19 de enero de 2026

# Procesador de MyJS: *jsp*

***Memoria del Grupo 59***

Andrés Súnico

## Índice



## 1 Introducción

El desarrollo del procesador *jsp* se ha centrado en la experiencia del usuario (*UX*), priorizando tres aspectos clave: una gestión de errores sólida y clara, una interfaz de línea de comandos (*CLI*) intuitiva, y un rendimiento eficiente.

Por ello, se ha elegido *Rust* como el lenguaje de desarrollo. Ofrece una gestión de memoria eficiente, además de integrar *clap*, una de las mejores bibliotecas para desarollar aplicaciones *CLI*.

Gracias al uso del patrón de *inyección de dependencias* en todo el proyecto, el código fuente es altamente extensible y modular.

# 2 Información Adicional

El código fuente del procesador se puede encontrar en github.com/suuniqo, así como los tests y las dependencias del proyecto.

# 3 Opciones de la Práctica

Además de las opciones comunes a todos los grupos, se han implementado las opciones:

### 3.1 Específicas del grupo

- Comentarios de bloque (/* */)
- Cadenas con comillas dobles (" ")
- Sentencia repetitiva do-while
- Asignación con *y* lógico (*&=*)
- Análisis Sintáctico Ascendente

### 3.2 Adicionales

Para que el procesador esté más completo, se han implementado adicionalmente los operadores:

- Aritméticos: suma ($+$) y multiplicación ($*$)
- Relacionales: menor ($<$) e igual ($==$)
- Lógicos: negación (!) e Y lógico (&&)
- Unarios: más ($+$) y menos ($-$)

Además se ha escogido implementar el tratamiento de secuencias de escape (*\n* y *\t*) y de las *keywords true* y *false*;

# 4 Análisis Léxico

El Analizador Léxico o *Lexer* es uno de los 3 módulos principales del procesador.

Al ser la primera capa de procesamiento, es el encargado de manejar el fichero fuente y convertirlo en una lista de *tokens* para el Analizador Sintáctico.

### 4.1 Tokens

Con el fin de lograr un procesamiento eficiente, tanto en memoria como en complejidad, se han minimizado el número de *tokens* con atributos (tan sólo 4 de los 33 *tokens* usarán un atributo).

Cabe notar, además, que se ha decidido no hacer uso del *token* fin de fichero (*EOF*). Esto es porque el *Lexer* se ha implementado como un iterador de *tokens*, de modo que el final del flujo se detecta naturalmente cuando se consume el iterador.

**Cuadro 1: Listado de *tokens***

| Elemento | Código | Atributo |
|---|---|---|
| boolean | Bool | - |
| do | Do | - |
| float | Float | - |
| function | Func | - |
| if | If | - |
| int | Int | - |
| let | Let | - |
| read | Read | - |
| return | Ret | - |
| string | Str | - |
| void | Void | - |
| while | While | - |
| write | Write | - |
| constante real | FloatLit | Número |
| constante entera | IntLit | Número |
| Cadena | StrLit | Cadena |
| Identificador | Id | Posición |
| &= | AndAssign | - |
| = | Assign | - |
| , | Comma | - |
| ; | Semi | - |
| ( | LParen | - |
| ) | RParen | - |
| { | LBrack | - |
| } | RBrack | - |
| Suma ($+$) | Sum | - |
| Por ($*$) | Mul | - |
| Y lógico (&&) | And | - |
| Negación (!) | Not | - |
| Menor ($<$) | Lt | - |
| Igual ($==$) | Eq | - |
| Menos ($-$) | Sub | - |
| Más ($+$) | Sum | - |
| false | False | - |
| true | True | - |

### 4.2 Errores

Cada tipo de error consta de un mensaje diferente y de una severidad, distinguiéndose *error* de *warning* (que no impediría la compilación del programa).

El *Lexer* puede generar nueve excepciones distintas, siendo todas recuperables. De entre ellas, *Carácter inválido* sirve de *fallback*.

Por aclarar, una cadena malformada es aquella que contiene caracteres *ASCII* no gráficos.

Sólo se emite un *warning*, *Secuencia de Escape inválida*. Como se muestra en Acciones Semánticas, al detectar una secuencia incorrecta no se descartara el *token* cadena, sino que se conserva literalmente (por ejemplo, la secuencia \q, se sustituye por esos dos mismos caracteres)[1].

**Cuadro 2: Listado de errores del *Lexer***

| Error | Severidad |
|---|---|
| Carácter inválido | *error* |
| Comentario inacabado | *error* |
| Cadena inacabada | *error* |
| Cadena malformada | *error* |
| Overflow de Cadena | *error* |
| Overflow de Entero | *error* |
| Overflow de Real | *error* |
| Formato de Real inválido | *error* |
| Secuencia de Escape inválida | *warning* |

### 4.3 Gramática

Se define la gramática del *Lexer* como la tupla $G = (T, N, S, P)$, dónde:

$T = \{\text{Todo carácter } \textit{ASCII}\} \cup \{\textit{EOF}\}$
$N = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O\}$

$P$ se compone de la regla del axioma:

$S \to delS \mid , \mid ; \mid ( \mid ) \mid \{ \mid \} \mid + \mid * \mid \% \mid =A \mid !B \mid <C \mid >D \mid \&E \mid |F \mid dG \mid "H \mid c_1 I \mid /J \mid \textit{EOF}$

Y del resto de reglas:

$A \to = \mid \lambda$
$B \to = \mid \lambda$
$C \to = \mid \lambda$
$D \to = \mid \lambda$
$E \to = \mid \lambda$
$F \to \mid$
$G \to dG \mid .K \mid \lambda$
$K \to dL$
$L \to dL \mid \lambda$
$H \to c_2 H \mid \backslash M \mid "$
$M \to \text{n}H \mid \text{t}H$
$I \to c_3 I \mid \lambda$
$J \to *N \mid \lambda$
$N \to c_4 N \mid *O$
$O \to c_5 N \mid *O \mid /S$

$del := \{\textit{ASCII}\text{ delimitadores}^2\}$
$gra := \{\textit{ASCII}\text{ con código } c : 32 \leq c \leq 126\}$
$d := \{0, 1, \ldots, 9\}$
$l := \{\text{a, b, } \ldots \text{, z, A, B, } \ldots \text{, Z}\}$
$c_1 := l \cup \{\_\}$
$c_2 := gra \setminus \{\backslash, "\}$
$c_3 := c_1 \cup d$
$c_4 := T \setminus \{*, \textit{EOF}\}$
$c_5 := T \setminus \{*, /, \textit{EOF}\}$

El lenguaje generado por esta gramática, $L(G)$, está compuesto por el conjunto de todos los *tokens* válidos del lenguaje de programación *MyJS*. Por tanto, dada una cadena de símbolos terminales, la gramática $G$ es capaz de detectar si forma o no un *token* válido.

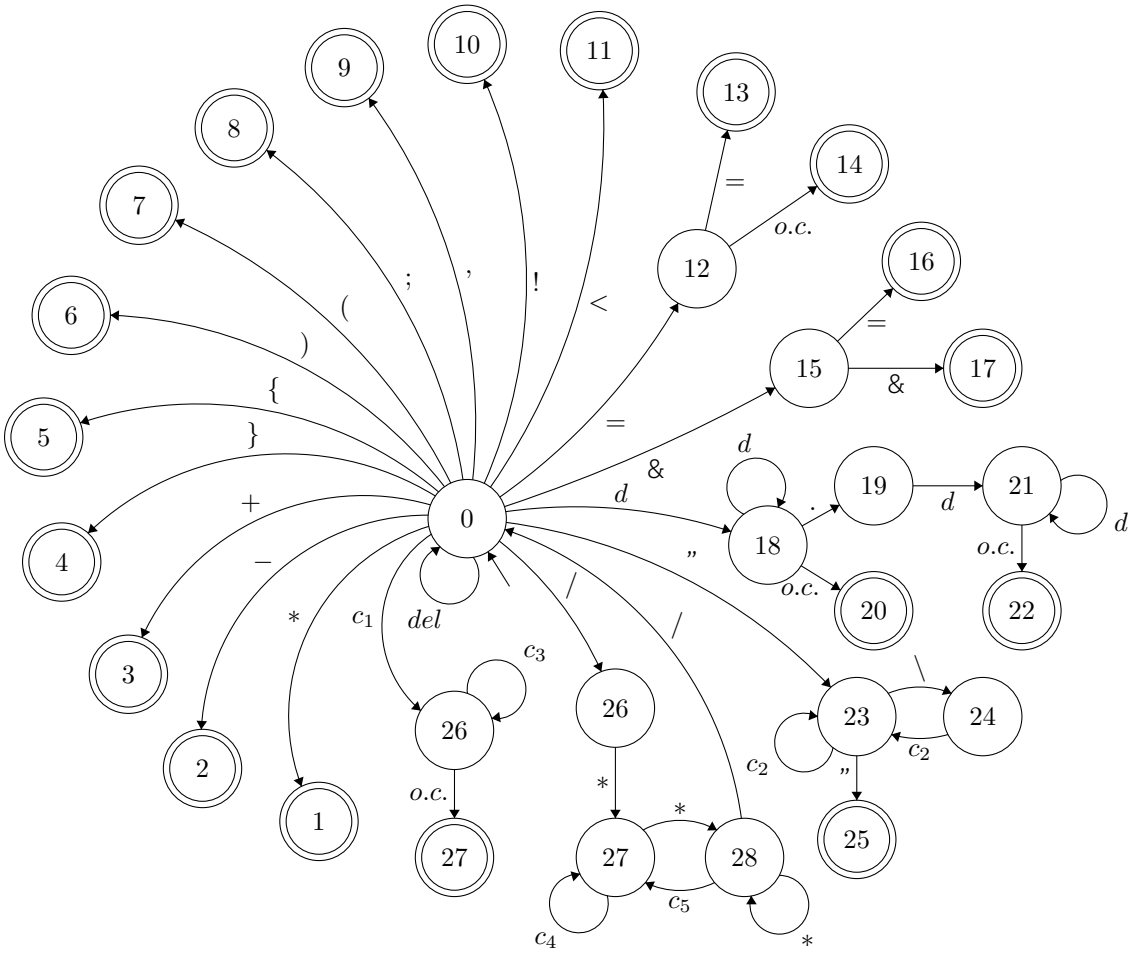
[1] Se ha elegido este comportamiento para que el procesador sea fiel a la documentación oficial de *ECMAScript*.
[2] La definición de delimitador se toma de la documentación oficial de *ECMAScript*.

### 4.4 Autómata

A continuación se muestra el autómata finito determinista o *FDA* que reconoce el lenguaje generado por la gramática $G$. Nótese que una transición "o.c." ocurre al leer un carácter que no corresponda a otra transición del estado.

Se considera un error y se detiene la ejecución cuando el autómata lee un carácter con el que no puede transitar. Solo se alcanza un estado final cuando se ha reconocido un *token* exitosamente.

Como se explica en el siguiente apartado, un autómata no va a ser un modelo suficientemente potente como para representar las operaciones de un *Lexer*. Va a ser necesario complementarlo con el conjunto de Acciones Semánticas detallado en la siguiente sección.




### 4.5 Acciones Semánticas

Las acciones semánticas son operaciones adicionales que se ejecutan durante las transiciones del autómata, con el propósito de aumentar la expresividad cuando es necesario. Resultan especialmente útiles para realizar conversiones de tipos o para simplificar la ejecución de otras acciones más complejas.

Por claridad, se dividen en varios grupos:

#### 4.5.1 General

**READ** Segunda acción de toda transición menos 12:14, 18:20, 21:22, 24:23, 29:30

```
chr := read()
```

#### 4.5.2 Errores

**MALFORMED_STR** Si en el estado $23$ o $24$ se recibe un carácter *ASCII* no gráfico

```
reporter.emit(MalformedStrLit)
```

**UNTERM_STR** Si en el estado $23$ o $24$ se recibe *EOF*

```
reporter.emit(UntermStrLit)
```

**INV_FLOAT_FMT** Si en el estado $19$ no se puede transitar

```
reporter.emit(InvFloatFmt)
```

**UNTERM_COMM** Si en el estado $27$ o $28$ no se puede transitar

```
reporter.emit(UntermComment)
```

**INV_CHAR** Ante cualquier error no manejado en el resto de acciones

```
reporter.emit(StrayChar)
```

#### 4.5.3 Generación Directa

**GEN_MUL** En la transición $0{:}1$

```
gen_token(Mul, -)
```

**GEN_SUB** En la transición $0{:}2$

```
gen_token(Sub, -)
```

**GEN_SUM** En la transición $0{:}3$

```
gen_token(Sum, -)
```

**GEN_RBRACK** En la transición $0{:}4$

```
gen_token(RBrack, -)
```

**GEN_LBRACK** En la transición $0{:}5$

```
gen_token(LBrack, -)
```

**GEN_RPAREN** En la transición $0{:}6$

```
gen_token(RParen, -)
```

**GEN_LPAREN** En la transición $0{:}7$

```
gen_token(LParen, -)
```

**GEN_SEMI** En la transición $0{:}8$

```
gen_token(Semi, -)
```

**GEN_COMMA** En la transición $0{:}9$

```
gen_token(Comma, -)
```

**GEN_NOT** En la transición $0{:}10$

```
gen_token(Not, -)
```

**GEN_LT** En la transición $0{:}11$

```
gen_token(Lt, -)
```

**GEN_EQ** En la transición $12{:}13$

```
gen_token(Eq, -)
```

**GEN_ASSIGN** En la transición $12{:}14$

```
gen_token(Assign, -)
```

**GEN_ANDASSIGN** En la transición $15{:}16$

```
gen_token(AndAssign, -)
```

**GEN_AND** En la transición $15{:}17$

```
gen_token(And, -)
```

#### 4.5.4 Generación de Números

**INIT_NUM** En la transición 0:18

```
num := val(chr)
```

**INIT_DEC** En la transición 20:21

```
dec := 10
num := num + val(chr) / dec
```

**GEN_DEC** En la transición 21:22

```
if (num > 3.4028235e38) {
    reporter.emit(FloatOverflow)
} else {
    gen_token(FloatLit, num)
}
```

**ADD_INTDIG** En la transición 18:18

```
num := num * 10 + val(chr)
```

**ADD_DECDIG** En las transiciones 21:21

```
dec := dec * 10
num := num + vald(chr) / dec
```

**GEN_INT** En la transición 18:20

```
if (num > 32767) {
    reporter.emit(IntOverflow)
} else {
    gen_token(IntLit, num)
}
```

#### 4.5.5 Generación de Cadenas e Identificadores

**INIT_STR** En la transición 0:23

```
lex := ""
len := 0
```

**ADD_CHAR_STR** En la transición 23:23

```
lex.concat(chr)
len := len + 1
```

**ADD_CHAR_ID** En la transición 0:26, 26:26

```
lex.concat(chr)
```

**ADD_ESCSEQ** En la transición 24:23

```
len := len + 1
switch (chr) {
    case 'n' -> lex.concat('\n')
    case 't' -> lex.concat('\t')
    default  -> {
        reporter.warn(InvEscSeq)
        lex.concat('\\')
        lex.concat(chr)
        len := len + 1
    }
}
```

**INIT_ID** En la transición 0:26

```
lex := ""
```

**GEN_STR** En la transición 23:25

```
if (len > 64) {
    reporter.emit(StrOverflow)
} else {
    gen_token(StrLit, lex)
}
```

**GEN_ID** En la transición 26:27

```
code := search_keyword(lex)

if (code != null) {
    gen_token(code, -)
} else {
    pos := symtable_search(lex)

    if (pos == null) {
        pos := symtable_insert(lex)
    }
    gen_token(Id, pos)
}
```

# 5 La Tabla de Símbolos

Se trata de un tipo abstracto de datos encargado de gestionar la información relevante a los identificadores del programa. Todos los módulos del procesador van a necesitar acceder a ella con distintos propósitos por lo que es importante que tanto la inserción como la consulta de datos sea eficiente.

## 5.1 Estructura y Organización

### 5.1.1 Entradas

La información de los identificadores se va a guardar en la tabla de símbolos en forma de entradas. Como en *MyJS* no existen los *arrays* se distinguen únicamente 2 tipos:

**Entrada Básica:** Para todos los tipos básicos, es decir, *int*, *float*, *string* y *bool*.

**Lexema:** Nombre de la variable.
**Tipo:** Tipo de la variable.
**Desplazamiento:** Desplazamiento en memoria relativo a su ámbito.

**Entrada Función:** Para las funciones. Nótese que 'Tipos Argumentos' es un puntero a una lista de tipos.

**Lexema:** Nombre de la función.
**Tipo Retorno:** Tipo que devuelve la función, pudiendo ser *Void*.
**Tipos Argumentos:** Lista de los tipos de los parámetros en orden.
**Etiqueta:** Etiqueta que se usará para navegar a la función en el código ensamblador.

Cada entrada va a tener una estructura de 'llave-valor', dónde el lexema del identificador actúa como llave, y sus atributos (toda su información relevante) como valor. Como cada llave identifica de forma única cada entrada, se puede optimizar el complejidad de acceso e inserción a $O(1)$ usando *hashmaps*.

### 5.1.2 Ámbitos

No siempre se puede acceder a cada variable de un programa. Por ejemplo, desde una función no se puede acceder a una variable local de otra. Por ello, por cada ámbito se va a crear una tabla de símbolos distinta. Además, como *MyJS* es un lenguaje sin anidamiento de funciones, en cada momento habrá como máximo 2 tablas de símbolos activas: la global y, opcionalmente, la de una función.

De esta manera, se puede comprender una tabla de símbolos como una *stack* de ámbitos (es decir, tablas de símbolos locales), dónde el Analizador Semántico será el encargado de apilar y desapilar ámbitos al entrar y salir de funciones respectivamente.

# 6 Análisis Sintáctico

El siguiente gran módulo del procesador es el Analizador Sintáctico o *Parser*. El *Parser* consume los *tokens* del *Lexer* y los utiliza para producir el árbol sintáctico abstracto o *AST*.

El *AST* es una *TAD* que representa la estructura sintáctica del fichero fuente, dónde cada nodo corresponde a una construcción sintáctica del lenguaje.

Para este proyecto, se ha escogido implementar un *Parser* ascendente de tipo $SLR(1) \in LR(1)$. A diferencia de un analizador descendente $LL(1)$, se construye el *AST* desde las hojas hasta la raiz, lo que suele permitir una generación más directa y eficiente del árbol.

### 6.1 Gramática

Se define la gramática del *Parser* como la tupla $G = (T, N, P, R)$, dónde:

$T = \{$Todo *token* definido por el *Lexer*$\}$
$N = \{E, R, RR, U, UU, EE, V, S, L, Q, X, B, T, M, F, FF, F1, F2, F3, H, A, K, C, P\}$

$R$ se compone de:

$P \rightarrow BP \mid FP \mid \lambda$
$C \rightarrow BC \mid \lambda$
$F \rightarrow$ Func $FF$ LBrack $C$ RBrack
$FF \rightarrow F1\ F2\ F3$
$F1 \rightarrow H$
$F2 \rightarrow$ Id
$F3 \rightarrow$ LParen $A$ RParen
$H \rightarrow T \mid$ Void
$A \rightarrow T$ Id $K \mid$ Void
$K \rightarrow$ Comma $T$ Id $K \mid \lambda$
$B \rightarrow$ If LParen $E$ RParen $S \mid$ Do LBrack $C$ RBrack While LParen $E$ RParen Semi
$B \rightarrow S \mid$ Let $MT$ Id Semi $\mid$ Let $MT$ Id Assign $E$ Semi
$M \rightarrow \lambda$
$T \rightarrow$ Int $\mid$ Float $\mid$ Bool $\mid$ Str
$S \rightarrow$ Write $E$ Semi $\mid$ Read Id Semi $\mid$ Ret $X$ Semi
$S \rightarrow$ Id Assign $E$ Semi $\mid$ Id AndAssign $E$ Semi $\mid$ Id LParen $L$ RParen Semi
$L \rightarrow EQ \mid \lambda$
$Q \rightarrow$ Comma $EQ \mid \lambda$
$X \rightarrow E \mid \lambda$
$E \rightarrow E$ And $R \mid R$
$R \rightarrow R$ Eq $RR \mid RR$
$RR \rightarrow RR$ Lt $U \mid U$
$U \rightarrow U$ Sum $UU \mid UU$
$UU \rightarrow UU$ Mul $EE \mid EE$
$EE \rightarrow$ Not $EE \mid$ Sub $EE \mid$ Sum $EE \mid V$
$V \rightarrow$ Id LParen $L$ RParen $\mid$ LParen $E$ RParep $\mid$ IntLit $\mid$ FloatLit $\mid$ StrLit $\mid$ True $\mid$ False $\mid$ Id

$G$ se trata de una gramática de contexto libre y el lenguaje que genera, $L(G)$, está compuesto por la estructura de todos los programas sintácticamente correctos.

De este modo, un fichero fuente sintácticamente correcto se puede interpretar como una palabra $w \in L(G)$ y visto así, el *AST* se trata justamente del árbol de derivación de $w$, por lo que es fácil de obtener a partir de la secuencia de reglas aplicadas por el *Parser* para reducir $w$ al axioma $S$. Está secuencia de reglas se llama el *parse* de un programa, y los próximos apartados se centran en como hallarlo.

### 6.2 Autómata

$G$ se trata de una gramática de contexto libre, por lo su lenguaje no puede reconocerse directamente con un *AFD*. Sin embargo, si se aumenta su gramática a $G' = (T, N, P', R')$, con $R' = R \cup \{P' \rightarrow P\}$, y se verifica que $G' \in SLR(1)$, entonces la colección completa de conjuntos de ítems $LR(0)$ de $G'$ sí que representa un *AFD*: el que reconoce todos los prefijos viables de $G$. Nótese que $L(G) = L(G')$ por lo que son equivalentes.

Los estados de dicho autómata representan los estados intermedios de análisis tras consumir una secuencia de símbolos potencialmente correcta, y se construye mediante las operaciones de *closure* y de *goto*.

## 6.3 Tablas de Acción y Goto

El autómata se puede codificar mediante las tablas de acción y *goto*, que si se pueden construir sin conflictos, garantizan que $G' \in SLR(1)$, es decir, que el analizador puede utilizar estas tablas para reconocer cualquier cadena de $L(G)$ de forma determinista y generar su *parse*.

**Cuadro 3: Tabla de Acción**

| | if | do | while | int | float | str | bool | void | let | func | ret | read | write | true | false | FloatLit | IntLit | StrLit | Id | = | &= | , | ; | ( | ) | { | } | + | - | * | && | ! | < | == | $ |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | s7 | s5 | | | | | | | s6 | s4 | s9 | s10 | s11 | | | | | | s12 | | | | | | | | | | | | | | | | r1 |
| **1** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | acc |
| **2** | s7 | s5 | | | | | | | s6 | s4 | s9 | s10 | s11 | | | | | | s12 | | | | | | | | | | | | | | | | r1 |
| **3** | s7 | s5 | | | | | | | s6 | s4 | s9 | s10 | s11 | | | | | | s12 | | | | | | | | | | | | | | | | r1 |
| **4** | | | | s22 | s21 | s19 | s20 | s15 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **5** | | | | | | | | | | | | | | | | | | | | | | | | | | s24 | | | | | | | | |
| **6** | | | | r18 | r18 | r18 | r18 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **7** | | | | | | | | | | | | | | | | | | | | | | | | s26 | | | | | | | | | | |
| **8** | r27 | r27 | | | | | | | r27 | r27 | r27 | r27 | r27 | | | | | | r27 | | | | | | | r27 | | | | | | | | | r27 |
| **9** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | r16 | s35 | | | | s43 | s44 | | | s37 | | | |
| **10** | | | | | | | | | | | | | | | | | | | s45 | | | | | | | | | | | | | | | | |
| **11** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **12** | | | | | | | | | | | | | | | | | | | | s49 | s48 | | | s47 | | | | | | | | | | | |
| **13** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | r2 |
| **14** | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | r3 |
| **15** | | | | | | | | | | | | | | | | | | | r10 | | | | | | | | | | | | | | | | |
| **16** | | | | | | | | | | | | | | | | | | | r11 | | | | | | | | | | | | | | | | |
| **17** | | | | | | | | | | | | | | | | | | | r14 | | | | | | | | | | | | | | | | |
| **18** | | | | | | | | | | | | | | | | | | | | | | | | | | s50 | | | | | | | | | |
| **19** | | | | | | | | | | | | | | | | | | | r19 | | | | | | | | | | | | | | | | |
| **20** | | | | | | | | | | | | | | | | | | | r20 | | | | | | | | | | | | | | | | |
| **21** | | | | | | | | | | | | | | | | | | | r21 | | | | | | | | | | | | | | | | |
| **22** | | | | | | | | | | | | | | | | | | | r22 | | | | | | | | | | | | | | | | |
| **23** | | | | | | | | | | | | | | | | | | | s51 | | | | | | | | | | | | | | | | |
| **24** | s7 | s5 | | | | | | | s6 | | s9 | s10 | s11 | | | | | | s12 | | | | | | | r4 | | | | | | | | | |
| **25** | | | | s22 | s21 | s19 | s20 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **26** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **27** | | | | | | | | | | | | | | | | | | | | | | | r17 | | | | | | | | s57 | | | | |
| **28** | | | | | | | | | | | | | | | | | | | | | | | s58 | | | | | | | | | | | | |
| **29** | | | | | | | | | | | | | | | | | | | | | | r38 | r38 | s59 | r38 | | | r38 | | r38 | r38 | | r38 | r38 | |
| **30** | | | | | | | | | | | | | | | | | | | | | | r39 | r39 | | r39 | | | r39 | | r39 | r39 | | r39 | r39 | |
| **31** | | | | | | | | | | | | | | | | | | | | | | r40 | r40 | | r40 | | | r40 | | r40 | r40 | | r40 | r40 | |
| **32** | | | | | | | | | | | | | | | | | | | | | | r41 | r41 | | r41 | | | r41 | | r41 | r41 | | r41 | r41 | |
| **33** | | | | | | | | | | | | | | | | | | | | | | r42 | r42 | | r42 | | | r42 | | r42 | r42 | | r42 | r42 | |
| **34** | | | | | | | | | | | | | | | | | | | | | | r43 | r43 | | r43 | | | r43 | | r43 | r43 | | r43 | r43 | |
| **35** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **36** | | | | | | | | | | | | | | | | | | | | | | r46 | r46 | | r46 | | | r46 | | r46 | r46 | | r46 | r46 | |
| **37** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **38** | | | | | | | | | | | | | | | | | | | | | | r48 | r48 | | r48 | | | r48 | | r48 | r48 | | r48 | r48 | |
| **39** | | | | | | | | | | | | | | | | | | | | | | r50 | r50 | | r50 | | | r50 | | s62 | r50 | | r50 | r50 | |
| **40** | | | | | | | | | | | | | | | | | | | | | | r52 | r52 | | r52 | | | s63 | | | r52 | | r52 | r52 | |
| **41** | | | | | | | | | | | | | | | | | | | | | | r54 | r54 | | r54 | | | | | | r54 | | s64 | r54 | |
| **42** | | | | | | | | | | | | | | | | | | | | | | r56 | r56 | | r56 | | | | | | r56 | | | s65 | |
| **43** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **44** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **45** | | | | | | | | | | | | | | | | | | | | | | | s68 | | | | | | | | | | | | |
| **46** | | | | | | | | | | | | | | | | | | | | | | | s69 | | | | | | | | s57 | | | | |
| **47** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | r30 | | | s43 | s44 | | | s37 | | | |
| **48** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **49** | | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **50** | s7 | s5 | | | | | | | s6 | | s9 | s10 | s11 | | | | | | s12 | | | | | | | r4 | | | | | | | | | |
| **51** | | | | | | | | | | | | | | | | | | | | | | | | r13 | | | | | | | | | | | |
| **52** | | | | | | | | | | | | | | | | | | | | | | | | s75 | | | | | | | | | | | |
| **53** | s7 | s5 | | | | | | | s6 | | s9 | s10 | s11 | | | | | | s12 | | | | | | | r4 | | | | | | | | | |
| **54** | | | | | | | | | | | | | | | | | | | | | | | | | | s78 | | | | | | | | | |
| **55** | | | | | | | | | | | | | | | | | | | s79 | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **56** | | | | | | | | | | | | | | | | | | | | | | | | | s80 | | | | | | s57 | | | | |
| **57** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **58** | r32 | r32 | | | | | | | r32 | r32 | r32 | r32 | r32 | | | | | | r32 | | | | | | | | r32 | | | | | | | | r32 |
| **59** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | r30 | | | s43 | s44 | | | s37 | | | |
| **60** | | | | | | | | | | | | | | | | | | | | | | | | | s83 | | | | | | s57 | | | | |
| **61** | | | | | | | | | | | | | | | | | | | | | r47 | r47 | | r47 | | | | r47 | | r47 | r47 | | r47 | r47 | |
| **62** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **63** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **64** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **65** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **66** | | | | | | | | | | | | | | | | | | | | | r59 | r59 | | r59 | | | | r59 | | r59 | r59 | | r59 | r59 | |
| **67** | | | | | | | | | | | | | | | | | | | | | r60 | r60 | | r60 | | | | r60 | | r60 | r60 | | r60 | r60 | |
| **68** | r33 | r33 | | | | | | | r33 | r33 | r33 | r33 | r33 | | | | | r33 | | | | | | | | r33 | | | | | | | | r33 |
| **69** | r34 | r34 | | | | | | | r34 | r34 | r34 | r34 | r34 | | | | | r34 | | | | | | | | r34 | | | | | | | | r34 |
| **70** | | | | | | | | | | | | | | | | | | | | | s88 | | | r28 | | | | | | s57 | | | | |
| **71** | | | | | | | | | | | | | | | | | | | | | | | | s90 | | | | | | | | | | |
| **72** | | | | | | | | | | | | | | | | | | | | | | s91 | | | | | | | | s57 | | | | |
| **73** | | | | | | | | | | | | | | | | | | | | | | s92 | | | | | | | | s57 | | | | |
| **74** | | | | | | | | | | | | | | | | | | | | | | | | | | r93 | | | | | | | | |
| **75** | | | | s22 | s21 | s19 | s20 | s94 | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **76** | | | | | | | | | | | | | | | | | | | | | | | | | r58 | | | | | | | | |
| **77** | | | | | | | | | | | | | | | | | | | | | | | | | | r5 | | | | | | | | |
| **78** | | | s97 | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **79** | | | | | | | | | | | | | | | | | | | s98 | | | s99 | | | | | | | | | | | | |
| **80** | | | | | | | | | | | s9 | s10 | s11 | | | | | s12 | | | | | | | | | | | | | | | | |
| **81** | | | | | | | | | | | | | | | | | | | | | r57 | r57 | | r57 | | | | | | r57 | | | s65 | |
| **82** | | | | | | | | | | | | | | | | | | | | | | | | s101 | | | | | | | | | | |
| **83** | | | | | | | | | | | | | | | | | | | | | r44 | r44 | | r44 | | | | r44 | | r44 | r44 | | r44 | r44 | |
| **84** | | | | | | | | | | | | | | | | | | | | | r49 | r49 | | r49 | | | | r49 | | r49 | r49 | | r49 | r49 | |
| **85** | | | | | | | | | | | | | | | | | | | | | r51 | r51 | | r51 | | | | r51 | | s62 | r51 | | r51 | r51 | |
| **86** | | | | | | | | | | | | | | | | | | | | | r53 | r53 | | r53 | | | | s63 | | | r53 | | r53 | r53 | |
| **87** | | | | | | | | | | | | | | | | | | | | | r55 | r55 | | r55 | | | | | | r55 | | s64 | r55 | |
| **88** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **89** | | | | | | | | | | | | | | | | | | | | | | | | r31 | | | | | | | | | | |
| **90** | | | | | | | | | | | | | | | | | | | | | | s103 | | | | | | | | | | | | |
| **91** | r36 | r36 | | | | | | | r36 | r36 | r36 | r36 | r36 | | | | | r36 | | | | | | | | r36 | | | | | | | | r36 |
| **92** | r37 | r37 | | | | | | | r37 | r37 | r37 | r37 | r37 | | | | | r37 | | | | | | | | r37 | | | | | | | | r37 |
| **93** | r15 | r15 | | | | | | | r15 | r15 | r15 | r15 | r15 | | | | | r15 | | | | | | | | | | | | | | | | r15 |
| **94** | | | | | | | | | | | | | | | | | | | | | | | | r8 | | | | | | | | | | |
| **95** | | | | | | | | | | | | | | | | | | s104 | | | | | | | | | | | | | | | | |
| **96** | | | | | | | | | | | | | | | | | | | | | | | | s105 | | | | | | | | | | |
| **97** | | | | | | | | | | | | | | | | | | | | | | | s106 | | | | | | | | | | | |
| **98** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **99** | r25 | r25 | | | | | | | r25 | r25 | r25 | r25 | r25 | | | | | r25 | | | | | | | | r25 | | | | | | | | r25 |
| **100** | r26 | r26 | | | | | | | r26 | r26 | r26 | r26 | r26 | | | | | r26 | | | | | | | | r26 | | | | | | | | r26 |
| **101** | | | | | | | | | | | | | | | | | | | | | r45 | r45 | | r45 | | | | r45 | | r45 | r45 | | r45 | r45 | |
| **102** | | | | | | | | | | | | | | | | | | | | | s88 | | | r28 | | | | | | s57 | | | | |
| **103** | r35 | r35 | | | | | | | r35 | r35 | r35 | r35 | r35 | | | | | r35 | | | | | | | | r35 | | | | | | | | r35 |
| **104** | | | | | | | | | | | | | | | | | | | | | s109 | | | r6 | | | | | | | | | | |
| **105** | | | | | | | | | | | | | | | | | | | | | | | | | r12 | | | | | | | | | |
| **106** | | | | | | | | | | | | | s31 | s30 | s33 | s34 | s32 | s29 | | | | | | s35 | | | | s43 | s44 | | | s37 | | | |
| **107** | | | | | | | | | | | | | | | | | | | | | | s112 | | | | | | | | s57 | | | | |
| **108** | | | | | | | | | | | | | | | | | | | | | | | | r29 | | | | | | | | | | |
| **109** | | | | s22 | s21 | s19 | s20 | | | | | | | | | | | | | | | | | | | | | | | | | | | |
| **110** | | | | | | | | | | | | | | | | | | | | | | | | r9 | | | | | | | | | | |
| **111** | | | | | | | | | | | | | | | | | | | | | | | | s114 | | | | | | s57 | | | | |
| **112** | r24 | r24 | | | | | | | r24 | r24 | r24 | r24 | r24 | | | | | r24 | | | | | | | | r24 | | | | | | | | r24 |
| **113** | | | | | | | | | | | | | | | | | | s115 | | | | | | | | | | | | | | | | |
| **114** | | | | | | | | | | | | | | | | | | | | | | s116 | | | | | | | | | | | | |
| **115** | | | | | | | | | | | | | | | | | | | | | s109 | | | r6 | | | | | | | | | | |
| **116** | r23 | r23 | | | | | | | r23 | r23 | r23 | r23 | r23 | | | | | r23 | | | | | | | | r23 | | | | | | | | r23 |
| **117** | | | | | | | | | | | | | | | | | | | | | | | | r7 | | | | | | | | | | |

**Cuadro 4: Tabla de Goto**

| | E | R | RR | U | UU | EE | V | S | L | Q | X | B | T | M | F | FF | F1 | F2 | F3 | H | A | K | C | P |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **0** | | | | | | | | 8 | | | | 3 | | | 2 | | | | | | | | | 1 |
| **1** | | | | | | | | | | | | | | | | | | | | | | | | |
| **2** | | | | | | | | 8 | | | | 3 | | | 2 | | | | | | | | | 13 |
| **3** | | | | | | | | 8 | | | | 3 | | | 2 | | | | | | | | | 14 |
| **4** | | | | | | | | | | | | | 16 | | | 18 | 23 | | | 17 | | | | |
| **5** | | | | | | | | | | | | | | | | | | | | | | | | |
| **6** | | | | | | | | | | | | | | 25 | | | | | | | | | | |
| **7** | | | | | | | | | | | | | | | | | | | | | | | | |
| **8** | | | | | | | | | | | | | | | | | | | | | | | | |
| **9** | 27 | 42 | 41 | 40 | 39 | 38 | 36 | | | | 28 | | | | | | | | | | | | | |
| **10** | | | | | | | | | | | | | | | | | | | | | | | | |
| **11** | 46 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **12** | | | | | | | | | | | | | | | | | | | | | | | | |
| **13** | | | | | | | | | | | | | | | | | | | | | | | | |
| **14** | | | | | | | | | | | | | | | | | | | | | | | | |
| **15** | | | | | | | | | | | | | | | | | | | | | | | | |
| **16** | | | | | | | | | | | | | | | | | | | | | | | | |
| **17** | | | | | | | | | | | | | | | | | | | | | | | | |
| **18** | | | | | | | | | | | | | | | | | | | | | | | | |
| **19** | | | | | | | | | | | | | | | | | | | | | | | | |
| **20** | | | | | | | | | | | | | | | | | | | | | | | | |
| **21** | | | | | | | | | | | | | | | | | | | | | | | | |
| **22** | | | | | | | | | | | | | | | | | | | | | | | | |
| **23** | | | | | | | | | | | | | | | | | | 52 | | | | | | |
| **24** | | | | | | | | 8 | | | | 53 | | | | | | | | | | | 54 | |
| **25** | | | | | | | | | | | | | 55 | | | | | | | | | | | |
| **26** | 56 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **27** | | | | | | | | | | | | | | | | | | | | | | | | |
| **28** | | | | | | | | | | | | | | | | | | | | | | | | |
| **29** | | | | | | | | | | | | | | | | | | | | | | | | |
| **30** | | | | | | | | | | | | | | | | | | | | | | | | |
| **31** | | | | | | | | | | | | | | | | | | | | | | | | |
| **32** | | | | | | | | | | | | | | | | | | | | | | | | |
| **33** | | | | | | | | | | | | | | | | | | | | | | | | |
| **34** | | | | | | | | | | | | | | | | | | | | | | | | |
| **35** | 60 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **36** | | | | | | | | | | | | | | | | | | | | | | | | |
| **37** | | | | | | 61 | 36 | | | | | | | | | | | | | | | | | |
| **38** | | | | | | | | | | | | | | | | | | | | | | | | |
| **39** | | | | | | | | | | | | | | | | | | | | | | | | |
| **40** | | | | | | | | | | | | | | | | | | | | | | | | |
| **41** | | | | | | | | | | | | | | | | | | | | | | | | |
| **42** | | | | | | | | | | | | | | | | | | | | | | | | |
| **43** | | | | | | 66 | 36 | | | | | | | | | | | | | | | | | |
| **44** | | | | | | 67 | 36 | | | | | | | | | | | | | | | | | |
| **45** | | | | | | | | | | | | | | | | | | | | | | | | |
| **46** | | | | | | | | | | | | | | | | | | | | | | | | |
| **47** | 70 | 42 | 41 | 40 | 39 | 38 | 36 | | 71 | | | | | | | | | | | | | | | |
| **48** | 72 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **49** | 73 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **50** | | | | | | | | 8 | | | | 53 | | | | | | | | | | | 74 | |
| **51** | | | | | | | | | | | | | | | | | | | | | | | | |
| **52** | | | | | | | | | | | | | | | | | | | 76 | | | | | |
| **53** | | | | | | | | 8 | | | | 53 | | | | | | | | | | | 77 | |
| **54** | | | | | | | | | | | | | | | | | | | | | | | | |
| **55** | | | | | | | | | | | | | | | | | | | | | | | | |
| **56** | | | | | | | | | | | | | | | | | | | | | | | | |
| **57** | | 81 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **58** | | | | | | | | | | | | | | | | | | | | | | | | |

| | | | | | | | | | | | | | | | | | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| **59** | 70 | 42 | 41 | 40 | 39 | 38 | 36 | | 82 | | | | | | | | | | | | | | | |
| **60** | | | | | | | | | | | | | | | | | | | | | | | | |
| **61** | | | | | | | | | | | | | | | | | | | | | | | | |
| **62** | | | | | | 84 | 36 | | | | | | | | | | | | | | | | | |
| **63** | | | | | 85 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **64** | | | | 86 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **65** | | | 87 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **66** | | | | | | | | | | | | | | | | | | | | | | | | |
| **67** | | | | | | | | | | | | | | | | | | | | | | | | |
| **68** | | | | | | | | | | | | | | | | | | | | | | | | |
| **69** | | | | | | | | | | | | | | | | | | | | | | | | |
| **70** | | | | | | | | | | 89 | | | | | | | | | | | | | | |
| **71** | | | | | | | | | | | | | | | | | | | | | | | | |
| **72** | | | | | | | | | | | | | | | | | | | | | | | | |
| **73** | | | | | | | | | | | | | | | | | | | | | | | | |
| **74** | | | | | | | | | | | | | | | | | | | | | | | | |
| **75** | | | | | | | | | | | | | 95 | | | | | | | | 96 | | | |
| **76** | | | | | | | | | | | | | | | | | | | | | | | | |
| **77** | | | | | | | | | | | | | | | | | | | | | | | | |
| **78** | | | | | | | | | | | | | | | | | | | | | | | | |
| **79** | | | | | | | | | | | | | | | | | | | | | | | | |
| **80** | | | | | | | | 100 | | | | | | | | | | | | | | | | |
| **81** | | | | | | | | | | | | | | | | | | | | | | | | |
| **82** | | | | | | | | | | | | | | | | | | | | | | | | |
| **83** | | | | | | | | | | | | | | | | | | | | | | | | |
| **84** | | | | | | | | | | | | | | | | | | | | | | | | |
| **85** | | | | | | | | | | | | | | | | | | | | | | | | |
| **86** | | | | | | | | | | | | | | | | | | | | | | | | |
| **87** | | | | | | | | | | | | | | | | | | | | | | | | |
| **88** | 102 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **89** | | | | | | | | | | | | | | | | | | | | | | | | |
| **90** | | | | | | | | | | | | | | | | | | | | | | | | |
| **91** | | | | | | | | | | | | | | | | | | | | | | | | |
| **92** | | | | | | | | | | | | | | | | | | | | | | | | |
| **93** | | | | | | | | | | | | | | | | | | | | | | | | |
| **94** | | | | | | | | | | | | | | | | | | | | | | | | |
| **95** | | | | | | | | | | | | | | | | | | | | | | | | |
| **96** | | | | | | | | | | | | | | | | | | | | | | | | |
| **97** | | | | | | | | | | | | | | | | | | | | | | | | |
| **98** | 107 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **99** | | | | | | | | | | | | | | | | | | | | | | | | |
| **100** | | | | | | | | | | | | | | | | | | | | | | | | |
| **101** | | | | | | | | | | | | | | | | | | | | | | | | |
| **102** | | | | | | | | | | 108 | | | | | | | | | | | | | | |
| **103** | | | | | | | | | | | | | | | | | | | | | | | | |
| **104** | | | | | | | | | | | | | | | | | | | | | | 110 | | |
| **105** | | | | | | | | | | | | | | | | | | | | | | | | |
| **106** | 111 | 42 | 41 | 40 | 39 | 38 | 36 | | | | | | | | | | | | | | | | | |
| **107** | | | | | | | | | | | | | | | | | | | | | | | | |
| **108** | | | | | | | | | | | | | | | | | | | | | | | | |
| **109** | | | | | | | | | | | | | 113 | | | | | | | | | | | |
| **110** | | | | | | | | | | | | | | | | | | | | | | | | |
| **111** | | | | | | | | | | | | | | | | | | | | | | | | |
| **112** | | | | | | | | | | | | | | | | | | | | | | | | |
| **113** | | | | | | | | | | | | | | | | | | | | | | | | |
| **114** | | | | | | | | | | | | | | | | | | | | | | | | |
| **115** | | | | | | | | | | | | | | | | | | | | | | 117 | | |
| **116** | | | | | | | | | | | | | | | | | | | | | | | | |
| **117** | | | | | | | | | | | | | | | | | | | | | | | | |

### 6.4 Algoritmo

El recorrido de las tablas se realizará con un *stack* dónde se irán apilando los símbolos no terminales hasta ser reducidos al axioma. Además, cada vez que se efectúe una reducción, el índice de la regla utilizada se añadirá al *parse*, quedando este completamente formado al terminar el procedimiento.

En la tabla acción las celdas quieren decir:

- $sN$ : desplazar y apilar el estado n-ésimo
- $rN$ : reducir por la regla n-ésima
- $acc$ : aceptar la cadena

En la tabla goto:

- $N$ : apilar el estado n-ésimo

En ambas tablas una celda en blanco indica un error sintáctico.

### 6.5 Conflictos

Hay dos tipos de conflictos, conflictos recucción-reducción y conflictos desplazamiento-reducción. Ambos ocurren cuando se intenta escribir una acción en una celda no vacía de la tabla acción.

Como no han habido colisiones durante la construcción de la tabla, se confirma que $G' \in SLR(1)$. Por tanto, se va a poder implementar el *Parser* $LR(1)$ exitosamente con la gramática escogida.

### 6.6 Errores

El *Parser* emite 10 excepciones diferentes, dónde *Token inesperado* sirve de *fallback*.

Cabe destacar que *Token inesperado*, a pesar de ser la excepción genérica del *Parser*, es dinámica, es decir, su mensaje varía según la celda en la que se lance, tomando el formato:

*"expected <opciones> before <token inesperado>"*

La lista de *tokens* esperados se obtiene realizando una busqueda en anchura en las tablas por cada *token* con una celda no vacía en la fila del fallo.

Los *tokens* que desplazan siempre serán válidos, pero hay casos en los que un *token* reduce una o varias veces pero acaba terminando en error, por lo que es necesario simular cada uno de ellos hasta llegar a un desplazamiento o a un error.

El *Parser* también es capaz de recuperarse de cualquier error, aunque este proceso es más complejo que en el *Lexer*. En la sección del Gestor de Errores se explicará en mayor detalle pero, a alto nivel, al llegar a una celda de error se intentará resincronizar la pila de estados a través de inserciones, sustituciones o eliminaciones.

No obstante, aunque en la práctica nunca lo hará, la resincronización puede llegar a fallar, en cuyo caso se termina el análisis. De hecho, el *Parser* es el único módulo que puede interrumpir la ejecución del programa prematuramente, es decir, antes de procesar el fichero fuente por completo.

**Cuadro 5: Listado de errores del *Parser***

| Error | Severidad |
|---|---|
| Token inesperado | *error* |
| Delimitador desparejado | *error* |
| Delimitador sin cerrar | *error* |
| Palabra reservadada usada como identificador | *error* |
| Punto y coma ausente | *error* |
| Coma sobrante en la lista de parámetros de una función | *error* |
| Coma sobrante en la lista de argumentos en una llamada | *error* |
| Tipo ausente en declaración | *error* |
| Tipo de retorno ausente | *error* |
| Lista de parámetros ausente | *error* |
| Lista de parámetros vacía | *error* |

# 7 Análisis Semántico

El último módulo del procesador es el Analizador Semántico, que se encarga de asegurar que el programa sea coherente más allá de las reglas sintácticas. Hace comprobaciones de tipo, se asegura de que no hay redeclaraciones, comprueba que se llama a las funciones con el número correcto de parámetros, etc.

En realidad, en este procesador, el Analizador Semántico es un submódulo del *Parser*, ya que sigue la estrategia de *Traducción dirigida por la sintaxis* o *TDS*. Esta consiste en asociar una serie de acciones semánticas a cada regla de la gramática y ciertos atributos a algunos de sus símbolos, construyendo lo que se llama una *Gramática de Atributos*.

El producto final del Analizador Semántico será la Tabla de Símbolos, que irá rellenando con las variables y funciones que encuentre, así como sus respectivos atributos, a medida que se analice el fichero fuente.

## 7.1 Errores

El *Analizador Semántico* puede producir 6 excepciones distintas.

La excepción *Tipo inesperado* sirve la función de error de tipo genérico, que se emitirá siempre que se espere un tipo concreto y se encuentre otro. De nuevo, su mensaje es dinámico y cambia según el tipo esperado y el encontrado.

Como en los módulos anteriores, todas los errores son recuperables. Esta recuperación se hace a través de los tipos especiales *type_error* y *type_ok*, que se propagan a través del *AST*.

**Cuadro 6: Listado de errores del *Analizador Semántico***

| Error | Severidad |
|---|---|
| Tipo de retorno incorrecto | *error* |
| Identificador redeclarado | *error* |
| Tipo inesperado | *error* |
| Función sin declarar | *error* |
| Retorno fuera de función | *error* |
| LLamada incorrecta | *error* |

## 7.2 Acciones Semánticas

Como el *Parser* de este procesador es de tipo $LR(1)$, es decir, construye el *AST* desde las hojas hasta la raiz, las acciones semánticas se ejecutarán cada vez que el *Parser* realice una reducción. De este modo cada regla de la gramática tendrá una única regla asociada a ella.

A continuación se muestran las acciones semánticas del Analizador Semántico. Cabe destacar que esto es una simplificación, ya que muchas acciones, en particular aquellas relacionadas con la gestión de errores, son mucho más complejas en realidad.

En cuanto a la notación se ha optado por un *Esquema de Traducción* (*EdT*). Por motivos de legibilidad, las acciones no se escriben intercaladas en la producción, sino como un bloque de código inmediatamente posterior.

Esto supondría un problema de no ser porque en este trabajo todas las acciones semánticas se ejecutan al reducir, y ninguna producción contiene acciones intercaladas. Por tanto toda producción se ajusta al siguiente esquema:

$$X \to \alpha \, \{\text{acción}\}$$

Y, en consecuencia, cada bloque de código que aparece tras una producción debe interpretarse de este modo, es decir, como una acción sintetizada al final de la producción.

La primera acción que se ejecuta es la creación de la tabla de símbolos. Esta es la única acción que no se puede asociar a ninguna regla, ya que por ser el *Parser* ascendente, no se sabe cual será la primera regla en reducirse:

```
1 symtable = symtable_make()
2
3 scope_global = scope_make()
4 scope_global.despl = 0
5
6 symtable.scopes.push(scope_global)
```

El resto de acciones serán:

1. $PP \to P$

```
symtable_free(symtable)
```

2. $P \to BP$

```
P.type = if B.ret_type == null
    then type_ok
else type_error
```

3. $P \to FP$
4. $P \to \lambda$
5. $C \to BC_1$

```
C.ret_type = if B.ret_type == C1.ret_type
    then B.ret_type
else if B.ret_type  == null
    then C1.ret_type
else if C1.ret_type == null
    then B.ret_type
else type_error
```

6. $C \to \lambda$

```
B.ret_type = null
```

7. $F \to$ Func $FF$ LBrack $C$ RBrack

```
scope_local = symtable.scopes.pop()
scope_free(scope_local)

F.type = if C.ret_type == null
    || (C.ret_type != type_error && C.ret_type == FF.ret_type)
        then type_ok
else type_error
```

8. $FF \to F1\,F2\,F3$

```
scope_local = scope_make()
scope_local.despl = 0

scope_local.add_type(F2.pos, F3.type -> F1.type)
scope_local.add_label(F2.pos, label_make())

if F3.params != null {
    for (type, pos) in (F3.type, F3.params) {
        scope_local.add_type(pos, type)
        scope_local.add_despl(pos, scope_local.despl)

        scope_local.despl += type.size
    }
}

symtable.scopes.push(scope_local)
```

9. $F1 \rightarrow H$

```
F1.type = H.type
```

10. $F2 \rightarrow$ Id

```
F2.pos = Id.pos
```

11. $F3 \rightarrow$ LParen $A$ RParen

```
F3.type = A.type
F3.params = A.params
```

12. $H \rightarrow T$

```
H.type = T.type
```

13. $H \rightarrow$ Void

```
H.type = type_void
```

14. $A \rightarrow T$ Id $K$

```
A.type = T.type x K.type
A.params = Id.pos x K.params
```

15. $A \rightarrow$ Void

```
A.type = type_void
A.params = null
```

16. $K \rightarrow$ Comma $T$ Id $K_1$

```
K.type = T.type x K1.type
K.params = Id.pos x K1.params
```

17. $K \rightarrow \lambda$
18. $B \rightarrow$ If LParen $E$ RParen $S$

```
B.ret_type = S.ret_type

B.type = if E.type == type_bool
    then type_ok
else type_error
```

19. $B \rightarrow$ Do LBrack $C$ RBrack While LParen $E$ RParen Semi

```
B.ret_type = C.ret_type

B.type = if E.type == type_bool
    then type_ok
else type_error
```

20. $B \rightarrow S$

```
B.type = S.type
B.ret_type = S.ret_type
```

21. $B \rightarrow$ Let $MT$ Id Semi

```
scope = symtable.scopes.peek()

scope.add_type(Id.pos, T.type)
scope.add_despl(Id.pos, T.type.size)

scope.despl += T.type.size
```

22. $B \rightarrow$ Let $MT$ Id Assign $E$ Semi

```
scope = symtable.scopes.peek()

scope.add_type(Id.pos, T.type)
scope.add_despl(Id.pos, T.type.size)

scope.despl += T.type.size

B.type = if E.type == T.type
    then type_ok
else type_error
```

23. $M \rightarrow \lambda$
24. $T \rightarrow$ Int

```
T.type = type_int
T.type.size = 1
```

25. $T \rightarrow$ Float

```
T.type = type_float
T.type.size = 2
```

26. $T \rightarrow$ Bool

```
T.type = type_bool
T.type.size = 1
```

27. $T \rightarrow$ Str

```
T.type = type_str
T.type.size = 64
```

28. $S \rightarrow$ Write $E$ Semi

```
S.type = if E.type in {type_int, type_float, type_str}
    then type_ok
else type_error
```

29. $S \rightarrow$ Read Id Semi

```
scope = symtable.scopes.peek()
type = scope.search_type(Id.pos)

if type == null {
    scope.add_type(Id.pos, type_int)
    scope.add_despl(Id.pos, scope.despl)

    scope.despl += 1
    type = type_int
}

S.type = if type in {type_int, type_float, type_str}
    then type_ok
else type_error
```

30. $S \rightarrow$ Ret $X$ Semi

```
S.ret_type = X.type
```

31. $S \rightarrow$ Id Assign $E$ Semi

```
scope = symtable.scopes.peek()
type = scope.search_type(Id.pos)

if type == null {
    scope.add_type(Id.pos, type_int)
    scope.add_despl(Id.pos, scope.despl)

    scope.despl += 1
    type = type_int
}

S.type = if type == E.type
    then type_ok
else type_error
```

32. $S \rightarrow$ Id AndAssign $E$ Semi

```
scope = symtable.scopes.peek()
type = scope.search_type(Id.pos)

if type == null {
    scope.add_type(Id.pos, type_int)
    scope.add_despl(Id.pos, scope.despl)

    scope.despl += 1
    type = type_int
}

S.type = if type == E.type && type == type_bool
    then type_ok
else type_error
```

33. $S \rightarrow$ Id LParen $L$ RParen Semi

```
type = symtable.scopes.peek().search_type(Id.pos)

S.type = if type == null
    then type_error
else if type == L.type -> ret_type
    then type_ok
else type_error
```

34. $L \rightarrow EQ$

```
L.type = E.type x Q.type
```

35. $L \rightarrow \lambda$

```
L.type = type_void
```

36. $Q \rightarrow$ Comma $EQ_1$

```
Q.type = E.type x Q1.type
```

37. $Q \rightarrow \lambda$

```
Q.type = null
```

38. $X \rightarrow E$

```
X.type = E.type
```

39. $X \rightarrow \lambda$

```
X.type = type_void
```

40. $E \rightarrow E_1$ And $R$

```
E.type = if E1.type == R.type && E1.type == type_bool
    then type_bool
else type_error
```

41. $E \rightarrow R$

```
E.type = R.type
```

42. $R \rightarrow R_1$ Eq $RR$

```
R.type = if R1.type in {type_int, type_float} && R1.type == RR.type
    then type_bool
else type_error
```

43. $R \rightarrow RR$

```
R.type = RR.type
```

44. $RR \rightarrow RR_1$ Lt $U$

```
RR.type = if RR1.type in {type_int, type_float} && RR1.type == U.type
    then type_bool
else type_error
```

45. $RR \rightarrow U$

```
RR.type = U.type
```

46. $U \rightarrow U_1$ Sum $UU$

```
U.type = if U1.type in {type_int, type_float} && U1.type == UU.type
    then U1.type
else type_error
```

47. $U \rightarrow UU$

```
U.type = UU.type
```

48. $UU \rightarrow UU_1$ Mul $EE$

```
UU.type = if UU1.type in {type_int, type_float} && UU1.type == EE.type
    then UU1.type
else type_error
```

49. $UU \rightarrow EE$

```
UU.type = EE.type
```

50. $EE \rightarrow$ Not $EE_1$

```
EE.type = if EE1.type == type_bool
    then type_bool
else type_error
```

51. $EE \rightarrow$ Sub $EE_1$

```
EE.type = if EE1.type in {type_int, type_float}
    then EE1.type
else type_error
```

52. $EE \rightarrow$ Sum $EE_1$

```
EE.type = if EE1.type in {type_int, type_float}
    then EE1.type
else type_error
```

53. $EE \rightarrow V$

```
EE.type = V.type
```

54. $V \rightarrow$ Id LParen $L$ RParen

```
type = symtable.scopes.peek().search_type(Id.pos)

V.type = if type == null
    then type_error
else if type == L.type -> ret_type
    then ret_type
else type_error
```

55. $V \rightarrow$ LParen $E$ RParep

```
V.type = E.type
```

56. $V \rightarrow$ IntLit

```
V.type = type_int
V.type.size = 1
```

57. $V \rightarrow$ FloatLit

```
V.type = type_float
V.type.size = 2
```

58. $V \rightarrow$ StrLit

```
V.type = type_str
V.type.size = 64
```

59. $V \rightarrow$ True

```
V.type = type_bool
V.type.size = 1
```

60. $V \rightarrow$ False

```
V.type = type_bool
V.type.size = 1
```

61. $V \rightarrow$ Id

```
scope = symtable.scopes.peek()
type = scope.search_type(Id.pos)

if type == null {
    scope.add_type(Id.pos, type_int)
    scope.add_despl(Id.pos, scope.despl)

    scope.despl += 1
    type = type_int
}

V.type = type
```

# 8 Gestión de Errores

El Gestor de Errores, es el componente más importante de un procesador de cara a la *UX*. Es fundamental emitir errores que sean claros y se entiendan con facilidad.

Es por esto que se han decidido seguir estas pautas durante el diseño del Gestor de Errores:

- Todo error debe entenderse por sí mismo, es decir, en su mensaje debe estar toda la información necesaria para saber porque ha ocurrido, sin necesidad de consultar el fichero fuente.
- Los errores deben ser localizables en el fichero fuente, es decir, se debe proporcionar como mínimo el número de línea y columna de cada error, así como de su contexto.
- El gestor de errores debe ser conservador en la clasificación de los errores. Solo se emitirán diagnósticos específicos cuando el tipo de error pueda determinarse de forma inequívoca. Se intentará minimizar el uso de heurísticas y suposiciones para refinar el diagnóstico, ya que un diagnóstico incorrecto es inaceptable, incluso si solo falla 1 de cada 100 veces.

Además, el Gestor de Errores va a ser capaz de generar sugerencias en determinados errores. Es decir, como corregirlos a través de sustituciones, inserciones o eliminaciones de texto.

## 8.1 Estructura de un Diagnóstico

Cada diagnóstico va a estar formado por varios componentes, que juntos sirven la función de maximizar la claridad de cada error:

- Mensaje principal: Descripción general del error, que incluye el nombre del fichero, así como el número de línea y columna dónde se produce.
- Rango del error: Fragmento del fichero en el que se localiza el error, subrayado y resaltado de color rojo en la línea correspondiente, y acompañado de una nota con información más específica sobre el problema detectado.
- Rangos de notas: Otros intervalos del fichero, subrayados y resaltados de color azul, que aportan contexto adicional para la comprensión del error.
- Sugerencia: Cuando es posible determinar con certeza como corregir el error, se emite un breve mensaje que indica como modificar el fichero para resolver el fallo.

Para ilustrar la importancia de todos estos elementos, se considera un error del *Parser*: *Delimitador Desemparejado*. Este error ocurre cuando un paréntesis abierto se cierra con una llave o vicebersa.

Usando sólo el mensaje principal y el rango del error, el diagnóstico quedaría así:

```
tests/challenge/parser.txt:27:1: error: mismatched closing delimiter `}`
   |
27 | }
   | ^ mismatched closing delimiter
```

Este diagnóstico está bastante incompleto. Permite localizar el error pero sólo parcialmente, ya que muestra el delimitador cerrado desemparejado pero no el abierto. Esto también significa que el diagnóstico no se puede entender por sí mismo, el usuario tendría que abrir el fichero y ver porqué la llave está desemparejada y con que.

Para remediar esto, el *Parser* mantiene una pila con todos los delimitadores abiertos sin cerrar, por lo que se puede acceder fácilmente a la información que falta:

```
tests/challenge/parser.txt:27:1: error: mismatched closing delimiter `}`
   |
25 |     write(var
   |           ^ unclosed delimiter
   :
   |
27 | }
   | ^ mismatched closing delimiter
```

Además, este error casi siempre ocurre porque el usuario se ha olvidado de cerrar el delimitador abierto. Por ello, el delimitador cerrado desemparejado suele corresponder a otro delimitador abierto anterior que, en caso de encontrarse, puede proporcionar aún más contexto:

```
tests/challenge/parser.txt:27:1: error: mismatched closing delimiter `}`
   |
21 | function int operation(void) {
   |                              - closing delimiter possibly meant for this
   :
   |
25 |     write(var
   |           ^ unclosed delimiter
   :
   |
27 | }
   | ^ mismatched closing delimiter
```

Ahora el error es mucho más claro. Es localizable y también comprensible sin necesidad de consultar el fichero fuente. Además, con todo este contexto, es fácil ver que el error se puede arreglar cerrando el paréntesis abierto de la línea 25, por lo que es posible añadir una sugerencia:

```
tests/challenge/parser.txt:27:1: error: mismatched closing delimiter `}`
   |
21 | function int operation(void) {
   |                              - closing delimiter possibly meant for this
   :
   |
25 |     write(var
   |           ^ unclosed delimiter
   :
   |
27 | }
   | ^ mismatched closing delimiter
   |
  --> help: insert `)` and `;` after `var`
   |
25 |     write(var);
   |               ++
```

Todo este proceso se lleva a cabo de forma automática durante la generación de los diagnósticos del procesador. Gracias a la información mantenida por el *Parser*, se crean diagnósticos dinámicos y enriquecidos por su contexto. Esto permite emitir errores más informativos, robustos y accionables, capaces de guiar al usuario directamente hacia la causa del problema y su posible corrección.

## 8.2 Recuperación de Errores

Como se ha comentado a lo largo de la memoria, todos los módulos del procesador implementan recuperación de errores con el fin de reportar la mayor cantidad de errores por ejecución, minimizando el número de ejecuciones del procesador necesarias para corregir por completo el fichero fuente.

La estrategia de recuperación varía según el módulo:

### 8.2.1 Recuperación del Lexer

El *Lexer* tiene la recuperación de errores más sencilla de todas. Al encontrar un error, el Analizador Léxico intenta emitit un *token* ficticio que ayuda a que tanto el *Parser* como el Analizador Semántico generen diagnósticos más coherentes.

Por ejemplo, si ocurre un error de *Overflow de Entero*, como se sabe que el usuario quería escrbir una constante de tipo entero, se emite un *token* constante entera sin valor. Se emite un *token* ficticio en todos los diagnósticos del *Lexer* menos en *Carácter inválido*, en la que simplemente se salta el carácter y se continua analizando.

Las ventajas de esta estrategia frente a no emitir ningún *token* ficticio y continuar el análisis sin intervención pueden observarse en el siguiente fragmento:

```
1 let float real = 9238923.;
2
3 real = 3;
```

Que produce los siguientes diagnósticos:

```
lexer_text.txt:3:30: error: integer literal out of range for 16-byte type
   |
 3 | let string real = 92389233892;
   |                   ^^^^^^^^^^^ maximum is 32767

lexer_text.txt:3:30: error: mismatched types
   |
 3 | let string real = 92389233892;
   |     ------        ^^^^^^^^^^^ expected `string`, found `int`
   |     |
   |     expected `string` because of this
```

Gracias a que se genera el *token* fantasma, a pesar de producir un error, se puede detectar el error semántico en la declaración, ya que el usuario, hubiese *overflow* o no, estaba intentando asignar a una variable de tipo *string* una constante entera. Además si no se hubiese generado este *token*, el procesador habría emitido también errores sintácticos aparentemente sin sentido, ya que al omitir el *token* constante entera el *Parser* habría encontrado un *token* = seguido de un punto y coma.

### 8.2.2 Recuperación del Parser

Como se comentó brevemente en la sección de errores del *Parser*, cuando el Analizador Sintáctico encuentra un error, intenta encontrar una secuencia de cambios que permita resincronizar la pila de estados con el *token* que lo provocó.

Para ello se realiza una búsqueda heurística en anchura por el autómata $LR(1)$. Cada error se intenta corregir mediante una secuencia de inserciones, una secuencia de eliminaciones o una sustitución. Cada arreglo se puntua según el número de inserciones y/o eliminaciones que conlleva, y se selecciona la solución con la puntuación más baja.

Si no se encuentra una corrección adecuada, o si el arreglo resultara demasiado costoso o inconsistente, se emite el error y se interrumpe la ejecución del procesador.

La clasificación de los errores se realiza analizando los contenidos de la pila de estados junto al *token* que produjo el error. Esta estrategia se ha escogido frente a una clasificación manual basada en las celdas de las tablas de acción y *goto* por su flexibilidad. Una clasificación manual resulta demasiado laboriosa, ya que no solo hay cientos de celdas en las tablas, sino que cualquier modificación de la gramática obligaría a rehacer rehacer por completo el manejo de los errores.

A continuación se muetran un ejemplo de cada tipo de recuperación: Por inserción, eliminación y sustitución.

```
tests/challenge/parser.txt:37:21: error: expected a statement or a function, found `int`
   |
36 | int a = 4;
   |          - after this
   |
37 | int wellwell(void) {}
   | ^^^ expected a statement or a function
   |
  --> help: insert `function` before `int`
   |
37 | function int wellwell(void) {}
   | ++++++++
```

```
tests/challenge/parser.txt:52:14: error: trailing comma in a function call
   |
52 | chachi(hola,);
   |             ^ here
   |
  --> help: remove the trailing comma
   |
52 - chachi(hola,);
52 + chachi(hola);
   |
```

```
tests/challenge/parser.txt:45:6: error: expected `;` or a binary operator, found `,`
   |
45 | b = 4,
   |      ^ expected `;` or a binary operator
   |
  --> help: replace `,` by `;`
   |
45 - b = 4,
45 + b = 4;
   |
```

### 8.2.3 Recuperación del Analizador Semántico

La recuperación de este módulo ya se vio en parte en sus acciones semánticas. Al encontrar un error se propaga el tipo especial *tipo_error*, que permite continuar el análisis y la emisión de más excepciones cuando se reducen conjuntos de símbolos sin errores.

Es importante también como se comporta cuando ocurren errores sintácticos. Se considera el siguiente fragmento:

```
1 let var = ;
2
3 var = "hola";
```

El analizador sintáctico para recuperar los errores de ese fichero insertará un tipo tras el *let* y una expresión tras el = en la primera línea. Sin embargo, estas inserciones no tienen porque ser coherentes semánticamente y desde luego no tienen porque ser compatibles con la tercera línea. Por ello, si el Analizador Semántico ve que los *tokens* que produjeron el error son ficticios, no emite ningún diagnóstico y simplemente asigna *tipo_error* y continua. Habrá que esperar a que se corrijan los errores sintácticos para emitir los semánticos:

```
sem_test.txt:1:5: error: missing type in a variable declaration
  |
1 | let var = ;
  |     ^^^ expected type
  |
 --> help: add the missing type
  |
1 | let type var = ;
  |     ++++
```

Sin embargo, si el fragmento fuese el siguiente:

```
1 int var = 3;
2
3 var = "hola";
```

Ahora sí que se emitirán diagnósticos ya que se sabe que el tipo de *var* es *int*, aunque la sentencia genere un error:

```
sem_test.txt:1:1: error: expected a function or a statement, found `int`
  |
1 | int var = 3;
  | ^^^ expected a function or a statement
  |
 --> help: insert `let` before `int`
  |
1 | let int var = 3;
  | +++

sem_test.txt:3:13: error: mismatched types
  |
1 | int var = 3;
  | --- expected `int` due to it's declaration
  :
  |
3 | var = "hola";
  |       ^^^^^^ expected `int`, found `string`
```

# Anexo

# A Casos de Prueba

Se va a probar el funcionamiento del procesador con 6 ficheros fuentes distintos. La mitad de ellos serán correctos y la otra incorrectos. En los casos correctos se volcará el fichero de *tokens* y de la tabla de símbolos; en los incorrectos los diagnósticos generados por el gestor de errores.

## A.1 Casos Correctos

**fib.javascript** Fichero con un estilo limpio y estándar.

```
/* This function computes the nth fibonacci number */
function int fib(int n) {
    if (n == 0)
        return 0;

    if (n == 1)
        return 1;

    let int a = 0;
    let int b = 1;

    do {
        let int c = a + b;
        a = b; b = c;

        n = n + -1;
    } while (!(n < 2));

    return b;
}
```

**Fichero de *tokens*:**

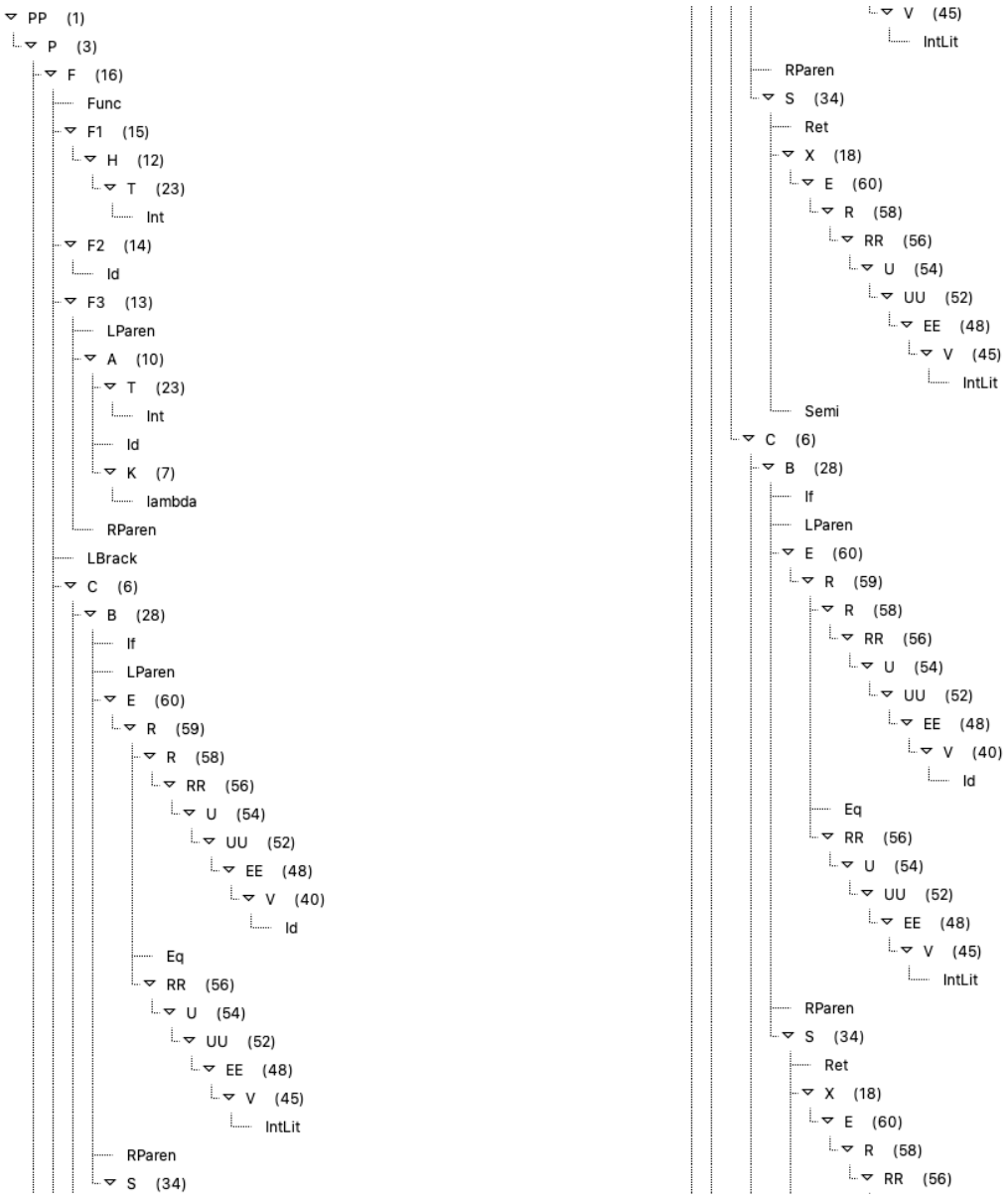
```
<Func, >
<Int, >
<Id, 0>
<LParen, >
<Int, >
<Id, 1>
<RParen, >
<LBrack, >
<If, >
<LParen, >
<Id, 1>
<Eq, >
<IntLit, 0>
<RParen, >
<Ret, >
<IntLit, 0>
<Semi, >
<If, >
<LParen, >
<Id, 1>
<Eq, >
<IntLit, 1>
<RParen, >
<Ret, >
<IntLit, 1>
<Semi, >
<Let, >
<Int, >
<Id, 2>
<Assign, >
<IntLit, 0>
<Semi, >
<Let, >
<Int, >
<Id, 3>
<Assign, >
<IntLit, 1>
<Semi, >
<Do, >
<LBrack, >
<Let, >
<Int, >
<Id, 4>
<Assign, >
<Id, 2>
<Sum, >
<Id, 3>
<Semi, >
<Id, 2>
<Assign, >
<Id, 3>
<Semi, >
<Id, 3>
<Assign, >
<Id, 4>
<Semi, >
<Id, 1>
<Assign, >
<Id, 1>
<Sum, >
<Sub, >
<IntLit, 1>
<Semi, >
<RBrack, >
<While, >
<LParen, >
<Not, >
<LParen, >
<Id, 1>
<Lt, >
<IntLit, 2>
<RParen, >
<RParen, >
<Semi, >
<Ret, >
<Id, 3>
<Semi, >
<RBrack, >
```

**Tabla de símbolos:**

```
table #0:
* 'fib'
* 'n'
* 'a'
* 'b'
* 'c'
```

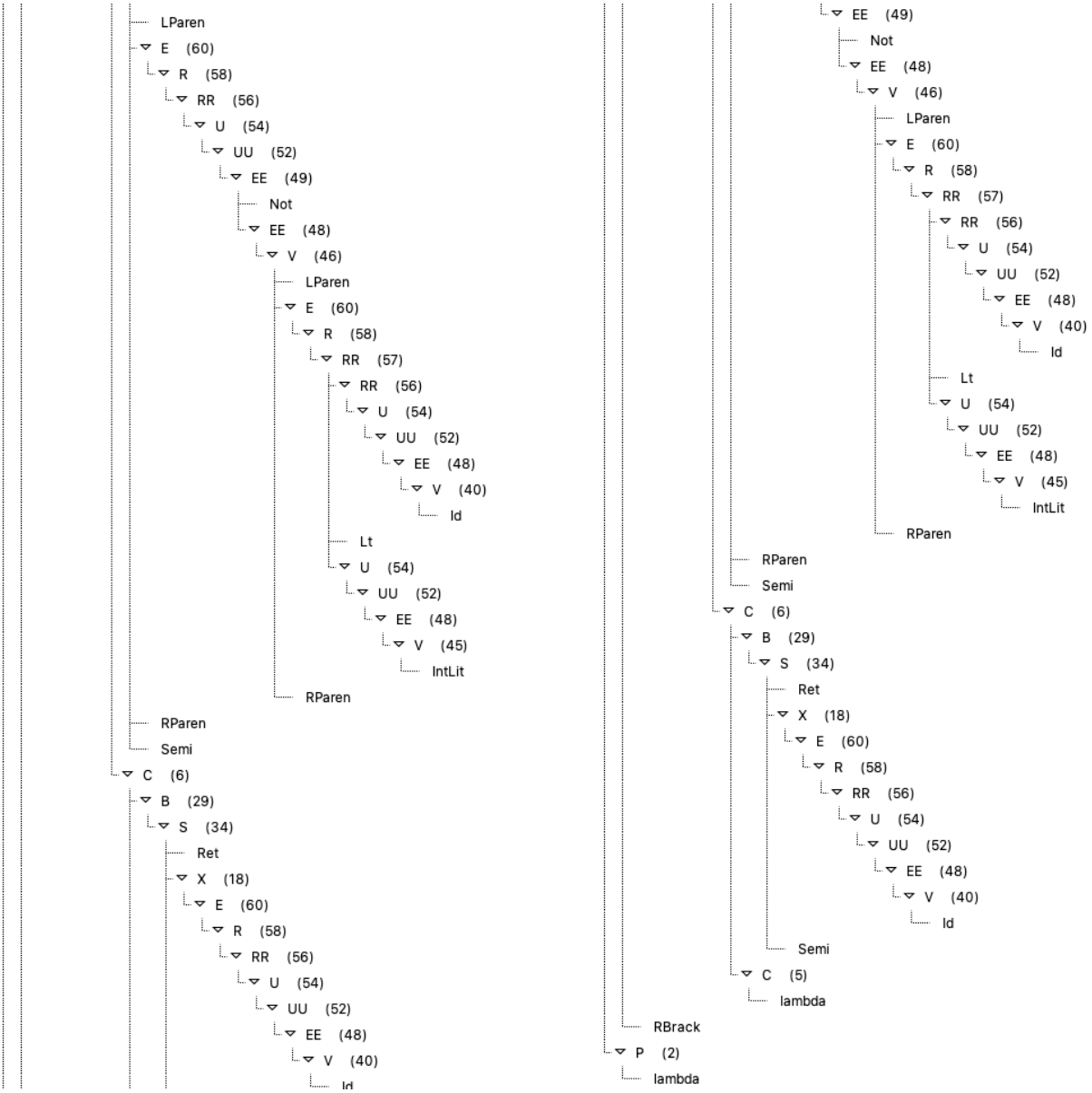
**Parse:**

```
ascending 23 12 15 14 23 7 10 13 40 48 52 54 56 58 45 48 52 54 56 59 60 45 48 52 54 56 58 60 18 34
      28 40 48 52 54 56 58 45 48 52 54 56 59 60 45 48 52 54 56 58 60 18 34 28 19 23 45 48 52 54 56
    58 60 26 19 23 45 48 52 54 56 58 60 26 19 23 40 48 52 54 40 48 52 55 56 58 60 26 40 48 52 54
    56 58 60 39 29 40 48 52 54 56 58 60 39 29 40 48 52 54 45 48 51 52 55 56 58 60 39 29 5 6 6 6 6
    40 48 52 54 56 45 48 52 54 57 58 60 46 48 49 52 54 56 58 60 24 40 48 52 54 56 58 60 18 34 29 5
     6 6 6 6 6 6 16 2 3 1
```


PP (1)
P (3)
F (16)
Func
F1 (15)
H (12)
T (23)
Int
F2 (14)
Id
F3 (13)
LParen
A (10)
T (23)
Int
Id
K (7)
lambda
RParen
LBrack
C (6)
B (28)
If
LParen
E (60)
R (59)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Eq
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)
V (45)
IntLit
RParen
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (28)
If
LParen
E (60)
R (59)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Eq
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)


R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Eq
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (24)
Do
LBrack
C (6)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (55)


R (58)
RR (56)
U (55)
U (54)
UU (52)
EE (48)
V (40)
Id
Sum
UU (52)
EE (48)
V (40)
Id
Semi
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
EE (48)
V (40)
Id
Semi
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (55)
U (54)
UU (52)
EE (48)
V (40)
Id
Sum
UU (52)
EE (51)
Sub
EE (48)
V (45)
IntLit
Semi
C (5)
lambda
RBrack
While
LParen
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (49)
Not
EE (48)
V (46)
LParen
E (60)


LParen
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (49)
Not
EE (48)
V (46)
LParen
E (60)
R (58)
RR (57)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Lt
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
RParen
Semi
C (6)
B (29)
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
EE (49)
Not
EE (48)
V (46)
LParen
E (60)
R (58)
RR (57)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Lt
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
RParen
Semi
C (6)
B (29)
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
C (5)
lambda
RBrack
P (2)
lambda

**factorial.javascript** Fichero con un código más comprimido y comentarios más raros.

```
/*****************************************/
/*          * / * N FACTORIAL * / *          */
/*****************************************/

/*
* This function computes n! (n factorial)
*/
function int factorial(int n) {
    if(n==0)return 1;
    if(n<2)return n;
    let int res=n;
    do{n=n + -1;res=res*n;}while(!(n<2));
    return res;
}

/* read n and write n! */
let int n=0;read n;let int res=factorial(n);write(res);

/*** eof ***/
```

**Fichero de *tokens*:**

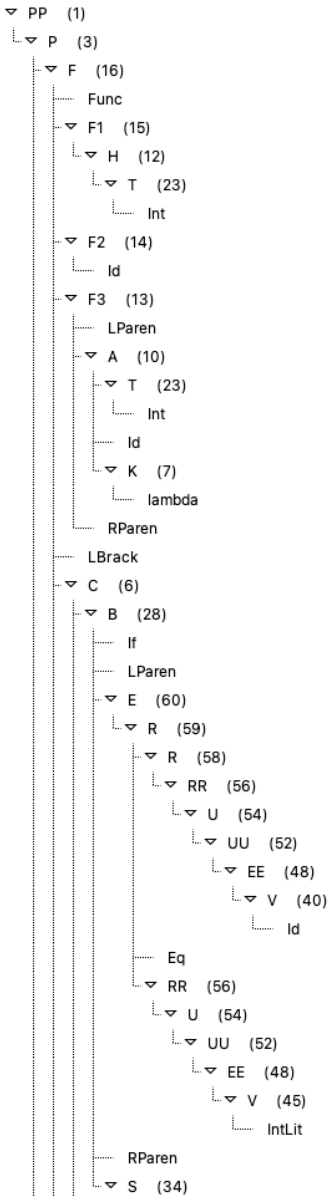
```
<Func, >
<Int, >
<Id, 0>
<LParen, >
<Int, >
<Id, 1>
<RParen, >
<LBrack, >
<If, >
<LParen, >
<Id, 1>
<Eq, >
<IntLit, 0>
<RParen, >
<Ret, >
<IntLit, 1>
<Semi, >
<If, >
<LParen, >
<Id, 1>
<Lt, >
<IntLit, 2>
<RParen, >
<Ret, >
<Id, 1>
<Semi, >
<Let, >
<Int, >
<Id, 2>
<Assign, >
<Id, 1>
<Semi, >
<Do, >
<LBrack, >
<Id, 1>
<Assign, >
<Id, 1>
<Sum, >
<Sub, >
<IntLit, 1>
<Semi, >
<Id, 2>
<Assign, >
<Id, 2>
<Mul, >
<Id, 1>
<Semi, >
<RBrack, >
<While, >
<LParen, >
<Not, >
<LParen, >
<Id, 1>
<Lt, >
<IntLit, 2>
<RParen, >
<RParen, >
<Semi, >
<Ret, >
<Id, 2>
<Semi, >
<RBrack, >
<Let, >
<Int, >
<Id, 1>
<Assign, >
<IntLit, 0>
<Semi, >
<Read, >
<Id, 1>
<Semi, >
<Let, >
<Int, >
<Id, 2>
<Assign, >
<Id, 0>
<LParen, >
<Id, 1>
<RParen, >
<Semi, >
<Write, >
<LParen, >
<Id, 2>
<RParen, >
<Semi, >
```

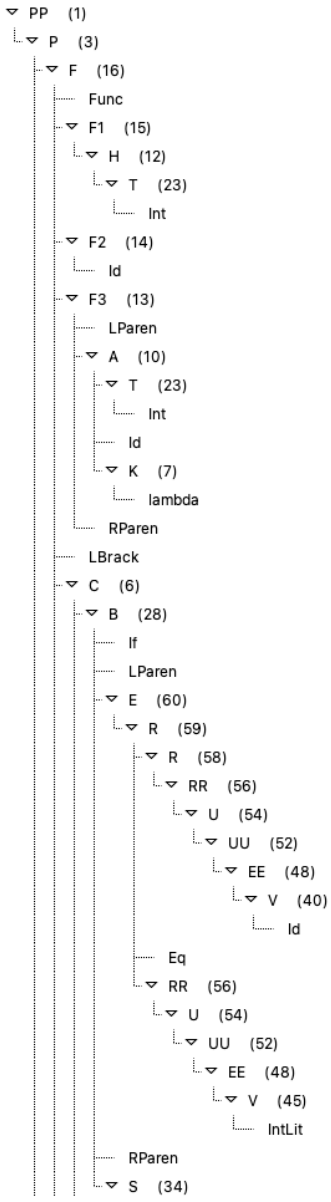
**Tabla de símbolos:**

```
table #0:
* 'factorial'
* 'n'
* 'res'
```

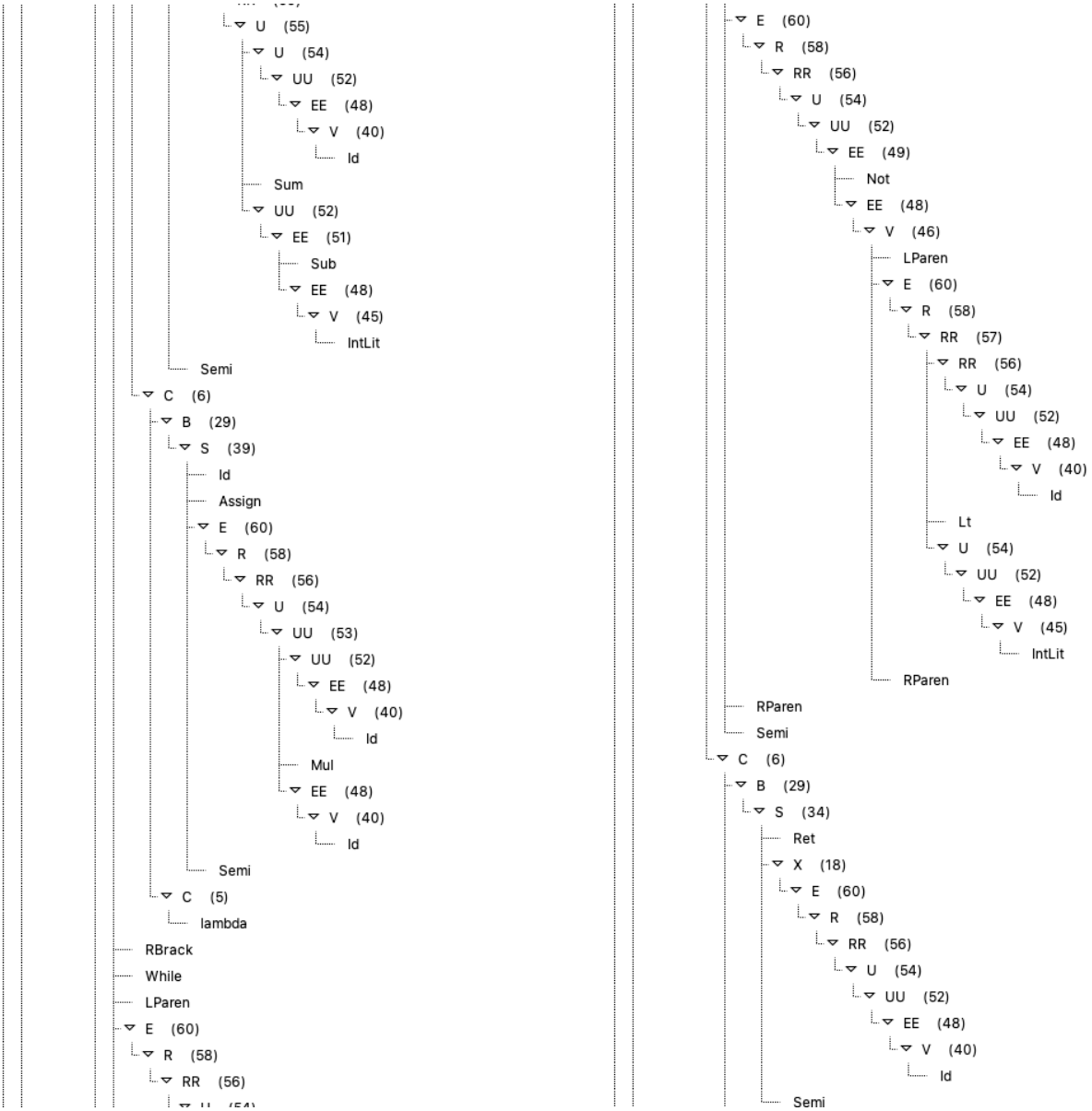
**Parse:**

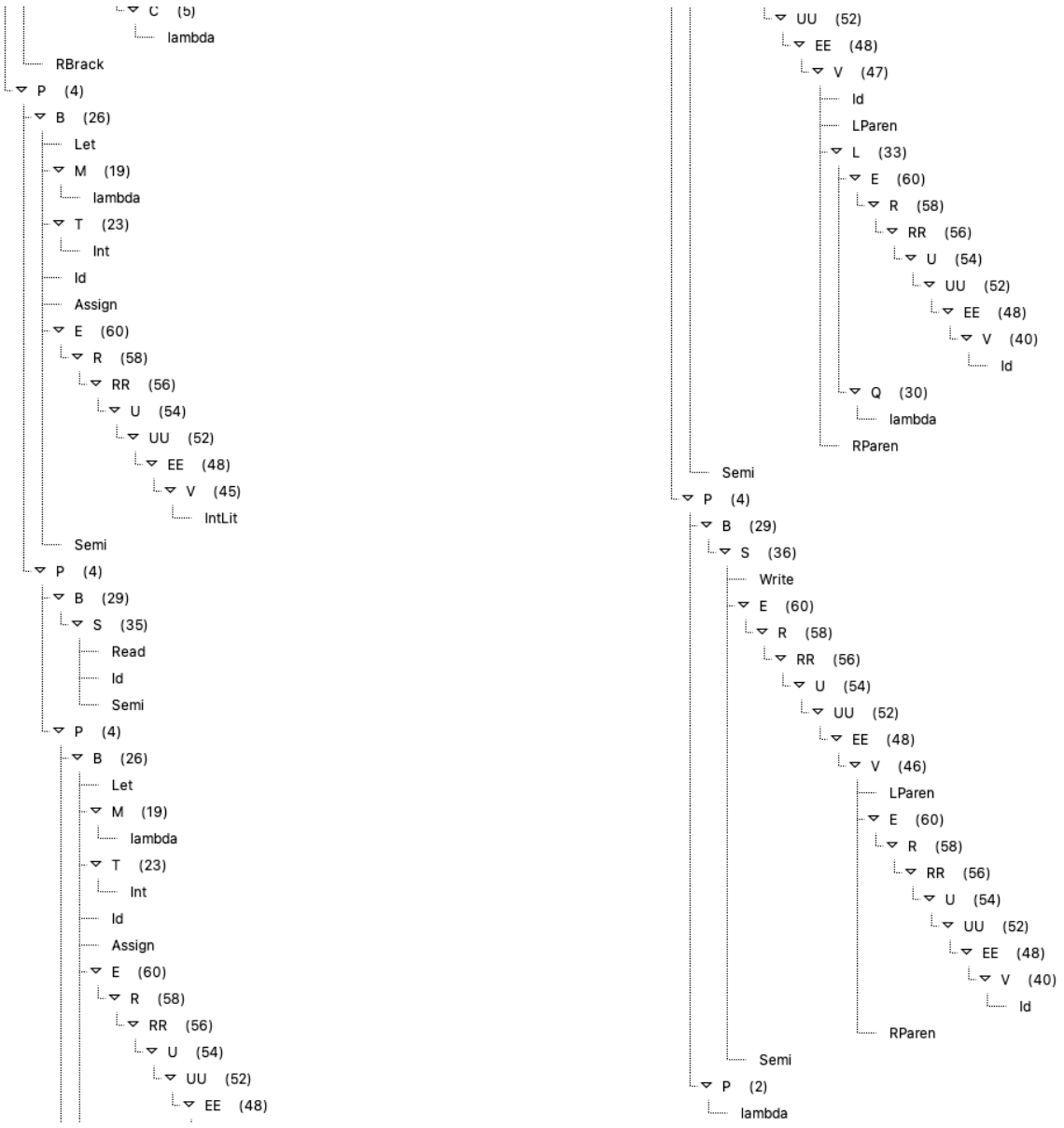
```
ascending 23 12 15 14 23 7 10 13 40 48 52 54 56 58 45 48 52 54 56 59 60 45 48 52 54 56 58 60 18 34
      28 40 48 52 54 56 45 48 52 54 57 58 60 40 48 52 54 56 58 60 18 34 28 19 23 40 48 52 54 56 58
    60 26 40 48 52 54 45 48 51 52 55 56 58 60 39 29 40 48 52 40 48 53 54 56 58 60 39 29 5 6 6 40
    48 52 54 56 45 48 52 54 57 58 60 46 48 49 52 54 56 58 60 24 40 48 52 54 56 58 60 18 34 29 5 6
    6 6 6 6 16 19 23 45 48 52 54 56 58 60 26 35 29 19 23 40 48 52 54 56 58 60 30 33 47 48 52 54 56
     58 60 26 40 48 52 54 56 58 60 46 48 52 54 56 58 60 36 29 2 4 4 4 4 3 1
```


PP (1)
P (3)
F (16)
Func
F1 (15)
H (12)
T (23)
Int
F2 (14)
Id
F3 (13)
LParen
A (10)
T (23)
Int
Id
K (7)
lambda
RParen
LBrack
C (6)
B (28)
If
LParen
E (60)
R (59)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Eq
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)


PP (1)
P (3)
F (16)
Func
F1 (15)
H (12)
T (23)
Int
F2 (14)
Id
F3 (13)
LParen
A (10)
T (23)
Int
Id
K (7)
lambda
RParen
LBrack
C (6)
B (28)
If
LParen
E (60)
R (59)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Eq
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)


V (45)
IntLit
RParen
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (28)
If
LParen
E (60)
R (58)
RR (57)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Lt
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
C (6)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
C (6)
B (24)
Do
LBrack
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (55)
U (54)
UU (52)


U (55)
U (54)
UU (52)
EE (48)
V (40)
Id
Sum
UU (52)
EE (51)
Sub
EE (48)
V (45)
IntLit
Semi
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (53)
UU (52)
EE (48)
V (40)
Id
Mul
EE (48)
V (40)
Id
Semi
C (5)
lambda
RBrack
While
LParen
E (60)
R (58)
RR (56)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (49)
Not
EE (48)
V (46)
LParen
E (60)
R (58)
RR (57)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Lt
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
RParen
Semi
C (6)
B (29)
S (34)
Ret
X (18)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi


C (5)
lambda
RBrack
P (4)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
P (4)
B (29)
S (35)
Read
Id
Semi
P (4)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
UU (52)
EE (48)
V (47)
Id
LParen
L (33)
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Q (30)
lambda
RParen
Semi
P (4)
B (29)
S (36)
Write
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (46)
LParen
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
RParen
Semi
P (2)
lambda

**fuzz.javascript** Fichero correcto pero caótico, con el objectivo de obtener *edgecases*.

```
let int global=13;
function void nothing(void) { read global; do{global=global+-1;}while(!(global<0));}
function int main(float b, string d){let string aux="this is a str";
/*** this is a comment ***/ let float foo=1203.123; let int bar=2398;
/* semicolons */let int weird &= 3+-+-+-+-+-+-+-+-+5; let float oper=2.0000; let boolean bool=
    false;
let boolean george=true;george&=bool;}/*** / eof /***/
```

**Fichero de *tokens*:**

```
<Let, >
<Int, >
<Id, 0>
<Assign, >
<IntLit, 13>
<Semi, >
<Func, >
<Void, >
<Id, 1>
<LParen, >
<Void, >
<RParen, >
<LBrack, >
<Read, >
<Id, 0>
<Semi, >
<Do, >
<LBrack, >
<Id, 0>
<Assign, >
<Id, 0>
<Sum, >
<Sub, >
<IntLit, 1>
<Semi, >
<RBrack, >
<While, >
<LParen, >
<Not, >
<LParen, >
<Id, 0>
<Lt, >
<IntLit, 0>
<RParen, >
<RParen, >
<Semi, >
<RBrack, >
<Func, >
<Int, >
<Id, 2>
<LParen, >
<Float, >
<Id, 3>
<Comma, >
<Str, >
<Id, 4>
<RParen, >
<LBrack, >
<Let, >
<Str, >
<Id, 5>
<Assign, >
<StrLit, "this is a str">
<Semi, >
<Let, >
<Float, >
<Id, 6>
<Assign, >
<FloatLit, 1203.23>
<Semi, >
<Let, >
<Int, >
<Id, 7>
<Assign, >
<IntLit, 2398>
<Semi, >
<Let, >
<Int, >
<Id, 8>
<AndAssign, >
<IntLit, 3>
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sub, >
<Sum, >
<Sum, >
<IntLit, 5>
<Semi, >
<Let, >
<Float, >
<Id, 9>
<Assign, >
<FloatLit, 2>
<Semi, >
<Let, >
<Bool, >
<Id, 10>
<Assign, >
<False, >
<Semi, >
<Let, >
<Bool, >
<Id, 11>
<Assign, >
<True, >
<Semi, >
<Id, 11>
<AndAssign, >
<Id, 10>
<Semi, >
<RBrack, >
```

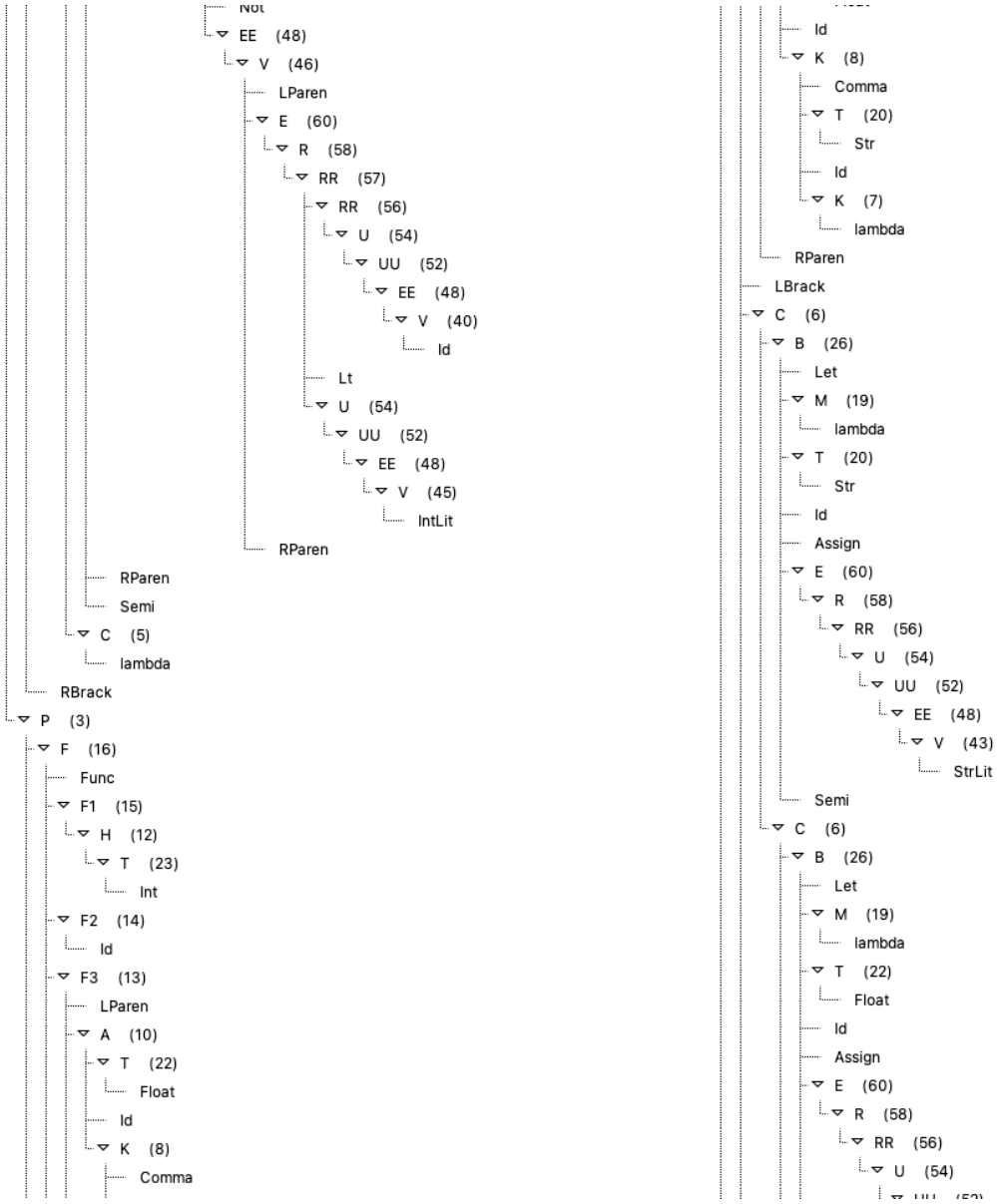
**Tabla de símbolos:**

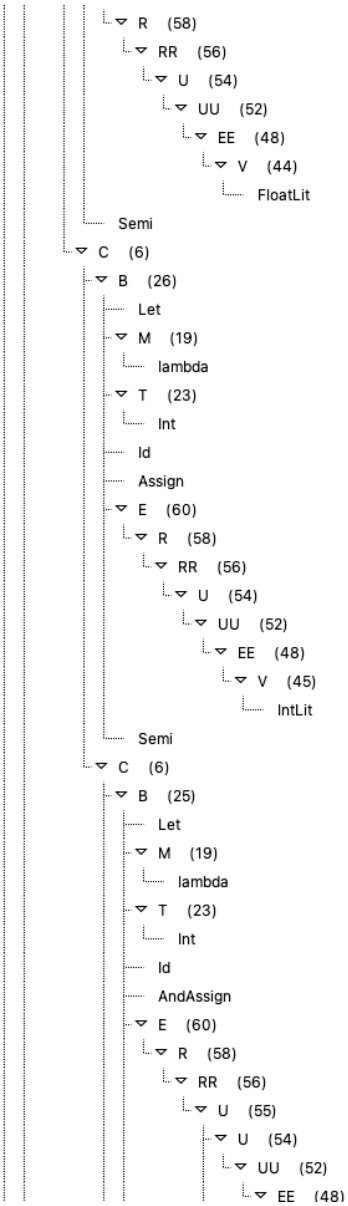
```
table #0:
* 'global'
* 'nothing'
* 'main'
* 'b'
* 'd'
* 'aux'
* 'foo'
* 'bar'
* 'oper'
* 'bool'
* 'george'
```

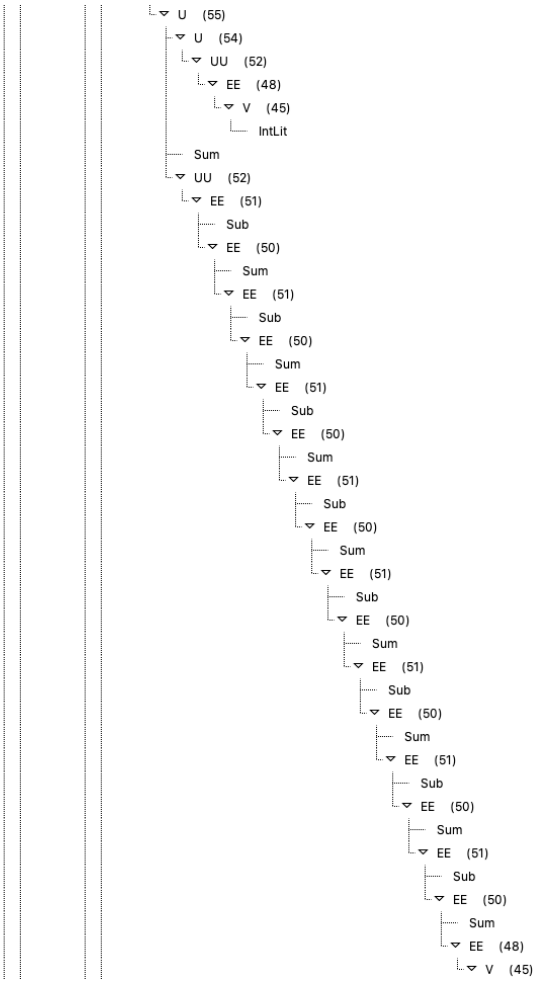
**Parse:**

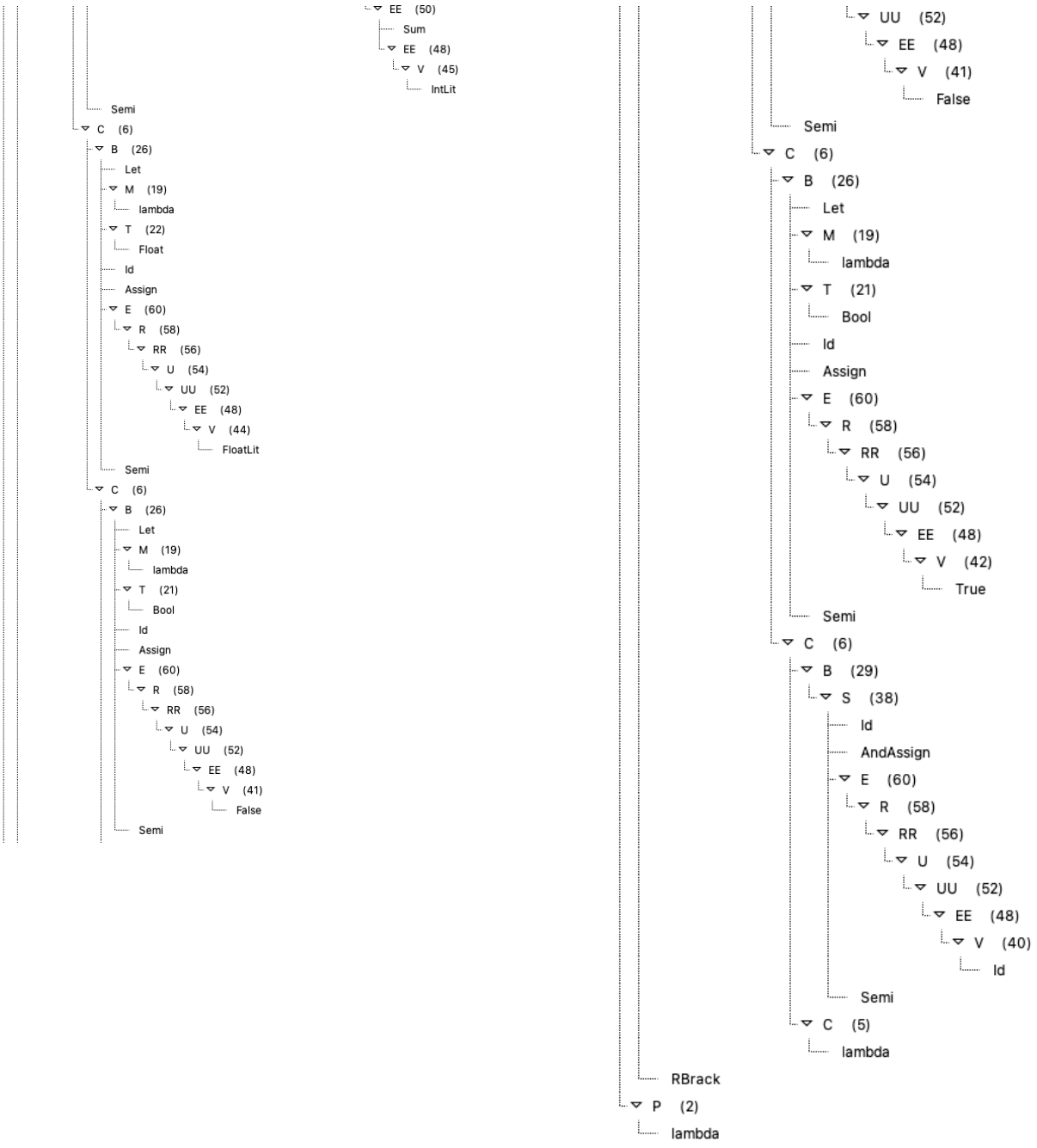
```
ascending 19 23 45 48 52 54 56 58 60 26 11 15 14 9 13 35 29 40 48 52 54 45 48 51 52 55 56 58 60 39 29 5
    6 40 48 52 54 56 45 48 52 54 57 58 60 46 48 49 52 54 56 58 60 24 5 6 6 16 23 12 15 14 22 20 7 8 10
    13 19 20 43 48 52 54 56 58 60 26 19 22 44 48 52 54 56 58 60 26 19 23 45 48 52 54 56 58 60 26 19 23
    45 48 52 54 45 48 50 51 50 51 50 51 50 51 50 51 50 51 50 51 50 51 52 55 56 58 60 25 19 22 44 48 52
    54 56 58 60 26 19 21 41 48 52 54 56 58 60 26 19 21 42 48 52 54 56 58 60 26 40 48 52 54 56 58 60 38
    29 5 6 6 6 6 6 6 6 6 16 2 3 3 4 1
```


PP (1)
P (4)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
P (3)
F (16)
Func
F1 (15)
H (11)
Void
F2 (14)
Id
F3 (13)
LParen
A (9)
Void
RParen
LBrack
C (6)
B (29)
S (35)
Read
Id
Semi
C (6)
B (24)
Do
Semi
C (6)
B (24)
Do
LBrack
C (6)
B (29)
S (39)
Id
Assign
E (60)
R (58)
RR (56)
U (55)
U (54)
UU (52)
EE (48)
V (40)
Id
Sum
UU (52)
EE (51)
Sub
EE (48)
V (45)
IntLit
Semi
C (5)
lambda
RBrack
While
LParen
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (49)
Not
EE (48)
V (46)
LParen


Not
EE (48)
V (46)
LParen
E (60)
R (58)
RR (57)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Lt
U (54)
UU (52)
EE (48)
V (45)
IntLit
RParen
RParen
Semi
C (5)
lambda
RBrack
P (3)
F (16)
Func
F1 (15)
H (12)
T (23)
Int
F2 (14)
Id
F3 (13)
LParen
A (10)
T (22)
Float
Id
K (8)
Comma
Id
K (8)
Comma
T (20)
Str
Id
K (7)
lambda
RParen
LBrack
C (6)
B (26)
Let
M (19)
lambda
T (20)
Str
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (43)
StrLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (22)
Float
Id
Assign
E (60)
R (58)
RR (56)
U (54)


R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (44)
FloatLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (23)
Int
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Semi
C (6)
B (25)
Let
M (19)
lambda
T (23)
Int
Id
AndAssign
E (60)
R (58)
RR (56)
U (55)
U (54)
UU (52)
EE (48)


U (55)
U (54)
UU (52)
EE (48)
V (45)
IntLit
Sum
UU (52)
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (51)
Sub
EE (50)
Sum
EE (48)
V (45)


EE (50)
Sum
EE (48)
V (45)
IntLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (22)
Float
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (44)
FloatLit
Semi
C (6)
B (26)
Let
M (19)
lambda
T (21)
Bool
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (41)
False
Semi
UU (52)
EE (48)
V (41)
False
Semi
C (6)
B (26)
Let
M (19)
lambda
T (21)
Bool
Id
Assign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (42)
True
Semi
C (6)
B (29)
S (38)
Id
AndAssign
E (60)
R (58)
RR (56)
U (54)
UU (52)
EE (48)
V (40)
Id
Semi
C (5)
lambda
RBrack
P (2)
lambda

### A.2 Casos Incorrectos

**unterm.javascript** Fichero con errores de sentencias incompletas.

```
/* unterminated declaration */
let int = 2;

/* unfinished strings */
let string foo = "im not finishing this string
let string bar = "im not finishing this one either\
let string rep = "last one\q

/* unfinished float */
let float x = 478234.

/*** * / * ///** / this comment is finished /* /* ***/
/*** this one is not ** ****** *** /*

function int main {
    let string useless = "this variable won't be recognised";
    return 0;
}
```

**Diagnósticos:**

```
unterm.javascript:5:18: error: missing terminating character '"' on string literal
    |
  5 | let string foo = "im not finishing this string
    |                  ^ started here
    |
unterm.javascript:6:18: error: missing terminating character '"' on string literal
    |
  6 | let string bar = "im not finishing this one either\
    |                  ^ started here
    |
unterm.javascript:7:27: warning: unknown escape sequence '\q'
    |
  7 | let string rep = "last one\q
    |                           ^^ interpreted as \\q
    |
unterm.javascript:7:18: error: missing terminating character '"' on string literal
    |
  7 | let string rep = "last one\q
    |                  ^ started here
    |
unterm.javascript:10:15: error: expected digit after '.' in float literal
    |
 10 | let float x = 478234.
    |               ^^^^^^^ perhaps you meant '478234.0'
    |
unterm.javascript:13:1: error: unterminated block comment
    |
 13 | /*** this one is not ** ****** *** /*
    | ^^ started here
    |
unterm.javascript:2:9: error: expected 'identifier' before '='
    |
  2 | let int = 2;
    |         ^ before this token
    |
```

**overflow.javascript** Fichero con errores de constantes fuera del rango admitido.

```
/* parse error, missing type */
function hello() {}

/* string with 64 characters */
let string foo = "ffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffffff";

/* string with 65 characters */
let string bar = "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";

/* max int16 */
let int imax = 32767;
let int imax_plus_one = 32768;

/* max float */
let float fmax = 340282346638528859811704183484516925440.0;
let float fmax_plus_more = 3402823466385288598117041834845169254382923892341.0;
let float lots_of_decimals = 0.23470283498209348092184092384529038092 8;
```

**Diagnósticos:**

```
overflow.javascript:8:18: error: string literal is too long
   |
 8 | let string bar = "bbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbbb";
   |                  ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ length is 65 but maximum is 64
   |
overflow.javascript:12:25: error: integer literal out of range for 16-byte type
   |
12 | let int imax_plus_one = 32768;
   |                         ^^^^^ maximum is 32767
   |
overflow.javascript:16:28: error: float literal out of range for 32-byte type
   |
16 | let float fmax_plus_more = 3402823466385288598117041834845169254382923892341.0;
   |                            ^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^^ maximum is 3.4028235e38
   |
overflow.javascript:2:10: error: expected 'int', 'float', 'string', 'boolean' or 'void' before 'hello'
   |
 2 | function hello() {}
   |          ^^^^^ before this token
   |
```

**noise.javascript** Fichero de ruido, repleto de errores diferentes.

```
if condition write "no parenthesis";
123.45.67 12abc 123456. 78
"bad\escape" "no closing quote
/* nested comment */ /* comment */ /* unclosed comment*/
=!= &== != <=> &&|!!
"unterminated string again /* unclosed comment
"bad string with \x illegal escape" "" ""empty""
"Hello^[World"
/* final comment */ /* another unclosed
```

**Diagnósticos:**

```
noise.javascript:2:7: error: illegal character '.' in program
   |
 2 | 123.45.67 12abc 123456. 78
   |       ^ here
   |
noise.javascript:2:17: error: expected digit after '.' in float literal
   |
 2 | 123.45.67 12abc 123456. 78
   |                 ^^^^^^^ perhaps you meant '123456.0'
   |
noise.javascript:3:5: warning: unknown escape sequence '\e'
   |
 3 | "bad\escape" "no closing quote
   |     ^^ interpreted as \\e
   |
noise.javascript:3:14: error: missing terminating character '"' on string literal
   |
 3 | "bad\escape" "no closing quote
   |              ^ started here
   |
noise.javascript:5:14: error: illegal character '>' in program
   |
 5 | =!= &== != <=> &&|!!
   |              ^ here
   |
noise.javascript:5:18: error: illegal character '|' in program
   |
 5 | =!= &== != <=> &&|!!
   |                  ^ here
   |
noise.javascript:6:1: error: missing terminating character '"' on string literal
   |
 6 | "unterminated string again /* unclosed comment
   | ^ started here
   |
noise.javascript:7:18: warning: unknown escape sequence '\x'
   |
 7 | "bad string with \x illegal escape" "" ""empty""
   |                  ^^ interpreted as \\x
   |
noise.javascript:8:7: error: malformed string literal, contains control character '\u{1b}'
   |
 8 | "Hello\u{1b}World"
   |       ^^^^^^ remove this character
   |
noise.javascript:9:21: error: unterminated block comment
   |
 9 | /* final comment */ /* another unclosed
   |                     ^^ started here
   |
noise.javascript:1:4: error: expected '(' before 'condition'
   |
 1 | if condition write "no parenthesis";
   |    ^^^^^^^^^ before this token
   |
```