



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:

$E \rightarrow \lambda \mid NE \mid OE$

$O \rightarrow + \mid - \mid * \mid /$

$N \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

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:

$E \rightarrow NT \mid (E)T$

$T \rightarrow \lambda \mid OE$

$O \rightarrow + \mid - \mid * \mid /$

$N \rightarrow 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$

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:

The screenshot shows the JFLAP interface for a grammar. On the left, the grammar rules are listed:

- E \rightarrow NT
- E \rightarrow (E)T
- T \rightarrow OE
- T \rightarrow λ
- O \rightarrow +
- O \rightarrow -
- O \rightarrow *
- O \rightarrow /
- N \rightarrow 9

On the right, the LR(0) item sets are displayed in two tables. The top table shows the FIRST and FOLLOW sets for each non-terminal:

	FIRST	FOLLOW
E	{ (, 9 }	{ \$,) }
N	{ 9 }	{ \$,), *, +, -, / }
O	{ *, +, -, / }	{ (, 9 }
T	{ λ , *, +, -, / }	{ \$,) }

The bottom table shows the LR(0) items and their transitions:

	()	*	+	-	/	9	\$
E	(E)T						NT	
N							9	
O			*	+	-	/		
T		λ	OE	OE	OE	OE		λ

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 \rightarrow NT | (E)T

T \rightarrow λ | OE

F \rightarrow λ | NF

O \rightarrow + | - | * | /

N \rightarrow 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:

The screenshot shows the JFLAP application window titled "JFLAP : (grammar_with_parenthesis_numbers.jff)". The interface includes a menu bar (File, Input, Test, Convert, Help) and a toolbar with buttons for "Do Selected", "Do Step", "Do All", "Next", and "Parse".

On the left, under the "Editor" tab, the grammar rules are listed:

Non-terminal	Production
E	→ NT
E	→ (E)T
T	→ OE
T	→ λ
O	→ +
O	→ -
O	→ *
O	→ /
N	→ 9
F	→ NF
F	→ λ

On the right, under the "Build LR(1) Parse" tab, the LR(0) item sets are displayed in two tables. The top table shows the FIRST and FOLLOW sets for each non-terminal.

	FIRST	FOLLOW
E	{(, 9}	{\$,)}
F	{λ, 9}	{}
N	{9}	{\$, 9,), *, +, -, /}
O	{*, +, -, /}	{(, 9}
T	{λ, *, +, -, /}	{\$,)}

The bottom table shows the LR(0) items and their transitions for each non-terminal.

	()	*	+	-	/	9	\$
E	(E)T						NT	
F							NF	
N							9	
O			*	+	-	/		
T		λ	OE	OE	OE	OE		λ

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

JFLAP : (grammar_with_parenthesis_numbers_variableAsing.jff)

File Input Test Convert Help

Editor Build LL(1) Parse

Do Selected Do Step Do All Next Parse

Table Text Size

E	→	NT
E	→	(E)T
T	→	OE
T	→	λ
O	→	+
O	→	-
O	→	*
O	→	/
N	→	9
A	→	V
A	→	E
A	→	=VE
V	→	a

Table Text Size

	FIRST	FOLLOW
A	{ a, (, 9, = }	{ }
E	{ (, 9 }	{ \$,) }
N	{ 9 }	{ \$,), *, +, -, / }
O	{ *, +, -, / }	{ (, 9 }
T	{ λ, *, +, -, / }	{ \$,) }
V	{ a }	{ (, 9 }

Table Text Size

	()	*	+	-	/	9	=	a	\$
A	E						E	=VE	V	
E	(E)T						NT			
N							9			
O			*	+	-	/				
T		λ	OE	OE	OE	OE				
V									a	λ

Table Text Size