Memoria de la Práctica de Procesadores de Lenguajes

Diego José Abengózar Vilar, Alejandro García Castellanos,
Ignacio Javier Encinas Ramos

Grupo 82

January 18, 2020

Índice

1 Diseño del Analizador Léxico

1.1 Tokens

<PuntoComa, - >
<CorcheteAbrir, - >
<CorcheteCerrar, - >
<ID, posTS> (Identificador)
<ENT, valor> (Dato de tipo entero)
<CAD, lex> (Dato de tipo cadena)
<ParentesisCerrar, - >
<ParentesisAbrir, - >
<SUMA, - > (Operador suma)
<MENOR, - > (Operador lógico menor)
<NOT, - > (Operador lógico de negación)
<ASIG, - > (Operador de asignación)
<ASIGOR, - > (Asignación con o lógico)
<DEC, - > ("var")
<TipoVarENT, - > ("int")
<TipoVarLOG, - > ("boolean")
<TipoVarCAD, - > ("string")
<Print, - >
<Input, - >
<Coma, - >
<Return, ->
<DECFunc, - > ("function")
<IF, - >
<ELSE, - >

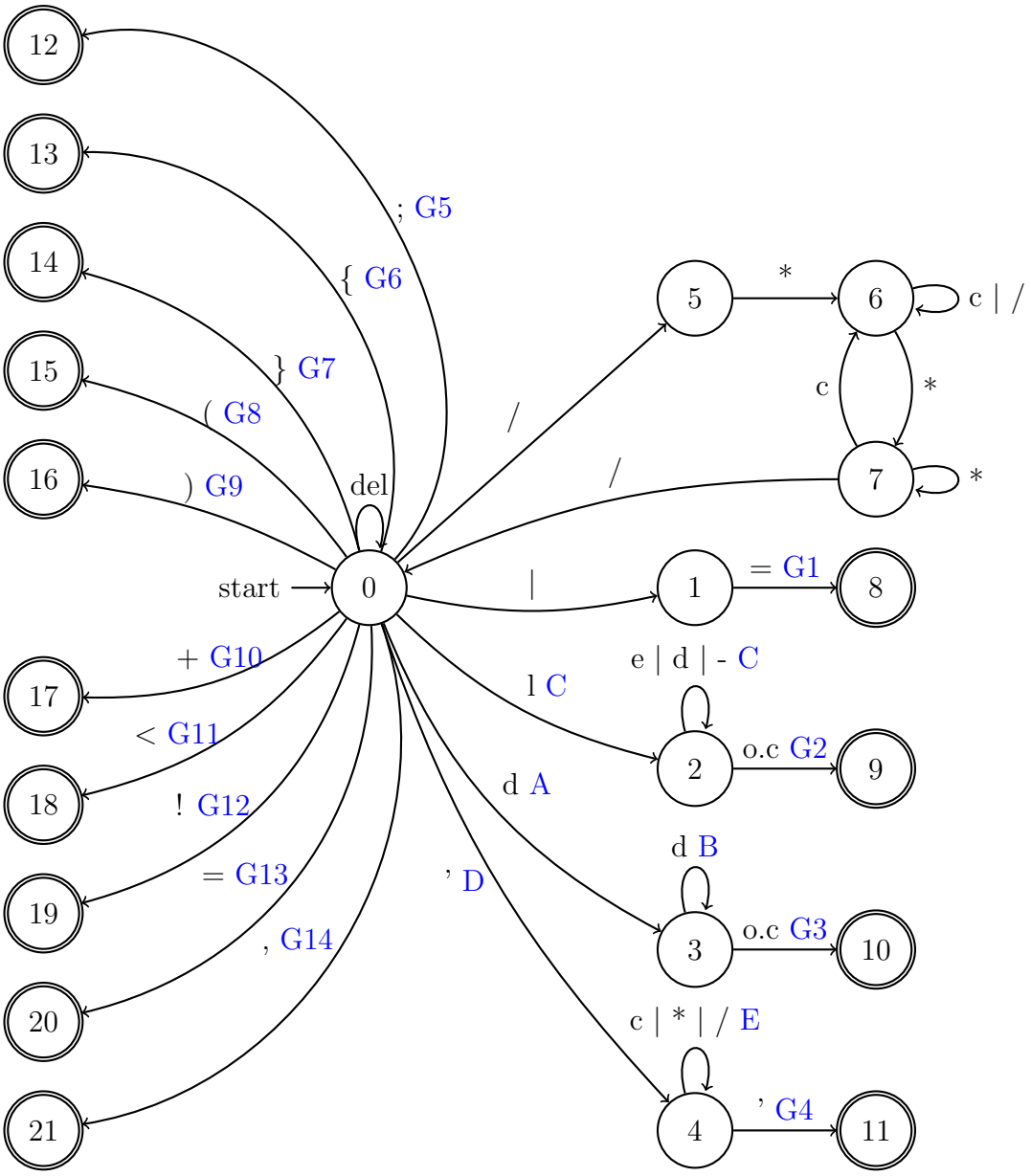
1.2 Gramática

$G(N, T, S, P)$

$S = A$
$N = \{ A, B, C, D, E, F, G, H \}$
$T = \{ del, ;, \{, \}, (,), +, <, !, =, ,, l, d, ', /, \_, *, c \}$

$P:$
$A \rightarrow delA \mid ; \mid \{ \mid \} \mid (\mid) \mid + \mid < \mid ! \mid = \mid ,$
$A \rightarrow |B \mid lC \mid dD \mid 'E \mid /F$
$B \rightarrow =$
$C \rightarrow lC \mid dD \mid \_C \mid \lambda$
$D \rightarrow dD \mid \lambda$
$E \rightarrow cE \mid *E \mid /E \mid '$
$F \rightarrow *G$
$G \rightarrow cG \mid /G \mid *H$
$H \rightarrow /A \mid cG \mid *H$
$Donde, c = T - \{*, /\}$

1.3 Autómata Finito Determinista



1.4 Acciones Semánticas

```
Lee ∀ transicion menos o.c

C:  CONCAT()

G1: GEN_TOKEN(ASIGOR, −)

G2: if(lex ∈ palRes) GEN_TOKEN(palRes, −)
else if(FlagDeclUso = Decl)
     if (estaEnTSActual(lex))
          Error("Variable ya declarada")
     else
          p = INSERTAR_TS(lex)
          GEN_TOKEN(ID, p)
else
     p = BUSCA_TS(lex)
     if (p = null) p = INSERTAR_TS(lex)
     GEN_TOKEN(ID, p)

A: num = valor(d)

B: num = num * 10 + valor(d)

D: cont = 0

E: cont = cont + 1
   CONCAT()

G3: if(num >= 2^15) Error("Numero se sale del rango")
     else GEN_TOKEN(ENT, num)

G4: if(num > 64) Error("Exceso de caracteres en la cadena")
      else GEN_TOKEN(CAD, lex)

G5: GEN_TOKEN(PuntoComa, −)

G6: GEN_TOKEN(CorcheteAbrir, −)

G7: GEN_TOKEN(CorcheteCerrar, −)

G8: GEN_TOKEN(ParentesisAbrir, −)

G9: GEN_TOKEN(ParentesisCerrar, −)

G10: GEN_TOKEN(SUMA, −)

G11: GEN_TOKEN(MENOR, −)

G12: GEN_TOKEN(NOT, −)
```

G_{13} : GEN_TOKEN(ASIG , −)

G_{14} : GEN_TOKEN(Coma , −)

Donde, palRes = {var, int, boolean, string, print, input, function, return, if, else}

1.5 Errores

Los errores que pueden ocurrir son errores de transiciones imprevistas, error de que un número esté fuera de rango y error de identificador ya declarado previamente (cuando el FlagDeclUso= DECL).

1.6 Matriz de Transiciones

MT_AFD	|	**letra**	**digito**	'	/	_	**carácter**	*	**delimitador**
→ 0	1 lee	2 C	3A	4 D	5 lee	-1 error	-1 error	-1 error	0 lee
1	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error
2	9 G2	2 C	2 C	9 G2	9 G2	2 C	9 G2	9 G2	9 G2
3	10 G3	10 G3	3 B	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3
4	4 E	4 E	4 E	11 G4	4 E	4 E	4 E	4 E	4 E
5	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	6 lee	-1 error
6	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	7 lee	6 lee
7	6 lee	6 lee	6 lee	6 lee	0 lee	6 lee	6 lee	7 lee	6 lee

MT_AFD	;	{	}	(	)	+	<	!	=	,
→ 0	12 G5	13 G6	14 G7	15 G8	16 G9	17 G10	18 G11	19 G12	20 G13	21 G14
1	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	8 G1	-1 error
2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2	9 G2
3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3	10 G3
4	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E	4 E
5	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error	-1 error
6	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee
7	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee	6 lee

2 Tabla de Símbolos: Estructura e implementación

Contiene la información de los identificadores, de los cuales se guardan los campos: lexema, tipo y desplazamiento. Para las funciones, además, se guardará el número de parámetros, su tipo, la forma de paso de parámetros y el tipo del valor de retorno.

La tabla de símbolos estará formada por dos matrices de tamaño dinámico; la primera contendran los indentificadores de ámbito global y la segunda del local. Así pues, esta segunda se creará al encontrar la declaración de una función y se borrará al acabar de ser declarada. También se utiliza un flag de declaración o uso (FlagDeclUso), un flag para saber cual es la tabla actual y dos más para el valor del desplazamiento en cada una de las tablas.

Sin embargo, en la implementación actual sólo se usa una tabla y siempre se supone que está el FlagDeclUso = Uso, pero en el caso de que no este declarada la variable se insertará en la tabla actual, ya que requerimos de la implementacion del Analizador

Semántico para poder saber cuando se cambia de ámbito y cuando se están declarando o usando identificadores. Así que, la acción semántica que genera los tokens de los identificadores quedaría temporalmente así:

```
$G_2$: if(lex $\in$ palRes)
          GEN_TOKEN(palRes, -)
       else if((p:= BUSCA_TS(lex))=NULL)
          p:=INSERTAR_TS(lex)
          GEN_TOKEN(ID, p)
```

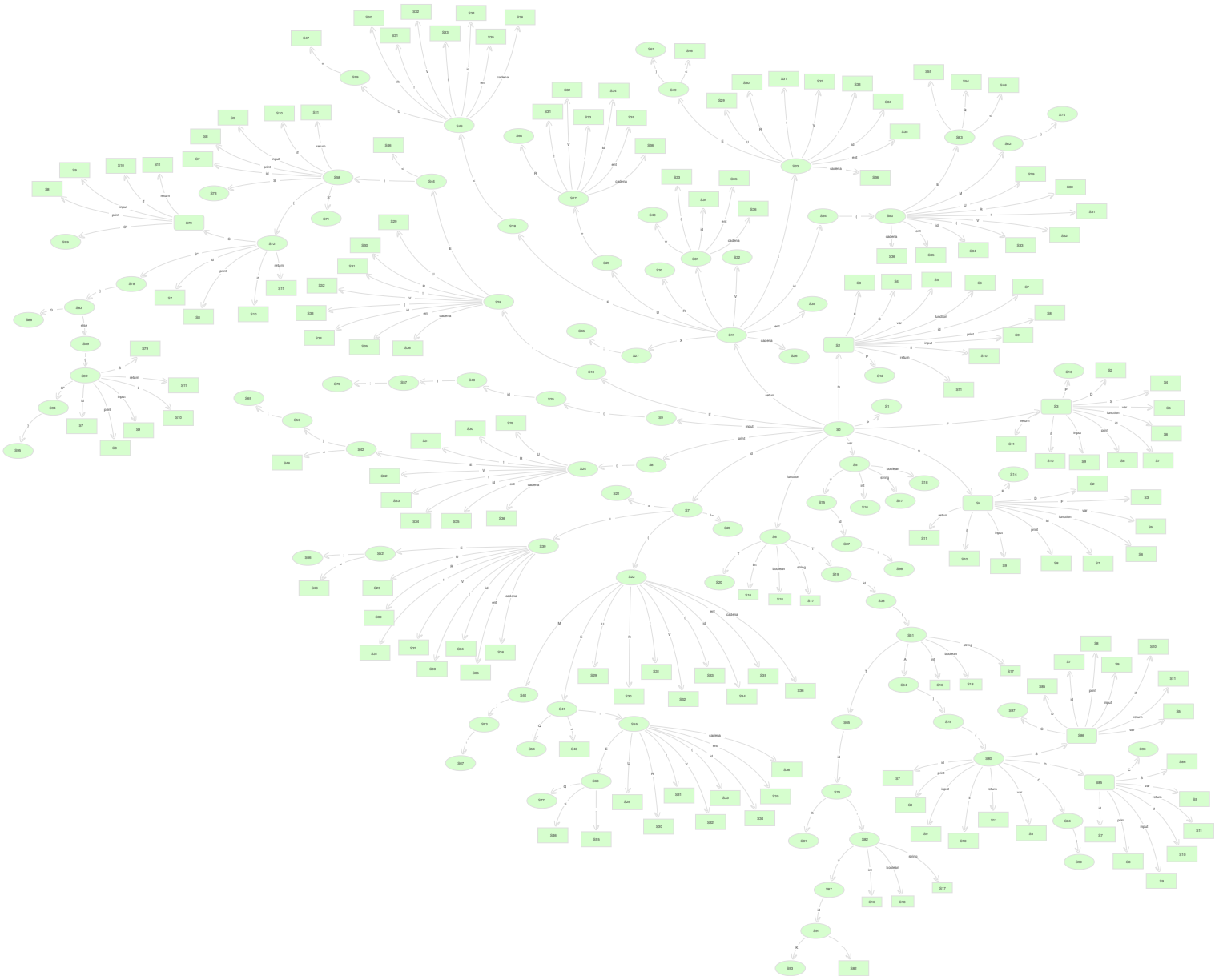
3 Diseño del Analizador Sintáctico

3.1 Gramática

$Terminales = \{\ ;\ \{\ \}\ id\ ent\ cadena\ (\ )\ +\ <\ !\ =\ |=\ var\ int\ boolean\ string\ print\ input\ ,\ return\ function\ if\ else\ \}$
$NoTerminales = \{\ P\ D\ T\ F\ T1\ A\ K\ C\ S\ L\ M\ Q\ S1\ G\ X\ E\ U\ R\ V\ S2\ \}$
$Axioma = P$
$Producciones = \{$
$P \rightarrow D\ P$
$P \rightarrow F\ P$
$P \rightarrow S\ P$
$D \rightarrow var\ T\ id\ ;$
$T \rightarrow int$
$T \rightarrow string$
$T \rightarrow boolean$
$F \rightarrow function\ T1\ id\ (\ A\ )\ \{\ C\ \}$
$T1 \rightarrow lambda$
$T1 \rightarrow T$
$A \rightarrow T\ id\ K$
$A \rightarrow lambda$
$K \rightarrow lambda$
$K \rightarrow ,\ T\ id\ K$
$C \rightarrow D\ C$
$C \rightarrow S\ C$
$C \rightarrow lambda$
$S \rightarrow id\ L\ E\ ;$
$S \rightarrow id\ (\ M\ )\ ;$
$S \rightarrow print\ (\ E\ )\ ;$
$S \rightarrow input\ (\ id\ )\ ;$
$S \rightarrow if\ (\ E\ )\ S1$
$S \rightarrow return\ X\ ;$
$L \rightarrow |=$
$L \rightarrow =$
$M \rightarrow E\ Q$
$M \rightarrow lambda$
$Q \rightarrow lambda$
$Q \rightarrow ,\ E\ Q$
$S1 \rightarrow \{\ S2\ \}\ G$
$S1 \rightarrow S$
$G \rightarrow else\ \{\ S2\ \}$
$G \rightarrow lambda$
$X \rightarrow E$
$X \rightarrow lambda$
$E \rightarrow E < U$
$E \rightarrow U$
$U \rightarrow U + R$
$U \rightarrow R$
$R \rightarrow !\ V$
$R \rightarrow V$
$V \rightarrow (\ E\ )$

$V \rightarrow id$
$V \rightarrow id \ (\ M \ )$
$V \rightarrow ent$
$V \rightarrow cadena$
$S2 \rightarrow S \ S2$
$S2 \rightarrow S$
$P \rightarrow lambda$
}

3.2 Autómata Reconocedor de Prefijos Viables[1]



[1]Los estados con forma de rectángulo redondeado son aquellos con bucles a si mismos. La etiqueta de dicha arista es la misma que la arista que conecta dicho estado y su antecedente

3.2.1 Estados del autómata

S_0={P1 →• P, P → • DP, P → • SP, P → • , D → • var T id;,
F → • function T1 id(A){C}, S → • id L E;, S → • id(M);
S → • print(E);, S → • input(id);, S → • if(E) S1,
S → • return X;}
S_1={P1 →P •}
S_2={P →D • P, P → • DP, P → • SP, P → • , D → • var T id;,
F → • function T1 id(A){C}, S → • id L E;, S → • id(M);
S → • print(E);, S → • input(id);, S → • if(E) S1,
S → • return X;}
S_3={P →F • P, P → • DP, P → • FP, P → • SP, P → •,
D → • var T id;,F → • function T1 id(A){C},
S → • id L E;, S → • id(M);, S → • print(E);,
S → • input(id);, S → • if(E) S1, S → • return X;}
S_4={P →S • P, P → • DP, P → • FP, P → • SP, P → •,
D → • var T id;, F → • function T1 id(A){C},
S → • id L E;, S → • id(M);, S → • print(E);,
S → • input(id);, S → • if(E) S1, S → • return X;}
S_5={D → var • T id;, T → • int , T → • string , T → • boolean}
S_6={F → function • T1 id(A){C}, T1 → •, T1 → • T,
T → • int , T → • string , T → • boolean}
S_7={S → id • L E;, S → id • (M);, L → |=, L → • =}
S_8={S → print • (E);}
S_9={S → input • (id);}
S_{10}={S → if • (E) S1}
S_{11}={S →return • X;, X → •, X → • E, E → • E < U, E → • U,
U → • U + R, U → • R, R → • ! V, R → • V, V → • (E),
V → • id , V → • id(M) , V → • ent , V → • cadena}
S_{12}={P →DP •}
S_{13}={P →FP •}
S_{14}={P →SP •}
S_{15}={D →var T • id;}
S_{16}={T →int •}
S_{17}={T →string •}
S_{18}={T →boolean •}
S_{19}={F →function T1 • id(A){C}}
S_{20}={T1 →T •}
S_{21}={L →= •}
S_{22}={S →id (•M);, M → • E Q, M → •, E → • E < U, E → • U,
U → • U + R, U → • R, R → • ! V, R → • V, V → • (E),
V → • id , V → • id(M) , V → • ent , V → • cadena}
S_{23}={L →|= •}
S_{24}={S →print (• E);, E → • E < U, E → • U, U → • U + R,
U → • R, R → • ! V, R → • V, V → • (E), V → • id ,
V → • id(M) , V → • ent , V → • cadena}
S_{25}={S →input (• id)}
S_{26}={S →if (• E) S1, E → • E < U, E → • U, U → • U + R,
U → • R, R → • ! V, R → • V, V → • (E), V → • id ,
V → • id(M) , V → • ent , V → • cadena}
S_{27}={S →return X • ;}
S_{28}={X →E •, E → E • < U}
S_{29}={E →U •, U → U • + R}
S_{30}={U →R •}
S_{31}={R →! • V, V → • (E), V → • id , V → • id(M) ,
V → • ent , V → • cadena}
S_{32}={R →V •}
S_{33}={V →(• E), E → • E < U, E → • U, U → • U + R,

U → • R, R → • ! V, R → • V, V → • (E), V → • id,
V → • id(M), V → • ent, V → • cadena}
S_{34}={V →id •, V → id • (M)}
S_{35}={V →ent •}
S_{36}={V →cadena •}
S_{37}={D →var T id •;}
S_{38}={F →function T1 id • (A){C}}
S_{39}={S →id L • E, E → • E < U, E → • U, U → • U + R,
U → • R, R → • ! V, R → • V, V → • (E), V → • id,
V → • id(M), V → • ent, V → • cadena}
S_{40}={S →id (M •);}
S_{41}={M →E • Q, E → E • < U, Q → •, Q → • ,EQ}
S_{42}={S →print(E •);, E → E • < U}
S_{43}={S →input(id •);}
S_{44}={S →if(E •) S1, E → E • < U}
S_{45}={S →return X; •}
S_{46}={E →E < • U, U → • R, U → • U + R, R→ • ! V, R → • V
V → • (E), V → • id, V → • id(M), V → • ent, V → • cadena}
S_{47}={U →U + • R, R → • ! V, V → • (E), V → • id(M),
V → • ent, V → • cadena}
S_{48}={R →! V •}
S_{49}={V →(E •), E → E • < U}
S_{50}={V →id(• M), M → • E Q, M → •, E → • E < U, E → • U,
U → • U + R, U → • R, R → • ! V, R → • V, V → • (E),
V → • id, V → • id(M), V → • ent, V → • cadena}
S_{51}={F →function T1 id(• A){C}, A → • T id K, A → •, T → • int,
T → • string, T → • boolean}
S_{52}={S →id L E • ;, E → E • < U}
S_{53}={S →id(M) • ;}
S_{54}={M →E Q •}
S_{55}={Q →, • E Q, E → • E < U, E → • U,
U → • U + R, U → • R, R → • ! V, R → • V, V → • (E),
V → • id, V → • id(M), V → • ent, V → • cadena}
S_{56}={S →print(E) • ;}
S_{57}={S →input(id) • ;}
S_{58}={S →if(E) • S1, S1 → • {S2}G, S1 → • S, S → • id L E;,
S → • id(M);, S → • print(E);, S → • input(id);,
S → • if(E)S1, S → • return X ;}
S_{59}={E →E < U •, U →U • + R}
S_{60}={U →U + R •}
S_{61}={V →(E) •}
S_{62}={V →id(M •)}
S_{63}={M →E • Q, E → E • < U, Q → •, Q → • ,EQ}
S_{64}={F →function T1 id(A •){C}}
S_{65}={A →T • id K}
S_{66}={S →id L E ; •}
S_{67}={S →id (M); •}
S_{68}={Q →,E • Q, E → E • < U, Q → •, Q → • ,EQ}
S_{69}={S →print(E); •}
S_{70}={S →input(id); •}
S_{71}={S →if(E) S1 •}
S_{72}={S1 →{• S2}G, S2 → • S S2, S2 → • S, S → • id L E;,
S → • id(M);, S → • print(E);, S → • input(id);,
S → • if(E)S1, S → • return X ;}
S_{73}={S1 →S •}
S_{74}={V →id(M) •}
S_{75}={F →function T1 id (K) • {C}}
S_{76}={A →T id •K , K → •, K → • , T id K}

S_{77}={Q → , E Q •}
S_{78}={S1 →{S2 • } G}
S_{79}={S2 →S • S2, S2 → S •, S2 → • S S2, S → • id L E;,
S → • id(M);, S → • print(E); S → • if(E)S1;,
S → • input(id);, S → return X;}
S_{80}={F →function T1 id (K) { • C}, C → • D C , C → • ,
D → • var T id ;, S → • id L E , S → • id (M);,
S → • print (E);, S → • input (id);, S → • if (E) S1,
S → • return X ;}
S_{81}={A →T id K •}
S_{82}={K → , • T id K, T → • int , T → • string , T → • boolean}
S_{83}={S1 →{S2} • G , G → • else {S2} , G → •}
S_{84}={F →function T1 id (K) {C • }}
S_{85}={C →D • C, C → • D C, C → • S C, C → •, D → • var T id ;,
S → • id L E ;, S → • id (M) ;, S → • print (E) ;,
S → • input (id) ;, S → • if (E) S1, S → • return X ;}
S_{86}={C →S • C, C → • D C, C → • S C, C → •, D → • var T id ;,
S → • id L E ;, S → • id (M) ;, S → • print (E) ;,
S → • input (id) ;, S → • if (E) S1, S → • return X ;}
S_{87}={K → , T • id K}
S_{88}={S1 →{S2} G •}
S_{89}={G →else • {S2}}
S_{90}={F →function T1 id (K) {C} •}
S_{91}={K → , T id • K, K → •, K → • , T id K}
S_{92}={G →else { • S2}, S2 → • S S2, S2 → • S, S → • id L E ;,
S → • id (M) ; , S → • print (E) ;, S → • input(id);,
S → • if (E) S1, S → • return X ;}
S_{93}={K → , T id K •}
S_{94}={G →else { S2 • }}
S_{95}={G →else { S2 } •}
S_{96}={C →D C •}
S_{97}={C →S C •}
S_{98}={D →var T id ; •}
S_{99}={S2 →S S2 •}

3.3 Conflictos

Como podemos observar en la tabla de decisión no hay ningún conflicto.

Los posibles conflictos son:

Reducción-Reducción

Podríamos ver como en los posibles estados con este conflicto, en nuestro caso ninguno, se verifica que

$\forall$ {A → α •, B → β •} $\subset S_x \Rightarrow$ Follow(A) $\cap$ Follow(B) = $\emptyset$ (Esto lo podemos observar al no tener dos entradas de reducción en la misma celda de cada fila de S_x)

Reducción-Desplazamiento

Podemos ver como en los posibles estados con este conflicto, S_0, S_2, S_3, S_4, S_6, S_{11}, S_{22}, S_{28}, S_{29}, S_{34}, S_{41}, S_{50}, S_{51}, S_{59}, S_{63}, S_{68}, S_{76}, S_{79}, S_{80}, S_{83}, S_{85}, S_{86}, S_{91}, se verifica

$\forall$ {A → α • b γ, C → β •} $\subset S_x \Rightarrow$ b $\notin$ Follow(C) (Esto lo podemos observar al no tener una entrada de desplazamiento y otra de reducción en la misma celda de cada fila de S_x)

Por ejemplo, para los estados S_0, S_2, S_3, S_4: {var, function, id, print, input, if, return} $\notin$ Follow(P) = { $ }
En el estado S_6: {int, string, boolean} $\notin$ Follow(T1) = { id }
Y así sucesivamente con el resto de estados.

3.4 Errores

En las celdas vacías de cada fila se lanzan los siguientes errores:

S_0, S_4, S_7, S_{14}, S_{85}, S_{86}, S_{96}, S_{97}: Error 1: "Sentencia no válida"

S_1: Error -1 : "No se pudo derivar la raíz"

S_2, S_5, S_{12}, S_{15}, S_{37}, S_{98}: Error 2: "Declaración incorrecta de variable"

S_3, S_6, S_{13}, S_{19}, S_{38}, S_{51}, S_{64}, S_{65}, S_{75}, S_{76}, S_{80}, S_{81}, S_{82}, S_{84}, S_{87}, S_{90}, S_{91}, S_{93}: Error 3: "Declaración incorrecta de función"

S_8, S_{24}, S_{42}, S_{56}, S_{69}: Error 4: "Sentencia print incorrecta"

S_9, S_{25}, S_{43}, S_{57}, S_{70}: Error 5: "Sentencia input incorrecta"

S_{10}, S_{26}, S_{44}, S_{58}, S_{71}, S_{73}: Error 6: "Sentencia condicional simple incorrecta"

S_{11}, S_{27}, S_{28}, S_{45}: Error 7: "Sentencia return incorrecta"

S_{16}, S_{17}, S_{18}, S_{20}: Error 8: "Tipo incorrecto"

S_{21}, S_{23}, S_{39}, S_{52}, S_{66}: Error 9: "Asignación incorrecta"

S_{22}, S_{40}, S_{41}, S_{53}, S_{54}, S_{55}, S_{63}, S_{67}, S_{68}, S_{77}: Error 10: "Llamada a función incorrecta"

S_{29}, S_{30}, S_{31}, S_{32}, S_{33}, S_{34}, S_{35}, S_{36}, S_{46}, S_{47}, S_{48}, S_{49}, S_{50}, S_{59}, S_{60}, S_{61}, S_{62}, S_{74}: Error 11: "Expresión incorrecta"

S_{72}, S_{78}, S_{79}, S_{83}, S_{88}, S_{89}, S_{92}, S_{94}, S_{95}, S_{99} : Error 12 "Sentencia condicional compuesta incorrecta"

3.5 Tabla de Decisión

	;	{	}	id	entero	cadena	(	)	+	<	!	=	|=	var	int	boolean	string	print	input	,	return	function	if	else	$	P	D	T	F	T'	A	K	C	S	L	M	Q	S'	G	X	E	U	R	V	S"
S0				d7										d5				d8	d9		d11	d6	d10		r49	1	2		3				4													
S1																									a																					
S2				d7										d5				d8	d9		d11	d6	d10		r49	12	2		3				4													
S3				d7										d5				d8	d9		d11	d6	d10		r49	13	2		3				4													
S4				d7										d5				d8	d9		d11	d6	d10		r49	14	2		3				4													
S5															d16	d18	d17										15																			
S6				r9											d16	d18	d17										20		19																	
S7							d22					d21	d23																					39											
S8							d24																																						
S9							d25																																						
S10							d26																																						
S11	r35			d34	d35	d36	d33				d31																													27	28	29	30	32	
S12																									r1																				
S13																									r2																				
S14																									r3																				
S15				d37																																								
S16				r5																																								
S17				r6																																								
S18				r7																																								
S19				d38																																								
S20				r10																																								
S21				r25	r25	r25	r25				r25																																		
S22				d34	d35	d36	d33	r27			d31																														41	29	30	32	
S23				r24	r24	r24	r24				r24																																		
S24				d34	d35	d36	d33				d31																														42	29	30	32	
S25				d43																																								
S26				d34	d35	d36	d33				d31																														44	29	30	32	
S27	d45																																											
S28	r34									d46																																		
S29	r37							r37	d47	r37											r37																								
S30	r39							r39	r39	r39											r39																								
S31				d34	d35	d36	d33																																					48	
S32	r41							r41	r41	r41											r41																								
S33				d34	d35	d36	d33				d31																														49	29	30	32	
S34	r43						d50	r43	r43	r43											r43																								
S35	r45							r45	r45	r45											r45																								
S36	r46							r46	r46	r46											r46																								
S37	d98																																											
S38							d51																																					
S39				d34	d35	d36	d33				d31																														52	29	30	32	
S40								d53																																				
S41								r28		d46											d55																		54						
S42								d56		d46																																				
S43								d57																																				
S44								d58		d46																																				
S45		r23	r23											r23				r23	r23		r23	r23	r23		r23																			
S46				d34	d35	d36	d33				d31																															59	30	32	
S47				d34	d35	d36	d33				d31																																60	32	
S48	r40							r40	r40	r40											r40																								
S49								d61		d46																																				

	;	{	}	id	entero	cadena	(	)	+	<	!	=	|=	var	int	boolean	string	print	input	,	return	function	if	else	$	P	D	T	F	T'	A	K	C	S	L	M	Q	S'	G	X	E	U	R	V	S"
S50				d34	d35	d36	d33	r27			d31																										62					63	29	30	32	
S51								r12							d16	d18	d17												65		64															
S52	d66									d46																																				
S53	d67																																													
S54								r26																																						
S55				d34	d35	d36	d33				d31																															68	29	30	32	
S56	d69																																													
S57	d70																																													
S58		d72		d7															d8	d9		d11		d10										73				71							
S59	r36							r36	d47	r36											r36																									
S60	r38							r38	r38	r38											r38																									
S61	r42							r42	r42	r42											r42																									
S62								d74																																						
S63								r28		d46											d55																54									
S64								d75																																						
S65				d76																																										
S66			r18	r18										r18				r18	r18		r18	r18	r18		r18																			
S67			r19	r19										r19				r19	r19		r19	r19	r19		r19																			
S68								r28		d46											d55																77									
S69			r20	r20										r20				r20	r20		r20	r20	r20		r20																			
S70			r21	r21										r21				r21	r21		r21	r21	r21		r21																			
S71			r22	r22										r22				r22	r22		r22	r22	r22		r22																			
S72				d7														d8	d9		d11		d10										79											78
S73			r31	r31										r31				r31	r31		r31	r31	r31		r31																			
S74	r44							r44	r44	r44											r44																									
S75		d80																																												
S76								r13													d82																									
S77								r29																																						
S78			d83																																											
S79			r48	d7														d8	d9		d11		d10										79											99
S80			r17	d7										d5				d8	d9		d11		d10				85						84	86										
S81								r11																																						
S82															d16	d18	d17											87																	
S83			r33	r33										r33				r33	r33		r33	r33	r33	d89	r33														88						
S84			d90																																											
S85			r17	d7										d5				d8	d9		d11		d10										96	86										
S86			r17	d7										d5				d8	d9		d11		d10				85						97	86										
S87				d91																																										
S88			r30	r30										r30				r30	r30		r30	r30	r30		r30																			
S89		d92																																												
S90				r8										r8				r8	r8		r8	r8	r8		r8																			
S91								r13													d82														93										
S92				d7														d8	d9		d11		d10										79											94
S93								r14																																						
S94			d95																																											
S95			r32	r32										r32				r32	r32		r32	r32	r32		r32																			
S96			r15																																											
S97			r16																																											
S98			r4	r4										r4				r4	r4		r4	r4	r4		r4																			
S99			r47																																											

4 Anexo de Pruebas Semántico

Error 1:

```
1  var int a;
2  var int b;
3  /*65 caracteres*/
4  cadena = 'aaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaaa';
5  a = 3;
6  b = a;
7  var boolean c;
8  c = a < b;
9  if (c) {
10  b = 1;
11 }
12 c = b < a;
13 if (c) b = 4;
14 a = a + b;
15 print (a);
16 print (b);
```

```
> Error Lexico: Exceso de caracteres en la cadena.  Linea: 4
```

Error 2:

```
1 var int a;
2 var int b;
3 a = 33333;
4 b = a;
5  if (a  <  b) b  =  1;
6 if (b  <  a) b  =  8;
7  a = a  +  b;
8 print (a);
9 print (b);
```

```
> Error Lexico: Numero fuera de rango.  Linea: 3
```

Error 3:

```
1  var string texto; /*Comentario bueno*/
2   function imprime (string msg)
3  {
4    print (msg);
5  }
6  / Comentario malo*/
7  function pideTexto ()
8  {
9    print ( 'Introduce un texto' );
10   input (texto);
11 }
12  pideTexto();
13 var string textoAux;
14 textoAux = texto;
15 imprime (textoAux);
```

```
> Error Lexico: Transicion no prevista.  Linea: 6
```

Prueba 1 Correcta:

```
1  var int a;
2  var int b;
3  a = 3;
4  b = a;
5   var boolean c;
6  c = a  <  b;
7  if (c) b  =  1;
8  c = b  <  a;
9  if (c) b  =  4;
10  a = a  +  b;
11 print (a);
12 print (b);
```

Tokens:

<DEC, >
<TipoVarENT, >
<ID, 0>
<PuntoComa, >
<DEC, >
<TipoVarENT, >
<ID, 1>
<PuntoComa, >
<ID, 0>
<ASIG, >
<ENT, 3>
<PuntoComa, >
<ID, 1>
<ASIG, >
<ID, 0>
<PuntoComa, >
<DEC, >
<TipoVarLOG, >
<ID, 2>
<PuntoComa, >
<ID, 2>
<ASIG, >
<ID, 0>
<MENOR, >
<ID, 1>
<PuntoComa, >
<IF, >
<ParentesisAbrir, >
<ID, 2>
<ParentesisCerrar, >
<ID, 1>
<ASIG, >
<ENT, 1>
<PuntoComa, >

<ID, 2>
<ASIG, >
<ID, 1>
<MENOR, >
<ID, 0>
<PuntoComa, >
<IF, >
<ParentesisAbrir, >
<ID, 2>
<ParentesisCerrar, >
<ID, 1>
<ASIG, >
<ENT, 4>
<PuntoComa, >
<ID, 0>
<ASIG, >
<ID, 0>
<SUMA, >
<ID, 1>
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<ID, 0>
<ParentesisCerrar, >
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<ID, 1>
<ParentesisCerrar, >
<PuntoComa, >

Tabla de símbolos:

Tabla Simbolos #1:

* LEXEMA: 'a'

* LEXEMA: 'b'

* LEXEMA: 'c'

Prueba 2 Correcta:

```
1 var boolean booleano;
2 function boolean bisiesto (int a)
3 { var int bis;
4   print ('Es bisiesto?');
5   input(bis);
6   return (!(a + 4 < 0));
7 }
8 function int dias (int m, int a)
```

```
9  {
10   var int dd;
11   print ('di cuantos dias tiene el mes ');
12   print (m);
13   input(dd);
14   if (bisiesto(a)) dd = dd + 1;
15   return dd;
16 }
17 var boolean zzz;
```

Tokens:

<DEC, >
<TipoVarLOG, >
<ID, 0>
<PuntoComa, >
<DECFunc, >
<TipoVarLOG, >
<ID, 1>
<ParentesisAbrir, >
<TipoVarENT, >
<ID, 2>
<ParentesisCerrar, >
<CorcheteAbrir, >
<DEC, >
<TipoVarENT, >
<ID, 3>
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<CAD, "Es bisiesto?">
<ParentesisCerrar, >
<PuntoComa, >
<Input, >
<ParentesisAbrir, >
<ID, 3>
<ParentesisCerrar, >
<PuntoComa, >
<Return, >
<ParentesisAbrir, >
<NOT, >
<ParentesisAbrir, >
<ID, 2>
<SUMA, >
<ENT, 4>
<MENOR, >
<ENT, 0>
<ParentesisCerrar, >
<ParentesisCerrar, >
<PuntoComa, >

<CorcheteCerrar , >
<DECFunc, >
<TipoVarENT, >
<ID, 4>
<ParentesisAbrir , >
<TipoVarENT, >
<ID, 5>
<Coma, >
<TipoVarENT, >
<ID, 2>
<ParentesisCerrar , >
<CorcheteAbrir , >
<DEC, >
<TipoVarENT, >
<ID, 6>
<PuntoComa, >
<Print , >
<ParentesisAbrir , >
<CAD, "di cuantos dias tiene el mes ">
<ParentesisCerrar , >
<PuntoComa, >
<Print , >
<ParentesisAbrir , >
<ID, 5>
<ParentesisCerrar , >
<PuntoComa, >
<Input , >
<ParentesisAbrir , >
<ID, 6>
<ParentesisCerrar , >
<PuntoComa, >
<IF , >
<ParentesisAbrir , >
<ID, 1>
<ParentesisAbrir , >
<ID, 2>
<ParentesisCerrar , >
<ParentesisCerrar , >
<ID, 6>
<ASIG, >
<ID, 6>
<SUMA, >
<ENT, 1>
<PuntoComa, >
<Return , >
<ID, 6>
<PuntoComa, >
<CorcheteCerrar , >

<DEC, >
<TipoVarLOG, >
<ID, 7>
<PuntoComa, >

Tabla de símbolos:

Tabla Simbolos #1:

* LEXEMA: 'booleano'
* LEXEMA: 'bisiesto'
* LEXEMA: 'a'
* LEXEMA: 'bis'
* LEXEMA: 'dias'
* LEXEMA: 'm'
* LEXEMA: 'dd'
* LEXEMA: 'zzz'

Prueba 3 Correcta:

```
1  var int a;
2  var int b;
3  var int c;
4  print ('Introduce el primer operando');
5  input (a);
6  print ('Introduce el segundo operando');
7  input (b);
8  function int suma (int num1, int num2)
9  {
10   var int res;
11   res = num1+num2;
12   return res;
13 }
14 c = suma (a, b);
15 print (c);
```

Tokens:

<DEC, >
<TipoVarENT, >
<ID, 0>
<PuntoComa, >
<DEC, >
<TipoVarENT, >
<ID, 1>
<PuntoComa, >

<DEC, >
<TipoVarENT, >
<ID, 2>
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<CAD, "Introduce el primer operando">
<ParentesisCerrar, >
<PuntoComa, >
<Input, >
<ParentesisAbrir, >
<ID, 0>
<ParentesisCerrar, >
<PuntoComa, >
<Print, >
<ParentesisAbrir, >
<CAD, "Introduce el segundo operando">
<ParentesisCerrar, >
<PuntoComa, >
<Input, >
<ParentesisAbrir, >
<ID, 1>
<ParentesisCerrar, >
<PuntoComa, >
<DECFunc, >
<TipoVarENT, >
<ID, 3>
<ParentesisAbrir, >
<TipoVarENT, >
<ID, 4>
<Coma, >
<TipoVarENT, >
<ID, 5>
<ParentesisCerrar, >
<CorcheteAbrir, >
<DEC, >
<TipoVarENT, >
<ID, 6>
<PuntoComa, >
<ID, 6>
<ASIG, >
<ID, 4>
<SUMA, >
<ID, 5>
<PuntoComa, >
<Return, >
<ID, 6>
<PuntoComa, >

```
<CorcheteCerrar , >
<ID, 2>
<ASIG, >
<ID, 3>
<ParentesisAbrir , >
<ID, 0>
<Coma, >
<ID, 1>
<ParentesisCerrar , >
<PuntoComa, >
<Print , >
<ParentesisAbrir , >
<ID, 2>
<ParentesisCerrar , >
<PuntoComa, >
```

Tabla de símbolos:

```
Tabla Simbolos #1:

 * LEXEMA: 'a'

 * LEXEMA: 'b'

 * LEXEMA: 'c'

 * LEXEMA: 'suma'

 * LEXEMA: 'num1'

 * LEXEMA: 'num2'

 * LEXEMA: 'res'
```

5 Anexo de Pruebas Sintáctico

Error 1:

```
1  var int a;
2  var b;
3  a = 3;
4  b = a;
5  if (a < b) b = 1;
6  if (b < a) b = 8;
7  a = a + b;
8  print (a);
9  print (b);
```

```
> Error Sintactico: Declaracion incorrecta de variable.  Linea: 2
```

Error 2:

```
1  var string texto;
2  function pideTexto ()
3  {
4    print ('Introduce un texto');
5    input (texto);
6  }
7  function imprime (string msg,)
8  {
9    print (msg);
10 }
11 pideTexto();
12 var string textoAux;
13 textoAux = texto;
14 imprime (textoAux);
```

```
> Error Sintactico: Declaracion incorrecta de funcion.  Linea: 7
```

Error 3:

```
1  var int a;
2  var int b;
3  a = 3;
4  b = a;
5  var boolean c;
6  c = a < b;
7  if (c) {
8   b = 1;
9  } else {
10 c = b < a;
11 if (c) b = 4;
12 a = a + b;
13 print (a);
14 print (b);
```

```
> Error Sintactico: Sentencia condicional compuesta incorrecta.  Linea: 14
```

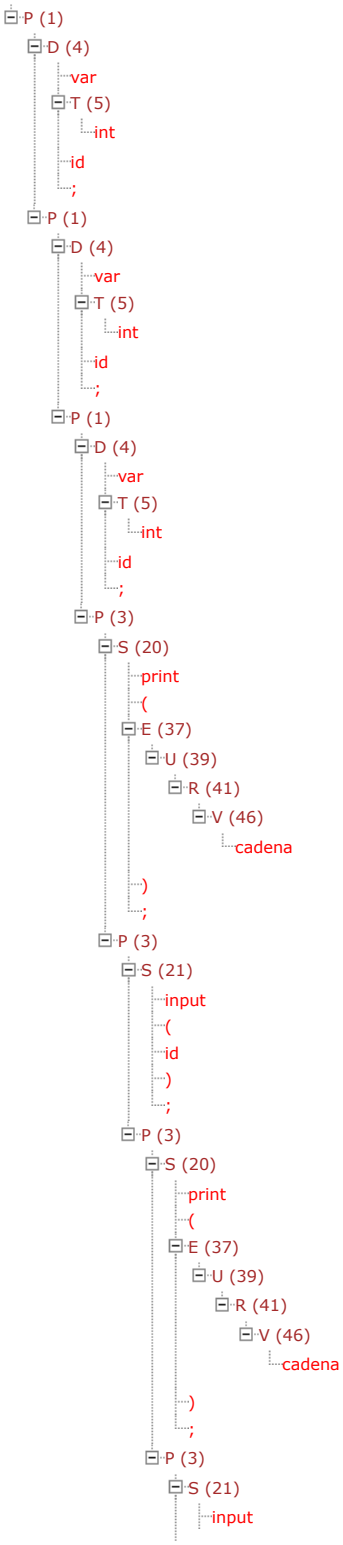
Prueba 1 Correcta:

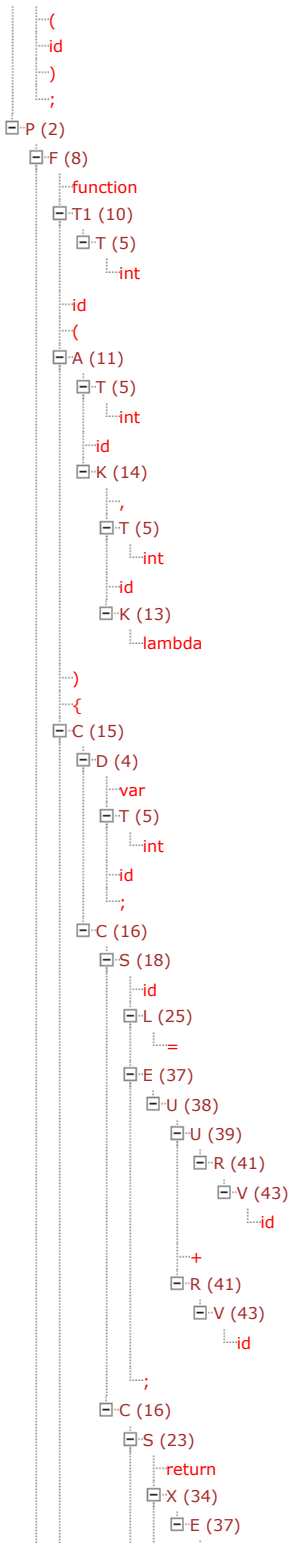
```
1  var int a;
2  var int b;
3  var int c;
4  print ('Introduce el primer operando');
5  input (a);
6  print ('Introduce el segundo operando');
7  input (b);
8  function int suma (int num1, int num2)
9  {
10   var int res;
11   res = num1+num2;
12   return res;
13 }
14 c = suma (a, b);
15 print (c);
```

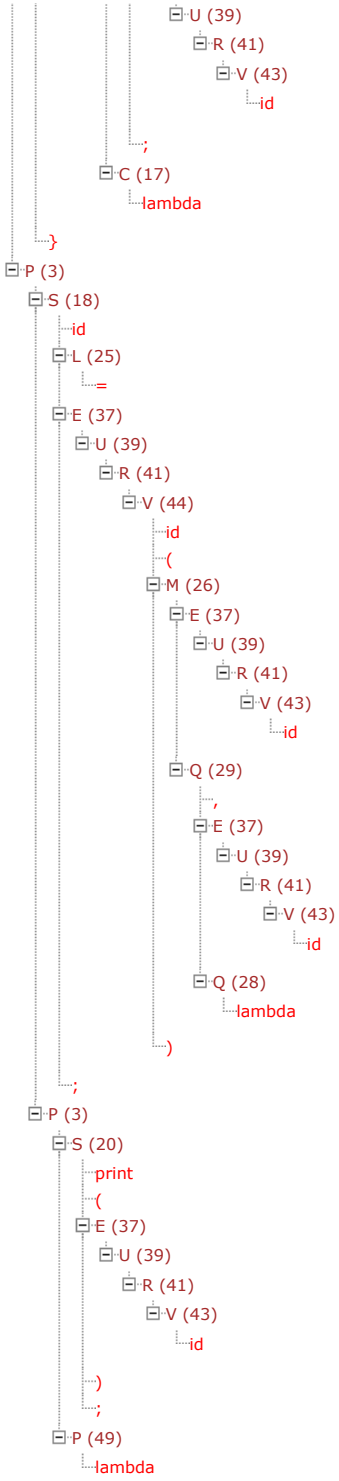
Parse a Derechas:

A 5 4 5 4 5 4 46 41 39 37 20 21 46 41 39 37 20 21 5 10 5 5 13 14 11 5 4 25 43 41 39 43 41 38 37 18 43 41 39 37 34 23 17 16 16 15 8 25 43 41 39 37 43 41 39 37 28 29 26 44 41 39 37 18 43 41 39 37 20 49 3 3 2 3 3 3 3 1 1 1

Árbol sintáctico:




(
id
)
;
P (2)
F (8)
function
T1 (10)
T (5)
int
id
(
A (11)
T (5)
int
id
K (14)
,
T (5)
int
id
K (13)
lambda
)
{
C (15)
D (4)
var
T (5)
int
id
;
C (16)
S (18)
id
L (25)
=
E (37)
U (38)
U (39)
R (41)
V (43)
id
+
R (41)
V (43)
id
;
C (16)
S (23)
return
X (34)
E (37)


U (39)
R (41)
V (43)
id
;
C (17)
lambda
}
P (3)
S (18)
id
L (25)
=
E (37)
U (39)
R (41)
V (44)
id
(
M (26)
E (37)
U (39)
R (41)
V (43)
id
Q (29)
,
E (37)
U (39)
R (41)
V (43)
id
Q (28)
lambda
)
;
P (3)
S (20)
print
(
E (37)
U (39)
R (41)
V (43)
id
)
;
P (49)
lambda

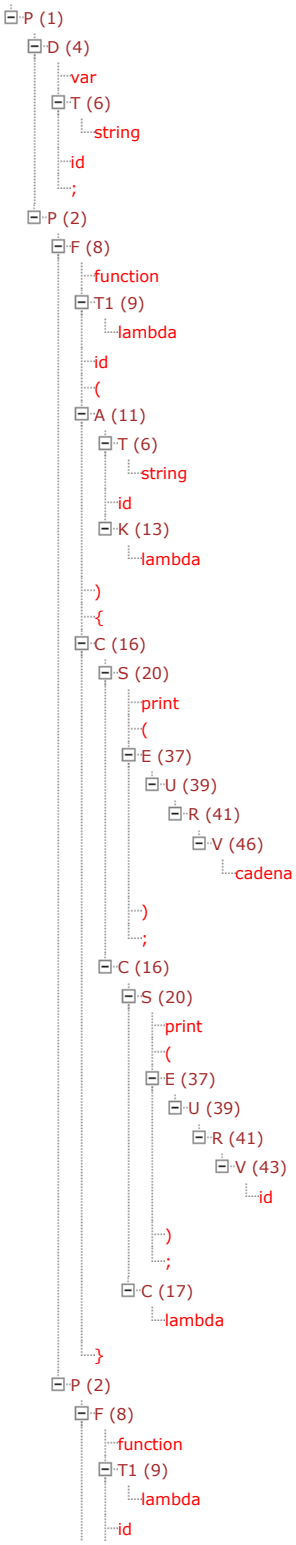
Prueba 2 Correcta:

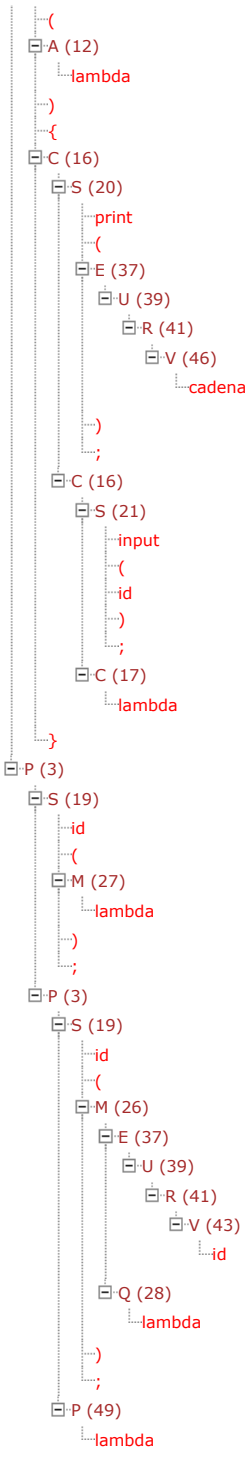
```
1  var string texto;
2  function imprime (string msg)
3  {
4    print ('Mensage introducido:');
5    print (msg);
6  }
7  function pideTexto ()
8  {
9    print ('Introduce un texto');
10   input (texto);
11 }
12 pideTexto();
13 imprime (texto);
```

Parse a Derechas:

A 6 4 9 6 13 11 46 41 39 37 20 43 41 39 37 20 17 16 16 8 9 12 46 41 39 37 20 21 17 16 16 8 27 19 43 41 39 37 28 26 19 49 3 3 2 2 1

Árbol sintáctico:




(
A (12)
lambda
)
{
C (16)
S (20)
print
(
E (37)
U (39)
R (41)
V (46)
cadena
)
;
C (16)
S (21)
input
(
id
)
;
C (17)
lambda
}
P (3)
S (19)
id
(
M (27)
lambda
)
;
P (3)
S (19)
id
(
M (26)
E (37)
U (39)
R (41)
V (43)
id
Q (28)
lambda
)
;
P (49)
lambda

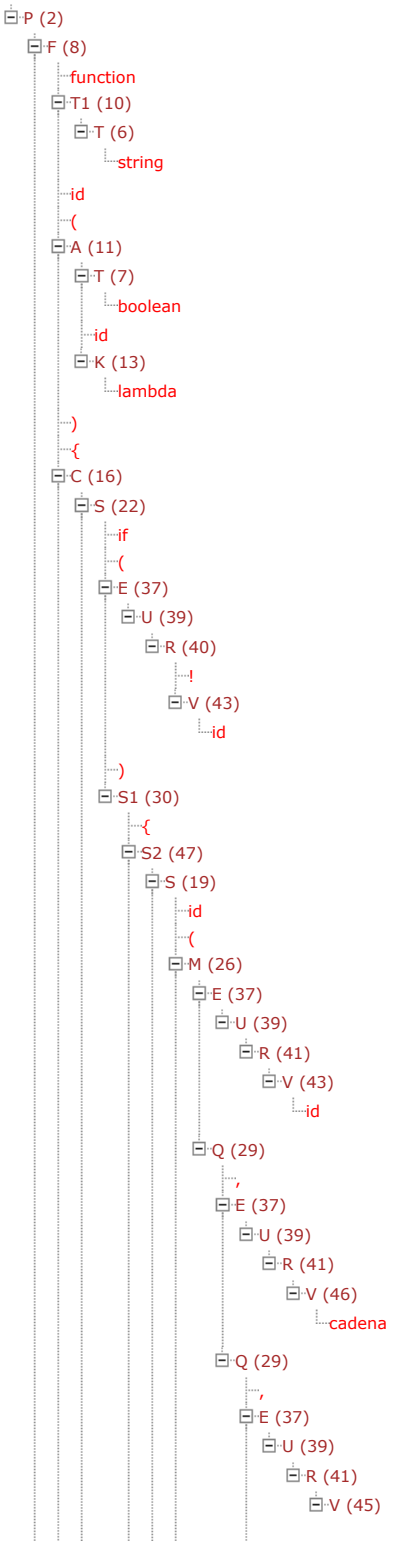
Prueba 3 Correcta:

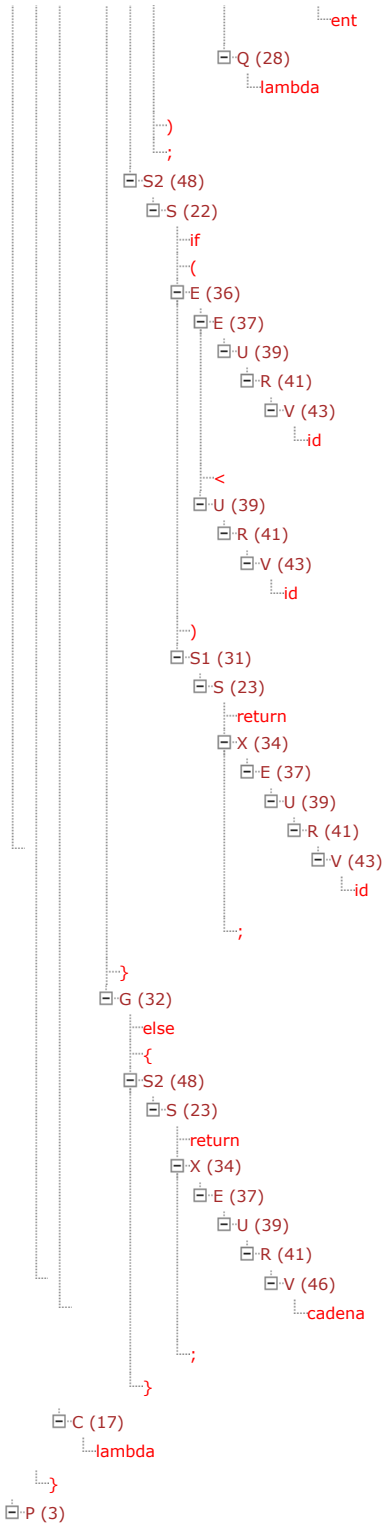
```
function string cadena (boolean log)
{
  if (!log)
  {
    imprime (s, 'hola', 33);
    if (uno < UNO)  return s;
  }
  else
  {
    return 'Fin';
  }
}
s = 'El factorial ';

if (num < 0)    print ('No existe el factorial de un negativo.');
imprime (cadena (booleano), 'recursivo es: ', Factorial (num));
```

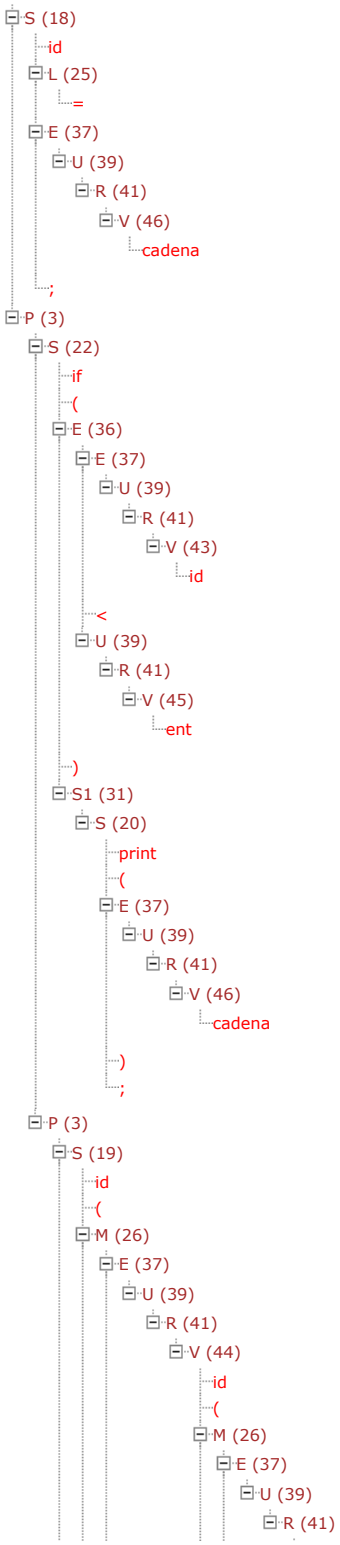
Parse a Derechas:

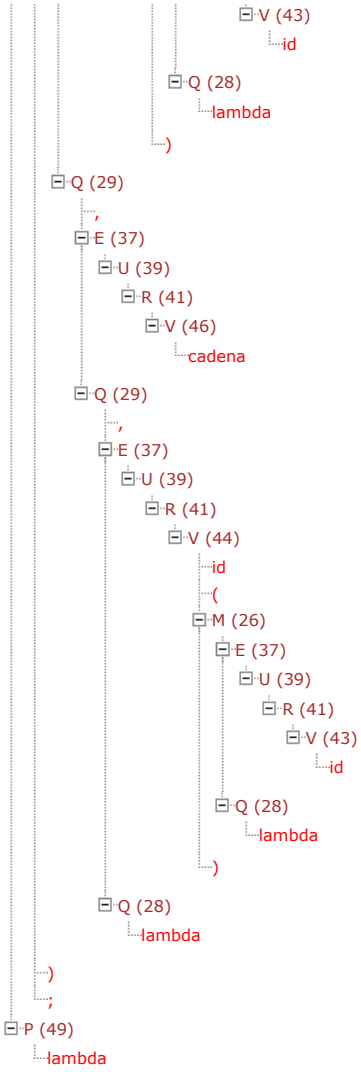
A 6 10 7 13 11 43 40 39 37 43 41 39 37 46 41 39 37 45 41 39 37 28 29 29 26 19 43 41 39 37 43 41 39 36 43 41 39 37 34 23 31 22 48 47 46 41 39 37 34 23 48 32 30 22 17 16 8 25 46 41 39 37 18 43 41 39 37 45 41 39 36 46 41 39 37 20 31 22 43 41 39 37 28 26 44 41 39 37 46 41 39 37 43 41 39 37 28 26 44 41 39 37 28 29 29 26 19 49 3 3 3 2

Árbol sintáctico:




ent
Q (28)
lambda
)
;
S2 (48)
S (22)
if
(
E (36)
E (37)
U (39)
R (41)
V (43)
id
<
U (39)
R (41)
V (43)
id
)
S1 (31)
S (23)
return
X (34)
E (37)
U (39)
R (41)
V (43)
id
;
}
G (32)
else
{
S2 (48)
S (23)
return
X (34)
E (37)
U (39)
R (41)
V (46)
cadena
;
}
C (17)
lambda
}
P (3)


S (18)
id
L (25)
=
E (37)
U (39)
R (41)
V (46)
cadena
;
P (3)
S (22)
if
(
E (36)
E (37)
U (39)
R (41)
V (43)
id
<
U (39)
R (41)
V (45)
ent
)
S1 (31)
S (20)
print
(
E (37)
U (39)
R (41)
V (46)
cadena
)
;
P (3)
S (19)
id
(
M (26)
E (37)
U (39)
R (41)
V (44)
id
(
M (26)
E (37)
U (39)
R (41)


V (43)
id
Q (28)
lambda
)
Q (29)
,
E (37)
U (39)
R (41)
V (46)
cadena
Q (29)
,
E (37)
U (39)
R (41)
V (44)
id
(
M (26)
E (37)
U (39)
R (41)
V (43)
id
Q (28)
lambda
)
Q (28)
lambda
)
;
P (49)
lambda