

# Your Paper

You

March 24, 2025

## 1 Ejercicios y Actividades

### 1.1 Actividad 1

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

$$\begin{array}{lcl} s; & \rightarrow & s; \\ & & | \varepsilon \end{array}$$

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



Figure 1: Árbol sintáctico de la actividad 1

### 1.2 Actividad 2

Considere la siguiente gramática:

$$\begin{array}{lcl} \text{rex} & \rightarrow & \text{rex " | " rexp} \\ & & | \text{rexp rexp} \\ & & | \text{rexp "*" } \\ & & | \text{"(" rexp ")"} \\ & & | \text{letra} \end{array}$$

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

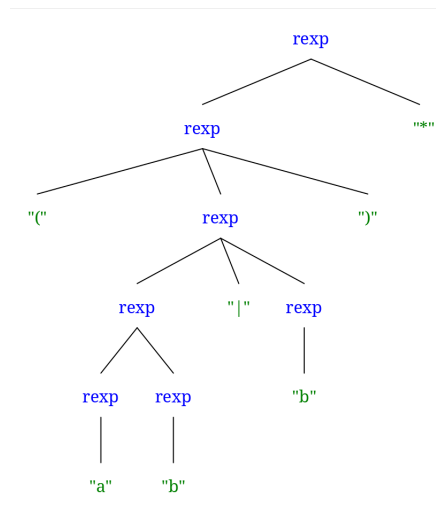


Figure 2: Árbol sintáctico de la activiad 2a

### 1.3 Actividad 3

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

a)

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

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.

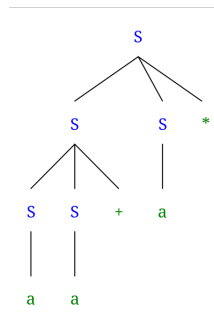


Figure 3: Árbol sintáctico de la activiad 3a

b)

$$S \rightarrow OS1 \mid 01 \text{ con la cadena } 000111.$$

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.

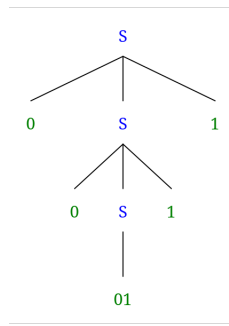


Figure 4: Árbol sintáctico de la actividad 3b

c)

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

Las cadenas siempre van a estar conformadas por la letra "a", y de los símbolos "+" y "\*", es muy similar a la gramática 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.

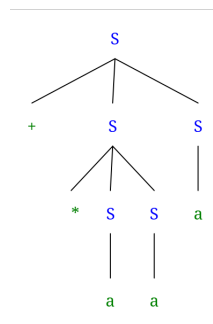


Figure 5: Árbol sintáctico de la actividad 3c

## 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 árbol ahí.

S también puede ser reemplazado por la cadena vacía, esto indicando que la rama del árbol ahí termina. El lenguaje que se termina generando es el conjunto de todas las cadenas que tienen tanto el mismo número 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:

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

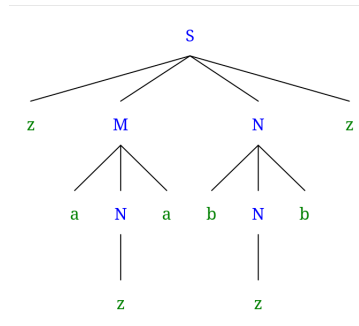


Figure 6: Árbol sintáctico de la actividad 5

## 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

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

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

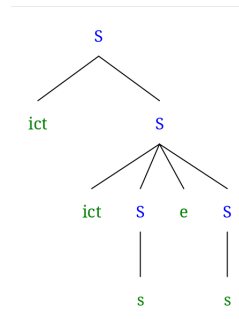


Figure 7: Árbol sintáctico de la actividad 6

## 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)

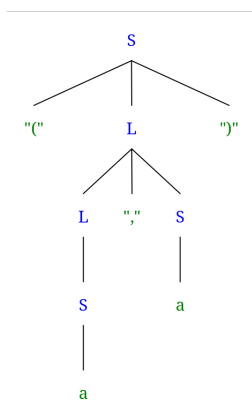


Figure 8: Árbol sintáctico de la actividad 7a

b)

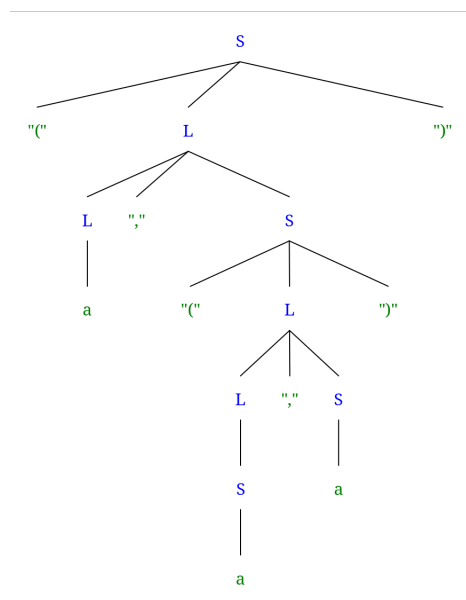
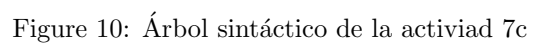


Figure 9: Árbol sintáctico de la actividad 7b

c)



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

6

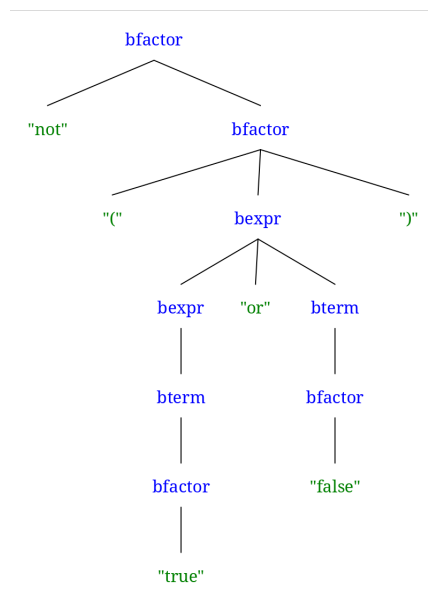


Figure 11: Árbol sintáctico de la actividad 8

## 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:

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

## 1.11 Actividad 11

Dada la gramática  $S \rightarrow (S) \mid \mathbf{x}$ , 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 Actividad 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.

- $\text{Exp} \rightarrow (\text{Exp})$
- $\text{Exp} \rightarrow \text{Exp Op Exp}$
- $\text{Exp} \rightarrow \text{id}$
- $\text{Exp} \rightarrow \text{num}$
- $\text{Op} \rightarrow + \mid - \mid * \mid /$

1.  $\text{Exp Op Exp} [\text{Exp} \rightarrow \text{Exp Op Exp}]$
2.  $(\text{Exp}) \text{ Op Exp} [\text{Exp} \rightarrow (\text{Exp})]$
3.  $(\text{Exp}) \text{ Op id} [\text{Exp} \rightarrow \text{id}]$
4.  $(\text{Exp}) * \text{id} [\text{Op} \rightarrow *]$
5.  $(\text{Exp Op Exp}) * \text{id} [\text{Exp} \rightarrow \text{Exp Op Exp}]$
6.  $(\text{Exp Op id}) * \text{id} [\text{Exp} \rightarrow \text{id}]$
7.  $(\text{Exp} + \text{id}) * \text{id} [\text{Op} \rightarrow +]$
8.  $(\text{id} + \text{id}) * \text{id} [\text{Exp} \rightarrow \text{id}]$



### 1.13 Actividad 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,  $F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$

- $E \rightarrow E + S \mid E - S \mid S$
- $S \rightarrow S * F \mid S / F \mid F$
- $F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$

Luego eliminamos la recursividad por la izquierda.

- $E \rightarrow SE'$
- $E' \rightarrow +TE' \mid -TE' \mid \varepsilon$
- $T \rightarrow FT'$
- $T' \rightarrow *FT' \mid /FT' \mid \varepsilon$
- $F \rightarrow (E) \mid \mathbf{id} \mid \mathbf{num}$

### 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  $\leftarrow$  0
5:   if E() and index == length(input) then
6:     return true
7:   else
8:     return false
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

```

---

▷ La cadena es válida  
 ▷ La cadena no es válida

---

```

1: function S_PRIME
2:   if MATCH('*') then
3:     if not F then
4:       return false
5:     end if
6:     if not S_PRIME then
7:       return false
8:     end if
9:     return true
10:  else if MATCH('/') then
11:    if not F then
12:      return false
13:    end if
14:    if not S_PRIME then
15:      return false
16:    end if
17:    return true
18:  else
19:    return true
20:  end if
21: end function
22: function F
23:   if MATCH('(') then
24:     if not E then
25:       return false
26:     end if
27:     if not MATCH(')') then
28:       return false
29:     end if
30:     return true
31:   else if ISIDORNUM then
32:     CONSUMEIDORNUM
33:     return true
34:   else
35:     return false
36:   end if
37: end function
38: function MATCH(expected)
39:   if index < length(input) and input[index] == expected then
40:     index ← index + 1
41:     return true
42:   else
43:     return false
44:   end if
45: end function
46: function ISIDORNUM
47:   if index ≥ length(input) then
48:     return false
49:   end if
50:   currentChar ← input[index]
51:   return ISLETTER(currentChar) or ISDIGIT(currentChar)
52: end function

```

---

---

```
1: function CONSUMEIDORNUM
2:   while  $index < \text{length}(input)$  and ISLETTERORDIGIT( $input[index]$ ) do
3:      $index \leftarrow index + 1$ 
4:   end while
5: end function
```

---

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

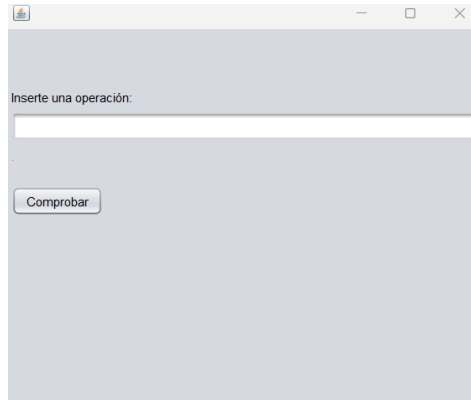


Figure 12: La ineterfaz del programa final.

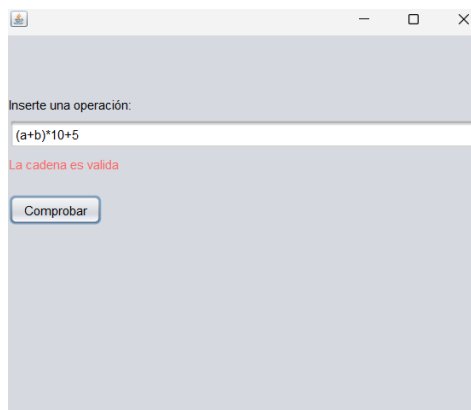


Figure 13: Probando el programa con la entrada "(a+b)\*10+5".

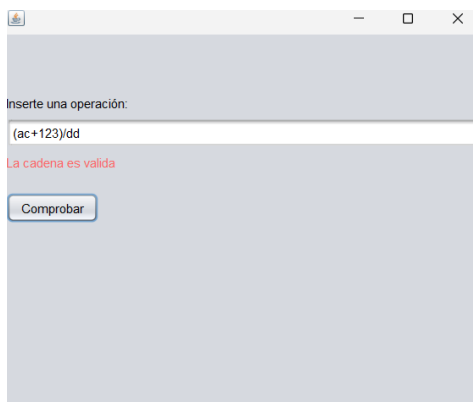


Figure 14: Probando el programa con la entrada " $(ac+123)/dd$ ".

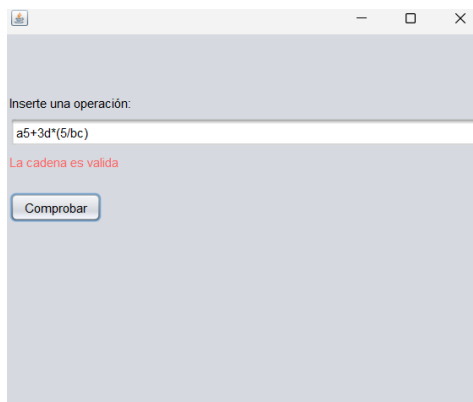


Figure 15: Probando el programa con la entrada " $a5+3d*(5/bc)$ ".

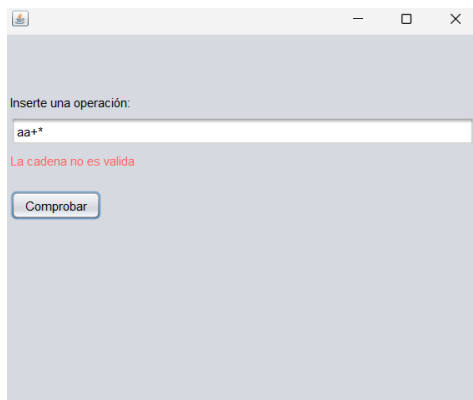


Figure 16: Probando el programa con la entrada " $aa+^*$ ".

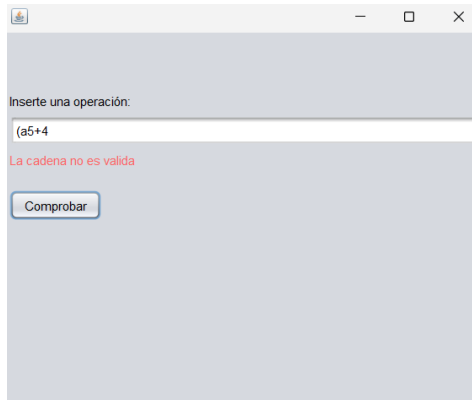


Figure 17: Probando el programa con la entrada "(a5+4)".

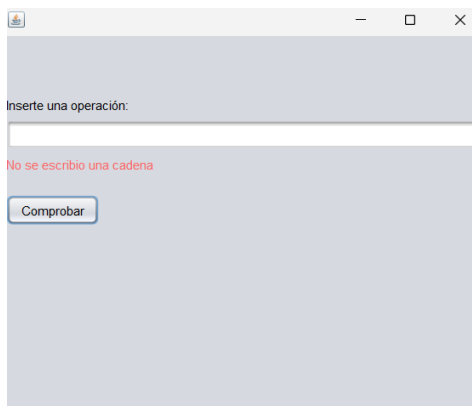


Figure 18: Probando el programa con una cadena vacía.

## 2 Bibliografía

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