Memoria

Grupo 76

Xin Ru Huang

Lucía Jiang

Jaime Jiménez Jiménez

Opciones asignadas al grupo 76:

- Sentencias: sentencia repetitiva (for).
- Operadores especiales: Asignación con resta (-=)
- Comentarios de bloque (/* */)
- Cadenas con comillas dobles (“ ”)
- Técnicas de Análisis Sintáctico: descendente recursivo

Opciones elegidas:

- Operadores aritméticos: suma (+) y resta (-)
- Operadores relacionales: menor (<) y mayor (>)
- Operadores lógicos: or-lógico (||) e y-lógico (&&)

Introducción

Esta memoria resume el diseño del Procesador de Lenguajes realizado por el grupo 76. La práctica consiste en el diseño e implementación de un compilador, que realiza las fases de Análisis Léxico, Sintáctico y Semántico; incluyendo la Tabla de Símbolos y el Gestor de Errores.

Diseño léxico

El Analizador Léxico es el único que tiene acceso al fichero fuente y se encarga de leer carácter por carácter el código de entrada, transformándolo en distintos componentes significativos del lenguaje (tokens) que recibirá el Analizador Sintáctico. Así como rellenar algunos campos de la Tabla de Símbolos y detectar ciertos errores.

1) Tokens

Los tokens se representan con el par <código, atributo>.

Los códigos serán numéricos y solo se rellenará el atributo cuando más de un lexema concuerda con un mismo patrón (id, enteros y cadena).

<1, posTS> //id
<2,-> // =
<3,-> // (
<4,-> //)
<5, -> // {
<6,-> // }
<7,-> // +
<8,-> / /-
<9,-> // -=
<10, -> // <
<11,-> // >
<12,-> // &&
<13,-> // ||
<14,valor> // int
<15, lexema> // cadena
<16,-> // ,
<17,-> // ;

Tokens de palabras reservadas:

<21, -> // int
<22, -> // boolean
<23, -> // if
<24, -> // for
<25, -> // let
<26, -> // string
<27, -> // print
<28, -> // input

2) Gramática regular

Se ha diseñado la siguiente gramática regular que permite implementar los tokens anteriores:

G=(N, T, P, S)

Axioma: S

Símbolos terminales: T={ d + - < > & | = ( ) del “ l }

Símbolos no terminales: N={ A B C D E F G H I J }

Producciones: P={

S → dA | + | -B | < | > | &C | |D | “E | lF | /G | = | (|) | { | } | delS | /G

A → dA | λ

B → = | λ

C → &

D → |

E → oc_7E | ”

F → lF | dF | _F | λ

G → *H

H → oc_5H| *I

I → oc_6H | *I | /S

}

d = dígitos del 0 al 9

del = ‘ ’, ‘\t’, ‘\n’, ‘\r’

l = letras a..z

oc1 = cualquier otro carácter menos d.

oc2 = cualquier otro carácter menos ‘ = ’.

oc4 = cualquier otro carácter menos l, m, _, d.

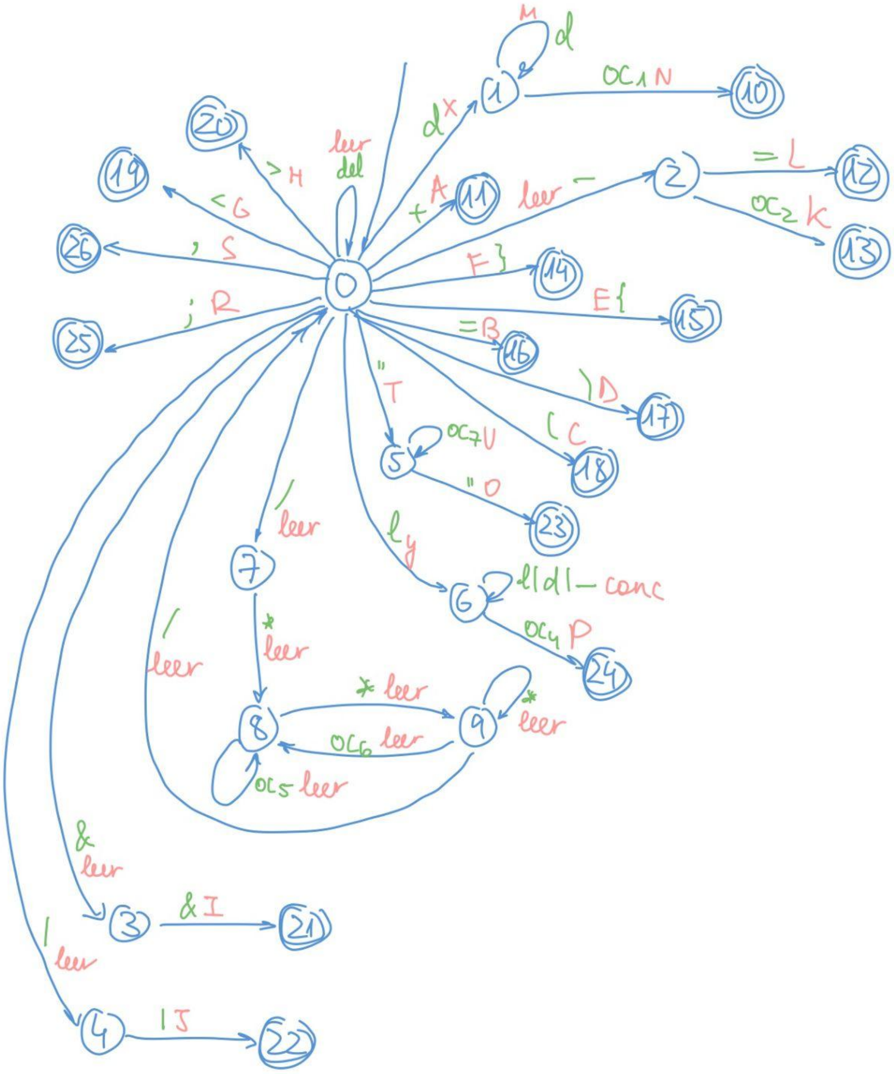
oc5 = cualquier otro carácter menos ‘ * ’ .

oc6 = cualquier otro carácter menos ‘ / ’ , ‘ * ’.

oc7 = cualquier otro carácter menos ‘ ” ’.

3.1) Autómata Finito Determinista

El autómata obtenido tras la gramática es:



Matriz de transiciones

	d		Letra may.		letra		/		*		"		(		)		{		}		=		+		-		<		>		&		|		,		;		_		del	
1	1	X		0	6	Y	7	LEER		0	5	Q	18	C	17	D	15	E	14	F	16	B	11	A	2	LEER	19	G	20	H	3	LEER	4	LEER	26	S	25	R		0	0	LEER
2	1	M	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N	10	N
3	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K	12	L	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K	13	K
4		3		3		3		3		3		3		3		3		3		3		3		3		3		3		3	21	I		3		3		3		3		3
5		4		4		4		4		4		4		4		4		4		4		4		4		4		4		4		4	22	J		4		4		4		4
6	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	23	O	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC	5	CONC
7	6	CONC	6	CONC	6	CONC	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	24	P	6	CONC	24	P
8		7		7		7		7	8	LEER		7		7		7		7		7		7		7		7		7		7		7		7		7		7		7		7
9	8	LEER	8	LEER	8	LEER	8	LEER	9	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER
10	8	LEER	8	LEER	8	LEER	0	LEER	9	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER	8	LEER
11																																									
12																																								
13																																								
14																																								
15																																								
16																																								
17																																								
18																																								
19																																								
20																																								
21																																								
22																																								
23																																								
24																																								
25																																								
26																																								
27																																								

4) Acciones semánticas

Se muestra en el autómata en qué transición se debe de realizar cada acción.

A: Gen_token(7,-) /* + */

B: Gen_token(2,-) /* = */

C: Gen_token(3,-) /* (*/

D: Gen_token(4,-) /*) */

E: Gen_token(5,-) /* { */

F: Gen_token(6,-) /* } */

G: Gen_token(10,-) /* < */

H: Gen_token(11,-) /* > */

I: Gen_token(12,-) /* && */

J: Gen_token(13,-) /* || */

K: Gen_token(8,-) /* - */

L: Gen_token(9,-) /* -= */

N: if (valor>32767) then error(51)
else Gen_token(int, valor)

O: if (contador>64) then error(52)
else Gen_token(cadena, palabra)

R: Gen_token(17,-) /* ; */

S: Gen_token(16,-) /* , */

Q: palabra = “”
LEER

X: valor = digitoLeido
LEER

Y: palabra = caracterLeido
LEER

LEER: leer el siguiente carácter

CONCATENAR: palabra = palabra + caracterLeido
LEER

M: valor = valor*10+digitoLeido
LEER

T: contador = 0

U: CONC
contador++

```
P: p = BuscaTPR(lexema)
        if p!=null then Gen_token(pal_reservada[cód_TPR], )
        else /*es un identificador*/
                p = BuscaTS(lexema)
        if p!=null /*ya se ha declarado previamente*/
                Gen_token(id,p)

        else /*no se ha declarado previamente*/
                if (zona_decl){ /*se está declarando*/
                        p = añadirTS(lexema)
                        Gen_token(id,p)
                }
                else{ /*no se está declarando*/
                        p = añadirTSglobal(lexema)
                        Gen_token(id,p)        }
```

5) Errores

Los errores que detecta el error léxico son:

- No llegue al estado 0 d, ‘ + ’, ‘ - ’, ‘ < ’, ‘ > ’, ‘ & ’, ‘ | ’, ‘ = ’, ‘ (’, ‘) ’, ‘ “ ’, l, del
- No llegue al estado 3 ‘&’.
- No llegue al estado 4 ‘ | ’.
- No llegue al estado 7 ‘ * ’.
- Tipo de errores:
    - Se esperaba un carácter distinto al leído.
    - Carácter que no pertenece al alfabeto.
    - El valor del número supera el rango permitido (16 bits).
    - La cadena supera la longitud máxima (64 caracteres).

Diseño sintáctico

El Análisis Sintáctico se encarga de comprobar si la secuencia de tokens que va recibiendo del Análisis Léxico tiene una sintaxis correcta, y va construyendo el árbol sintáctico utilizando reglas de Gramática de Contexto Libre. A diferencia del Análisis Léxico, en el que los terminales eran los caracteres del fichero de entrada, en el Análisis Sintáctico serán los tokens. El modo de gestión de errores que hemos empleado es de detectar uno y parar.

Cabe destacar que los tokens se reciben a medida que el léxico los obtiene. Una vez procesado el token vuelve a pedir otro token hasta leer el fichero fuente entero.

Gramática G = (T, N, S, P)

Terminales = { let id if ( ) { } for , ; int string boolean entero cadena print input return = < > + - || && λ eof -= function }

NoTerminales = { P B F T S X C L Q H A K E O R U V M N }

Axioma = P

Producciones = { /* todavía no factorizado */

P → B P | F P | eof
B → let T id ; | if (E) S | S | for (M ; E ; N) { C }
T → int | boolean | string
S → id = E ; | return X ; | id (L) ; | print (E) ; | input (id) ;
X → E | λ
C → B C | λ
L → E Q | λ
Q → , E Q | λ
F → function id H (A) { C }
H → T | λ
A → T id K | λ
K → ,T id K | λ
M→ id = E | λ
E → E || O | O ////Como el o-lógico tiene menos precedencia, añadimos el símbolo no terminal O para priorizar y-lógico.
O → O && R | R
R → R > U | R < U | U
U → U + V | U - V | V
V → id | (E) | id (L) | entero | cadena
N → id = E | id -= E | λ

}

El tipo de Analizador Sintáctico asignado a nuestro grupo es el descendente predictivo (sin retroceso). Por ello, necesitamos crear una gramática LL para que dado el token solo se pueda aplicar una única regla. Para ello factorizamos la gramática y también evitamos la recursividad por la izquierda:

Factorización

S → id S1 | return X ; | print (E) ; | input (id) ;

S1 → = E ; | (L) ; | -= E ;

V → id V1 | (E) | entero | cadena

V1→ (L) | λ

N → id N1| λ

N1 → = E | -= E

Eliminación de la recursividad por la izquierda:

Utilizando la fórmula:	Se transforma en:
A → A β	A → αA’
A → α	A’ → βA’
	A’ → λ

De manera que se obtienen las siguientes producciones:

E → O E1

E1 → || O E1 | λ

O → R O1

O1 → && R O1 | λ

R → U R1

R1→ > U R1 | < U R1| λ

U → V U1

U1→ + V U1 | - V U1 | λ

Tablas First y Follow

A continuación, mostramos una tabla en la que se recogen los First y Follow de todos los no terminales, que utilizaremos para comprobar que la gramática utilizada es adecuada para un Análisis Sintáctico descendente predictivo.

Símbolos	First	Follow
P	let if id return print input for function eof	$
B	let if id return print input for	let if id return print input for function eof $ }
F	function	let if id return print input for function eof
T	int boolean string	id
S	id return print input	let if id return print input for function eof $ }
S1	= (-=	let if id return print input for function eof $ }
X	id entero cadena (λ	;
C	let if id return print input for λ	}
L	id entero cadena (λ	)
Q	, λ	)
H	int boolean string λ	(
A	int boolean string λ	)
K	, λ	)
E	id entero cadena (	;)
E1	|| λ	; let if for id input print return function)
O	id, entero, cadena (	||
O1	&& λ	||
R	id entero cadena (	&&
R1	< > λ	&&
U	id entero cadena (	< >
U1	=+ - λ	< >
V	id entero cadena (	=+ -
V1	(λ	+ -
M	id λ	;
N	id	)
N1	= -= λ	)

Comprobación LL(1)

Tras la eliminación de la recursividad por la izquierda, comprobamos si nuestra gramática cumple las condiciones LL(1).

Una gramática es LL(1) si y solo si, para cada par de producciones $A \rightarrow \alpha \mid \beta$ se cumple que:

1. **FIRST(α) ∩ FIRST(β) =** ∅
2. Como mucho, o $\alpha=\lambda$, o $\beta=\lambda$. Suponiendo que $\beta^* \Longrightarrow \lambda$, entonces también comprobamos que **FIRST(α) ∩ FOLLOW(A) =** ∅

A continuación comprobamos las producciones para las par de producciones que cumplan lo anterior:

P → BP | FP | eof

FIRST(BP) ∩ FIRST(FP) = ∅

FIRST(BP) ∩ FIRST(eof) = ∅

FIRST(FP) ∩ FIRST(eof) = ∅

B → let T id ; | if (E) S | S | for (M ; E ; N) { C }

FIRST(let T id ;) ∩ FIRST(if (E) S) = ∅

FIRST(let T id ;) ∩ FIRST(S) = ∅

FIRST(let T id ;) ∩ FIRST(for (M ; E ; N) { C }) = ∅

FIRST(if (E) S) ∩ FIRST(S) = ∅

FIRST(if (E) S) ∩ FIRST(for (M ; E ; N) { C }) = ∅

FIRST(S) ∩ FIRST(for (M ; E ; N) { C }) = ∅

T → int | boolean | string

FIRST(int) ∩ FIRST(boolean) = ∅

FIRST(int) ∩ FIRST(string) = ∅

FIRST(boolean) ∩ FIRST(string) = ∅

S → id S1; | return X ; | print (E) ; | input (id) ;

FIRST(id S1;) ∩ FIRST(return X ;) = ∅

FIRST(id S1;) ∩ FIRST(print (E) ;) = ∅

FIRST(id S1;) ∩ FIRST(input (id) ;) = ∅

FIRST(return X ;) ∩ FIRST(print (E) ;) = ∅

FIRST(return X ;) ∩ FIRST(input (id) ;) = ∅

FIRST(print (E) ;) ∩ FIRST(input (id) ;) = ∅

S1 → = E ; | (L) ; | -= E ;

FIRST(= E ;) ∩ FIRST((L) ;) = ∅

FIRST(= E ;) ∩ FIRST(-= E ;) = ∅

FIRST((L) ;) ∩ FIRST(-= E ;) = ∅

X → E | λ

FIRST(E) ∩ FIRST(λ) = ∅

FIRST(E) ∩ FOLLOW(X) = ∅

C → B C | λ

FIRST(B C) ∩ FIRST(λ) = ∅

FIRST(B C) ∩ FOLLOW(C) = ∅

L → E Q | λ

FIRST(E Q) ∩ FIRST(λ) = ∅

FIRST(E Q) ∩ FOLLOW(L) = ∅

Q → , E Q | λ

FIRST(, E Q) ∩ FIRST(λ) = ∅

FIRST(, E Q) ∩ FOLLOW(Q) = ∅

H → T | λ

FIRST(T) ∩ FIRST(λ) = ∅

FIRST(T) ∩ FOLLOW(H) = ∅

A → T id K | λ

FIRST(T id K) ∩ FIRST(λ) = ∅

FIRST(T id K) ∩ FOLLOW(A) = ∅

K → , T id K | λ

FIRST(, T id K) ∩ FIRST(λ) = ∅

FIRST(, T id K) ∩ FOLLOW(K) = ∅

M → id = E | λ

FIRST(id = E) ∩ FIRST(λ) = ∅

FIRST(id = E) ∩ FOLLOW(M) = ∅

E1 → || O E1 | λ

FIRST(|| O E1) ∩ FIRST(λ) = ∅

FIRST(|| O E1) ∩ FOLLOW(E1) = ∅

O1 → && R O1 | λ

FIRST(&& R O1) ∩ FIRST(λ) = ∅

FIRST(&& R O1) ∩ FOLLOW(O1) = ∅

R1 → < U R1 | > U R1 | λ

FIRST(< U R1) ∩ FIRST(> U R1) = ∅

FIRST(< U R1) ∩ FIRST(λ) = ∅

FIRST(> U R1) ∩ FIRST(λ) = ∅

FIRST(< U R1) ∩ FOLLOW(R1) = ∅

FIRST(> U R1) ∩ FOLLOW(R1) = ∅

U1 → + V U1 | - V U1 | λ

FIRST(+ V U1) ∩ FIRST(- V U1) = ∅

FIRST(+ V U1) ∩ FIRST(λ) = ∅

FIRST(- V U1) ∩ FIRST(λ) = ∅

FIRST(+ V U1) ∩ FOLLOW(U1) = ∅

FIRST(- V U1) ∩ FOLLOW(U1) = ∅

V → id V1 | entero | cadena

FIRST(id V1) ∩ FIRST(entero) = ∅

FIRST(id V1) ∩ FIRST(cadena) = ∅

FIRST(entero) ∩ FIRST(cadena) = ∅

V1 → (L) | λ

FIRST((L)) ∩ FIRST(λ) = ∅

FIRST((L)) ∩ FOLLOW(V1) = ∅

N → id N1 | λ

FIRST(id N1) ∩ FIRST(λ) = ∅

FIRST(id N1) ∩ FOLLOW(N) = ∅

N1 → = E | -= E

FIRST(= E) ∩ FIRST(-= E) = ∅

Como todas las producciones cumplen las condiciones LL(1), concluimos con que la gramática final es apta para un Analizador Sintáctico Descendente recursivo predictivo.

Ampliamos el conjunto de no terminales debido a la introducción de nuevos no terminales para evitar la recursividad por la izquierda.

Por último, hemos añadido la producción P0 → P y asignado como nuevo axioma P0, para el diseño del Análisis Semántico.

Gramática resultante

Gramática G = (T, N, S, P)

Terminales = { let id if ( ) { } for , ; int string boolean entero cadena print input return = < > + - || && λ eof -= function }

NoTerminales = { P0 P B F T S S1 X C L Q H A K E E1 O O1 R R1 U U1 V V1 M N N1 }

Axioma = P0

Producciones = {

1	P0 → P	31	A → λ
2	P → B P	32	K → , T id K
3	P → F P	33	K → λ
4	P → eof	34	M → id = E
5	B → let T id ;	35	M → λ
6	B → if (E) S	36	E → O E1
7	B → S	37	E1 → || O E1
8	B → for (M ; E ; N) { C }	38	E1 → λ
9	T → string	39	O → R O1
10	T → int	40	O1 → && R O1
11	T → boolean	41	O1 → λ
12	S → id S1	42	R → U R1
13	S → print (E) ;	43	R1 → < U R1
14	S → input (id) ;	44	R1 → > U R1
15	S → return X ;	45	R1 → λ
16	S1 → = E ;	46	U → V U1
17	S1 → (L) ;	47	U1 → + V U1
18	S1 → -= E ;	48	U1 → - V U1
19	X → E	49	U1 → λ
20	X → λ	50	V → id V1
21	C → B C	51	V → entero
22	C → λ	52	V → cadena
23	L → E Q	53	V → (E)
24	L → λ	54	V1 → λ
25	Q → , E Q	55	V1 → (L)
26	Q → λ	56	N → id N1
27	F → function id H (A) { C }	57	N → λ
28	H → T	58	N1 → = E
29	H → λ	59	N1 → -= E
30	A → T id K		

}

Ya con la gramática transformada en LL(1) podemos empezar nuestra implementación.

Hemos definido una función para cada símbolo no terminal que tiene la siguiente estructura:

$N \rightarrow \alpha_1 \mid \alpha_2 \mid ...$

N() {

Por cada regla $N \rightarrow \alpha_i$ hacemos lo siguiente:

Añadimos el número al parse para la construcción del árbol sintáctico.

equiparar(t) //equiparamos cada símbolo terminal que aparezca en FIRST(α_i), en el orden en el que aparezca en la regla.

Llamada a las funciones que representan cada símbolo no terminal que pertenezca a α_i

Si existe la regla $N \rightarrow \lambda$, añadimos el número de dicha regla al parse cuando no se puedan ejecutar las otras reglas $N \rightarrow \alpha_i$

}

Las funciones se van llamando de manera recursiva, de manera que si al terminar la ejecución de la primera función llamada, la del axioma; si la cadena se ha leído entera el Analizador Sintáctico termina con éxito. En caso contrario, se ha producido un error sintáctico.

La función equiparar verifica que el token leído es uno de los que se espera el Analizador Sintáctico y pide el siguiente token al Analizador Léxico.

Diseño semántico

El Analizador Semántico obtiene el árbol sintáctico, y se encarga de comprobar que el significado del fichero sea correcto.

A continuación mostramos el diseño de la Traducción Dirigida por la Sintaxis, en concreto, un Esquema de Traducción. En ella se ven reflejados los atributos y las acciones semánticas, cuya principal función es identificar los tipos de los atributos para insertar en la Tabla de Símbolos y de llamar al Gestor de Errores.

Cabe destacar que se han cambiado los nombres de {S1, E1, O1, R1, U1, V1, N1} a {S', E', O', R', U', V', N'} respectivamente. Esto es debido a que estamos acostumbrados a anotar con 1 aquellos no terminales que aparecen también a la izquierda de la producción para poder distinguirlas. Seguiremos este convenio.

Para el diseño semántico hemos utilizado un Esquema de Traducción y hemos tenido en cuenta los siguientes aspectos:

- Las variables que no hayan sido declaradas se considerarán como globales y enteras, como sucede en las reglas 11, 13, 33, etc.
- Las únicas zonas de declaración serán en las reglas 4 y 26 (al ser necesario introducir los parámetros de la función en la TS local.
- Todos los atributos utilizados son sintetizados (la información se pasa de los nodos hijos a los padres). Los atributos que hemos necesitado son: tipo del identificador, tipo de retorno de una función, tipo de los argumentos de una función (que será un array), el ancho de una variable (cadena: 64 palabras; enteros: 1 palabra; lógicos: 1 palabra)

Leyenda

crearTS(): crea una Tabla de Símbolos

destruirTS: destruye la Tabla de Símbolos Actual

TSG: Tabla de Símbolos Global

TSL: Tabla de Símbolos Local

ActTS: Tabla de Símbolos Activa

despG: desplazamiento global

buscaTS: busca si existe id en ActTS

insertaTipoTS: inserta el tipo de un id en la Tabla de Símbolos

insertarDespTS: inserta el desplazamiento en la Tabla de Símbolos

ancho: función que devuelve el ancho de un tipo

buscarTipoTS: busca el tipo de un id

zonaDecl: zona declaración

buscarArgTS: busca argumentos de una función

insertarTS: insertar función en TSG

tipoRetTS: devuelve el tipo que retorna una función

Traducción Dirigida por la Sintaxis

0. **P'** → { TSG:=crearTS(); ActTS:=TSG; despG:=0; despL:=0; zonaDecl:=false }
   P { destruirTS(TSG) }
1. **P → B P1** {}
2. **P → F P1** {}
3. **P → eof** {}
4. **B** → { zonaDecl:=true }
   let T id ; { if (buscaTS(id.pos)) then B.tipo:=tipo_error; Error("Ya existe la variable id")
   else insertaTipoTS(id.pos, T.tipo)
   if (ActTS=TSG) then insertarDespTS(id.pos, despG)
   despG:=despG+T.ancho
   else insertarDespTS(id.pos, despL)
   despL:=despL+T.ancho
   B.tipo:=tipo_ok
   B.tipoRet:=vacío
   zonaDecl:=false }
5. **B → if (E) S** { B.tipo:= if (E.tipo=lóg) then S.tipo
   else tipo_error; Error("La condición debe ser una expresión lógica")
   B.tipoRet:=S.tipoRet }
6. **B → S** { B.tipo:=S.tipo; B.tipoRet:=S.tipoRet }
7. **B → for (M ; E ; N) { C }** { B.tipoRet:=C.tipoRet
   B.tipo:= if (E.tipo!=lóg) then tipo_error; Error("La condición debe ser una expresión lógica.")

else if (M.tipo=tipo_ok and N.tipo!=tipo_error) then C.tipo
else tipo_error; Error("Implementación incorrecta del bucle for.") }

8. **T → string** { T.tipo:=cadena; T.ancho:=64 }
9. **T → int** { T.tipo:=entero; T.ancho:=1 }
10. **T → boolean** { T.tipo:=lóg; T.ancho:=1 }
11. **S → id S'** { if (!buscaTS(id.pos) then
if (S'.tipo=entero) then insertarTipoTSG(id.pos, entero)
insertarDespTSG(id.pos, despG)
despG:=despG+1
S.tipo:=tipo_ok
else S.tipo:=tipo_error; Error("El valor debe ser un entero.")
else if (S'.tipo=tipo_ok and buscarArgTS(id.pos)=S'.tipoArg) then
S.tipo:=tipo_ok //Si es la llamada de una función
else if (S'.tipo=buscarTipoTS(id.pos)) then S.tipo:=tipo_ok
else S.tipo:=tipo_error; Error("La llamada de la función es incorrecta o la asignación es incorrecta.")
S.tipoRet:=S'.tipoRet }
12. **S → print (E) ;** { S.tipo:= if (E.tipo=cadena or E.tipo=entero) then tipo_ok
else tipo_error; Error("Solo se pueden imprimir cadenas o enteros.")
S.tipoRet:=vacío }
13. **S → input (id) ;** { if (!buscarTS(id.pos)) then S.tipo:=tipo_ok
insertarTipoTSG(id.pos, entero)
insertarDespTSG(id.pos, despG)
despG:=despG+1

else if (buscarTipoTS(id.pos)=entero o buscarTipoTS(id.pos)=cadena)
then S.tipo:=tipo_ok
else S.tipo:=tipo_error; Error("Solo se pueden leer enteros o cadenas.")
S.tipoRet:=vacío }
14. **S → return X ;** { S.tipoRet:=X.tipo
S.tipo:= if (X.tipo=tipo_error) then tipo_error; Error("La expresión es errónea.")
else tipo_ok }

15. **S'** → **= E ;** { S'.tipo:=E.tipo; S'.tipoRet:=vacío; S'.tipoArg:=vacío }

16. **S'** → **(L) ;** { S'.tipo:=tipo_ok; S'.tipoRet:=vacío; S'.tipoArg:=L.tipo }
17. **S'** → **-= E ;** { S'.tipo:= if (E.tipo=entero) then E.tipo
    else tipo_error; Error("Solo se pueden restar números enteros.")
    S'.tipoRet:=vacío; S'.tipoArg:=vacío}

18. **X** → **E** { X.tipo:=E.tipo }
19. **X** → **λ** { X.tipo:=vacío }

20. **C** → **B C1** { C.tipo:= if (B.tipo=tipo_ok) then C1.tipo
    else tipo_error;
    C.tipoRet:= if (C1.tipoRet=vacío or C1.tipoRet=B.tipoRet) then B.tipoRet
    else if (B.tipoRet=vacío) then C1.tipoRet
    else tipo_error; Error("Error en los valores devueltos por la función.") }
21. **C** → **λ** { C.tipo:=tipo_ok; C.tipoRet:=vacío }

22. **L** → **E Q** { L.tipo:=E.tipo x Q.tipo }
23. **L** → **λ** { L.tipo:=vacío }

24. **Q** → **, E Q1** { Q.tipo:= E.tipo x Q1.tipo }
25. **Q** → **λ** { Q.tipo:=vacío }

26. **F** → **function id** { if (buscaTS(id.pos)) then F.tipo:=tipo_error
    else
    if (ActTS!=TSG) then Error("No se puede crear una función dentro de otra.")
    insertarTS(id.pos, nueva_et()

    TSL:=crearTS()
    ActTS:=TSL
    despL:=0
    F.tipo:=vacío //Para poder hacer las siguientes comprobaciones }
    H ({ zonaDecl:= true }
    A { zonaDecl:=false }
) { if (F.tipo!=tipo_error) then insertarTS(id.pos, A.tipo → H.tipo }

{ C } { if (F.tipo!=tipo_error)
then if (C.tipo=tipo_error) then Error("Sentencias incorrectas");
if (C.tipoRet!=H.tipo) then Error("El tipo del valor de retorno es incorrecto.")
destruirTS(ActTS)
ActTS:=TSG }

27. **H → T** { H.tipo:=T.tipo }
28. **H → λ** { H.tipo:=vacío }

29. **A → T id K** { if(!buscarTS(id.pos)) then
insertarTipoTS(id.pos, T.tipo)
insertarDespTS(id.pos, despL) //como son parámetros de la función, local se insertará siempre en la TS local
despL:=despL+T.ancho
A.tipo:=T.tipo x K.tipo
else
A.tipo:=tipo_error; Error("Los argumentos de la función están mal definidos.")}
30. **A → λ** { A.tipo:=vacío }
31. **K → , T id K1** { if(!buscarTS(T.tipo)) then
insertarTipoTS(id.pos, T.tipo)
insertarDespTS(id.pos, despL) //como son parámetros de la función, local se insertarán siempre en la TS local
despL:=despL+T.ancho
K.tipo:=T.tipo x K1.tipo
else
K.tipo:=tipo_error; Error("Los argumentos de la función están mal definidos.") }
32. **K → λ** { K.tipo:=vacío }

33. **M → id = E** { if (!buscaTS(id.pos)) then
if (E.tipo=entero) then insertarTipoTSG(id.pos, entero)
insertarDespTSG(id.pos, despG)
despG:=despG+1
M.tipo:=tipo_ok
else M.tipo:=tipo_error; Error("El valor asignado debe ser un entero.")
else if (E.tipo=buscarTipoTS(id.pos)) then M.tipo:=tipo_ok

else M.tipo:=tipo_error; Error("El tipo del valor asignado no coincide con el tipo de la variable.") }

34. **M → λ** { M.tipo:=tipo_ok }

35. **E → O E'** { E.tipo:= if (E'.tipo=vacío) then O.tipo;
else if (O.tipo=lóg and E'.tipo=lóg) then lóg;
else tipo_error; Error("Los tipos no concuerdan.") }

36. **E' → || O E1'** { E'.tipo:= if ((E1'.tipo=vacío or E1'.tipo=lóg) and O.tipo=lóg)
then lóg;
else tipo_error; Error("Los tipos no concuerdan.")}

37. **E' → λ** { E1.tipo:=vacío }

38. **O → R O'** { O.tipo:= if (O'.tipo=vacío) then R.tipo;
else if (R.tipo=lóg and O'.tipo=lóg) then lóg;
else O.tipo:=tipo_error; Error("Los tipos no concuerdan.") }

39. **O' → && R O1'** { O1.tipo:= if ((O1'.tipo=vacío or O1'.tipo=lóg) and R.tipo=lóg)
then lóg;
else tipo_error; Error("Los tipos no concuerdan.")}

40. **O' → λ** { O1.tipo:=vacío }

41. **R → U R'** { R.tipo:= if (R'.tipo=vacío) then U.tipo;
else if (U.tipo=entero and R'.tipo=log) then lóg;
else tipo_error; Error("Los tipos no concuerdan.") }

42. **R' → < U R1'** { R'.tipo:= if (R1'.tipo=vacío and U.tipo=entero)
then lóg;
else tipo_error; Error("Los tipos no concuerdan.")}

43. **R' → > U R1'** { R1.tipo:= if (R1'.tipo=vacío and U.tipo=entero)
then lóg;
else tipo_error; Error("Los tipos no concuerdan.")}

44. **R' → λ** { R'.tipo:=vacío; }

45. **U → V U'** { U.tipo:= if (U'.tipo=vacío) then V.tipo;
else if (V.tipo=entero and U'.tipo=entero) then entero;
else tipo_error; Error("Los tipos no concuerdan.") }

46. **U' → + V U1'** { U'.tipo:= if ((U1'.tipo=vacío or U1'.tipo=entero) and V.tipo=entero)
then entero;
else tipo_error; Error("Los tipos no concuerdan.")}

47. **U' → - V U1'** { U'.tipo:= if ((U1'.tipo=vacío or U1'.tipo=entero) and V.tipo=entero)
then entero;
else tipo_error; Error("Los tipos no concuerdan.")}

48. **U' → λ** { U'.tipo:=vacío; }

49. **V → id V'** { if (V'.tipo=vacío)
then if (!buscaTS(id.pos)) then insertarTipoTS(id.pos, entero)
insertarDespTS(id.pos, despG)
despG:=despG+1
V.tipo:=entero
else V.tipo:=buscarTipoTS(id.pos)
else if (V'.tipo=buscarArgTS(id.pos)) then V.tipo:=tipoRetTS(id.pos)
else V.tipo:=tipo_error; Error("La llamada de la función es incorrecta.") }

50. **V → entero** { V.tipo:=entero }
51. **V → cadena** { V.tipo:=cadena }
52. **V → (E)** { V.tipo:=E.tipo }
53. **V' → λ** { V'.tipo:=vacío }
54. **V' → (L)** { V'.tipo:=L.tipo }

55. **N → id N'** { N.tipo:= if (!buscarTS(id.pos)) then tipo_error; Error("El contador debe estar definido anteriormente")
else
if (buscarTipoTS(id.pos)=N'.tipo) then tipo_ok
else tipo_error; Error("La actualización del contador es incorrecta.")
56. **N → λ** { N.tipo:=tipo_ok }
57. **N1 → = E** { N1.tipo:=E.tipo }

58. **N1 → -= E** { N1.tipo:= if (E.tipo=entero) then entero
else tipo_error; Error("Solo se pueden restar enteros.")}

Tratamiento de errores

El analizador semántico aparte de introducir los atributos en la tabla de símbolos también se encarga de detectar los errores semánticos. Como se puede apreciar en el esquema que aparece previamente, las acciones semánticas son las responsables de comprobar si es correcto el código fuente, y en caso contrario lanzan el respectivo error con una descripción y la línea en la que sucedió.

El modo de gestionar los errores es detectar un error y parar.

Diseño de la tabla de símbolos

La tabla de símbolos (TS) va a tener un tamaño dinámico y un método de organización HASH para las posiciones de los identificadores en la tabla. Hemos elegido este método de organización ya que es el que nos resulta más práctico para definir las posiciones.

El formato de nuestra tabla de símbolos va a tener el siguiente formato:

- Cada tabla irá encabezada por la línea:

  TABLA FUNCIÓN ***nombre_de_la_función #i*** (Para las funciones)

  TABLA PRINCIPAL *#1* (TS global)

  Donde *i* representa el orden en el que se creó la tabla, por tanto, para la TS global siempre será 1.

- Cada tabla contendrá los lexemas de los identificadores (variables) introducidos y sus correspondientes atributos, de la forma:

  *** Lexema: '*identificador*'**

  Atributos:
  +tipo: '*tipo*'
  +despl: '*desplazamiento*'

- Para las funciones el formato será similar al de identificadores que sean variables pero incluyendo información sobre los argumentos, el tipo que retorna y su etiqueta. En este caso no hay información del desplazamiento ya que no tienen.

  *** Lexema: '*identificador*'**

  Atributos:
  +tipo: '*funcion*'
  +numParam: *número_parámetros*
  +TipoParam1: '*tipo_parámetro_1*'
  +ModoParam1: 1 //Los parámetros se pasan siempre por valor.
  (Repetir el formato con los n parámetros de la función)
  ……
  +TipoRetorno: '*tipo_retorno*'
  +EtiqFunction: '*etiqueta_nueva*'

Anexo

Casos de éxito

Caso de éxito 1

Fichero Fuente:

```
let int a;
let   int  b;
let boolean bbb;
a = 3;
b=a;
if (a < b) b -= 1;
a = a - b;
print (a) ;
print(b);
```

Volcado de tokens

< 25, > //token de let
< 21, > //token de declarando int
< 1, 97 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 21, > //token de declarando int
< 1, 98 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 97314 > //token de identificador
< 17, > //token de punto y coma
< 1, 97 > //token de identificador
< 2, > //token de asignacion
< 14, 3 > //token de entero
< 17, > //token de punto y coma
< 1, 98 > //token de identificador
< 2, > //token de asignacion
< 1, 97 > //token de identificador
< 17, > //token de punto y coma
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 97 > //token de identificador
< 10, > //token de menor que
< 1, 98 > //token de identificador
< 4, > //token de cerrar parentesis
< 1, 98 > //token de identificador
< 9, > //token de asignacion resta
< 14, 1 > //token de entero
< 17, > //token de punto y coma
< 1, 97 > //token de identificador
< 2, > //token de asignacion
< 1, 97 > //token de identificador
< 8, > //token de resta
< 1, 98 > //token de identificador
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 98 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma

Volcado de la tabla de símbolos

TABLA FUNCION ff # 2:

* Lexema : 'ss'

Atributos:

+tipo: 'log'

+despl: 0

TABLA PRINCIPAL # 1:

* Lexema : 'ff'
Atributos:
+tipo: 'funcion'
+numParam: 1
+TipoParam1: 'log'
+ModoParam1: 1
+TipoRetorno: 'log'
+EtiqFuncion: 'et_2'
* Lexema : 'n1'
Atributos:
+tipo: 'ent'
+despl: 0
* Lexema : 'n2'
Atributos:
+tipo: 'ent'
+despl: 66
* Lexema : 'l1'
Atributos:
+tipo: 'log'
+despl: 1
* Lexema : 'l2'
Atributos:
+tipo: 'log'
+despl: 67
* Lexema : 'varglobal'
Atributos:
+tipo: 'ent'
+despl: 68
* Lexema : 'cad'
Atributos:
+tipo: 'cadena'
+despl: 2

Parse:

D 1 2 5 10 2 5 10 2 5 11 2 7 12 16 36 39 42 46 51 49 45 41 38 2 7 12 16 36 39 42 46 50 54 49 45 41 38 2 6 36 39 42 46 50 54 49 43 46 50 54 49 45 41 38 12 18 36 39 42 46 51 49 45 41 38 2 7 12 16 36 39 42 46 50 54 48 50 54 49 45 41 38 2 7 13 36 39 42 46 50 54 49 45 41 38 2 7 13 36 39 42 46 50 54 49 45 41 38 4

Árbol sintáctico al final de todos los casos.

Caso de éxito 2

Fichero fuente:

```
let int    n1;
let boolean l1;
let string  cad;
let int    n2;
let boolean l2;
input (n1);
l1 = l2;
if (l1&& l2) cad = "hello";
n2 -= n1 - 378;
print( 33 + n1 + n2);
function ff boolean(boolean ss) {
        varglobal = 8;
        if (l1) l2 = ff (ss);
        return ss;
}

if (ff(l2))  print (varglobal);
```

Volcado de tokens:

```
< 25, > //token de let
< 21, > //token de declarando int
< 1, 3459 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 3397 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 26, > //token de string
< 1, 98246 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 21, > //token de declarando int
< 1, 3460 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 3398 > //token de identificador
< 17, > //token de punto y coma
< 28, > //token de input
< 3, > //token de abrir parentesis
< 1, 3459 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 1, 3397 > //token de identificador
< 2, > //token de asignacion
< 1, 3398 > //token de identificador
< 17, > //token de punto y coma
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 3397 > //token de identificador
< 12, > //token de y logico
< 1, 3398 > //token de identificador
< 4, > //token de cerrar parentesis
< 1, 98246 > //token de identificador
< 2, > //token de asignacion
< 15, "hello" > //token de cadena
< 17, > //token de punto y coma
< 1, 3460 > //token de identificador
< 9, > //token de asignacion resta
< 1, 3459 > //token de identificador
< 8, > //token de resta
< 14, 378 > //token de entero
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 14, 33 > //token de entero
< 7, > //token de suma
< 1, 3459 > //token de identificador
```

< 7, > //token de suma
< 1, 3460 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 30, > //token de function
< 1, 3264 > //token de identificador
< 22, > //token de declarando boolean
< 3, > //token de abrir parentesis
< 22, > //token de declarando boolean
< 1, 3680 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 1, -129199734 > //token de identificador
< 2, > //token de asignacion
< 14, 8 > //token de entero
< 17, > //token de punto y coma
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 3397 > //token de identificador
< 4, > //token de cerrar parentesis
< 1, 3398 > //token de identificador
< 2, > //token de asignacion
< 1, 3264 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 3680 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 29, > //token de return
< 1, 3680 > //token de identificador
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 3264 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 3398 > //token de identificador
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, -129199734 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma

Volcado de TS:

TABLA FUNCION ff # 2:

* Lexema : 'ss'
    Atributos:
        +tipo: 'log'
        +despl: 0

TABLA PRINCIPAL # 1:

* Lexema : 'ff'
    Atributos:
        +tipo: 'funcion'
            +numParam: 1
            +TipoParam1: 'log'
            +ModoParam1: 1
            +TipoRetorno: 'log'
        +EtiqFuncion: 'et_2'

* Lexema : 'n1'
    Atributos:
        +tipo: 'ent'
        +despl: 0
* Lexema : 'n2'
    Atributos:
        +tipo: 'ent'
        +despl: 66
* Lexema : 'l1'
    Atributos:
        +tipo: 'log'
        +despl: 1
* Lexema : 'l2'
    Atributos:
        +tipo: 'log'
        +despl: 67
* Lexema : 'varglobal'
    Atributos:
        +tipo: 'ent'
        +despl: 68
* Lexema : 'cad'
    Atributos:
        +tipo: 'cadena'
        +despl: 2

Parse

D 1 2 5 10 2 5 11 2 5 9 2 5 10 2 5 11 2 7 14 2 7 12 16 36 39 42 46 50 54 49 45 41 38 2 6 36 39 42 46 50 54 49 45 40 42 46 50 54 49 45 41 38 12 16 36 39 42 46 52 49 45 41 38 2 7 12 18 36 39 42 46 50 54 48 51 49 45 41 38 2 7 13 36 39 42 46 51 47 50 54 47 50 54 49 45 41 38 3 27 28 11 30 11 33 21 7 12 16 36 39 42 46 51 49 45 41 38 21 6 36 39 42 46 50 54 49 45 41 38 12 16 36 39 42 46 50 55 23 36 39 42 46 50 54 49 45 41 38 26 49 45 41 38 21 7 15 19 36 39 42 46 50 54 49 45 41 38 22 2 6 36 39 42 46 50 55 23 36 39 42 46 50 54 49 45 41 38 26 49 45 41 38 13 36 39 42 46 50 54 49 45 41 38 4

Árbol sintáctico al final de todos los casos.

Caso de éxito 3

```
let string texto;
function alert (string msg) {
        print ("Mensaje introducido:");
        print (msg);
}
function pideTexto () {
        print ("Introduce un texto corto");
        input (texto);
}
pideTexto();
alert       (texto);
```

Volcado de tokens:

< 25, > //token de let
< 26, > //token de string
< 1, 110256354 > //token de identificador
< 17, > //token de punto y coma
< 30, > //token de function
< 1, 92899676 > //token de identificador
< 3, > //token de abrir parentesis
< 26, > //token de string
< 1, 108417 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 27, > //token de print
< 3, > //token de abrir parentesis
< 15, "Mensaje introducido:" > //token de cadena
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 108417 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 30, > //token de function
< 1, -731642680 > //token de identificador
< 3, > //token de abrir parentesis
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 27, > //token de print
< 3, > //token de abrir parentesis
< 15, "Introduce un texto corto" > //token de cadena
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 28, > //token de input
< 3, > //token de abrir parentesis
< 1, 110256354 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 1, -731642680 > //token de identificador
< 3, > //token de abrir parentesis
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 1, 92899676 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 110256354 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma

Volcado de la TS:
TABLA PRINCIPAL # 1:

* Lexema : 'texto'
    Atributos:
    +tipo: 'null'

* Lexema : 'msg'
    Atributos:
    +tipo: 'null'

* Lexema : 'alert'
    Atributos:
    +tipo: 'null'

* Lexema : 'pideTexto'
    Atributos:
    +tipo: 'null'

Parse:

D 1 2 5 9 3 27 29 30 9 33 21 7 13 36 39 42 46 52 49 45 41 38 21 7 13 36 39 42 46 50 54 49 45 41 38 22 3 27 29 31 21 7 13 36 39 42 46 52 49 45 41 38 21 7 14 22 2 7 12 17 24 2 7 12 17 23 36 39 42 46 50 54 49 45 41 38 26 4

Árbol sintáctico al final de todos los casos.

Caso de éxito 4

```
let boolean booleano;

print (4+5+77);
function bisiesto boolean (int a)   {
        return (a + 4 > 0 || a + 100 > 0 || a - 400 > 0);
}
function dias int (int m, int a) {
        let int dd;
        print ("di cuantos dias tiene el mes ");
        print (m);
        input(dd);
        if (bisiesto(a)) dd = dd + 1;
        return dd;
}
function esFechaCorrecta boolean (int d, int m, int a)      {
        return (d < dias (m, a));
}
function demo ()          {
        if (esFechaCorrecta(25, 10, 2021)) print ("ok");
}
let boolean aaa111; demo();
```

Volcado de tokens

```
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 2006063431 > //token de identificador
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 14, 4 > //token de entero
< 7, > //token de suma
< 14, 5 > //token de entero
< 7, > //token de suma
< 14, 77 > //token de entero
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 30, > //token de function
< 1, 1087772166 > //token de identificador
< 22, > //token de declarando boolean
< 3, > //token de abrir parentesis
< 21, > //token de declarando int
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 29, > //token de return
< 3, > //token de abrir parentesis
< 1, 97 > //token de identificador
< 7, > //token de suma
< 14, 4 > //token de entero
< 11, > //token de mayor que
< 14, 0 > //token de entero
< 13, > //token de o logico
< 1, 97 > //token de identificador
< 7, > //token de suma
< 14, 100 > //token de entero
< 11, > //token de mayor que
< 14, 0 > //token de entero
< 13, > //token de o logico
< 1, 97 > //token de identificador
< 8, > //token de resta
< 14, 400 > //token de entero
< 11, > //token de mayor que
< 14, 0 > //token de entero
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 30, > //token de function
< 1, 3083127 > //token de identificador
< 21, > //token de declarando int
```

< 3, > //token de abrir parentesis
< 21, > //token de declarando int
< 1, 109 > //token de identificador
< 16, > //token de coma
< 21, > //token de declarando int
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 25, > //token de let
< 21, > //token de declarando int
< 1, 3200 > //token de identificador
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 15, "di cuantos dias tiene el mes " > //token de cadena
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 109 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 28, > //token de input
< 3, > //token de abrir parentesis
< 1, 3200 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 1087772166 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 1, 3200 > //token de identificador
< 2, > //token de asignacion
< 1, 3200 > //token de identificador
< 7, > //token de suma
< 14, 1 > //token de entero
< 17, > //token de punto y coma
< 29, > //token de return
< 1, 3200 > //token de identificador
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 30, > //token de function
< 1, 336753094 > //token de identificador
< 22, > //token de declarando boolean
< 3, > //token de abrir parentesis
< 21, > //token de declarando int
< 1, 100 > //token de identificador
< 16, > //token de coma
< 21, > //token de declarando int
< 1, 109 > //token de identificador
< 16, > //token de coma
< 21, > //token de declarando int
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 29, > //token de return
< 3, > //token de abrir parentesis
< 1, 100 > //token de identificador
< 10, > //token de menor que
< 1, 3083127 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 109 > //token de identificador
< 16, > //token de coma
< 1, 97 > //token de identificador
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 30, > //token de function
< 1, 3079651 > //token de identificador
< 3, > //token de abrir parentesis
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 23, > //token de if
< 3, > //token de abrir parentesis
< 1, 336753094 > //token de identificador
< 3, > //token de abrir parentesis
< 14, 25 > //token de entero
< 16, > //token de coma
< 14, 10 > //token de entero
< 16, > //token de coma
< 14, 2021 > //token de entero
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 27, > //token de print
< 3, > //token de abrir parentesis
< 15, "ok" > //token de cadena
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, -1425419728 > //token de identificador
< 17, > //token de punto y coma
< 1, 3079651 > //token de identificador
< 3, > //token de abrir parentesis
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 0, > //token de fin de fichero
< 0, > //token de fin de fichero

Tabla de símbolos

TABLA FUNCION bisiesto # 2:

* Lexema : 'a'
	Atributos:
		+tipo: 'ent'
		+despl: 0

TABLA FUNCION dias # 3:

* Lexema : 'dd'
	Atributos:
		+tipo: 'ent'
		+despl: 2
* Lexema : 'a'
	Atributos:
		+tipo: 'ent'
		+despl: 0
* Lexema : 'm'
	Atributos:
		+tipo: 'ent'
		+despl: 1

TABLA FUNCION esFechaCorrecta # 4:

* Lexema : 'a'
	Atributos:
		+tipo: 'ent'
		+despl: 0
* Lexema : 'd'
	Atributos:
		+tipo: 'ent'
		+despl: 2
* Lexema : 'm'
	Atributos:
		+tipo: 'ent'
		+despl: 1

TABLA FUNCION demo # 5:

TABLA PRINCIPAL # 1:

* Lexema : 'bisiesto'
	Atributos:
		+tipo: 'funcion'
			+numParam: 1
			+TipoParam1: 'ent'
			+ModoParam1: 1
			+TipoRetorno: 'log'
		+EtiqFuncion: 'et_2'
* Lexema : 'esFechaCorrecta'
	Atributos:

```
                +tipo: 'funcion'
                        +numParam: 3
                         +TipoParam1: 'ent'
                          +ModoParam1: 1
                         +TipoParam2: 'ent'
                          +ModoParam2: 1
                         +TipoParam3: 'ent'
                          +ModoParam3: 1
                         +TipoRetorno: 'log'
                +EtiqFuncion: 'et_4'
* Lexema : 'booleano'
        Atributos:
                +tipo: 'log'
                +despl: 0
* Lexema : 'dias'
        Atributos:
                +tipo: 'funcion'
                        +numParam: 2
                         +TipoParam1: 'ent'
                          +ModoParam1: 1
                         +TipoParam2: 'ent'
                          +ModoParam2: 1
                         +TipoRetorno: 'ent'
                +EtiqFuncion: 'et_3'
* Lexema : 'aaa111'
        Atributos:
                +tipo: 'log'
                +despl: 1
* Lexema : 'demo'
        Atributos:
                +tipo: 'funcion'
                        +numParam: 0
                         +TipoRetorno: 'vacio'
                +EtiqFuncion: 'et_5'
```

Parse

D 1 2 5 11 2 7 13 36 39 42 46 51 47 51 47 51 49 45 41 38 3 27 28 11 30 10 33 21 7 15 19 36 39 42 46 53 36 39 42 46 50 54 47 51 49 44 46 51 49 45 41 37 39 42 46 50 54 47 51 49 44 46 51 49 45 41 37 39 42 46 50 54 48 51 49 44 46 51 49 45 41 38 49 45 41 38 22 3 27 28 10 30 10 32 10 33 21 5 10 21 7 13 36 39 42 46 52 49 45 41 38 21 7 13 36 39 42 46 50 54 49 45 41 38 21 7 14 21 6 36 39 42 46 50 55 23 36 39 42 46 50 54 49 45 41 38 26 49 45 41 38 12 16 36 39 42 46 50 54 47 51 49 45 41 38 21 7 15 19 36 39 42 46 50 54 49 45 41 38 22 3 27 28 11 30 10 32 10 32 10 33 21 7 15 19 36 39 42 46 53 36 39 42 46 50 54 49 43 46 50 55 23 36 39 42 46 50 54 49 45 41 38 25 36 39 42 46 50 54 49 45 41 38 26 49 45 41 38 49 45 41 38 22 3 27 29 31 21 6 36 39 42 46 50 55 23 36 39 42 46 51 49 45 41 38 25 36 39 42 46 51 49 45 41 38 25 36 39 42 46 51 49 45 41 38 26 49 45 41 38 13 36 39 42 46 52 49 45 41 38 22 2 5 11 2 7 12 17 24 4

Árbol sintáctico al final de todos los casos.

Caso de éxito 5

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

function f1 int(int f1, boolean b1)
{
		print(ss);
		x -= xx-f1;
		boolean_1 = boolean_1|| boolean_2;
		return (x);
}


function f2 boolean(int f2, boolean b1)
{
		print ((04+5+77+(88+f2)));
		return boolean_1&&boolean_2&& b1;
}

x = x + 6  - z  + 1  - (2 + y  + 6) ;
print (f1 (x, f2 (3, boolean_2)));
```

Volcado de tokens:

```
< 25, > //token de let
< 21, > //token de declarando int
< 1, 120 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 21, > //token de declarando int
< 1, 3840 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 26, > //token de string
< 1, 3680 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 2058423770 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 22, > //token de declarando boolean
< 1, 2058423771 > //token de identificador
< 17, > //token de punto y coma
< 25, > //token de let
< 21, > //token de declarando int
< 1, 121 > //token de identificador
< 17, > //token de punto y coma
< 30, > //token de function
< 1, 3211 > //token de identificador
< 21, > //token de declarando int
< 3, > //token de abrir parentesis
< 21, > //token de declarando int
< 1, 3211 > //token de identificador
< 16, > //token de coma
< 22, > //token de declarando boolean
< 1, 3087 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 3680 > //token de identificador
```

< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 1, 120 > //token de identificador
< 9, > //token de asignacion resta
< 1, 3840 > //token de identificador
< 8, > //token de resta
< 1, 3211 > //token de identificador
< 17, > //token de punto y coma
< 1, 2058423770 > //token de identificador
< 2, > //token de asignacion
< 1, 2058423770 > //token de identificador
< 13, > //token de o logico
< 1, 2058423771 > //token de identificador
< 17, > //token de punto y coma
< 29, > //token de return
< 3, > //token de abrir parentesis
< 1, 120 > //token de identificador
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 30, > //token de function
< 1, 3212 > //token de identificador
< 22, > //token de declarando boolean
< 3, > //token de abrir parentesis
< 21, > //token de declarando int
< 1, 3212 > //token de identificador
< 16, > //token de coma
< 22, > //token de declarando boolean
< 1, 3087 > //token de identificador
< 4, > //token de cerrar parentesis
< 5, > //token de abrir llaves
< 27, > //token de print
< 3, > //token de abrir parentesis
< 3, > //token de abrir parentesis
< 14, 4 > //token de entero
< 7, > //token de suma
< 14, 5 > //token de entero
< 7, > //token de suma
< 14, 77 > //token de entero
< 7, > //token de suma
< 3, > //token de abrir parentesis
< 14, 88 > //token de entero
< 7, > //token de suma
< 1, 3212 > //token de identificador
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 29, > //token de return
< 1, 2058423770 > //token de identificador
< 12, > //token de y logico
< 1, 2058423771 > //token de identificador
< 12, > //token de y logico
< 1, 3087 > //token de identificador
< 17, > //token de punto y coma
< 6, > //token de cerrar llaves
< 1, 120 > //token de identificador
< 2, > //token de asignacion
< 1, 120 > //token de identificador
< 7, > //token de suma
< 14, 6 > //token de entero
< 8, > //token de resta
< 1, 122 > //token de identificador
< 7, > //token de suma
< 14, 1 > //token de entero
< 8, > //token de resta
< 3, > //token de abrir parentesis
< 14, 2 > //token de entero
< 7, > //token de suma
< 1, 121 > //token de identificador
< 7, > //token de suma
< 14, 6 > //token de entero
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 27, > //token de print
< 3, > //token de abrir parentesis
< 1, 3211 > //token de identificador
< 3, > //token de abrir parentesis
< 1, 120 > //token de identificador
< 16, > //token de coma
< 1, 3212 > //token de identificador
< 3, > //token de abrir parentesis
< 14, 3 > //token de entero
< 16, > //token de coma
< 1, 2058423771 > //token de identificador
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 4, > //token de cerrar parentesis
< 17, > //token de punto y coma
< 0, > //token de fin de fichero
< 0, > //token de fin de fichero

Tabla de símbolos

TABLA FUNCION f1 # 2:

* Lexema : 'f1'
	Atributos:
		+tipo: 'ent'
		+despl: 1
* Lexema : 'b1'
	Atributos:
		+tipo: 'log'
		+despl: 0

TABLA FUNCION f2 # 3:

* Lexema : 'f2'
	Atributos:
		+tipo: 'ent'
		+despl: 1
* Lexema : 'b1'
	Atributos:
		+tipo: 'log'
		+despl: 0

TABLA PRINCIPAL # 1:

* Lexema : 'xx'
	Atributos:
		+tipo: 'ent'
		+despl: 1
* Lexema : 'ss'
	Atributos:
		+tipo: 'cadena'
		+despl: 2
* Lexema : 'x'
	Atributos:
		+tipo: 'ent'
		+despl: 0
* Lexema : 'y'
	Atributos:
		+tipo: 'ent'
		+despl: 68
* Lexema : 'boolean_2'
	Atributos:
		+tipo: 'log'
		+despl: 67
* Lexema : 'z'
	Atributos:
		+tipo: 'ent'
		+despl: 69
* Lexema : 'boolean_1'
	Atributos:
		+tipo: 'log'
		+despl: 66
* Lexema : 'f1'

Atributos:
+tipo: 'funcion'
+numParam: 2
+TipoParam1: 'ent'
+ModoParam1: 1
+TipoParam2: 'log'
+ModoParam2: 1
+TipoRetorno: 'ent'
+EtiqFuncion: 'et_2'

* Lexema : 'f2'
Atributos:
+tipo: 'funcion'
+numParam: 2
+TipoParam1: 'ent'
+ModoParam1: 1
+TipoParam2: 'log'
+ModoParam2: 1
+TipoRetorno: 'log'
+EtiqFuncion: 'et_3'

Parse
D 1 2 5 10 2 5 10 2 5 9 2 5 11 2 5 11 2 5 10 3 27 28 10 30 10 32 11 33 21 7 13 36 39 42 46 50 54 49 45 41 38 21 7 12 18 36 39 42 46 50 54 48 50 54 49 45 41 38 21 7 12 16 36 39 42 46 50 54 49 45 41 37 39 42 46 50 54 49 45 41 38 21 7 15 19 36 39 42 46 53 36 39 42 46 50 54 49 45 41 38 49 45 41 38 22 3 27 28 11 30 10 32 11 33 21 7 13 36 39 42 46 53 36 39 42 46 51 47 51 47 51 47 53 36 39 42 46 51 47 50 54 49 45 41 38 49 45 41 38 49 45 41 38 21 7 15 19 36 39 42 46 50 54 49 45 40 42 46 50 54 49 45 40 42 46 50 54 49 45 41 38 22 2 7 12 16 36 39 42 46 50 54 47 51 48 50 54 47 51 48 53 36 39 42 46 51 47 50 54 47 51 49 45 41 38 49 45 41 38 2 7 13 36 39 42 46 50 55 23 36 39 42 46 50 54 49 45 41 38 25 36 39 42 46 50 55 23 36 39 42 46 51 49 45 41 38 25 36 39 42 46 50 54 49 45 41 38 26 49 45 41 38 26 49 45 41 38 4

Árbol sintáctico al final de todos los casos.

Casos de fracaso

Como indicamos previamente, los errores se gestionarán de manera que si encontramos un error pararemos de analizar el resto del código.

Los dos primeros casos serán errores del léxico, en el que introduciremos caracteres no existentes en nuestro lenguaje, o que llegue un caracter inesperado. El siguiente caso será un código con sintaxis incorrecta. Finalmente, los dos últimos ejemplos contendrá errores semánticos.

Caso de fracaso 1 (léxico)

```
let int a;
// comentario de línea no permitido en nuestro programa
```

Volcado de error

Error léxico en línea 2: se esperaba el caracter '*'. Solo están permitidos los comentarios del tipo /* <comentario> */.

Caso de fracaso 2 (léxico)

```
funçtion ffç boolean(boolean ss) {
        print(“Hola”);
}
```

Volcado de error

Error léxico en línea 1: caracter 'ç' no reconocido

Caso de fracaso 3 (sintáctico)

```
let int n;
if (n<3 /*falta cerrar paréntesis*/
print(“a”);
```

Volcado de error

Error sintáctico en línea 3: se esperaba ')'

Caso de fracaso 4 (semántico)

```
let int n;
n = “hola”;
```

Volcado de error

Error semántico en línea 3: La llamada de la función es incorrecta o la asignación es incorrecta.

Caso de fracaso 5 (semántico)

```
function bisiesto boolean (int a)  {
        return (a + 1) ;
}
```

Volcado de error:

Error semántico en línea 4: El tipo del valor de retorno es incorrecto.

Árbol resultado de:

Gramática: C:\Users\Jaime\Documents\gramatica.txt

Parse: C:\Users\Jaime\Documents\parse-1.txt

- P0 (1)
  - P (2)
    - B (5)
      - let
      - T (10)
        - int
      - id
      - ;
    - P (2)
      - B (5)
        - let
        - T (10)
          - int
        - id
        - ;
      - P (2)
        - B (5)
          - let
          - T (11)
            - boolean
          - id
          - ;
        - P (2)
          - B (7)
            - S (12)
              - id
              - S1 (16)
                - =
                - E (36)
                  - O (39)
                    - R (42)
                      - U (46)

- V (51)
  - entero
- U1 (49)
  - lambd
- R1 (45)
  - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd
- ;

- P (2)
  - B (7)
    - S (12)
      - id
      - S1 (16)
        - =
        - E (36)
          - O (39)
            - R (42)
              - U (46)
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (49)
                  - lambd
              - R1 (45)
                - lambd
            - O1 (41)
              - lambd
          - E1 (38)
            - lambd
        - ;
    - P (2)
      - B (6)
        - if
        - (
        - E (36)
          - O (39)

- R (42)
  - U (46)
    - V (50)
      - id
      - V1 (54)
        - lambd
    - U1 (49)
      - lambd
  - R1 (43)
    - <
    - U (46)
      - V (50)
        - id
        - V1 (54)
          - lambd
      - U1 (49)
        - lambd
    - R1 (45)
      - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd

-)
- S (12)
  - id
  - S1 (18)
    - -=
    - E (36)
      - O (39)
        - R (42)
          - U (46)
            - V (51)
              - entero
            - U1 (49)
              - lambd
          - R1 (45)
            - lambd
        - O1 (41)
          - lambd

- E1 (38)
  - lambd
- ;
- P (2)
  - B (7)
    - S (12)
      - id
      - S1 (16)
        - =
        - E (36)
          - O (39)
            - R (42)
              - U (46)
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (48)
                  - -
                  - V (50)
                    - id
                    - V1 (54)
                      - lambd
                  - U1 (49)
                    - lambd
              - R1 (45)
                - lambd
            - O1 (41)
              - lambd
          - E1 (38)
            - lambd
        - ;
  - P (2)
    - B (7)
      - S (13)
        - print
        - (
        - E (36)
          - O (39)
            - R (42)

- U (46)
  - V (50)
    - id
    - V1 (54)
      - lambd
  - U1 (49)
    - lambd
- R1 (45)
  - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd
-)
- ;

- P (2)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
  - P (4)
    - eof

Árbol resultado de:

Gramática: C:\Users\Jaime\Documents\gramatica.txt

Parse: C:\Users\Jaime\Documents\parse-2.txt

- P0 (1)
  - P (2)
    - B (5)
      - let
      - T (10)
        - int
      - id
      - ;
    - P (2)
      - B (5)
        - let
        - T (11)
          - boolean
        - id
        - ;
      - P (2)
        - B (5)
          - let
          - T (9)
            - string
          - id
          - ;
        - P (2)
          - B (5)
            - let
            - T (10)
              - int
            - id
            - ;
          - P (2)
            - B (5)
              - let
              - T (11)
                - boolean
              - id
              - ;
            - P (2)
              - B (7)
                - S (14)
                  - input
                  - (
                  - id

-)
- ;

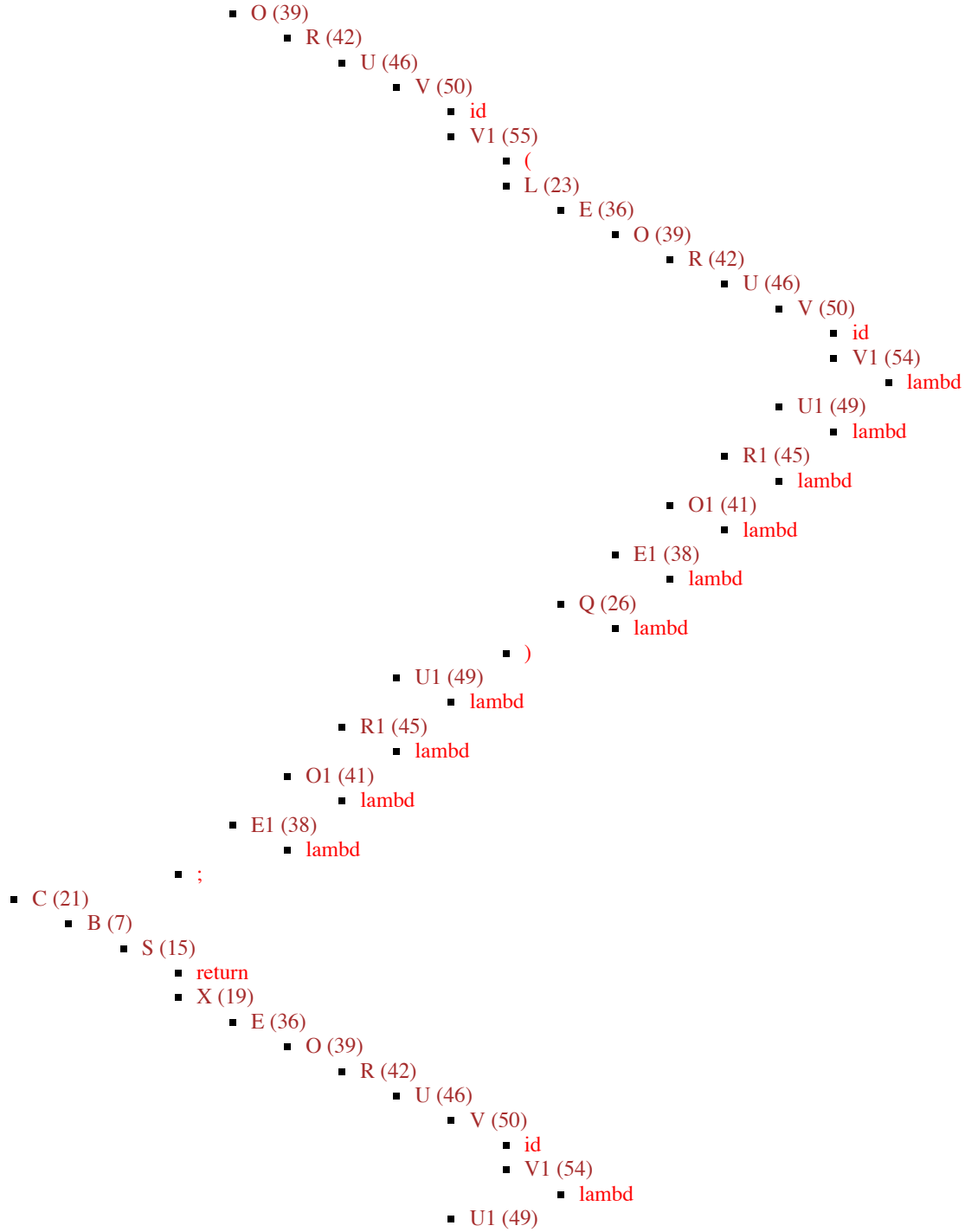
- P (2)
  - B (7)
    - S (12)
      - id
      - S1 (16)
        - =
        - E (36)
          - O (39)
            - R (42)
              - U (46)
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (49)
                  - lambd
              - R1 (45)
                - lambd
            - O1 (41)
              - lambd
          - E1 (38)
            - lambd
        - ;
  - P (2)
    - B (6)
      - if
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (40)
            - &&
            - R (42)
              - U (46)
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (49)

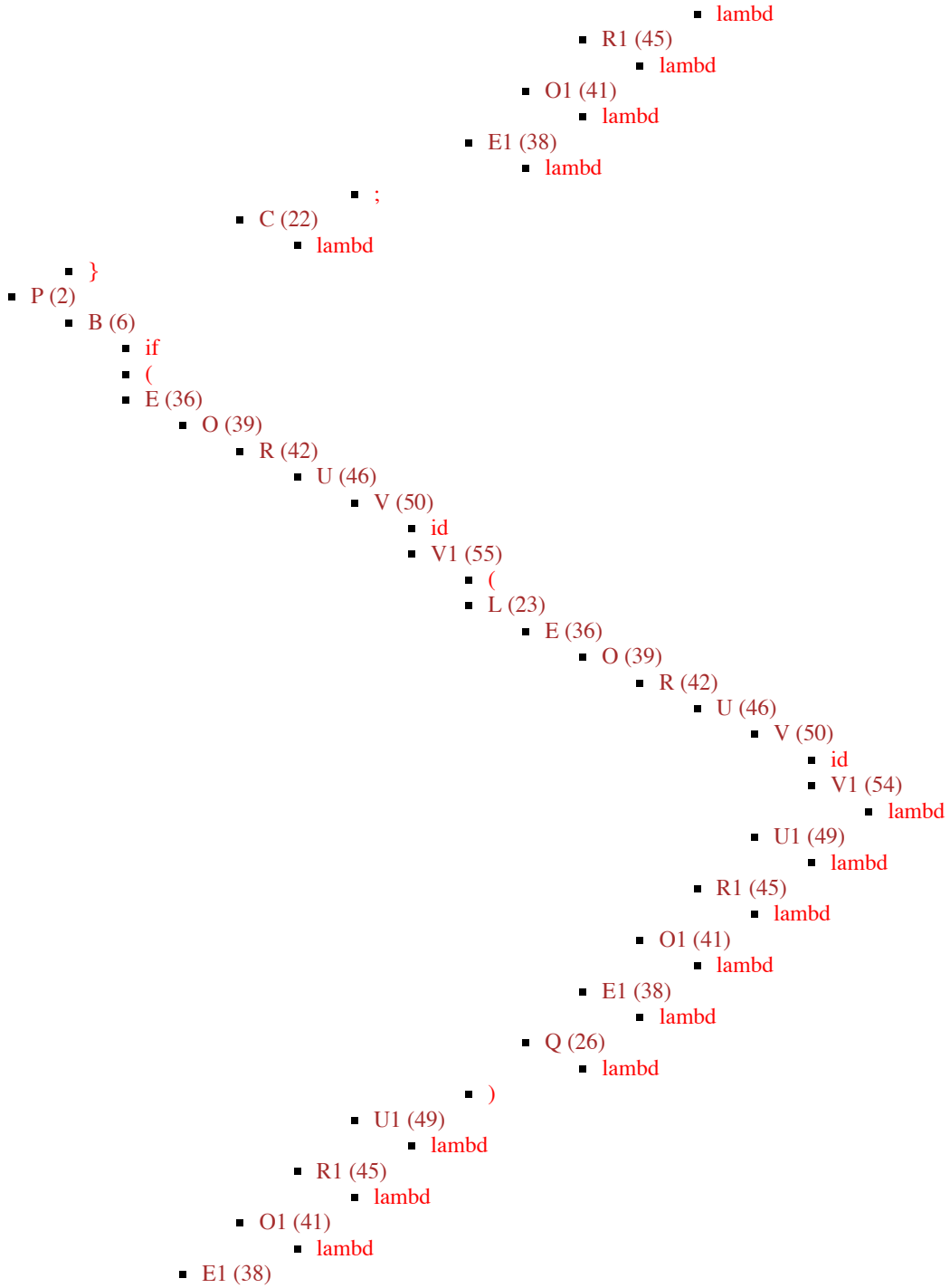
- lambd
- R1 (45)
  - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd
-)
- S (12)
  - id
  - S1 (16)
    - =
    - E (36)
      - O (39)
        - R (42)
          - U (46)
            - V (52)
              - cadena
            - U1 (49)
              - lambd
          - R1 (45)
            - lambd
        - O1 (41)
          - lambd
      - E1 (38)
        - lambd
    - ;
- P (2)
  - B (7)
    - S (12)
      - id
      - S1 (18)
        - -=
        - E (36)
          - O (39)
            - R (42)
              - U (46)
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (48)
                  - -
                  - V (51)
                    - entero
                  - U1 (49)
                    - lambd
              - R1 (45)
                - lambd
            - O1 (41)

- lambd
- E1 (38)
  - lambd
- ;

- P (2)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (51)
                - entero
              - U1 (47)
                - +
                - V (50)
                  - id
                  - V1 (54)
                    - lambd
                - U1 (47)
                  - +
                  - V (50)
                    - id
                    - V1 (54)
                      - lambd
                  - U1 (49)
                    - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
  - P (3)
    - F (27)
      - function
      - id
      - H (28)
        - T (11)
          - boolean
      - (
      - A (30)
        - T (11)
          - boolean
        - id
        - K (33)

- lambd
-)
- {
- C (21)
  - B (7)
    - S (12)
      - id
      - S1 (16)
        - =
        - E (36)
          - O (39)
            - R (42)
              - U (46)
                - V (51)
                  - entero
                - U1 (49)
                  - lambd
              - R1 (45)
                - lambd
            - O1 (41)
              - lambd
          - E1 (38)
            - lambd
        - ;
  - C (21)
    - B (6)
      - if
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - S (12)
        - id
        - S1 (16)
          - =
          - E (36)


O (39)
R (42)
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (26)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)


lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (22)
lambd
}
P (2)
B (6)
if
(
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (26)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)

- lambd
-)
- S (13)
  - print
  - (
  - E (36)
    - O (39)
      - R (42)
        - U (46)
          - V (50)
            - id
            - V1 (54)
              - lambd
          - U1 (49)
            - lambd
        - R1 (45)
          - lambd
      - O1 (41)
        - lambd
    - E1 (38)
      - lambd
  -)
  - ;
- P (4)
  - eof

Árbol resultado de:

Gramática: C:\Users\Jaime\Documents\gramatica.txt

Parse: C:\Users\Jaime\Documents\parse-3.txt

- P0 (1)
  - P (2)
    - B (5)
      - let
      - T (9)
        - string
      - id
      - ;
    - P (3)
      - F (27)
        - function
        - id
        - H (29)
          - lambd
        - (
        - A (30)
          - T (9)
            - string
          - id
          - K (33)
            - lambd
        -)
        - {
        - C (21)
          - B (7)
            - S (13)
              - print
              - (
              - E (36)
                - O (39)
                  - R (42)
                    - U (46)

- V (52)
  - cadena
- U1 (49)
  - lambd
- R1 (45)
  - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd
-)
- ;

- C (21)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
    - C (22)
      - lambd
- }

- P (3)
  - F (27)

- function
- id
- H (29)
  - lambd
- (
- A (31)
  - lambd
-)
- {
- C (21)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (52)
                - cadena
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
  - C (21)
    - B (7)
      - S (14)
        - input
        - (
        - id
        -)
        - ;
    - C (22)
      - lambd

- }
- P (2)
  - B (7)
    - S (12)
      - id
      - S1 (17)
        - (
        - L (24)
          - lambd
        -)
        - ;
    - P (2)
      - B (7)
        - S (12)
          - id
          - S1 (17)
            - (
            - L (23)
              - E (36)
                - O (39)
                  - R (42)
                    - U (46)
                      - V (50)
                        - id
                        - V1 (54)
                          - lambd
                      - U1 (49)
                        - lambd
                    - R1 (45)
                      - lambd
                  - O1 (41)
                    - lambd
                - E1 (38)
                  - lambd
              - Q (26)
                - lambd
            -)
            - ;
      - P (4)
        - eof

Árbol resultado de:

Gramática: C:\Users\Jaime\Documents\gramatica.txt

Parse: C:\Users\Jaime\Documents\parse-4.txt

- P0 (1)
  - P (2)
    - B (5)
      - let
      - T (11)
        - boolean
      - id
      - ;
    - P (2)
      - B (7)
        - S (13)
          - print
          - (
          - E (36)
            - O (39)
              - R (42)
                - U (46)
                  - V (51)
                    - entero
                  - U1 (47)
                    - +
                    - V (51)
                      - entero
                    - U1 (47)
                      - +
                      - V (51)
                        - entero
                      - U1 (49)
                        - lambd
              - R1 (45)
                - lambd
            - O1 (41)
              - lambd
            - E1 (38)
              - lambd
          -)
          - ;
      - P (3)
        - F (27)
          - function
          - id
          - H (28)

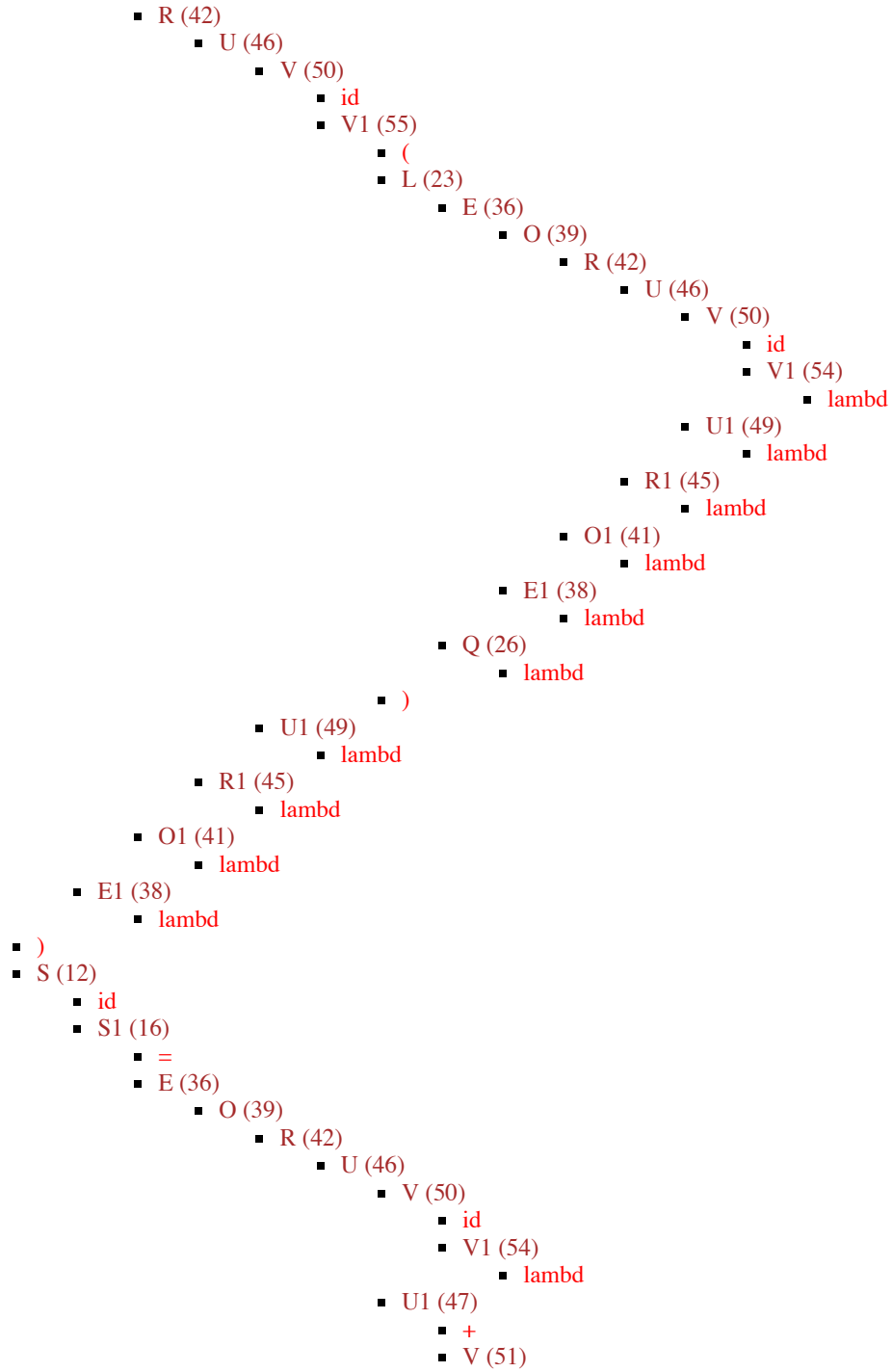

T (11)
boolean
(
A (30)
T (10)
int
id
K (33)
lambd
)
{
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)
R (42)
U (46)
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (47)
+
V (51)
entero
U1 (49)
lambd
R1 (44)
>
U (46)
V (51)
entero
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (37)
||
O (39)

- R (42)
  - U (46)
    - V (50)
      - id
      - V1 (54)
        - lambd
    - U1 (47)
      - +
      - V (51)
        - entero
      - U1 (49)
        - lambd
  - R1 (44)
    - >
    - U (46)
      - V (51)
        - entero
      - U1 (49)
        - lambd
    - R1 (45)
      - lambd
- O1 (41)
  - lambd
- E1 (37)
  - ||
  - O (39)
    - R (42)
      - U (46)
        - V (50)
          - id
          - V1 (54)
            - lambd
        - U1 (48)
          - -
          - V (51)
            - entero
          - U1 (49)
            - lambd
      - R1 (44)
        - >
        - U (46)
          - V (51)
            - entero
          - U1 (49)
            - lambd
        - R1 (45)
          - lambd
    - O1 (41)
      - lambd
  - E1 (38)

- lambd
-)
- U1 (49)
- lambd
- R1 (45)
- lambd
- O1 (41)
- lambd
- E1 (38)
- lambd
- ;
- C (22)
- lambd
- }
- P (3)
- F (27)
- function
- id
- H (28)
- T (10)
- int
- (
- A (30)
- T (10)
- int
- id
- K (32)
- ,
- T (10)
- int
- id
- K (33)
- lambd
-)
- {
- C (21)
- B (5)
- let
- T (10)
- int
- id
- ;
- C (21)
- B (7)
- S (13)
- print
- (
- E (36)
- O (39)
- R (42)

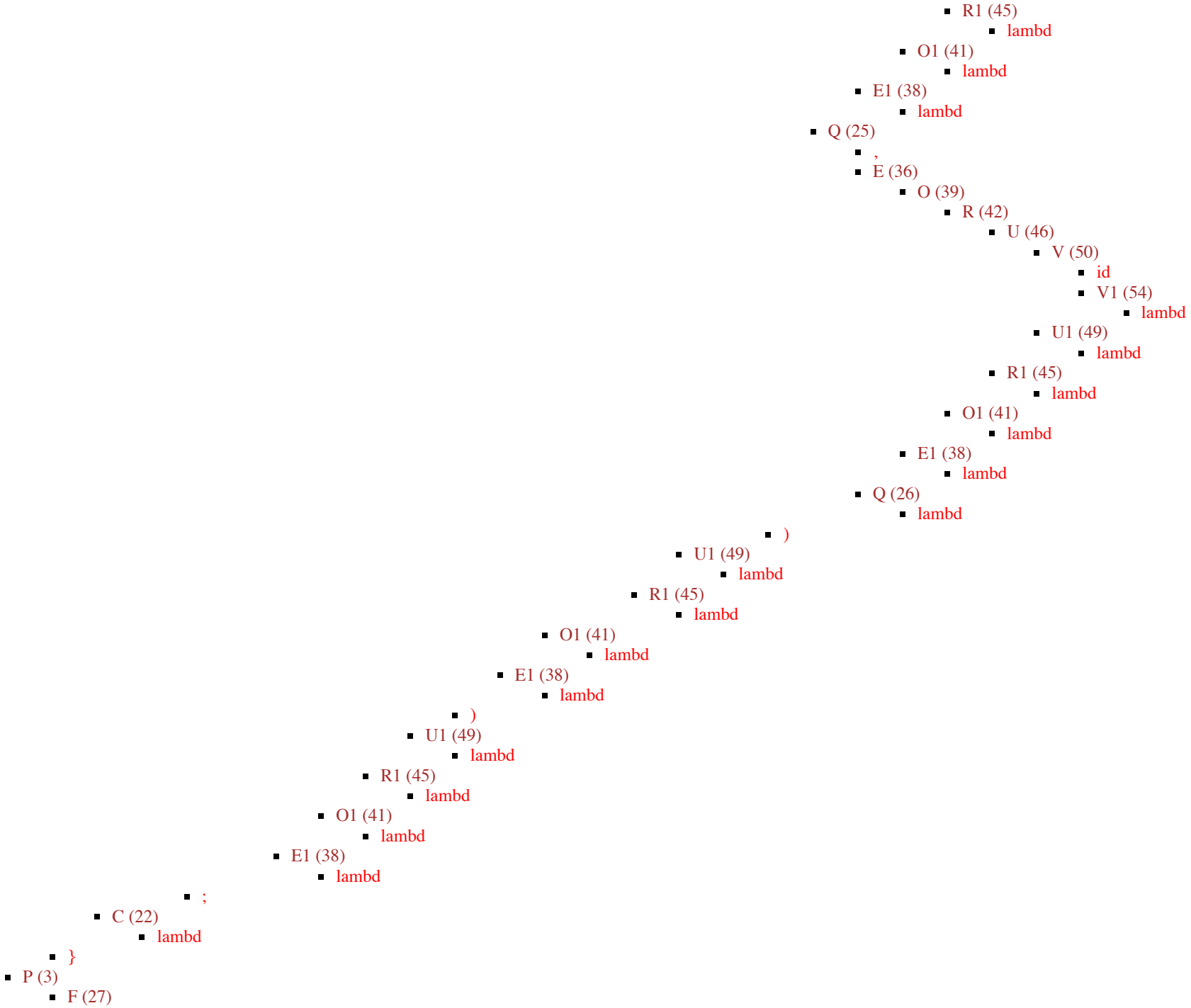
- U (46)
  - V (52)
    - cadena
  - U1 (49)
    - lambd
- R1 (45)
  - lambd
- O1 (41)
  - lambd
- E1 (38)
  - lambd
-)
- ;

- C (21)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
  - C (21)
    - B (7)
      - S (14)
        - input
        - (
        - id
        -)
        - ;
    - C (21)
      - B (6)
        - if
        - (
        - E (36)
          - O (39)

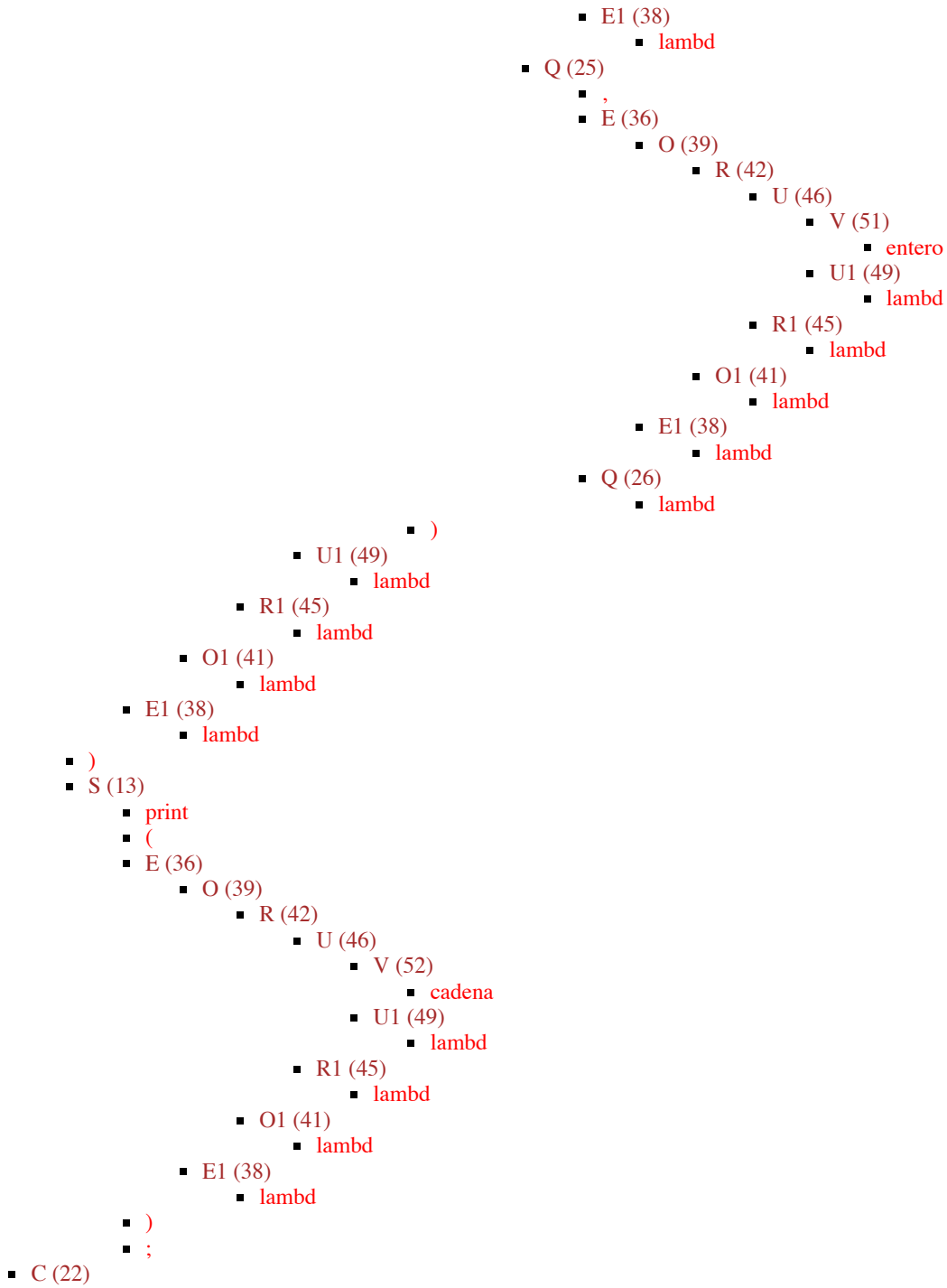

R (42)
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (26)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
S (12)
id
S1 (16)
=
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (47)
+
V (51)


entero
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (22)
lambd
}
P (3)
F (27)
function
id
H (28)
T (11)
boolean
(
A (30)
T (10)
int
id
K (32)
,
T (10)


int
id
K (32)
,
T (10)
int
id
K (33)
lambd
)
{
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)
R (42)
U (46)
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (43)
<
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd


R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (25)
,
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (26)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (22)
lambd
}
P (3)
F (27)

- function
- id
- H (29)
  - lambd
- (
- A (31)
  - lambd
-)
- {
- C (21)
  - B (6)
    - if
    - (
    - E (36)
      - O (39)
        - R (42)
          - U (46)
            - V (50)
              - id
              - V1 (55)
                - (
                - L (23)
                  - E (36)
                    - O (39)
                      - R (42)
                        - U (46)
                          - V (51)
                            - entero
                          - U1 (49)
                            - lambd
                        - R1 (45)
                          - lambd
                      - O1 (41)
                        - lambd
                    - E1 (38)
                      - lambd
                - Q (25)
                  - ,
                  - E (36)
                    - O (39)
                      - R (42)
                        - U (46)
                          - V (51)
                            - entero
                          - U1 (49)
                            - lambd
                        - R1 (45)
                          - lambd
                      - O1 (41)
                        - lambd


E1 (38)
lambd
Q (25)
,
E (36)
O (39)
R (42)
U (46)
V (51)
entero
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (26)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
S (13)
print
(
E (36)
O (39)
R (42)
U (46)
V (52)
cadena
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
;
C (22)

- lambd
- }
- P (2)
  - B (5)
    - let
    - T (11)
      - boolean
    - id
    - ;
  - P (2)
    - B (7)
      - S (12)
        - id
        - S1 (17)
          - (
          - L (24)
            - lambd
          -)
          - ;
    - P (4)
      - eof

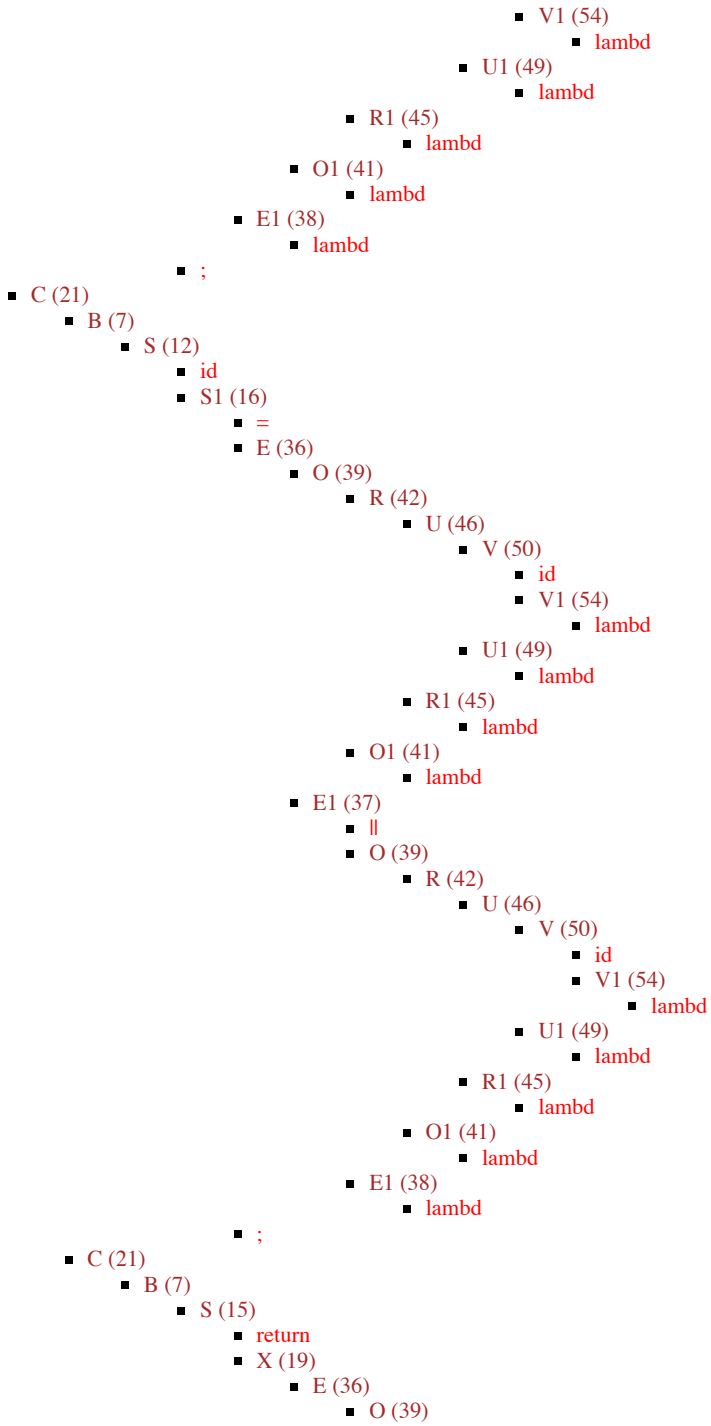
Árbol resultado de:

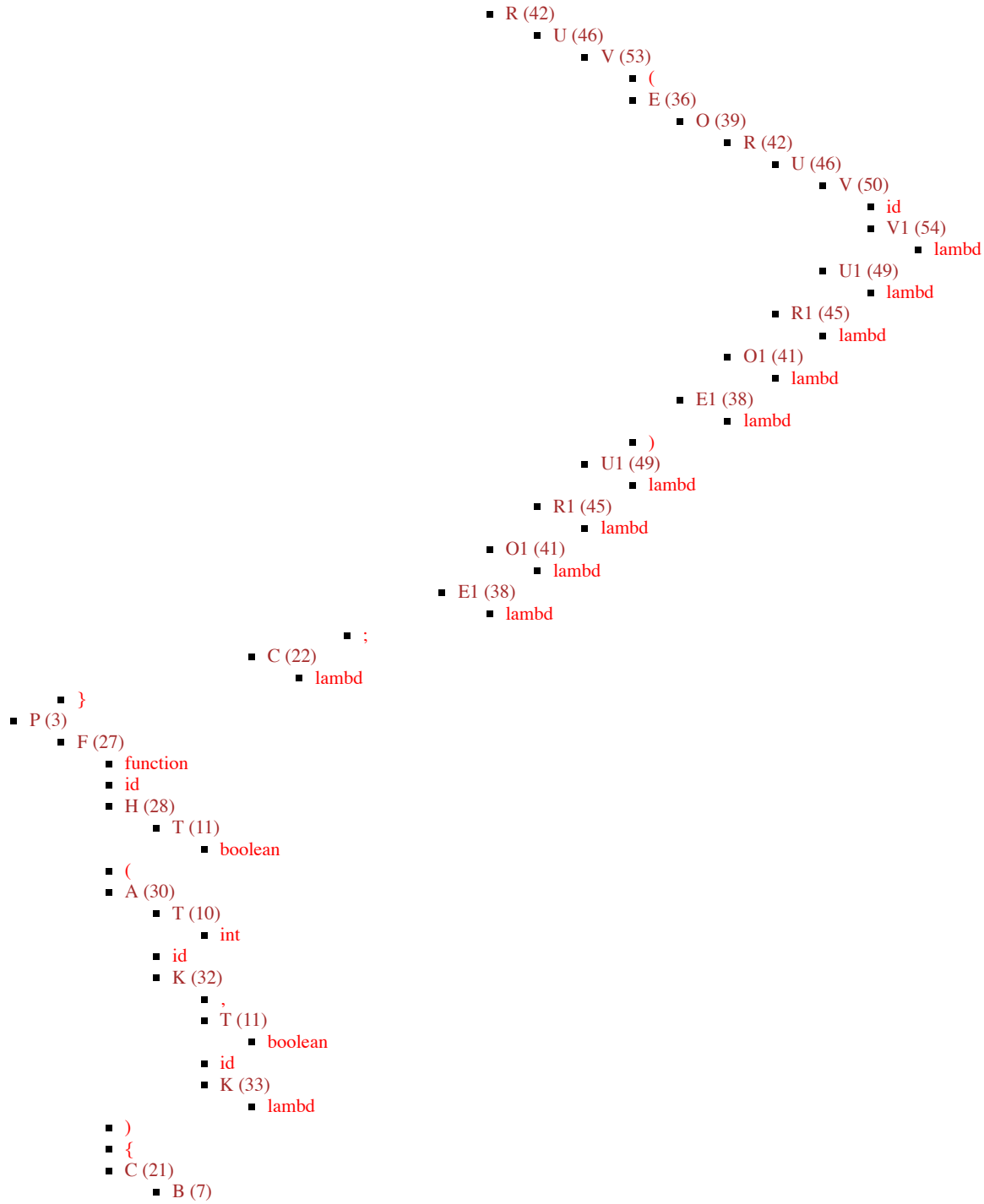
Gramática: C:\Users\Jaime\Documents\gramatica.txt

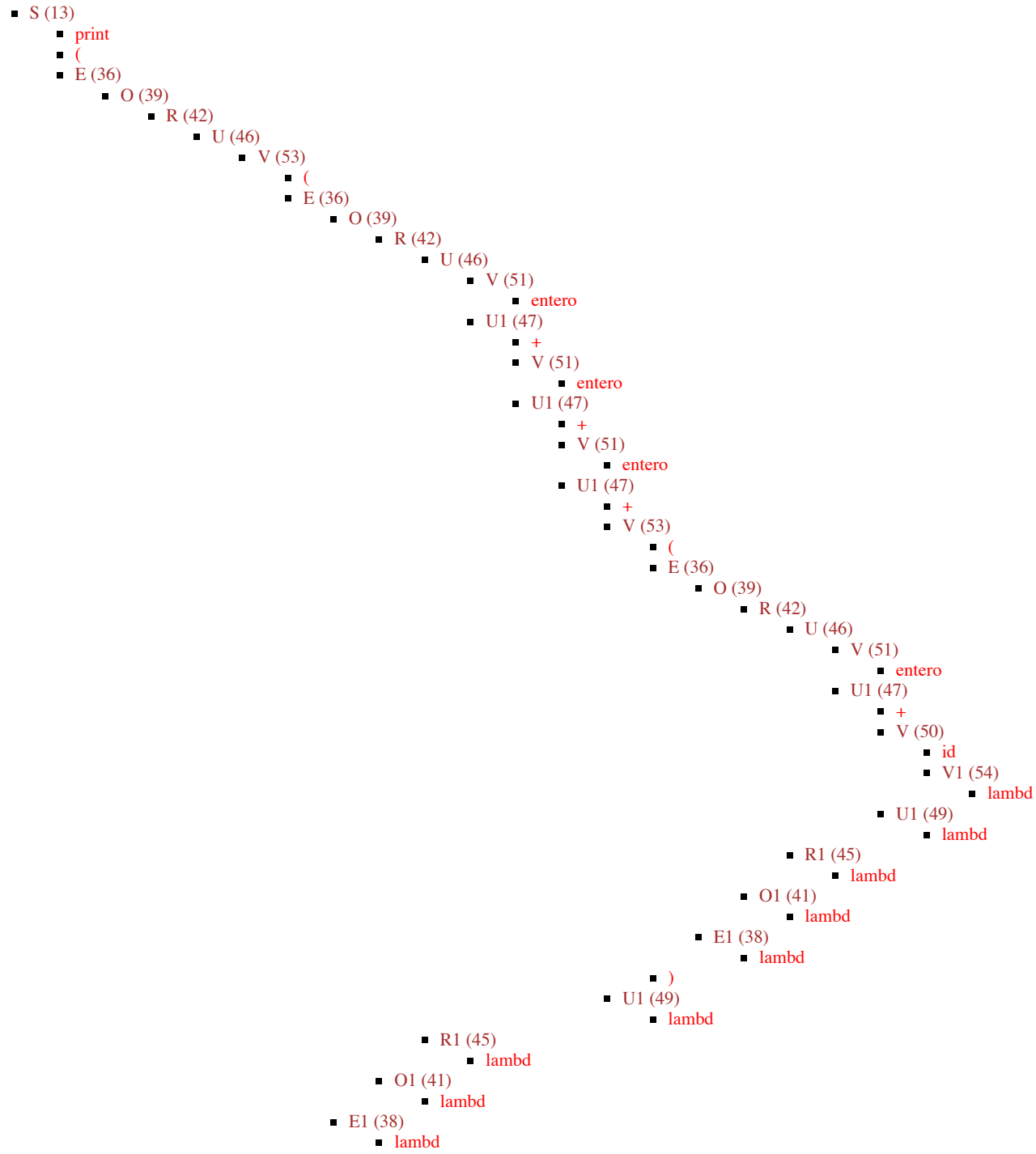
Parse: C:\Users\Jaime\Documents\parse-5.txt

- P0 (1)
  - P (2)
    - B (5)
      - let
      - T (10)
        - int
      - id
      - ;
    - P (2)
      - B (5)
        - let
        - T (10)
          - int
        - id
        - ;
      - P (2)
        - B (5)
          - let
          - T (9)
            - string
          - id
          - ;
        - P (2)
          - B (5)
            - let
            - T (11)
              - boolean
            - id
            - ;
          - P (2)
            - B (5)
              - let
              - T (11)
                - boolean
              - id
              - ;
            - P (2)
              - B (5)
                - let
                - T (10)
                  - int
                - id
                - ;
              - P (3)
                - F (27)
                  - function
                  - id
                  - H (28)
                    - T (10)

- int
- (
- A (30)
  - T (10)
    - int
  - id
  - K (32)
    - ,
    - T (11)
      - boolean
    - id
    - K (33)
      - lambd
-)
- {
- C (21)
  - B (7)
    - S (13)
      - print
      - (
      - E (36)
        - O (39)
          - R (42)
            - U (46)
              - V (50)
                - id
                - V1 (54)
                  - lambd
              - U1 (49)
                - lambd
            - R1 (45)
              - lambd
          - O1 (41)
            - lambd
        - E1 (38)
          - lambd
      -)
      - ;
  - C (21)
    - B (7)
      - S (12)
        - id
        - S1 (18)
          - -=
          - E (36)
            - O (39)
              - R (42)
                - U (46)
                  - V (50)
                    - id
                    - V1 (54)
                      - lambd
                  - U1 (48)
                    - -
                    - V (50)
                      - id

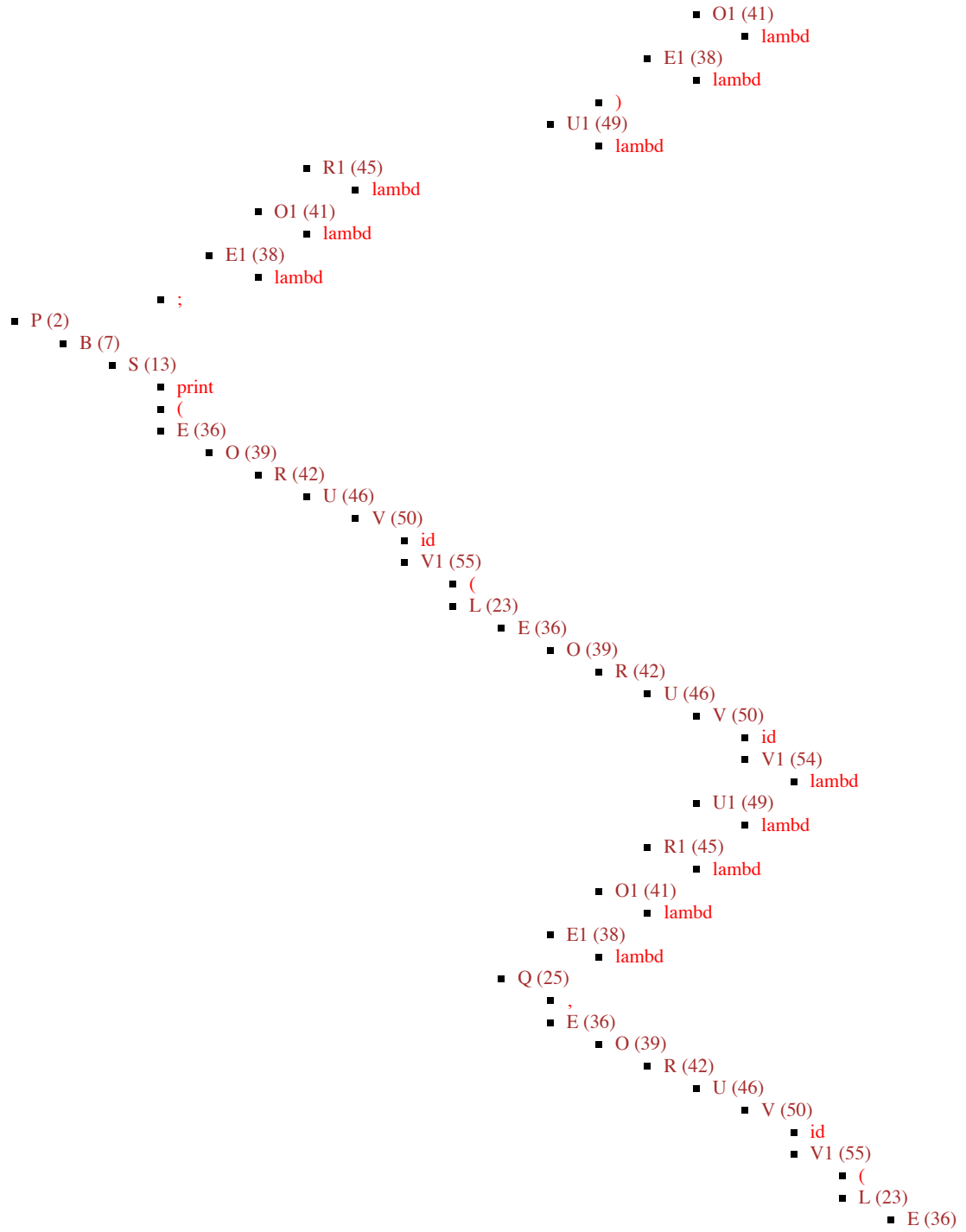

V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (21)
B (7)
S (12)
id
S1 (16)
=
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (37)
||
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)

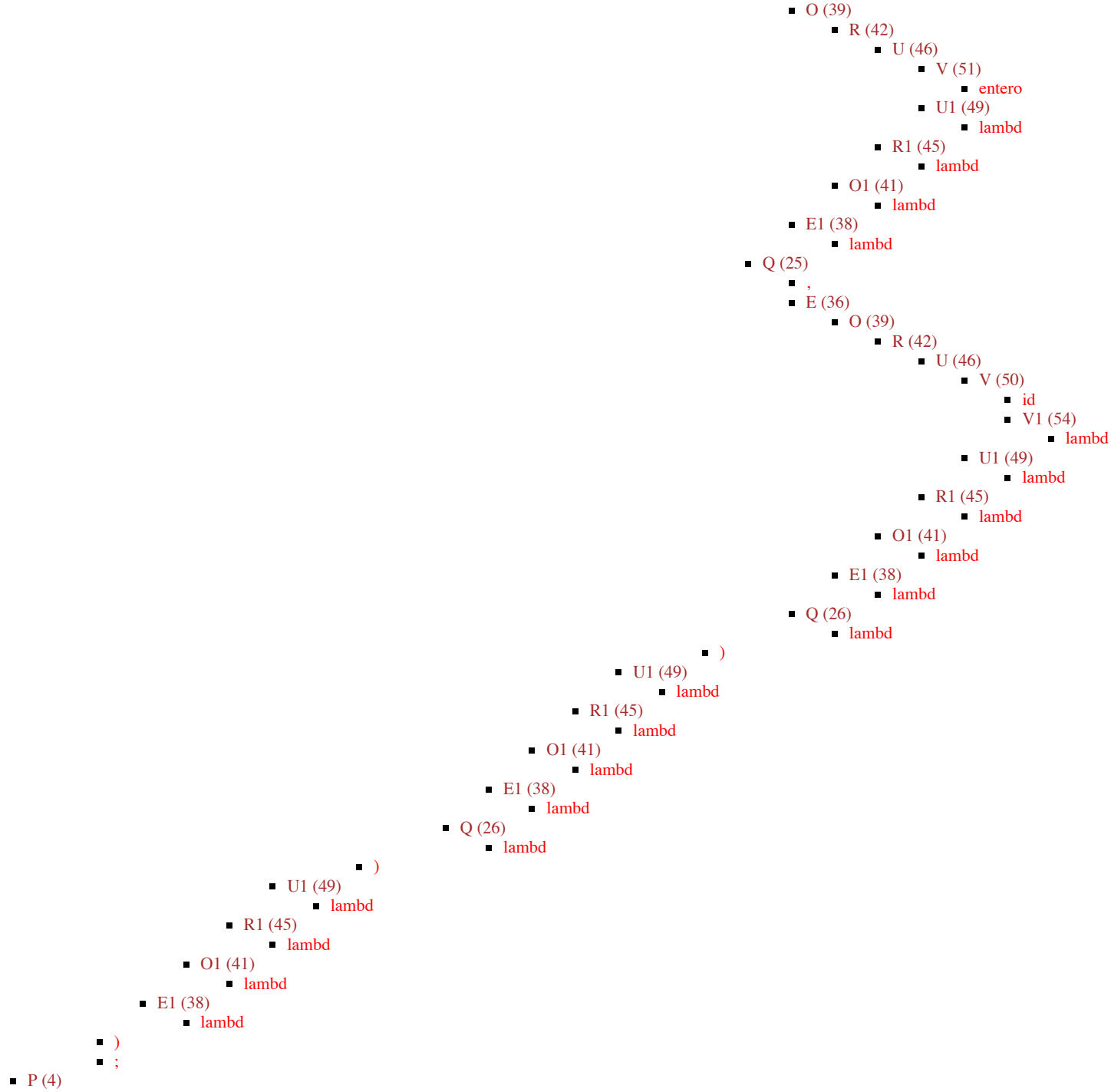

R (42)
U (46)
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;
C (22)
lambd
}
P (3)
F (27)
function
id
H (28)
T (11)
boolean
(
A (30)
T (10)
int
id
K (32)
,
T (11)
boolean
id
K (33)
lambd
)
{
C (21)
B (7)


S (13)
print
(
E (36)
O (39)
R (42)
U (46)
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (51)
entero
U1 (47)
+
V (51)
entero
U1 (47)
+
V (51)
entero
U1 (47)
+
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (51)
entero
U1 (47)
+
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd


)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
)
;
C (21)
B (7)
S (15)
return
X (19)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (40)
&&
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (40)
&&
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd


;
C (22)
lambd
}
P (2)
B (7)
S (12)
id
S1 (16)
=
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (47)
+
V (51)
entero
U1 (48)
-
V (50)
id
V1 (54)
lambd
U1 (47)
+
V (51)
entero
U1 (48)
-
V (53)
(
E (36)
O (39)
R (42)
U (46)
V (51)
entero
U1 (47)
+
V (50)
id
V1 (54)
lambd
U1 (47)
+
V (51)
entero
U1 (49)
lambd
R1 (45)
lambd


P (2)
B (7)
S (13)
print
(
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (54)
lambd
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
Q (25)
,
E (36)
O (39)
R (42)
U (46)
V (50)
id
V1 (55)
(
L (23)
E (36)
)
U1 (49)
lambd
R1 (45)
lambd
O1 (41)
lambd
E1 (38)
lambd
;


P (4)
;
)
E1 (38)
lambd
O1 (41)
lambd
R1 (45)
lambd
U1 (49)
lambd
)
Q (26)
lambd
E1 (38)
lambd
O1 (41)
lambd
R1 (45)
lambd
U1 (49)
lambd
)
Q (26)
lambd
E1 (38)
lambd
O1 (41)
lambd
R1 (45)
lambd
U1 (49)
lambd
V1 (54)
lambd
V (50)
id
U (46)
R (42)
O (39)
E (36)
,
Q (25)
E1 (38)
lambd
O1 (41)
lambd
R1 (45)
lambd
U1 (49)
lambd
V (51)
entero
U (46)
R (42)
O (39)