

# UNIVERSIDAD NACIONAL AUTÓNOMA DE MÉXICO

## Facultad de Ciencias

Integrantes:

Adrián Aguilera Moreno

Sebastián Alejandro Gutierrez Medina



Compiladores

## Tarea 02

1. Considera la siguiente gramática:

$$\begin{aligned} E &\rightarrow -E \mid (E) \mid VE' \\ E' &\rightarrow -E \mid \varepsilon \\ V &\rightarrow \text{id}V' \\ V' &\rightarrow (E) \mid \varepsilon \end{aligned}$$

1. Construye la tabla de parsing para un parser tipo LL(1) usando el cálculo de los conjuntos FIRST y FOLLOW que obtuviste en la tarea anterior.

$$\begin{aligned} (0)S &\rightarrow E\$ \\ (1)E &\rightarrow -E \\ (2)E &\rightarrow (E) \\ (3)E &\rightarrow VE' \\ (4)E' &\rightarrow -E \\ (5)E' &\rightarrow \varepsilon \\ (6)V &\rightarrow \text{id}V' \\ (7)V' &\rightarrow (E) \\ (8)V' &\rightarrow \varepsilon \end{aligned}$$

Table 1: Tabla de Parsing

	-	(	)	id	\$
S	(0)	(0)		(0)	
E	(1)	(2)		(3)	
E'	(4)		(5)		(5)
V				(6)	
V'	(8)	(7)	(8)		(8)

2. Muestra lo que se obtiene al ejecutar el algoritmo para procesar la cadena  $-\text{id}(-\text{id})-\text{id}$ . Incluye una tabla para ver el progreso del algoritmo donde se muestre el avance del procesamiento de la cadena y la evolución de la pila del parser.

Table 2: Tabla del progreso del algoritmo

Pila del Parser	Cadena	Regla aplicada
\$S	$-id(-id) - id\$$	
\$E	$-id(-id) - id\$$	$S \rightarrow E$
\$E-	$-id(-id) - id\$$	$E \rightarrow -E$
\$E	$id(-id) - id\$$	
\$E'V	$id(-id) - id\$$	$E \rightarrow VE'$
\$E'V'id	$id(-id) - id\$$	$V \rightarrow idV'$
\$E'V'	$(-id) - id\$$	
\$E')E(	$(-id) - id\$$	$V' \rightarrow (E)$
\$E')E	$-id) - id\$$	
\$E')E-	$-id) - id\$$	$E \rightarrow -E$
\$E')E	$id) - id\$$	
\$E')E'V	$id) - id\$$	$E \rightarrow VE'$
\$E')E'V'id	$id) - id\$$	$V \rightarrow idV'$
\$E')E'V'	$) - id\$$	
\$E')E'	$) - id\$$	$V' \rightarrow \epsilon$
\$E')	$) - id\$$	$E' \rightarrow \epsilon$
\$E'	$-id\$$	
\$E-	$-id\$$	$E' \rightarrow -E$
\$E	$id\$$	
\$E'V	$id\$$	$E \rightarrow VE'$
\$E'V'id	$id\$$	$V \rightarrow idV'$
\$E'V'	$\$$	
\$E'	$\$$	$V' \rightarrow \epsilon$
$\$$	$\$$	$E' \rightarrow \epsilon$
$\$$	$\$$	ACEPTADA

2. Considera el siguiente fragmento de una gramática que abstrae el comportamiento de expresiones del lenguaje C:

$$E \rightarrow *E \mid \&E \mid E = E \mid E -> E \mid id$$

Esta gramática es ambigua pero se puede transformar en una no-ambigua usando la precedencia de operadores. En particular, el acceso a campos de una estructura,  $E -> E$ , tiene mayor precedencia que la derreferenciación y las expresiones para direcciones; además, estas tres tienen mayor precedencia que las asignaciones <sup>1</sup>

1. Escribe una gramática equivalente tipo **LL(1)** que incluya la precedencia descrita, muestra el proceso o describe las técnicas que uses para obtener esta nueva gramática.

**Solución:** A continuación se muestra la gramática con precedencia (solo un reacomodo) y con ambigüedad:

$$E \rightarrow E -> E \mid *E \mid \&E \mid E = E \mid id$$

quitando ambigüedades de derecha a izquierda tenemos que

$$\begin{aligned} E &\rightarrow E -> P \mid P \\ P &\rightarrow *P \mid \&P \mid P = P \mid id \end{aligned}$$

<sup>1</sup>Puedes revisar la tabla disponible en esta página para consultar la precedencia y asociatividad de los operadores en C.

ahora, quitemos las ambigüedades de izquierda a derecha, esto es

$$\begin{aligned} E &\rightarrow E -> P \mid P \\ P &\rightarrow *A \mid \&A \mid A = P \mid A \\ A &\rightarrow id \end{aligned}$$

lo anterior con base a Eliminación de ambigüedad.

Quitando la recursividad por la izquierda tenemos que

$$\begin{aligned} (0) \ E &\rightarrow PE' \mid P \\ (1) \ E' &\rightarrow -> P \\ (2) \ P &\rightarrow *A \mid \&A \mid A = P \mid A \\ (3) \ A &\rightarrow id \end{aligned}$$

La gramática anterior es no recursiva por la izquierda y no ambigua.

- Muestra la tabla de parsing para la gramática del inciso anterior.

**Solución:** Ahora construyamos la tabla parsing, esto es

Table 3: FIRST y FOLLOW

	FIRST	FOLLOW
E	*, &, id	=
E'	->	=
P	*, &, id	=
A	id	=

luego, calculemos la tabla de parsing, a continuación se muestra Así, finalizamos este inciso.

Table 4: Tabla de Parsing

	->	*	&	=	id
E		(0)	(0)		(0)
E'	(1)				
P		(2)	(2)		(2)
A	(3)				

- Procesa la expresión  $* * a -> b -> c = \& * d$  usando el algoritmo para **LL** y mostrando los estados del parser.