**Práctica 1:**

Diseño e implementación de las clases AFN, AFD

**Alumno:**

Meza Vargas Brandon David

**Grupo:**

3CM13

**Profesor:**

Sánchez Juárez José

**Índice**

[**Objetivo** 3](#_Toc121764762)

[**Actividades** 3](#_Toc121764763)

[**Cuestionario** 5](#_Toc121764764)

[**Conclusiones** 15](#_Toc121764765)

# **Objetivo**

Se crearán las clases, para el AFN y para el AFD

# **Actividades**

Se tiene la expresión regular c\* b (c | bc\* b)\*. Usar el siguiente código para programar los tokens que acepta el AFD.

Programa AFD. Para ejecutar este programa se hace con la linea de comando: java DriverAFD cadena de entrada

**import java.util.\*;**

**public class DriverAFD {**

**public static void main(String[] args) {**

**ArgumentosToken tm = new ArgumentosToken(args);**

**AFD m = new AFD(tm);**

**m.activarAFD();**

**}**

**}**

**class ArgumentosToken {**

**private int indice;**

**String entrada;**

**public ArgumentosToken(String[] args) {**

**if(args.length > 0)**

**entrada = args[0];**

**else**

**entrada = "";**

**indice = 0;**

**System.out.println("entrada = " + entrada);**

**}**

**public char getSiguienteToken() {**

**if(indice < entrada.length())**

**return entrada.charAt(indice++);**

**else**

**return '$';**

**}**

**}**

**class AFD {**

**ArgumentosToken tm;**

**private char tokenActual;**

**public AFD(ArgumentosToken tm) {**

**this.tm = tm;**

**}**

**public void avanzar() {**

**tokenActual = tm.getSiguienteToken();**

**}**

**public void activarAFD() {**

**int estadoActual = 0;**

**avanzar();**

**while(tokenActual != '$') {**

**switch(estadoActual) {**

**case 0:**

**if(tokenActual == 'b') estadoActual = 1;**

**else if(tokenActual == 'c') estadoActual = 1;**

**break;**

**case 1:**

**if(tokenActual == 'b') estadoActual = 2;**

**else if(tokenActual == 'c') estadoActual = 3;**

**break;**

**case 2:**

**if(tokenActual == 'b') estadoActual = 4;**

**else if(tokenActual == 'c') estadoActual = 4;**

**break;**

**case 3:**

**if(tokenActual == 'b') estadoActual = 4;**

**else if(tokenActual == 'c') estadoActual = 3;**

**break;**

**case 4:**

**if(tokenActual == 'b') estadoActual = 4;**

**else if(tokenActual == 'c') estadoActual = 4;**

**break;**

**}**

**avanzar();**

**}**

**if(estadoActual == 2 || estadoActual == 3)**

**System.out.println("Acep");**

**else**

**System.out.println("rechazado");**

**}**

}

# **Cuestionario**

1**. Realizar todo el proceso de transformación de la expresión regular hasta la tabla de transiciones del AFD. De tal manera que se compruebe el programa DriverAFD**

**c\*b(c|bc\*b)\***

Lo primero que se hará será obtener el AFN de la ER para así obtener el AFN, reducirlo y obtener su tabla de transiciones.

**AFN**

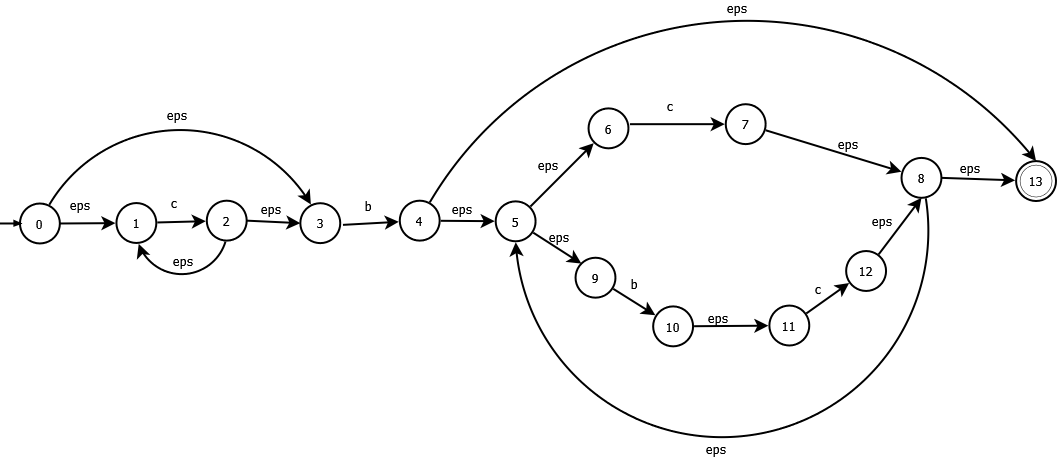
****

Ilustración . AFN de ER

Ahora haremos el proceso para obtener el AFN

**[Σ = {b, c}](https://es.wikipedia.org/wiki/%CE%A3" \l ":~:text=Sigma%20(%CE%A3%20%CF%83%20%CF%82)%20es,se%20usa%20la%20forma%20%CF%83.)**

Aplicando cerr-eps

Cerr-eps((0)) = {0, ,3, 1} = q0

q0:

q0 = {0, ,3, 1}

cerr-eps(mov(q0,b)) = cerr-eps({4}) = {4, 5, 6, 9, 13} = q1

cerr-eps(mov(q0,c)) = cerr-eps({2}) = {2, 3} = q2

q1:

q1 = {4, 5, 6, 9, 13}

cerr-eps(mov(q1,b)) = cerr-eps({10}) = {10, 11} = q3

cerr-eps(mov(q1,c)) = cerr-eps({7}) = {7, 8, 5, 6, 9, 13} = q4

q2:

q2 = {2, 3}

cerr-eps(mov(q2,b)) = cerr-eps({4}) = {4, 5, 6, 9, 13} = q1

cerr-eps(mov(q2,c)) = cerr-eps({Θ}) = { Θ }

q3:

q3 = {10, 11}

cerr-eps(mov(q3,b)) = cerr-eps({Θ }) = { Θ}

cerr-eps(mov(q3,c)) = cerr-eps({12}) = { 12, 8, 13, 5, 6, 9 } = q5

q4:

q4 = {7, 8, 5, 6, 9, 13}

cerr-eps(mov(q4,b)) = cerr-eps({10}) = {10,11} = q3

cerr-eps(mov(q4,c)) = cerr-eps({7}) = {7 , 8, 5, 6, 9, 13} = q4

q5:

q5 = {12, 8, 13, 5, 6, 9, 13}

cerr-eps(mov(q5,b)) = cerr-eps({10}) = {10, 11} = q3

cerr-eps(mov(q5,c)) = cerr-eps({7}) = { 7, 8, 5, 6, 9, 13} = q4

**AFD**

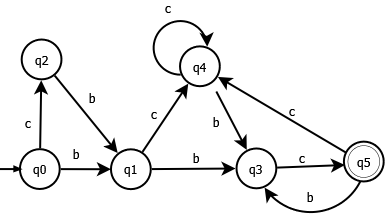


Ilustración . AFD

**Minimizando AFD**

M0 = {q2}

M1 = {q1, q4, q5}

M2 = {q3}

M3 = {q0}

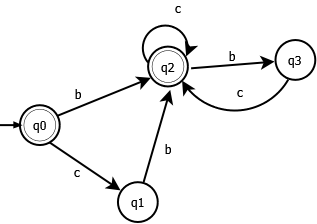


Ilustración . AFD minimizado

Así podemos poner a prueba el programa con distintas cadenas

Imagen que contiene Texto

Descripción generada automáticamente

Ilustración . Poniendo a prueba el programa

**2. Obtener el programa para reconocer las palabras clave if, else y return**

Para esta parte vamos a realizar el procedimiento del árbol con todas las palabras clave para llegar a la tabla de transiciones y poder construir el programa.

**IF**

Aumentando la ER

**if#**

Creando árbol, calculando la última y primera posición y obteniendo anulables

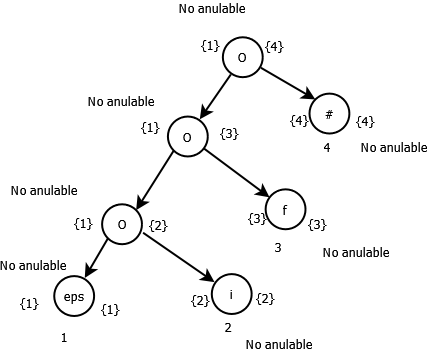


Ilustración . Árbol

Obteniendo siguiente posición

|  |  |
| --- | --- |
| Nodo i | Siguiente pos |
| 1 | {2} |
| 2 | {3} |
| 3 | {4} |
| 4 | {5} |
| 5 | - |

Calculando estados y tran y tranD de todo el vocabulario

**Primerapos(raíz) = {2} = q0**

**Tran[q0, i] = siguientepos(2) = {3} = q1**

**Tran[q0, f] = {** ø **}**

Con q1

**Tran [q1, i] = {** ø **}**

**Tran [q1,f] = {3}**

**tranD[q1, f] = siguientepos(3) = {4} = q2**

Obteniendo AFD mínimo

**Imagen en blanco y negro

Descripción generada automáticamente con confianza media**

Ilustración . AFD

Tabla de transiciones:

|  |  |  |
| --- | --- | --- |
| Estado | I | F |
| q0 | q1 | - |
| q1 | - | q2 |
| q2 | - | - |

Lo siguiente que se muestra es la parte del código modificada para esta expresión regular.

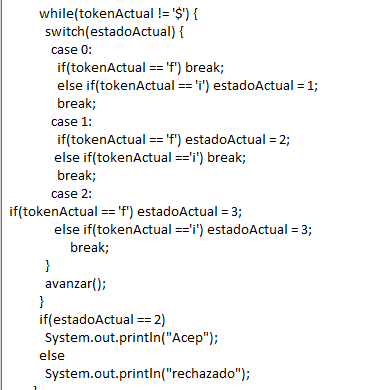


Ilustración . Código if

Y al probarlo tenemos:

Imagen que contiene Texto

Descripción generada automáticamente

Ilustración . Probando programa

Para las otras palabras clave, no será necesario realizar todo el proceso ya que al ser puras concatenaciones resulta en un afd muy sencillo con una tabla de transiciones igual de sencilla, por lo que solo se incluirá el AFD mínimo y su tabla de transiciones con sus pruebas del programa correspondientes.

**else**

**Imagen en blanco y negro

Descripción generada automáticamente con confianza baja**

Ilustración . AFD mínimo

|  |  |  |  |
| --- | --- | --- | --- |
| Estado | e | l | s |
| q0 | q1 | - | - |
| q1 | - | q2 | - |
| q2 | - | - | q3 |
| q3 | q4 | - | - |
| q4 | - | - | - |

El programa modificado queda de la siguiente manera:

Texto

Descripción generada automáticamente

Ilustración . Programa else

Y las pruebas del programa:

**Texto

Descripción generada automáticamente con confianza media**

Ilustración . Pruebas else

**return**

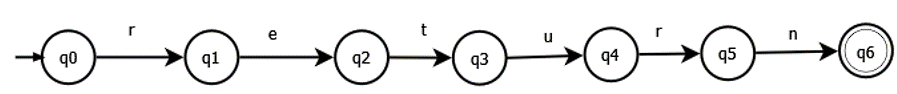
****

Ilustración 12. AFD mínimo

Aquí la tabla de transiciones

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Estado | r | e | t | u | N |
| q0 | q1 | - | - | - | - |
| q1 | - | q2 | - | - | - |
| q2 | - | - | q3 | - | - |
| q3 | - | - | - | q4 | - |
| q4 | q5 | - | - | - | - |
| q5 | - | - | - | - | q6 |
| q6 | - | - | - | - | - |

El programa modificado queda de la siguiente manera:

Una captura de pantalla de un celular de un mensaje en letras blancas

Descripción generada automáticamente con confianza baja

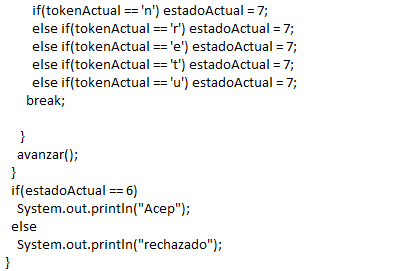


Ilustración 13. Programa return

Y las pruebas del programa:

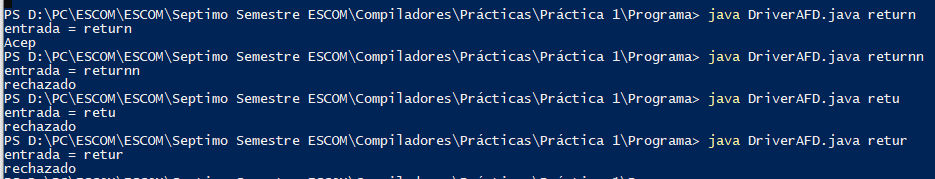
****

Ilustración 14. Pruebas else

# **Conclusiones**

Gracias a esta práctica reforcé mis conocimientos sobre pasar una er a afd y obtener su tabla de transiciones, esta última resulta muy importante pues gracias a esta se puede realizar un programa el cual nos indica que cadena acepta o no la expresión regular original, fue muy interesante implementar este programa y probarlo con varias expresiones regulares, entre ellas palabras clave que son usadas siempre cuando programamos.