Universidad Autónoma del Estado de Hidalgo

REPORTE DE PRÁCTICA NO. 3

2.4 Análisis sintáctico. Ejercicios

ALUMNO:
Mario Daniel Tellez Olivares

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;

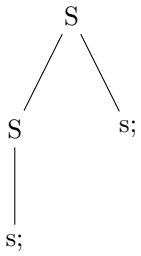
Explicación

Se debe construir una gramática que genere secuencias de "s;" repetidas. La producción recursiva por la derecha permite generar cualquier cantidad de "s;".

Resolución

$$S \rightarrow S\ s; \mid s;$$

Árbol Sintáctico



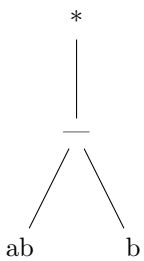
Ejercicio 2

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

Explicación

El árbol sintáctico muestra la estructura de la expresión regular, con '—' en la raíz y '*' abarcando la expresión completa.

Árbol Sintáctico



Ejercicio 3

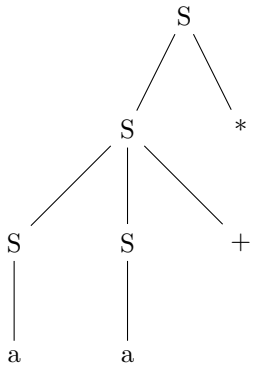
3. De las siguientes gramáticas, describa el lenguaje generado por la gramática y genere árboles sintácticos con las respectivas cadenas.
a) S → S S + — S S * — a con la cadena aa+a*.
b) S → 0 S 1 — 0 1 con la cadena 000111.
c) S → + S S — * S S — a con la cadena + * aaa.

(a) Gramática: $S \to SS+ \mid SS* \mid a$

3.1.1 Lenguaje Generado

El lenguaje generado por esta gramática consiste en expresiones aritméticas con operadores '+' y '*', y el terminal 'a' como único operando. Se genera en notación postfija.

3.1.2 Árbol Sintáctico para la cadena $aa + a*$



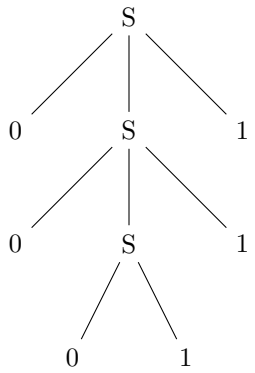
—

(b) Gramática: $S \to 0S1 \mid 01$

3.2.1 Lenguaje Generado

Esta gramática genera cadenas balanceadas de ceros y unos donde cada '0' tiene un '1' correspondiente.

3.2.2 Árbol Sintáctico para la cadena 000111



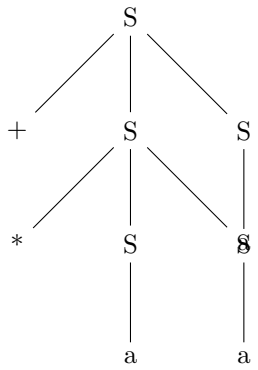
—

(c) Gramática: $S \to +SS \mid *SS \mid a$

3.3.1 Lenguaje Generado

Esta gramática genera expresiones aritméticas en notación prefija ('+' y '*' son operadores, 'a' es operando).

3.3.2 Árbol Sintáctico para la cadena $+*aaa$



Ejercicio 4

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

Explicación

Se genera un lenguaje balanceado de 'x' y 'y', asegurando correspondencia de apertura y cierre.

Gramática

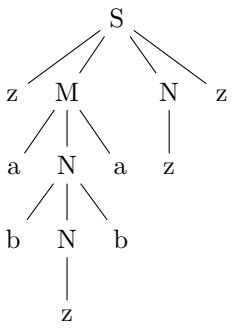
$$S \to zMNz$$

$$M \to aNa$$

$$N \to bNb$$

$$N \to z$$

Árbol Sintáctico



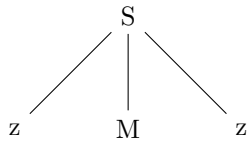
Ejercicio 5

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

Explicación

La gramática genera estructuras donde ‘b‘ siempre está rodeado de ‘z‘ o ‘a‘.

Árbol Sintáctico



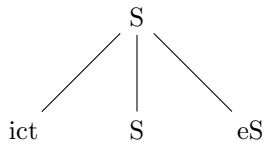
Ejercicio 6

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

Explicación

La ambigüedad se muestra con múltiples árboles posibles para la misma entrada.

Árbol Sintáctico



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)))

Explicación

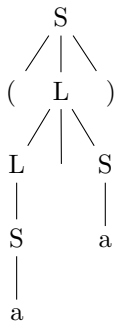
Se representan las listas anidadas.

Gramática

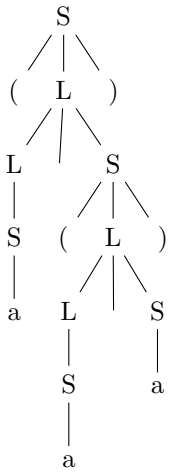
$$S \to (L) \quad | \quad a$$

$$L \to L, S \quad | \quad S$$

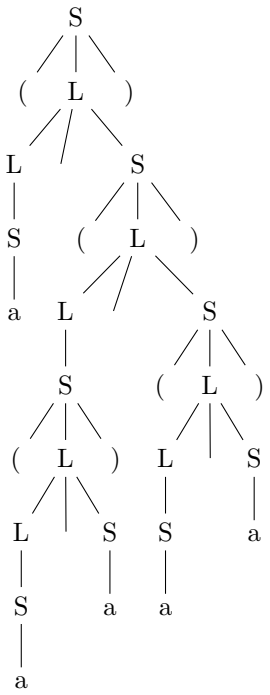
Árbol de Análisis para (a, a)



Árbol de Análisis para (a, (a, a))



Árbol de Análisis para (a, ((a, a), (a, a)))



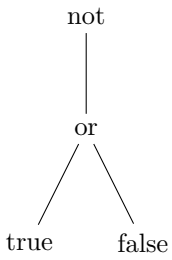
Ejercicio 8

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

Explicación

El operador ‘not‘ tiene ‘bexpr‘ como hijo.

Árbol Sintáctico



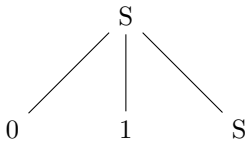
Ejercicio 9

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.

Explicación

Asegura que todo ‘0‘ tenga al menos un ‘1‘.

Árbol Sintáctico



Ejercicio 10

10.Elimine la recursividad por la izquierda de la siguiente gramática:
S → (L) — a
L → L , S — S

Explicación

Se elimina recursividad por la izquierda.

Resolución

$$S \rightarrow (L)|a$$

$$L \rightarrow SL'$$

$$L' \rightarrow, SL'|\varepsilon$$

Ejercicio 11

11. Dada la gramática S → (S) — x, escriba un pseudocódigo para el análisis sintáctico de esta gramática mediante el método descendente recursivo

Explicación

Se define un analizador descendente recursivo.

Código en Java

```
void S() {
  if (token == '(') {
    match('(');
    S();
    match(')');
  } else if (token == 'x') {
    match('x');
  } else {
    error();
  }
}
```

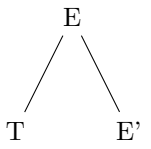
Ejercicio 12

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).

Explicación

Muestra el procesamiento de ‘LL(1)‘.

Árbol Sintáctico



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

Explicación

Se ajusta la gramática para incluir ‘-‘ y ‘/‘.

Resolución

$$E \to TE'$$

$$E' \to +TE'| - TE'|\varepsilon$$

$$T \to FT'$$

$$T' \to *FT'|/FT'|\varepsilon$$

$$F \to (E)|id|num$$

Ejercicio 14

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

Explicación

Se define un analizador recursivo para la gramática ajustada.

Gramática Modificada

La gramática extendida para incluir suma, resta, multiplicación y división, además de aceptar números como operandos:

$$E \rightarrow TE'$$

$$E' \rightarrow +TE' \mid -TE' \mid \varepsilon$$

$$T \rightarrow FT'$$

$$T' \rightarrow *FT' \mid /FT' \mid \varepsilon$$

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

Pseudocódigo

```
Procedimiento E()
    T()
    E'()

Procedimiento E'()
    Si token == '+' o token == '-'
        coincidir(token)
        T()
        E'()
    FinSi

Procedimiento T()
    F()
    T'()

Procedimiento T'()
    Si token == '*' o token == '/'
        coincidir(token)
        F()
        T'()
    FinSi

Procedimiento F()
    Si token == '('
        coincidir('(')
        E()
        coincidir(')')
    SinoSi token == 'id' o token == 'num'
        coincidir(token)
    Sino
        error()
    FinSi
```

Implementación en Java

```
public class Parser {
    private Token token;
```

```
    public void E() {
        T();
        E_();
    }

    public void E_() {
        if (token == Token.PLUS || token == Token.MINUS) {
            match(token);
            T();
            E_();
        }
    }

    public void T() {
        F();
        T_();
    }

    public void T_() {
        if (token == Token.TIMES || token == Token.DIVIDE) {
            match(token);
            F();
            T_();
        }
    }

    public void F() {
        if (token == Token.LPAREN) {
            match(Token.LPAREN);
            E();
            match(Token.RPAREN);
        } else if (token == Token.ID || token == Token.NUM) {
            match(token);
        } else {
            throw new RuntimeException("Error de sintaxis");
        }
    }

    private void match(Token expected) {
        if (token == expected) {
            token = getNextToken();
        } else {
            throw new RuntimeException("Error de sintaxis");
        }
    }
}
```