Diseño de lenguajes de programación
Febrero , 2026

Actvididad 3. Automatas en la fase de compilación

Los autómatas finitos (tanto deterministas como no deterministas) son herramientas fundamentales en la fase de análisis léxico de un compilador. Un autómata actúa como un reconocedor de patrones que determina si una secuencia de caracteres pertenece a un lenguaje específico, permitiendo identificar tokens válidos del código fuente. Durante el escaneo, el autómata transita entre estados según los caracteres de entrada: comienza en un estado inicial, realiza transiciones basadas en cada carácter leído, y determina si la secuencia es válida al alcanzar un estado de aceptación. Por ejemplo, un autómata puede reconocer si una cadena es un identificador válido (variable1), un número (3.14), o una palabra reservada (public). Los autómatas finitos deterministas (DFA) son especialmente eficientes porque garantizan exactamente una transición por símbolo de entrada, permitiendo un reconocimiento en tiempo lineal $O(n)$. La conversión de expresiones regulares a autómatas mediante algoritmos como Thompson (regex → NFA) o construcción directa (regex → DFA) es esencial para implementar analizadores léxicos eficientes, y la minimización de DFA asegura que el autómata resultante tenga el menor número de estados posible, optimizando tanto memoria como velocidad de procesamiento.

Lea cuidadosamente cada inciso y realizar lo solicitado.

- Para la parte de generación crear un repositorio en github y recuerde tenerlo organizado y separado por clases o módulos que considere necesarios.
- Incluya en su repositorio un video breve explicando su código y funcionamiento.
- Para la parte escrita entregar un pdf con los ejercicios realizados de manera legible y ordenada.

Problema 1: 50%

Construcción de DFA mediante el algoritmo de nullable, firstpos, lastpos y followpos

El algoritmo de construcción directa de un DFA a partir de una expresión regular utiliza las funciones nullable, firstpos, lastpos y followpos para construir el autómata sin pasar primero por un NFA. Este método asigna posiciones únicas a cada símbolo de la expresión regular y calcula sistemáticamente los estados y transiciones del DFA resultante.
Para cada expresión regular proporcionada, debe:

- Construya el árbol sintáctico de la expresión regular.
- Calcule las funciones correspondientes para cada nodo del árbol. Presentelos en una tabla para facilidad de lectura.
- Construya la tabla de transiciones resultante de usar las funciones previamente calculadas.
- Construya el DFA correspondiente
- Minimice el DFA resultante utilizando el algoritmo de partición de estados, recuerde el orden visto en clase.

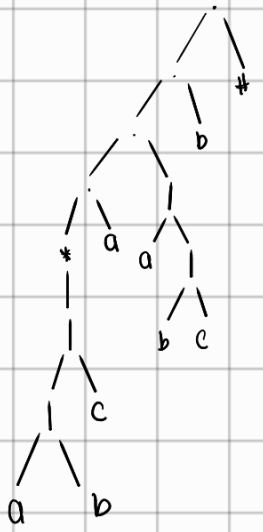
Realice el proceso completo para las siguientes expresiones regulares:

1. $(a|b|c)^*a(a|b|c)b$
2. $1^*0(1|00)(0|1)^*$

Link del Repositorio en donde se encuentra completo: https://github.com/Emadlgg/diseo_lenguajes/tree/7f9b2ca20cc66d4630b8544b40ded52778fbca4e/Actividad_3

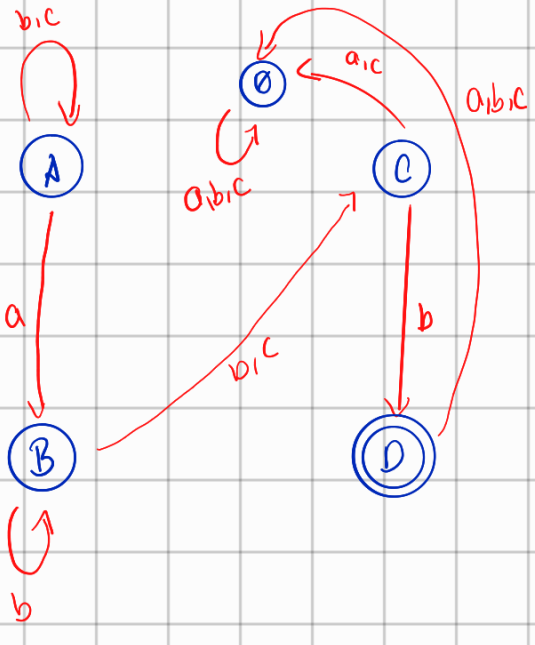
Actividad 3 – Parte 1

Expresión 1:
$(a|b|c)^*a(a|b|c)b\#$



nodo	Expresión	nullable	firstpos	lastpos
1	a	no	1	1
2	b	no	2	2
3	c	no	3	3
4	a	no	4	4
5	a	no	5	5
6	b	no	6	6
7	c	no	7	7
8	b	no	8	8
9	#	no	9	9
10	a|b	no	(1,2)	(1,2)
11	(a|b)|c	no	(1,2,3)	(1,2,3)
12	a|b	no	(5,6)	(5,6)
13	(a|b)|c	no	(5,6,7)	(5,6,7)
14	(a|b|c)*	Si	(1,2,3)	(1,2,3)
15	(a|b|c)*a	no	(1,2,3,4)	4
16	n15 · (a|b|c)	no	(1,2,3,4)	(5,6,7)
17	n16 · b	no	(1,2,3,4)	8
18	n17 · #	no	(1,2,3,4)	9

Estado	a	b	c
A	B	A	A
B	B	C	C
C	∅	D	∅
D	∅	∅	∅

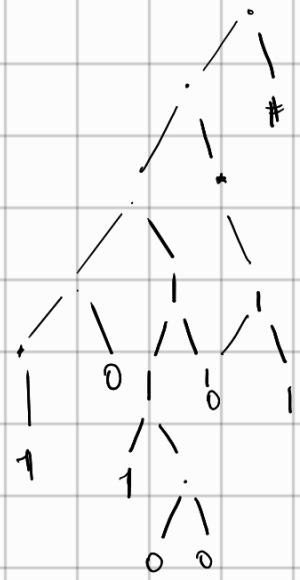


El AFD yaes el mismo

Expresion 2:

$1^*0(1|00)(011)^*\#$

1	0	1	0	0	0	1	#
1	2	3	4	5	6	7	8



nodo	Expresion	nullable	firstpos	lastpos
1	1	no	1	1
2	0	no	2	2
3	1	no	3	3
4	0	no	4	4
5	0	no	5	5
6	0	no	6	6
7	1	no	7	7
8	#	no	8	8
9	1^*	Si	1	1
10	$(011)^*$	Si	(6,7)	(6,7)
11	$1\|(0\cdot0)$	no	(3,4)	(3,5)
12	$0\|1$	no	(6,7)	(6,7)
13	1^*0	no	(1,2)	2
14	$n13\cdot(1100)$	no	(1,2)	(3,5)
15	$n14\cdot(011)$	no	(1,2)	(3,5,6,7)
16	$n15\cdot\#$	no	(1,2)	(8)