Análisis sintáctico. Ejercicios

Alumno: Norberto Hernández Cárdenas

Materia: Autómatas y Compiladores

Grupo: 3°

Semestre: 6°

Maestro: Eduardo Cornejo Velázquez

Instituto de Ciencias Básicas e Ingeniería

1 Ejercicios y Actividades

1.1 Actividad 1

a) Escriba una gramática que genere el conjunto de cadenas.
$\{s;, s;s;, s;s;s;, ...\}$

$$\begin{aligned} \text{s;} \quad &\rightarrow \quad \text{s;} \\ &\quad\;\; | \; \varepsilon \end{aligned}$$

b) Genere un árbol sintáctico para las cadena **s;s;**

Figure 1: Árbol sintáctico de la activiad 1

1.2 Actividad 2

Considere la siguiente gramática:

$$\begin{aligned} \text{rex} \quad &\rightarrow \quad \text{rex ” | ” rexp} \\ &\quad\;\; | \; \text{rexp rexp} \\ &\quad\;\; | \; \text{rexp ”*”} \\ &\quad\;\; | \; \text{”(” rexp ”)”} \\ &\quad\;\; | \; \textbf{letra} \end{aligned}$$

a) Genere un árbol sintáctico para la expresión regular (ab—b)*.

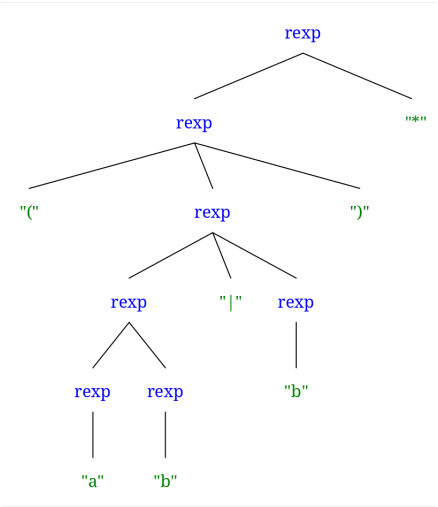

Figure 2: Árbol sintáctico de la activiad 2a

1.3 Actividad 3

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

$$S \to SS+ \mid SS* \mid a \text{ con la cadena } aa + a * .$$

Esta gramática genera un lenguaje conformado solamente por la letra "a" y de los símbolos "+" y "*". Este puede formar cadenas en las cuales se multiplican o suman las aa y siempre va quedar un más o asterisco del lado derecho.

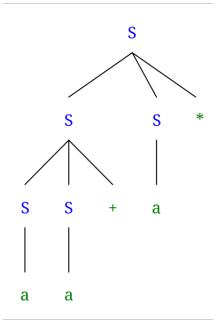

Figure 3: Árbol sintáctico de la activiad 3a

b)

$$S \to 0S1 \mid 01 \text{ con la cadena } 000111.$$

Esta cadena esta formada por una cantidad igual de 0 y 1, su dominio empieza desde 01 y puede llegar hasta una cantidad indefinida de 0 y 1.

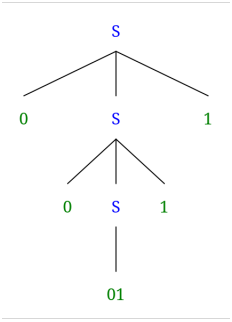

Figure 4: Árbol sintáctico de la activiad 3b

c)

$$S \to +SS \mid *SS \mid a \text{ con la cadena } + *aaa.$$

Las cadenas siempre van a estar conformadas por la letra "a", y de los símbolos "+" y "*", es muy similiar a la gramatica de a, donde las letras "a" se pueden encontrar siendo sumados o multiplicados y siempre van a estar acompañados por un símbolo solo que en esta ocasión siempre va haber un símbolo del lado izquierdo.

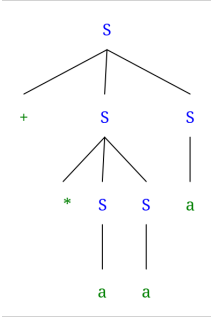

Figure 5: Árbol sintáctico de la activiad 3c

1.4 Actividad 4

¿Cuál es el lenguaje generado por la siguiente gramática?

$$S \rightarrow xSy \mid \varepsilon$$

A partir de S se puede producir "xSy", por ende cada vez que se produce esto se agregan 3 ramas, una a la izquierda con "x" y una a la derecha con "y", y en medio se puede empezar otra rama o terminar el arbol ahi.
S tambien puede ser reemplazado por la cadena vacía, esto indicando que la rama del arbol ahi termina.
El lenguaje que se termina generando es el conjunto de todas las cadenas que tienen tanto el mismo numero de x que de y.

$$L = \{x^n y^n \mid n \geq 0\}$$

1.5 Actividad 5

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

$$\begin{array}{ccc} S & \rightarrow & zMNz \\ M & \rightarrow & aNa \\ N & \rightarrow & bNb \\ N & \rightarrow & z \end{array}$$

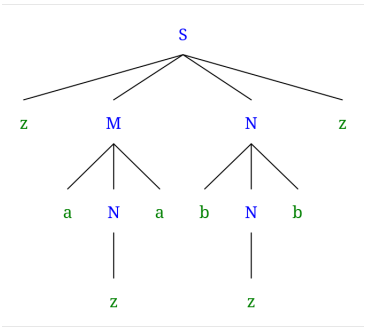

Figure 6: Árbol sintáctico de la activiad 5

1.6 Actividad 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

$$
\begin{aligned}
S &\rightarrow \text{ictS} \\
S &\rightarrow \text{ictSeS} \\
S &\rightarrow \text{s}
\end{aligned}
$$

La gramatica especifica que partiendo de "S" se puede continuar con 3 posibles caminos, en cuales 2 todavia se puede seguir con el arbol, siendo el que termina las ramas de los arboles es la tercera opción donde el valor final es "s".

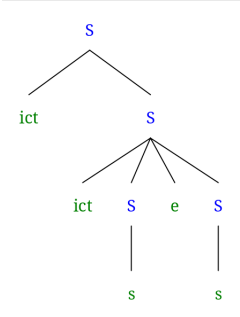

Figure 7: Árbol sintáctico de la activiad 6

1.7 Actividad 7

Considere la siguiente gramática

$$
\begin{aligned}
S &\rightarrow (L) \mid a \\
L &\rightarrow L\ ,S \mid S
\end{aligned}
$$

Encuéntrense árboles de análisis sintáctico para las siguientes frases:

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

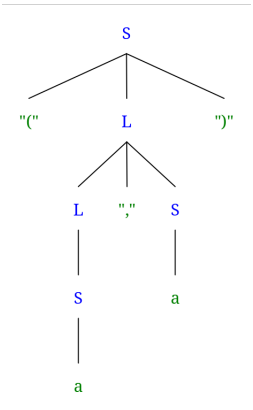

Figure 8: Árbol sintáctico de la activiad 7a

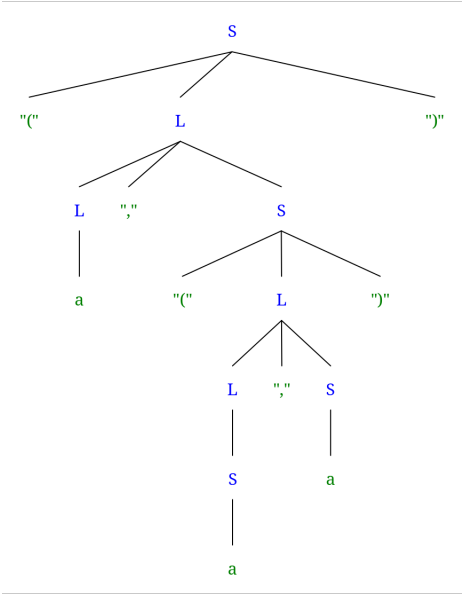

Figure 9: Árbol sintáctico de la activiad 7b

b)

c)

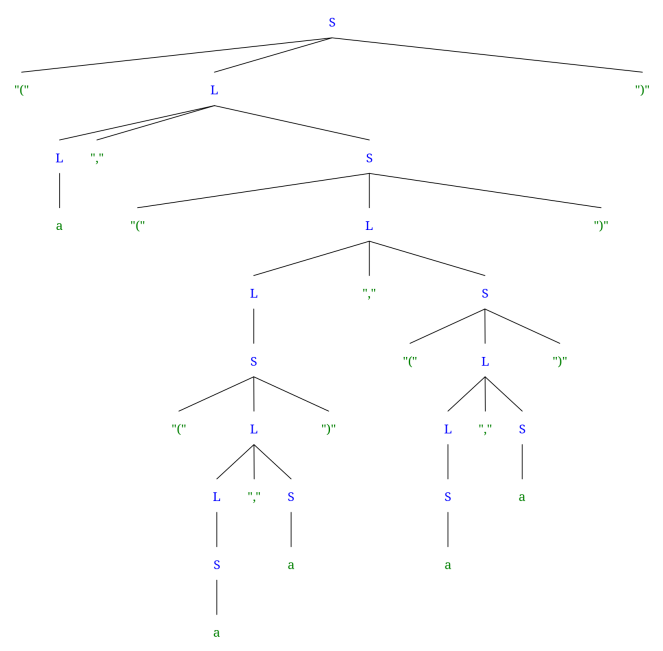

Figure 10: Árbol sintáctico de la activiad 7c

1.8 Actividad 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**

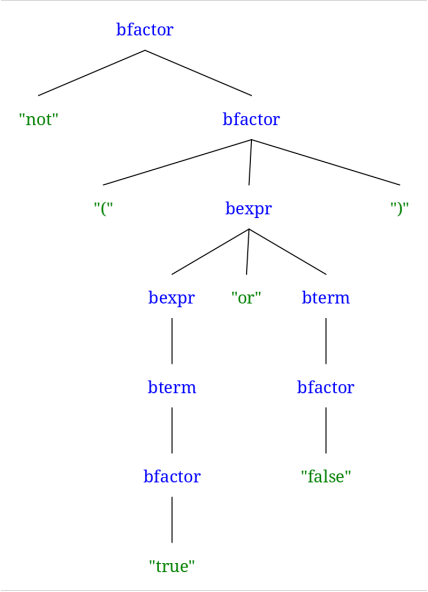

Figure 11: Árbol sintáctico de la activiad 8

1.9 Actividad 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.

$$
\begin{aligned}
S &\rightarrow 0L \mid L \\
L &\rightarrow 1L \mid 1S \mid \varepsilon
\end{aligned}
$$

1.10 Actividad 10

Elimine la recursividad por la izquierda de la siguiente gramática:

$$
\begin{aligned}
S &\rightarrow (L) \mid a \\
L &\rightarrow L\ ,S \mid S
\end{aligned}
$$

$$
\begin{aligned}
S &\rightarrow (L) \mid a \\
L &\rightarrow S\ L' \\
L' &\rightarrow ,S\ L' \mid \varepsilon
\end{aligned}
$$

1.11 Actividad 11

Dada la gramática S → (S) | **x**, escriban un pseudocódigo para el análisis sintáctico de esta gramática mediante em método descendente recursivo.

```
1: function ANALIZAR_S
2:     if siguiente_caracter() == '(' then
3:         consumir('(')                          ▷ Consumir el carácter dado
4:         analizar_S()                           ▷ Llamada recursiva para analizar S
5:         if siguiente_caracter() == ')' then
6:             consumir(')')
7:         else
8:             IMPRIMIR("Se esperaba ')'")
9:         end if
10:    else if siguiente_caracter() == 'x' then
11:        consumir('x')
12:    else
13:        IMPRIMIR("Se esperaba '(' o 'x'")
14:    end if
15: end function
16: function PRINCIPAL
17:    inicializar_entrada()                      ▷ Se recibe la cadena de entrada
18:    analizar_S()                               ▷ Inicializa el análisis
19:    if siguiente_caracter() == FIN_DE_ENTRADA then
20:        IMPRIMIR("La cadena es válida")
21:    else
22:        IMPRIMIR("La cadena no es válida")
23:    end if
24: end function
```

1.12 Actividas 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.

- Exp → **(** Exp **)**
- Exp → Exp Op Exp
- Exp → **id**
- Exp → **num**
- Op → **+** | **-** | ***** | **/**

1. Exp Op Exp [Exp → Exp Op Exp]
2. (Exp) Op Exp [Exp → (Exp)]
3. (Exp) Op **id** [Exp → **id**]
4. (Exp) * **id** [Op → *]
5. (Exp Op Exp) * **id** [Exp → Exp Op Exp]
6. (Exp Op **id**) * **id** [Exp → **id**]
7. (Exp + **id**) * **id**[Op → *]
8. (**id** + **id**) * **id**[Exp → **id**]

1.13 Actividad 13

La gramática 3.2, sólo maneja las operaciones de suma y multiplicacióm, 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**

- E → E + S | E - S | S
- S → S * F | S / F | F
- F → (E) | **id** | **num**

Luego eliminamos la recursividad por la izquierda.

- E → SE'
- E' → +TE' | -TE' | ε
- T → FT'
- T' → *FT' | /FT' | ε
- F → (E) | **id** | **num**

1.14 Actividad 14

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

```
1: Entrada: Una cadena de entrada input.
2: Salida: Verdadero si la cadena es válida y falso si la cadena no es válida.
3: function PARSE(input)
4:     index ← 0
5:     if E() and index == length(input) then
6:         return true                                   ▷ La cadena es válida
7:     else
8:         return false                                  ▷ La cadena no es válida
9:     end if
10: end function
11: function E()
12:     if not S() then
13:         return false
14:     end if
15:     if not E_PRIME() then
16:         return false
17:     end if
18:     return true
19: end function
20: function E_PRIME
21:     if MATCH('+') then
22:         if not S then
23:             return false
24:         end if
25:         if not E_PRIME then
26:             return false
27:         end if
28:         return true
29:     else if MATCH('-') then
30:         if not S then
31:             return false
32:         end if
33:         if not E_PRIME then
34:             return false
35:         end if
36:         return true
37:     else
38:         return true
39:     end if
40: end function
41: function S
42:     if not F then
43:         return false
44:     end if
45:     if not S_PRIME then
46:         return false
47:     end if
48:     return true
49: end function
```

```
function S_PRIME
    if MATCH('*') then
        if not F then
            return false
        end if
        if not S_PRIME then
            return false
        end if
        return true
    else if MATCH('/') then
        if not F then
            return false
        end if
        if not S_PRIME then
            return false
        end if
        return true
    else
        return true
    end if
end function
function F
    if MATCH('(') then
        if not E then
            return false
        end if
        if not MATCH(')') then
            return false
        end if
        return true
    else if ISIDORNUM then
        CONSUMEIDORNUM
        return true
    else
        return false
    end if
end function
function MATCH(expected)
    if index < length(input) and input[index] == expected then
        index ← index + 1
        return true
    else
        return false
    end if
end function
function ISIDORNUM
    if index ≥ length(input) then
        return false
    end if
    currentChar ← input[index]
    return ISLETTER(currentChar) or ISDIGIT(currentChar)
end function
```

```
1: function consumeIdOrNum
2:     while index < length(input) and isLetterOrDigit(input[index]) do
3:         index ← index + 1
4:     end while
5: end function
```

A continuación unas capturas de pantalla del programa en Java en ejecución.

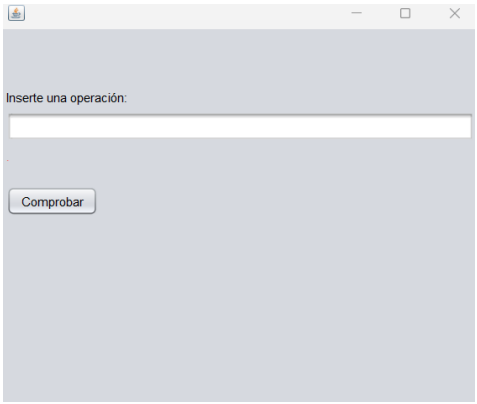

Figure 12: La ineterfaz del programa final.

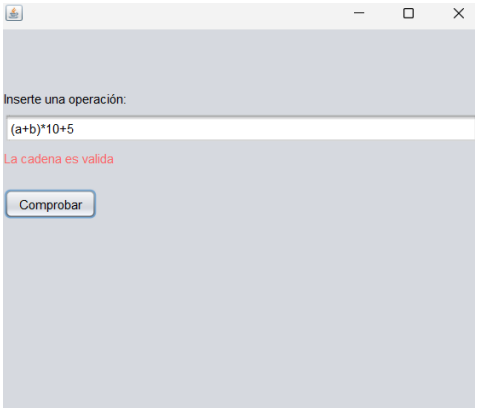

Figure 13: Probando el programa con la entrada "(a+b)*10+5".

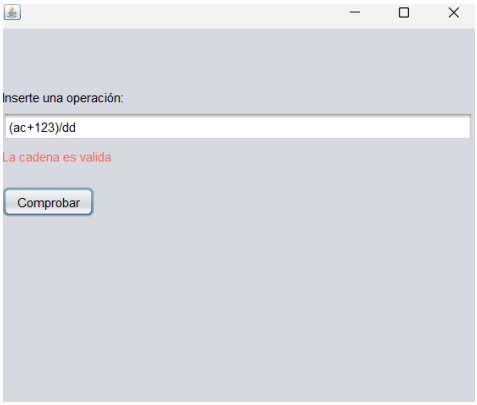

Figure 14: Probando el programa con la entrada "(ac+123)/dd".

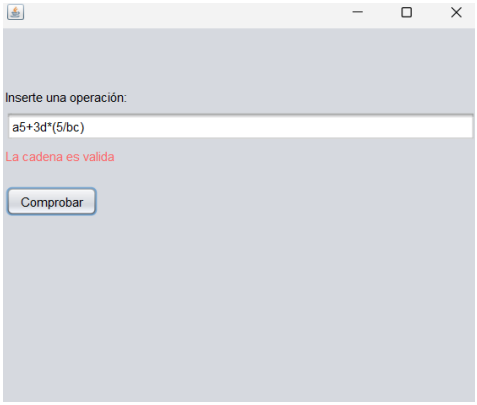

Figure 15: Probando el programa con la entrada "a5+3d*(5/bc)".

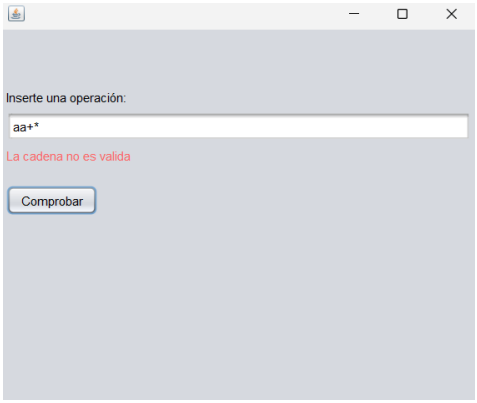

Figure 16: Probando el programa con la entrada "aa+*".

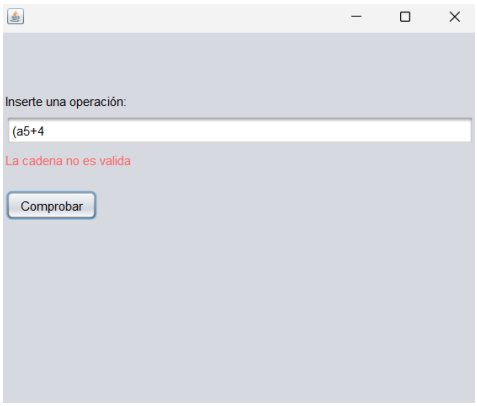

Figure 17: Probando el programa con la entrada "(a5+4".

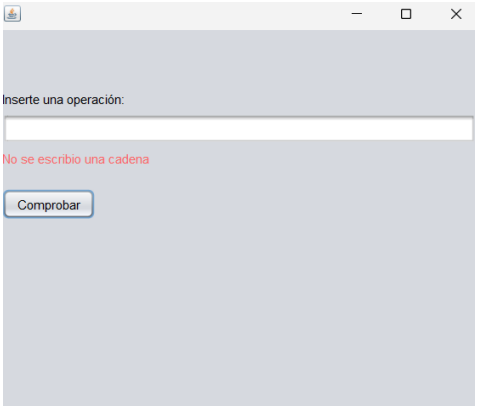

Figure 18: Probando el programa con una cadena vacía.

2 Bibliografía

- Carranza. D. (2024).*Compiladores Fases de análisis.* Ciudad de Querétaro, México. Editorial Transdigital. Página 64-69.