

REPORTE DE PRÁCTICA NO. 2.4

Ejercicios de analisis sintactico

ALUMNO: Axel Aldahir Gutiérrez Gómez
Dr. Eduardo Cornejo-Velázquez



1. Introducción

El análisis sintáctico desempeña un papel clave en la creación de compiladores y en la identificación de la estructura gramatical de un lenguaje de programación. Este proceso verifica que la secuencia de tokens generada durante la fase léxica cumpla con las reglas sintácticas del lenguaje, asegurando así su correcta organización jerárquica. Existen diferentes enfoques para llevar a cabo el análisis sintáctico, como el análisis descendente, que construye el árbol sintáctico desde la raíz hacia las hojas, y el análisis ascendente, que parte de los elementos básicos para formar estructuras más complejas.

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

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.

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).

3. Herramientas empleadas

1. Latex. Es un sistema de preparación de documentos basado en un lenguaje de marcado, utilizado para crear textos con alta calidad tipográfica, especialmente en áreas científicas y técnicas.

4. Desarrollo

Ejercicio 1

1. a) Escriba una gramática que genere el conjunto de cadenas $s;$, $s;s;$, $s;s;s;$, \dots .
- 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): $\{ S \rightarrow s; , S \rightarrow S s; \}$
- Simbolo inicial (S): $\{ S \}$

Gramatica generada: $S \rightarrow S s; , S \rightarrow s;$

Árbol sintactico

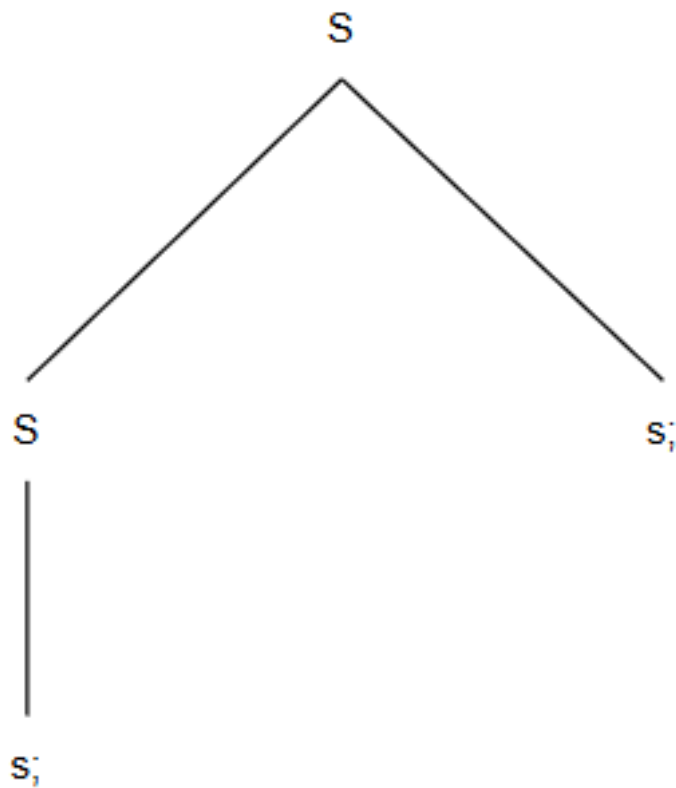


Figure 1: Arbol sintactico

Ejercicio 2

2. Considere la siguiente gramática:

$\text{rexp} \rightarrow \text{rexp} \mid \text{rexp}$

$\mid \text{rexp} \text{ rexp}$

$\mid \text{rexp} \text{ "*"}$

$\mid \text{"(" rexp "}"$

$\mid \text{letra}$

a) Genere un árbol sintáctico para la expresión regular $(ab-b)^*$.

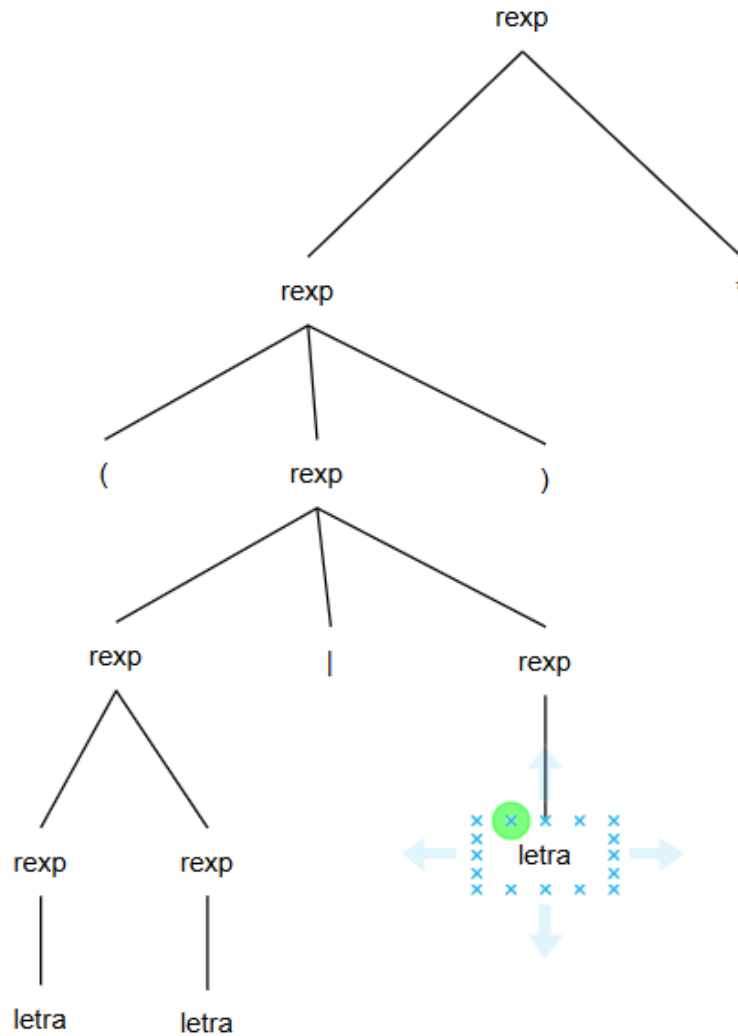


Figure 2: Arbol sintactico

Ejercicio 3

3. De las siguientes gramáticas, describa el lenguaje generado por la gramática y genere árboles sintácticos con las respectivas cadenas.

a) $S \rightarrow S S + | S S^* |$ a con la cadena $aa+a^*$.

b) $S \rightarrow 0 S 1 \mid 0 1$ con la cadena 000111.

c) $S \rightarrow + S S \mid * S S \mid a$ con la cadena $+ * aaa$

a) Descripción del lenguaje generado:

Esta gramática es capaz de generar cadenas compuestas por a, ya sea sola o en una suma de cadenas, y ocupa los operadores * y +

Arbol sintactico generado:

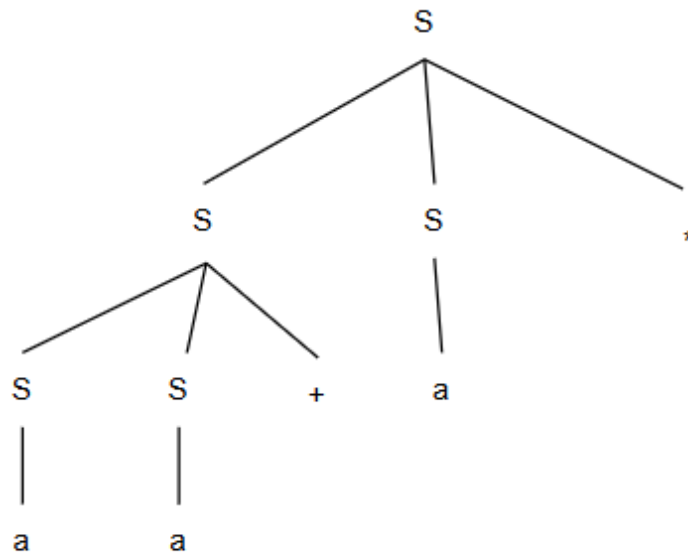


Figure 3: Arbol sintactico

b) Descripción del lenguaje generado:

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

Arbol sintactico generado:

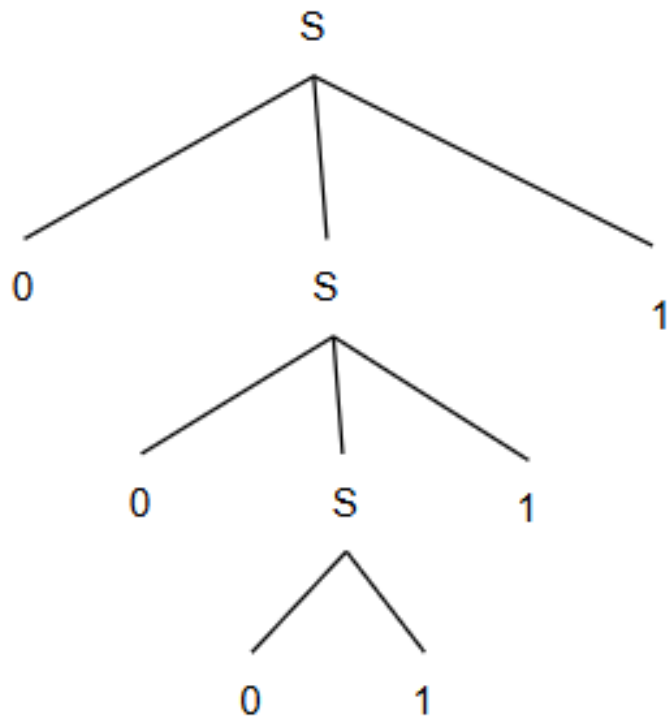


Figure 4: Enter Caption

c) Descripción del lenguaje generado:

Este lenguaje produce cadenas donde los operadores + y * van seguidos de una letra a

Árbol sintáctico generado:

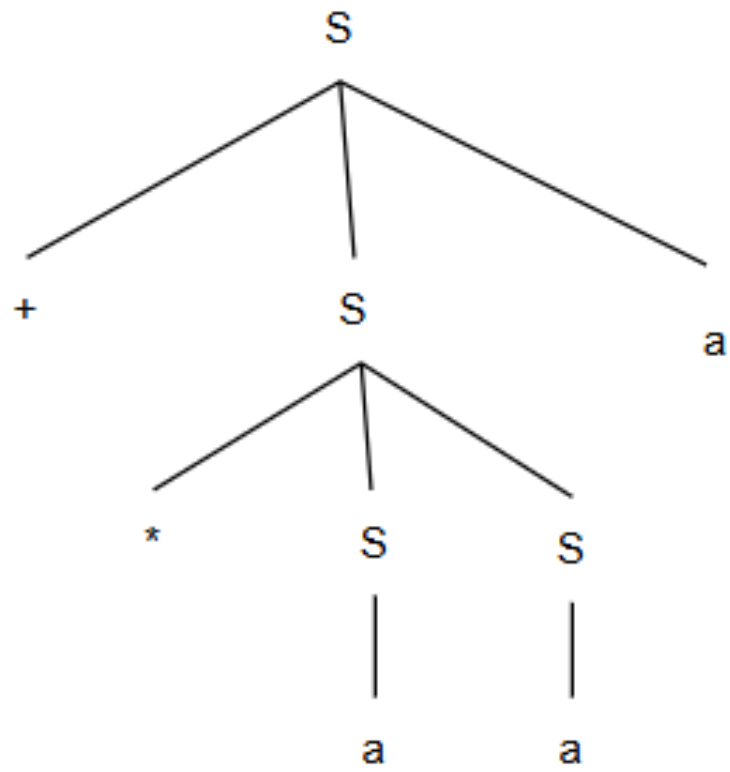


Figure 5: Enter Caption

Ejercicio 4

4. ¿Cuál es el lenguaje generado por la siguiente gramática? $S \rightarrow xSy \mid \epsilon$

El grámatica puede generar cadenas del tipo $L: \{\lambda, xy, xxyy, \dots\}$, donde siempre va a empezar con x y terminar con y, ademas de que siempre habra la misma cantidad de x y y.

Ejercicio 5

Genere el árbol sintáctico para la cadena zazabzbz utilizando la siguiente gramática:

$S \rightarrow zMNz$

$M \rightarrow aNa$

$N \rightarrow bNb$

$N \rightarrow z$

Arbol sintactico

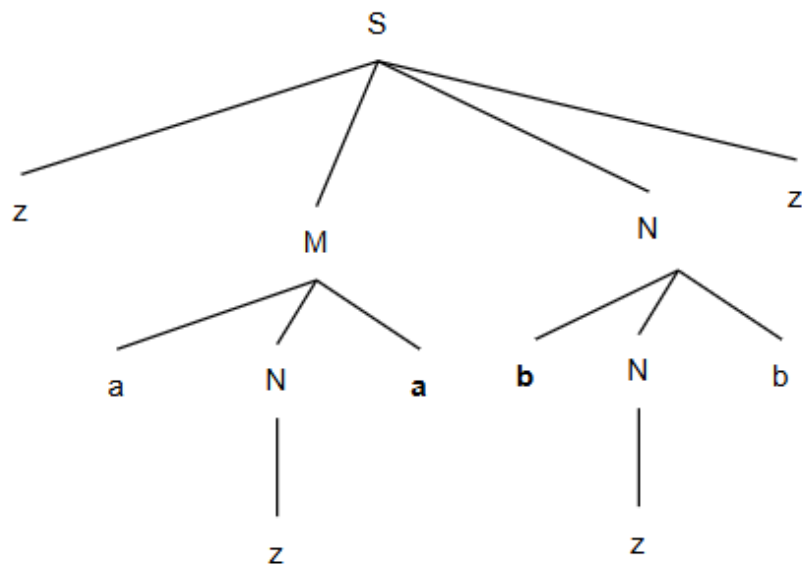


Figure 6: Enter Caption

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.

$S \rightarrow \text{ict}S$

$S \rightarrow \text{ictSe}S$

$S \rightarrow s$

Para el primer árbol se siguen las siguientes derivaciones:

– $S \rightarrow \text{ict}S$

– $\text{ict}S \rightarrow \text{ictictSe}S$

– $\text{ict}S \rightarrow \text{ictictSe}S$

– $\text{ictictSe}S \rightarrow \text{ictictse}S$

– $\text{ictictse}S \rightarrow \text{ictictses}$

B) Para el segundo árbol se siguen las siguientes derivaciones:

– $S \rightarrow \text{ictSe}S$

– $\text{ictSe}S \rightarrow \text{ictictSe}S$

– $\text{ictictSe}S \rightarrow \text{ictictse}S$

– $\text{ictictse}S \rightarrow \text{ictictses}$

Primer árbol

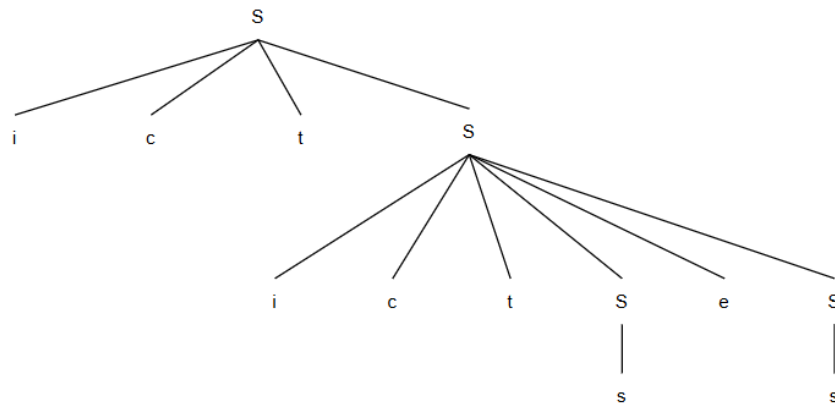


Figure 7: Arbol sintactico

Segundo arbol

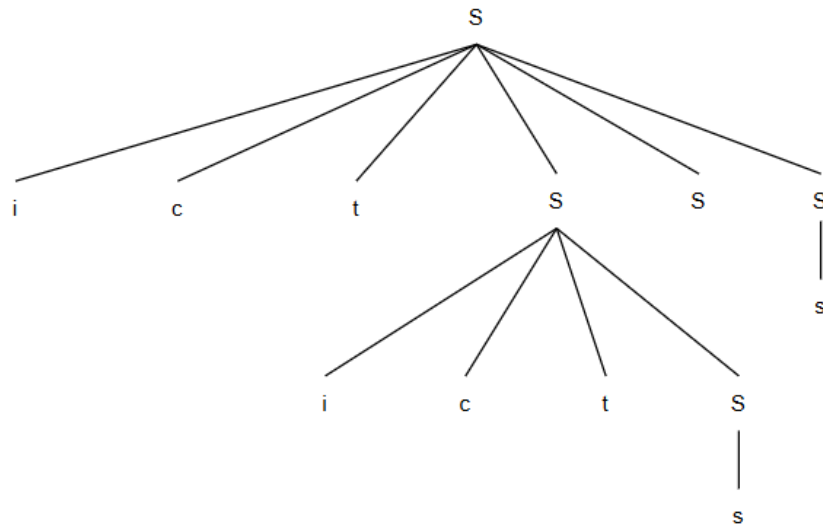


Figure 8: Arbol sintactico

Ejercicio 7

7. Considere la siguiente gramática

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

Encuéntrense árboles de análisis sintáctico para las siguientes frases:

- a) (a, a)
- b) (a, (a, a))
- c) (a, ((a, a), (a, a)))

Primer arbol

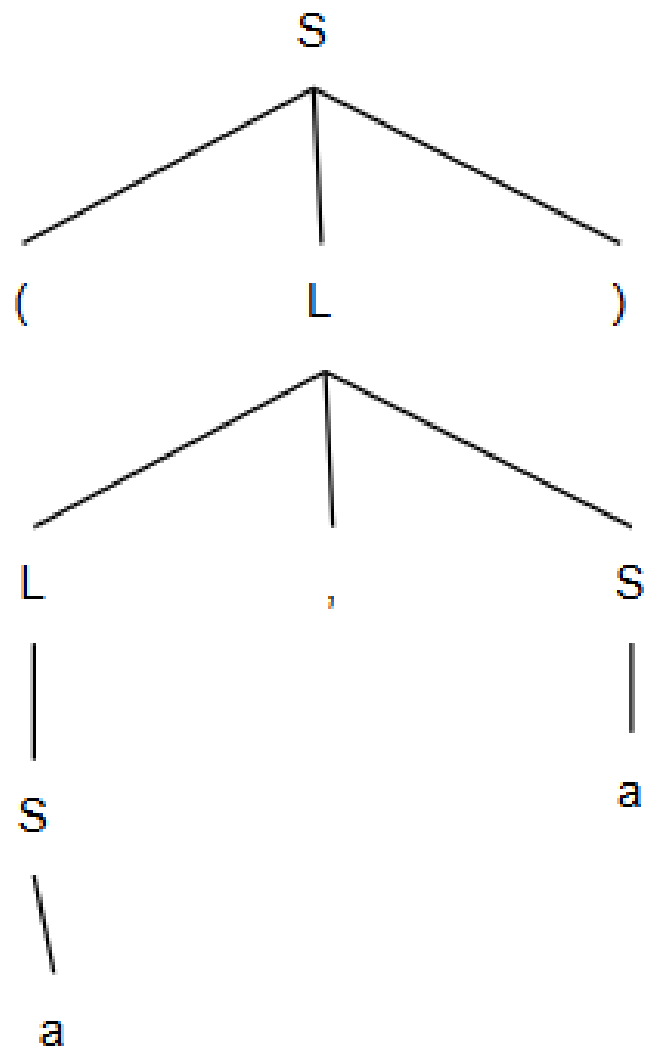


Figure 9: Arbol sintactico

Segundo arbol

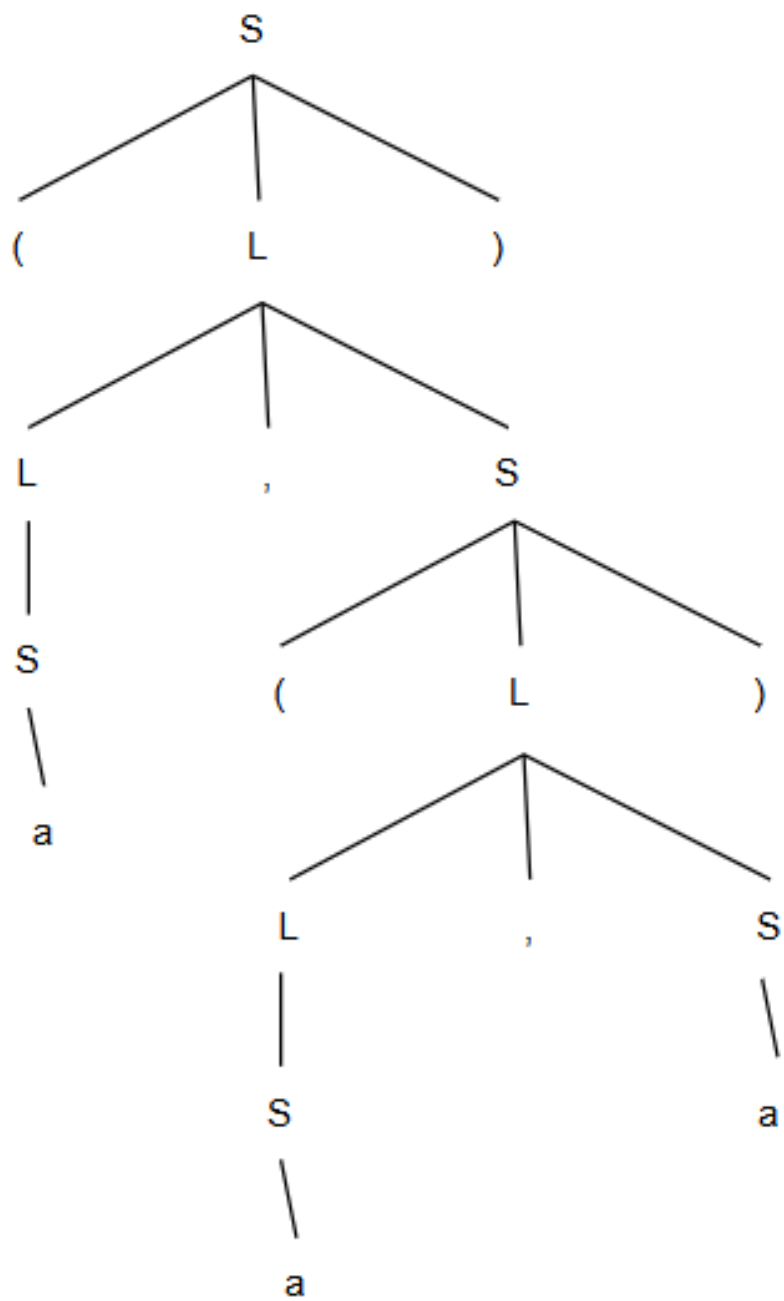


Figure 10: Arbol sintactico

Tercer arbol

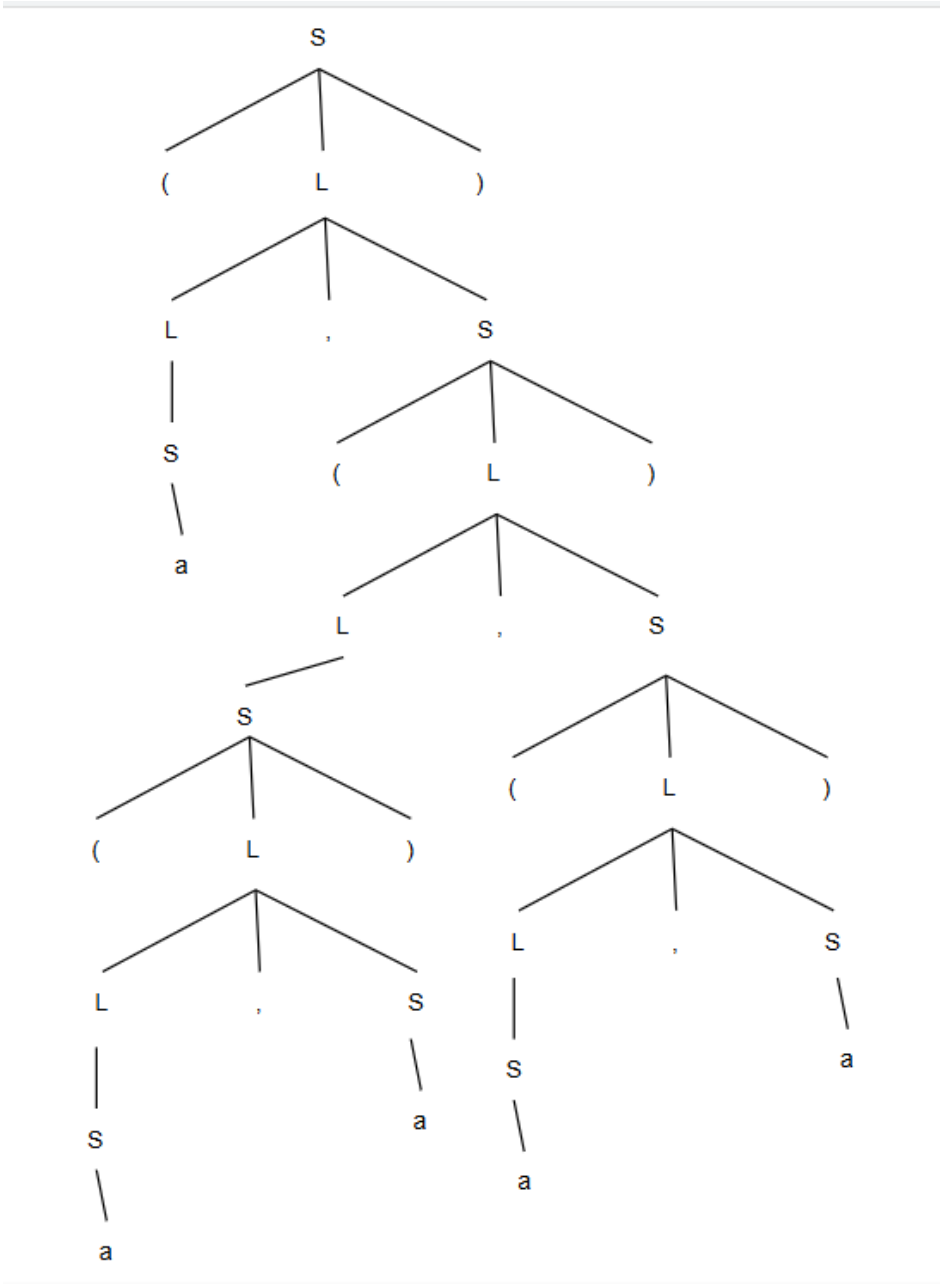


Figure 11: Arbol sintactico

Ejercicio 8

8. Constrúyase un árbol sintáctico para la frase not (true or false) y la gramática:

$\text{bexpr} \rightarrow \text{bexpr or bterm} \mid \text{bterm}$

$\text{bterm} \rightarrow \text{bterm and bfactor} \mid \text{bfactor}$

$\text{bfactor} \rightarrow \text{not bfactor} \mid (\text{bexpr}) \mid \text{true} \mid \text{false}$

Arbol sintactico

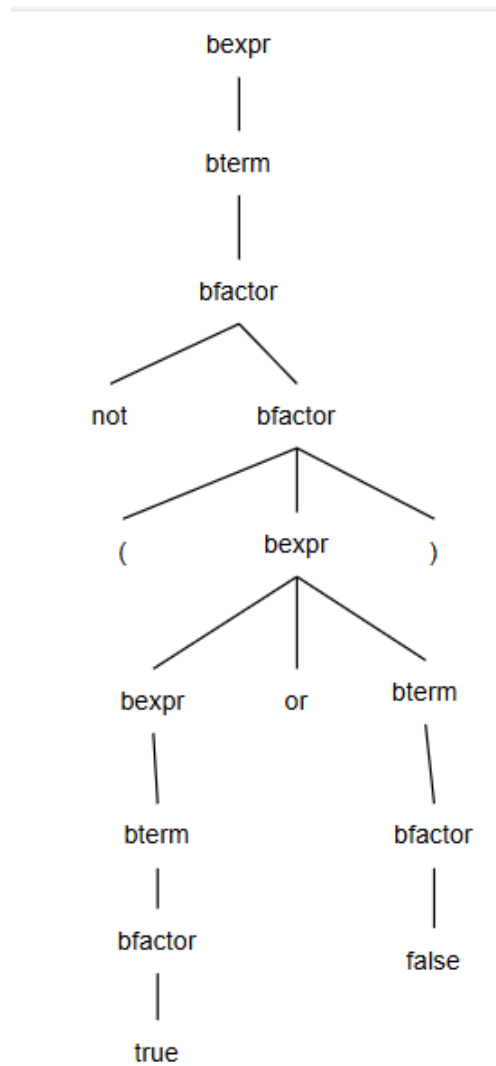


Figure 12: Arbol sintactico

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(Σ): {0, 1}

Conjuntodereglasdeproduccion(P): { $S \rightarrow 1S$; , $S \rightarrow 01S$; , $S \rightarrow \epsilon$; }

Símbolo inicial(S): {S}

Gramática generada:

- $S \rightarrow 1S$;
- $S \rightarrow 01S$;
- $S \rightarrow \epsilon$;

Ejercicio 10

10. Elimine la recursividad por la izquierda de la siguiente gramática:

$S \rightarrow (L) \mid a$

$L \rightarrow L , S \mid S$

Lo primero que se debe de hacer es analizar si realmente tiene recursividad, como podemos ver en la primera gramática no hay recursividad al no estar la S del lado derecho, pero en la segunda si existe la recursividad, ahora que lo sabemos tomamos esa regla y la reescribimos de la siguiente manera:

$L \rightarrow L , S \mid S$

$L \rightarrow SL$

$L \rightarrow , SL \mid \epsilon$

Y la gramática obtenida se vería algo así:

$S \rightarrow (L) \mid a$

$L \rightarrow SL$

$L \rightarrow , SL \mid \epsilon$

Ejercicio 11

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

Para realizar el análisis sintáctico de la gramática $S \rightarrow (S)x$ mediante el método descendente recursivo, creamos una función llamada analizarS(). Esta función verifica primero si el símbolo actual en la entrada es un paréntesis izquierdo '('; si es así, avanza al siguiente símbolo y llama recursivamente a analizarS() para analizar el contenido dentro del paréntesis. Si después encuentra un paréntesis derecho ')', también avanza y considera la derivación como válida. Si el símbolo actual es una 'x', simplemente avanza y considera esa parte como válida. Si no se cumple ninguno de estos casos, la función devuelve falso, indicando que la cadena no es válida.

La función avanzar() se encarga de mover el índice al siguiente símbolo en la entrada. Finalmente, la función iniciarAnálisis() comienza el proceso, inicializando el índice y llamando a analizarS(). Si analizarS() devuelve verdadero y el índice llega al final de la cadena, se considera que la cadena es válida; de lo contrario, se considera no válida.

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 (  $M[X, a] = X \rightarrow Y_1 Y_2 \dots Y_k$  ) {
        enviar de salida la producción  $X \rightarrow Y_1 Y_2 \dots Y_k$ ;
        extraer de la pila;
        meter  $Y_k, Y_{k-1}, \dots, Y_1$  en la pila, con  $Y_1$  en la parte superior;
    }
    establecer  $X$  con el símbolo de la cima de la pila;
}

```

Figure 13: Algoritmo de analisis

Table 13: Tabla

No terminal	Símbolo de entrada				
	id	+	*	()
E	$E \rightarrow TE^+$			$E \rightarrow TE^+$	
E^+		$E^+ \rightarrow +TE^+$		$E^+ \rightarrow \epsilon$	$E^+ \rightarrow \epsilon$
T	$T \rightarrow FT^+$			$T \rightarrow FT^+$	
T^+		$T^+ \rightarrow \epsilon$	$T^+ \rightarrow *FT^+$	$T^+ \rightarrow \epsilon$	$T^+ \rightarrow \epsilon$
F	$F \rightarrow id$			$F \rightarrow (E)$	

Figure 14: Tabla

Pila	Entrada	Acción
\$E	(id + id)*id\$	$E \rightarrow T E'$
\$E'T	(id + id)*id\$	$T \rightarrow F T'$
\$E'T'F	(id + id)*id\$	$F \rightarrow (E)$
\$E'T')E((id + id)*id\$	"(" concuerda
\$E'T')E	id + id)*id\$	$E \rightarrow T E'$
\$E'T')E'T	id + id)*id\$	$T \rightarrow F T'$
\$E'T')E'T'F	id + id)*id\$	$F \rightarrow id$
\$E'T')E'T'id	id + id)*id\$	"id" concuerda
\$E'T')E'T'	+id)*id\$	$T \rightarrow \epsilon$
\$E'T')E'	+id)*id\$	$E' \rightarrow +T E'$
\$E'T')E'T+	+id)*id\$	"+" concuerda
\$E'T')E'T	id)*id\$	$T \rightarrow F T'$
\$E'T')E'T'F	id)*id\$	$F \rightarrow id$
\$E'T')E'T'id	id)*id\$	"id" concuerda
\$E'T')E'T')id\$	$T' \rightarrow \epsilon$
\$E'T')E')id\$	$E' \rightarrow \epsilon$
\$E'T'))id\$	")" coincide
\$E'T'	*id\$	$T' \rightarrow *F T'$
\$E'T'F	*id\$	"*" coincide
\$E'T'F	id\$	$F \rightarrow id$
\$E'T'id	id\$	id coincide
\$E'T'	\$	$T' \rightarrow \epsilon$
\$E'	\$	$E' \rightarrow \epsilon$
\$	\$	aceptar()

Figure 15: Tabla

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, $F \rightarrow (E) \mid id \mid num$

Gramática:

$E \rightarrow E + T \mid T$

$T \rightarrow T F \mid F$

$F \rightarrow (E) \mid id$

Resultado aplicando eliminacion de recursividad:

$E \rightarrow T E$

$E \rightarrow +T E \mid T E \mid \epsilon$

$T \rightarrow FT$

$T \rightarrow FT \mid FT \mid \epsilon$

$F \rightarrow (E) \mid id \mid num$

Ejercicio 14

Planteamiento: Escriba un pseudocódigo (e implemente en Java) utilizando el método descendente recursivo para la gramática resultante del ejercicio anterior (ejercicio 13):

Pseudocódigo:

Inicio

Leer la expresión del usuario

try

Crear el objeto parser con la expresión

Llamar al método analizar para el objeto parser

Imprimir el texto "Expresión válida"

Capturar las excepciones

Imprimir el mensaje de error para la excepción

Clase Parser

Atributos:

input: cadena de texto que contiene la expresión

index: índice actual en la expresión

tokenAct: el token actual de la expresión

Constructor (input)

Quitar espacios en blanco de la expresión

Establecer el token Actual como el siguiente token en la expresión

Método para obtener el siguiente token

Si index es menor que la longitud de la expresión devolver el carácter en la posición index

En caso de que no, devolver el carácter nulo ("")

Siguiente Método: analizar

Se llama el método E

Si tokenActual no es ""

Llamar al método error con el mensaje "Expresión mal formada"

Método E

Llamar al método T

Llamar al método E

Método E

Si tokenAct es '+' o '-'

Obtener el operador actual

Establecer tokenAct como el siguiente token

Llamamos al método T

Llamamos al método E

Método T

Llamamos al método F

Llamamos al método T

Método T

Si el tokenActual es '*' o '/'

Obtenemos el operador actual

Establecemos el tokenActual como el siguiente token

Mandamos a llamar al método F

Llamamos al método T

Método F

Si tokenActual es '('

Establecemos tokenActual como el siguiente token

Mandamos llamar al método E

Si tokenAct es ')'

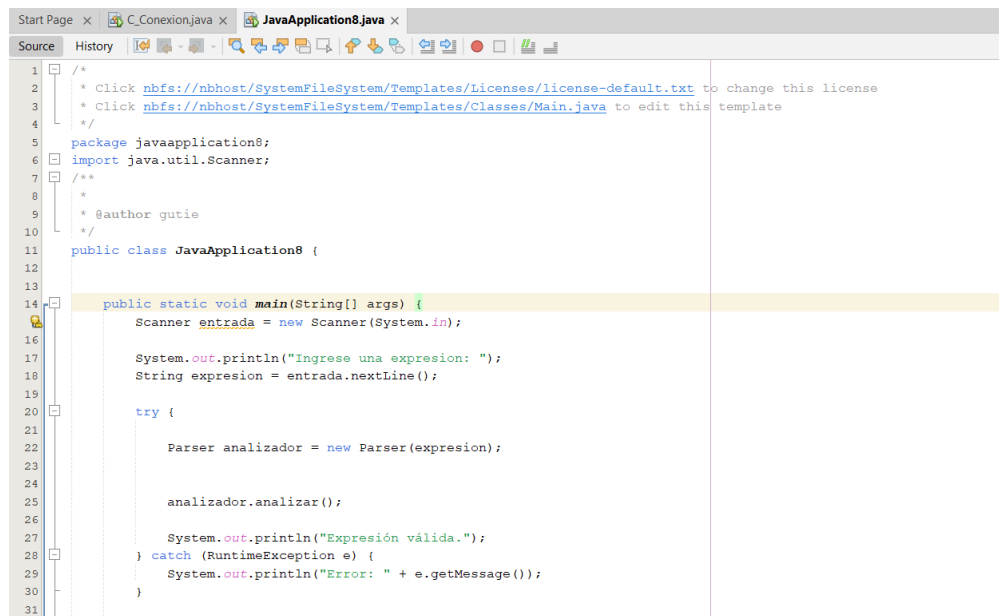
Establecemos tokenAct como el siguiente token

Si no

Llamamos al método error con el mensaje "Se esperaba ')".

Si no, Si tokenAct es un dígito
Mientras tokenAct sea un dígito
Establemos tokenAct como el siguiente token
Si no, Si tokenAct es una letra
Establecer tokenAct como el siguiente token
En caso de que no
Llamamos al método error con el mensaje "El token fue inesperado."
Método error(mensaje)
Lanzamos la excepción RuntimeException con el respectivo mensaje de error
Fin Clase

Programa en java



```
1  /**
2   * Click nbfs://nbhost/SystemFileSystem/Templates/Licenses/license-default.txt to change this license
3   * Click nbfs://nbhost/SystemFileSystem/Templates/Classes/Main.java to edit this template
4   */
5  package javaapplication8;
6  import java.util.Scanner;
7
8  /**
9   * @author gutie
10  */
11  public class JavaApplication8 {
12
13
14      public static void main(String[] args) {
15          Scanner entrada = new Scanner(System.in);
16
17          System.out.println("Ingrese una expresion: ");
18          String expresion = entrada.nextLine();
19
20          try {
21
22              Parser analizador = new Parser(expresion);
23
24              analizador.analizar();
25
26              System.out.println("Expresión válida.");
27          } catch (RuntimeException e) {
28              System.out.println("Error: " + e.getMessage());
29          }
30      }
31  }
```

Figure 16: Programa

```
Start Page x C_Conexion.java x JavaApplication8.java x
Source History
26
27      System.out.println("Expresión válida.");
28  } catch (RuntimeException e) {
29      System.out.println("Error: " + e.getMessage());
30  }
31
32      entrada.close();
33  }
34  }
35
36  class Parser {
37
38      private String input;
39      private int index = 0;
40      private char tokenAct;
41
42
43      public Parser(String input) {
44          this.input = input.replaceAll("\\s+", "");
45          this.tokenAct = obtenerSiguienteToken();
46      }
47
48
49      private char obtenerSiguienteToken() {
50          return (index < input.length()) ? input.charAt(index++) : '\0';
51      }
52
53
54      public void analizar() {
55          E();
56          if (tokenAct != '\0') {
```

Figure 17: Programa

```
Start Page x C_Conexion.java x JavaApplication8.java x
Source History
53
54      public void analizar() {
55          E();
56          if (tokenAct != '\0') {
57              error("Expresion mal formada.");
58          }
59      }
60
61
62      private void E() {
63          T();
64          E_();
65      }
66
67
68      private void E_() {
69          if (tokenAct == '+' || tokenAct == '-') {
70              char operador = tokenAct;
71              tokenAct = obtenerSiguienteToken();
72              T();
73              E_();
74          }
75      }
76
77
78      private void T() {
79          F();
80          T_();
81      }
82
83
```

Figure 18: Programa

```

private void T () {
    if (tokenAct == '*' || tokenAct == '/') {
        char operador = tokenAct;
        tokenAct = obtenerSiguienteToken();
        F();
        T_();
    }
}

private void F() {
    if (tokenAct == '(') {
        tokenAct = obtenerSiguienteToken();
        E();

        if (tokenAct == ')') {
            tokenAct = obtenerSiguienteToken();
        } else {
            error("Se esperaba ')'");
        }
    } else if (Character.isDigit(tokenAct)) {
        // Mientras el token actual sea un dígito
        while (Character.isDigit(tokenAct)) {
            tokenAct = obtenerSiguienteToken();
        }
    } else if (Character.isLetter(tokenAct)) {
        tokenAct = obtenerSiguienteToken();
    } else {
        error("El token fue inesperado.");
    }
}

```

Figure 19: Programa

```

Source History
96 tokenAct = obtenerSiguienteToken();
97 E();
98
99 if (tokenAct == ')') {
100 tokenAct = obtenerSiguienteToken();
101 } else {
102 error("Se esperaba ')'");
103 }
104 } else if (Character.isDigit(tokenAct)) {
105 // Mientras el token actual sea un dígito
106 while (Character.isDigit(tokenAct)) {
107 tokenAct = obtenerSiguienteToken();
108 }
109 } else if (Character.isLetter(tokenAct)) {
110 tokenAct = obtenerSiguienteToken();
111 } else {
112 error("El token fue inesperado.");
113 }
114 }
115
116 private void error(String mensaje) {
117 throw new RuntimeException(mensaje);
118 }
119 }
120

```

Figure 20: Programa

```

Output - JavaApplication8 (run) x
run:
Ingrese una expresion:
(55+8)/4
Expresión válida.
BUILD SUCCESSFUL (total time: 28 seconds)

```

Figure 21: Ejecución

5. Conclusiones

Considero que es algo complicado el utilizar los analizadores sintacticos, para comenzar es bastante complicado entender los fundamentos en el texto, entonces desarrollarlos poco a poco en la escala de dificultad es lo ideal, necesito ahondar más y releer para tener un conocimiento completo, considero que es un tema que no se me dio muy bien por la comprensión de la lógica y tratare de releer el texto apoyandome de diferentes fuentes para facilitar mi comprensión del tema, en lo personal he tenido problemas comprendiendo la lógica de programación, entonces repasare el tema.

Referencias Bibliográficas

References

- [1] Carranza Sahagún, D. U. (2024). Compiladores: Fase de análisis. Editorial TransDigital. ISBN: 978-607-26754-0-7., 8(1), 19.