TECNOLÓGICO NACIONAL DE MEXICO

INSTITUTO TECNOLÓGICO DE IZTAPALAPA

INTEGRANTES:

CUANENEMI CUANALO MARIO ALBERTO	181080030
FERMIN CRUZ ERIK	181080007
GUTIERREZ ARELLANO RAFAEL	181080022
PEREZ ARMAS FAUSTO ISAAC	181080037

ISC-6AM

LENGUAJES Y AUTOMATAS I

M.C. ABIEL TOMÁS PARRA HERNÁNDEZ

SEP 2020 / FEB 2021

ACTIVIDAD SEMANA 10

CUANENEMI CUANALO MARIO ALBERTO

Un autómata finito (AF) o máquina de estado finito es un modelo computacional que realiza cómputos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados finito, una función de transición, un estado inicial y un conjunto de estados finales. Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.
La conversión de un AFND-ε en un AFND se basa en el concepto de clausura-ε, que corresponde a una clausura transitiva contextualizada en la teoría de autómatas.

Dado un estado que se llama clausura-ε(q) al conjunto de todos los estados a los que se puede acceder a partir de que, procesando a lo más un único símbolo de la entrada. Puede definirse recursivamente de la siguiente manera:

(Base inductiva) Para todo estado q, $q \in$ clausura-ε(q).
(Inducción) Dados dos estados p y r, si p ∈ clausura-ε(q) y r ∈ δ(p,ε), entonces r ∈ clausura-ε(q).
El algoritmo para eliminar las transiciones vacías es el siguiente:

1. Se calcula la clausura-ε del estado inicial, formándose un conjunto *A* que corresponderá al estado inicial del nuevo autómata.
2. Para cada símbolo del alfabeto, se verifican los estados alcanzables a partir de algún estado contenido en *A*, y se calcula la clausura-ε de dichos estados alcanzables. Si dichas clausuras producen nuevos conjuntos distintos de *A*, estos serán nuevos estados a los que se accederá a partir de *A* y del símbolo correspondiente.
3. Se repite lo anterior para cada nuevo conjunto, hasta que no existan transiciones posibles para ningún símbolo del alfabeto.

Ejemplo

En el ejemplo de la figura, se tendrá inicialmente:

clausura-ε(*1*) = {1,2,3,4,6} = *A*
PARA A
Para el símbolo a: 4 va a 5, y clausura-ε(5) = {5,7} = B.
Para el símbolo b: no existen transiciones posibles.
Para B:
Para el símbolo a: no existen transiciones posibles.
Para el símbolo b: 5 va a 6, y clausura-ε(6) = {6} = C.
Para C:
Para el símbolo a: no existen transiciones posibles.
Para el símbolo b: no existen transiciones posibles.

En algunos casos puede ocurrir que al quitar las transiciones épsilon obtengamos directamente un AFD, pues la única razón de no-determinismo era justamente la presencia de dichas transiciones.
Todo AFND (QN, Σ, q0, δN, FN) puede convertirse en un AFD (QD, Σ, q0, δD, FD) equivalente, que mantiene el alfabeto Σ y el estado inicial q0 originales. La conversión implica pasar por un AFD intermedio con estados y transiciones redundantes, que al no ser accesibles a partir del estado inicial, son eliminados para obtener el AFD definitivo.

1. Primero se redefine el conjunto de estados $Q_N = \{q_0, q_1, ..., q_m\}$ original, como uno conformado por todos los subconjuntos de Q_N. Los nuevos estados finales serán todos aquellos estados que contengan a alguno de los estados finales originales.
2. Posteriormente, se redefine el conjunto de transiciones original, por transiciones del tipo $\delta_D(S,a)$, donde $a \in \Sigma$, y S es la unión de todos los estados q de Q_N para los cuales existía la transición $\delta_N(q,a)$.
3. Por último, se eliminan los **estados inaccesibles** o **inalcanzables** (junto con sus transiciones de salida), es decir, aquellos a los que no se puede acceder a partir del estado inicial. Luego de esta depuración, se obtiene el AFD final.

Minimización de un AFD

Dos estados de un autómata finito determinista son **estados equivalentes** si al unirse en un solo estado, pueden reconocer el mismo lenguaje regular que si estuviesen separados. Esta unión de estados implica la unión tanto de sus transiciones de entrada como de salida. Si dos estados no son equivalentes, se dice que son **estados distinguibles**. Un estado final con un estado no-final nunca serán equivalentes.

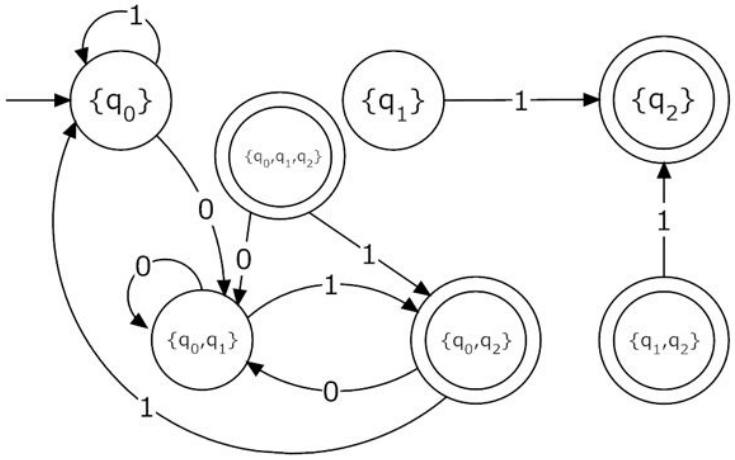
Un AFD está minimizado, si todos sus estados son *distinguibles* y *alcanzables*. Un algoritmo de minimización de AFD es el siguiente:

1. Eliminar los estados inaccesibles del autómata.
2. Construir una tabla con todos los pares (p, q) de estados restantes.

3. Marcar en la tabla aquellas entradas donde un estado es final y el otro es no-final, es decir, aquellos pares de estados que son claramente distinguibles.
4. Para cada par (p, q) y cada símbolo a del alfabeto, tal que $r = \delta(p,a)$ y $s = \delta(q,a)$:
    1. Si (r, s) ya ha sido marcado, entonces p y q también son distinguibles, por lo tanto marcar la entrada (p, q).
    2. De lo contrario, colocar (p, q) en una lista asociada a la entrada (r, s).
5. Agrupar a los pares de estados no marcados.

Luego del tercer paso, si la tabla creada queda completamente marcada, entonces el AFD inicial ya era mínimo.

La complejidad computacional del problema de minimizar un AFD es polinomial. De hecho, existen algoritmos más eficientes aún que el mostrado en este artículo (aunque menos intuitivos). Sin embargo, el problema de minimizar un autómata finito no determinista es NP-completo y PSPACE-completo.



FERMIN CRUZ ERIK

EXPRESIONES REGULARES

Es un equivalente algebraico para un autómata utilizado en muchos lugares como un lenguaje para describir patrones en texto que son sencillos pero muy útiles.

Pueden definir exactamente los mismos lenguajes que los autómatas pueden describir: Lenguajes regulares

Ofrecen algo que los autómatas no: Manera declarativa de expresar las cadenas que queremos aceptar.

Ejemplos de sus usos:

- Comandos de búsqueda, e.g., grep de UNIX

- Sistemas de formateo de texto: Usan notación de tipo expresión regular para describir patrones

- Convierte la expresión regular a un DFA o un NFA y simula el autómata en el archivo de búsqueda.

- Generadores de analizadores-léxicos. Como Lex o Flex.

- Los analizadores léxicos son parte de un compilador. Dividen el programa fuente en unidades lógicas(tokens), como while, n´umeros, signos (+,−,<, etc.)

- Produce un DFA que reconoce el token.

OPERANDOS

Si E y F son expresiones regulares, entonces E +F también lo es denotando la unión de L(E) y L(F).L(E+F) = L(E) $\cup$ L(F).

Si E y F son expresiones regulares, entonces EF también lo es denotando la concatenación de L(E) y L (F).L(EF) =L(E)L(F).

EJEMPLO PARA TRANSFORMAR UN DFA EN UNA EXPRESIÓN REGULAR

Ahora, vamos a ver uno de los métodos que se usan para transformar autómatas finitos deterministas en expresiones regulares, el método de eliminación de estados. Cuando tenemos un autómata finito, determinista o no determinista, podemos considerar que los símbolos que componen a sus transiciones son expresiones regulares. Cuando eliminamos un estado, tenemos que reemplazar todos los caminos que pasaban a través de él como transiciones directas que ahora se realizan con el ingreso de expresiones regulares, en vez de con símbolos. Los casos bases son los siguientes:

Realmente, el retorno podría verse como un caso particular de la unión, en donde q0 y q1 son el mismo estado. De esta forma, el camino que va directo desde q0 a q0 es “Y” y el que va desde q0 a q0 a través de q2 es “VW*X”.

A la hora de reducir un autómata, se recomienda partir eliminando primero todos los estados que no sean ni el de inicial ni los finales. Cuando se eliminen todos estos estados y el autómata tenga más de un estado inicial, se deben hacer tantas copias como estados de aceptación tenga el autómata.

En cada una de las copias, se debe elegir uno de los estados de aceptación diferentes. Todos los demás estados de aceptación de esta copia pasarán a ser estados ordinarios. Ahora se deben reducir todos los autómatas copias a expresiones regulares. La expresión regular final será la unión de todas las expresiones regulares resultantes de cada una de las copias.

GUTIERREZ ARELLANO RAFAEL

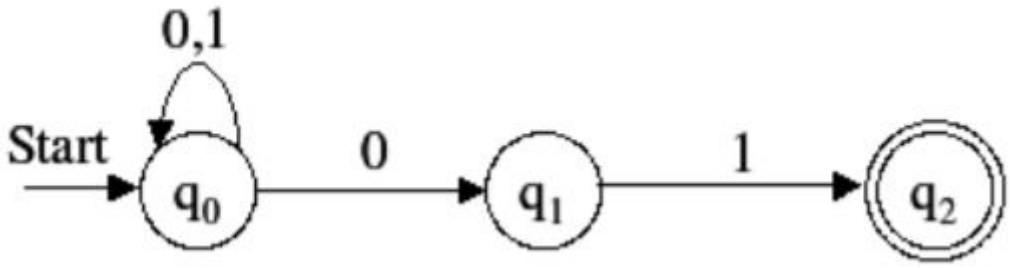
8 RE-FA

A mediados de los años 60, General Motors, preocupada por los elevados costos de los sistemas de control a base de relés, de lógica cableada, comenzó a trabajar con Digital en el desarrollo de un sistema de control que evitará los inconvenientes de la lógica programada. El resultado de la colaboración fue un equipo programado, denominado PDP-14, cuyo empleo no tardó en extenderse a otras industrias. En un principio, los autómatas programables solo trabajaban con control discreta (Si o No), por lo que los problemas que requerían la manipulación de magnitudes analógicas se dejaron para los tradicionales sistemas de control distribuido.

Funciones de transici´on extendidas (ˆδ)
Intuitivamente, un FA acepta una cadena w = a1a2 . . . an si hay una ruta en el diagrama de transiciones que:

1. Empieza en el estado de inicio.
2. Termina en un estado de aceptación.
3. Tiene una secuencia de etiquetas a1, a2, . . . , an.

Ejemplo: El siguiente AF acepta la cadena 01101:



Formalmente, extendemos la funci´on de transici´on δ a ˆδ(q, w), donde w puede ser cualquier cadena de s´ımbolos de entrada:

Base: ˆδ(q, ϱ) = q (i.e., nos quedamos en el mismo lugar si no recibimos una entrada).

• Induccion: ˆδ(q, w) = δ(ˆδ(q, x), a), donde x es una cadena, y a es un solo s´ımbolo (i.e., ver a donde va el AF con x, luego buscar la transición para el últimos sımbolo a partir de ese estado).

• Hecho importante con una prueba inductiva directa: ˆδ realmente representa rutas. Esto es, si w = a1a2 . . . an, y δ(pi , ai) = pi+1, ∀i = 0, 1, . . . , n − 1, entonces ˆδ(p0, w) = pn.

Aceptaci´on de Cadenas: Un AF A = (Q, Σ, δ, q0, F) acepta la cadena w si ˆδ(p0, w) est´a en F.

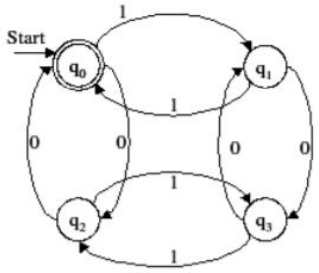
Lenguaje de un AF: Un AF acepta el lenguaje L(A) = w| ˆδ(p0, w) ∈ F. Algunas confusiones frecuentes Una gran fuente de confusión cuando se trabaja con autómatas (o matem´aticas en general) son los “errores de tipo”:

• Ejemplo: No confundir A, un FA, i.e., un programa, con L(A), el cual es del tipo “conjunto de cadenas”.

• Ejemplo: No confundir A, un FA, i.e., un programa, con L(A), el cual es del tipo “conjunto de cadenas”.

• Ejemplo engañoso: ¿Es un símbolo o una cadena de longitud 1? Respuesta: Depende del contexto, i.e., se usa en δ(q, a), donde es un s´ımbolo, o en ˆδ(q, a), donde es una cadena?

Ejemplo: DFA que acepta todas y s´olo las cadenas que tienen un n´umero par de 0’s y tambi´en un n´umero par de 1’s

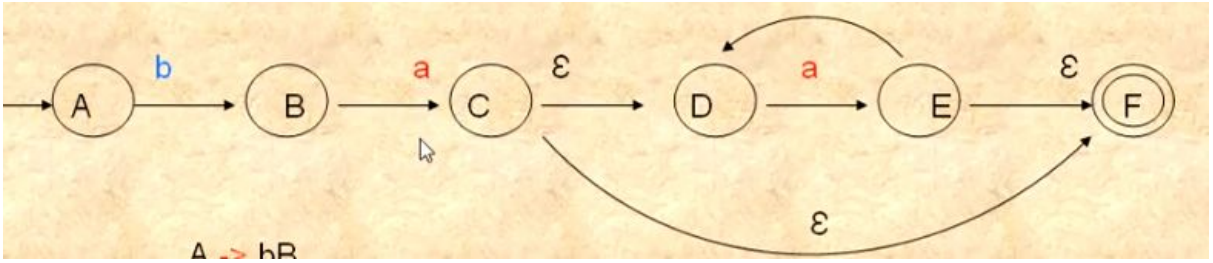


- $\hat{\delta}(q_0, \epsilon) = q_0$.
- $\hat{\delta}(q_0, 1) = \delta(\hat{\delta}(q_0, \epsilon), 1) = \delta(q_0, 1) = q_1$.
- $\hat{\delta}(q_0, 11) = \delta(\hat{\delta}(q_0, 1), 1) = \delta(q_1, 1) = q_0$.
- $\hat{\delta}(q_0, 110) = \delta(\hat{\delta}(q_0, 11), 0) = \delta(q_0, 0) = q_2$.
- $\hat{\delta}(q_0, 1101) = \delta(\hat{\delta}(q_0, 110), 1) = \delta(q_2, 1) = q_3$.
- $\hat{\delta}(q_0, 11010) = \delta(\hat{\delta}(q_0, 1101), 0) = \delta(q_3, 0) = q_1$.
- $\hat{\delta}(q_0, 110101) = \delta(\hat{\delta}(q_0, 11010), 1) = \delta(q_1, 1) = q_0$.

Representación tabular del autómata anterior:

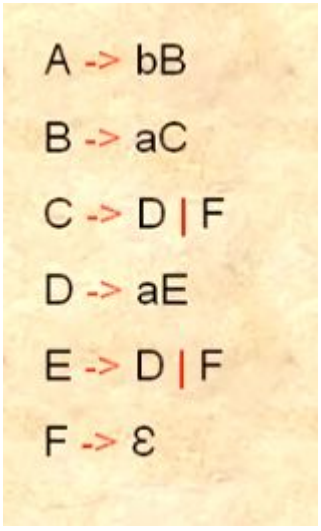
	0	1
$\cdot \rightarrow q_0$	q_2	q_1
q_1	q_3	q_0
q_2	q_0	q_3
q_3	q_1	q_2

Ejemplo: Problema 2.2.1.a
Ejemplo: Problema 2.2.4.a



Gramática regular por la derecha, cada una de las flechas de la transición corresponde con una producción de la gramática.

EJEMPLO:



En cada producción solo tenemos una variable.

Pasan por diferentes estados donde todos intervienen.

Se concatena en nuestro autómata una cadena vacía.

En paralelo se obtiene otra flecha de transición.

El conjunto de variables será **V={A,B,C,D,E,F}** conjunto de Terminales **T={a,b}** La producción es igual a la imagen de la izquierda y el símbolo inicial sería **s=A.**

PEREZ ARMAS FAUSTO ISAAC

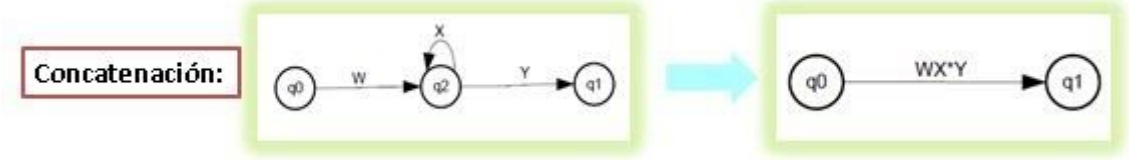
Los lenguajes descritos por expresiones regulares son los lenguajes reconocidos por los autómatas finitos. Existe un algoritmo para convertir una expresión regular en el autómata finito no determinístico correspondiente. El algoritmo construye a partir de la expresión regular un autómata con transiciones vacías, es decir un autómata que contiene arcos rotulados con ε. Luego este autómata con transiciones vacías se puede convertir en un autómata finito sin transiciones vacías que reconoce el mismo lenguaje.

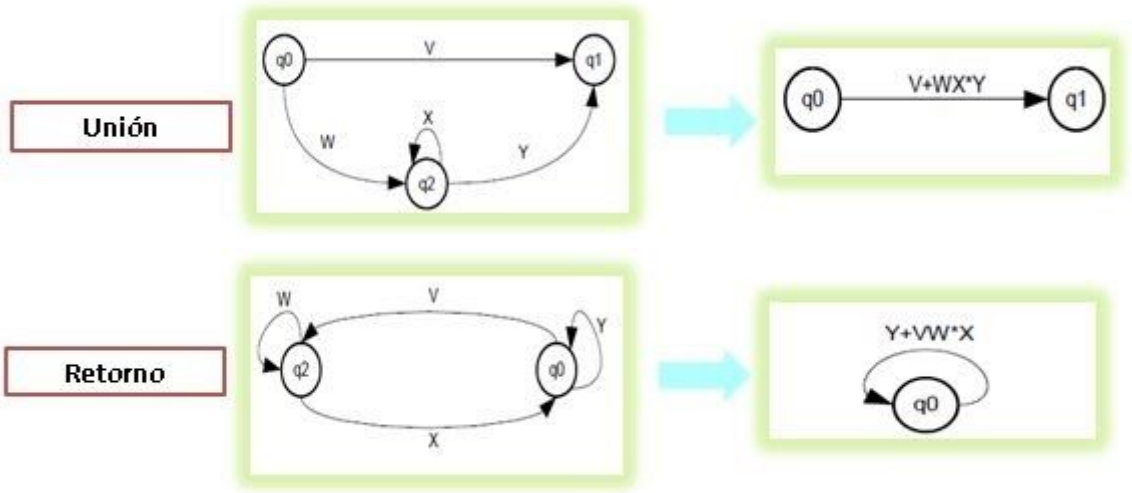
- Dada una expresión regular existe un autómata finito capaz de reconocer el lenguaje que ésta define.
- Recíprocamente, dado un autómata finito, se puede expresar mediante una expresión regular del lenguaje que reconoce.

Conversión de un AFD en una expresión regular mediante la eliminación de estados

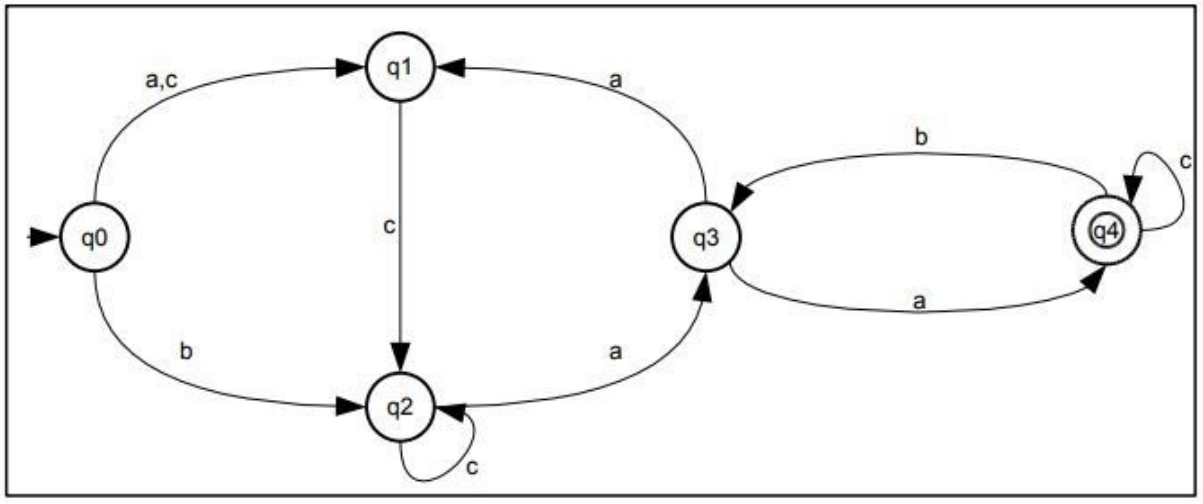
En este texto vamos a ver uno de los métodos que se usan para transformar autómatas finitos deterministas en expresiones regulares, el método de eliminación de estados.
Cuando tenemos un autómata finito, determinista o no determinista, podemos considerar que los símbolos que componen a sus transiciones son expresiones regulares. Cuando eliminamos un estado, tenemos que reemplazar todos los caminos que pasaban a través de él como transiciones
directas que ahora se realizan con el ingreso de expresiones regulares, en vez de con símbolos.

Los casos bases son los siguientes:

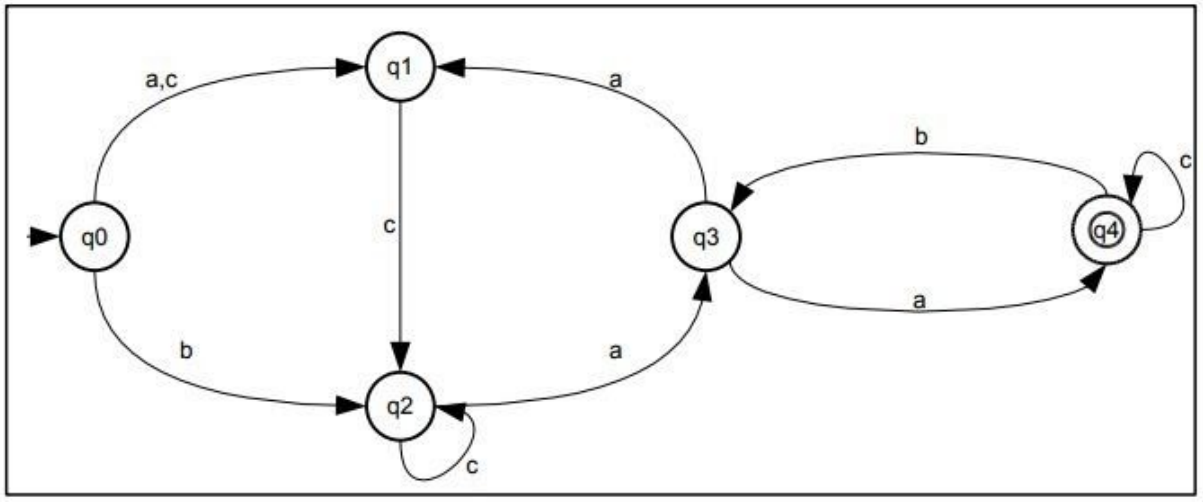




Ejemplo



1. Transformar todas aquellas transiciones que contemplan más de un símbolo en expresiones regulares del tipo “R+P.
Por lo tanto, la transición q_0 a q_1, queda como “a+c”. Todas las demás permanecen iguales, por ser símbolos.

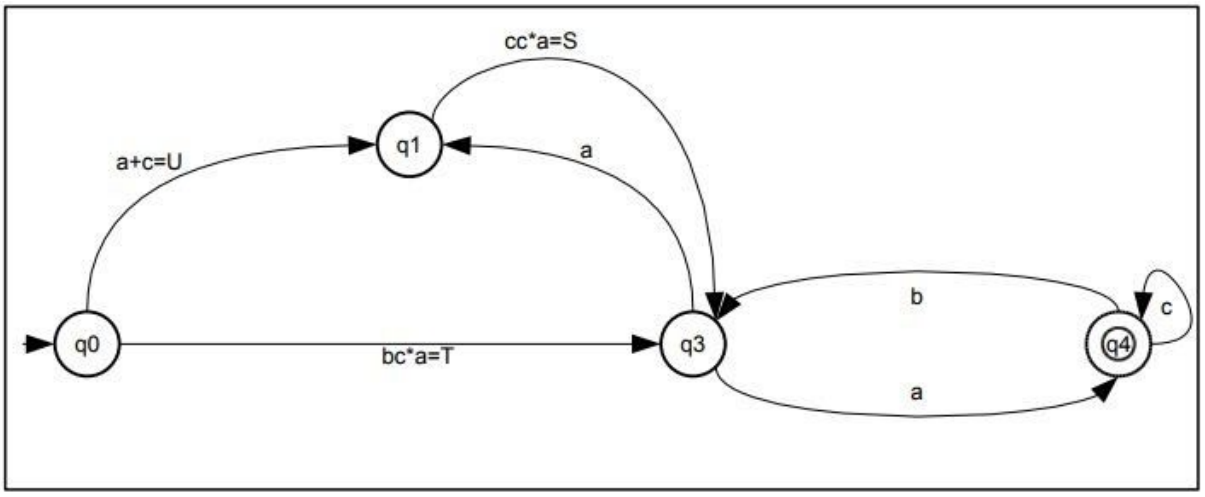


Se procede a eliminar el estado q_2. Verificar todos los caminos que atraviesan q_2:

- q_0 puede llegar a q_3 pasando por q_2.

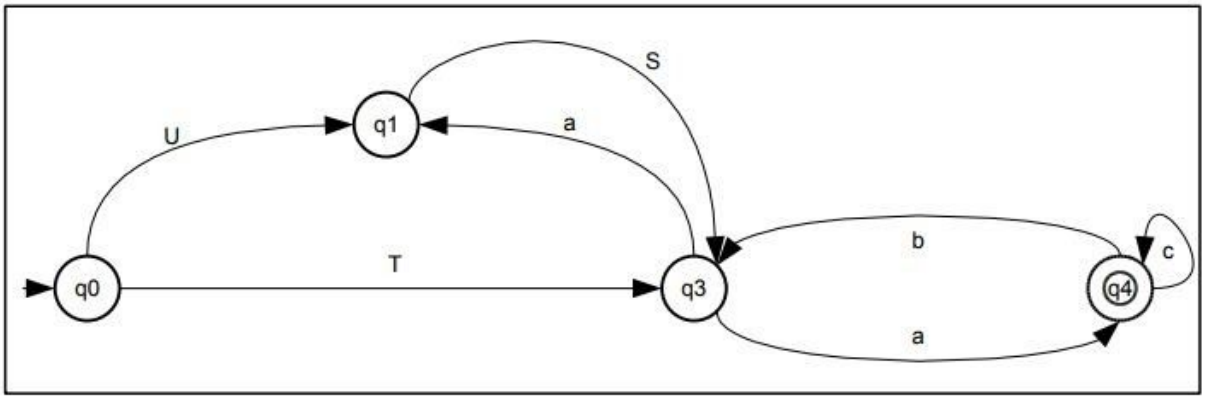
- q_1 puede llegar a q_3 pasando por q_2.

Al **eliminar es estado q2** las transiciones del AFD quedan:

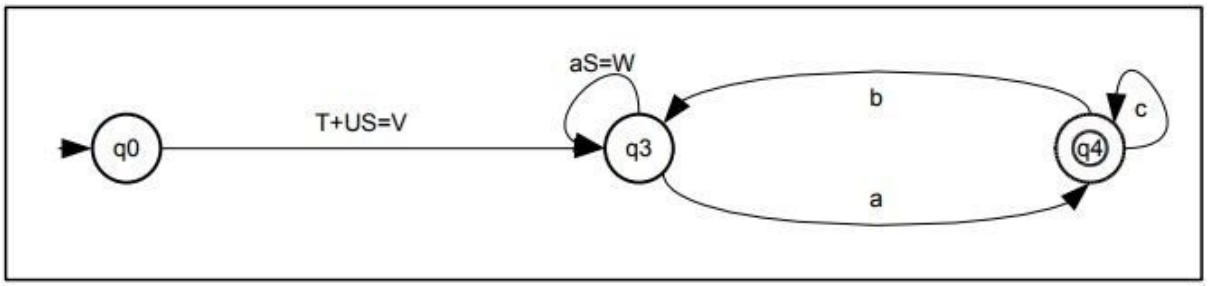


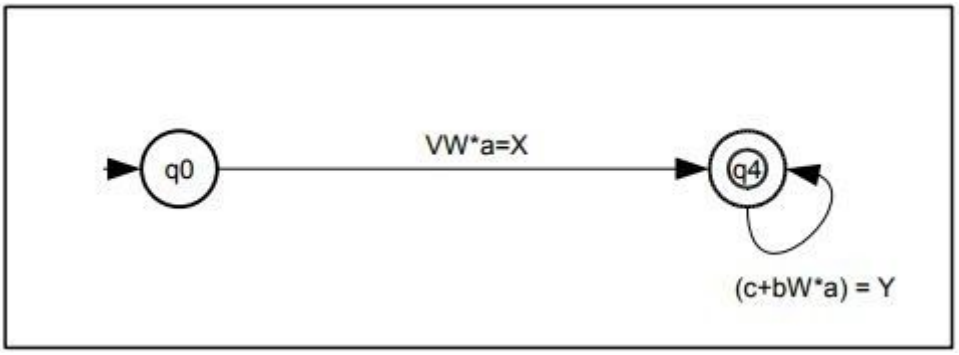
1) Para llegar a q_3, q_0 primero debe pasar por q_2 usando el símbolo “b”, luego puede pasar repetidas v
eces
por q_2 (o ninguna vez) usando cero, uno o varios símbolos “c” y, por último, pasa de q_2 a q_3 con el símbolo “a”. Esto queda expresado con la expresión regular
T = bc*a

2) Para pasar de q_1 a q_3, primero se debe llegar a q_2 con una “c”. Después de esto, se puede volver repetidas veces a q_2, usando el símbolo “c”.
Se puede pasar cero, una o muchas veces por q_2 de
esta forma.
Como último paso, se debe llegar desde q_2 a q_3 con una “a”. Al concatenar estos tres caminos se obtiene:
S = cc*a

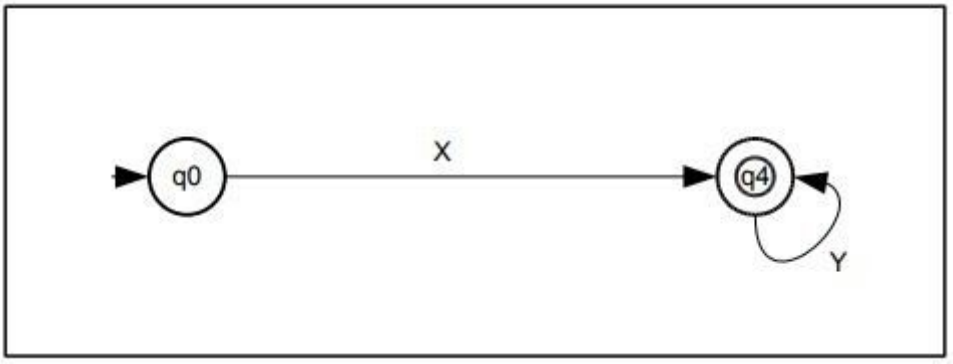


Al hacer las simplificaciones correspondiente, sólo se dejan las expresiones regulares resumidas por las letras mayúsculas (incluyendo a+c = U), y el autómata queda:
Eliminar q_1. Las transiciones afectadas son:
1) De q_0 a q_3, que puede hacerlo a través de q_1 o directamente, usando T. La expresión regular que va de q_0 a q_3 a través de q_1 es US y la expresión que va directamente de q_0 a q3 es T. Como puede ser una o la otra, como resultado queda la siguiente expresión: V = T + US
2) De q_3 a q_3, a través de q_1. Esto es: Primero llega a q_1 con una “a”, y luego pasa de q_1 a q_3, usando S. La expresión resultante es:
W = aS

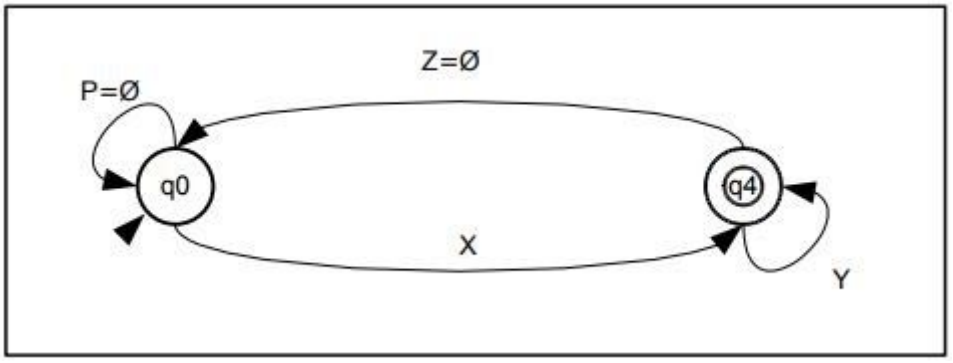




Eliminar q_3. Las transiciones afectadas son:
1) De q_0 a q_4, pasando por q_3: Primero va de q_0 a q_3 usando la expresión V. Luego de q_3 a q_3 repetidas veces, usando W. Por último, pasa de q_3 a q_4 usando una "a". La expresión regular queda : X = V.W*.a
2) De q_4 a q_4, pasando por q_3 o directamente por q4 con una "c". Para pasar a través de q_3, primero va de q_4 a q_3
con una "b". Luego de q_3 a q_3 usando W. Y, por último, pasa de q_3 a q_4 con una "a".
Como puede ir desde q_4 a q_4 usando la "c" o través de q_3, la expresión regular que queda es: Y = (c + bW*a)



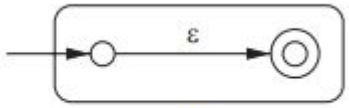
Al dejar las Expresiones Regulares de forma resumida, el último autómata queda:



Completando las transiciones vacías (de q_4 a q_1 y de q_1 a q_1) con las expresiones regulares vacías Z y P

De esta forma, se puede generar la expresión regular final a partir de la fórmula
(P + XY*Z)*XY*
Expresión Final = (P + XY*Z)*XY* = (Ø* + XY*Ø)* XY* = (ε+Ø)*XY* = ε*XY* = XY*

Conversión de una expresión regular a un autómata

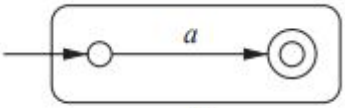


(a)

En la parte (a) vemos cómo se maneja la expresión ε. Puede verse fácilmente que el lenguaje del autómata es {ε}, ya que el único camino desde el estado inicial a un estado de aceptación está etiquetado con ε

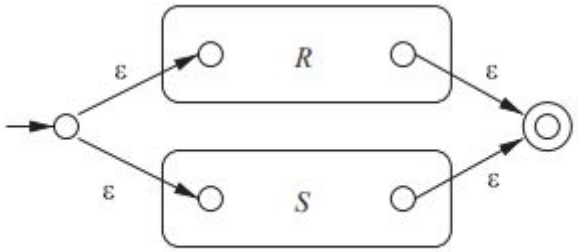
(b)

La parte (b) muestra la construcción de /0. Claramente, no existen caminos desde el estado inicial al de aceptación, por lo que /0 es el lenguaje de este autómata.



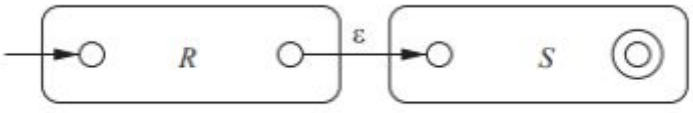
(c)

La parte (c) proporciona el autómata que reconoce una expresión regular a. Evidentemente, el lenguaje de este autómata consta de una cadena a, que es también L(a).



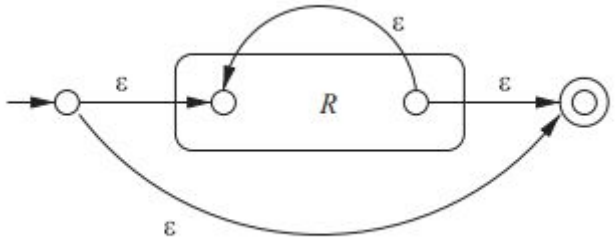
(a)

La expresión es de la forma R + S para dos expresiones R y S más pequeñas. Por tanto, el lenguaje del autómata de la (a) es L(R) ∪ L(S).



(b)

La expresión es de la forma RS para expresiones R y S más pequeñas. Por tanto, los caminos en el autómata de la (b) son todos y sólo los etiquetados con cadenas pertenecientes a L(R)L(S).



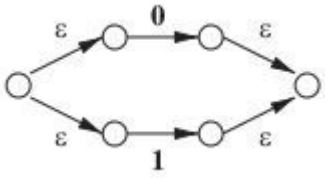
(c)

La expresión es de la forma R∗ para una expresión R más pequeña. Dicho camino acepta ε, que pertenece a L(R∗) sin importar qué expresión sea R.

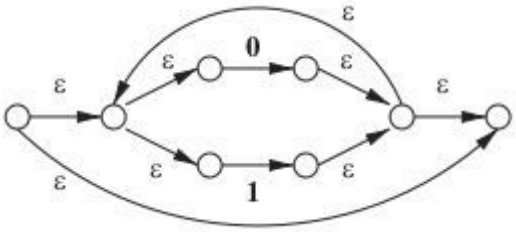
Ejemplo

Deseamos convertir la expresión regular (0+ 1)∗1(0 + 1) en un AFN-ε.

El **primer paso** consiste en construir un autómata para 0 + 1.



Luego Construiremos (0+1)* a partir de (0+1), Colocando una trición vacia en el nodo final de 0+1 devolviendome al inicio para las veces que se repita el (0+1), luego colocaremos un nodo anterior al nodo inicial de (0+1) y añadiremos una transición vacia entre este nodo y un nodo adelante del nodo final para cuando (0+1)^0



Finalmente colocaremos una transición 1 que conecte (0+1)* con (0+1) dando como resultado