MEMORIA
PRÁCTICA PDL

Grupo 74
Ignacio Gómez Valverde
Paula Labanderia Campos
Óscar Martínez Garcés

Índice

Introducción

Para nuestro pseudo-compilador de JavaScript hemos empleado el lenguaje Java, esto fue debido a que es el lenguaje que más dominamos todos los integrantes del grupo. Así mismo, cuenta con librerías con una gran variedad de estructuras de las que hicimos uso durante el desarrollo de la práctica.

Como mencionamos antes, se trata de un mini-compilador de JavaScript capaz de leer y entender la estructura general de un programa compuesto por funciones, variables y sentencias, las cuales veremos a continuación:

Variables:
- Tipos enteros, lógicos y cadenas.
- Cadenas con " ".
- Declaración de variables globales, locales y funciones.
- Declaración explícita de variables enteras en asignación o en llamadas a "input"

Operadores:
- Operadores aritméticos básicos. Suma (+), resta (-) y división (/).
- Operadores lógicos. Igualdad (==) y Or (||).
- Operadores de asignación. Igual (=) y OrIgual (|=).

Sentencias:
- Asignación.
- Condicionales simples (if sin else).
- Llamada a funciones con y sin retorno.
- Sentencia de selección múltiple (switch-case)

Operaciones de entrada/salida por terminal:
- input (entrada).
- alert (salida).

Comentarios de línea: // Esto es un comentario

Con respecto a la ejecución del compilador nos decantamos por un analizador descendente por tablas (con una gramática LL(1)) siguiendo uno de los algoritmos vistos en clase.

Respecto a la distribución del código hemos diseñado las siguientes clases y paquetes:

- Paquete Analizadores. Donde se encuentran las clases principales. El analizador léxico, el sintáctico, el semántico, el gestor de errores y el Main.

- Paquete Tablas. Encontramos las clases auxiliares. La tabla de símbolos, las entrada de la tabla de símbolos, la tabla de la gramática del análisis sintáctico, un objeto token y una tabla con palabras reservadas.

Analizador Léxico

Tokens:

< ALERT , - >	alert
< BOOLEAN , - >	boolean
< BREAK, - >	break
< CASE, - >	case
< FUNCTION, - >	function
< IF , - >	if
< INPUT, - >	input
<LET , - >	let
<NUMBER, - >	number
<RETURN, - >	return
<STRING, - >	string
<SWITCH, - >	switch
<ENTERAo, valor>	Constante entera
<CADENA, lexema>	Cadena
<ID, posId>	Identificador
<IGUAL, - >	=
<ORIGUAL, - >	|=
<COMA, - >	,
<PUNTOYCOMA, - >	;
<DOSPUNTOS, - >	:
<PARENTA, - >	(
<PARENTC, - >	)
<CORCHA, - >	{
<CORCHC, - >	}
<MAS, - >	+
<MENOS, - >	-
<DIVISION, - >	/
<OR, - >	||
<IGUALIGUAL, - >	==
<DEFAULT, - >	default
<EOF, - >	eof (end of file)

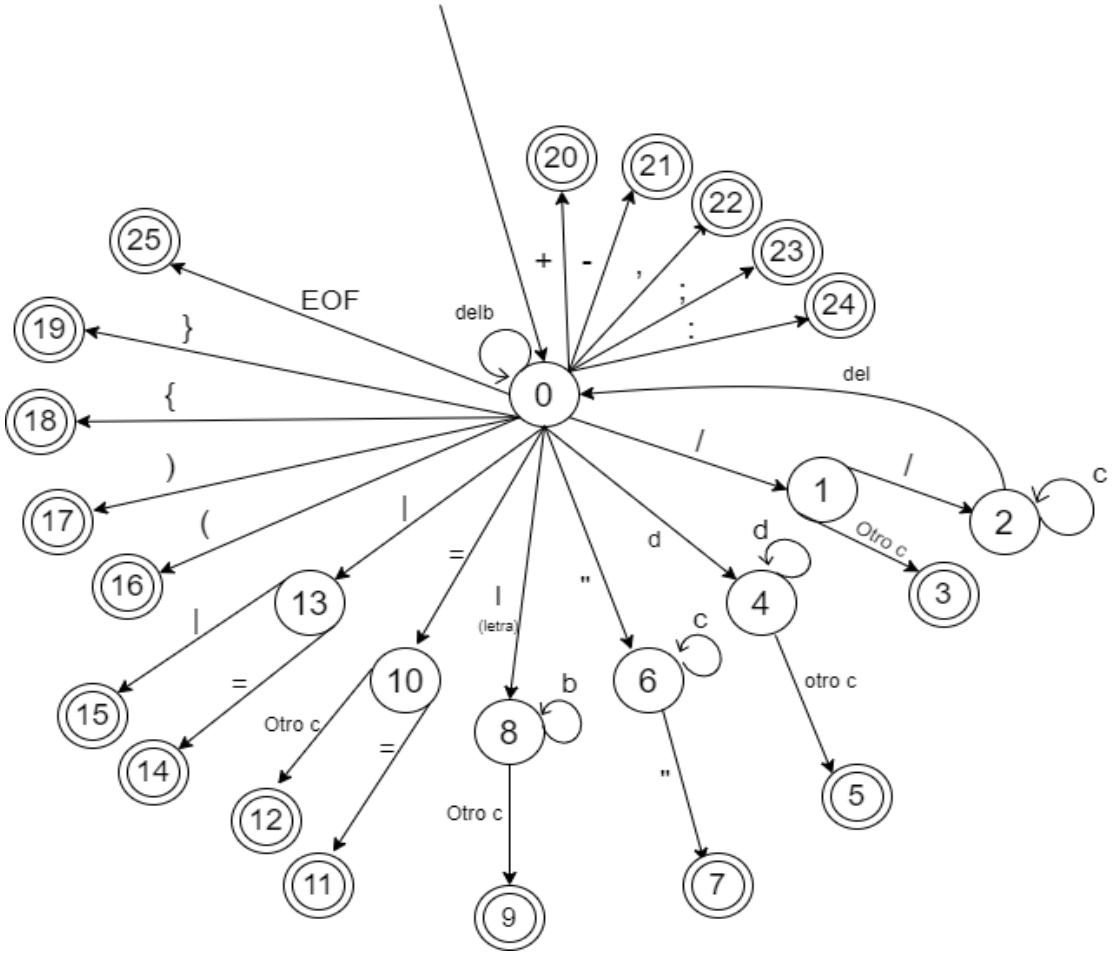
Gramática:

Leyenda:

c = cualquier carácter - {del}
del = {EOL, EOF}
delb = del U {þ}, tab
b = l U d U _
l = letras
d = dígito

S → delb S | / A | d B | “ C | l D | = E | | F | (|) | { | } | + | - | , | ; | : | EOF
A → / A’ | λ
A’ → c A’ | del S
B → d B | λ
C → c C | “
D → b D | λ
E → = | λ
F → | | =

Autómata:



Acciones semánticas analizador léxico

0-0: Leer
0-1: Leer
1-2: Leer
2-0: Leer
2-2: Leer
1-3: Generar token (DIVISION, -)
0-4: valor = valor(d) ; Leer
4-4: valor =valor*10 + valor(d) ; Leer
4-5: if (valor > valorMaximo){
    error("Valor fuera de rango.")
}else{
    Generar token (ENTERO, valor)
}
0-6: Leer; Lexema⊕Ø
6-6: Lexema := lexema ⊕ c; Leer
6-7: if(lexema.tamaño > tamañoMaximoCadenas){
    error("Tamaño de cadena excedido.")
}else{
    Generar token (CADENA,lexema); Leer
}
0-8: Lexema := lexema ⊕ l; Leer
8-8: Lexema := lexema ⊕ b; Leer
8-9: if(esPalabraReservada(lexema))
    Generar Token (lexema.toUpperCase(), -)
}else{
    if(zonaDeclaracion = true){
        if(buscarTS(lexema) = false){
            añadirTS(lexema, posId)
            GenerarToken(ID, posID)
        } else {
            GestorErrores(Variable ya existe)
        }
    } else{
        if(buscarTS(lexema) = false){
            GenerarToken(ID, posID)
        } else {
            GestorErrores(Variable no declarada)
        }
    } Leer;
0-10: Leer
10-11: Genera token(IGUALIGUAL,-); Leer
10-12:Generar token (IGUAL,-);
0-13: Leer;
13-14: Generar token (ORIGUAL,-); leer
13-15: Generar token(OR,-); leer

0-16: Generar token(PARENTA,-); Leer
0-17: Generar token(PARENTC,-); Leer
0-18:Generar token(CORCHA, -); Leer
0-19:Generar token(CORCHC, -); Leer
0-20: Generar token(MAS, -); Leer
0-21: Generar token(MENOS, -); Leer
0-22: Generar token(COMA,-); Leer
0-23:Generar token (PUNTOYCOMA, -); Leer
0-24:Generar token(DOSPUNTOS,-); Leer
0-25:Generar token(EOF,-);

Analizador Sintáctico

Gramática

Terminales = { if let id input alert eof (;) , { : } entero number boolean function string return cadena switch case default break = || |= == + - / }

NoTerminales = { E 1 R 2 U 3 G V 4 Z O S D L Q X B Y M N T F H A K C P }

Axioma = P

Producciones = {
E -> R 1
1 -> || R 1 | lambda
R -> U 2
2 -> == U 2 | lambda
U -> V 3
3 -> G V 3 | lambda
G -> + | -
V -> Z 4
4 -> / Z 4 | lambda
Z -> id O | (E) | entero | cadena
O -> lambda | (L)
S -> id D ; | alert (E) ; | input (id) ; | return X ;
D -> = E | |= E | (L)
L -> E Q | lambda
Q -> , E Q | lambda
X -> E | lambda
B -> if (E) S | let T id ; | S | switch (E) { Y }
Y -> case entero : C M
M -> break ; N | N
N -> Y | default : C | lambda
T -> number | boolean | string
F -> function H id (A) { C }
H -> T | lambda
A .> T id K | lambda
K -> , T id K | lambda
C -> B C | lambda
P -> B P | F P | eof | lambda
}

Demostración de gramática válida

Sabemos que es adecuada ya que es una gramática LL(1), no tiene recursividad por la izquierda en ninguno de sus estados, cumple la condición LL, está factorizada y no es ambigua en ningún caso.

Tabla LL

NT / T	if	let	id	input	alert	eof	(	;	)	,	{	:	}	entero	number	boolean	function	string	return	cadena	switch	case	default	break	=	||	|=	==	+	-	/	$
E			E -> R 1				E -> R 1							E -> R 1						E -> R 1												
1								1 -> lambda	1 -> lambda	1 -> lambda																1 -> || R 1						
R			R -> U 2				R -> U 2							R -> U 2						R -> U 2												
2								2 -> lambda	2 -> lambda	2 -> lambda																2 -> lambda		2 -> == U 2				
U			U -> V 3				U -> V 3							U -> V 3						U -> V 3												
3								3 -> lambda	3 -> lambda	3 -> lambda																3 -> lambda		3 -> lambda	3 -> G V 3	3 -> G V 3		
G																													G -> +	G -> -		
V			V -> Z 4				V -> Z 4							V -> Z 4						V -> Z 4												
4								4 -> lambda	4 -> lambda	4 -> lambda																4 -> lambda		4 -> lambda	4 -> lambda	4 -> lambda	4 -> / Z 4	
Z			Z -> id O				Z -> (E)							Z -> entero						Z -> cadena												
O							O -> (L)	O -> lambda	O -> lambda	O -> lambda																O -> lambda		O -> lambda	O -> lambda	O -> lambda	O -> lambda	
S			S -> id D ;	S -> input (id) ;	S -> alert (E) ;														S -> return X ;													
D							D -> (L)																		D -> = E		D -> |= E					
L			L -> E Q				L -> E Q		L -> lambda					L -> E Q						L -> E Q												
Q									Q-> lambda	Q -> , E Q																						
X			X -> E				X -> E	X -> lambda						X -> E						X -> E												
B	B -> if (E) S	B -> let T id ;	B -> S	B -> S	B -> S														B -> S		B -> switch (E) { Y }										
Y																						Y -> case entero : C M										
M													M -> N									M -> N	M -> N	M -> break ; N								
N													N -> lambda									N -> Y	N -> default : C									
F																	F -> function H id (A) { C }															
T															T -> number	T -> boolean		T -> string														
H			H -> lambda												H -> T	H -> T		H -> T														
A									A -> lambda						A -> T id K	A -> T id K		A -> T id K														
K									K -> lambda	K -> , T id K																						
C	C -> B C	C -> B C	C -> B C	C -> B C	C -> B C								C -> lambda						C -> B C		C -> B C	C -> lambda	C -> lambda	C -> lambda							
P	P -> B P	P -> B P	P -> B P	P -> B P	P -> B P	P -> eof											P -> F P		P -> B P			P -> B P										P -> lambda

Analizador Semántico

Gramática con Acciones semánticas (EdT)

Producciones = {
E → R 1 {1}
1 → || R 1 {2}
1 → lambda {3}
R → U 2 {4}
2 → == U 2 {5}
2 → lambda {6}
U → V 3 {7}
3 → G V 3 {8}
3 → lambda {9}
G → +
G → -
V → Z 4 {12}
4 → / Z 4 {13}
4 → lambda {14}
Z → id O {15}
Z → (E) {16}
Z → entero {17}
Z → cadena {18}
O → lambda
O → (L) {20}
S → id D ; {21}
S → alert (E) ; {22}
S → input (id {23.1}) ; {23.2}
S → return X ; {24}
D → = E {25}
D → |= E {26}
D → (L) {27}
L → E {28} Q {28.1}
L → lambda {29}
Q → , E {30} Q {30.1}
Q → lambda {31}
X → E {32}
X → lambda {33}
B → if (E) S {34}
B → let T id ; {35}
B → S {36}
B → switch (E) {37.1} { Y } {37.2}
Y → case entero : C M {38}
M → break ; N {39}
M → N {40}
N → Y {41}
N → default : C {42}

N → lambda
T -> number {44}
T→ boolean {45}
T → string {46}
F → function H id {47.1} (A) {47.3} { C } {47.4}
H → T {48}
H → lambda {49}
A → T id {50.1} K {50.2}
A → lambda {51}
K → , T id {52.1} K {52.2}
K → lambda {53}
C → B C {54}
C → lambda {55}
P → B P {56}
P → F P {57}
P → eof {59}
P → lambda
}

Acciones semánticas

{1}
E.tipo := if(1.tipo=void || R.tipo == 1.tipo)
then R.tipo
Else
tipo_Error

{2}
1.tipo := if(1.tipo=void || (1.tipo= boolean && R.tipo = boolean)
then R.tipo
Else
tipo_Error

{3}
1.tipo:= void

{4}
R.tipo := {
if(2.tipo = void)
then U.tipo
elseif(2.tipo = U.tipo)
then boolean
Else
tipo_Error
}

{5}

```
2.tipo := {
        if(2.tipo=void)
                then U.tipo
        elseif(2.tipo = U.tipo)
                then boolean
        Else
                tipo_Error
}
```

{6}

```
2.tipo := void
```

{7}

```
U.tipo := {
        if(3.tipo=void || 3.tipo = V.tipo)
                then V.tipo
        Else
                tipo_Error
}
```

{8}

```
3.tipo := {
        if(3.tipo=void || (3.tipo = entero && V.tipo = entero))
                Then V.tipo
        Else
                tipo_Error
        }
```

{9}

```
3.tipo = void
```

{12}

```
V.tipo := {
        if(4.tipo=void || (4.tipo = entero && Z.tipo = entero))
                Then Z.tipo
        Else
                tipo_Error
        }
```

{13}

```
4.tipo := {
        if(4.tipo=void || 4.tipo = entero && Z.tipo = entero))
                Then Z.tipo
        Else
                tipo_Error
        }
```

{14}

4.tipo := void

{15}

```
Z.tipo := {
        if(buscarTSLocal(idPos) || buscarTSPrincipal(idPos))
                then id.Tipo
        else
                tipo_Error
}
```

{16}

Z.tipo := E.tipo

{17}

Z.tipo := entero

{18}

Z.tipo := string

{20}

O.tipo := L.tipo

{21}

```
S.tipo := {
        id.tipo:= buscarTSTipo(id.pos);
        if(id.tipo = D.tipo)
                then tipo_ok
        else
                tipo_error}
```

{22}

```
S.tipo := {
        if(E.tipo = boolean || E.tipo = entero)
                then boolean
        Else
                tipo_Error
}
```

{23.1}

```
{
        id.tipo := buscarTSTipo(id.pos);
        if(id.tipo € {cadena, entero} )
                then tipo_ok
        else
                tipo_error;
}
```

{23.2}
{ }

{24}
{
tipoRetorno = buscarTSTipoRetorno(idFuncionActual);
if(X.tipo = tipoRetorno)
then tipo_OK
else
then tipo_Error
}
{25}
D.tipo := E.tipo

{26}
D.tipo := E.tipo

{27}
/*
* Para la comparación del tipo correcto de los parámetros guardamos todos los tipos
* en un ArrayList (ArrayListN) y esta acción semántica lo comparara con el ArrayList
de tipos que * hay guardado en la tabla de tipos (ArrayListTS)
*/
D.tipo := {
ArrayListTS = buscarTSTipoParametros(idPos)
if(ArrayListN == ArrayListTS)
then “function”
else
tipo_Error
}
{28}
ArraylistN.add(E.tipo)

{28.1}
zonaDeclaracion=false

{29}
ArraylistN.add(void)

{30}
ArraylistN.add(E.tipo)

{30.1}
zonaDeclaracion=false
{31}
{ }

{32}

X.tipo := E.tipo
zonaDeclaracion= false

{33}

X.tipo := void

{34}

B.tipo :=
if (E.tipo = boolean)
then E.tipo
else
tipo_Error
zonaDeclaracion = false

{35}

id.tipo := T.tipo
insertarTSTipoYdesp(id.pos,id.tipo, desp);
id.desp = desp + T.ancho

{36}

{ }

{37.1}

B.tipo :=
if (E.tipo = entero)
then tipo_ok
Else
tipo_Error}
ZonaDeclaracion(false)

{37.2}

{ }

{38}

{ }

{39}

{ }

{40}

{ }

{41}

{ }

{42}

{ }

{43}

N.tipo = void

{44}

T.tipo := ent, T.ancho:=2

{45}

T.tipo := boolean, T.ancho:=1

{46}

T.tipo := ent, T.ancho := 64

{47.1}

posIdFuncionActual = pila.peek()
insertarTipoPosIdFuncionActual(“function”);
insertTipoRetorno(H.tipo)
zonaDeclaracion = true

{47.3}

zonaDeclaracion = false

{47.4}

{ }

{48}

H.tipo := T.tipo

{49}

H.tipo := void

{50.1}

id.tipo := T.tipo
insertTSTipoYdesp(id.tipo, desp), desp = desp + T.ancho

{51}

A.tipo=void

{52.1}

{
id.tipo := T.tipo;
ArrayListTS.add(id.tipo) ;
insertTsNueva(id.pos,id.tipo, id.desp);
id.desp := T.ancho + desp;
}

{52.2}

{ }

{53}

insertNumParam(posIdFuncionActual, ArrayListN.size())

{54}

{ }

{55}

{ }

{56}

{ }

{57}

{ }

{59}

{ }

Diseño de tabla de símbolos

Para la tabla de símbolos hemos decidido dividirla en dos clases separadas, una será la tabla propiamente dicha y la usaremos para definir cada una de las entradas que se introducirán en la misma.

Todas las tablas de símbolos que usamos a lo largo de la ejecución se irán introduciendo en una pila de forma que la cima de ésta siempre sea la tabla en uso. Cuando se cierre una tabla será extraída de dicha pila y se pasará a almacenar la información en otra estructura para su posterior impresión.

Tabla de Símbolos

Cada tabla estará formada por cuatro campos principales:

- Un HashMap que será donde almacenamos todas las entradas de la tabla.
- El nombre de la tabla.
- El número de la tabla. Irá desde la Tabla Programa Principal con número 1 en adelante.
- Desplazamiento. Se irá incrementando en función del tipo de las variables que se vayan introduciendo en la tabla.

Estos cuatro parámetros contarán con sus respectivos getters y setters para poder ver sus valores y modificarlos. Además de estos métodos la clase contará con dos métodos para buscar variables uno por su posición y otro por su lexema, un método para añadir variables por su lexema y un método imprimir para pasar todas las tablas al fichero txt.

ObjetoTS

Cada una de las entradas del HashMap de la clase anterior estará formada por una clave que utilizaremos como la posición que ocupa en la tabla y un ObjetoTS. Diferenciaremos dos tipos de ObjetoTS, uno para variables y otro para declaración de funciones.

Variables:

- Lexema. Será el nombre de la variable.
- Tipo. Tipo de la variable. Entero, cadena o boolean.
- Desplazamiento. Posición de la variable con respecto al resto.

Declaración de funciones:

- Lexema. Nombre de la función.
- Tipo. Será “function”.
- Número de parámetros. Valor numérico entre 0 y “n” siendo “n” el número de parámetros.
- Tipo de parámetros. ArrayList que almacenará los tipos de los parámetros.
- Tipo de retorno. Guardará el tipo de retorno de la función o “void” si no tiene.
- Etiqueta. Será la etiqueta asociada a la función, en nuestro caso: Etiq_<lexema>

Anexo

Pruebas correctas

Prueba 1:

Código:

```
let number z;
let boolean boolean_1;
let number x;
let string ss;
let number xx;
let boolean boolean_2;


function number f1(number f1, boolean b1)
{
            alert(ss);
            x = xx/f1;
            boolean_1 = boolean_1|| boolean_2;
            return (x);
}


function boolean f2(number f2, boolean b1)
{
            input (y);
            alert ((4-5+77+(088-f2)));
            return (boolean_1||boolean_2||b1);
}

x =
  x + 6
    - z
    + 1
    / (2
    + y
    / 6)
      ;


alert (f1 (x, f2 (3, boolean_2)));
```

Tokens:

<LET, >
<NUMBER, >
<ID, 0>
<PUNTOYCOMA, >
<LET, >
<BOOLEAN, >
<ID, 1>
<PUNTOYCOMA, >
<LET, >
<NUMBER, >
<ID, 2>
<PUNTOYCOMA, >
<LET, >
<STRING, >
<ID, 3>
<PUNTOYCOMA, >
<LET, >
<NUMBER, >
<ID, 4>
<PUNTOYCOMA, >
<LET, >
<BOOLEAN, >
<ID, 5>
<PUNTOYCOMA, >
<FUNCTION, >
<NUMBER, >
<ID, 6>
<PARENTA, >
<NUMBER, >
<ID, 7>
<COMA, >
<BOOLEAN, >
<ID, 8>
<PARENTC, >
<CORCHA, >
<ALERT, >
<PARENTA, >
<ID, 3>
<PARENTC, >
<PUNTOYCOMA, >
<ID, 2>
<IGUAL, >
<ID, 4>
<DIVISION, >
<ID, 7>
<PUNTOYCOMA, >
<ID, 1>
<IGUAL, >
<ID, 1>
<OR, >
<ID, 5>
<PUNTOYCOMA, >
<RETURN, >
<PARENTA, >
<ID, 2>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<BOOLEAN, >
<ID, 9>
<PARENTA, >
<NUMBER, >
<ID, 10>
<COMA, >
<BOOLEAN, >
<ID, 11>
<PARENTC, >
<CORCHA, >
<INPUT, >
<PARENTA, >
<ID, 12>
<PARENTC, >
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<PARENTA, >
<ENTERO, 4>
<MENOS, >
<ENTERO, 5>
<MAS, >
<ENTERO, 77>
<MAS, >
<PARENTA, >
<ENTERO, 88>
<MENOS, >
<ID, 10>
<PARENTC, >
<PARENTC, >
<PARENTC, >
<PUNTOYCOMA, >
<RETURN, >
<PARENTA, >
<ID, 1>
<OR, >
<ID, 5>
<OR, >
<ID, 11>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<ID, 2>
<IGUAL, >
<ID, 2>
<MAS, >
<ENTERO, 6>
<MENOS, >
<ID, 0>
<MAS, >
<ENTERO, 1>
<DIVISION, >
<PARENTA, >
<ENTERO, 2>
<MAS, >
<ID, 12>
<DIVISION, >
<ENTERO, 6>
<PARENTC, >
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<ID, 6>
<PARENTA, >
<ID, 2>
<COMA, >
<ID, 9>
<PARENTA, >
<ENTERO, 3>
<COMA, >
<ID, 5>
<PARENTC, >
<PARENTC, >
<PARENTC, >
<PUNTOYCOMA, >
<EOF, >

Tabla de símbolos:

```
Tabla Programa Principal # 1:
* LEXEMA : 'z'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'boolean_1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'
* LEXEMA : 'x'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '2'
* LEXEMA : 'ss'
  ATRIBUTOS :
    + tipo : 'string'
    + despl : '3'
* LEXEMA : 'xx'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '67'
* LEXEMA : 'boolean_2'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '68'
* LEXEMA : 'f1'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'boolean'
    + TipoRetorno : 'entero'
    + EtiqFuncion : 'Etiq_f1'
* LEXEMA : 'f2'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'boolean'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_f2'
* LEXEMA : 'y'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '69'
```

```
Tabla Funcion f1 # 2:
* LEXEMA : 'f1'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'b1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'

Tabla Funcion f2 # 3:
* LEXEMA : 'f2'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'b1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'
```

Parse

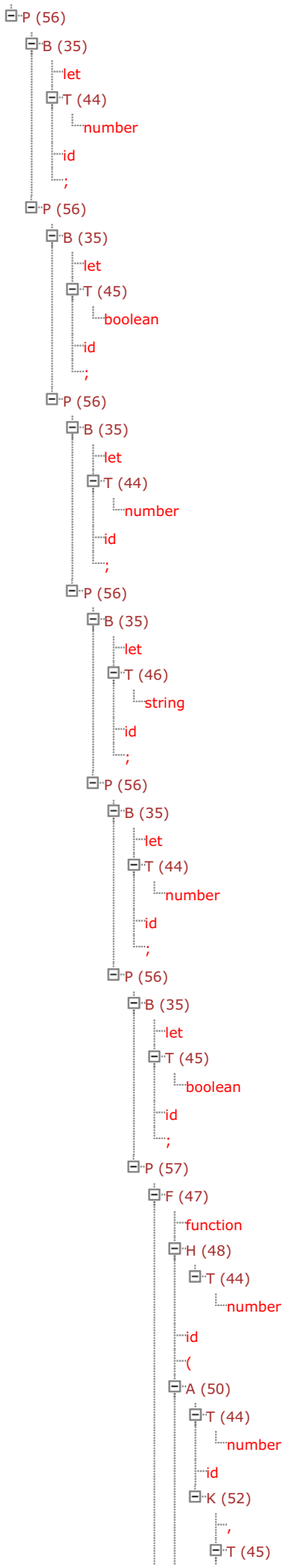
```
Descendente 56 35 44 56 35 45 56 35 44 56 35 46 56 35 44 56 35 45 57 47
48 44 50 44 52 45 53 54 36 22 1 4 7 12 15 19 14 9 6 3 54 36 21 25 1 4 7
12 15 19 13 15 19 14 9 6 3 54 36 21 25 1 4 7 12 15 19 14 9 6 2 4 7 12
15 19 14 9 6 3 54 36 24 32 1 4 7 12 16 1 4 7 12 15 19 14 9 6 3 14 9 6 3
55 57 47 48 45 50 44 52 45 53 54 36 23 54 36 22 1 4 7 12 16 1 4 7 12 17
14 8 11 12 17 14 8 10 12 17 14 8 10 12 16 1 4 7 12 17 14 8 11 12 15 19
14 9 6 3 14 9 6 3 14 9 6 3 54 36 24 32 1 4 7 12 16 1 4 7 12 15 19 14 9
6 2 4 7 12 15 19 14 9 6 2 4 7 12 15 19 14 9 6 3 14 9 6 3 55 56 36 21 25
1 4 7 12 15 19 14 8 10 12 17 14 8 11 12 15 19 14 8 10 12 17 13 16 1 4 7
12 17 14 8 10 12 15 19 13 17 14 9 6 3 14 9 6 3 56 36 22 1 4 7 12 15 20
28 1 4 7 12 15 19 14 9 6 3 30 1 4 7 12 15 20 28 1 4 7 12 17 14 9 6 3 30
1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 31 14 9 6 3 59
```

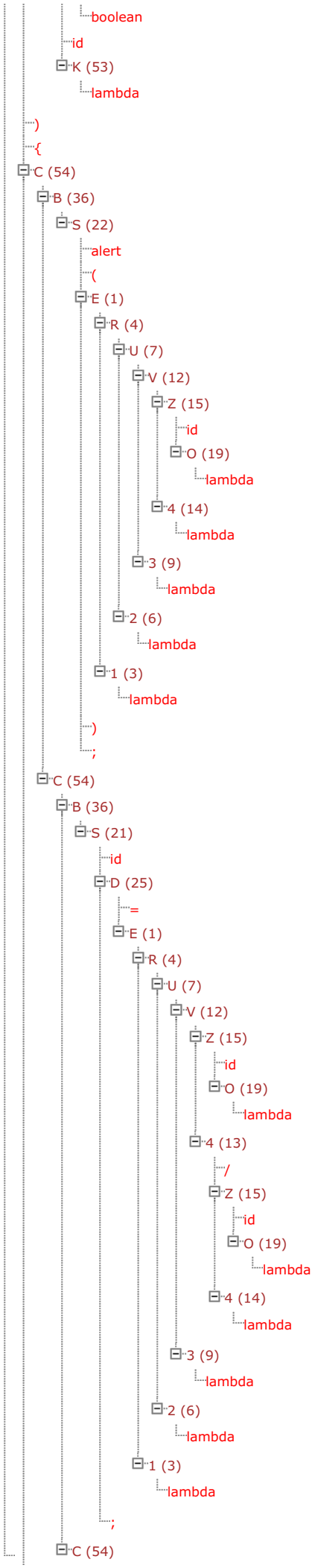
Árbol

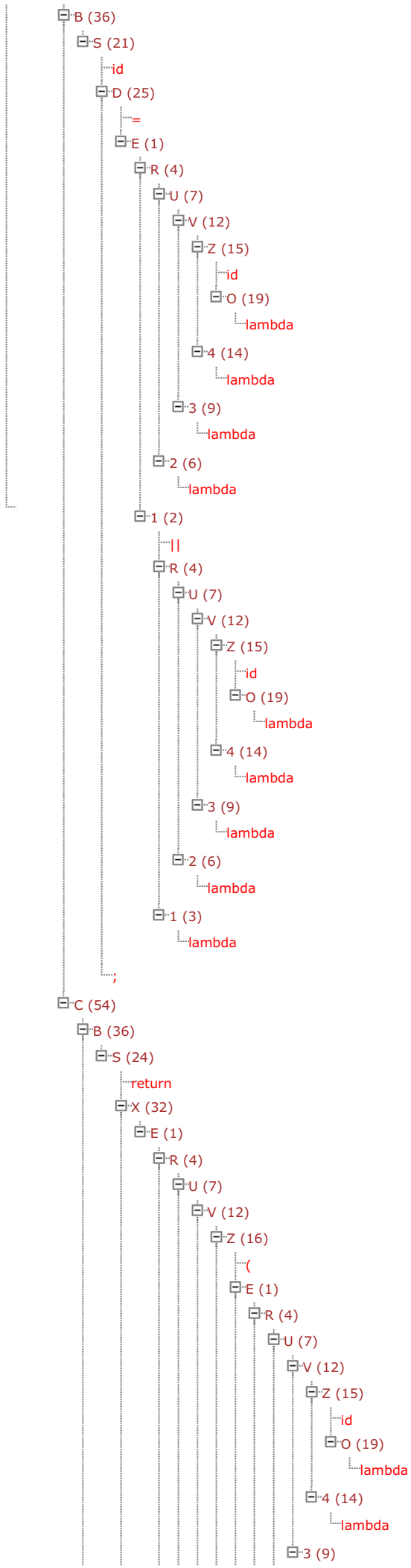
Árbol resultado de:

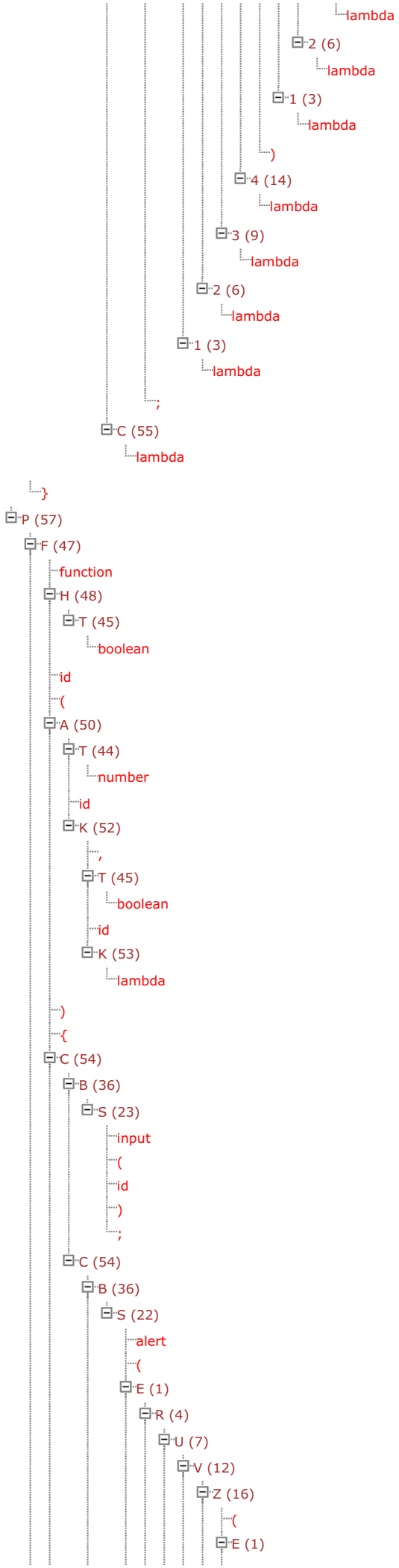
Gramática: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\VisorArbSt\Gramp1.txt

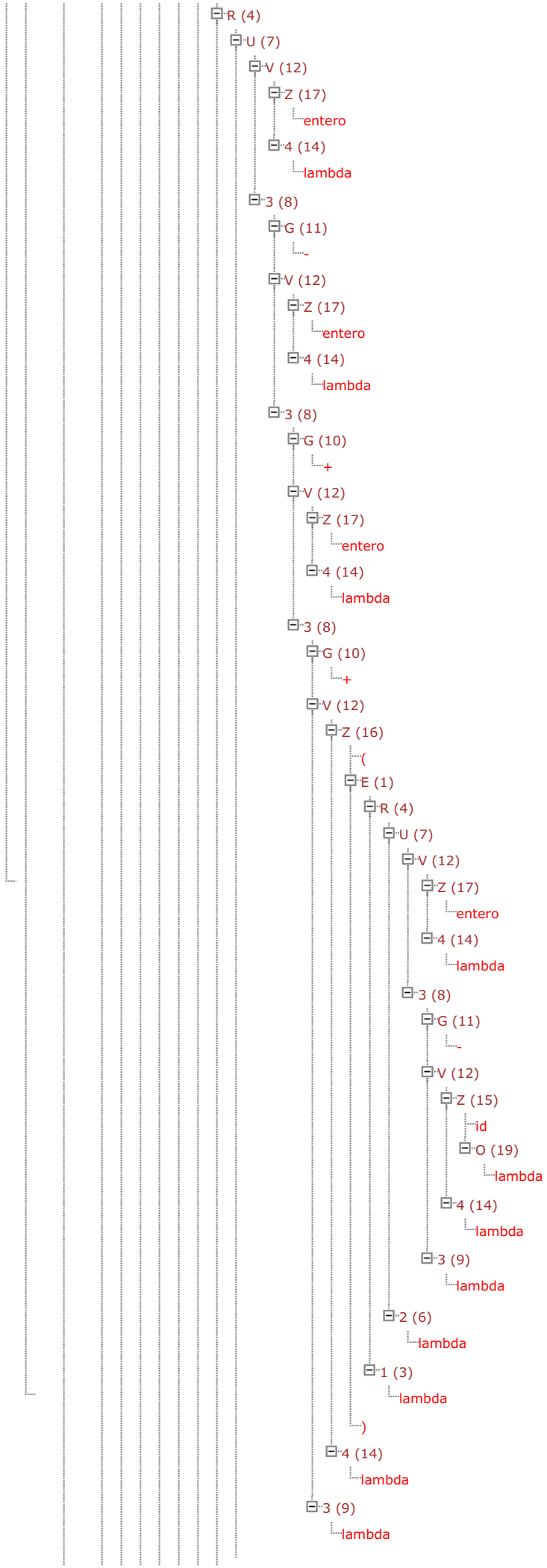
Parse: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\PracticaPDL\parse.txt

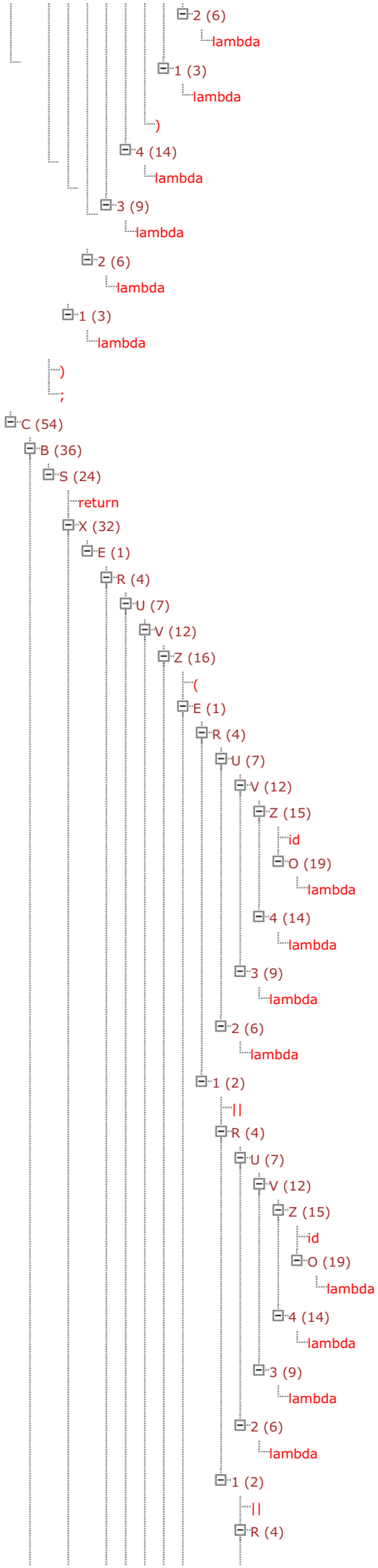


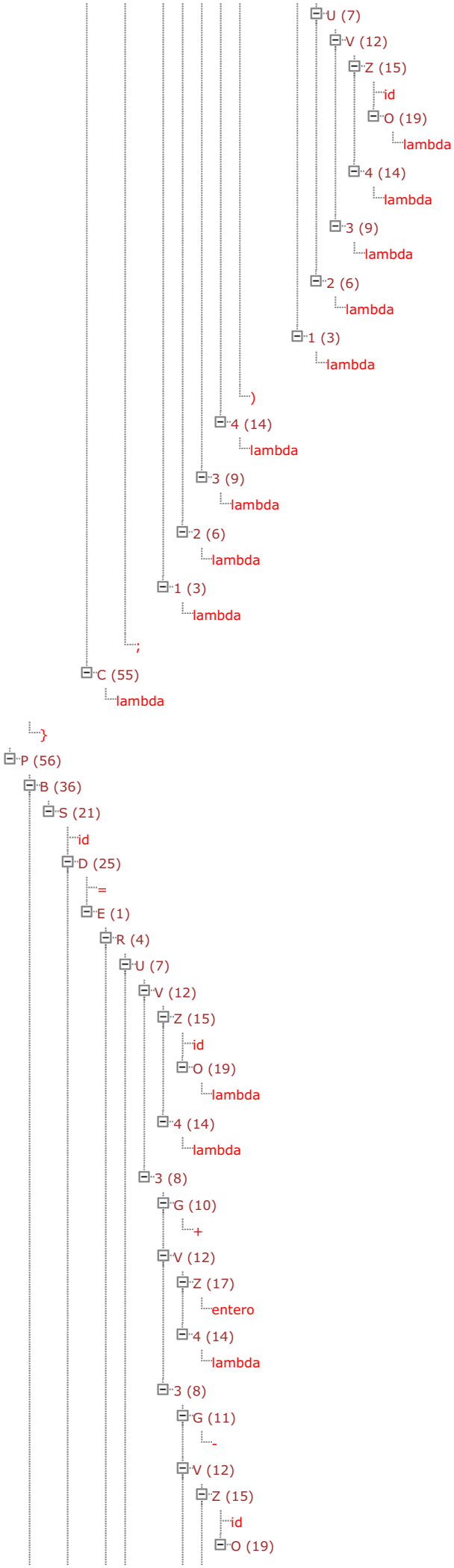

boolean
id
K (53)
lambda
)
{
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (13)
/
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)

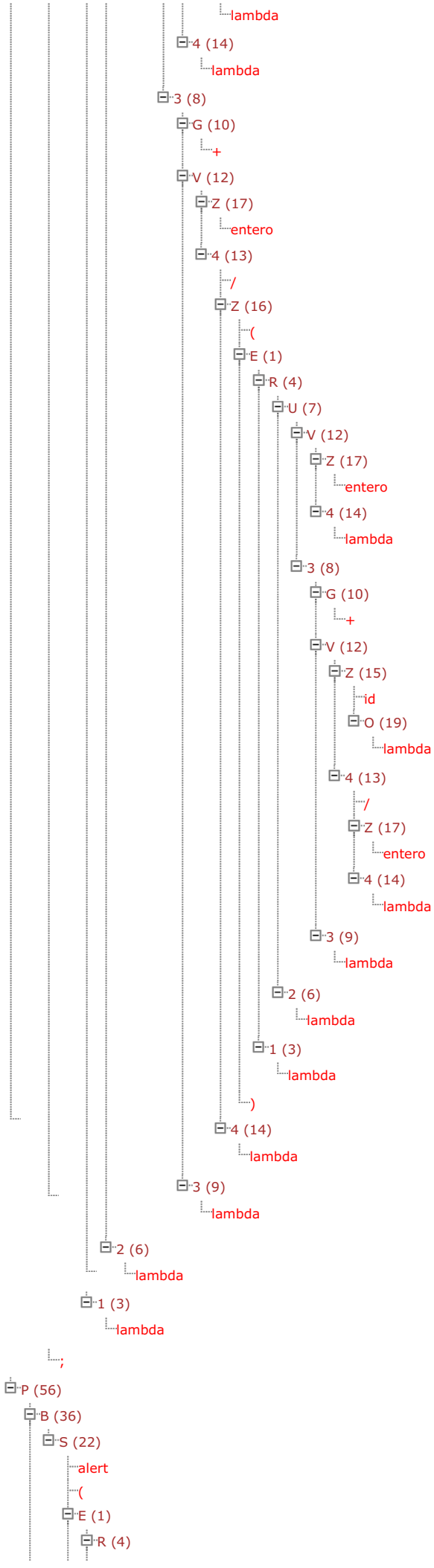

B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)

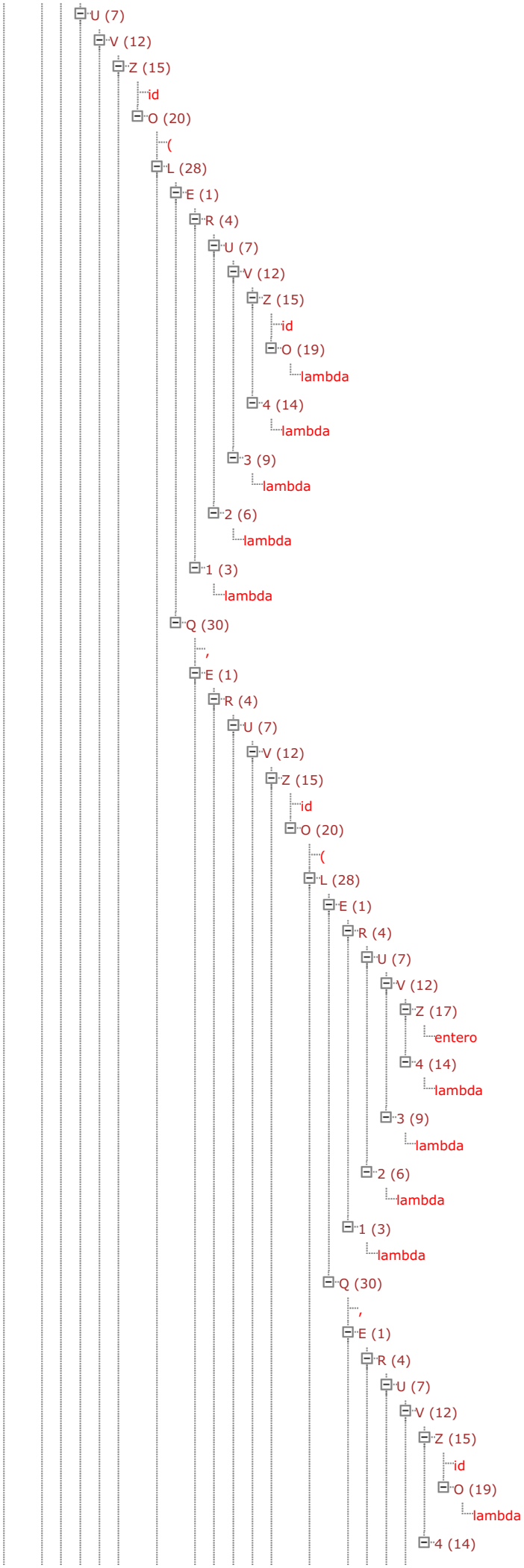

lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (45)
boolean
id
(
A (50)
T (44)
number
id
K (52)
,
T (45)
boolean
id
K (53)
lambda
)
{
C (54)
B (36)
S (23)
input
(
id
)
;
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)

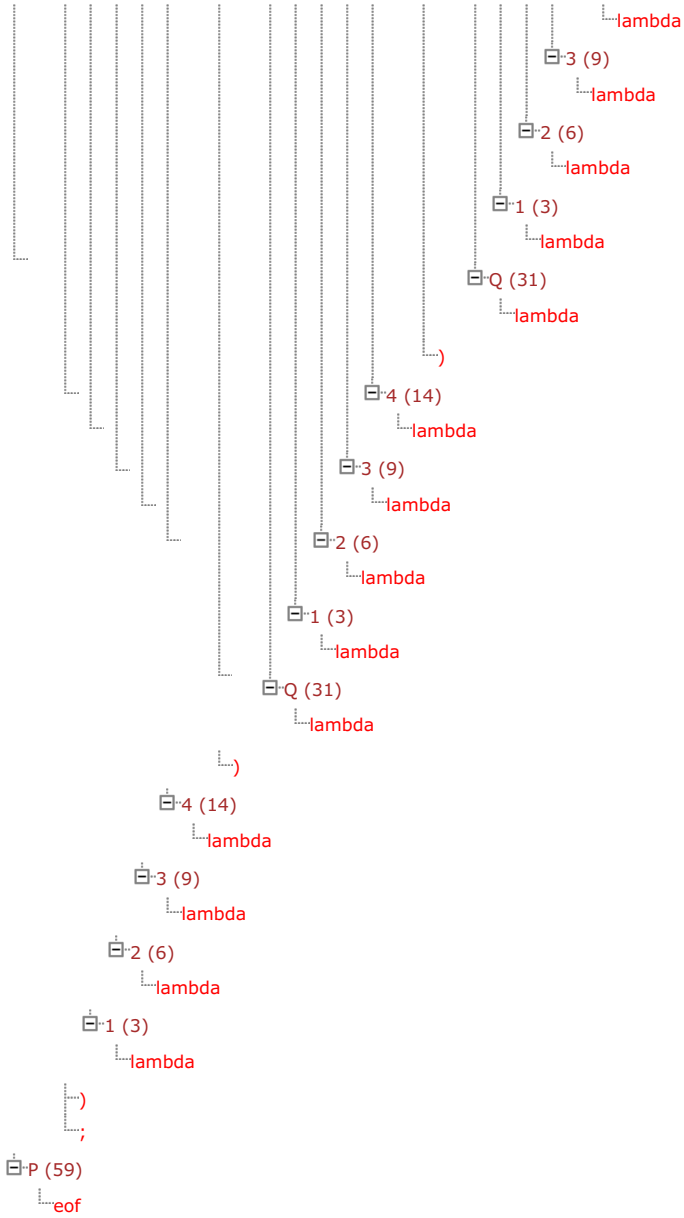

R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda


2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)


U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (56)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)


lambda
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (13)
/
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (15)
id
O (19)
lambda
4 (13)
/
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (36)
S (22)
alert
(
E (1)
R (4)


U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)


lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
P (59)
eof

Prueba 2

Código

```
let boolean booleano;
function boolean bisiesto (number a)
{
      return (a - 4 == 0 || a + 100 == 0 || a - 400 == 0);
}
function number dias (number m, number a)
{
      let number dd;
      alert ("di cuantos dias tiene el mes ");
      alert (m);
      input(dd);
      if (bisiesto(a)) dd = dd / 1;
      return dd;
}
function boolean esFechaCorrecta (number d, number m, number a)
{
      return m==1 || m==12 || d==1 || d == dias (m, a);
}
function demo ()
{

      if (esFechaCorrecta(25, 20, 2020)) alert (9999);
      return;
}
let number a1b2c3d4e5f6g7h8i9j0;
demo();
```

Tokens

```
<LET, >
<BOOLEAN, >
<ID, 0>
<PUNTOYCOMA, >
<FUNCTION, >
<BOOLEAN, >
<ID, 1>
<PARENTA, >
<NUMBER, >
<ID, 2>
<PARENTC, >
<CORCHA, >
<RETURN, >
<PARENTA, >
<ID, 2>
<MENOS, >
<ENTERO, 4>
<IGUALIGUAL, >
<ENTERO, 0>
<OR, >
<ID, 2>
<MAS, >
<ENTERO, 100>
<IGUALIGUAL, >
<ENTERO, 0>
<OR, >
<ID, 2>
<MENOS, >
<ENTERO, 400>
<IGUALIGUAL, >
<ENTERO, 0>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<NUMBER, >
<ID, 3>
<PARENTA, >
<NUMBER, >
<ID, 4>
<COMA, >
<NUMBER, >
<ID, 5>
<PARENTC, >
<CORCHA, >
<LET, >
<NUMBER, >
<ID, 6>
```

```
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<CADENA, "di cuantos
dias tiene el mes ">
<PARENTC, >
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<ID, 4>
<PARENTC, >
<PUNTOYCOMA, >
<INPUT, >
<PARENTA, >
<ID, 6>
<PARENTC, >
<PUNTOYCOMA, >
<IF, >
<PARENTA, >
<ID, 1>
<PARENTA, >
<ID, 5>
<PARENTC, >
<PARENTC, >
<ID, 6>
<IGUAL, >
<ID, 6>
<DIVISION, >
<ENTERO, 1>
<PUNTOYCOMA, >
<RETURN, >
<ID, 6>
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<BOOLEAN, >
<ID, 7>
<PARENTA, >
<NUMBER, >
<ID, 8>
<COMA, >
<NUMBER, >
<ID, 9>
<COMA, >
<NUMBER, >
<ID, 10>
<PARENTC, >
<CORCHA, >
<RETURN, >
<ID, 9>
<IGUALIGUAL, >
<ENTERO, 1>
<OR, >
<ID, 9>
<IGUALIGUAL, >
<ENTERO, 12>
<OR, >
<ID, 8>
<IGUALIGUAL, >
<ENTERO, 1>
<OR, >
<ID, 8>
<IGUALIGUAL, >
<ID, 3>
<PARENTA, >
<ID, 9>
<COMA, >
<ID, 10>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<ID, 11>
<PARENTA, >
<PARENTC, >
<CORCHA, >
<IF, >
<PARENTA, >
<ID, 7>
<PARENTA, >
<ENTERO, 25>
<COMA, >
<ENTERO, 20>
<COMA, >
<ENTERO, 2020>
<PARENTC, >
<PARENTC, >
<ALERT, >
<PARENTA, >
<ENTERO, 9999>
<PARENTC, >
<PUNTOYCOMA, >
<RETURN, >
<PUNTOYCOMA, >
<CORCHC, >
<LET, >
<NUMBER, >
<ID, 12>
<PUNTOYCOMA, >
<ID, 11>
<PARENTA, >
<PARENTC, >
<PUNTOYCOMA, >
<EOF, >
```

Tabla de símbolos

```
Tabla Programa Principal # 1:
* LEXEMA : 'booleano'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '0'
* LEXEMA : 'bisiesto'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 1
    + TipoParam1 : 'entero'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_bisiesto'
```

```
* LEXEMA : 'dias'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'entero'
    + TipoRetorno : 'entero'
    + EtiqFuncion : 'Etiq_dias'
* LEXEMA : 'esFechaCorrecta'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 3
    + TipoParam1 : 'entero'
    + TipoParam2 : 'entero'
    + TipoParam3 : 'entero'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_esFechaCorrecta'
* LEXEMA : 'demo'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 0
    + TipoRetorno : 'void'
    + EtiqFuncion : 'Etiq_demo'
* LEXEMA : 'a1b2c3d4e5f6g7h8i9j0'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'

Tabla Funcion bisiesto # 2:
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'

Tabla Funcion dias # 3:
* LEXEMA : 'm'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'
* LEXEMA : 'dd'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '2'

Tabla Funcion esFechaCorrecta # 4:
* LEXEMA : 'd'
  ATRIBUTOS :
```

```
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'm'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '2'

Tabla Funcion demo # 5:
```

Parse

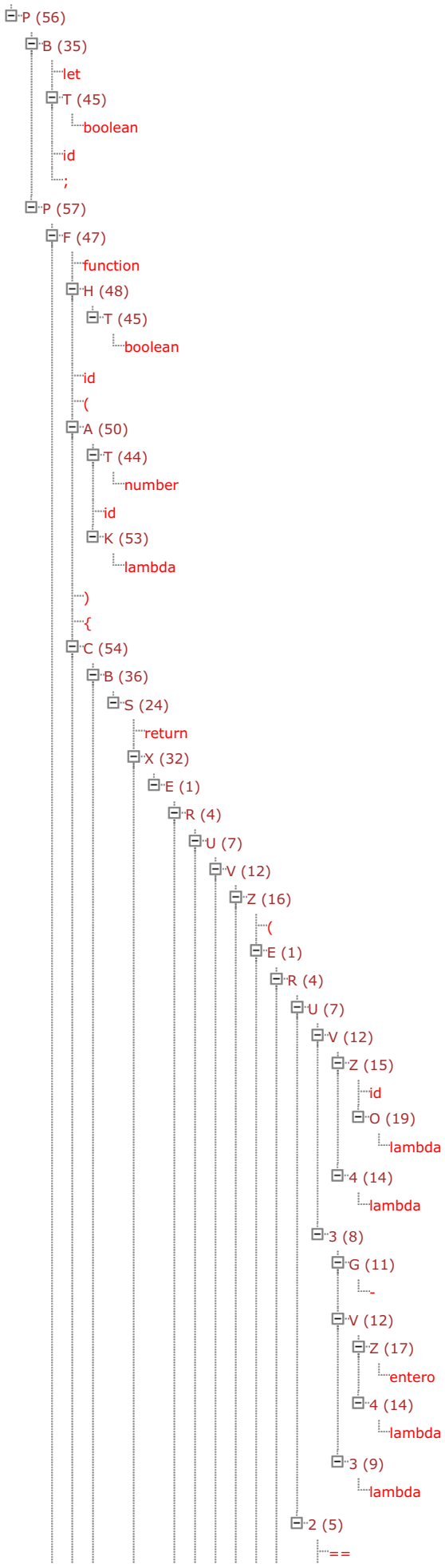
```
Descendente 56 35 45 57 47 48 45 50 44 53 54 36 24 32 1 4 7 12 16 1 4 7
12 15 19 14 8 11 12 17 14 9 5 7 12 17 14 9 6 2 4 7 12 15 19 14 8 10 12
17 14 9 5 7 12 17 14 9 6 2 4 7 12 15 19 14 8 11 12 17 14 9 5 7 12 17 14
9 6 3 14 9 6 3 55 57 47 48 44 50 44 52 44 53 54 35 44 54 36 22 1 4 7 12
18 14 9 6 3 54 36 22 1 4 7 12 15 19 14 9 6 3 54 36 23 54 34 1 4 7 12 15
20 28 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 21 25 1 4 7 12 15 19 13 17 14
9 6 3 54 36 24 32 1 4 7 12 15 19 14 9 6 3 55 57 47 48 45 50 44 52 44 52
44 53 54 36 24 32 1 4 7 12 15 19 14 9 5 7 12 17 14 9 6 2 4 7 12 15 19
14 9 5 7 12 17 14 9 6 2 4 7 12 15 19 14 9 5 7 12 17 14 9 6 2 4 7 12 15
19 14 9 5 7 12 15 20 28 1 4 7 12 15 19 14 9 6 3 30 1 4 7 12 15 19 14 9
6 3 31 14 9 6 3 55 57 47 49 51 54 34 1 4 7 12 15 20 28 1 4 7 12 17 14 9
6 3 30 1 4 7 12 17 14 9 6 3 30 1 4 7 12 17 14 9 6 3 31 14 9 6 3 22 1 4
7 12 17 14 9 6 3 54 36 24 33 55 56 35 44 56 36 21 27 29 59
```

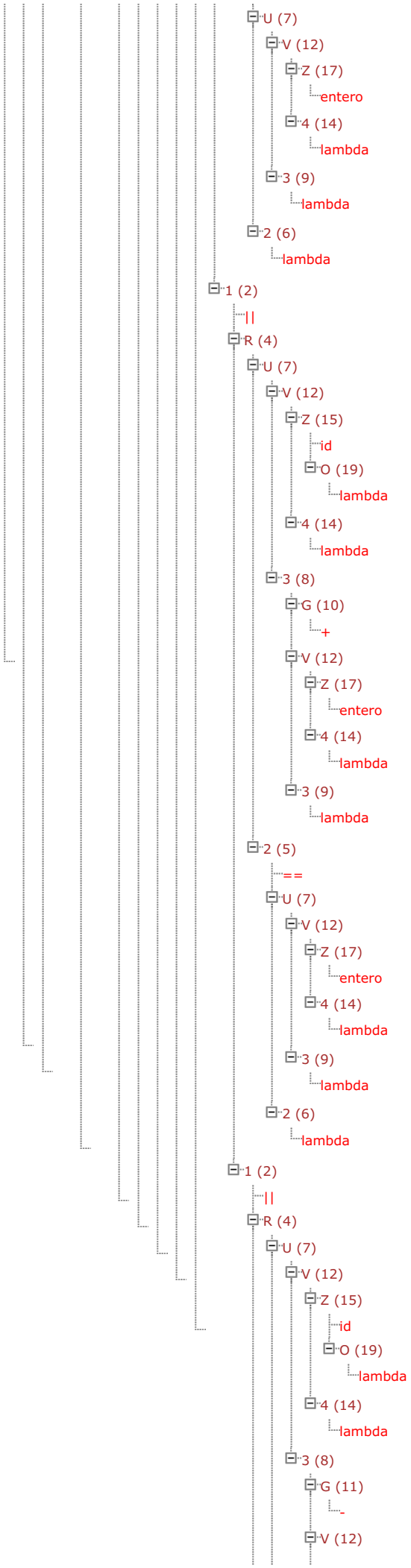
Árbol

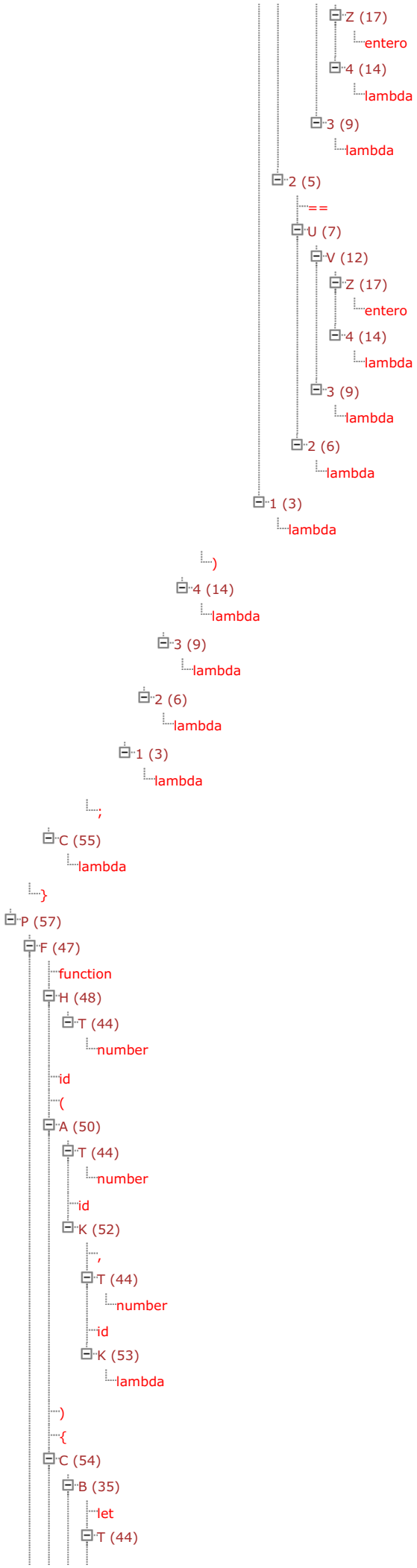
Árbol resultado de:

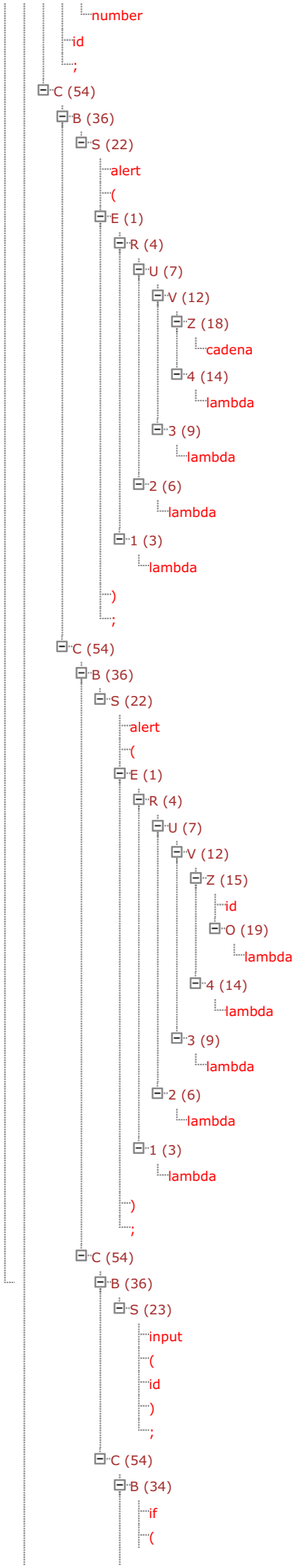
Gramática: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\VisorArbSt\Gramp1.txt

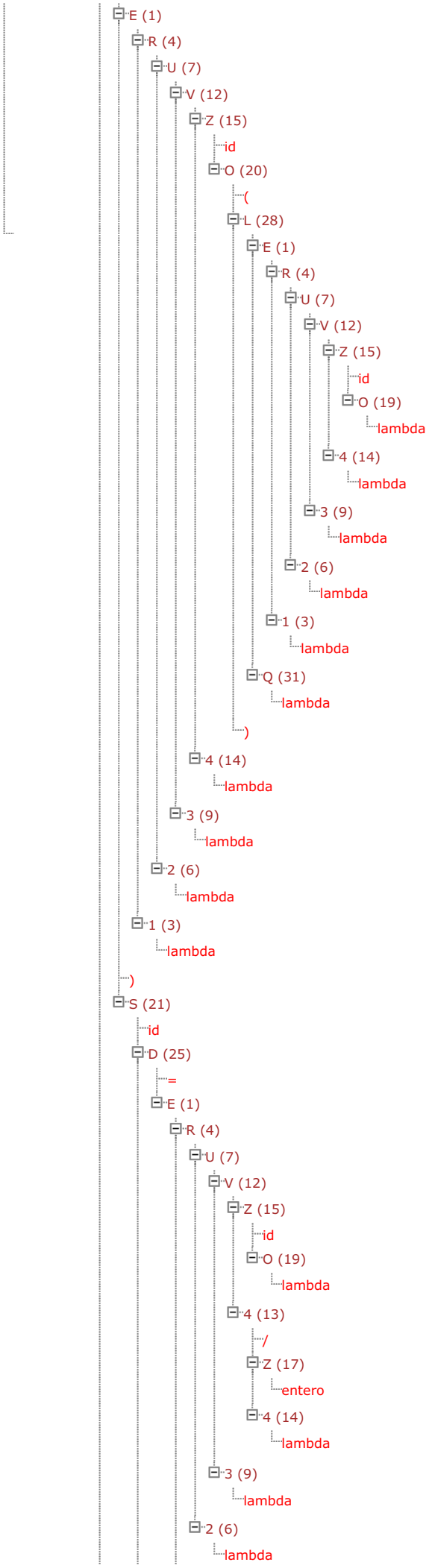
Parse: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\PracticaPDL\parse.txt

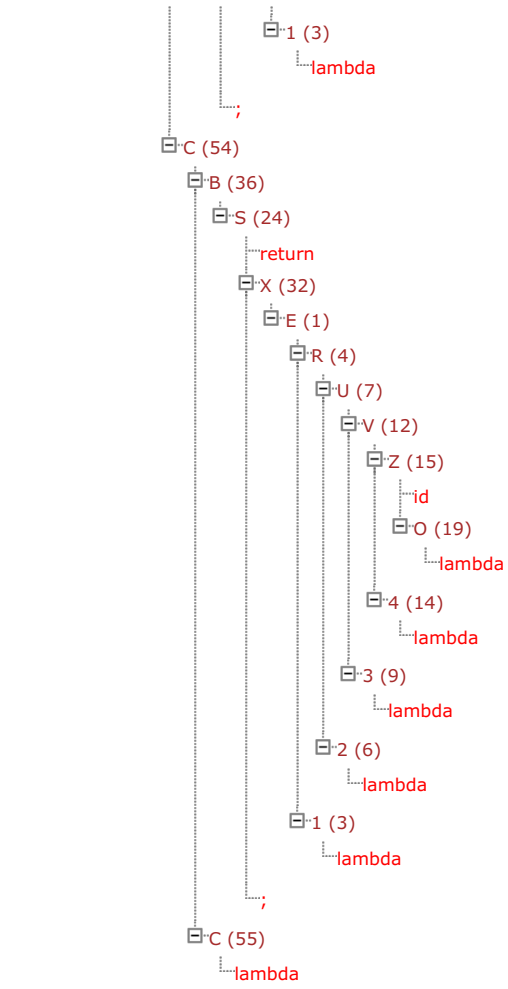


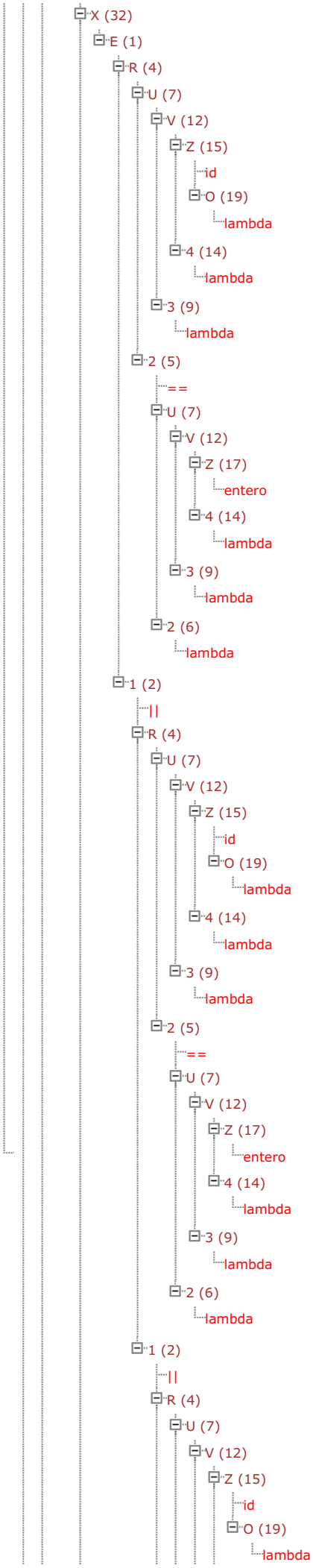

U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (11)
-
V (12)

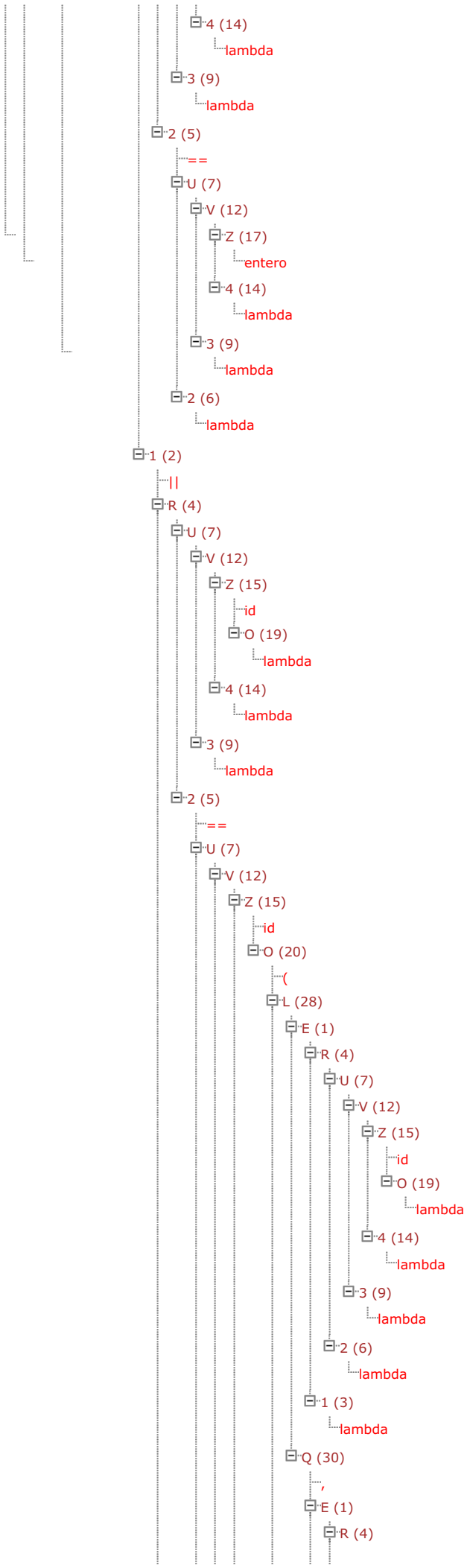

Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (44)
number
id
(
A (50)
T (44)
number
id
K (52)
,
T (44)
number
id
K (53)
lambda
)
{
C (54)
B (35)
let
T (44)

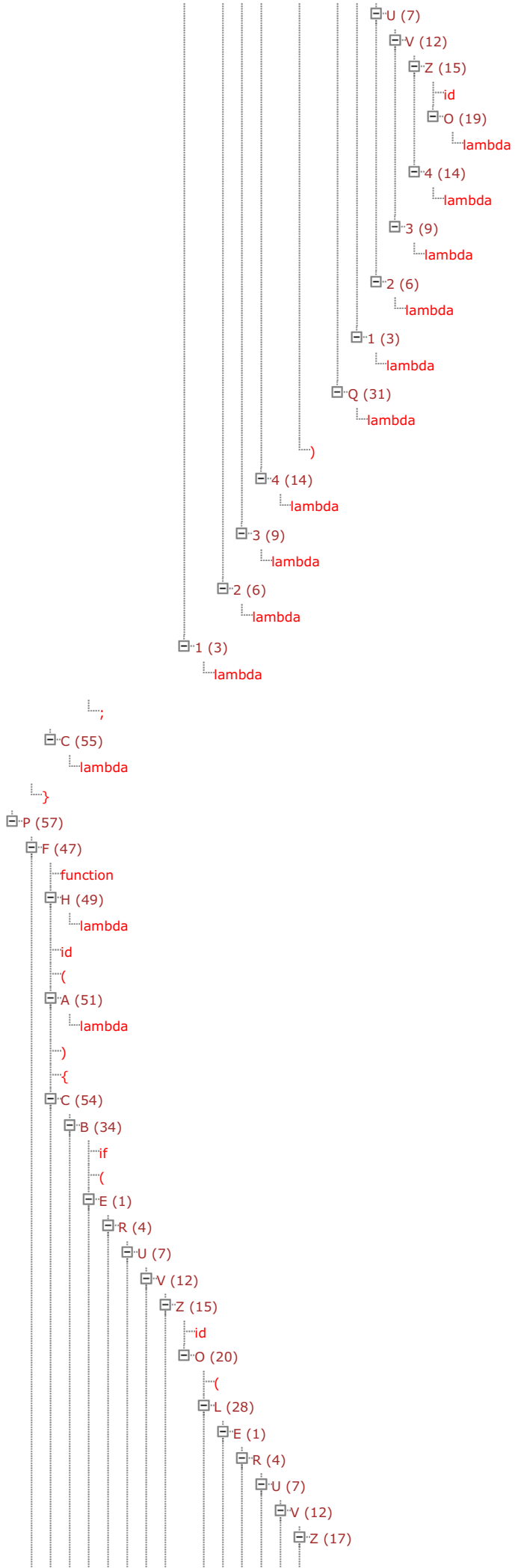

number
id
;
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (18)
cadena
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (23)
input
(
id
)
;
C (54)
B (34)
if
(


E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (13)
/
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda

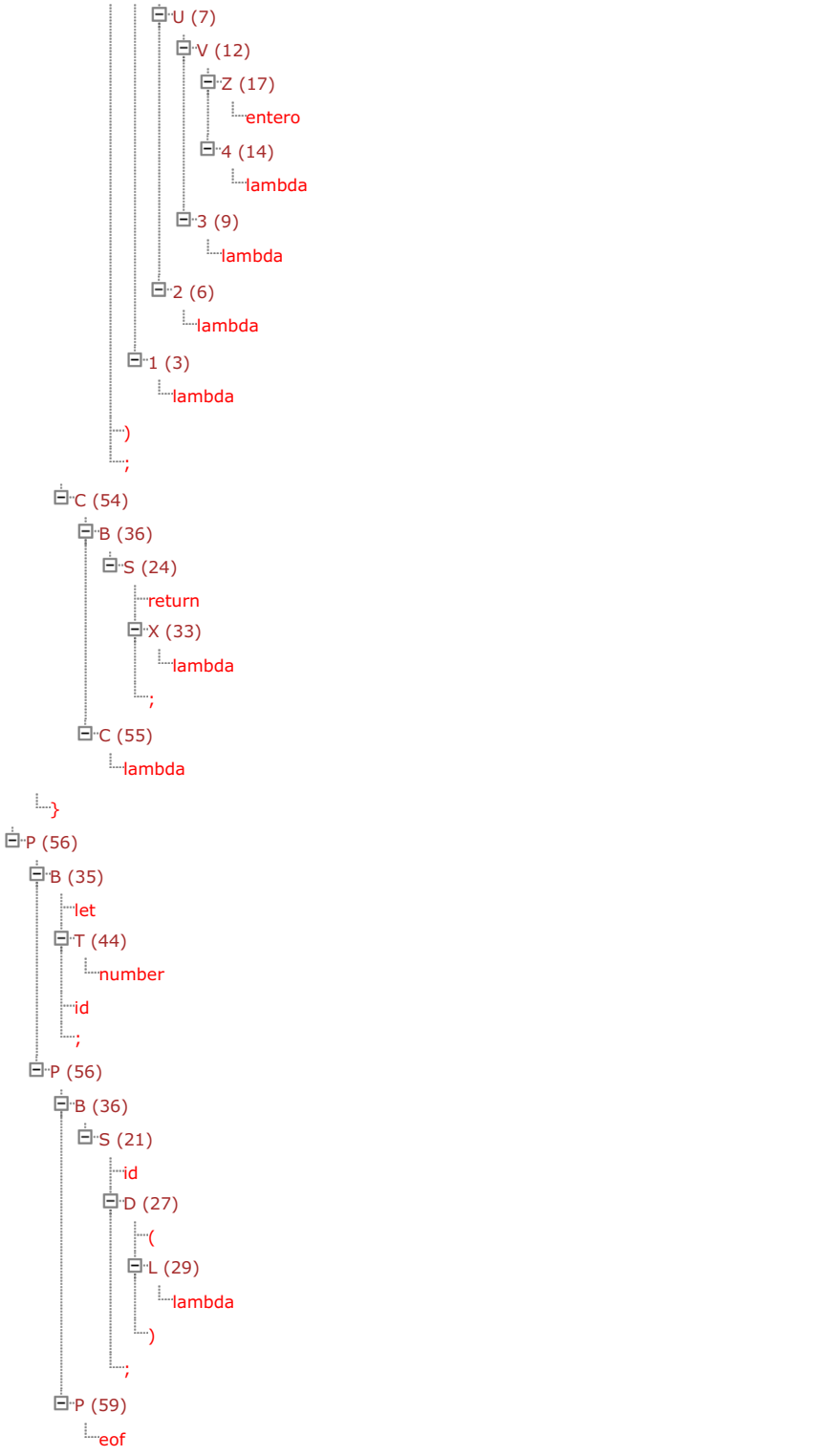

1 (3)
lambda
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (45)
boolean
id
(
A (50)
T (44)
number
id
K (52)
,
T (44)
number
id
K (52)
,
T (44)
number
id
K (53)
lambda
)
{
C (54)
B (36)
S (24)
return


X (32)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda


4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)


U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (49)
lambda
id
(
A (51)
lambda
)
{
C (54)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (17)


entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (22)
alert
(
E (1)
R (4)


U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (24)
return
X (33)
lambda
;
C (55)
lambda
}
P (56)
B (35)
let
T (44)
number
id
;
P (56)
B (36)
S (21)
id
D (27)
(
L (29)
lambda
)
;
P (59)
eof

Prueba 3

Código

```
let boolean booleano;
function boolean bisiesto (number a)
{
      return (a + 4 == 0 || a - 100 == 0 || a / 400 == 0);
}
function number dias (number m, number a)
{
      switch (m)
      {
            case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                  return 31; break;
            case 4: case 6: case 9: case 11:
                  return 30;
            case 2: if (bisiesto (a)) return 29;
                  return(28);
            default:return(0);
      }
}
function boolean esFechaCorrecta (number d, number m, number a)
{
      return (d == dias (m, a));
}
function demo ()
{

      if (esFechaCorrecta(20, 10, 2020)) alert ("ok");
}
let boolean zxy;
demo();
```

Tokens:

```
<LET, >                 <ID, 2>                 <ENTERO, 0>
<BOOLEAN, >             <PARENTC, >             <OR, >
<ID, 0>                 <CORCHA, >              <ID, 2>
<PUNTOYCOMA, >          <RETURN, >              <MENOS, >
<FUNCTION, >            <PARENTA, >             <ENTERO, 100>
<BOOLEAN, >             <ID, 2>                 <IGUALIGUAL, >
<ID, 1>                 <MAS, >                 <ENTERO, 0>
<PARENTA, >             <ENTERO, 4>             <OR, >
<NUMBER, >              <IGUALIGUAL, >          <ID, 2>
```

<DIVISION, >
<ENTERO, 400>
<IGUALIGUAL, >
<ENTERO, 0>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<NUMBER, >
<ID, 3>
<PARENTA, >
<NUMBER, >
<ID, 4>
<COMA, >
<NUMBER, >
<ID, 5>
<PARENTC, >
<CORCHA, >
<SWITCH, >
<PARENTA, >
<ID, 4>
<PARENTC, >
<CORCHA, >
<CASE, >
<ENTERO, 1>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 3>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 5>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 7>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 8>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 10>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 12>
<DOSPUNTOS, >
<RETURN, >
<ENTERO, 31>
<PUNTOYCOMA, >
<BREAK, >
<PUNTOYCOMA, >
<CASE, >
<ENTERO, 4>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 6>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 9>
<DOSPUNTOS, >
<CASE, >
<ENTERO, 11>
<DOSPUNTOS, >
<RETURN, >
<ENTERO, 30>
<PUNTOYCOMA, >
<CASE, >
<ENTERO, 2>
<DOSPUNTOS, >
<IF, >
<PARENTA, >
<ID, 1>
<PARENTA, >
<ID, 5>
<PARENTC, >
<PARENTC, >
<RETURN, >
<ENTERO, 29>
<PUNTOYCOMA, >
<RETURN, >
<PARENTA, >
<ENTERO, 28>
<PARENTC, >
<PUNTOYCOMA, >
<DEFAULT, >
<DOSPUNTOS, >
<RETURN, >
<PARENTA, >
<ENTERO, 0>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<CORCHC, >
<FUNCTION, >
<BOOLEAN, >
<ID, 6>
<PARENTA, >
<NUMBER, >
<ID, 7>
<COMA, >
<NUMBER, >
<ID, 8>
<COMA, >
<NUMBER, >
<ID, 9>
<PARENTC, >
<CORCHA, >
<RETURN, >
<PARENTA, >
<ID, 7>
<IGUALIGUAL, >
<ID, 3>
<PARENTA, >
<ID, 8>
<COMA, >
<ID, 9>
<PARENTC, >
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<ID, 10>
<PARENTA, >
<PARENTC, >
<CORCHA, >
<IF, >
<PARENTA, >
<ID, 6>
<PARENTA, >
<ENTERO, 20>
<COMA, >
<ENTERO, 10>
<COMA, >
<ENTERO, 2020>
<PARENTC, >
<PARENTC, >
<ALERT, >
<PARENTA, >
<CADENA, "ok">
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<LET, >
<BOOLEAN, >
<ID, 11>
<PUNTOYCOMA, >
<ID, 10>
<PARENTA, >
<PARENTC, >
<PUNTOYCOMA, >
<EOF, >

Tabla de símbolos

```
Tabla Programa Principal # 1:
* LEXEMA : 'booleano'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '0'
* LEXEMA : 'bisiesto'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 1
    + TipoParam1 : 'entero'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_bisiesto'
* LEXEMA : 'dias'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'entero'
    + TipoRetorno : 'entero'
    + EtiqFuncion : 'Etiq_dias'
* LEXEMA : 'esFechaCorrecta'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 3
    + TipoParam1 : 'entero'
    + TipoParam2 : 'entero'
    + TipoParam3 : 'entero'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_esFechaCorrecta'
* LEXEMA : 'demo'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 0
    + TipoRetorno : 'void'
    + EtiqFuncion : 'Etiq_demo'
* LEXEMA : 'zxy'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'

Tabla Funcion bisiesto # 2:
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
```

```
Tabla Funcion dias # 3:
* LEXEMA : 'm'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'

Tabla Funcion esFechaCorrecta # 4:
* LEXEMA : 'd'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'm'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'
* LEXEMA : 'a'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '2'

Tabla Funcion demo # 5:
```

Parse

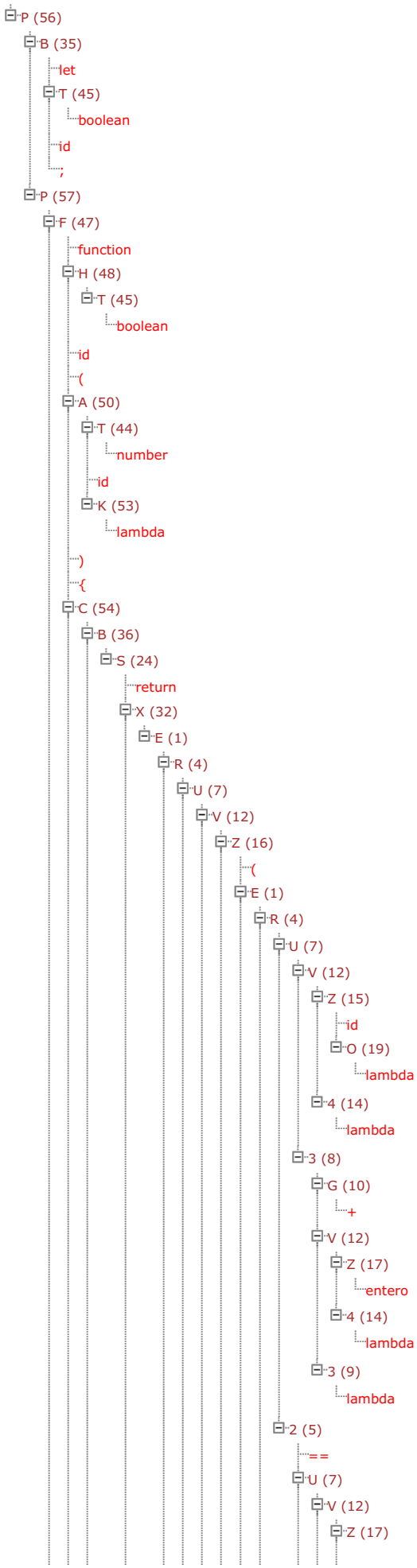
```
Descendente 56 35 45 57 47 48 45 50 44 53 54 36 24 32 1 4 7 12 16 1 4 7
12 15 19 14 8 10 12 17 14 9 5 7 12 17 14 9 6 2 4 7 12 15 19 14 8 11 12
17 14 9 5 7 12 17 14 9 6 2 4 7 12 15 19 13 17 14 9 5 7 12 17 14 9 6 3
14 9 6 3 55 57 47 48 44 50 44 52 44 53 54 37 1 4 7 12 15 19 14 9 6 3 38
55 40 41 38 55 40 41 38 55 40 41 38 55 40 41 38 55 40 41 38 55 40 41 38
54 36 24 32 1 4 7 12 17 14 9 6 3 55 39 41 38 55 40 41 38 55 40 41 38 55
40 41 38 54 36 24 32 1 4 7 12 17 14 9 6 3 55 40 41 38 54 34 1 4 7 12 15
20 28 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 24 32 1 4 7 12 17 14 9 6 3 54
36 24 32 1 4 7 12 16 1 4 7 12 17 14 9 6 3 14 9 6 3 55 40 42 54 36 24 32
1 4 7 12 16 1 4 7 12 17 14 9 6 3 14 9 6 3 55 55 57 47 48 45 50 44 52 44
52 44 53 54 36 24 32 1 4 7 12 16 1 4 7 12 15 19 14 9 5 7 12 15 20 28 1
4 7 12 15 19 14 9 6 3 30 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 14 9 6 3
55 57 47 49 51 54 34 1 4 7 12 15 20 28 1 4 7 12 17 14 9 6 3 30 1 4 7 12
17 14 9 6 3 30 1 4 7 12 17 14 9 6 3 31 14 9 6 3 22 1 4 7 12 18 14 9 6 3
55 56 35 45 56 36 21 27 29 59
```

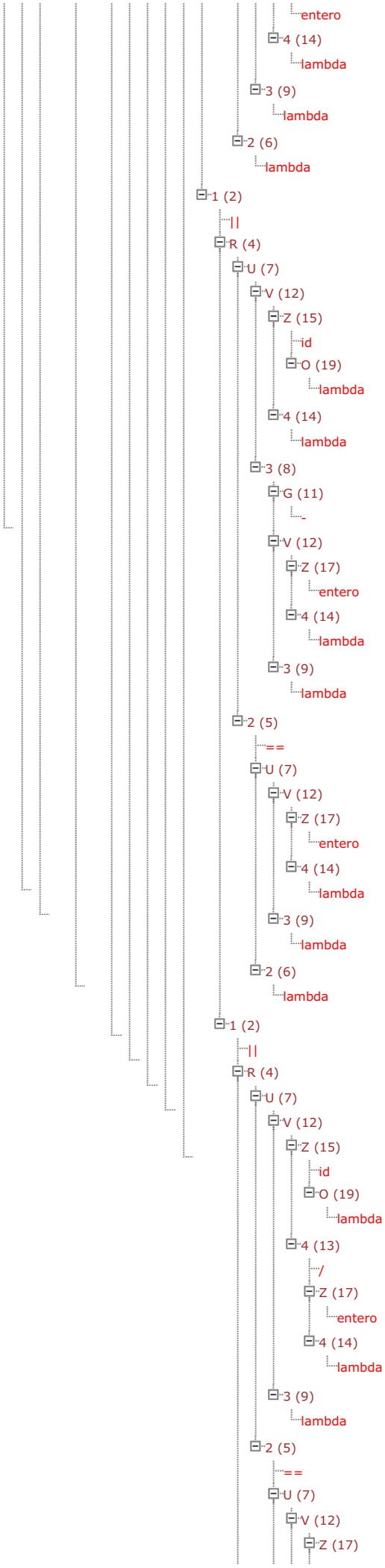
Árbol

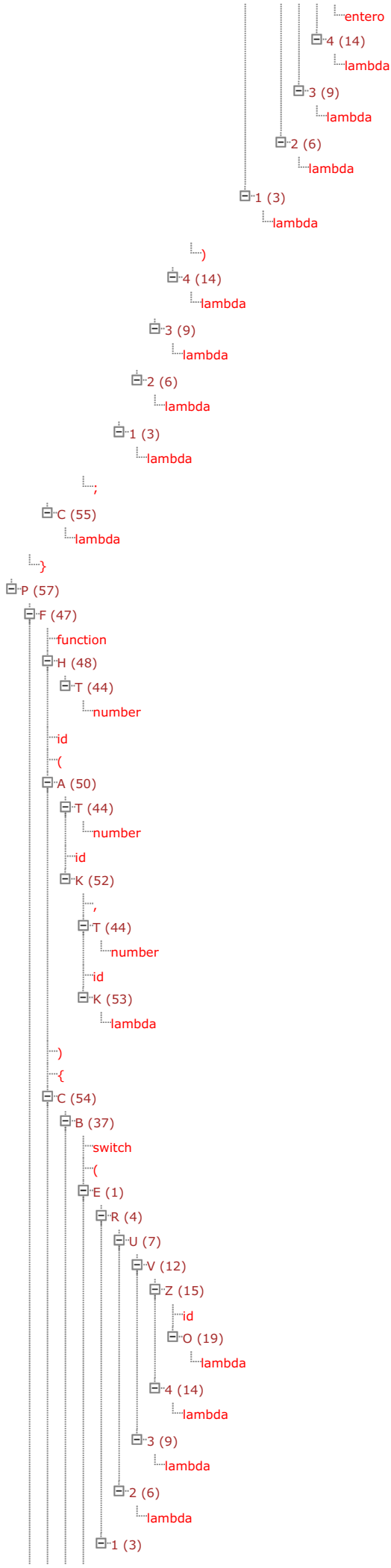
Árbol resultado de:

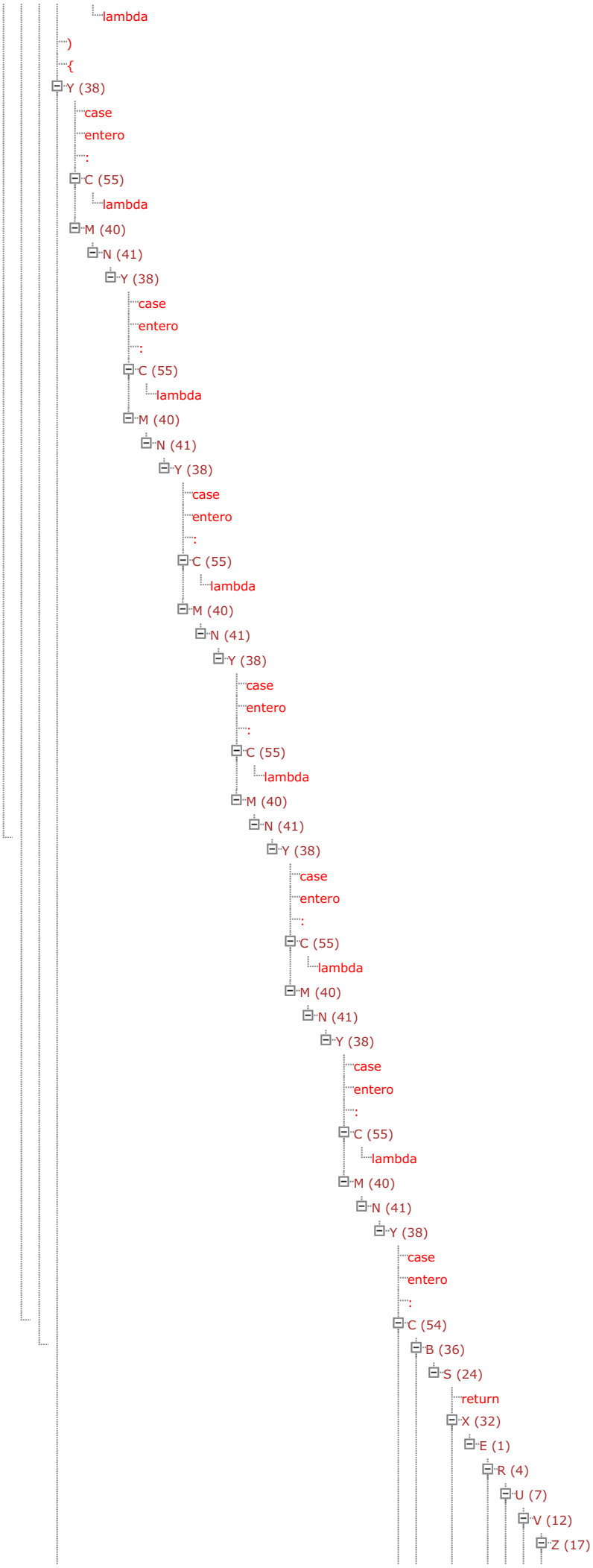
Gramática: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\VisorArbSt\Gramp1.txt

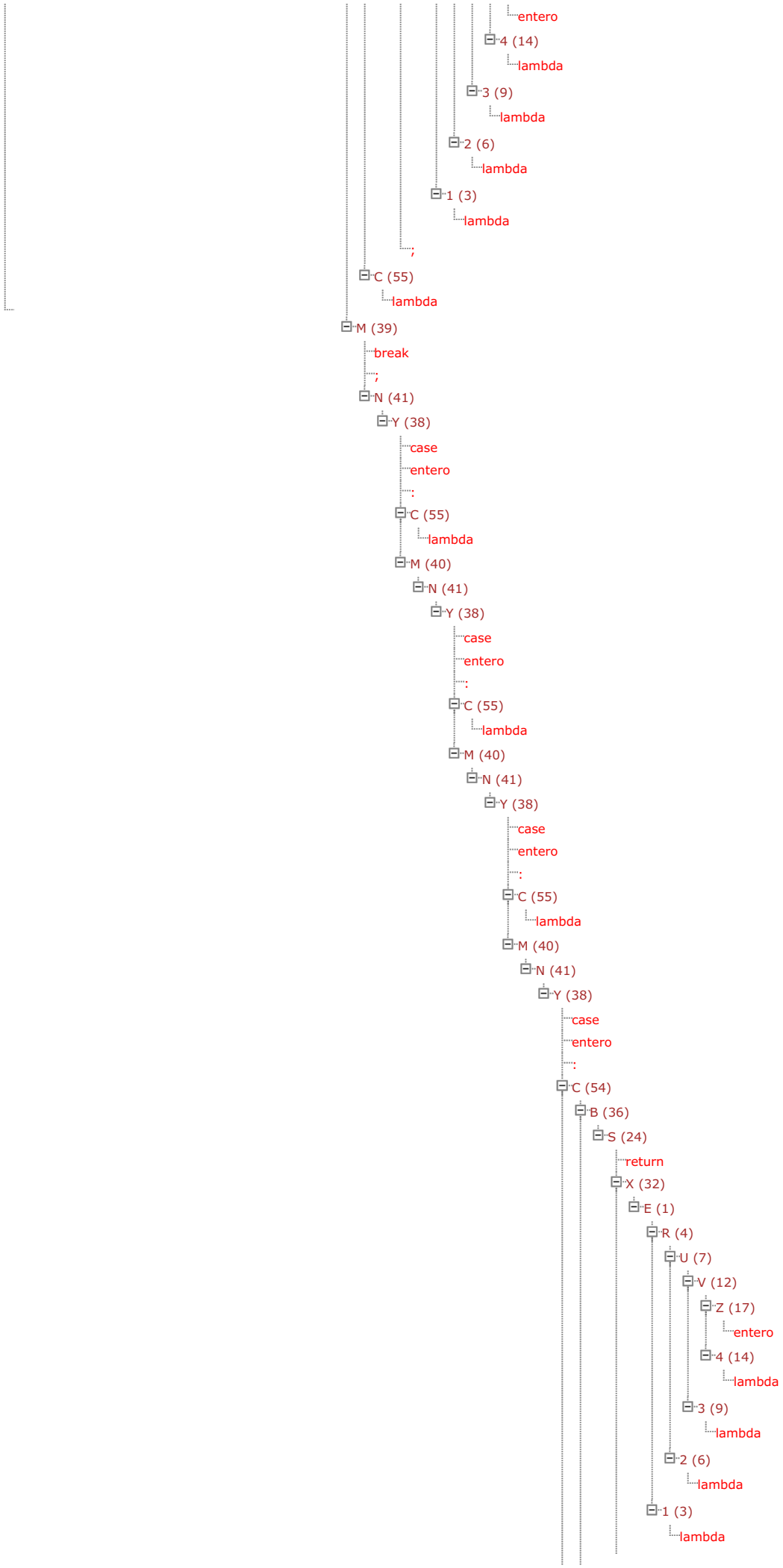
Parse: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\PracticaPDL\parse.txt

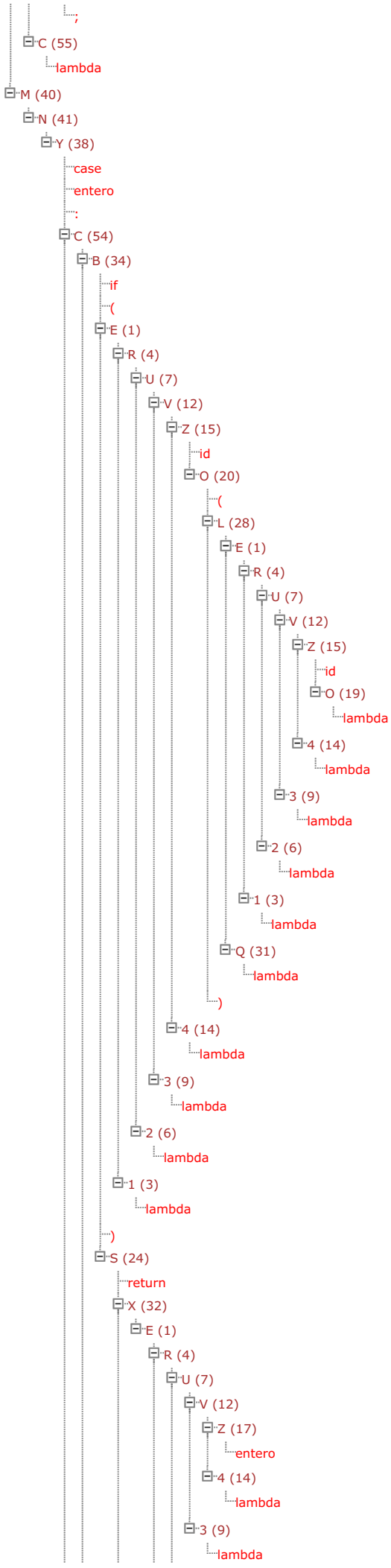


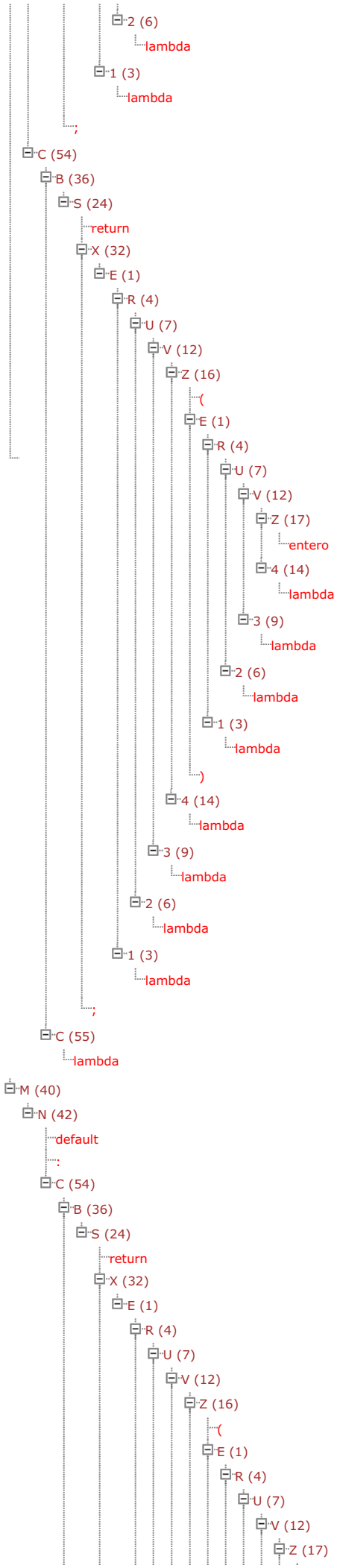

entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (13)
/
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (17)

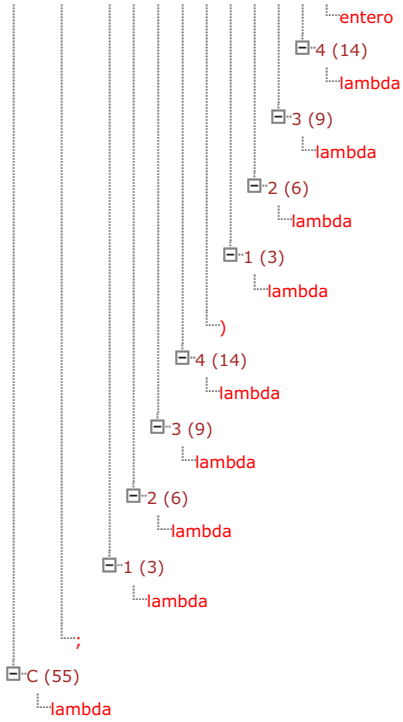

entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (44)
number
id
(
A (50)
T (44)
number
id
K (52)
,
T (44)
number
id
K (53)
lambda
)
{
C (54)
B (37)
switch
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)

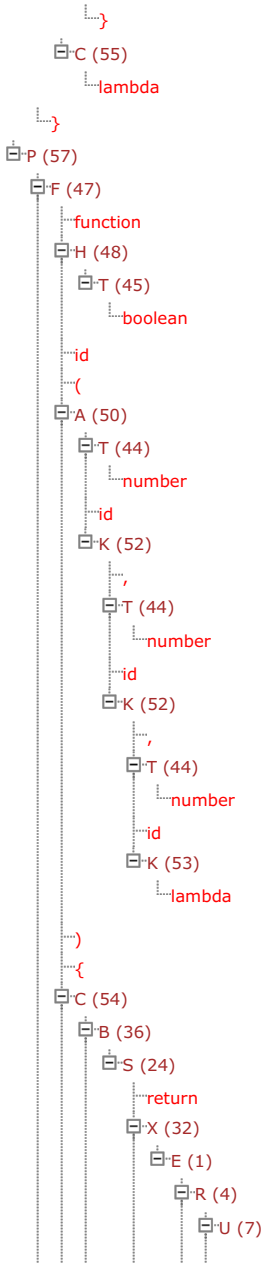

lambda
)
{
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (17)

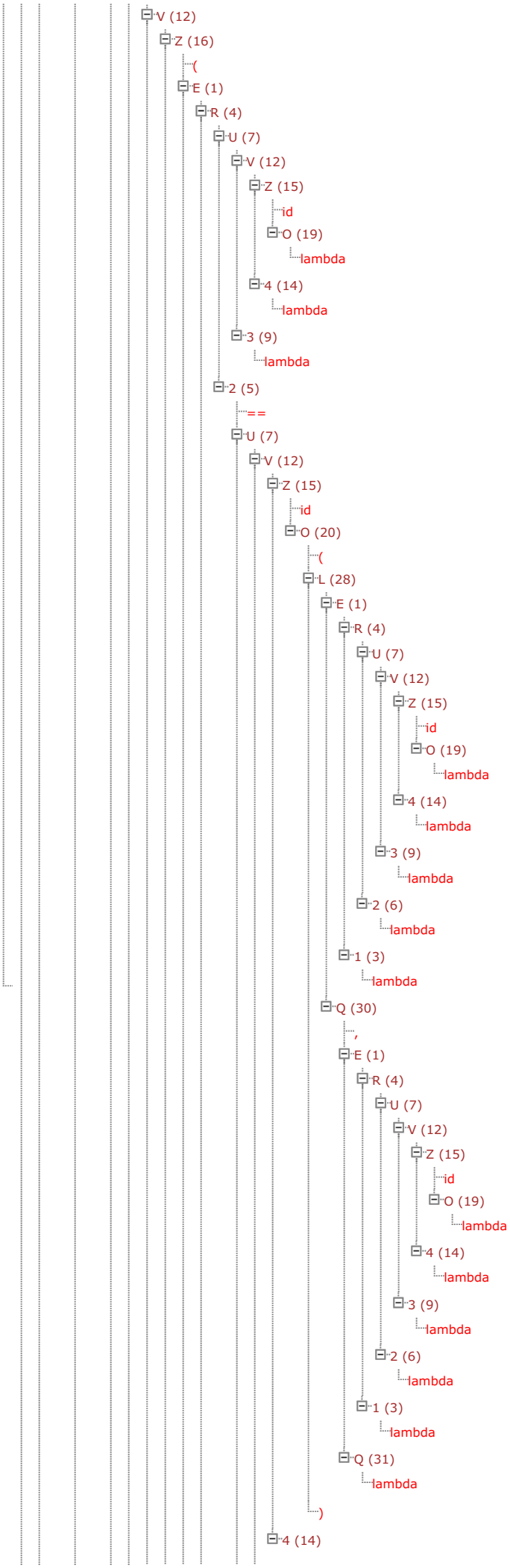

entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
M (39)
break
;
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda

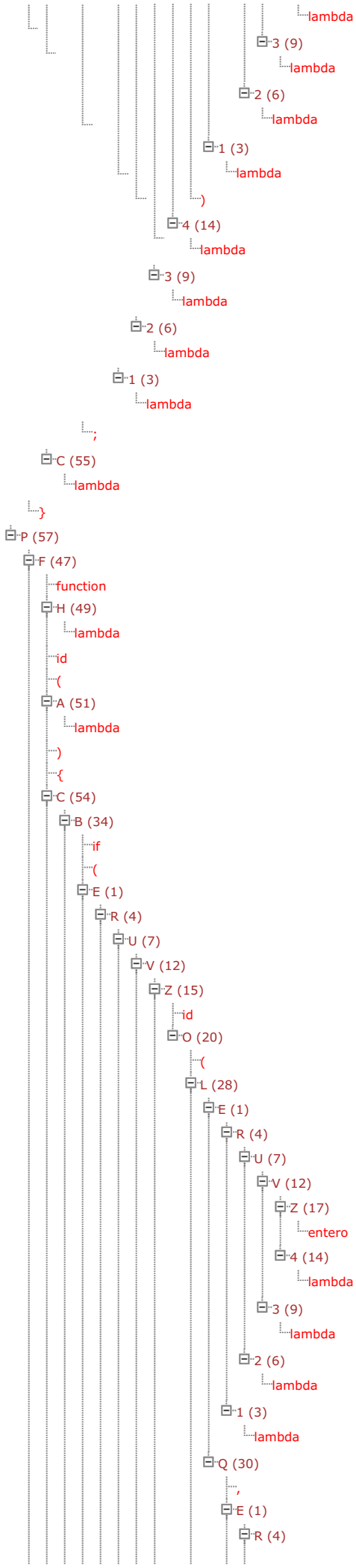

;
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (54)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda

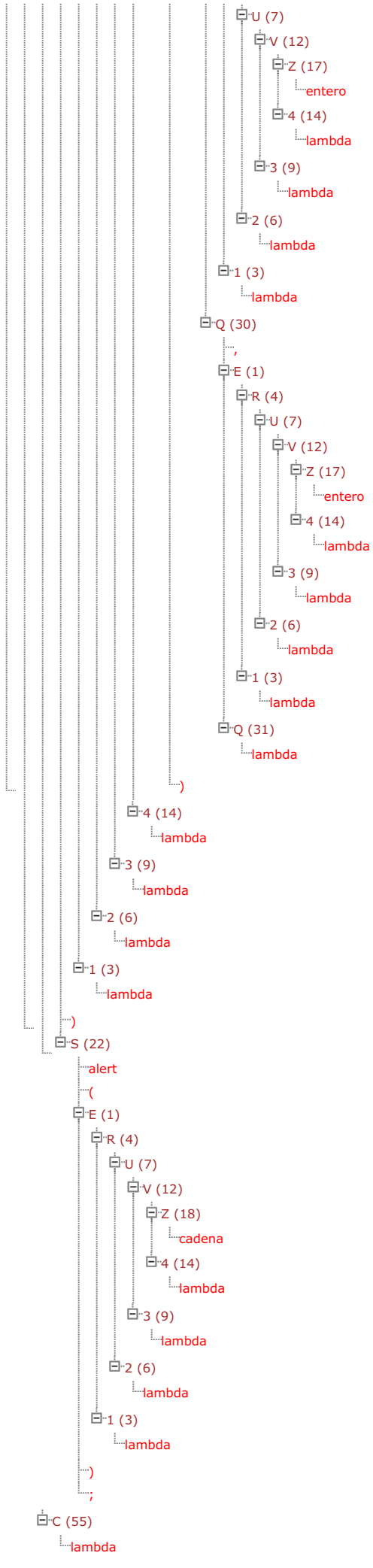

2 (6)
lambda
1 (3)
lambda
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
M (40)
N (42)
default
:
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)

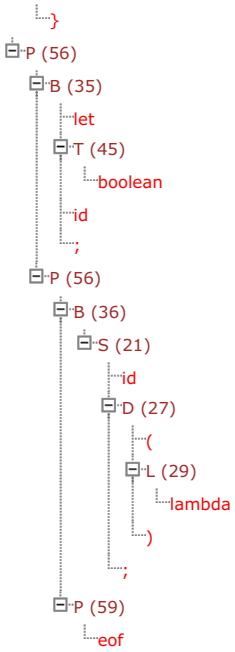

entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda


}
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (45)
boolean
id
(
A (50)
T (44)
number
id
K (52)
,
T (44)
number
id
K (52)
,
T (44)
number
id
K (53)
lambda
)
{
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)


V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)


lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (49)
lambda
id
(
A (51)
lambda
)
{
C (54)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)


U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (18)
cadena
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (55)
lambda


}
P (56)
B (35)
let
T (45)
boolean
id
;
P (56)
B (36)
S (21)
id
D (27)
(
L (29)
lambda
)
;
P (59)
eof

Prueba 4

Código

```
let number x;
let number xx;
let string ss;
let boolean boolean_1;
let boolean boolean_2;
let number y;

function number f1(number f1, boolean b1)
{
            alert(ss);
            x = xx/f1;
            boolean_1 = boolean_1|| boolean_2;
            return (01234);
}


function boolean f2(number f2, boolean b1)
{
            alert ((04-5+77/(88-f2)));
            return boolean_1||boolean_2|| b1;
}

x =
  x + 6
    - z
    - 1
    / (2
    - y
    / 6)
      ;


alert (f1 (x, f2 (3, boolean_2)));
```

Tokens

```
<LET, >                 <STRING, >              <ID, 4>
<NUMBER, >              <ID, 2>                 <PUNTOYCOMA, >
<ID, 0>                 <PUNTOYCOMA, >          <LET, >
<PUNTOYCOMA, >          <LET, >                 <NUMBER, >
<LET, >                 <BOOLEAN, >             <ID, 5>
<NUMBER, >              <ID, 3>                 <PUNTOYCOMA, >
<ID, 1>                 <PUNTOYCOMA, >          <FUNCTION, >
<PUNTOYCOMA, >          <LET, >                 <NUMBER, >
<LET, >                 <BOOLEAN, >             <ID, 6>
```

```
<PARENTA, >
<NUMBER, >
<ID, 7>
<COMA, >
<BOOLEAN, >
<ID, 8>
<PARENTC, >
<CORCHA, >
<ALERT, >
<PARENTA, >
<ID, 2>
<PARENTC, >
<PUNTOYCOMA, >
<ID, 0>
<IGUAL, >
<ID, 1>
<DIVISION, >
<ID, 7>
<PUNTOYCOMA, >
<ID, 3>
<IGUAL, >
<ID, 3>
<OR, >
<ID, 4>
<PUNTOYCOMA, >
<RETURN, >
<PARENTA, >
<ENTERO, 1234>
<PARENTC, >
<PUNTOYCOMA, >
<CORCHC, >
<FUNCTION, >
<BOOLEAN, >
<ID, 9>
<PARENTA, >
<NUMBER, >
<ID, 10>
<COMA, >
<BOOLEAN, >
<ID, 11>
<PARENTC, >
<CORCHA, >
<ALERT, >
<PARENTA, >
<PARENTA, >
<ENTERO, 4>
<MENOS, >
<ENTERO, 5>
<MAS, >
<ENTERO, 77>
<DIVISION, >
<PARENTA, >
<ENTERO, 88>
<MENOS, >
<ID, 10>
<PARENTC, >
<PARENTC, >
<PARENTC, >
<PUNTOYCOMA, >
<RETURN, >
<ID, 3>
<OR, >
<ID, 4>
<OR, >
<ID, 11>
<PUNTOYCOMA, >
<CORCHC, >
<ID, 0>
<IGUAL, >
<ID, 0>
<MAS, >
<ENTERO, 6>
<MENOS, >
<ID, 12>
<MENOS, >
<ENTERO, 1>
<DIVISION, >
<PARENTA, >
<ENTERO, 2>
<MENOS, >
<ID, 5>
<DIVISION, >
<ENTERO, 6>
<PARENTC, >
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<ID, 6>
<PARENTA, >
<ID, 0>
<COMA, >
<ID, 9>
<PARENTA, >
<ENTERO, 3>
<COMA, >
<ID, 4>
<PARENTC, >
<PARENTC, >
<PARENTC, >
<PUNTOYCOMA, >
<EOF, >
```

Tabla de símbolos

```
Tabla Programa Principal # 1:
* LEXEMA : 'x'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'xx'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '1'
* LEXEMA : 'ss'
  ATRIBUTOS :
    + tipo : 'string'
    + despl : '2'
```

```
* LEXEMA : 'boolean_1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '66'
* LEXEMA : 'boolean_2'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '67'
* LEXEMA : 'y'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '68'
* LEXEMA : 'f1'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'boolean'
    + TipoRetorno : 'entero'
    + EtiqFuncion : 'Etiq_f1'
* LEXEMA : 'f2'
  ATRIBUTOS :
    + tipo : 'function'
    + numParam : 2
    + TipoParam1 : 'entero'
    + TipoParam2 : 'boolean'
    + TipoRetorno : 'boolean'
    + EtiqFuncion : 'Etiq_f2'
* LEXEMA : 'z'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '69'

Tabla Funcion f1 # 2:
* LEXEMA : 'f1'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'b1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'
Tabla Funcion f2 # 3:
* LEXEMA : 'f2'
  ATRIBUTOS :
    + tipo : 'entero'
    + despl : '0'
* LEXEMA : 'b1'
  ATRIBUTOS :
    + tipo : 'boolean'
    + despl : '1'
```

Parse

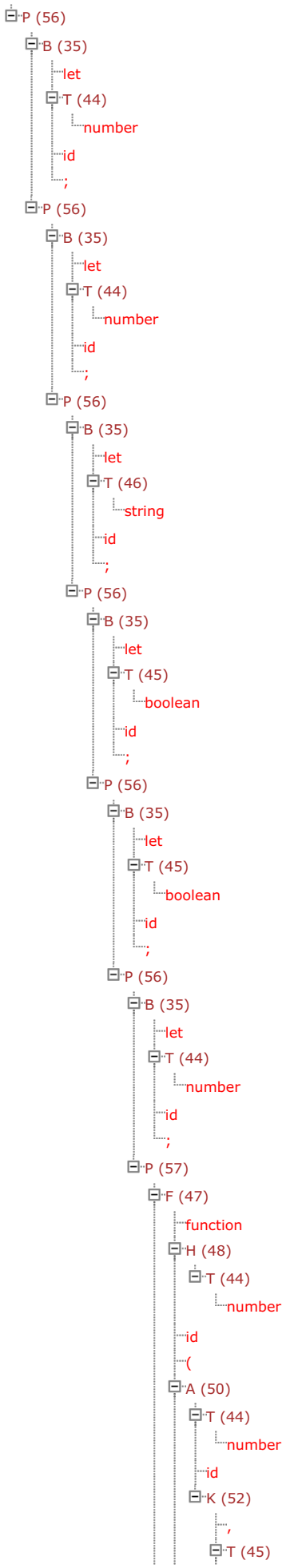
```
Descendente 56 35 44 56 35 44 56 35 46 56 35 45 56 35 45 56 35 44 57 47
48 44 50 44 52 45 53 54 36 22 1 4 7 12 15 19 14 9 6 3 54 36 21 25 1 4 7
12 15 19 13 15 19 14 9 6 3 54 36 21 25 1 4 7 12 15 19 14 9 6 2 4 7 12
15 19 14 9 6 3 54 36 24 32 1 4 7 12 16 1 4 7 12 17 14 9 6 3 14 9 6 3 55
57 47 48 45 50 44 52 45 53 54 36 22 1 4 7 12 16 1 4 7 12 17 14 8 11 12
17 14 8 10 12 17 13 16 1 4 7 12 17 14 8 11 12 15 19 14 9 6 3 14 9 6 3
14 9 6 3 54 36 24 32 1 4 7 12 15 19 14 9 6 2 4 7 12 15 19 14 9 6 2 4 7
12 15 19 14 9 6 3 55 56 36 21 25 1 4 7 12 15 19 14 8 10 12 17 14 8 11
12 15 19 14 8 11 12 17 13 16 1 4 7 12 17 14 8 11 12 15 19 13 17 14 9 6
3 14 9 6 3 56 36 22 1 4 7 12 15 20 28 1 4 7 12 15 19 14 9 6 3 30 1 4 7
12 15 20 28 1 4 7 12 17 14 9 6 3 30 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3
31 14 9 6 3 59
```

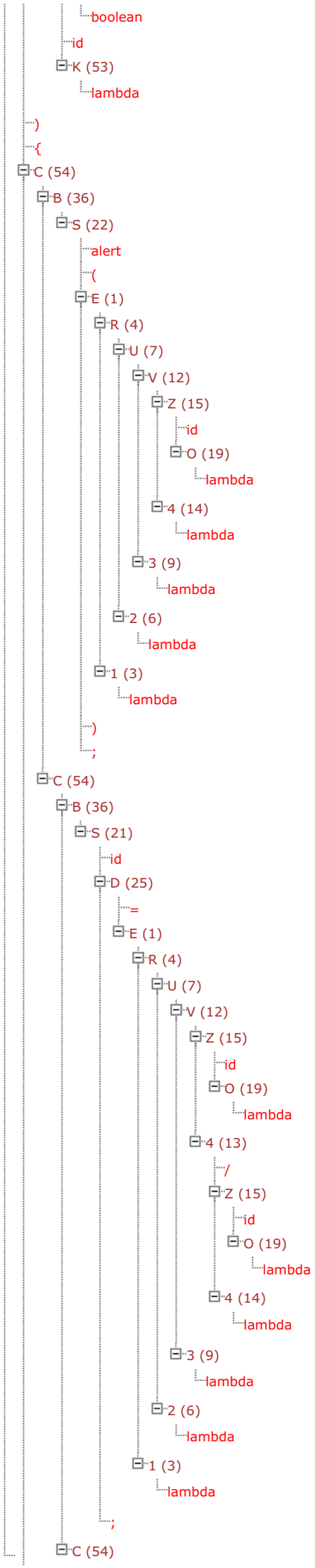
Árbol

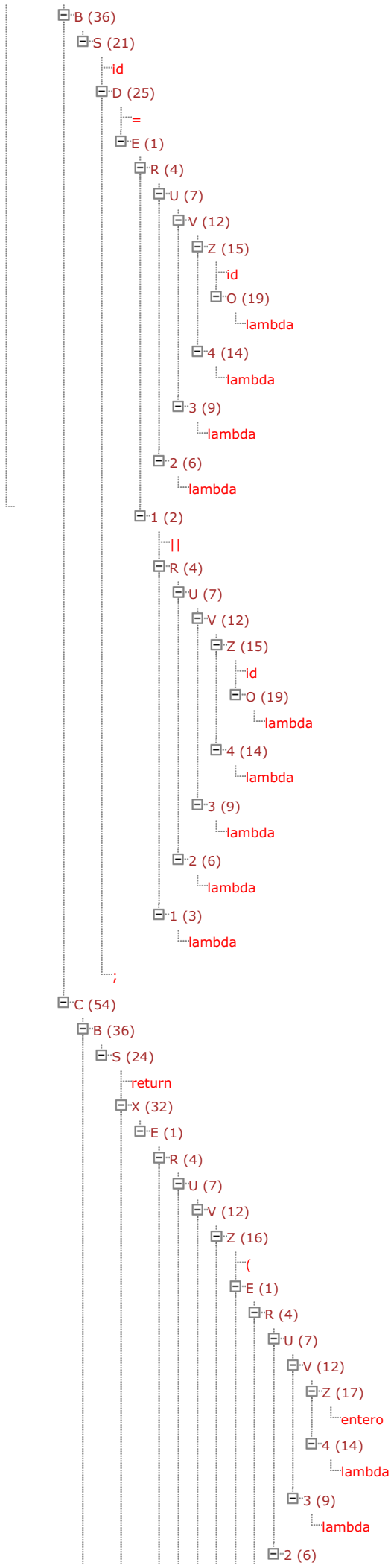
Árbol resultado de:

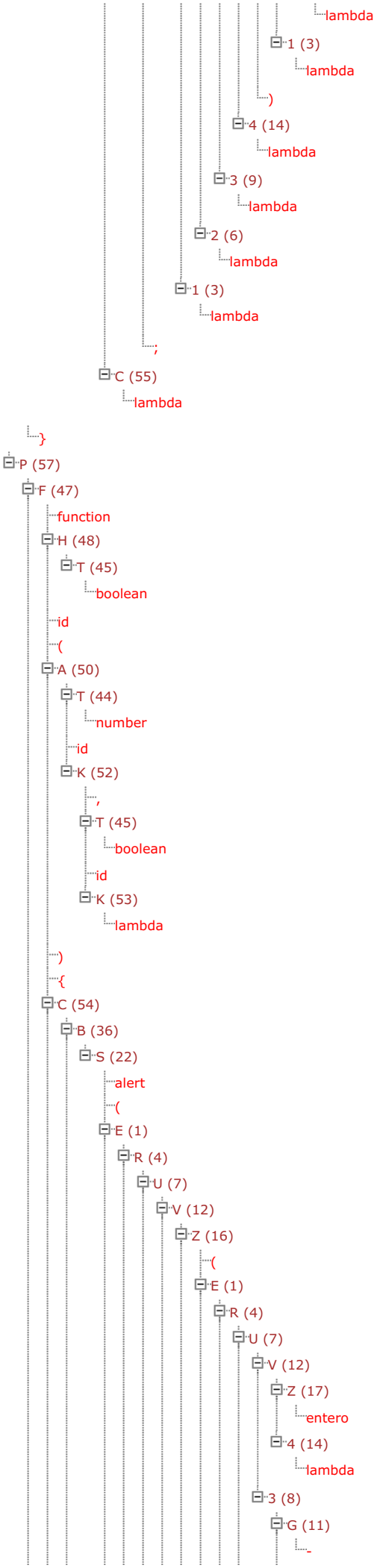
Gramática: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\VisorArbSt\Gramp1.txt

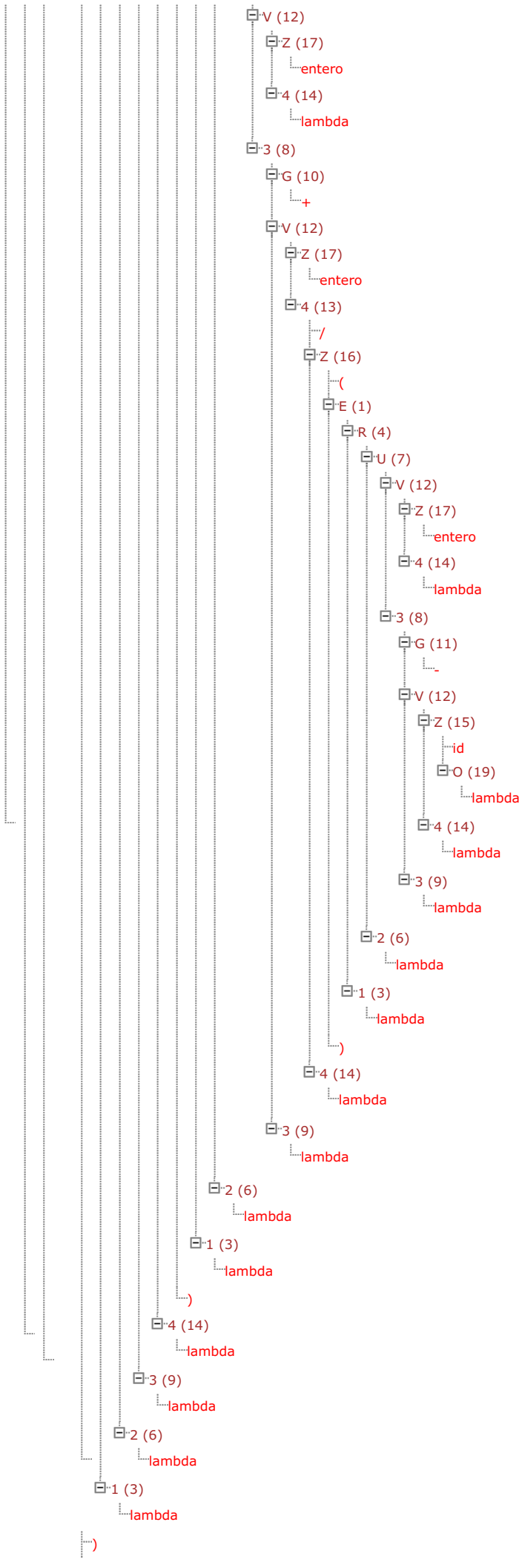
Parse: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\PracticaPDL\parse.txt

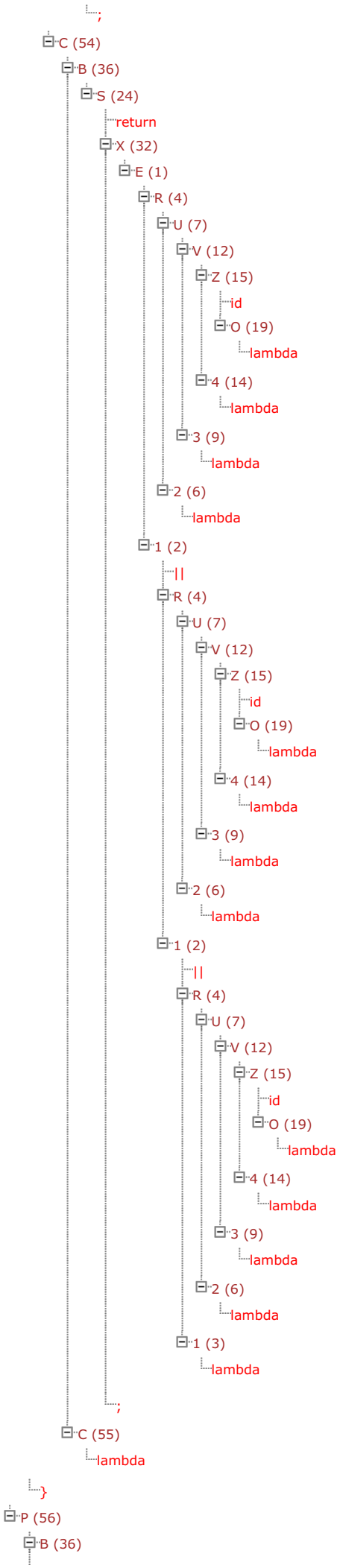


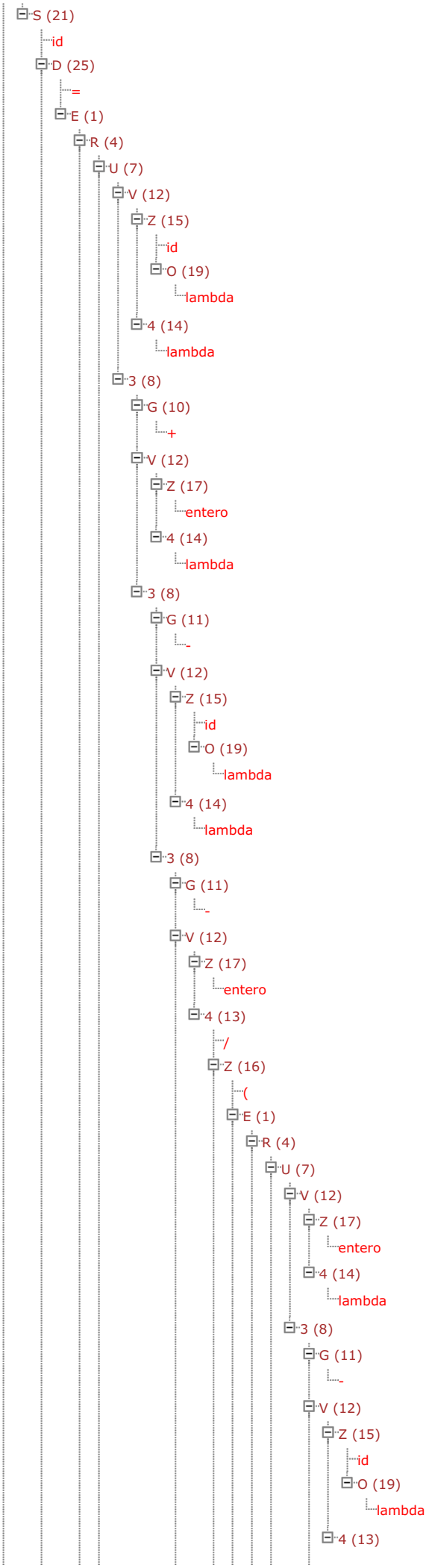

boolean
id
K (53)
lambda
)
{
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (54)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (13)
/
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)

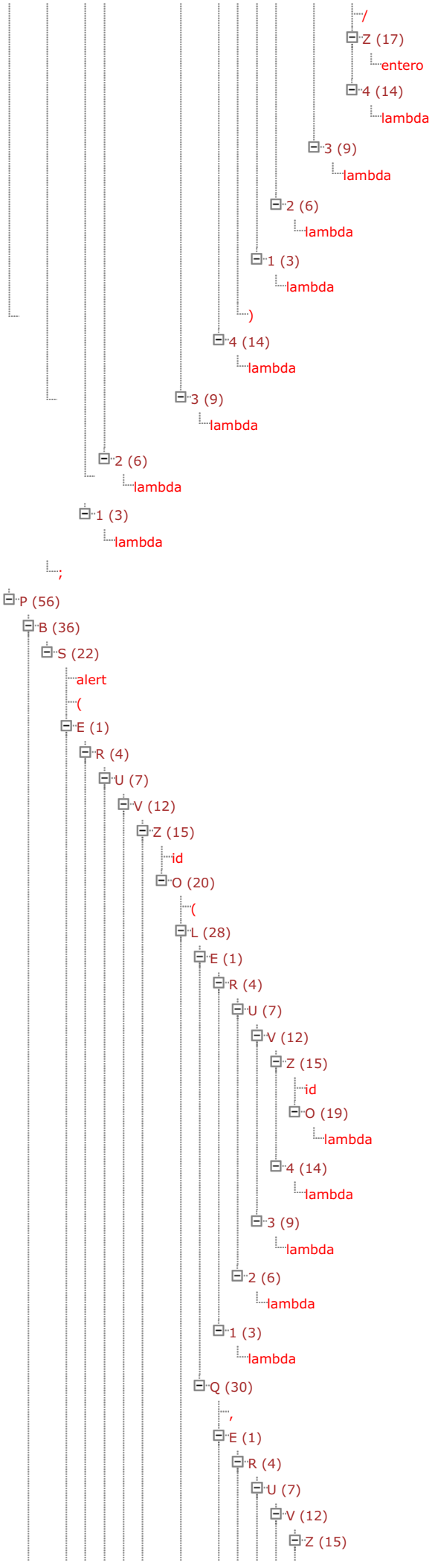

B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)

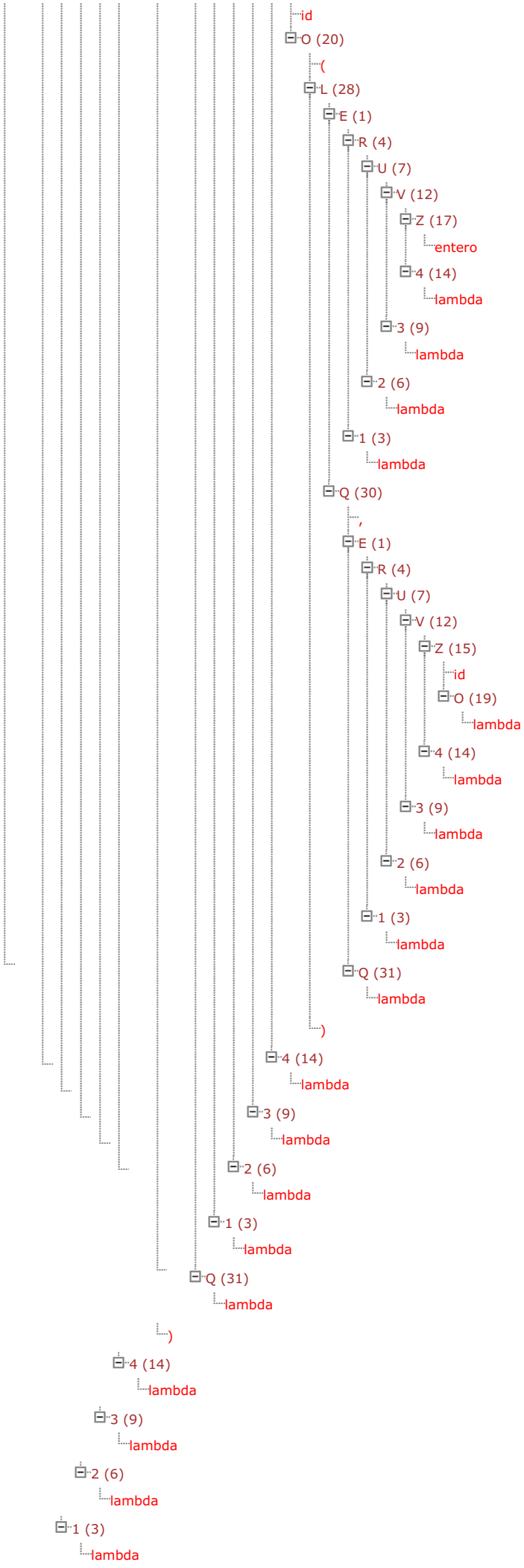

lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (45)
boolean
id
(
A (50)
T (44)
number
id
K (52)
,
T (45)
boolean
id
K (53)
lambda
)
{
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-

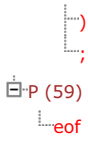

V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (13)
/
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)


;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (56)
B (36)


S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (17)
entero
4 (13)
/
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (13)


/
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (15)


id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (30)
,
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda


)
;
P (59)
eof

Prueba 5

Código

```
let number a;
let number b;
let boolean bbb;
a = 3;
b = a;
if (a == b) b = 1;
if (b == a) b = 8888;
a = a - b;
alert (a);
alert(b);
```

Tokens

```
<LET, >
<NUMBER, >
<ID, 0>
<PUNTOYCOMA, >
<LET, >
<NUMBER, >
<ID, 1>
<PUNTOYCOMA, >
<LET, >
<BOOLEAN, >
<ID, 2>
<PUNTOYCOMA, >
<ID, 0>
<IGUAL, >
<ENTERO, 3>
<PUNTOYCOMA, >
<ID, 1>
<IGUAL, >
<ID, 0>
<PUNTOYCOMA, >
<IF, >
<PARENTA, >
<ID, 0>
<IGUALIGUAL, >
<ID, 1>
<PARENTC, >
<ID, 1>
<IGUAL, >
<ENTERO, 1>
<PUNTOYCOMA, >
<IF, >
<PARENTA, >
<ID, 1>
<IGUALIGUAL, >
<ID, 0>
<PARENTC, >
<ID, 1>
<IGUAL, >
<ENTERO, 8888>
<PUNTOYCOMA, >
<ID, 0>
<IGUAL, >
<ID, 0>
<MENOS, >
<ID, 1>
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<ID, 0>
<PARENTC, >
<PUNTOYCOMA, >
<ALERT, >
<PARENTA, >
<ID, 1>
<PARENTC, >
<PUNTOYCOMA, >
<EOF, >
```

Parse

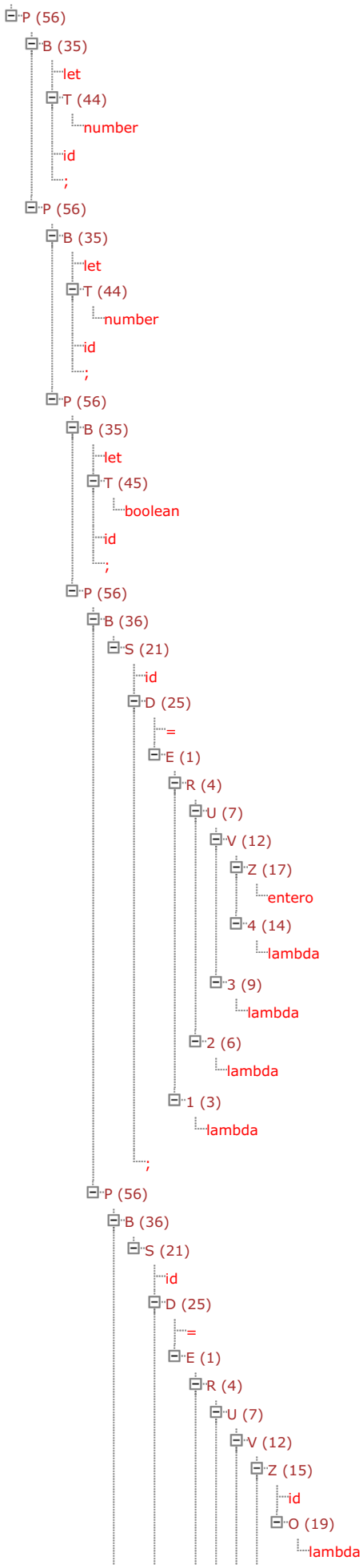
```
Descendente 56 35 44 56 35 44 56 35 45 56 36 21 25 1 4 7 12 17 14 9 6 3
56 36 21 25 1 4 7 12 15 19 14 9 6 3 56 34 1 4 7 12 15 19 14 9 5 7 12 15
19 14 9 6 3 21 25 1 4 7 12 17 14 9 6 3 56 34 1 4 7 12 15 19 14 9 5 7 12
15 19 14 9 6 3 21 25 1 4 7 12 17 14 9 6 3 56 36 21 25 1 4 7 12 15 19 14
8 11 12 15 19 14 9 6 3 56 36 22 1 4 7 12 15 19 14 9 6 3 56 36 22 1 4 7
12 15 19 14 9 6 3 59
```

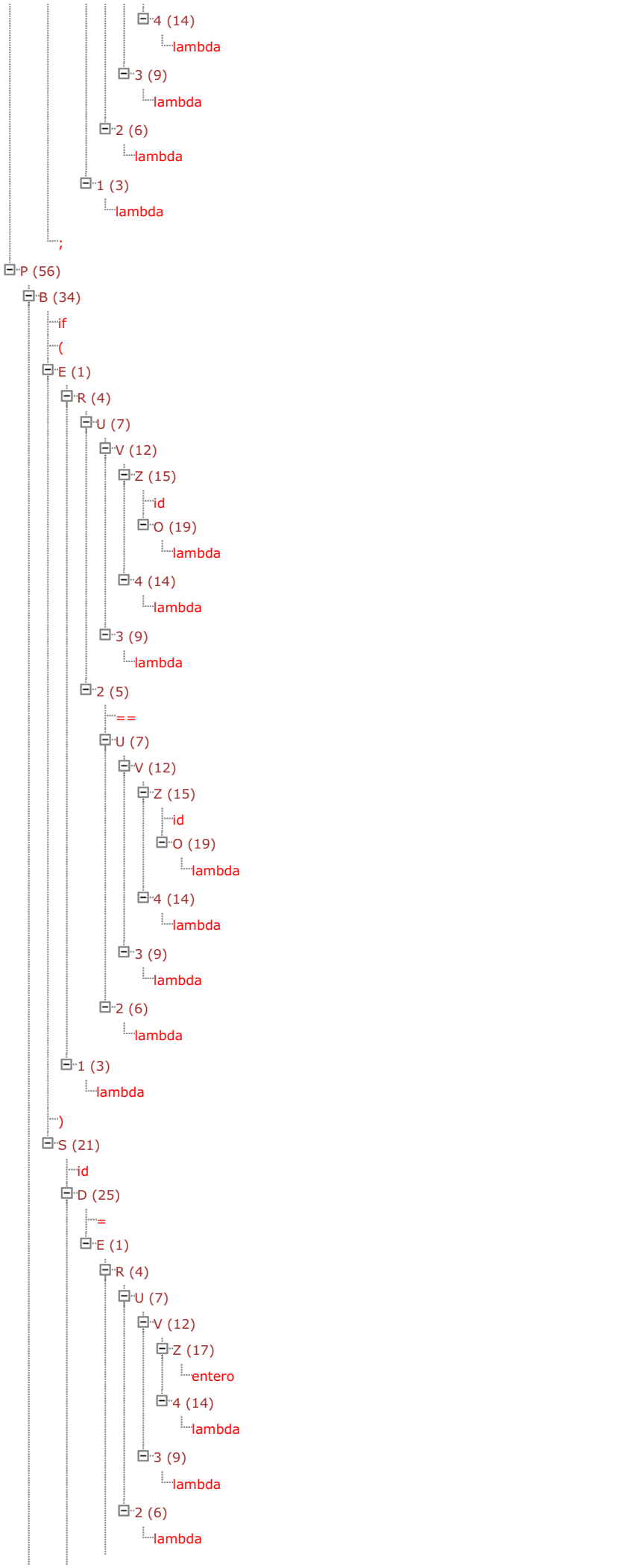
Árbol

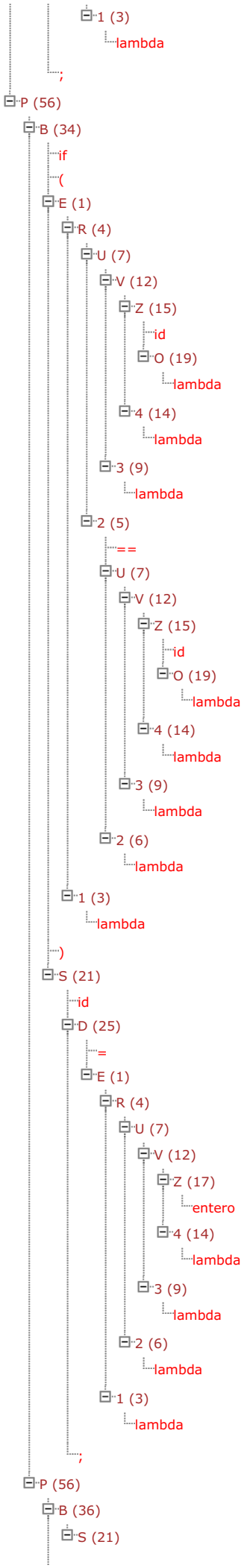
Árbol resultado de:

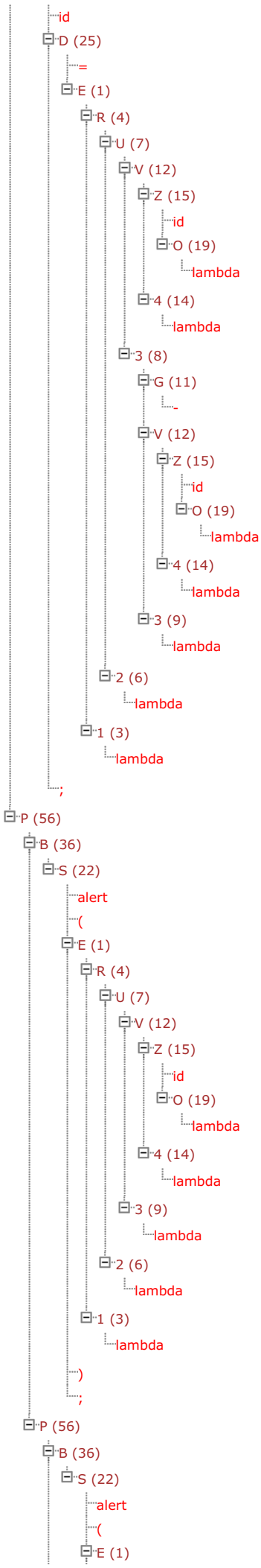
Gramática: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\VisorArbSt\Gramp1.txt

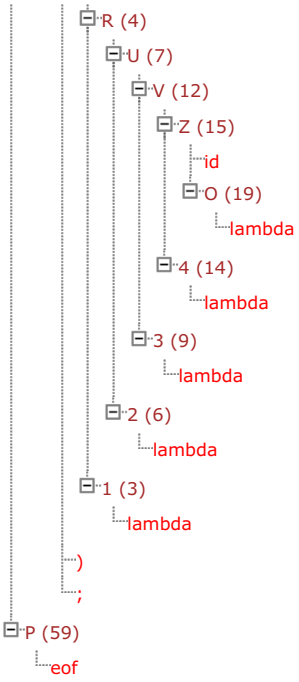
Parse: C:\Users\paula\OneDrive\Escritorio\3º CURSO\Procesadores de Lenguajes\practica\PracticaPDL\parse.txt




4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda


1 (3)
lambda
;
P (56)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (36)
S (21)


id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
P (56)
B (36)
S (22)
alert
(
E (1)


R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
P (59)
eof

Pruebas erróneas

Prueba 1

Código

```
let number n1;
let string cad;
let number n2;
let boolean l2;
input (n1);
l1 = l2; //L1 no declarada
if (l1|| l2) cad = "hello";
n2 = n1 - 378;

alert(      33
            /
            n1
            -
            n2);

function boolean ff(boolean ss)
{
      varglobal = 3;
      if (l1) l2 = ff (ss);
      return ss;
}



if (ff(l1)) alert (varglobal);
```

Errores

```
Errores:
Error 14. Tipos incorrectos en la asignacion. Linea: 6
Error 12. Tipos de expresion incorrectos. Linea: 7
Error 18. Tipos de la expresion incompatibles. Linea: 8
Error 18. Tipos de la expresion incompatibles. Linea: 20
```

Prueba 2

Código

```
let boolean a;
let number b;
let boolean b; //Nombre de la variable ya en uso
a = 3;
b = a;
if (a == b) b = 1;
a = b / b; //tipos incorrectos en la asignacion
alert(b);
```

Errores

```
Errores:
Error 6. Nombre de la variable ya en uso. Linea: 3
```

Prueba 3

Código

```
let string texto;
let boolean a;
function pideTexto ()
{
      alert ("Introduce un texto");
      input (a);//variable input incorrecta
}
function print (string msg)
{
      alert ("Mensaje introducido:");
      alert (msg);
}
pideTxto(); //funcion no declarada
print (texto);
```

Errores

```
Errores:
Error 15. Tipo de variable input incorrecta. Linea: 6
Error 17. Error en los parametros de la llamada a una funcion. Linea:
13
Error 14. Tipos incorrectos en la asignacion. Linea: 13
```

Prueba 4

Código

```
let boolean booleano;
function boolean bisiesto (number a)
{
      return (a - 4 == 0 || a + 100 == 0 || a - 400 == 0);
}
function number dias (number m, number a)
{
      let cadena = “hola”
      let number dd;
      alert ("di cuantos dias tiene el mes ");
      alert (m);
      input(dd);
      if (bisiesto(a)) dd = dd / 1;
      return cadena;
}
function boolean esFechaCorrecta (number d, number m, number a)
{
      return m==1 || m==12 || d==1 || d == dias (m, a);
}
function demo ()
{

      if (esFechaCorrecta(25, 20, “hola”)) alert (9999);
      return;
}
let number a1b2c3d4e5f6g7h8i9j0;
demo();
```

Errores

```
Errores:
Error 10. Sentencia mal escrita. Linea: 8
```

/* A nosotros nos saldria la siguiente información por terminal:
 * Casilla vacia: [T id]. Linea: 8
 */

Prueba 5

Código

```
let boolean booleano;
function boolean bisiesto (number a)
{     let number bis;
      alert ("Es bisiesto?");
      input(bis);
      return ((a - 4 == 0));
}
function number dias (boolean m, number a)
{
      switch (m) //tipo incompatible en el switch

      {
            case 1: case 3: case 5: case 7: case 8: case 10: case 12:
                  return 31; break;
            case 4: case 6: case 9: case 11:
                  return 30;
            case 2: if (bisiesto (a)) return 29;
                  return(28);
            default:return(0);
      }
}
function boolean esFechaCorrecta (number d, number m, number a)
{
      return m==1 || m==12 || d==1 || d == dias (m, a);
}
funcion demo ()  //token no valido
{

      if (esFechaCorrecta(20, 10, 2020)) alert ("ok");
}
let number a1b2c3d4e5f6g7h8i9j0;
demo();
```

Errores

```
Errores:
Error 19. Tipos del Switch incompatibles. Linea: 10
Error 10. Sentencia mal escrita. Linea: 26
```

/* A nosotros nos saldria la siguiente información por terminal:
 * Casilla vacia: [D id]. Linea: 26
 */