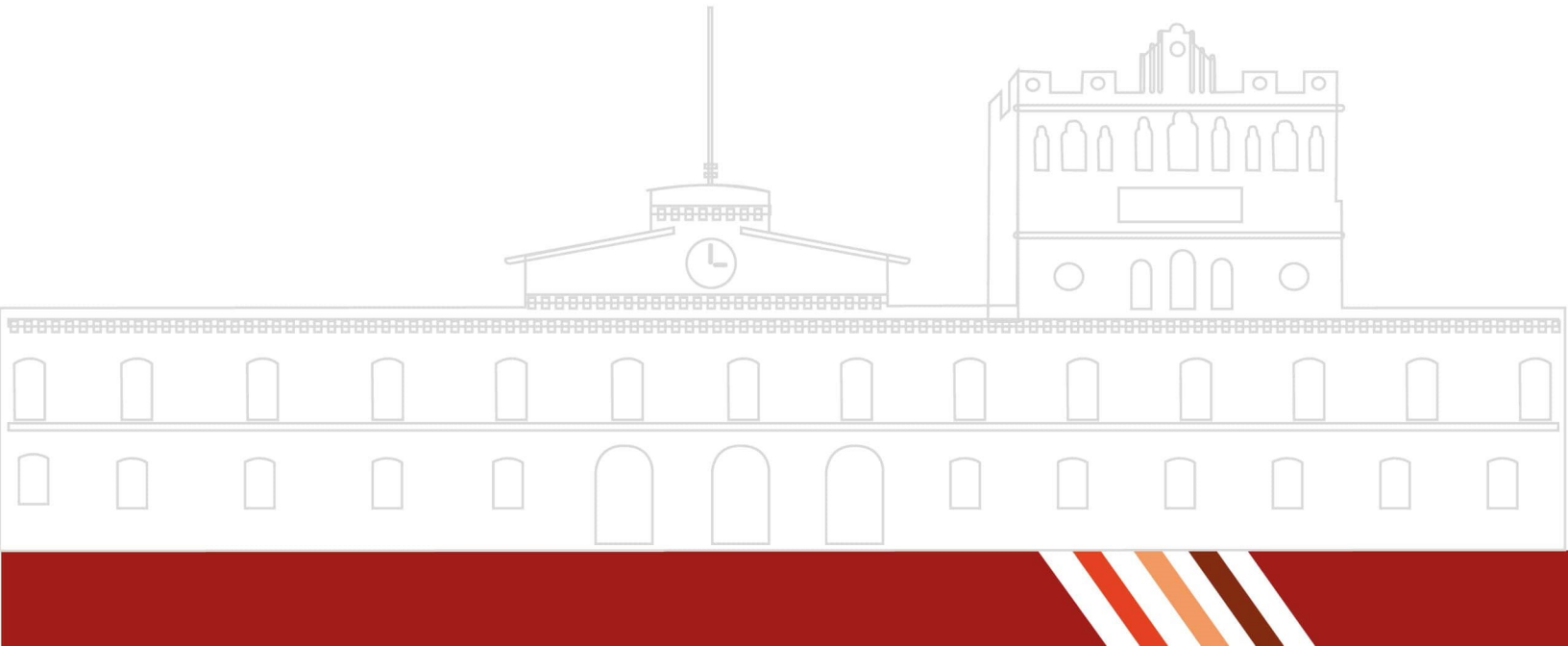


Universidad Autonoma del Estado de Hidalgo

Instituto de Ciencias Basicas e Ingenieria

Licenciatura en Ciencias Computacionales

REPORTE DE PRÁCTICA NO. 3

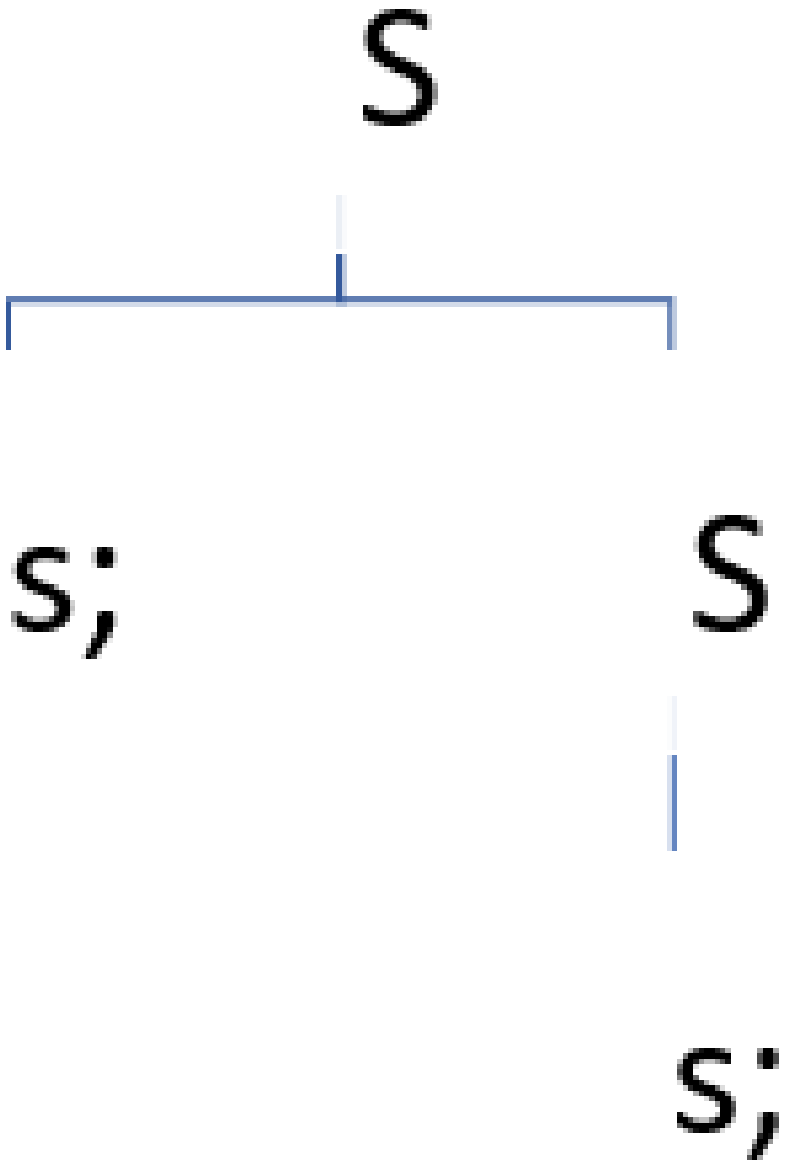
Automatas y Compiladores
2.4 Análisis sintáctico. Ejercicios

ALUMNO: Diego Martinez Ortiz

Dr. Eduardo Cornejo-Velázquez

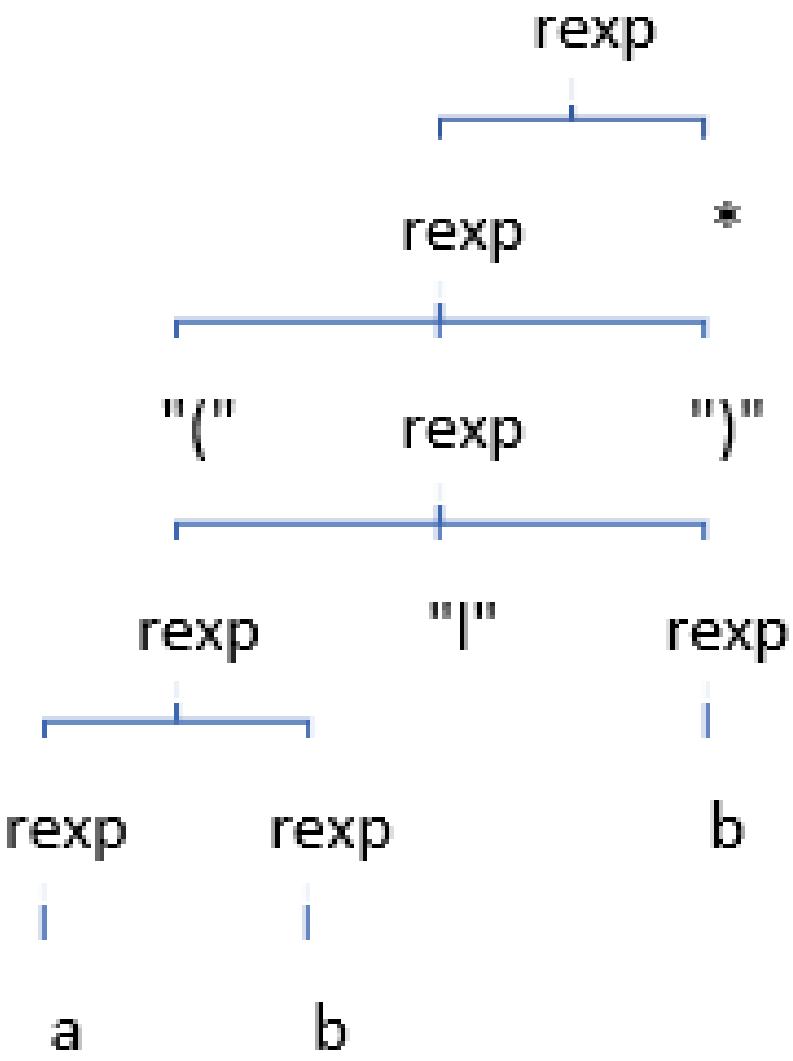
Ejercicio 1

Escriba una gramática que genere el conjunto de cadenas {s; , s;s; , s;s;s; , ...}.
Gramática: S $\rightarrow s;$
$S \rightarrow S; s;$
Genere un árbol sintáctico para la cadena `s;s;`.



Ejercicio 2

Considere la siguiente gramática: rexp $\rightarrow$ *rexp* | *rexp*
rexprexp
rexp $*$
(*rexp*)
letra Genere un árbol sintáctico para la expresión regular $(ab|b)*$.

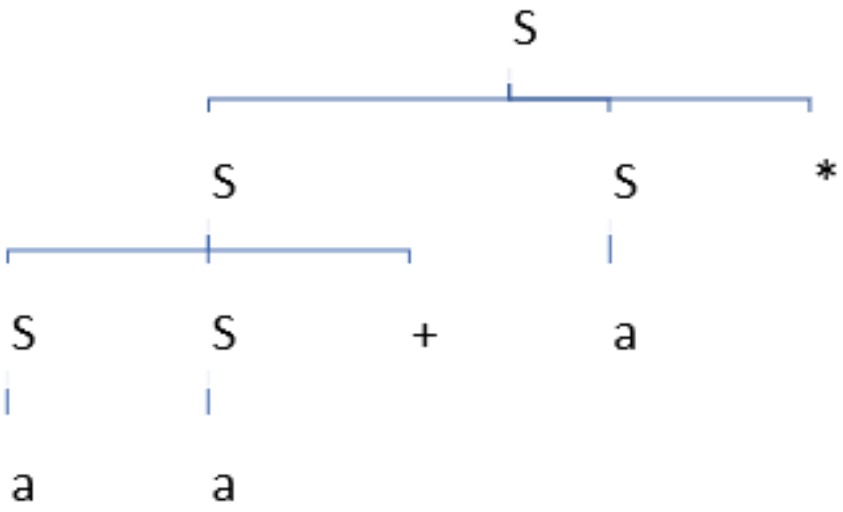


Ejercicio 3

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

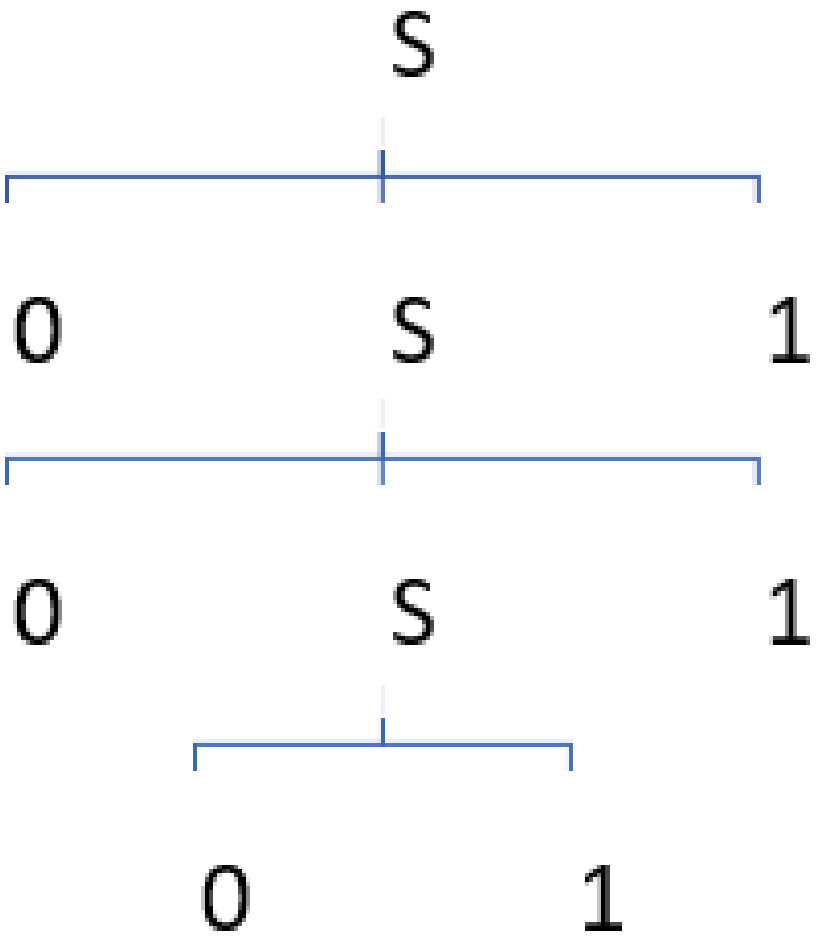
Primera gramática

S $\rightarrow SS+ \mid SS* \mid a$ Cadena: `aa+a*`



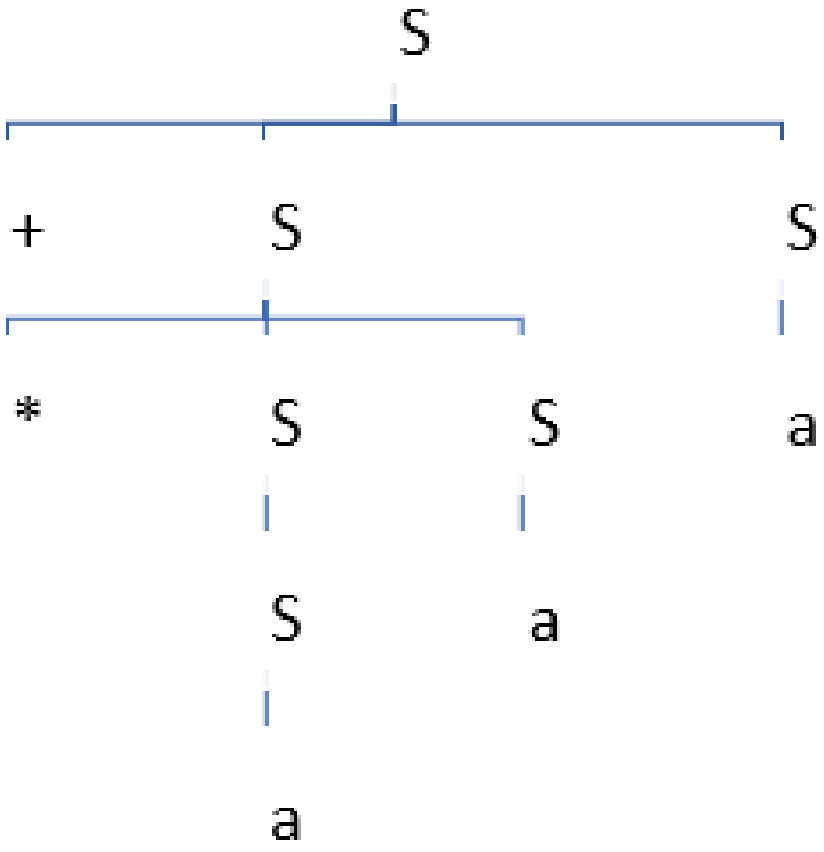
Segunda gramática

S $\rightarrow 0S1 \mid 01$ Cadena: `000111`



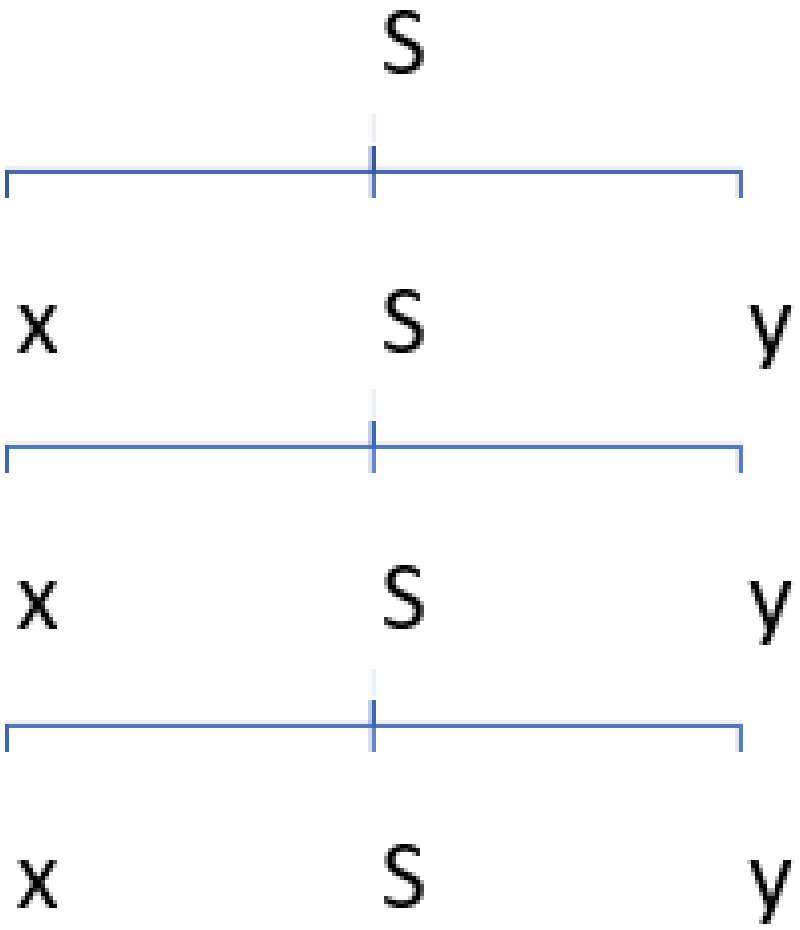
Tercera gramática

S $\rightarrow +SS \mid *SS \mid a$ Cadena: `+ * aaa`



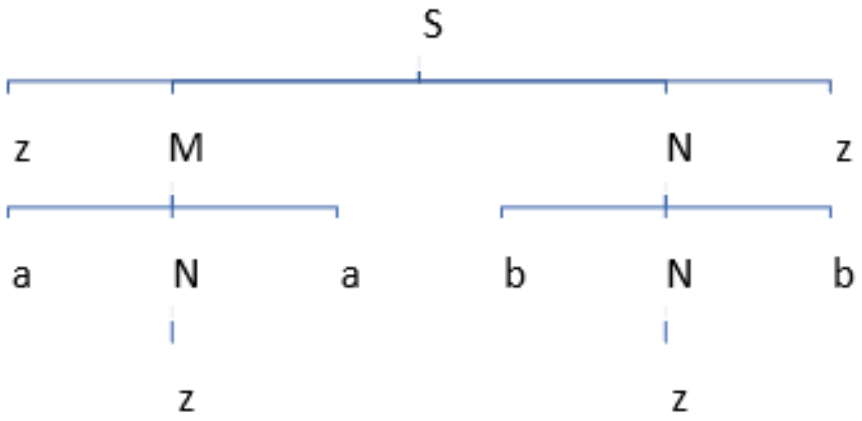
Ejercicio 4

¿Cuál es el lenguaje generado por la siguiente gramática? S $\rightarrow xSy \mid \epsilon$ El lenguaje generado es {xy, xxyy, xxxyyy, ...}.



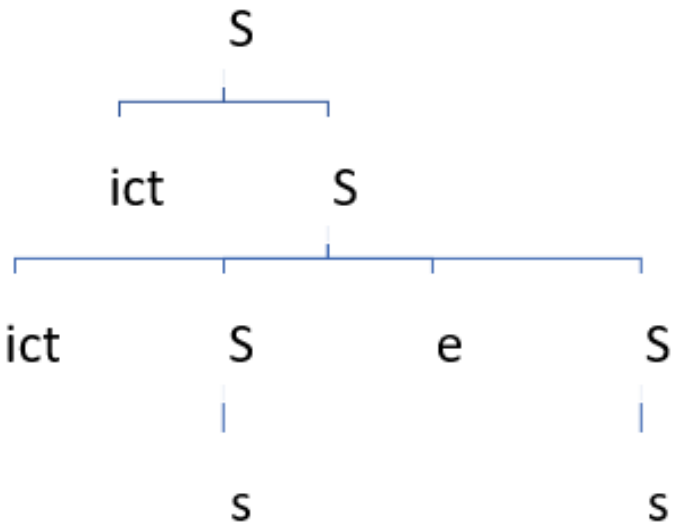
Ejercicio 5

Genere el árbol sintáctico para la cadena `zazabzbz` utilizando la siguiente gramática: S $\rightarrow zMNz$
$M \rightarrow aNa$
$N \rightarrow bNb \mid z$

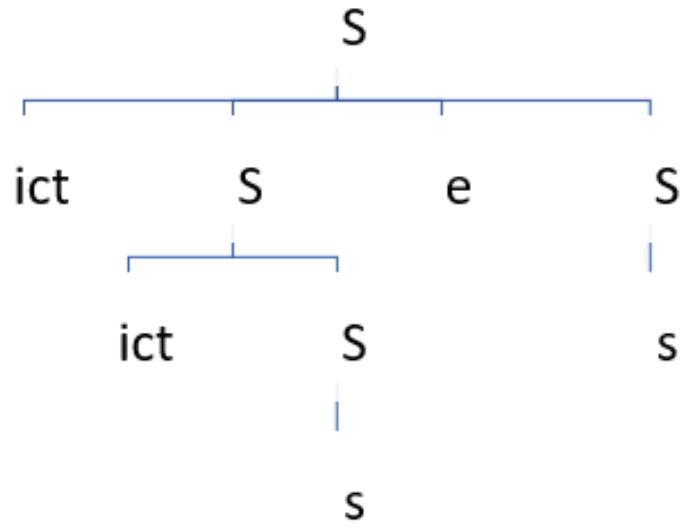


Ejercicio 6

Demuestre que la siguiente gramática es ambigua mostrando distintas derivaciones para `ictictses`. S $\rightarrow ictS \mid ictSeS \mid s$



1.

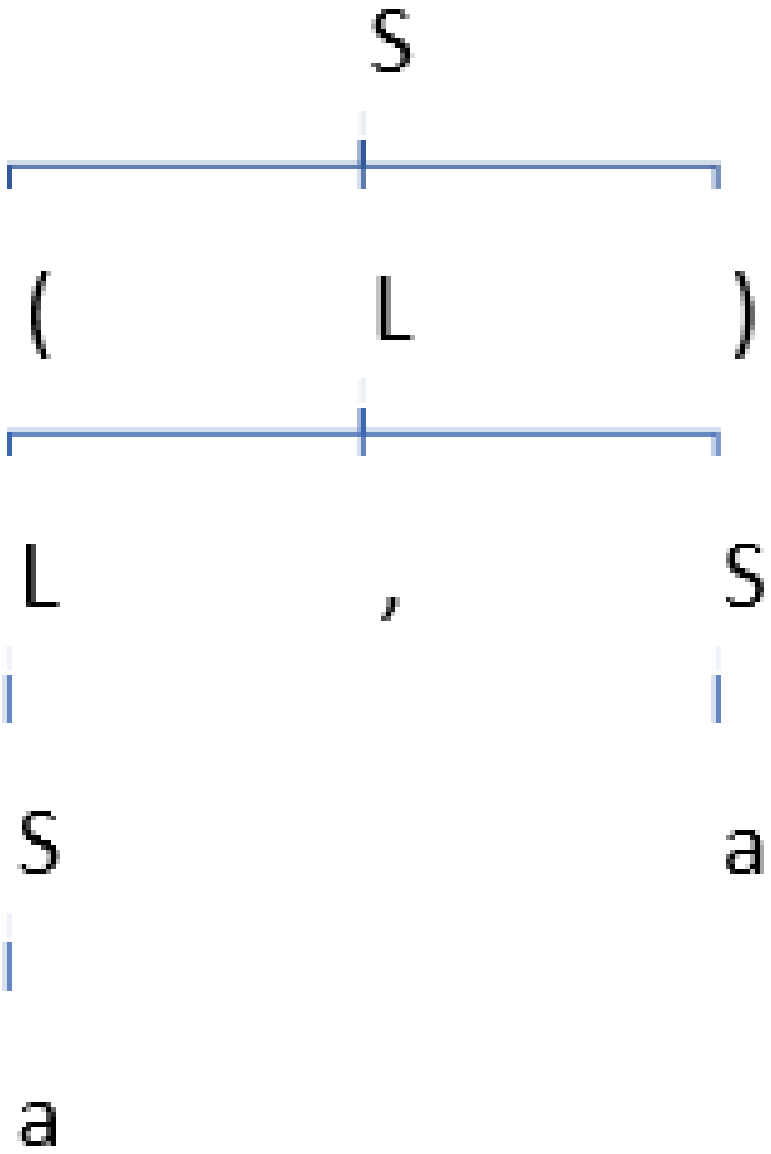


2

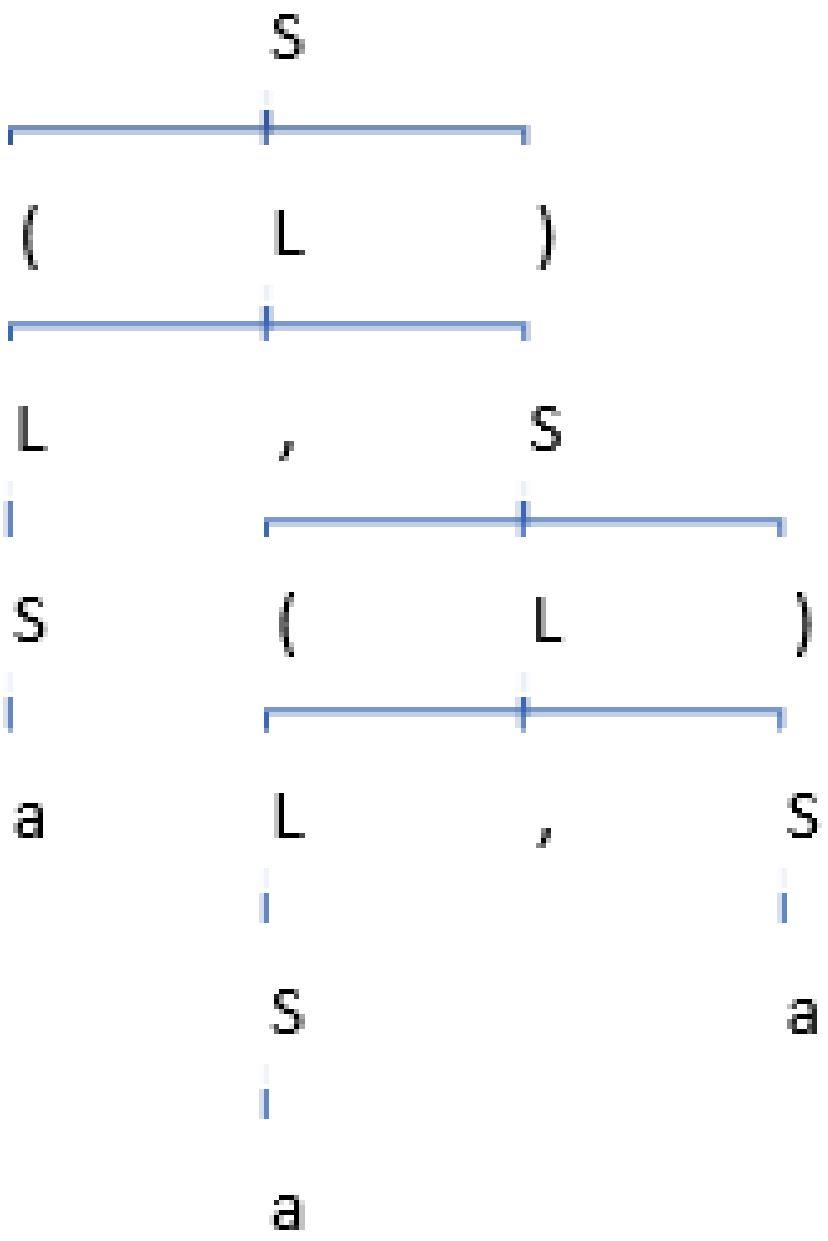
Ejercicio 7

S $\rightarrow (L) \mid a$
$L \rightarrow L, S \mid S$ Genere árboles de análisis sintáctico para:

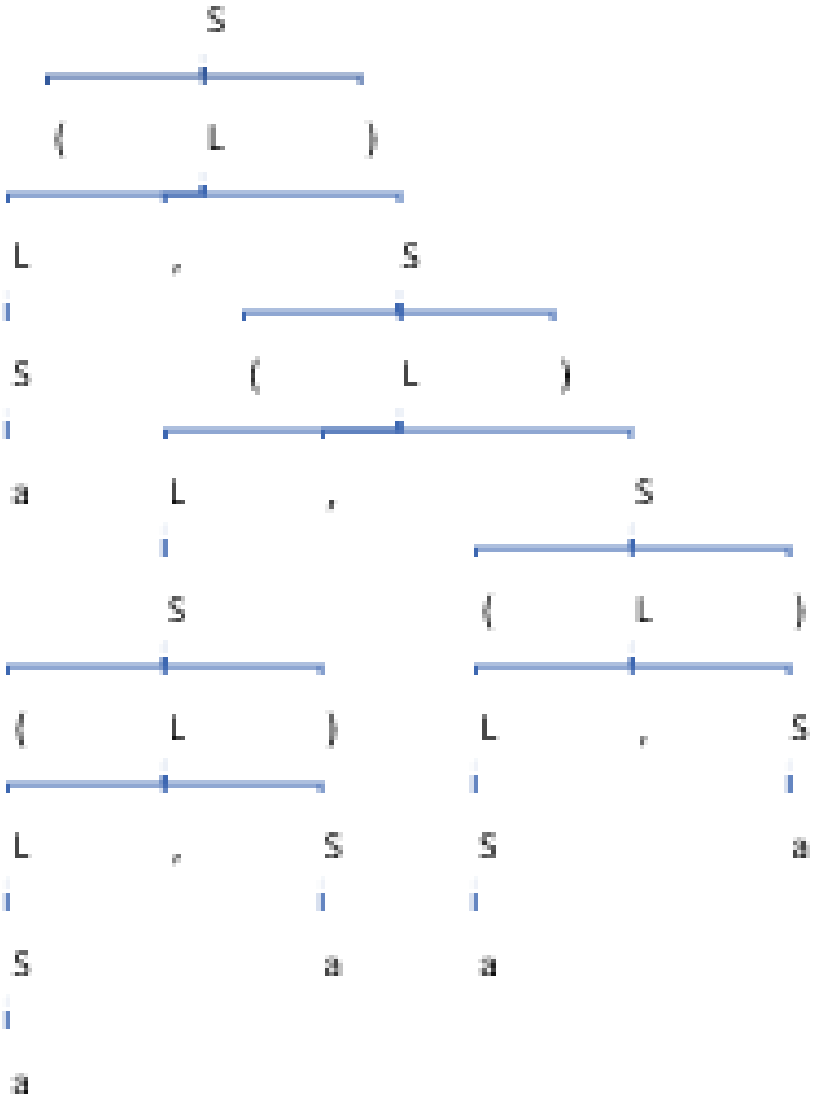
- (a, a)



- $(a, (a, a))$



- $(a, ((a, a), (a, a)))$



Ejercicio 8

Construya un árbol sintáctico para la frase `not (true or false)` con la siguiente gramática: bexpr $\rightarrow bexprorbterm \mid bterm$
$bterm \rightarrow btermandbfactor \mid bfactor$
$bfactor \rightarrow notbfactor \mid (bexpr) \mid true \mid false$

Ejercicio 9

Diseñe una gramática para el lenguaje de cadenas de 0s y 1s donde todo 0 vaya seguido de al menos un 1:
S $\rightarrow 10S \mid 1S \mid 1$

Ejercicio 10

Elimine la recursividad por la izquierda de: $S \rightarrow (L) \mid a$
$L \rightarrow L, S \mid S$ Nueva gramática: $S \rightarrow (L) \mid a$
$L \rightarrow SL'$
$L' \rightarrow, SL' \mid \epsilon$

Ejercicio 11

Pseudocódigo en Java para el análisis sintáctico descendente recursivo:

```
import java.util.*;

public class RecursiveParser {
    private List<String> tokens;
    private int index = 0;

    public RecursiveParser(List<String> tokens) {
        this.tokens = tokens;
    }

    private String tokenActual() {
        return (index < tokens.size()) ? tokens.get(index) : "$";
    }

    private void consumir(String esperado) {
        if (tokenActual().equals(esperado)) {
            index++;
        } else {
            throw new RuntimeException("Error de sintaxis: se esperaba " + esperado);
        }
    }

    public void analizar() {
        S();
        if (!tokenActual().equals("$")) {
            throw new RuntimeException("Error: tokens extra no procesados");
        }
    }

    private void S() {
        if (tokenActual().equals("(")) {
            consumir("(");
            S();
            consumir(")");
        } else if (tokenActual().equals("x")) {
            consumir("x");
        } else {
            throw new RuntimeException("Error de sintaxis en S");
        }
    }

    public static void main(String[] args) {
        List<String> entrada = Arrays.asList("(", "x", ")");
```

```
        RecursiveParser parser = new RecursiveParser(entrada);
        parser.analizar();
        System.out.println("An lisis sint ctico exitoso.");
    }
}
```

Ejercicio 12

Tabla de movimientos del analizador sintáctico predictivo para `(id+id)*id`.

Ejercicio 13

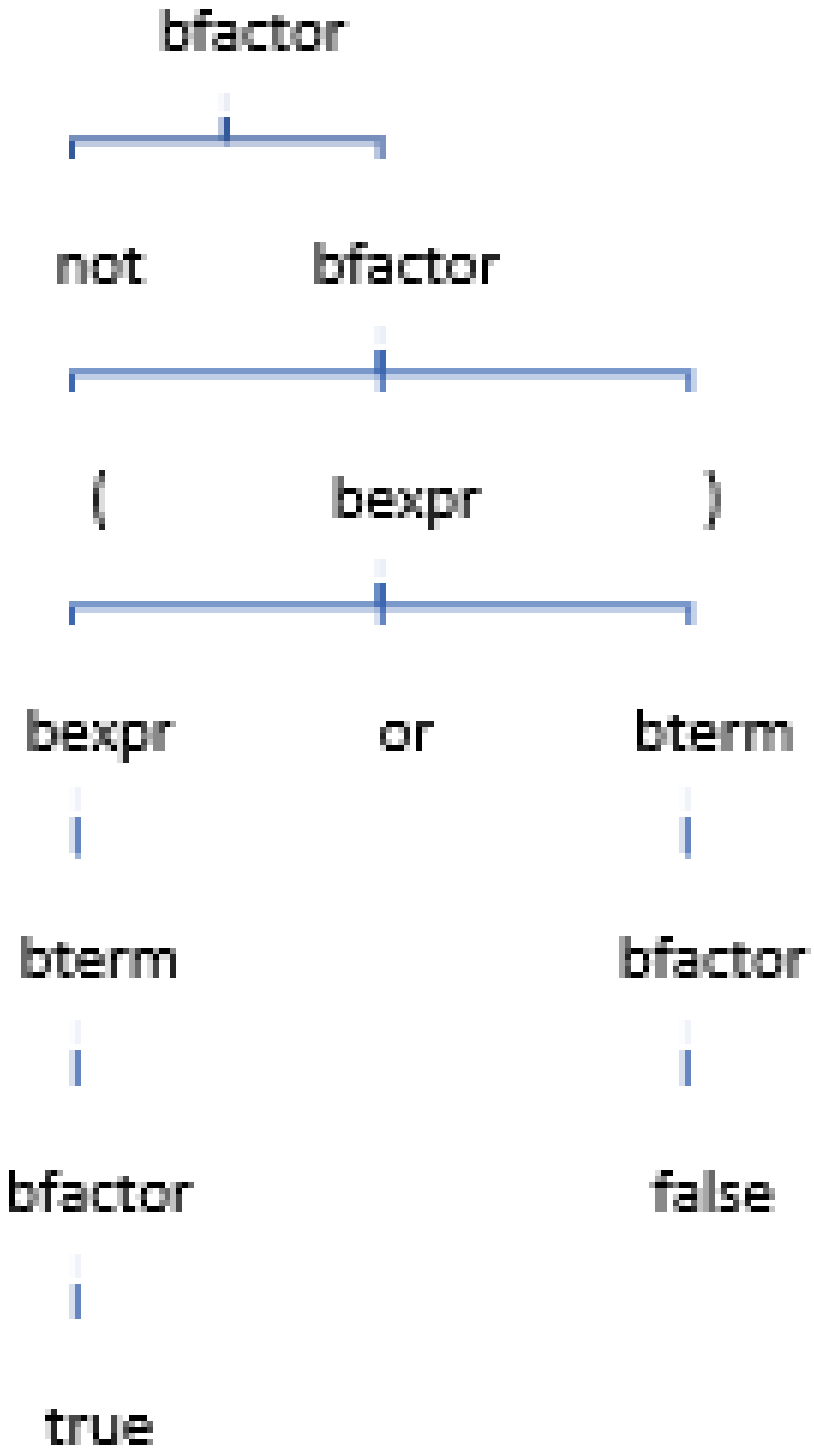
Gramática modificada para admitir resta y división: $\mathrm{E} \to E + T \mid E - T \mid T$
$T \to T * F \mid T/F \mid F$
$F \to (E) \mid id \mid num$
Eliminando la recursión por la izquierda: $\mathrm{E} \to TE'$
$E' \to +TE' \mid -TE' \mid \epsilon$
$T \to FT'$
$T' \to *FT' \mid /FT' \mid \epsilon$
$F \to (E) \mid id \mid num$

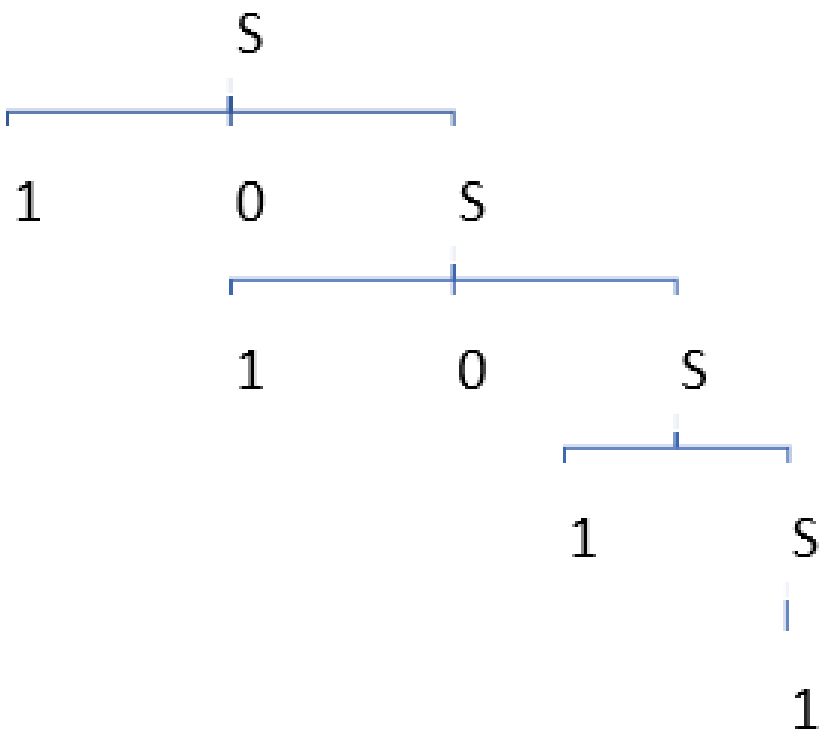
Ejercicio 14

Código en Java para el análisis sintáctico descendente recursivo:

```
import java.util.*;

public class Parser {
    private List<String> tokens;
    private int index = 0;

    public Parser(List<String> tokens) {
        this.tokens = tokens;
    }

    private String tokenActual() {
        return (index < tokens.size()) ? tokens.get(index) : "$";
    }

    private void consumir(String esperado) {
        if (tokenActual().equals(esperado)) {
            index++;
        } else {
            throw new RuntimeException("Error de sintaxis: se esperaba " + esperado);
        }
    }

    public void analizar() {
        E();
        if (!tokenActual().equals("$")) {
            throw new RuntimeException("Error: tokens extra no procesados");
        }
```

```
    }

    private void E() {
        T();
        E_prima();
    }

    private void E_prima() {
        if (tokenActual().equals("+") || tokenActual().equals("-")) {
            String op = tokenActual();
            consumir(op);
            T();
            E_prima();
        }
    }

    private void T() {
        F();
        T_prima();
    }

    private void T_prima() {
        if (tokenActual().equals("*") || tokenActual().equals("/")) {
            String op = tokenActual();
            consumir(op);
            F();
            T_prima();
        }
    }

    private void F() {
        if (tokenActual().equals("(")) {
            consumir("(");
            E();
            consumir(")");
        } else if (tokenActual().matches("[a-zA-Z0-9]+")) { // Permite id y num
            consumir(tokenActual());
        } else {
            throw new RuntimeException("Error de sintaxis en F");
        }
    }

    public static void main(String[] args) {
        List<String> entrada = Arrays.asList("(", "id", "+", "num", ")", "*", "id", "-", "n
        Parser parser = new Parser(entrada);
        parser.analizar();
        System.out.println("An lisis sint ctico exitoso.");
    }
}
```


bfactor
not
bfactor
(
bexpr
)
bexpr
or
bterm
bterm
bfactor
bfactor
false
true



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 → id
$E' T' id	id * id + id$	Concuerda(id)
$E' T'	* id + id$	T' → * F T'
$E' T' * F	* id + id$	Concuerda(*)
$E' T' F	id + id$	F → id
$E' T' id	id + id$	Concuerda(id)
$E' T'	+ id$	T' → ε
$E'	+ id$	E' → + T E'
$E' + T	+ id$	Concuerda(+)
$E' T	id$	T → F T'
$E' T' F	id$	F → id
$E' T' id	id$	Concuerda(id)
$E' T'	$	T' → ε
$E'	$	E' → ε
$	$	Aceptar