
# Universidad Carlos III
# Grado en ingeniería informática
## Curso Procesadores del Lenguaje 2024-25
## Laboratorio drLL 1
Curso 2024-25

Fecha: **05/03/2025** - ENTREGA: drLL 1

GRUPO: **403**

Alumnos: **Mario Ramos Salsón (100495849)**

**Miguel Yubero Espinosa (100495984)**

# Explicación de los avances realizados

Al revisar el código propuesto en el fichero drLL.c, se nos pide responder a la siguiente pregunta:

**¿Qué elementos léxicos proporciona rd_lex()?**

El analizador léxico rd_lex(), nos proporciona los siguientes elementos léxicos:

- Números enteros (T_NUMBER).
- Variables de tipo Letter o Letter[Digit] (T_VARIABLE).
- Operadores aritméticos (+, -, *, /) (T_OPERATOR).
- Caracteres literales individuales.
- Fin de archivo (EOF).

**Gramática propuesta:**

E -> λ | NE | OE

O -> + | - | * | /

N -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

*Aclaración*: Expresión = E | Operador = O | Número = N

Al probar la gramática propuesta en JFLAP comprobamos que cumple con las condiciones LL(1).

Pero podemos ver que nos falta la inclusión de paréntesis para que cumpla la notación prefija. Por ello, rediseñamos la gramática, y ahora nos queda de la siguiente manera:

E -> NT | (E)T

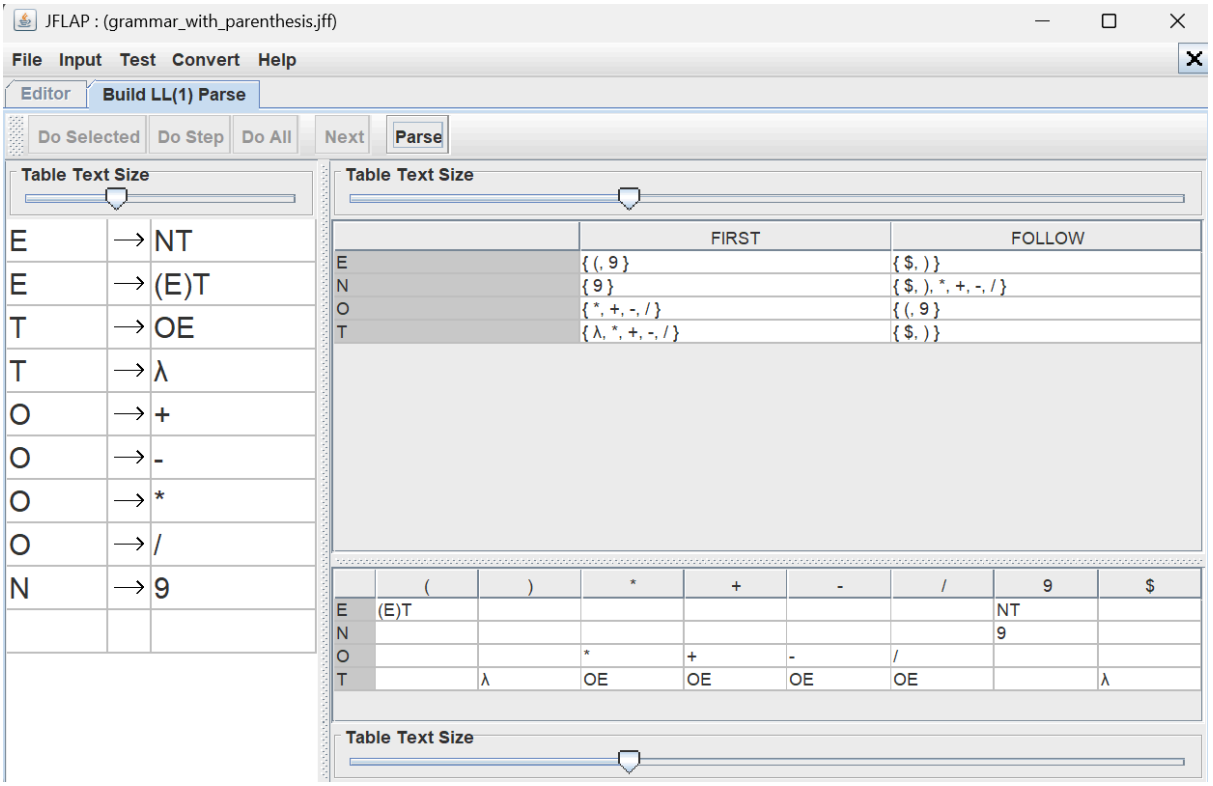
T -> λ | OE

O -> + | - | * | /

N -> 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

*Aclaración*: T = Expresion Tail ( E')

Al probar la gramática en JFLAP como LL(1), vemos que igual cumple con las condiciones, y obtenemos los siguientes conjuntos PRIMERO y SIGUIENTE:




Observación: En esta imagen se prueba la gramática en JFLAP solo con el dígito 9, es decir, el símbolo terminal “9” en el No Terminal “N”. Pero si añadimos todos los dígitos del 0 al 9 no habría ningún problema.

Pero nuevamente, modificamos la gramática para dar recursividad a los dígitos/números, y queda de la siguiente forma:

E -> NT | (E)T

T -> λ | OE

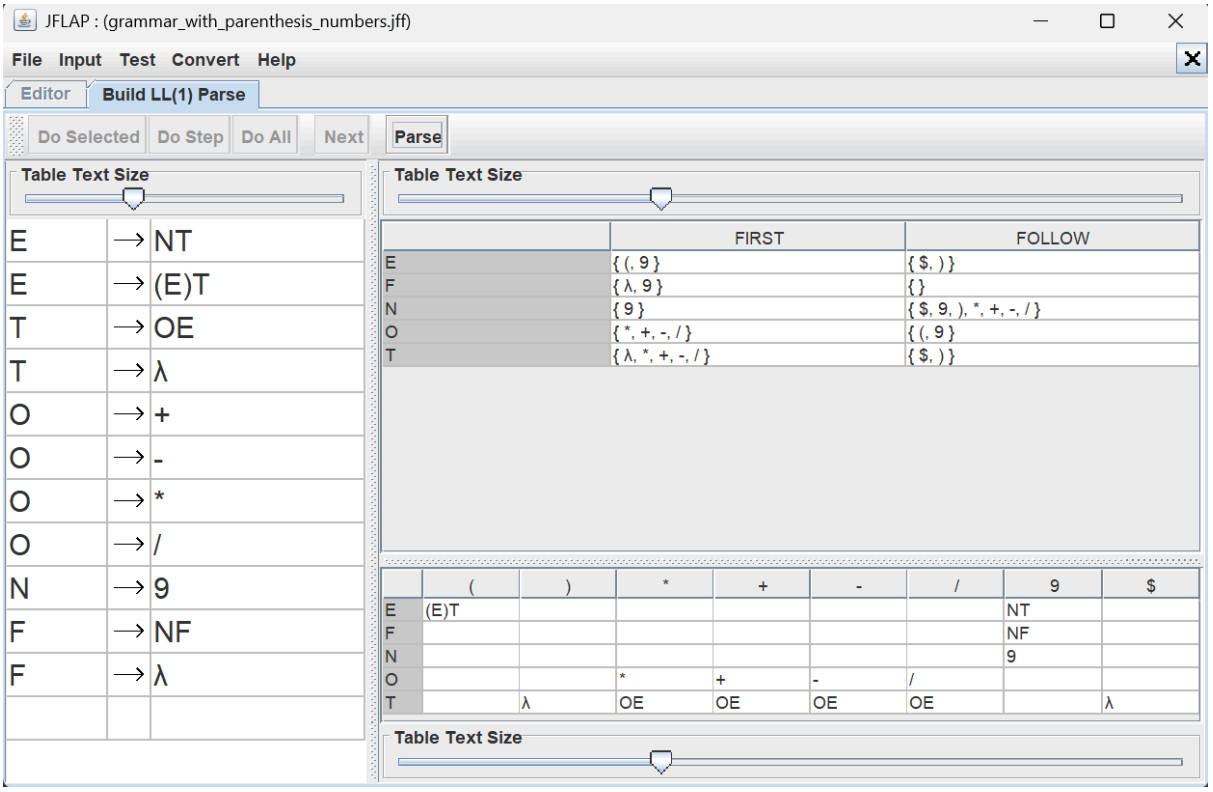
F -> λ | NF

O -> + | - | * | /

N -> 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

*Aclaración*: F = Para dar recursividad a los dígitos/números.

Al probarla en JFLAP nuevamente, cumple con las condiciones LL(1) y obtenemos los siguientes conjuntos PRIMERO y SIGUIENTE:




Pero tras inspeccionar el código, hemos visto que en la gramática no es necesario incluir recursividad para operar con número de más de un dígito, ya que en el código propuesto en drLL.c al leer varios dígitos de forma consecutiva los considera como un único dígito.

Ejemplo:

- Si la entrada es 123, scanf("%d", &tokens.number); lo interpreta directamente como un solo número entero (tokens.number = 123).
- No se necesita recursividad ni una regla gramatical especial para manejar múltiples dígitos, ya que la lectura del número ocurre de una sola vez.

Por lo tanto, en la gramática no es necesario definir reglas recursivas para construir números de más de un dígito, ya que el analizador léxico (rd_lex()) ya los devuelve como una única unidad (T_NUMBER).

A continuación, expandimos la gramática, incluyendo variables y especificaciones adicionales:

A -> E | = VE | V
E -> NT | (E)T
T -> λ | OE
O-> + | - | * | /
N-> [0-9]
V-> [a-z A-z]+

*Aclaración*: V = Variable | A = Axioma