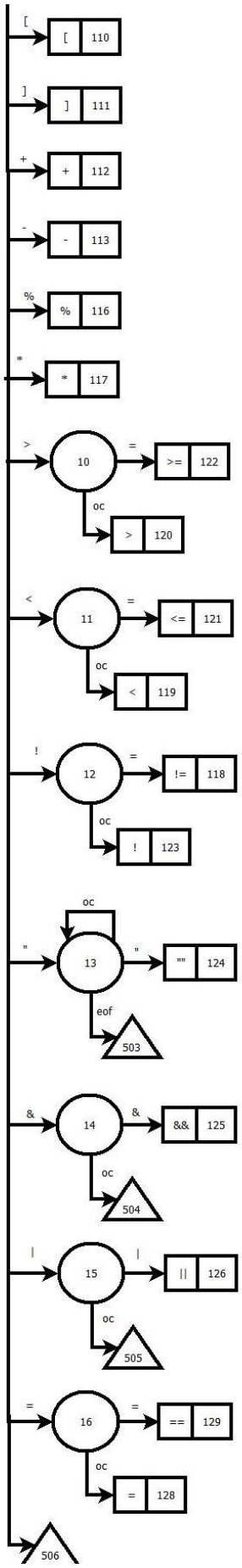
DELIMITADORES	**TOKEN**
tab	Sin token
nl (New line)	Sin token
eol (End of line)	Sin token
eof (End of file)	Sin token
eb	Sin token

ESTRUCTURA MINIMA

package main
 	
func main(){
 	
}

Autómata


l,d
0
l
1
oc
id
100
d
d
2
.
3
d
4
oc
nd
102
oc
ne
101
oc
500
oc
/
5
/
6
nl
0
oc
*
7
*
8
/
0
oc
oc
/
115
eof, nl
501
.
.
103
;
;
104
,
,
105
:
9
=
:=
123
oc
502
(
(
106
)
)
107
{
{
108
}
}
109


[
[110
]
] 111
+
+ 112
-
- 113
%
% 116
*
* 117
>
10
=
>= 122
oc
> 120
<
11
=
<= 121
oc
< 119
!
12
=
!= 118
oc
! 123
oc
"
13
"
"" 124
eof
503
&
14
&
&& 125
oc
504
|
15
|
|| 126
oc
505
=
16
=
== 129
oc
= 128
506

Matriz de transición

```
//      0   1   2   3   4   5   6   7   8   9  10  11  12  13  14  15  16  17  18  19  20  21  22  23  24  25  26  27  28  29
/*E\C    l   d   /   .   ;   ,   :   =   (   )   {   }   [   ]   +   -   %   *   >   <   !   "   &   |  eb  eol nl tab  eof oc  */
/*0*/   { 1 , 2 ,115,103,104,105, 9 ,16 ,106,107,108,109,110,111,112,113,116,117,10 ,11 ,12 ,13 ,14 ,15 ,0  ,0  ,0  ,0  ,0  ,506},
/*1*/   { 1 , 1 ,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100,100},
/*2*/   {101, 2 ,101, 3 ,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101,101},
/*3*/   {500, 4 ,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500,500},
/*4*/   {102, 4 ,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102,102},
//  /*5*/ {115,115,6  ,115,115,115,115,115,115,115,115,115,115,115,115,115,115,7  ,115,115,115,115,115,115,115,115,115,115,115,115},
/*6*/   {6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,6  ,0  ,0  ,6  ,6  ,6  },
/*7*/   {7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,8  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,501,7  },
/*8*/   { 7 ,7  ,0  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  ,7  },
/*9*/   {502,502,502,502,502,502,502,123,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502},
/*10*/  {120,120,120,120,120,120,120,122,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120,120},
/*11*/  {119,119,119,119,119,119,119,121,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119,119},
/*12*/  {502,502,502,502,502,502,502,118,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502,502},
/*13*/  {13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,13 ,124,13 ,13 ,13 ,13 ,507,13 ,503,13 },
/*14*/  {504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,504,125,504,504,504,504,504,504,504},
/*15*/  {505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,505,126,505,505,505,505,505,505,505},
/*15*/  {128,128,128,128,128,128,128,129,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128,128}
```

BNF

101 102
<valor_numerico>: = D^+ $(.D^+)$?

124
<valor_cadena>::= “ASCII”

212 213
<valor_log>::= true | false

118
<op_log>::= Not (!=)

126 125
<<op_log1>::= OR (||) | AND (&&)

112 113 117 115 116
<op_num>::= + | - | * | / | %

120 122 119 121 129
<op_rela>::= > | >= | < | <= | ==

<exp_num>::=<valor_nume>
| <valor_num><op_num><valor_num>
| <op_num><valor_num>
| <op_num><valor_num><op_num><valor_num>

<exp_rela>::= <valor_num><op_rela><valor_num>
| <id><op_rela><id>
| <id><op_rela><valor_num>
| <valor_num><op_rela><id>

<exp_log>::= <exp_rela>(<op_log> | <op_log1>)<exp_rela>

200 201 202 201
<go>::= package main fun main () <block>

<block>::= {<statement>}

<statement>::= <Statement_list>

209 210
<Statement_list>::= if (<exp_log>) {<statement>} <else>

210
else {<statement>}

211 100 123 104 100 104
for (<id> := <exp_num> ; <id> <op_rel><exp_num> ;
100
<id> (++ | - -))
215 214
{<statement>} (continue | break)

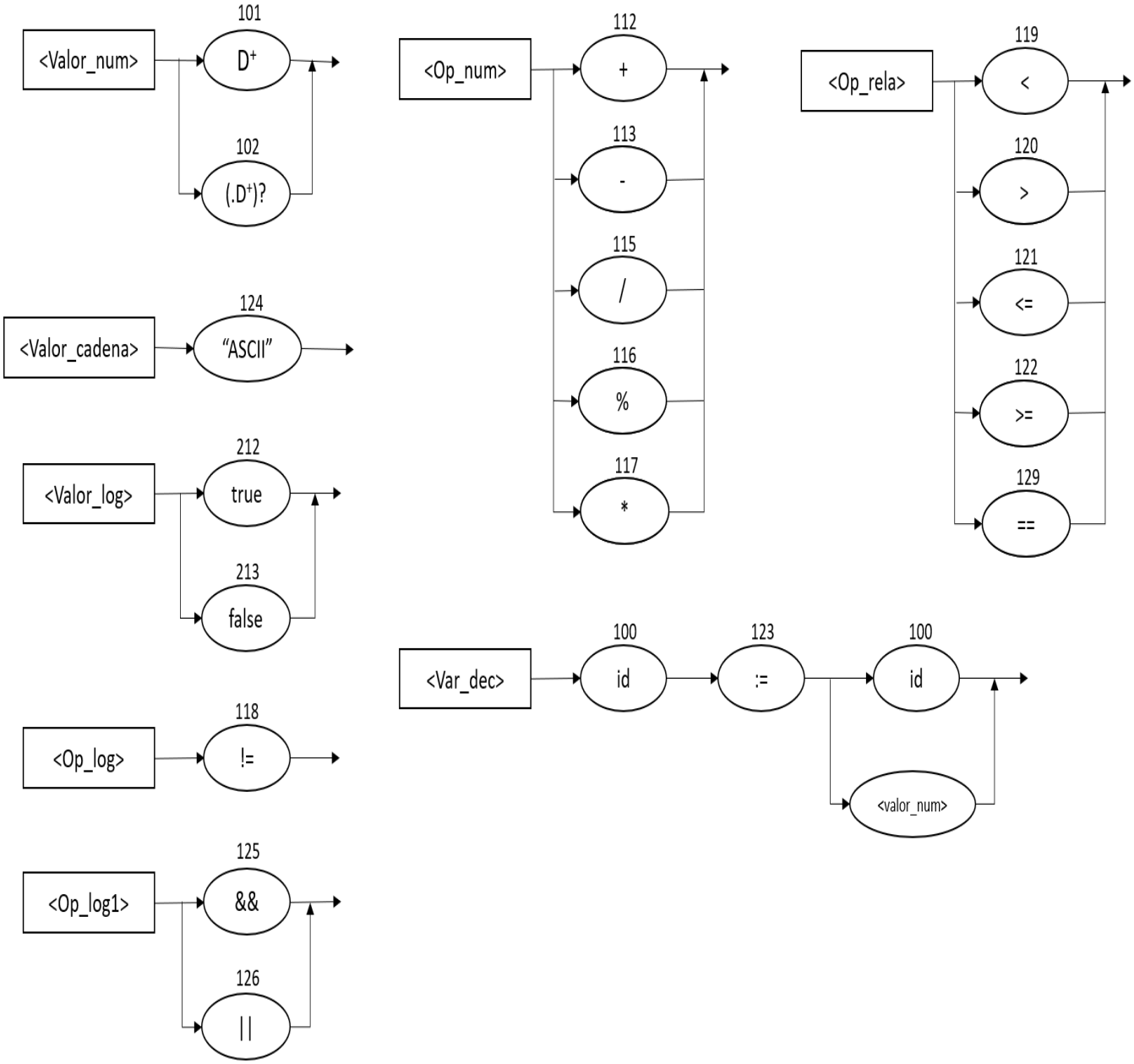
203 106 106 106
print (<valor_cadena>) | (<valor_cadena> (,<id>))

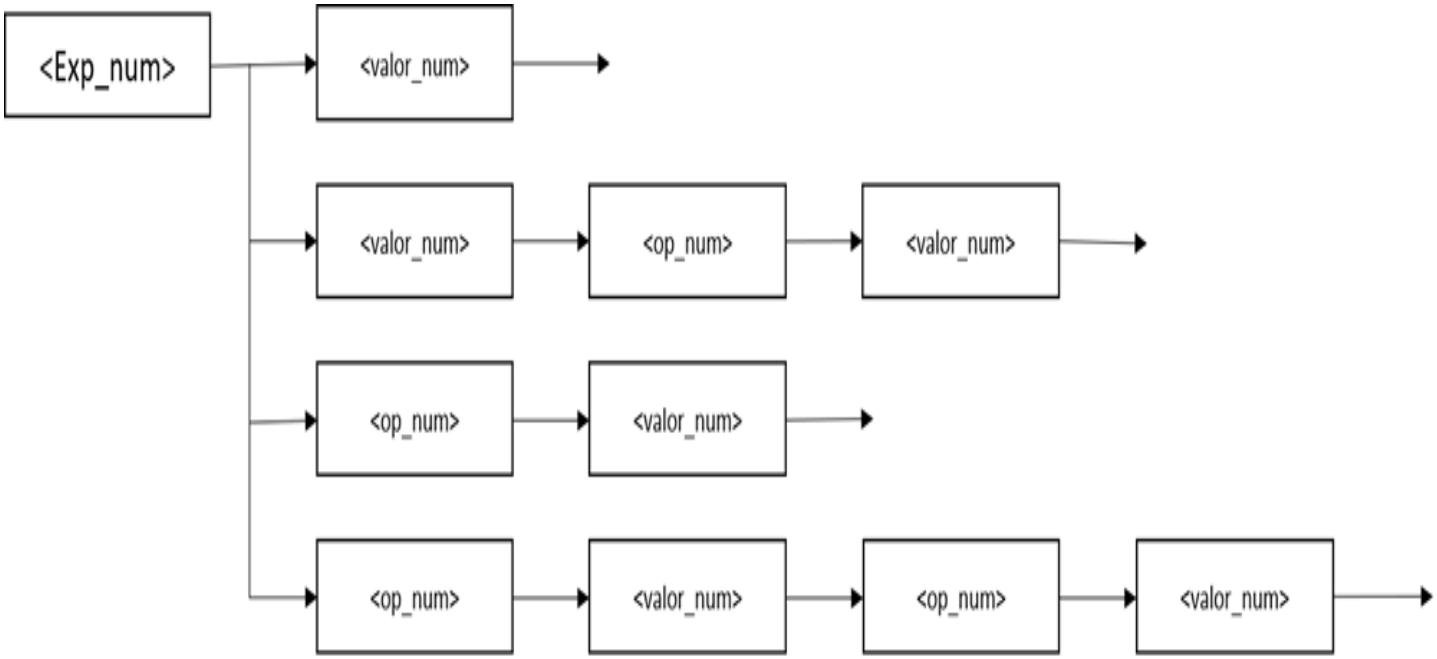
204 106 100 106
scan (<valor_cadena> (, <id>))

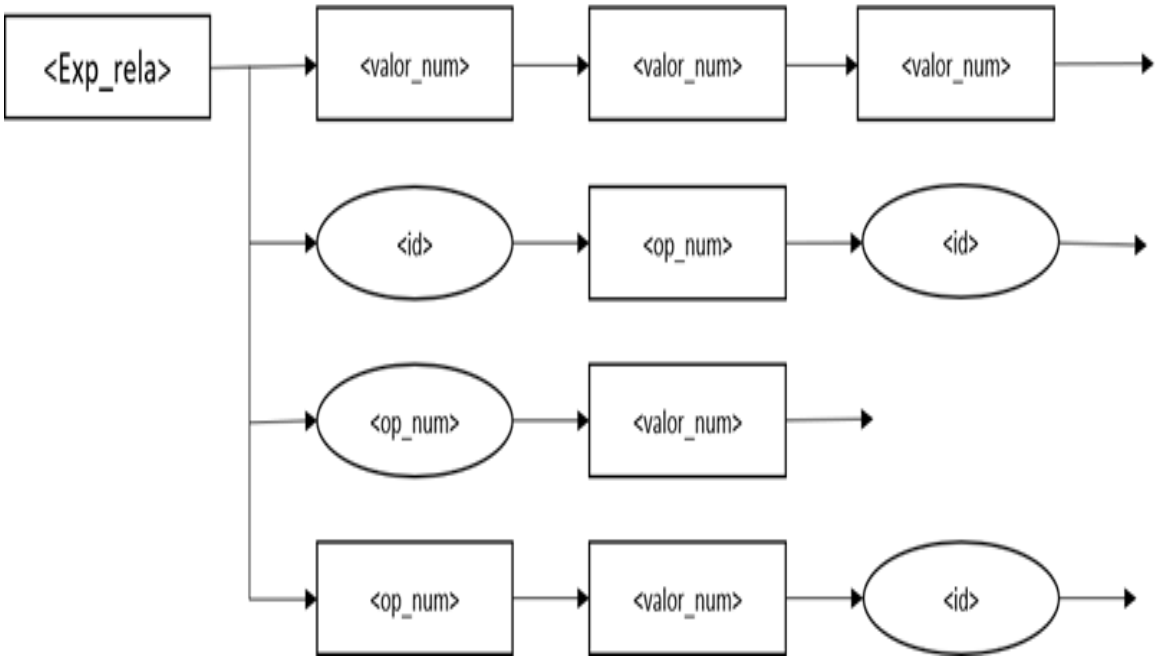
<id>::= <L (L | D)*>

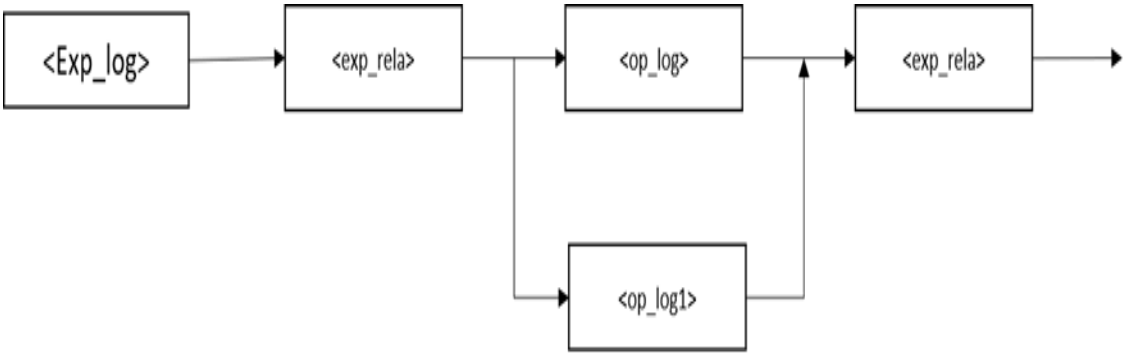
<var_dec>::= <id>:=(<id> | <valor_num>)

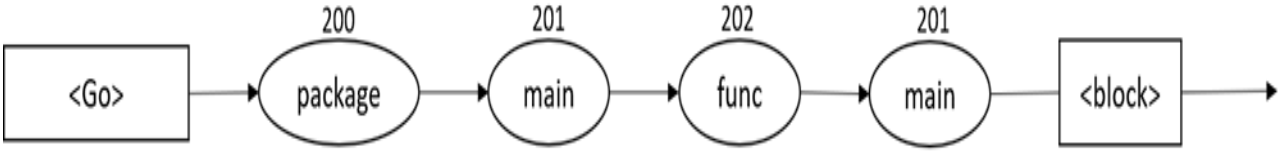
DIAGRAMA DE BLOQUES

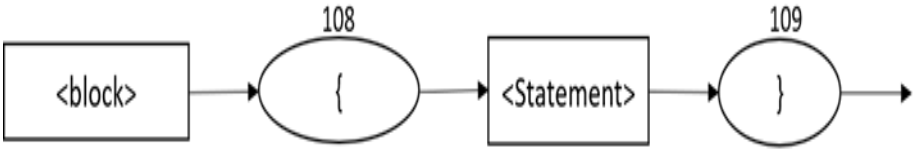


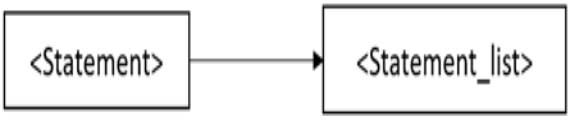

<Exp_num>
<valor_num>
<valor_num>
<op_num>
<valor_num>
<op_num>
<valor_num>
<op_num>
<valor_num>
<op_num>
<valor_num>

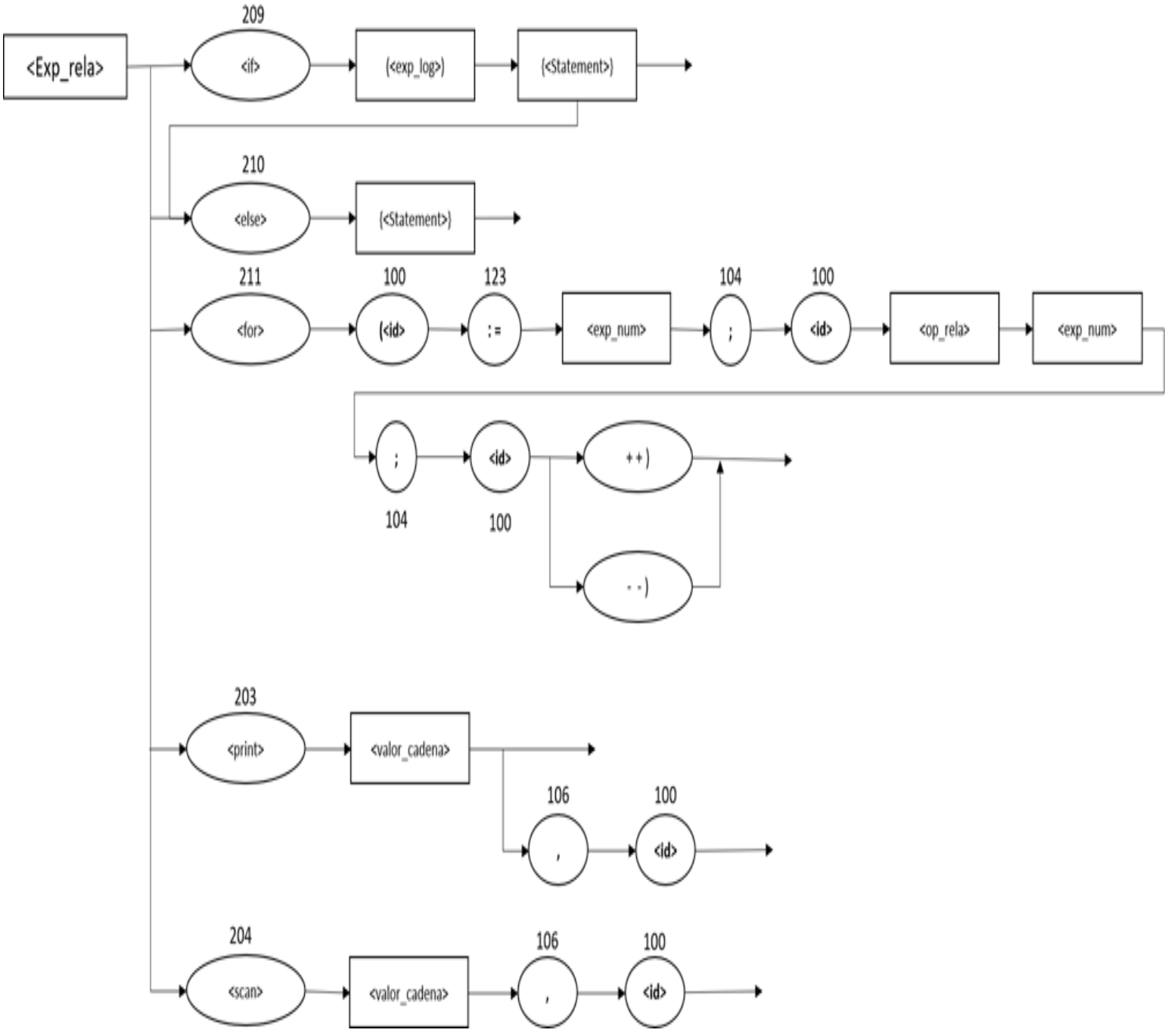

<Exp_rela>
<valor_num>
<valor_num>
<valor_num>
<id>
<op_num>
<id>
<op_num>
<valor_num>
<op_num>
<valor_num>
<id>


<Exp_log>
<exp_rela>
<op_log>
<exp_rela>
<op_log1>


<Go>
200
package
201
main
202
func
201
main
<block>


<block>
108
{
<Statement>
109
}


<Statement>
<Statement_list>


<Exp_rela>
209
<if>
(<exp_log>)
(<Statement>)
210
<else>
(<Statement>)
211
<for>
100
(<id>
123
:=
<exp_num>
104
;
100
<id>
<op_rela>
<exp_num>
;
104
<id>
100
++)
--)
203
<print>
<valor_cadena>
106
,
100
<id>
204
<scan>
<valor_cadena>
106
,
100
<id>