Memoria Analizador Sintáctico

Ignacio Gómez Valverde
Paula Labandeira Campos
Óscar Martínez Garcés

Índice

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

Anexo

Caso 1:

Código fuente:

```
-----------------------------
let number a;
let number b;
let boolean bbb;
a = 3;
b = a;
let boolean c;
c = a == b;
if (c) b = 1;
if (b == a) b = 4;
a = a / b;
alert (a);
alert(b);
-------------------------
```

Parse:

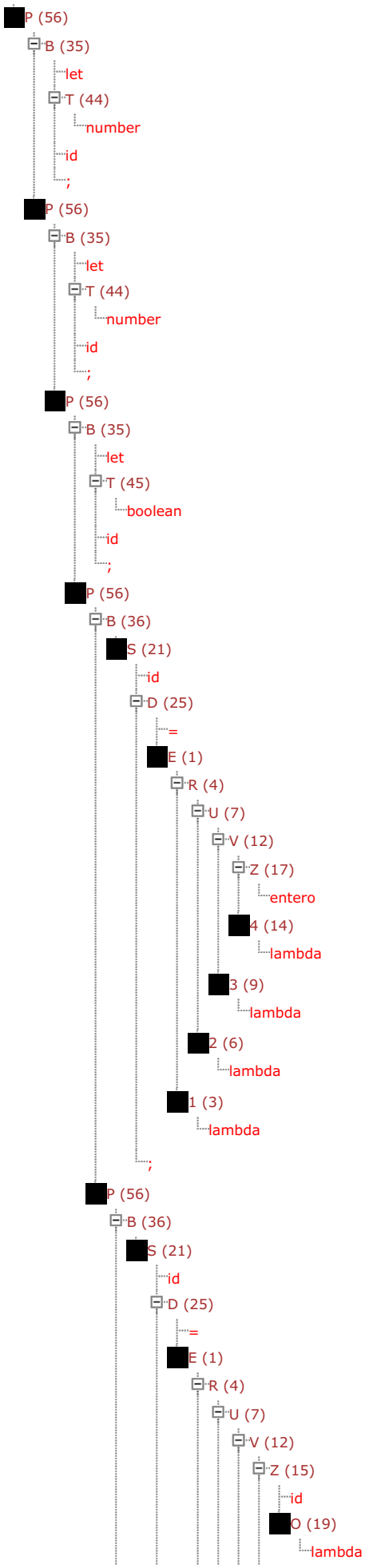
Descendente 56 35 44 56 35 44 56 35 45 56 36 21 25 1 4 7 12 17 14 9 6 3 56 36 21 25 1 4 7 12 15 19 14 9 6 3 56 35 45 56 36 21 25 1 4 7 12 15 19 14 9 5 7 12 15 19 14 9 6 3 56 34 1 4 7 12 15 19 14 9 6 3 21 25 1 4 7 12 17 14 9 6 3 56 34 1 4 7 12 15 19 14 9 5 7 12 15 19 14 9 6 3 21 25 1 4 7 12 17 14 9 6 3 56 36 21 25 1 4 7 12 15 19 13 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

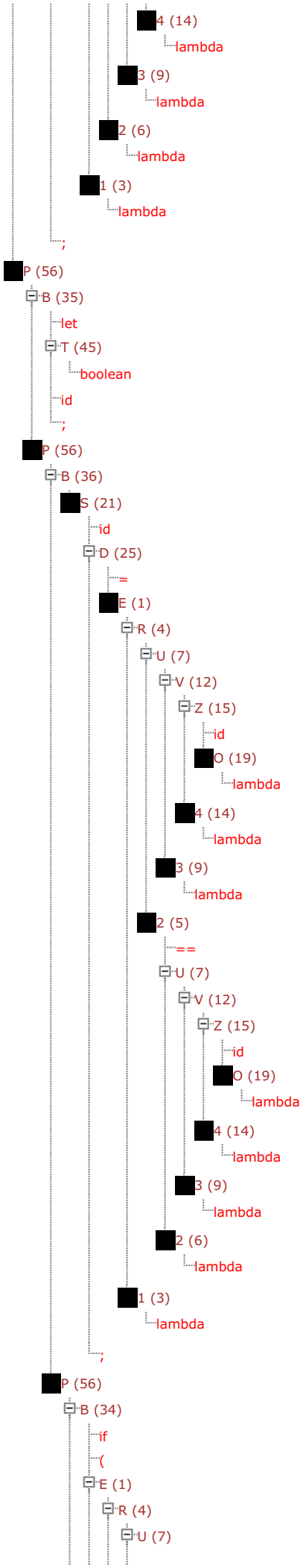
Árbol:

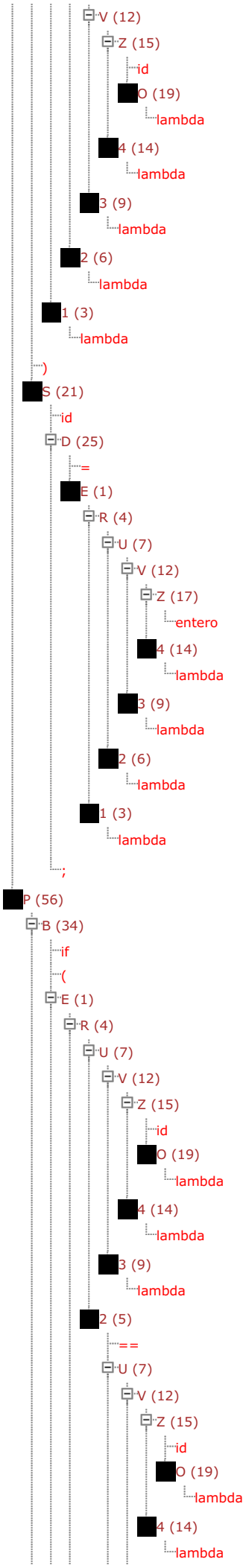
Árbol resultado de:

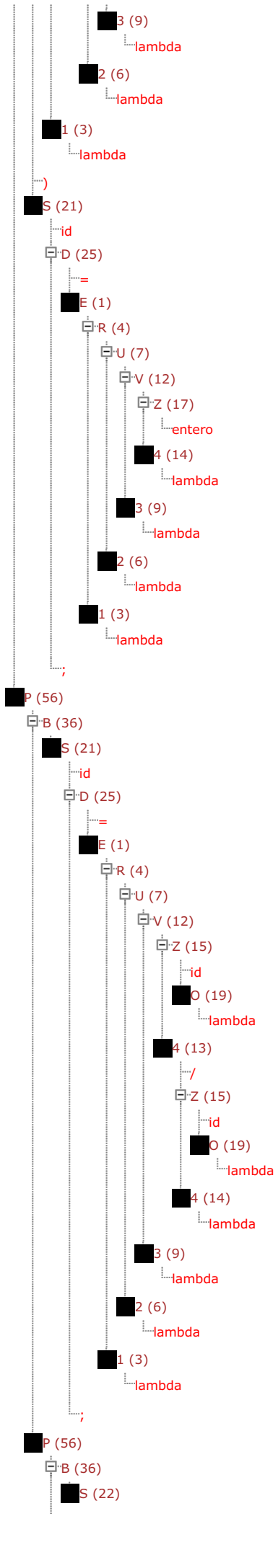
Gramática: C:\Users\inamo\Downloads\VisorArbSt\gramatica.txt

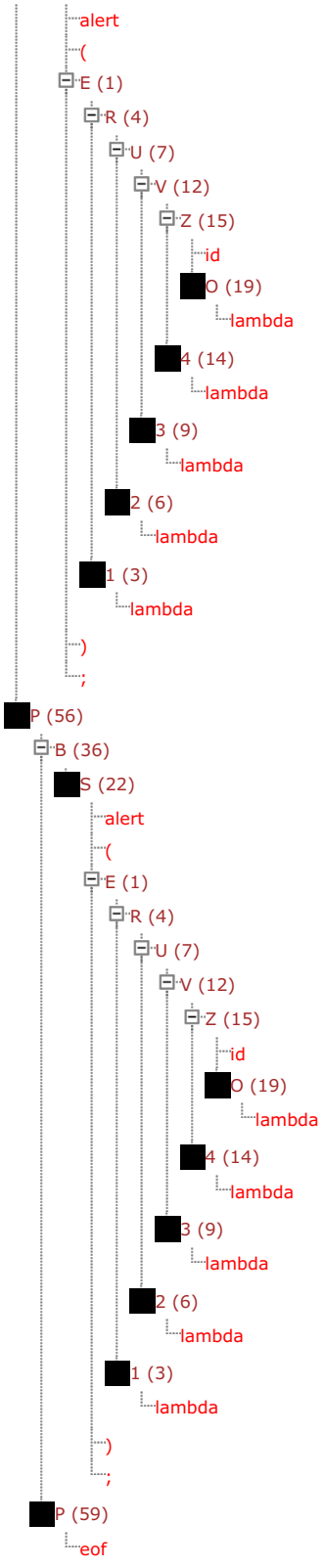
Parse: C:\Users\inamo\Downloads\VisorArbSt\parsep5.txt




4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (35)
let
T (45)
boolean
id
;
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 (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
;
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 (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 (13)
/
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

Caso 2:

Código fuente:

```
        let     boolean         b;let number x;
input (z);
alert (z);
x=z;
alert (z-1);
b=b||b;
alert ("PdL");
input (esto_es_un_nombre_de_variable_global_de_tipo_entero);
if (b) z =
  x - 6
    + z
    - (1
    - 2
    - y)
    / 8;
```

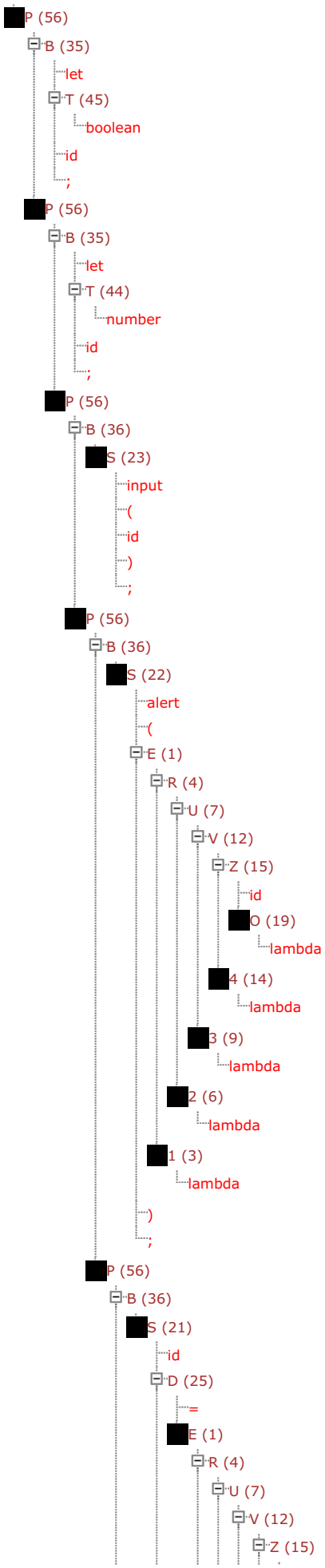
Parse:
Descendente 56 35 45 56 35 44 56 36 23 56 36 22 1 4 7 12 15 19 14 9 6 3 56 36 21 25 1 4 7 12 15 19 14 9 6 3 56 36 22 1 4 7 12 15 19 14 8 11 12 17 14 9 6 3 56 36 21 25 1 4 7 12 15 19 14 9 6 2 4 7 12 15 19 14 9 6 3 56 36 22 1 4 7 12 18 14 9 6 3 56 36 23 56 34 1 4 7 12 15 19 14 9 6 3 21 25 1 4 7 12 15 19 14 8 11 12 17 14 8 10 12 15 19 14 8 11 12 16 1 4 7 12 17 14 8 11 12 17 14 8 11 12 15 19 14 9 6 3 13 17 14 9 6 3 59

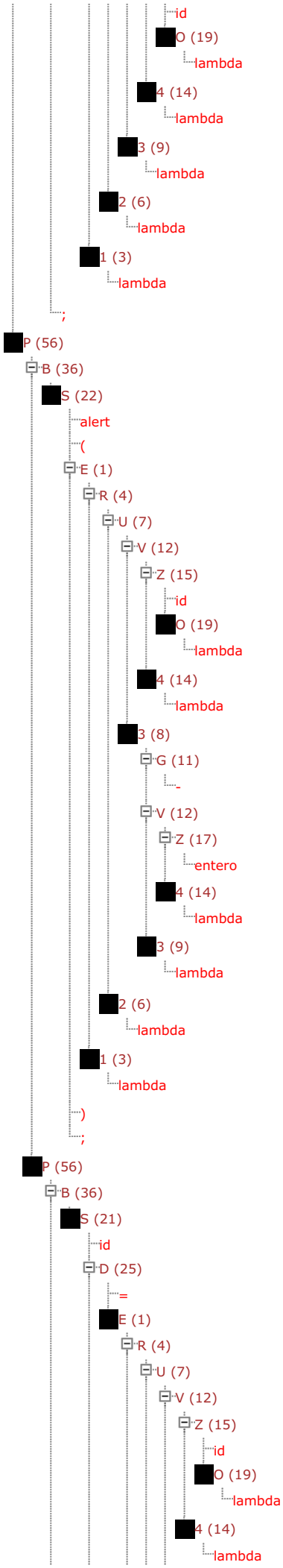
Árbol:

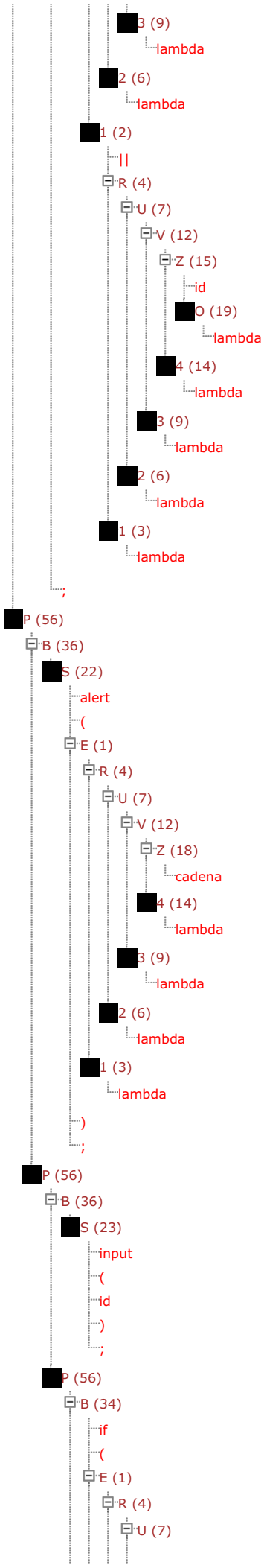
Árbol resultado de:

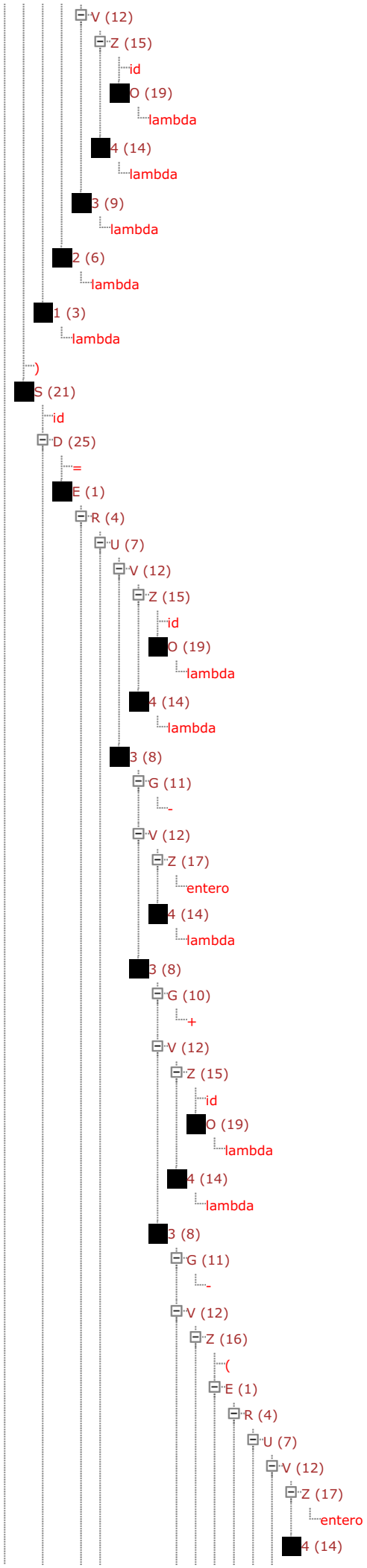
Gramática: C:\Users\inamo\Downloads\VisorArbSt\gramatica.txt

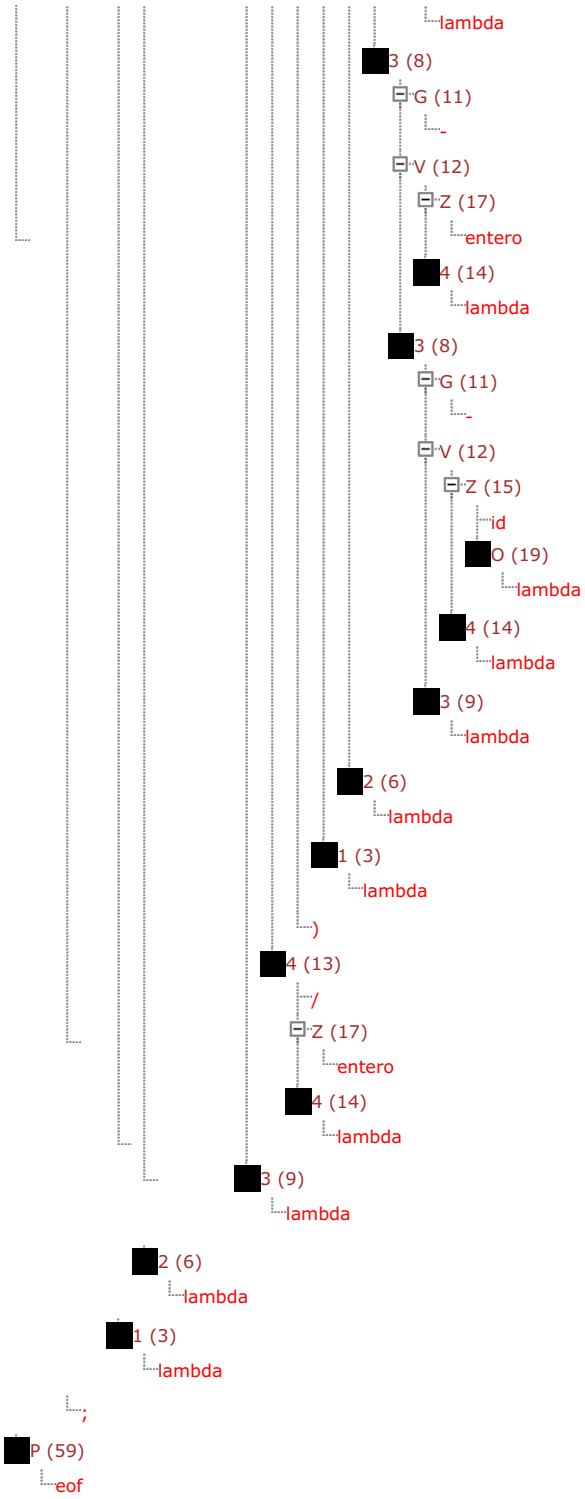
Parse: C:\Users\inamo\Downloads\VisorArbSt\parsep6.txt




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 (8)
G (11)
-
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 (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
;
P (56)
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
)
;
P (56)
B (36)
S (23)
input
(
id
)
;
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 (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 (14)
lambda
3 (8)
G (11)
-
V (12)
Z (17)
entero
4 (14)
lambda
3 (8)
G (10)
+
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (8)
G (11)
-
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 (11)
-
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
4 (13)
/
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (59)
eof

Caso 3:

Código fuente:

```
let number number1;
let string cadena;input(cadena);
let boolean logico1;let boolean logico2;

number1 = 873;

number2 = 378;
if (logico1|| logico2) cadena = "hello";

function string ff(string ss)
{
        global = 33;
        logico1 =  logico2;
        if (logico1) ss = ff (cadena);
        return ss;
}

function string funcion (string logico2)
{       let number var;

  switch (number1){
        case 0:         logico1 = number1 == number2;break;
        case 8888: alert(1234);
        case 3333: logico2="";
  }

  return logico2;
}

alert(cadena);
cadena = ((ff(funcion(cadena))));
```

Parse:

Descendente 56 35 44 56 35 46 56 36 23 56 35 45 56 35 45 56 36 21 25 1 4 7 12 17 14 9 6 3 56 36 21 25 1 4 7 12 17 14 9 6 3 56 34 1 4 7 12 15 19 14 9 6 2 4 7 12 15 19 14 9 6 3 21 25 1 4 7 12 18 14 9 6 3 57 47 48 46 50 46 53 54 36 21 25 1 4 7 12 17 14 9 6 3 54 36 21 25 1 4 7 12 15 19 14 9 6 3 54 34 1 4 7 12 15 19 14 9 6 3 21 25 1 4 7 12 15 20 28 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 54 36 24 32 1 4 7 12 15 19 14 9 6 3 55 57 47 48 46 50 46 53 54 35 44 54 37 1 4 7 12 15 19 14 9 6 3 38 54 36 21 25 1 4

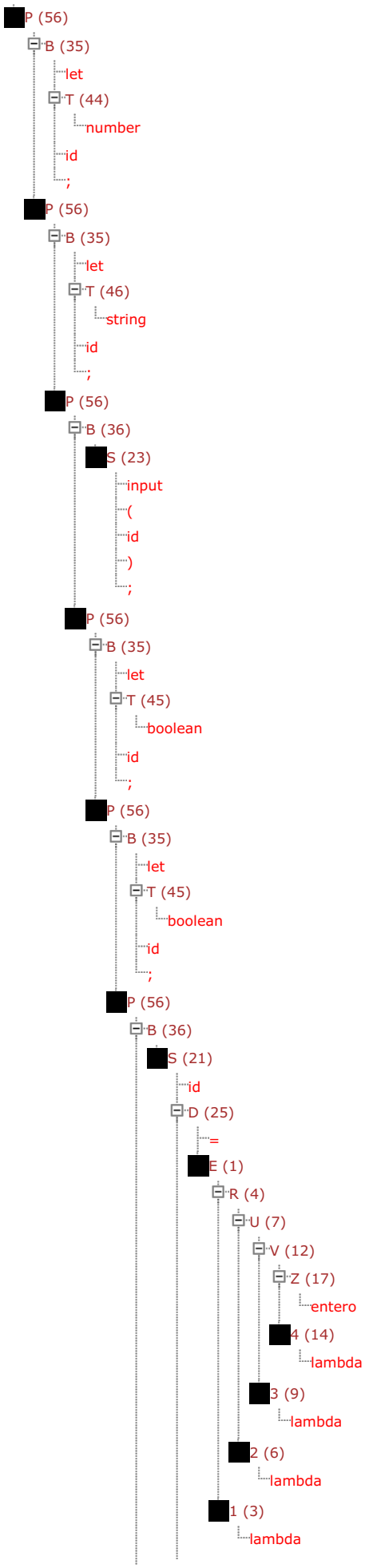
7 12 15 19 14 9 5 7 12 15 19 14 9 6 3 55 39 41 38 54 36 22 1 4 7 12 17 14 9 6 3 55 40 41 38 54 36 21 25 1 4 7 12 18 14 9 6 3 55 40 43 54 36 24 32 1 4 7 12 15 19 14 9 6 3 55 56 36 22 1 4 7 12 15 19 14 9 6 3 56 36 21 25 1 4 7 12 16 1 4 7 12 16 1 4 7 12 15 20 28 1 4 7 12 15 20 28 1 4 7 12 15 19 14 9 6 3 31 14 9 6 3 31 14 9 6 3 14 9 6 3 14 9 6 3 59

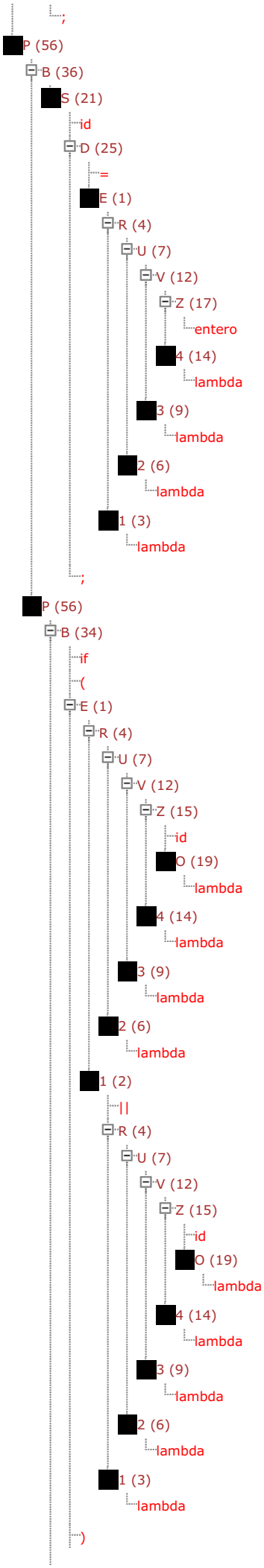
Árbol:

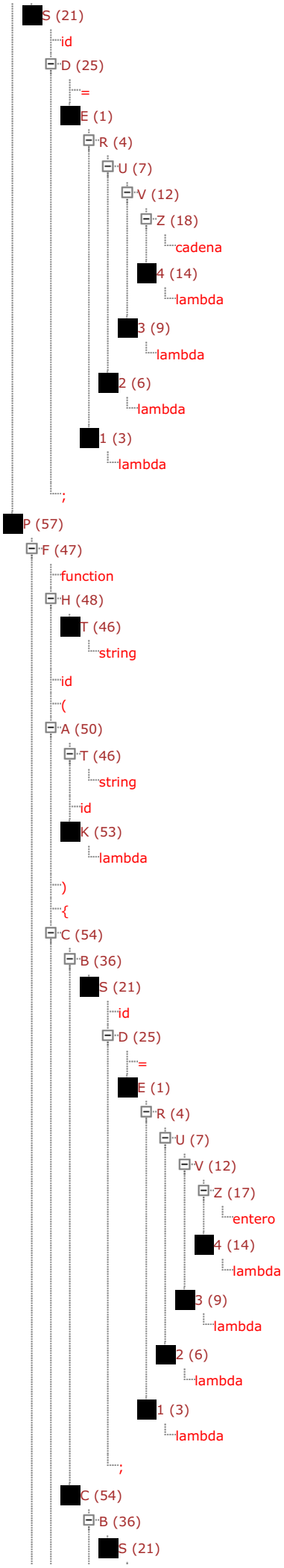
Árbol resultado de:

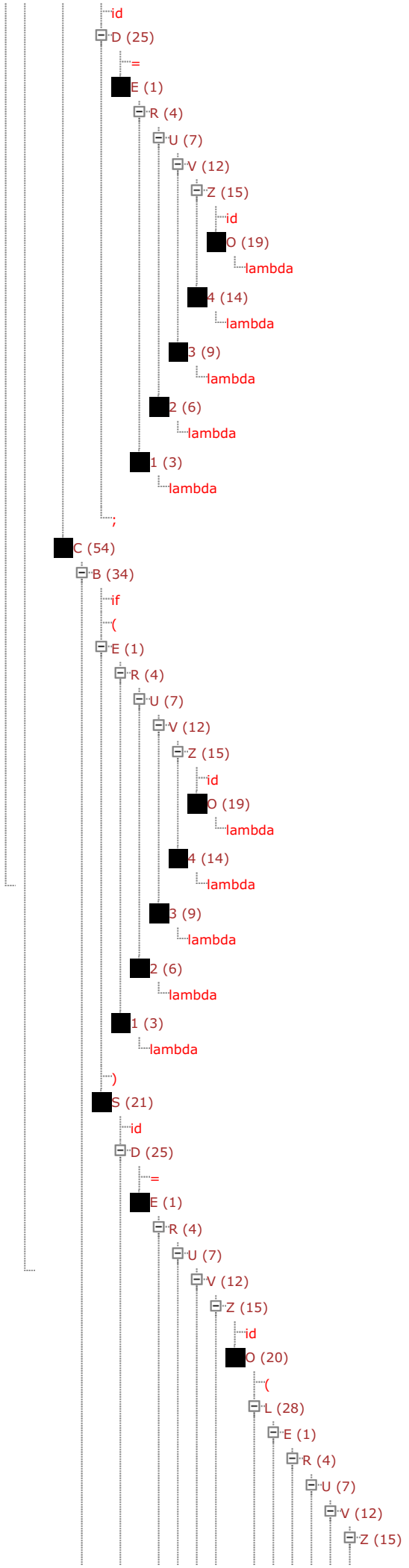
Gramática: C:\Users\inamo\Downloads\VisorArbSt\gramatica.txt

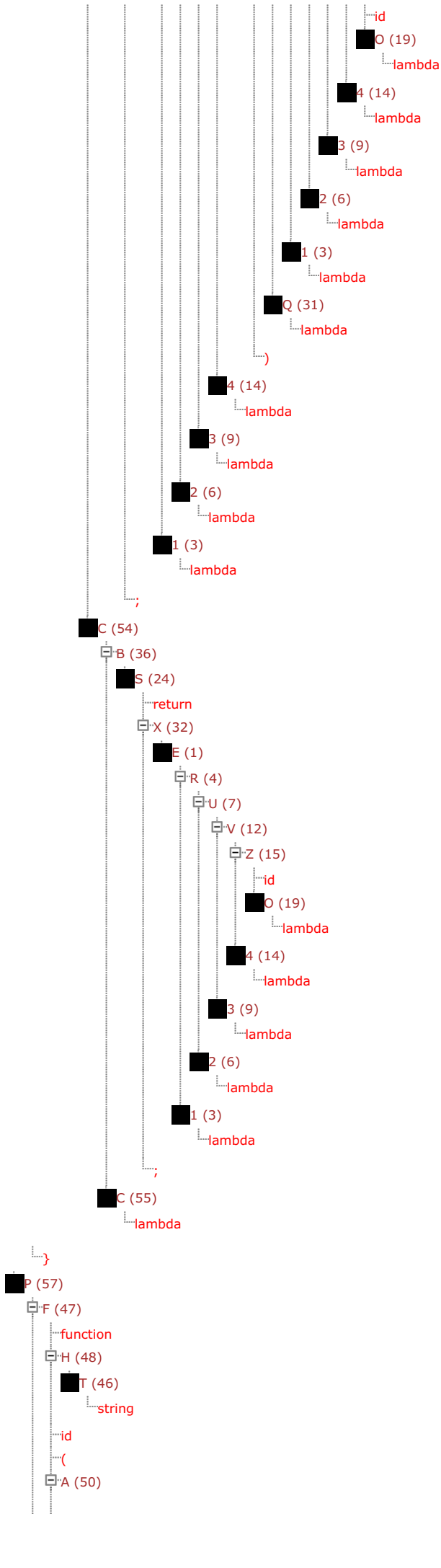
Parse: C:\Users\inamo\Downloads\VisorArbSt\parsep7.txt

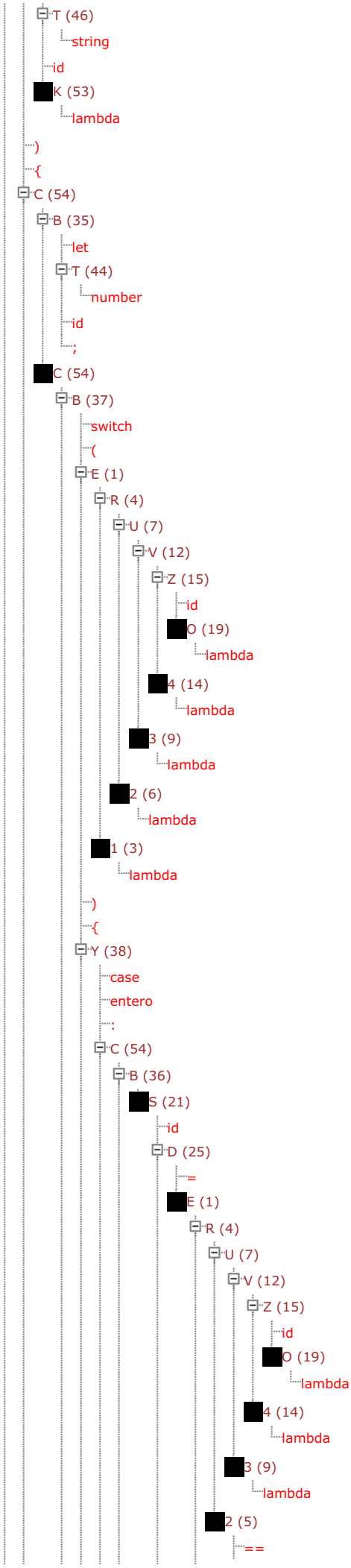


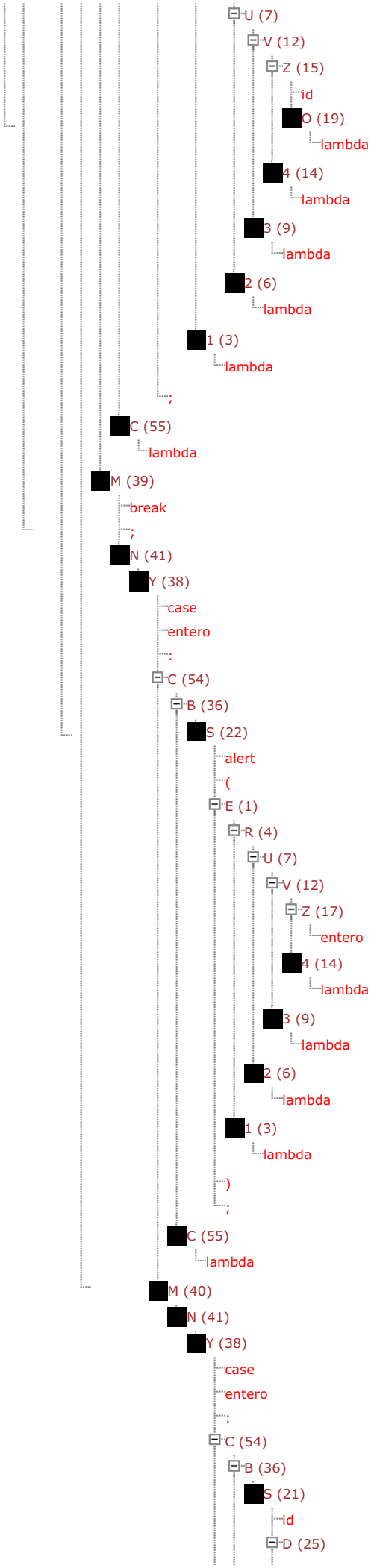

;
P (56)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (56)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (2)
||
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda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

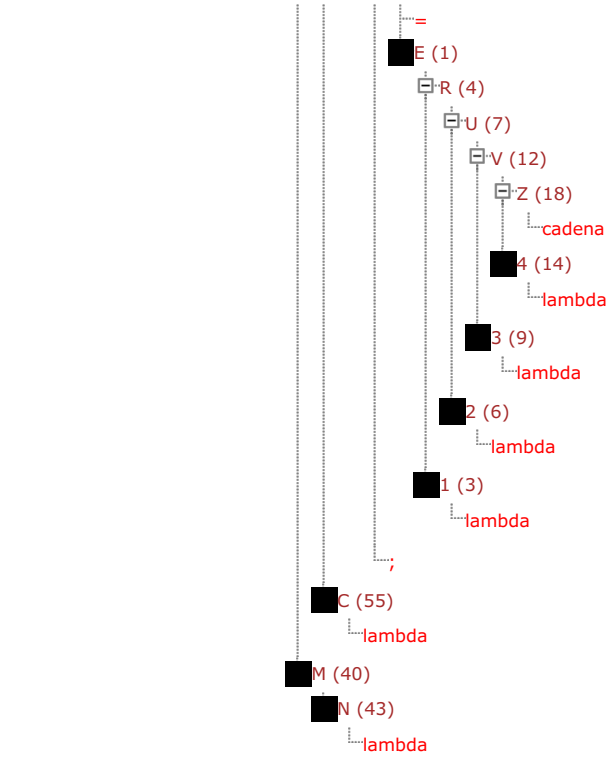

S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (18)
cadena
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (57)
F (47)
function
H (48)
T (46)
string
id
(
A (50)
T (46)
string
id
K (53)
lambda
)
{
C (54)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)
B (36)
S (21)

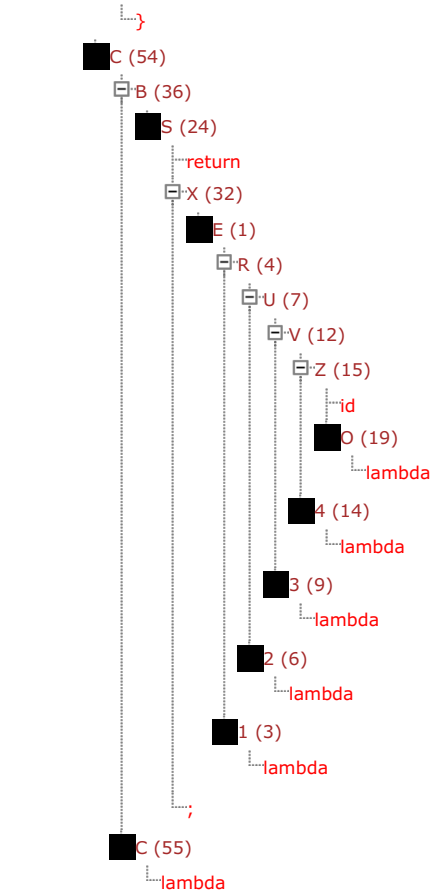

id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)
B (34)
if
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (20)
(
L (28)
E (1)
R (4)
U (7)
V (12)
Z (15)

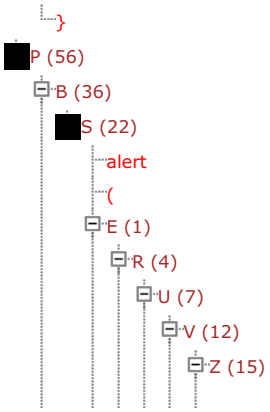

id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
Q (31)
lambda
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (54)
B (36)
S (24)
return
X (32)
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
}
P (57)
F (47)
function
H (48)
T (46)
string
id
(
A (50)

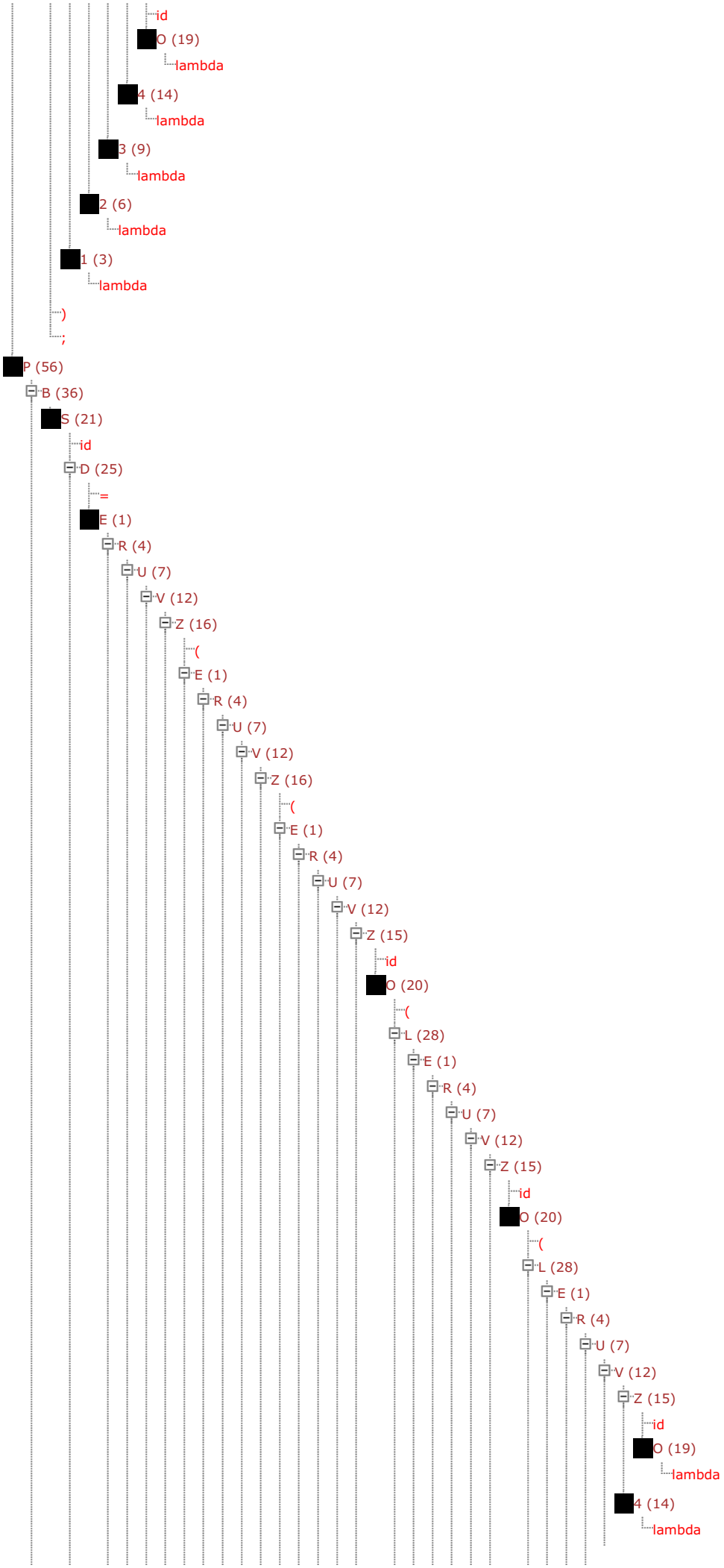

T (46)
string
id
K (53)
lambda
)
{
C (54)
B (35)
let
T (44)
number
id
;
C (54)
B (37)
switch
(
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
{
Y (38)
case
entero
:
C (54)
B (36)
S (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (5)
==

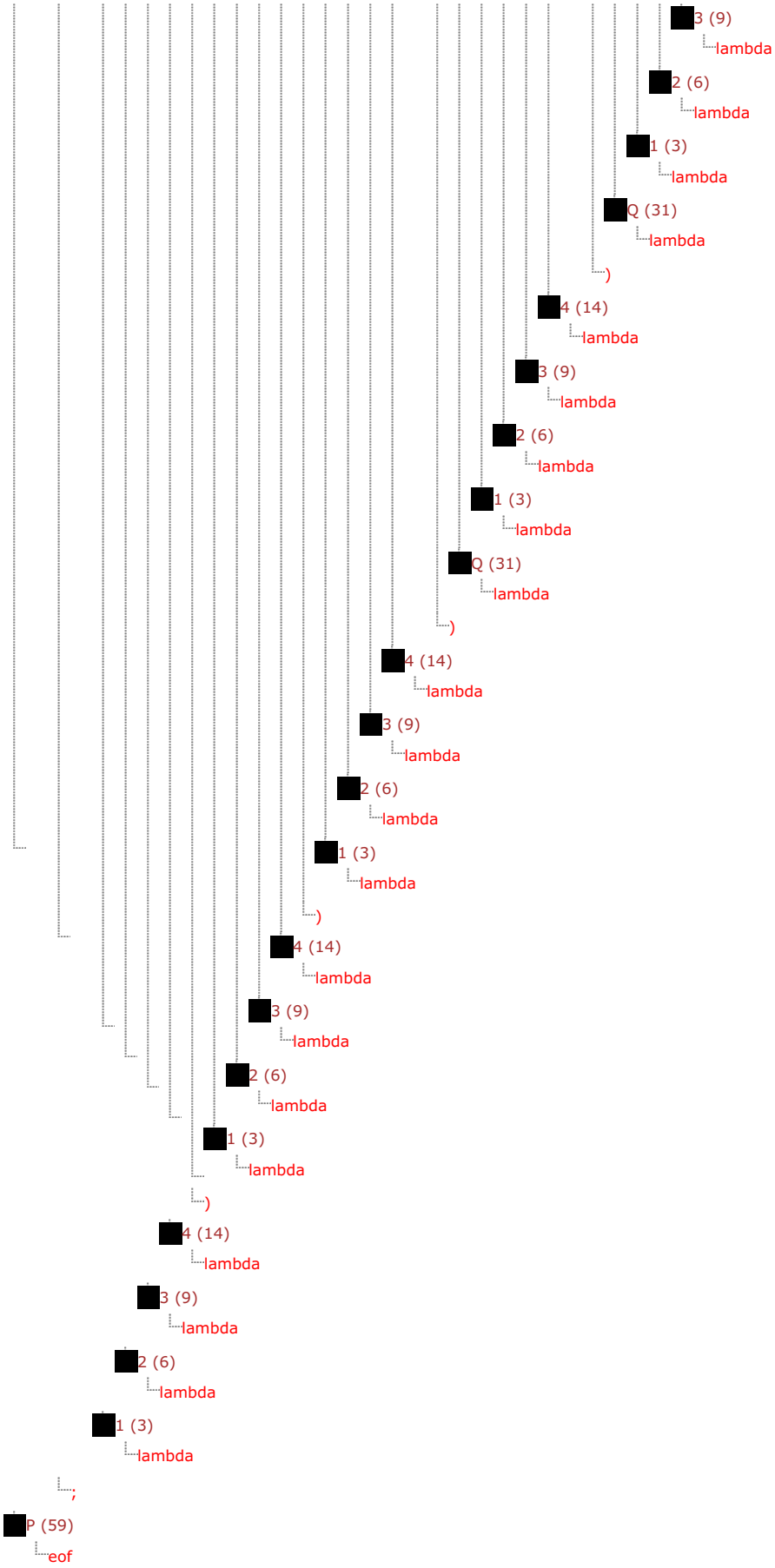

U (7)
V (12)
Z (15)
id
O (19)
lambda
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
C (55)
lambda
M (39)
break
;
N (41)
Y (38)
case
entero
:
C (54)
B (36)
S (22)
alert
(
E (1)
R (4)
U (7)
V (12)
Z (17)
entero
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
)
;
C (55)
lambda
M (40)
N (41)
Y (38)
case
entero
:
C (54)
B (36)
S (21)
id
D (25)


=
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
M (40)
N (43)
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 (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 (21)
id
D (25)
=
E (1)
R (4)
U (7)
V (12)
Z (16)
(
E (1)
R (4)
U (7)
V (12)
Z (16)
(
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 (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
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
)
4 (14)
lambda
3 (9)
lambda
2 (6)
lambda
1 (3)
lambda
;
P (59)
eof

Caso 4:

Código fuente:

```
let number a;
let number b;
let number int;
alert ("Introduce el primer operando")
input (a);
alert ("Introduce el segundo operando");
input (b);
function number operacion (number num1, number num2)
{
        return num1+num2;
}

int = operacion (a, b);
alert (int);
```

Parse:

Descendente 56 35 44 56 35 44 56 35 44 56 36 22 1 4 7 12 18 14 9 6 3

Errores: Falta punto y coma en el alert de la 4 línea.

Caso 5:

Código fuente:

```
let number x;
let number z;
let boolean b;
input (z);
alert (z);
x=z;
alert (z/1; //falta un paréntesis
b=b||b;if (b) z =
  x / 6
    + z
    / (1
    - 2
    - y)
    / 8;
```

Parse: Descendente 56 35 44 56 35 44 56 35 45 56 36 23 56 36 22 1 4 7 12 15 19 14 9 6 3 56 36 21 25 1 4 7 12 15 19 14 9 6 3 56 36 22 1 4 7 12 15 19 13 17 14 9 6 3

Errores:
Falta) en la 8 línea.

Caso 6:

Código fuente:

```
let number a;
let number b;
let number int;
alert ("Introduce el primer operando");
input (a);
alert ("Introduce el segundo operando");
input (b);
function number (number num1, number num2)
{
        return num1-num2;
}

int = 0;
alert (operacion (a, b));
```

Parse:

Descendente 56 35 44 56 35 44 56 35 44 56 36 22 1 4 7 12 18 14 9 6 3 56 36 23 56 36 22 1 4 7 12 18 14 9 6 3 56 36 23 57 47 48 44

Errores: Falta el id de la función al declararla.