

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 12

Cuanenemi Cuanalo Mario Alberto

3.1 Expresiones regulares Ahora vamos a desviar nuestra atención de las descripciones tipo máquina de los lenguajes, autómatas finitos deterministas y no deterministas, a un tipo de expresión algebraica: la “expresión regular”. Comprobaremos que las expresiones regulares pueden definir de forma exacta los mismos lenguajes que describen los distintos tipos de autómatas: los lenguajes regulares. Sin embargo, las expresiones regulares ofrecen algo que los autómatas no proporcionan: una forma declarativa para expresar las cadenas que deseamos aceptar. Por tanto, las expresiones regulares sirven como lenguaje de entrada de muchos sistemas que procesan cadenas. Algunos ejemplos son los siguientes: 1. Comandos de búsqueda tales como el comando grep de UNIX o comandos equivalentes para localizar cadenas en los exploradores web o en los sistemas de formateo de texto. Estos sistemas emplean una notación de tipo expresión regular para describir los patrones que el usuario desea localizar en un archivo. Los distintos sistemas de búsqueda convierten la expresión regular bien en un AFD o en un AFN y simulan dicho autómata sobre el archivo en que se va a realizar la búsqueda. 2. Generadores de analizadores léxicos, como Lex o Flex. Recuerde que un analizador léxico es el componente de un compilador que divide el programa fuente en unidades lógicas o sintácticas formadas por uno o más caracteres que tienen un significado. Entre las unidades lógicas o sintácticas se incluyen las palabras clave (por ejemplo, while), identificadores (por ejemplo, cualquier letra seguida de cero o más letras y/o dígitos) y signos como + o <=. Un generador de analizadores léxicos acepta descripciones de las formas de las unidades lógicas, que son principalmente expresiones regulares, y produce un AFD que reconoce qué unidad lógica aparece a continuación en la entrada.

3.1.1 Operadores de las expresiones regulares

Las expresiones regulares denotan lenguajes. Por ejemplo, la expresión regular $0^* + 10^*$ define el lenguaje que consta de todas las cadenas que comienzan con un 0 seguido de cualquier número de 1s o que comienzan por un 1 seguido de cualquier número de 0s. No esperamos que el lector sepa ya cómo interpretar las expresiones regulares, por lo que por el momento tendrá que aceptar como un acto de fe nuestra afirmación acerca del lenguaje de esta expresión. Enseguida definiremos todos los símbolos empleados en esta expresión, de modo que pueda ver por qué nuestra interpretación de esta expresión regular es la correcta. Antes de describir la notación de las expresiones regulares, tenemos que estudiar las tres operaciones sobre los lenguajes que representan los operadores de las expresiones regulares. Estas operaciones son: 1. La unión de dos lenguajes L y M , designada como $L \cup M$, es el conjunto de cadenas que pertenecen a L , a M o a ambos. Por ejemplo, si $L = \{001, 10, 111\}$ y $M = \{\epsilon, 001\}$, entonces $L \cup M = \{\epsilon, 001, 10, 111\}$. 2. La concatenación de los lenguajes L y M es el conjunto de cadenas que se puede formar tomando cualquier cadena de L y concatenándola con cualquier cadena de M . Recuerde la Sección 1.5.2, donde definimos la concatenación de una pareja de cadenas; el resultado de la concatenación es una

cadena seguida de la otra. Para designar la concatenación de lenguajes se emplea el punto o ningún operador en absoluto, aunque el operador de concatenación frecuentemente se llama “punto”. Por ejemplo, si $L = \{001, 10, 111\}$ y $M = \{\epsilon, 001\}$, entonces $L.M$, o simplemente LM , es $\{001, 10, 111, 001001, 10001, 111001\}$. Las tres primeras cadenas de LM son las cadenas de L concatenadas con ϵ . Puesto que ϵ es el elemento identidad para la concatenación, las cadenas resultantes son las mismas cadenas de L . Sin embargo, las tres últimas cadenas de LM se forman tomando cada una de las cadenas de L y concatenándolas con la segunda cadena de M , que es 001 . Por ejemplo, la concatenación de la cadena 10 de L con la cadena 001 de M nos proporciona la cadena 10001 para LM .

3. La clausura (o asterisco, o clausura de Kleene)¹ de un lenguaje L se designa mediante L^* y representa el conjunto de cadenas que se pueden formar tomando cualquier número de cadenas de L , posiblemente con repeticiones (es decir, la misma cadena se puede seleccionar más de una vez) y concatenando todas ellas. Por ejemplo, si $L = \{0, 1\}$, entonces L^* es igual a todas las cadenas de 0s y 1s. Si $L = \{0, 11\}$, entonces L^* constará de aquellas cadenas de 0s y 1s tales que los 1s aparezcan por parejas, como por ejemplo 011 , 11110 y ϵ , pero no 01011 ni 101 . Más formalmente, L^* es la unión infinita $\bigcup_{i \geq 0} L^i$, donde $L^0 = \{\epsilon\}$, $L^1 = L$ y L^i , para $i > 1$ es $LL \cdots L$ (la concatenación de i copias de L).

EJEMPLO 3.1 Dado que la idea de clausura de un lenguaje es algo engañosa, vamos a estudiar algunos ejemplos. Primero, sea $L = \{0, 11\}$. $L^0 = \{\epsilon\}$, independientemente de qué lenguaje sea L ; la potencia 0 representa la selección de 1. El término “clausura de Kleene” hace referencia a S. C. Kleene, quien ideó la notación de las expresiones regulares y este operador.

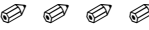
Capítulo 3 Lenguajes y expresiones regulares 73

Uso del operador Veamos en primer lugar el operador presentado en la Sección 1.5.2, donde lo aplicamos a un alfabeto, por ejemplo, Σ^* . Dicho operador sirve para formar todas las cadenas cuyos símbolos han sido seleccionados de un alfabeto Σ . El operador de clausura es prácticamente igual, aunque existen algunas diferencias sutiles de tipos. Supongamos que L es el lenguaje que contiene cadenas de longitud 1 y que para cada símbolo a perteneciente a Σ existe una cadena a en L . Luego aunque L y Σ “parezcan” lo mismo, son de tipos diferentes; L es un conjunto de cadenas y Σ es un conjunto de símbolos. Por otro lado, L^* designa el mismo lenguaje que Σ^* .

0 cadenas de L . $L^1 = L$, representa la elección de una cadena de L . Por tanto, los dos primeros términos de la expansión de L^* nos da $\{\epsilon, 0, 11\}$. A continuación considere L^2 . Seleccionamos dos cadenas de L , permitiendo repeticiones, de modo que tenemos cuatro posibilidades. Estas cuatro selecciones nos dan $L^2 = \{00, 011, 110, 1111\}$. De forma similar, L^3 es el conjunto de cadenas que se pueden formar eligiendo tres posibilidades de las dos cadenas de L , lo que nos proporciona $\{000, 0011, 0110, 1100, 01111, 11011, 11110, 111111\}$. Para calcular L^* , tenemos que calcular L^i para cada i y hallar la unión de todos estos lenguajes. L^i tiene 2^i miembros. Aunque cada L^i es finito, la unión de un número infinito de términos L^i generalmente es un lenguaje infinito, como en el caso de nuestro ejemplo. Sea ahora L el conjunto de todas las cadenas de 0s. Observe que L es infinito, a diferencia de en el ejemplo anterior en el que era un lenguaje finito. Sin embargo, no

es difícil descubrir que es L^* . $L_0 = \{\epsilon\}$, como siempre. $L_1 = L$. L_2 es el conjunto de cadenas que se pueden formar tomando una cadena de 0s y concatenarla con otra cadena de 0s. El resultado sigue siendo una cadena de 0s. De hecho, toda cadena de 0s se puede escribir como la concatenación de dos cadenas de 0s (no olvide que ϵ es una “cadena de 0”; esta cadena siempre puede ser una de las dos cadenas que concatenemos). Por tanto, $L_2 = L$. Del mismo modo, $L_3 = L$, etc. Luego la unión infinita $L^* = L_0 \cup L_1 \cup L_2 \cup \dots$ es L en el caso particular de que el lenguaje L sea el conjunto de todas las cadenas de 0s. Veamos un último ejemplo. $/0^* = \{\epsilon\}$. Observe que $/00 = \{\epsilon\}$, mientras que $/0^i$, para todo $i \geq 1$, es el conjunto vacío, ya que no podemos seleccionar ninguna cadena en el conjunto vacío. De hecho, $/0$ es uno de los dos lenguajes cuya clausura no es infinita. *

3.1.2 Construcción de expresiones regulares

Todos los tipos de álgebras se inician con las expresiones elementales, que normalmente son constantes y/o variables. Las álgebras nos permiten construir más expresiones aplicando un cierto conjunto de operadores a las expresiones elementales y a las expresiones previamente construidas. Normalmente, también se necesitan algunos métodos que permitan agrupar operadores con sus operandos, tales como los paréntesis. Por ejemplo, la familiar álgebra de la aritmética se inicia con constantes, como los números enteros y reales, más las variables y se construyen expresiones más complejas utilizando operadores aritméticos como $+$ y \times . El álgebra de las expresiones regulares sigue también este patrón, utilizando constantes y variables que representan lenguajes y operadores para las tres operaciones mencionadas en la Sección 3.1.1 (la unión, el punto y el asterisco). Podemos describir las expresiones regulares recursivamente del siguiente modo. En esta  74 Introducción a la teoría de autómatas, lenguajes y computación definición, no sólo describimos lo que son las expresiones regulares válidas, sino que para cada expresión regular E , describimos el lenguaje que representa, al que denominaremos $L(E)$. BASE. El caso básico consta de tres partes: 1. Las constantes ϵ y $/0$ son expresiones regulares, que representan a los lenguajes $\{\epsilon\}$ y $/0$, respectivamente. Es decir, $L(\epsilon) = \{\epsilon\}$ y $L(/0) = \emptyset$. / 2. Si a es cualquier símbolo, entonces a es una expresión regular. Esta expresión representa el lenguaje $\{a\}$. Es decir, $L(a) = \{a\}$. Observe que utilizamos la fuente en negrita para indicar la expresión correspondiente a un símbolo. La correspondencia, por ejemplo, que a hace referencia a a , es obvia. 3. Una variable, normalmente escrita en mayúsculas e itálicas, como L , representa cualquier lenguaje. PASO INDUCTIVO. Existen cuatro partes en el paso de inducción, una para cada uno de los tres operadores y otra para la introducción de paréntesis. 1. Si E y F son expresiones regulares, entonces $E + F$ es una expresión regular que representa la unión de $L(E)$ y $L(F)$. Es decir, $L(E + F) = L(E) \cup L(F)$. 2. Si E y F son expresiones regulares, entonces EF es una expresión regular que representa la concatenación de $L(E)$ y $L(F)$. Es decir, $L(EF) = L(E)L(F)$. Observe que el punto puede utilizarse opcionalmente para explicitar el operador de concatenación, bien como una operación sobre lenguajes o como el operador en una expresión regular. Por ejemplo, 0.1 es una expresión regular que significa lo mismo que 01 y que representa el lenguaje $\{01\}$. Sin embargo, nosotros vamos a evitar el uso del punto en la concatenación de expresiones regulares. 2 3.

Si E es una expresión regular, entonces E^* es una expresión regular, que representa la clausura de $L(E)$. Es decir, $L(E^*) = L(E)^*$. 4. Si E es una expresión regular, entonces (E) , una E encerrada entre paréntesis, es también

Fermin Cruz Erik

AUTÓMATAS FINITOS Y EXPRESIONES REGULARES

Un autómata finito como tal se refiere a un modelo computacional que realiza cálculos en forma automática sobre una entrada para producir una salida.

Este modelo está conformado por un alfabeto, un conjunto de estados finitos, una función de transición, un estado inicial y un conjunto de estados finales.

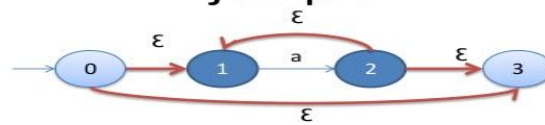
Su funcionamiento se basa en una función de transición, que recibe a partir de un estado inicial una cadena de caracteres pertenecientes al alfabeto (la entrada), y que va leyendo dicha cadena a medida que el autómata se desplaza de un estado a otro, para finalmente detenerse en un estado final o de aceptación, que representa la salida.

EXPRESIONES REGULARES EN AUTÓMATAS FINITOS

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.

Ejemplo



Cerradura ϵ de un estado: El estado mismo y los estados que conduce una transición ϵ

$$\bar{0} = \{0, 1, 3\}$$

$$\bar{1} = \{1\}$$

$$\bar{2} = \{1, 2, 3\}$$

$$\bar{3} = \{3\}$$

$$\bar{0} = \{0, 1, 3\}$$

Gutierrez Arellano Rafael

Autómatas finitos y expresiones regulares

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

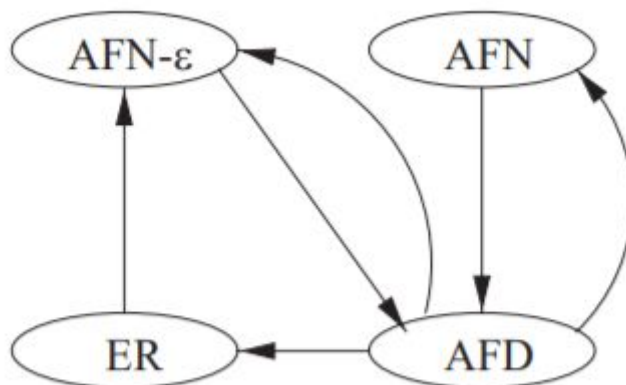
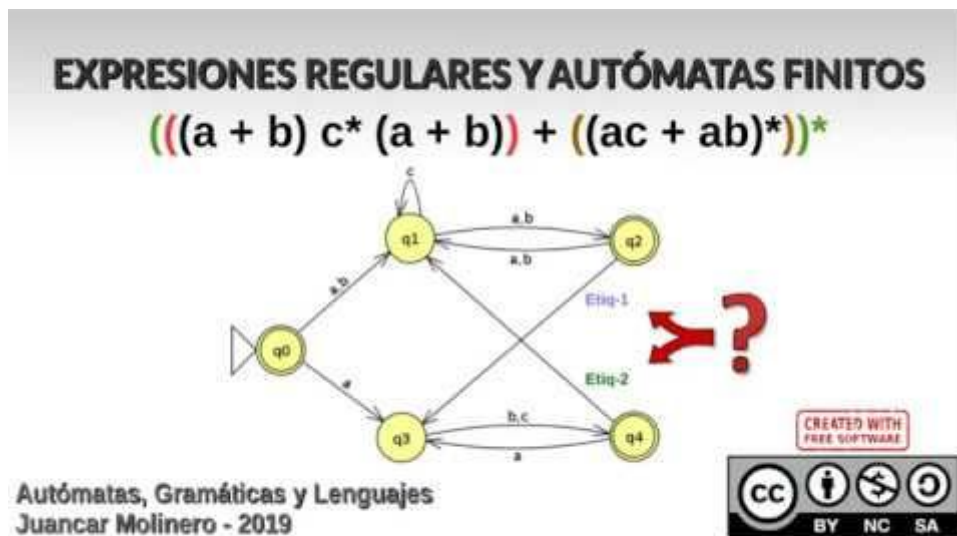
Ofrecen algo que los autómatas no: Manera declarativa de expresar las cadenas que queremos aceptar

Ejemplos de sus usos

- Comandos de búsqueda, e.g., grep de UNIX
- Sistemas de formateo de texto: Usan notación de tipo expresión regular
- Convierte la expresión regular a un DFA o un NFA y simula el autómata en el archivo de búsqueda
- Generadores de analizadores-lexicos. Como Lex o Flex.
- Los analizadores léxicos son parte de un compilador. Dividen el programa fuente en unidades lógicas (tokens), como while, números, signos ('+', '-', '<', etc.)
- Produce un DFA que reconoce el token

Aunque las expresiones regulares describen los lenguajes de manera completamente diferente a como lo hacen los autómatas finitos, ambas notaciones representan exactamente el mismo conjunto de lenguajes, que hemos denominado “lenguajes regulares”. Ya hemos demostrado que los autómatas finitos deterministas

y los dos tipos de autómatas finitos no deterministas (con y sin transiciones ϵ) aceptan la misma clase de lenguajes. Para demostrar que las expresiones regulares definen la misma clase, tenemos que probar que: 1. Todo lenguaje definido mediante uno de estos autómatas también se define mediante una expresión regular. Para demostrar esto, podemos suponer que el lenguaje es aceptado por algún AFD. 2. Todo lenguaje definido por una expresión regular puede definirse mediante uno de estos autómatas. Para esta parte de la demostración, lo más sencillo es probar que existe un AFN con transiciones- ϵ que acepta el mismo lenguaje.



Vamos a convertir el AFD de la Figura 3.4 en una expresión regular. Este AFD acepta todas las cadenas que tienen al menos un 0. Para comprender por qué, observe que el autómata va desde el estado inicial 1 al estado de aceptación 2 tan pronto como recibe una entrada 0. Después, el autómata permanece en el estado 2 para todas las secuencias de entrada.

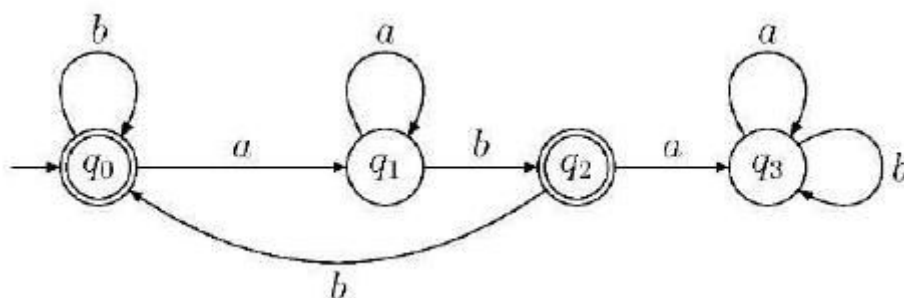
A continuación se especifican las expresiones básicas de la construcción del Teorema 3.4.

$R_{11}^{(0)}$	$\varepsilon + 1$
$R_{12}^{(0)}$	0
$R_{21}^{(0)}$	0
$R_{22}^{(0)}$	$(\varepsilon + 0 + 1)$

Por ejemplo, $R^{(0)}_{11}$ tiene el término ε porque los estados inicial y final son el mismo, el estado 1. Contiene el término 1 porque existe un arco desde el estado 1 al estado 1 sobre la entrada 1. Otro ejemplo, $R^{(0)}_{12}$ es 0 porque hay un arco etiquetado como 0 desde el estado 1 hasta el estado 2. No existe el término ε porque los estados inicial y final son diferentes. Como tercer ejemplo, tenemos $R^{(0)}_{21} = 0$, porque no existe un arco desde el estado 2 al estado 1. Ahora tenemos que abordar la parte inductiva, construyendo expresiones más complejas que la primera teniendo en cuenta los caminos que pasan por el estado 1, luego los caminos que pasan por los estados 1 y 2, es decir, cualquier camino. La regla para calcular las expresiones $R^{(1)}_{ij}$ es un caso particular de la regla general dada en la parte inductiva del Teorema 3.4:

	Por sustitución directa	Simplificada
$R_{11}^{(1)}$	$\varepsilon + 1 + (\varepsilon + 1)(\varepsilon + 1)^*(\varepsilon + 1)$	1^*
$R_{12}^{(1)}$	$0 + (\varepsilon + 1)(\varepsilon + 1)^*0$	1^*0
$R_{21}^{(1)}$	$0 + 0(\varepsilon + 1)^*(\varepsilon + 1)$	0
$R_{22}^{(1)}$	$\varepsilon + 0 + 1 + 0(\varepsilon + 1)^*0$	$\varepsilon + 0 + 1$

Figura 3.5. Expresiones regulares para caminos que sólo pueden pasar a través del estado 1.



Autómatas finitos y expresiones regulares

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.

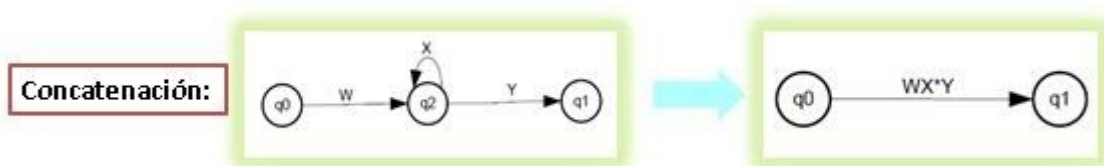
Conversión de un AFD en una expresión regular mediante la eliminación de estados

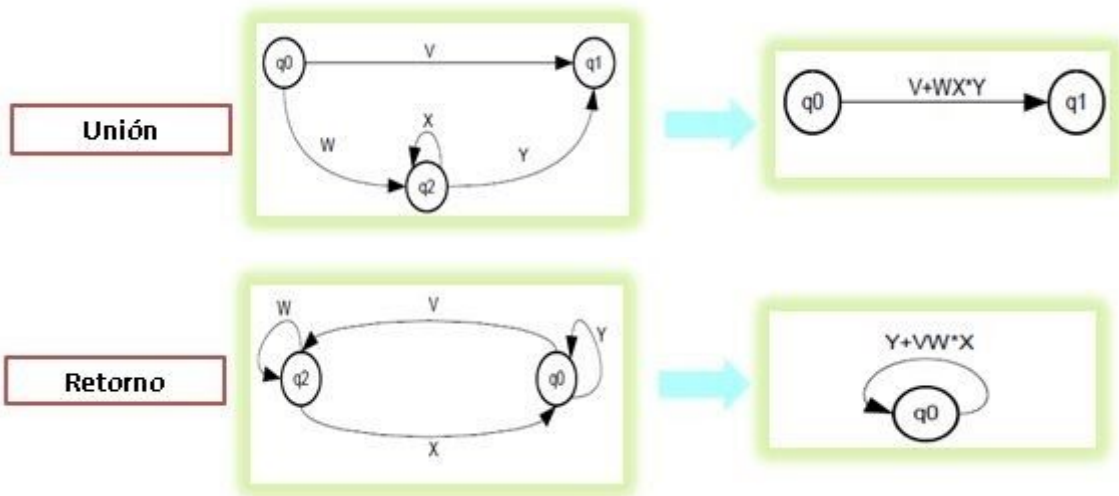
En este texto vamos a ver uno de los métodos que se usan para transformar autómatas finitos deterministas en expresiones regulares, el método de eliminación de estados.

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

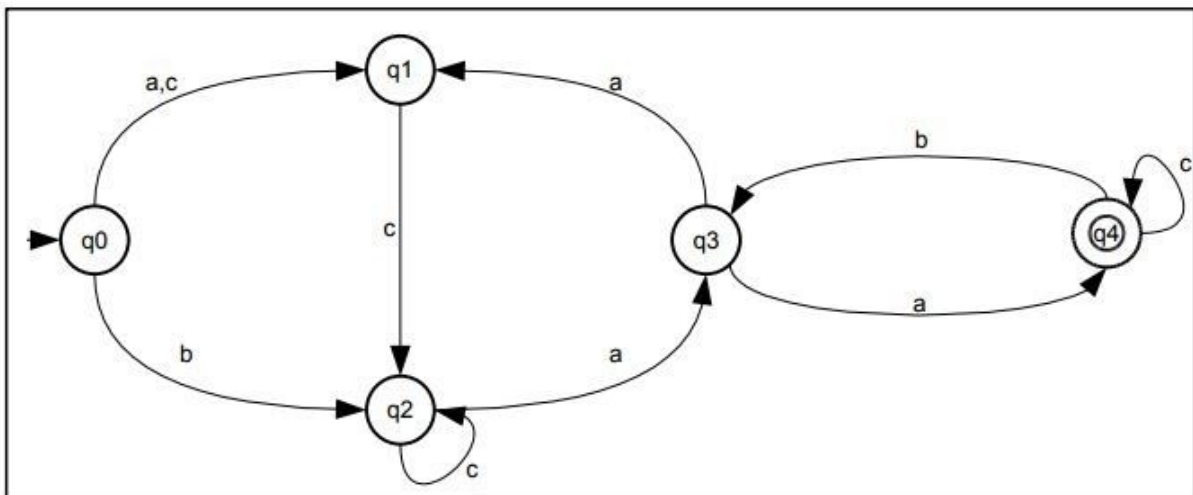
directas que ahora se realizan con el ingreso de expresiones regulares, en vez de con símbolos.

Los casos bases son los siguientes:



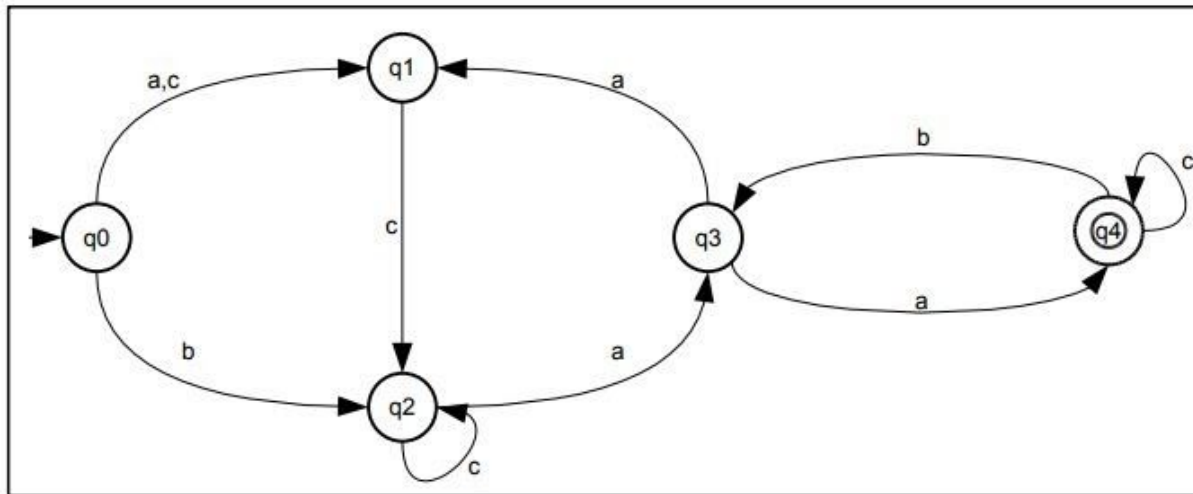


Ejemplo



1. Transformar todas aquellas transiciones que contemplan más de un símbolo en expresiones regulares del tipo "R+P".

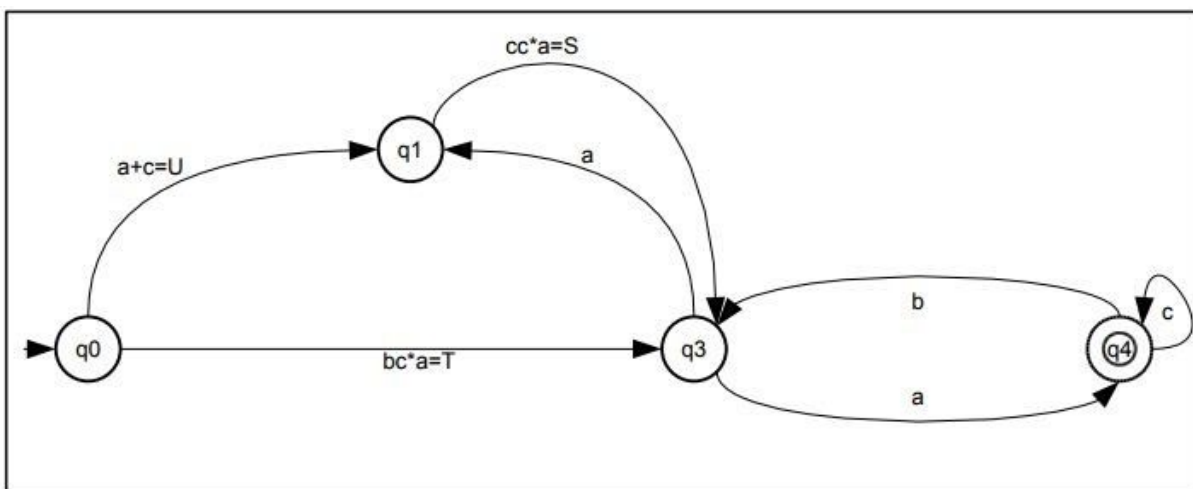
Por lo tanto, la transición q_0 a q_1 , queda como "a+c". Todas las demás permanecen iguales, por ser símbolos.



Se procede a eliminar el estado q2. Verificar todos los caminos que atraviesan q2:

- q0 puede llegar a q3 pasando por q2.
- q1 puede llegar a q3 pasando por q2.

Al eliminar el estado q2 las transiciones del AFD quedan:



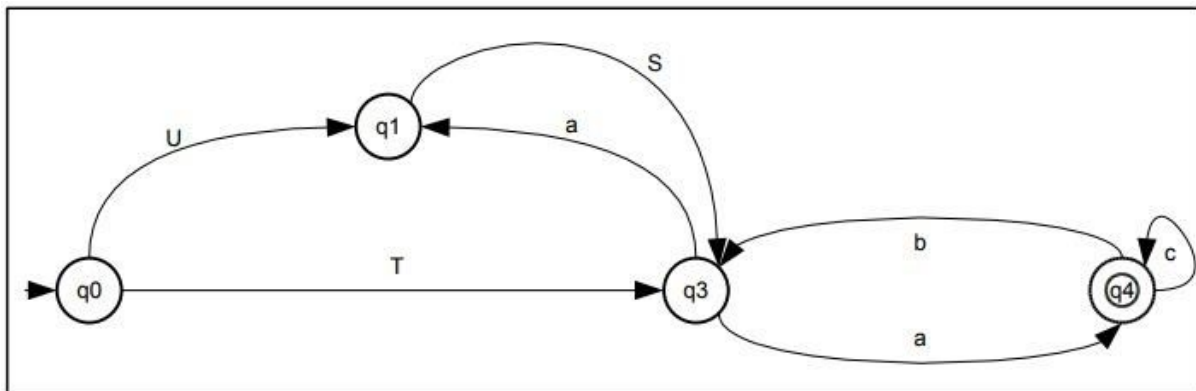
- 1) Para llegar a q3, q0 primero debe pasar por q2 usando el símbolo "b", luego puede pasar repetidas veces por q2 (o ninguna vez) usando cero, uno o varios símbolos "c" y, por último, pasa de q2 a q3 con el símbolo "a". Esto queda expresado con la expresión regular $T = bc^*a$

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:

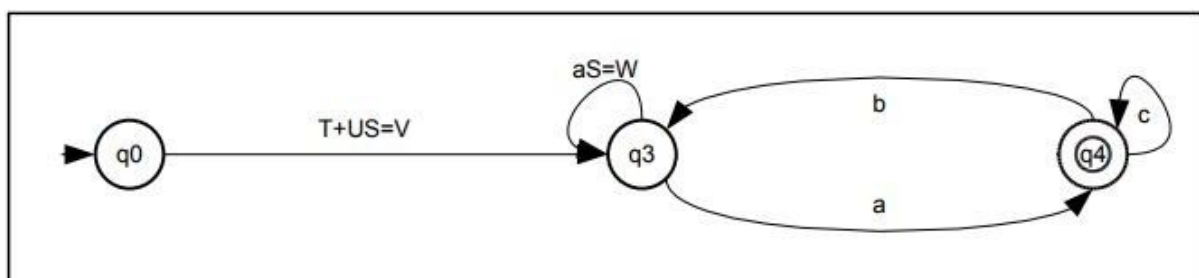
1) De q_0 a q_3 , que puede hacerlo a través de q_1 o directamente, usando T. La expresión regular que va de q_0 a q_3 a través de q_1 es US y la expresión que va directamente de q_0 a q_3 es T. Como puede ser una o la

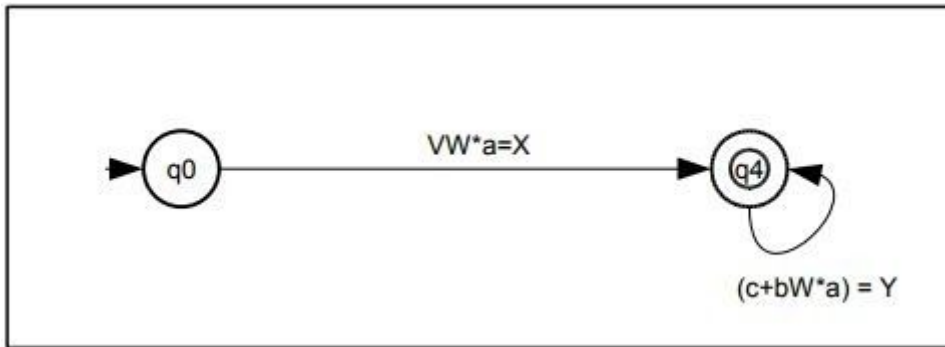
otra, como

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:

$W = aS$



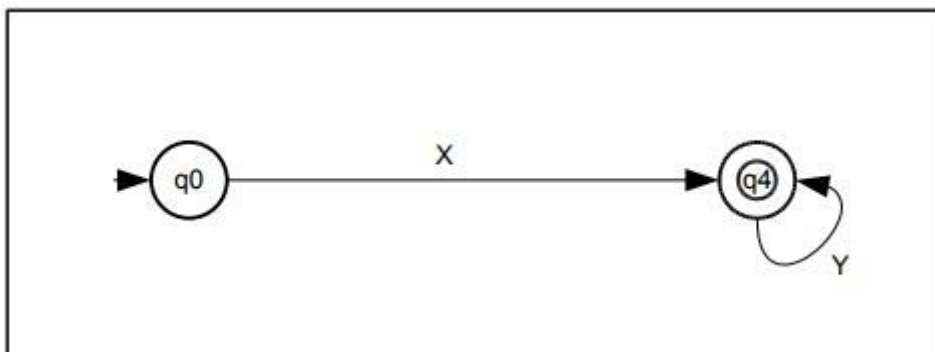


Eliminar q_3 . Las transiciones afectadas son:

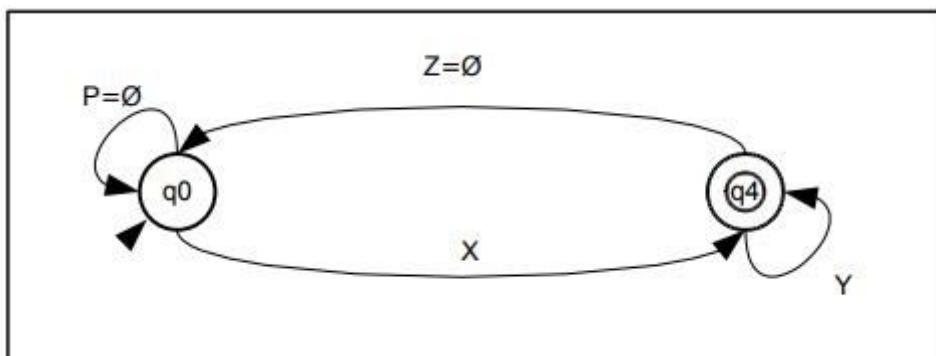
1) De q_0 a q_4 , pasando por q_3 : Primero va de q_0 a q_3 usando la expresión V . Luego de q_3 a q_3 repetidas veces, usando W . Por último, pasa de q_3 a q_4 usando una "a". La expresión regular queda : $X = V.W^*.a$

2) De q_4 a q_4 , pasando por q_3 o directamente por q_4 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:

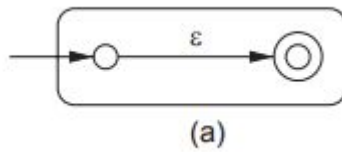


Completando las transiciones vacías (de q_4 a q_1 y de q_1 a q_1) con las expresiones regulares vacías Z y P

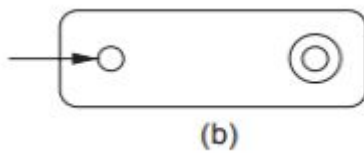
De esta forma, se puede generar la expresión regular final a partir de la fórmula $(P + XY^*Z)^*XY^*$

$$\text{Expresión Final} = (P + XY^*Z)^*XY^* = (\emptyset^* + XY^*\emptyset)^*XY^* = (\epsilon + \emptyset)^*XY^* = \epsilon^*XY^* = XY^*$$

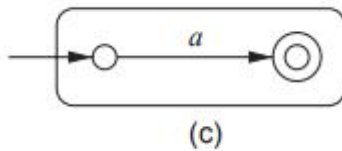
Conversión de una expresión regular a un autómata



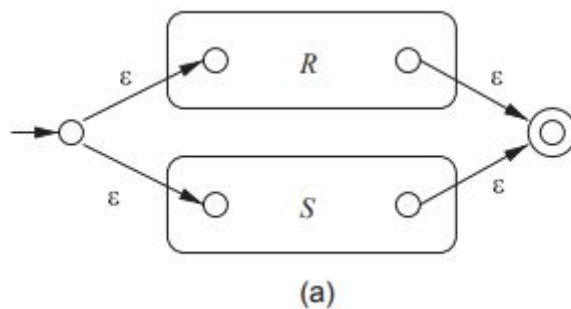
En la parte (a) vemos cómo se maneja la expresión ϵ . Puede verse fácilmente que el lenguaje del autómata es $\{\epsilon\}$, ya que el único camino desde el estado inicial a un estado de aceptación está etiquetado con ϵ .



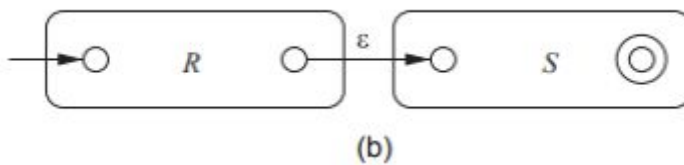
La parte (b) muestra la construcción de \emptyset . Claramente, no existen caminos desde el estado inicial al de aceptación, por lo que \emptyset es el lenguaje de este autómata.



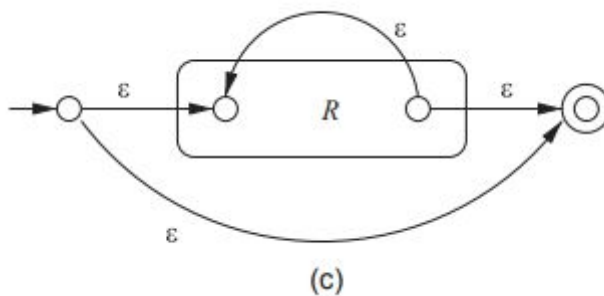
La parte (c) proporciona el autómata que reconoce una expresión regular a . Evidentemente, el lenguaje de este autómata consta de una cadena a , que es también $L(a)$.



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 $L(R) \cup L(S)$.



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 $L(R)L(S)$.

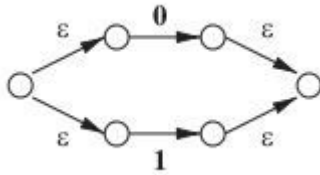


La expresión es de la forma R^* para una expresión R más pequeña. Dicho camino acepta ϵ , que pertenece a $L(R^*)$ sin importar qué expresión sea R .

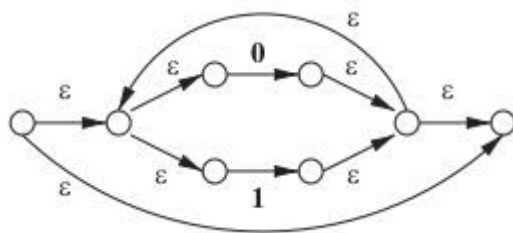
Ejemplo

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 transición vacía 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 vacía 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

