Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 2.3

Ejercicios

ALUMNO: Carlos Alberto Sánchez Lara
Dr. Eduardo Cornejo-Velázquez

1. Introducción

Los analisis sintacticos son algo completamente necesario en la creacion de compiladores y juega un papel clave en identificacion de la estructura gramatical de un lenguaje de programacion, por eso en este reporte daremos resolucion a 14 problemas que se centran en el tema de analisis sintactico.

2. Marco teórico

Análisis sintactico

[?]La principal tarea de un analizador sintáctico es indicar si la secuencia de componentes léxicos, encontrados en el análisis léxico, están en el orden correspondiente a las reglas gramaticales del lenguaje

Gramaticas libres de contexto

[?]Una de las maneras de implementar analizadores léxicos es utilizando gramáticas libres de contexto. Una gramática libre de contexto es una especificación para la estructura sintáctica de un lenguaje de programación (Louden, 2004).

Árbol de análisis sintáctico

[?]Un árbol de análisis sintáctico correspondiente a una derivación, es un árbol etiquetado en el cual los nodos interiores están etiquetados por no terminales, es decir, un árbol sintáctico muestra de forma gráfica la manera en que el símbolo inicial de una gramática deriva a una secuencia de componentes.

Tabla de analisis sintactico

[?]Una tabla de análisis sintáctico es esencialmente un arreglo bidimensional indizado por no terminales y terminales que contienen opciones de producción a emplear en el paso apropiado del análisis sintáctico (incluyendo el signo monetario $ para representar el final de la entrada) (Louden, 2004).

3. Herramientas empleadas

1. Latex. Es una herramienta de edicion de texto que maneja una interfaz muy parecida a la ocupada en los lenguajes de programacion, usado normalmente en trabajos formales.

4. Desarrollo

Ejercicio 1

1. a) Escriba una gramática que genere el conjunto de cadenas s; , s;s; , s;s;s; , ...
b) Genere un árbol sintáctico para la cadena s;s;

Conjunto de no terminales (N): $\{S\}$
Conjunto de terminales (sum): $\{s, ;\}$
Conjunto de reglas de producción (P):{ S → s; , S → S s; }
Simbolo inicial (S):{ S }
Gramatica generada: S → S s; , S → s;

Árbol sintactico

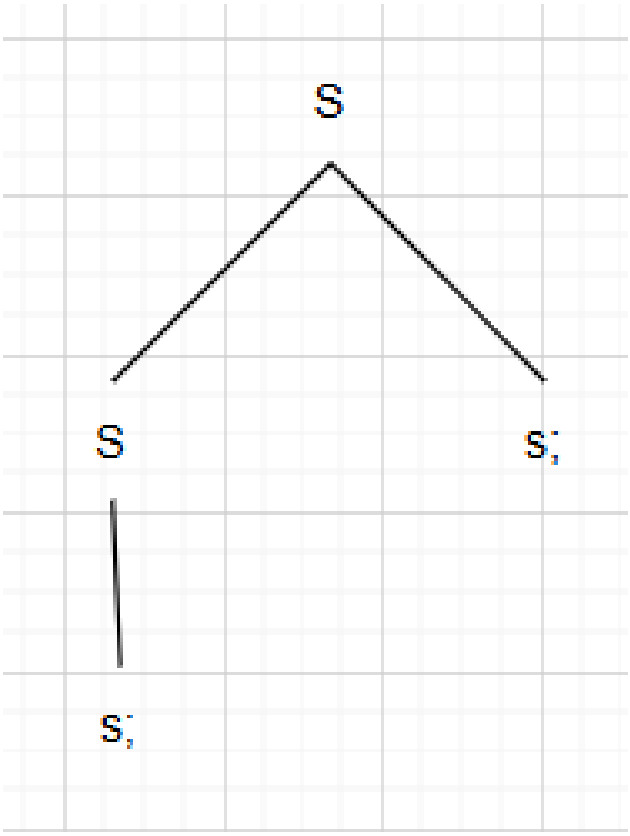


Figure 1: Arbol sintactico

Ejercicio 2

2. Considere la siguiente gramática:
rexp → rexp "|" rexp
|rexp rexp
|rexp "*"
|"(" rexp ")"
|letra
a) Genere un árbol sintáctico para la expresión regular (ab|b)*.

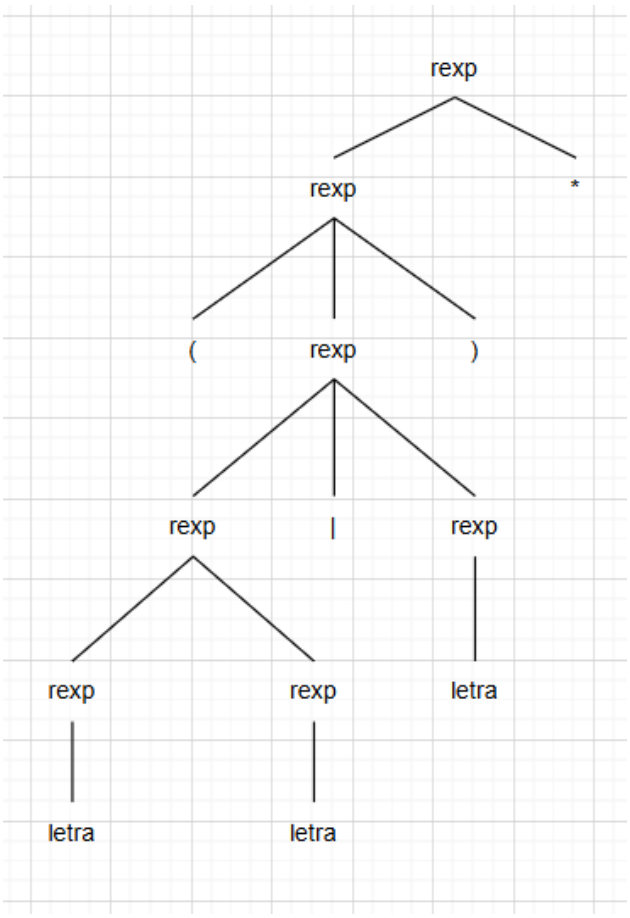


Figure 2: Enter Caption

Ejercicio 3

3. De las siguientes gramáticas, describa el lenguaje generado por la gramática y genere árboles sintácticos con las respectivas cadenas.
a) S → S S + | S S * | a con la cadena aa+a*.
b) S → 0 S 1 | 0 1 con la cadena 000111.
c) S → + S S | * S S | a con la cadena + * aaa

a) Descripcion del lenguaje generado:

Esta gramatica es capaz de generar una suma de cadenas a, o multiplicacion de cadenas, y ocupa los operadores * y +

Arbol sintactico generado:

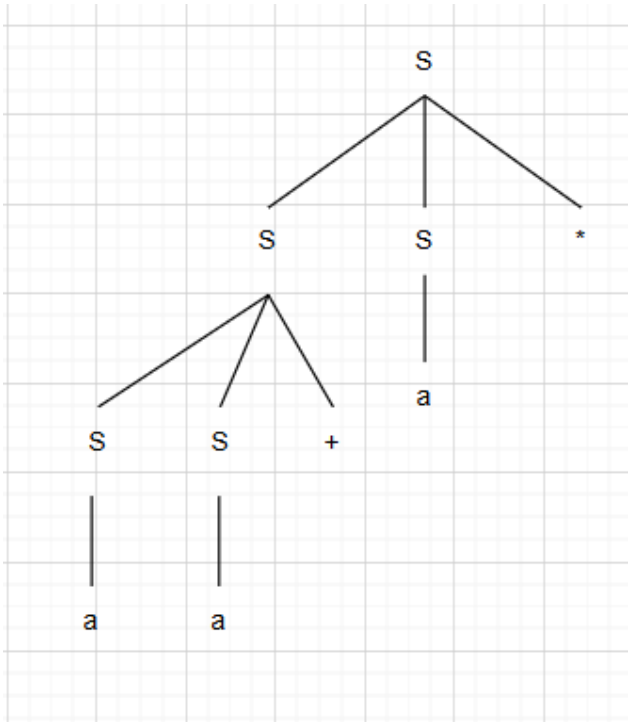


Figure 3: Arbol sintactico

b) Descripcion del lenguaje generado:

Es un lenguaje capaz de crear cadenas que comienzan con ceros y terminan con unos, y la cantidad de n 0 es igual a la cantidad de m 1

Arbol sintactico generado:

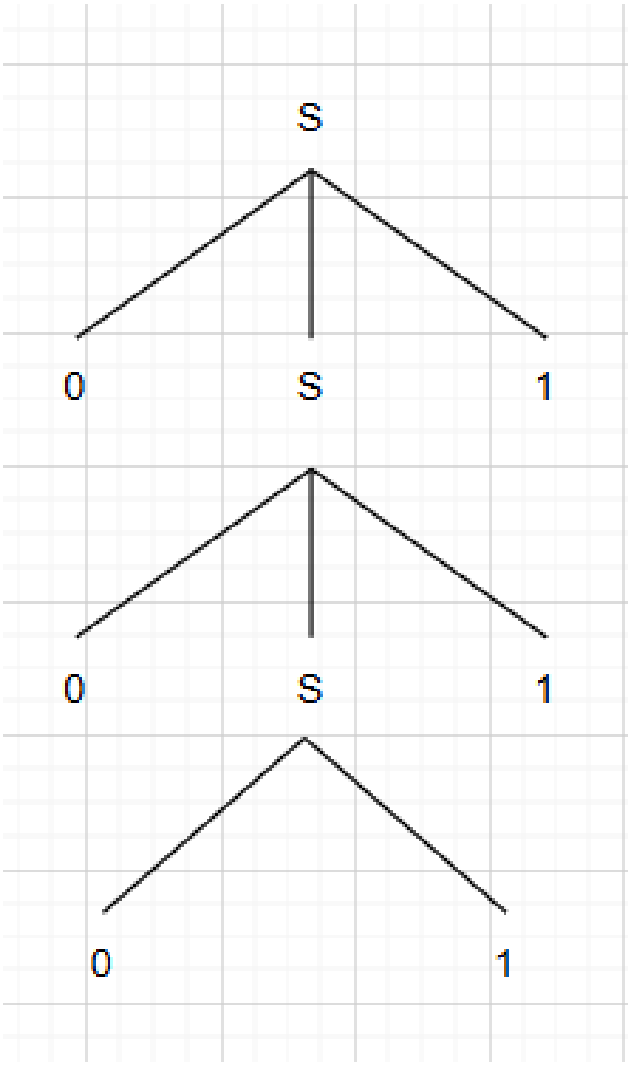


Figure 4: Arbol sintactico

c) Descripcion del lenguaje generado:

Este lenguaje produce cadenas donde los operadores + y * pueden comenzar o ir seguidos de una letra a

Arbol sintactico generado:

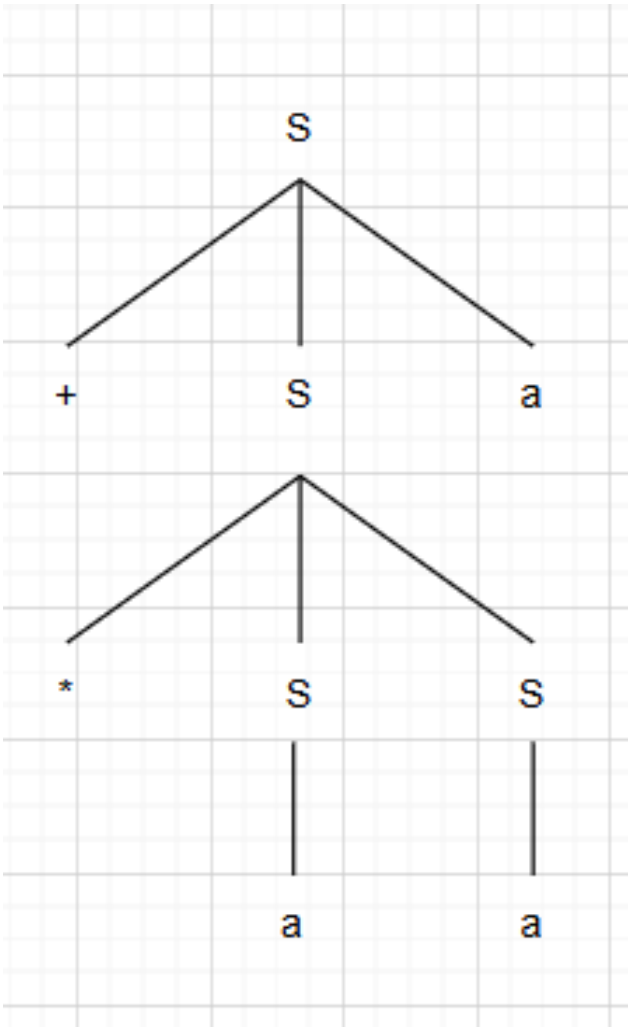


Figure 5: Arbol sintactico

Ejercicio 4

4. ¿Cuál es el lenguaje generado por la siguiente gramática? S → xSy |ϵ

El grámatica puede generar cadenas del tipo
L:$\{\lambda, xy, xxyy, ...\}$
donde comenzaran con x y terminaran con y, ademas de que siempre sera igual la cantidad de x e y.

Ejercicio 5

Genere el árbol sintáctico para la cadena zazabzbz utilizando la siguiente gramática:
S → zMNz
M → aNa
N → bNb
N → z

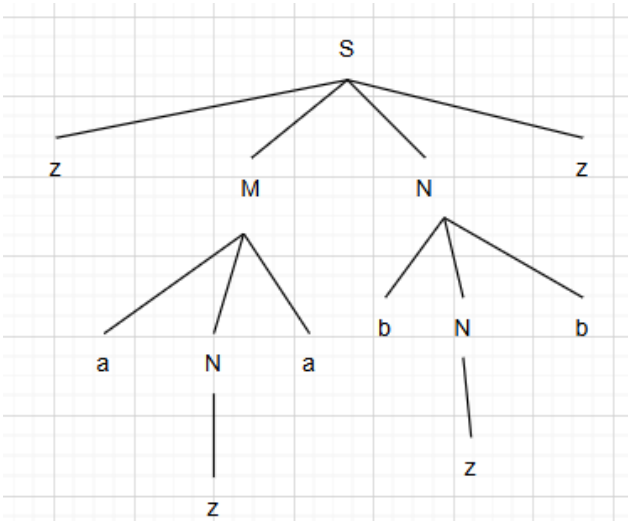
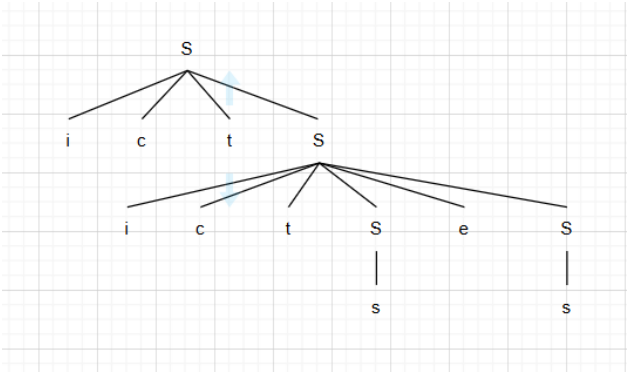


Figure 6: Arbol sintactico

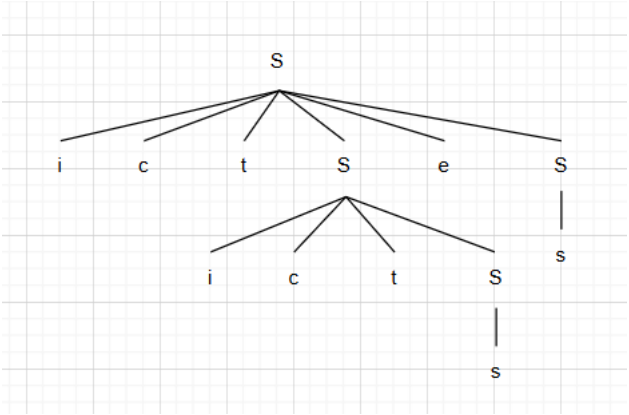
Ejercicio 6

6. Demuestre que la gramática que se presenta a continuación es ambigua, mostrando que la cadena ictictses tiene derivaciones que producen distintos árboles de análisis sintáctico.
S → ictS
S → ictSeS
S → s

A) Árbol 1
S→ ictS
ictS→ictictSeS
ictictSeS→ictictseS
ictictseS→ictictses



B) Arbol 2
ictSeS
ictSeS → ictictSeS
ictictSeS → ictictseS
ictictseS → ictictses



Ejercicio 7

7. Considere la siguiente gramática
S → (L) | a
L → L , S | S
Encuéntrense árboles de análisis sintáctico para las siguientes frases:
a) (a, a)
b) (a, (a, a))
c) (a, ((a, a), (a, a)))

Primer arbol:

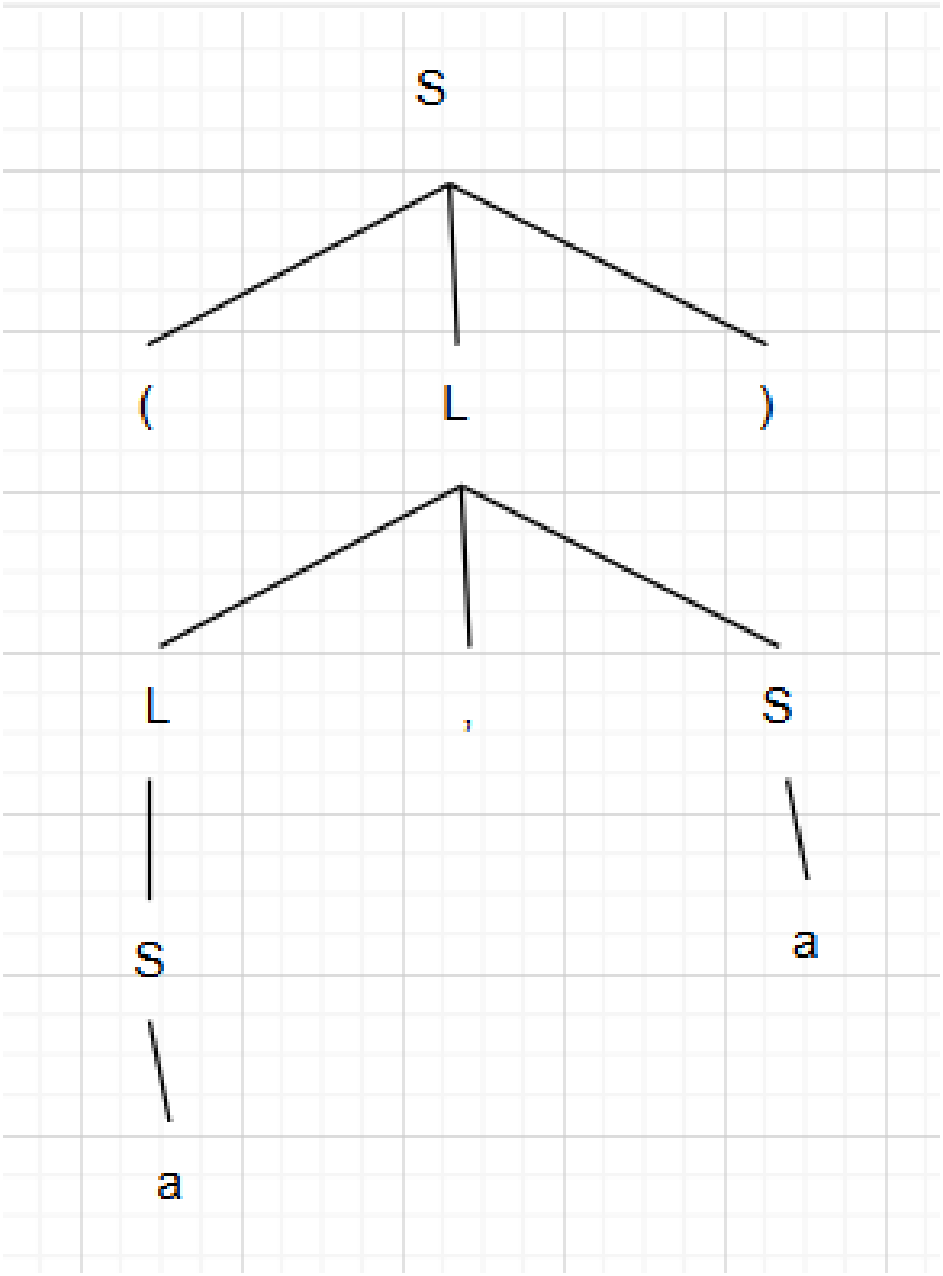
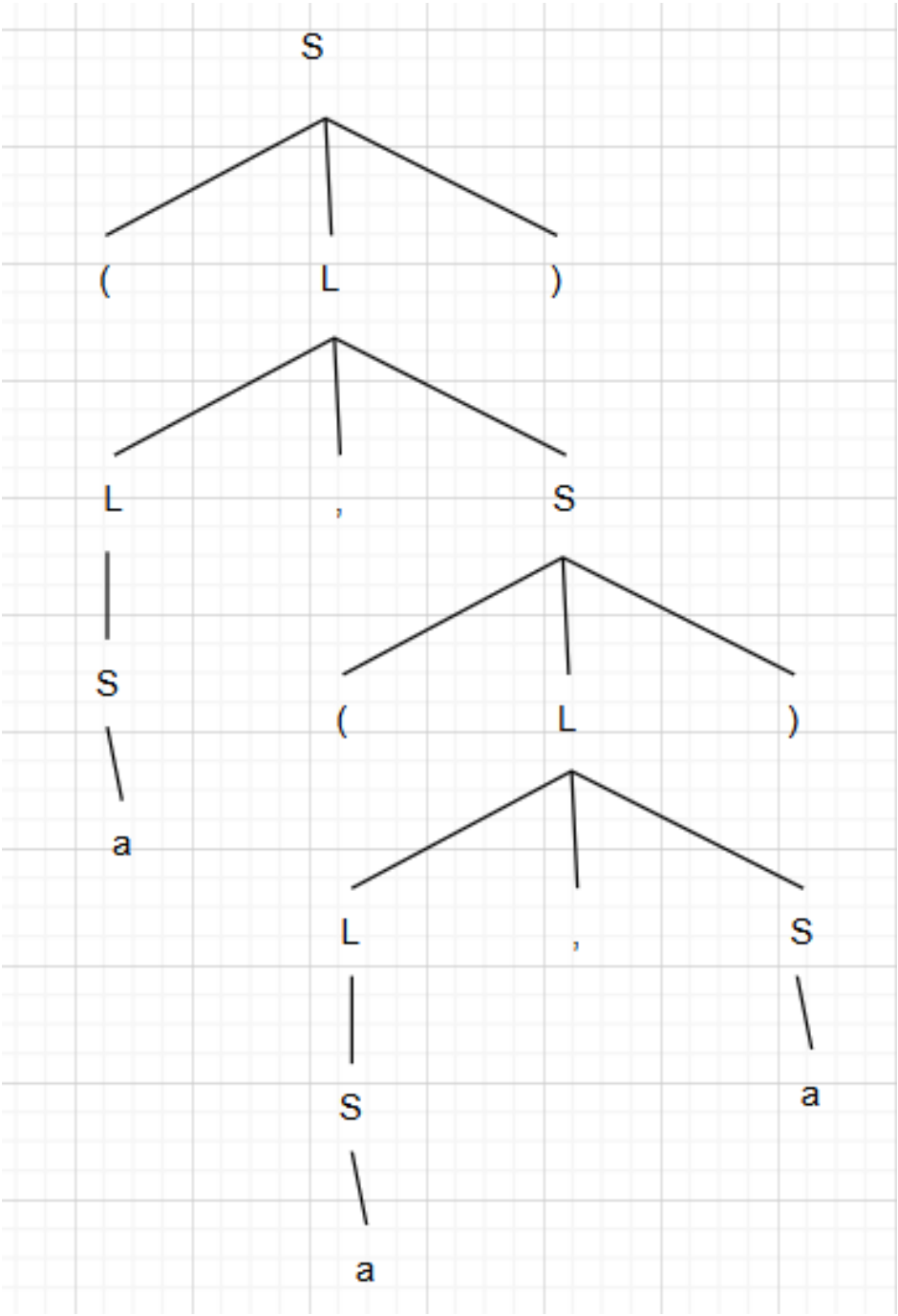
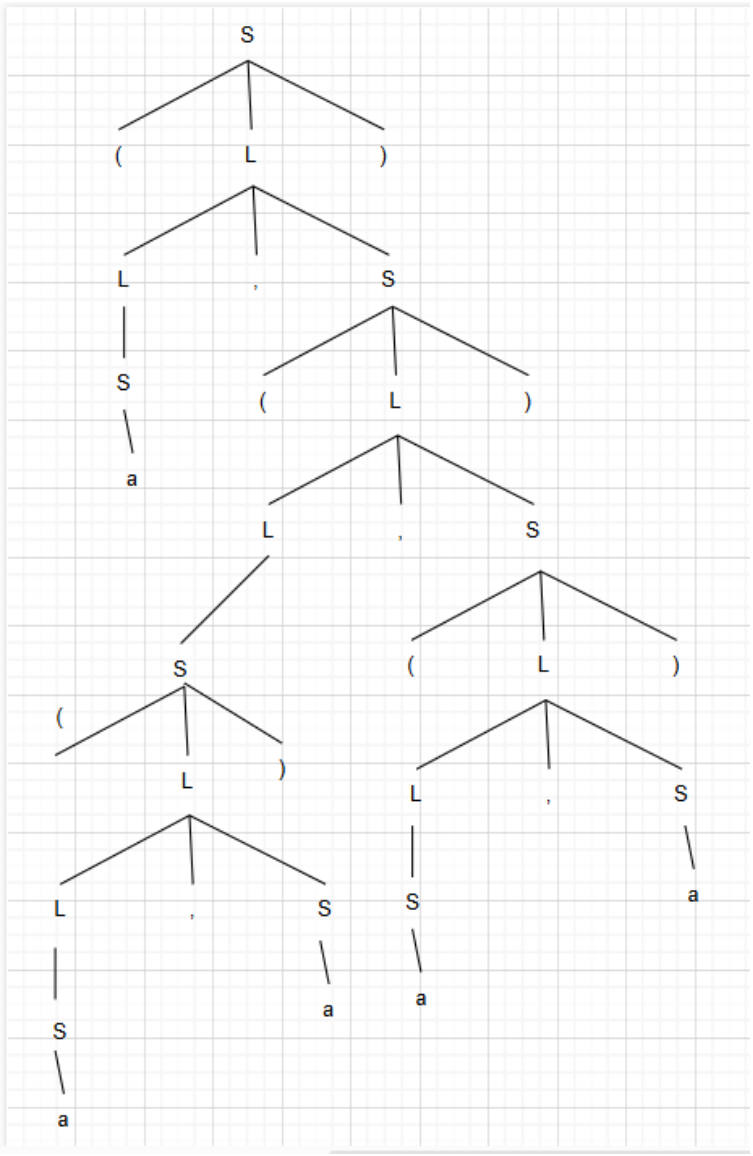


Figure 7: Arbol Sintactico

Segundo arbol:

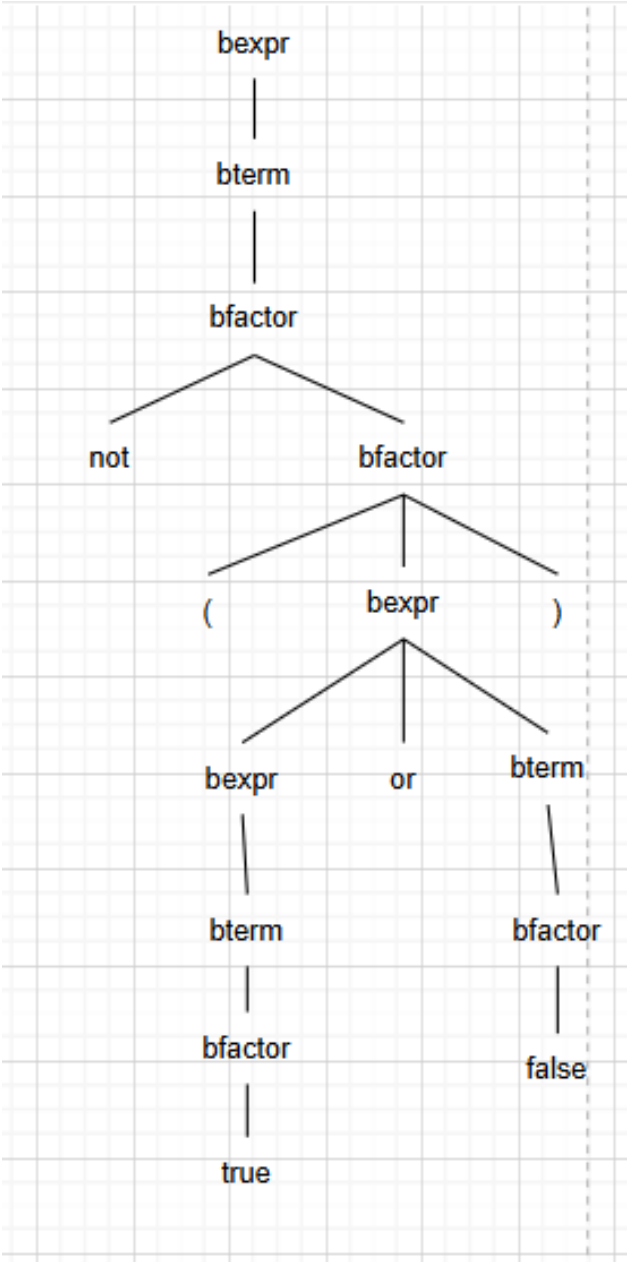


Tercer arbol:



Ejercicio 8

8. Constrúyase un árbol sintáctico para la frase not (true or false) y la gramática:
bexpr → bexpr or bterm | bterm
bterm → bterm and bfactor | bfactor
bfactor → not bfactor | (bexpr) | true | false



Ejercicio 9

9. Diseñe una gramática para el lenguaje del conjunto de todas las cadenas de símbolos 0 y 1 tales que todo 0 va inmediatamente seguido de al menos un 1.

Conjunto de no terminales(N): {S}
Conjunto de terminales($\sum$) : $\{0, 1\}$
$Conjuntodereglasdeproduccion(P)$:{ S $\rightarrow$ 1S; , S $\rightarrow$ 01S; , S $\rightarrow \epsilon$; }
Simbolo inicial(S): {S}

Gramatica generada:
– S $\rightarrow$ 1S;
– S $\rightarrow$ 01S;
– S $\rightarrow \epsilon$;

Ejercicio 10

10.Elimine la recursividad por la izquierda de la siguiente gramática:
S $\rightarrow$ (L) | a
L $\rightarrow$ L , S | S
De la gramatica solo L $\rightarrow$ L , S | S tiene recursividad, por eso tomamos esa regla y la reescribimos de la siguiente manera para eliminar la recursividad:
L $\rightarrow$ L, S | S
L $\rightarrow$ SL
L $\rightarrow$, SL| ϵ

Gramatica obtenida:
S $\rightarrow$ (L) | a
L $\rightarrow$ SL
L $\rightarrow$, SL| ϵ

Ejercicio 11

11. Dada la gramática S $\rightarrow$ (S) | x, escriba un pseudocódigo para el análisis sintáctico de esta gramática mediante el método descendente recursivo.

Para realizar el análisis sintáctico descendente recursivo de la gramática S$\rightarrow$(S)x, se inicia la posición en el primer carácter de la cadena de entrada y se llama a la función S. Si el carácter actual es un paréntesis de apertura (, se avanza a la siguiente posición, se llama recursivamente a S y, si el siguiente carácter es un paréntesis de cierre), se avanza nuevamente; de lo contrario, se muestra un error de sintaxis. Si el carácter actual es x, se avanza a la siguiente posición. Si el carácter no es ni (ni x, también se muestra un error. Al finalizar, si se ha llegado al final de la cadena, la entrada es aceptada; en caso contrario, se indica un error de sintaxis.

Ejercicio 12

12. Qué movimientos realiza un analizador sintáctico predictivo con la entrada (id+id)*id, mediante el algoritmo 3.2, y utilizándose la tabla de análisis sintáctico de la tabla 3.1. (Tómese como ejemplo la Figura 3.13).

Algoritmo 3.2: Análisis sintáctico predictivo, controlado por una tabla. (Aho, Lam, Sethi, & Ullman, 2008, pág. 226)

Entrada: Una cadena w y una tabla de análisis sintáctico M para la gramática G.

Salida: Si w está en el lenguaje de la gramática $L(G)$, una derivación por la izquierda de w; de lo contrario, una indicación de error.

Método: Al principio, el analizador sintáctico se encuentra en una configuración con w\$ en el búfer de entrada, y el símbolo inicial S de G en la parte superior de la pila, por encima de \$.

establecer *ip* para que apunte al primer símbolo de *w*;
establecer *X* con el símbolo de la parte superior de la pila;
while ($X \neq \$$) { /* mientras la pila no está vacía*/
  if (*X* es *a*) extraer de la pila y avanzar *ip*; /* *a*=símbolo al que apunta *ip* */
  else if (*X* es un terminal) *error*( )
    else if (*M*[*X*, *a*] es una entrada de error) *error*( )
      else if ($M[X, a] = X \rightarrow Y_1 Y_2 \ldots Y_k$) {
        enviar de salida la producción $X \rightarrow Y_1 Y_2 \ldots Y_k$;
        extraer de la pila;
        meter $Y_k, Y_{k-1}, \ldots, Y_1$ en la pila, con Y_1 en la parte superior;
        }
  establecer X con el símbolo de la cima de la pila;
}

Figure 8: Tabla de analisis sintactico

No terminal	Símbolo de entrada					
	id	+	*	(	)	$
E	E → TE '			E → TE '		
E '		E ' → +TE '			E ' → ε	E ' → ε
T	T → FT '			T → FT '		
T '		T ' → ε	T ' → *FT '		T ' → ε	T ' → ε
F	$F \rightarrow$ id			$F \rightarrow (E)$		

Pila	Entrada	Acción
$E	(id + id)*id$	E → T E′
$E’T	(id + id)*id$	T → F T′
$E’T’F	(id + id)*id$	F → (E)
$E’T’)E(	(id + id)*id$	”(” concuerda
$E’T’)E	id + id)*id$	E → T E′
$E’T’)E’T	id + id)*id$	T → F T′
$E’T’)E’T’F	id + id)*id$	F → id
$E’T’)E’T’id	id + id)*id$	”id” concuerda
$E’T’)E’T’	+id)*id$	T → ϵ
$E’T’)E’	+id)*id$	E′ → +T E′
$E’T’)E’T+	+id)*id$	”+” concuerda
$E’T’)E’T	id)*id$	T → F T′
$E’T’)E’T’F	id)*id$	F → id
$E’T’)E’T’id	id)*id$	”id” concuerda
$E’T’)E’T’	)*id$	T′ → ϵ
$E’T’)E’	)*id$	E′ → ϵ
$E’T’)	)*id$	”)” coincide
$E’T’	*id$	T′ → ∗F T′
$E’T’F	*id$	”*” coincide
$E’T’F	id$	F → id
$E’T’id	id$	id coincide
$E’T’	$	T′ → ϵ
$E’	$	E′ → ϵ
$	$	aceptar()

Ejercicio 13

13. La gramática 3.2, sólo maneja las operaciones de suma y multiplicación, modifique esa gramática para que acepte, también, la resta y la división; Posteriormente, elimine la recursividad por la izquierda de la gramática completa y agregue la opción de que F, también pueda derivar en num, es decir, F → (E) | id | num

Gramática:

E → E + T | T

T → T F | F

F → (E) | id

Gramatica modificada:

E → E + T | E - T | T

T → T * F | T / F | F

F → (E) | id | num

Resultado aplicando eliminacion de recursividad:

E → T E’

E’ → +T E| -T E | ϵ

$T- > FT'$

$T'- > FT'—/FT'—$

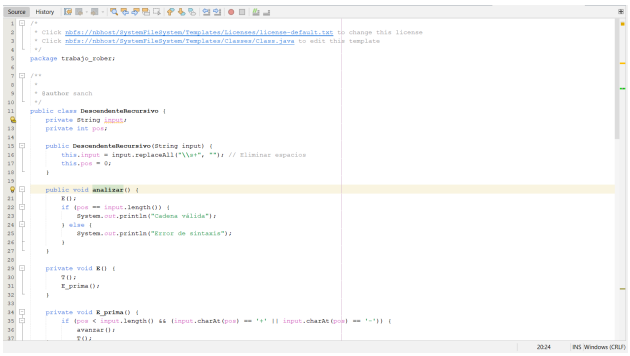
$F- > (E)—id—num$

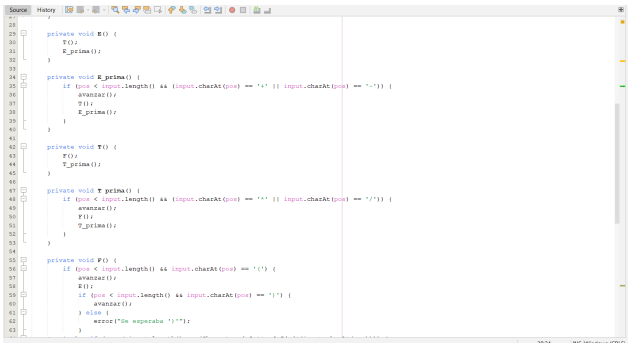
Ejercicio 14

Escriba un pseudocodigo (e implemente en Java) utilizando el metodo descendente recursivo para la gram ´atica resultante del ejercicio anterior (ejercicio 13).

Pseudocodigo:

entrada ← expresión a analizar pos ← 0 procedimiento analizar(entrada) pos ← 0 E() si pos = longitud(entrada) entonces escribir "Cadena válida" sino escribir "Error de sintaxis" procedimiento E() T() $E_prima()procedimientoE_prima()sientrada[pos] =' +'oentrada[pos] =' -'avanzar()T()E_prima()procedimientoT()F()T_prima()sientrada[pos] =' *'oentrada[pos] =' /'avanzar()F()T_prima()procedimientoF()sientrada[pos] =' ('avanzar()E()sientrada[pos] =')'avanzar()sinoescribir"Error : Seesperaba')'"sinosientrada[pos]esidonumavanzar()sinoescribir"Error : Seesperabaid, numo'('"procedimientoavanzar()pos ¸pos + 1$
Codigo en java:





5. Conclusiones

En esta actividad aprendi a realizar el analisis sintactico de diferentes gramaticas, confirmando que estubieran bien, ademas de crear arboles sintacticos y como recorrerlos, en general expandi mi conocimiento sobre el analisis sintactico.

Referencias Bibliográficas

References

[1] Carranza Sahagún, D. U. (**2024**). *Compiladores: Fase de análisis*. Editorial TransDigital. ISBN: 978-607-26754-0-7.