Ejercicio 1

Dada la siguiente gramática, construya la tabla M(N,T) y haga el análisis predictivo para la cadena s;s;s:

Gramática

sec-sent → sent sec-sent'
sec-sent' → **;** sec-sent
sec-sent' → ***e***
sent → **s**

Primeros

Primeros(sec-sent)={s}
Primeros(sec-sent')={;, e}
Primeros(sent)={s}

Siguientes

Siguientes(sec-sent)={$}
Siguientes(sec-sent')={$}
Siguientes(sent)={;, $}

	;	**s**	**$**
sec-sent		sec-sent → sent sec-sent'	extraer
sec-sent'	sec-sent' → ; sec-sent		sec-sent' → *e*
sent	extraer	sent → s	extraer

Stack	**Entrada**	**Acción**
$ sec-sent	s; s; s $	sec-sent → sent sec-sent'
$ sec-sent' sent	s; s; s $	sent → s
$ sec-sent' s	s; s; s $	match (s)
$ sec-sent'	; s; s $	sec-sent → ; sec-sent
$ sec-sent ;	; s; s $	match (;)
$ sec-sent	s; s $	sec-sent → sent sec-sent'
$ sec-sent' sent	s; s $	sent → s
$ sec-sent' s	s; s $	match (s)
$ sec-sent'	; s $	sec-sent → ; sec-sent
$ sec-sent ;	; s $	match (;)
$ sec-sent	s $	sec-sent → sent sec-sent'
$ sec-sent' sent	s $	sent → s
$ sec-sent' s	s $	match (s)

$ sec-sent'	$	sec-sent' → *e*
$	$	**Aceptar**

Ejercicio 2

Dada la siguiente gramática, construya la tabla M(N,T) y haga el análisis predictivo para la cadena if(0) if (1) otro else otro:

Gramática

sentencia → sent-if
sentencia → **otro**
sent-if → **if (**exp**)** sentencia parte-else
parte-else → **else** sentencia
parte-else → ***e***
exp → **0**
exp → **1**

Primeros

Primeros(**sentencia**) = {otro, if}
Primeros(**sent-if**) = {if}
Primeros(**parte-else**) = {else, e}
Primeros(**exp**) = {0, 1}

Siguientes

Siguientes(**sentencia**) = {$, else}
Siguientes(**sent-if**) = {$, else}
Siguientes(**parte-else**) = {$, else}
Siguientes(**exp**) = {)}

La gramática no es LL(1)

	otro	**if**	**(**	**)**	**else**	**0**	**1**	**$**
sentencia	sentencia → otro	sentencia → sent-if			extraer			extraer
sent-if		sent-if →if (exp) sentencia parte-else			extraer			extraer
parte-else					parte-else → else sentencia parte-else → *e*			parte-else →*e*
exp				extraer		exp → 0	exp → 1	

Stack	**Entrada**	**Acción**
$ sentencia	if(0) if(1) otro else otro$	sentencia → sent-if
$ sent-if	if(0) if(1) otro else otro $	sent-if → if(exp) sentencia parte-else
$ parte-else sentencia) exp (if	if(0) if(1) otro else otro $	match (if)
$ parte-else sentencia) exp (	(0) if(1) otro else otro $	match (()
$ parte-else sentencia) exp	0) if(1) otro else otro $	exp → 0
$ parte-else sentencia) 0	0) if(1) otro else otro $	match (0)
$ parte-else sentencia)	) if(1) otro else otro $	match ())
$ parte-else sentencia	if(1) otro else otro $	sentencia → sent-if
$ parte-else parte-else sentencia) exp (if	if(1) otro else otro$	sent-if → if(exp) sentencia parte-else
$ parte-else parte-else sentencia) exp (if	if(1) otro else otro $	match (if)
$ parte-else parte-else sentencia) exp (	(1) otro else otro $	match(()
$ parte-else parte-else sentencia) exp	1) otro else otro $	exp → 1
$ parte-else parte-else sentencia) 1	1) otro else otro $	match (1)
$ parte-else parte-else sentencia)	) otro else otro $	match())

$ parte-else parte-else sentencia	otro else otro $	sentencia → otro
$ parte-else parte-else otro	otro else otro $	match(otro)
$ parte-else parte-else	else otro $	parte-else → else sentencia
$ parte-else sentencia else	else otro $	match (else)
$ parte-else sentencia	otro $	sentencia → otro
$ parte-else otro	otro $	match (otro)
$ parte-else	$	parte-else → *e*
$	$	**Aceptar**

Ejercicio 3

Dada la siguiente gramática, construya la tabla M(N,T) y comente si la gramática LL(1).

Gramática

S → L
L → L **#** X
L → X
X → X **#**
X → ***e***

Primeros

Primeros(S)={#, *e*}
Primeros(L)={#, e}
Primeros(X)={#, e}

Siguientes

Siguientes(S)={$}
Siguientes(L)={$, #}
Siguientes(X)={$, #}

	#	**$**
S	S → L	S → L
L	L → L # X L → X	L → X
X	X → X # X → *e*	X → *e*

La gramática no es LL(1), se puede verificar puesto que:
- e ϵ Primeros(L) ⋀ Primeros(L) ⋂ Siguientes(L) ≠ ∅
- e ϵ Primeros(X) ⋀ Primeros(X) ⋂ Siguientes(X) ≠ ∅

Extra:

Análisis predictivo para la cadena ###

Stack	**Entrada**	**Acción**
$ S	### $	S → L
$ L	### $	L → L # X
$ X # L	### $	L → L # X
$ X # X # L	### $	L → X
$ X # X # X	### $	X → X #
$ X # X # # X	### $	X → e
$ X # X # #	### $	match (#)
$ X # X #	## $	match (#)
$ X # X	# $	X → e
$ X #	# $	match (#)
$ X	$	X → e
$	$	**Aceptar**

Ejercicio 4

Determine si la siguiente gramática es LL(1)

Gramática	Primeros	Siguientes	Observación
A → **a** A **a** | ***e***	Primeros(A)={a, *e*}	Siguientes(A)={$, a}	La gramática no es LL(1), se puede verificar puesto que: - a ϵ Primeros(A) ⋀ Primeros(A) ⋂ Siguientes(A) ≠ ∅

	a	**$**
A	A → a A a A → *e*	A → *e*

Basándonos en el pseudocódigo de la sesión 3.7 (diapositiva 7/28), el código en C++ solo será exitoso cuando tengamos una gramática LL(1) sin errores en la construcción. Como hay colisión, no se podrá llegar a un parseo correcto. De esta manera, la única cadena correctamente analizada será la vacía.

Link del archivo cpp: https://drive.google.com/file/d/1UCvzo8zNkE_v3_99g-QICKJcA_KjyEFM/view?usp=sharing

Ejercicio 5

Dada la siguiente gramática: elimine la recursividad, factorice por izquierda y construya la tabla M(N,T).

Gramática

E → EE**+**
E → EE*****
E → **num**

Eliminación de la recursividad

E → **num** F
F → E*****F | E**+**F | ***e***

Factorizacion por izquierda

E → **num** F
F → E op F
F → ***e***
op → ***** | **+**

Primeros

Primeros(E)={num}
Primeros(F)={num, *e*}
Primeros(op)={*, +}

Siguientes

Siguientes(E)={$, *, +}
Siguientes(F)={$, *, +}
Siguientes(op)={$, *, +, num}

	+	*****	**num**	**$**
E	extraer	extraer	E → num F	extraer
F	F → *e*	F → *e*	F → E op F	F → *e*
op	op → +	op → *	extraer	extraer

Extra:

Análisis predictivo para la cadena 2 3 4 +*

Stack	**Entrada**	**Acción**
$ E	2 3 4 + * $	E → num F
$ F num	2 3 4 + * $	match (num)
$ F	3 4 + * $	F → E op F
$ F op E	3 4 + * $	E → num F
$ F op F num	3 4	match (num)
$ F op F	4 + * $	F → E op F
$ F op F op E	4 + * $	E → num F
$ F op F op F num	4 + * $	match (num)
$ F op F op F	+ * $	F → *e*
$ F op F op	+ * $	op → +

$ F op F +	+ * $	match (+)
$ F op F	* $	F → *e*
$ F op	* $	op → *
$ F *	* $	match (*)
$ F	$	F → *e*
$	$	**Aceptar**

Ejercicio 5

Hacer la comparación de la gramática corregida y sin corregir. Hacer la tabla LL y una tabla de pila.

Gramática

S → S**a** | **a**

Eliminación de la recursividad

S → **a**S'
S' → **a**S' | ***e***

Primeros

Primeros(S)={a}

Primeros(S)={a}
Primeros(S')={a, *e*}

Siguientes

Siguientes(S)={$, a}

Siguientes(S)={$}
Siguientes(S')={$}

	a	**$**
S	S → a	extraer

	a	**$**
S	S → aS'	extraer
S'	S → aS'	S → *e*

Análisis predictivo para la cadena aaa

Stack	**Entrada**	**Acción**
$ S	aaa $	S → Sa
$ a S	aaa $	S → Sa
$ a a S	aaa $	S → Sa
$ a a a	aaa $	S → a
$ a a a	aaa $	match (a)
$ a a	aa $	match (a)
$ a	a $	match (a)
$	$	**Aceptar**

Análisis predictivo para la cadena aaa

Stack	**Entrada**	**Acción**
$ S	aaa $	S → aS'
$ S' a	aaa $	match (a)
$ S'	aa $	S' → aS'
$ S' S' a	aa $	match (a)
$ S' S'	a $	S' → aS'
$ S' S' S' a	a $	match (a)
$ S' S' S'	$	S' → *e*
$ S' S'	$	S' → *e*
$ S'	$	S' → *e*
$	$	**Aceptar**

Ejercicio extra, basado en el 4.9 del Louden

Gramática

lexp → atom
lexp → list
atom → **number**
atom → **identifier**
list → **(**lexp-seq**)**
lexp-seq → lexp lexp-seq'
lexp-seq' → **,** lexp-seq
lexp-seq' → ***e***

Primeros

Primeros(lexp)={number, identifier, (}
Primeros(atom)={number, identifier}
Primeros(list)={(}
Primeros(lexp-seq)={number, identifier, (}
Primeros(lexp-seq')={,, *e*}

Siguientes

Siguientes(lexp)={$, ,,)}
Siguientes(atom)={$, ,,)}
Siguientes(list)={$, ,,)}
Siguientes(lexp-seq)={)}
Siguientes(lexp-seq')={)}

	number	**identifier**	**(**	**)**	**,**	**$**
lexp	lexp → atom	lexp → atom	lexp → list	extraer	extraer	extraer
atom	atom → number	atom → identifier		extraer	extraer	extraer
list			list → (lexp-seq)	extraer	extraer	extraer
lexp-sep	lexp-seq → lexp lexp-seq'	lexp-seq → lexp lexp-seq'	lexp-seq → lexp lexp-seq'	extraer		
lexp-sep'				lexp-seq' → *e*	lexp-seq' → , lexp-seq	

Extra:

Análisis predictivo para la cadena (12, v)

Stack	**Entrada**	**Acción**
$ lexp	(12, v) $	lexp → list
$ list	(12, v) $	list → (lexp-seq)
$) lexp-seq (	(12, v) $	match (()
$) lexp-seq	12, v) $	lexp-seq → lexp lexp-seq'
$) lexp-seq' lexp	12, v) $	lexp → atom
$) lexp-seq' atom	12, v) $	atom → number
$) lexp-seq' number	12, v) $	match (number)
$) lexp-seq'	, v) $	lexp-seq' → , lexp-seq
$) lexp-seq ,	, v) $	match (,)
$) lexp-seq	v) $	lexp-seq → lexp lexp-seq'
$) lexp-seq' lexp	v) $	lexp → atom
$) lexp-seq' atom	v) $	atom → identifier
$) lexp-seq' identifier	v) $	match (identifier)
$) lexp-seq'	) $	lexp-seq' → *e*

$)	) $	match ())
$	$	**Aceptar**

Ejercicio extra, basado en el 4.10 del Louden

Gramática

declaration → type var-list
type → **int**
type → **float**
var-list → **identifier** var-list'
var-list' → **,** var-list
var-list' → ***e***

Primeros

Primeros(declaration)={int, float}
Primeros(type)={int, float}
Primeros(var-list)={identifier}
Primeros(var-list')={,, e}

Siguientes

Siguientes(declaration)={$}
Siguientes(type)={identifier}
Siguientes(var-list)={$}
Siguientes(var-list')={$}

	int	**float**	**identifier**	**,**	**$**
declaration	declaration → type var-list	declaration → type var-list			extraer
type	type → int	type → float	extraer		
var-list			var-list → identifier var-list'		extraer
var-list'				var-list' → , var-list	var-list' → e

Extra:

Análisis predictivo para la cadena int v1, v2, v3

Stack	**Entrada**	**Acción**
$ declaration	int v1, v2, v3 $	declaration → type var-list
$ var-list type	int v1, v2, v3 $	type → int
$ var-list int	int v1, v2, v3 $	match (int)
$ var-list	v1, v2, v3 $	var-list → identifier var-list'
$ var-list' identifier	v1, v2, v3 $	match (identifier)
$ var-list'	, v2, v3 $	var-list' → , var-list
$ var-list ,	, v2, v3 $	match (,)
$ var-list	v2, v3 $	var-list → identifier var-list'
$ var-list' identifier	v2, v3 $	match (identifier)
$ var-list'	, v3 $	var-list' → , var-list
$ var-list ,	, v3 $	match (,)
$ var-list	v3 $	var-list → identifier var-list'
$ var-list' identifier	v3 $	match (identifier)
$ var-list'	$	var-list' → *e*
$	$	**Aceptar**