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

Expresiones Regulares

- 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). Tokens como while, números, signos ($+, -, <$, etc.)
  - Produce un DFA que reconoce el token

Expresiones Regulares

Si E es una expresión regular, entonces $L(E)$ denota el lenguaje que define E. Las expresiones se construyen de la manera siguiente:

1. Las constantes ϵ y $\emptyset$ son expresiones regulares que representan a los lenguaje $L(\epsilon) = \{\epsilon\}$ y $L(\emptyset) = \emptyset$ respectivamente
2. Si a es un símbolo, entonces es una expresión regular que representan al lenguaje: $L(a) = \{a\}$

Operandos

1. Si E y F son expresiones regulares, entonces $E + F$ también lo es denotando la unión de $L(E)$ y $L(F)$. $L(E + F) = L(E) \cup L(F)$.
2. Si E y F son expresiones regulares, entonces EF también lo es denotando la concatenación de $L(E)$ y $L(F)$. $L(EF) = L(E)L(F)$.
3. Si E es una expresión regular, entonces E^* también lo es y denota la cerradura de $L(E)$. Osea $L(E^*) = (L(E))^*$
4. Si E es una expresión regular, entonces (E) también lo es. Formalmente: $L((E)) = L(E)$.

Precedencia

1. El asterisco de la cerradura tiene la mayor precedencia
2. Concatenación sigue en precedencia a la cerradura, el operador “dot”. Concatenación es asociativa y se sugiere agrupar desde la izquierda (i.e. 012 se agrupa (01)2).
3. La unión (operador $+$) tiene la siguiente precedencia, también es asociativa.
4. Los paréntesis pueden ser utilizados para alterar el agrupamiento

Ejemplos

- $L(001) = 001$.
- $L(0 + 10^*) = \{0, 1, 10, 100, 1000, \ldots\}$.
- $L((0(0 + 1))^*)$ = el conjunto de cadenas de 0's y 1's, de longitud par, de tal manera que cada posición impar tenga un 0.
- Expresión regular de cadenas que alterna 0's y 1's:
  1. $(01)^* + (10)^* + 0(10)^* + 1(01)^*$ (opción 1)
  2. $(\epsilon + 1)(01)^*(\epsilon + 0)$ (opción 2)

Ejemplos

1. Encuentra la expresión regular para el conjunto de cadenas sobre el alfabeto $\{a, b, c\}$ que tiene al menos una a y al menos una b
2. Encuentra la expresión regular para el conjunto de cadenas de 0's y 1's tal que cada par de 0's adyacentes aparece antes de cualquier par de 1's adyacentes

Soluciones

1. $c^*a(a+c)^*b(a+b+c)^* + c^*b(b+c)^*a(a+b+c)^*$
   Osea, cuando la primera *a* esta antes que la primera *b* o cuando la primera *b* está antes de la primera *a*
2. $(10+0)^*(\epsilon+1)(01+1)^*(\epsilon+1)$
   $(10+0)^*(\epsilon+1)$ es el conjunto de cadenas que no tienen dos 1's adyacentes. La segunda parte es el conjunto de cadenas que no tienen dos 0's adyacentes. De hecho $\epsilon+1$ lo podríamos eliminar porque se puede obtener el 1 de lo que sigue, por lo que podemos simplificarlo a: $(10+0)^*(01+1)^*(\epsilon+1)$

Equivalencia de Lenguajes de FA y Lenguajes RE

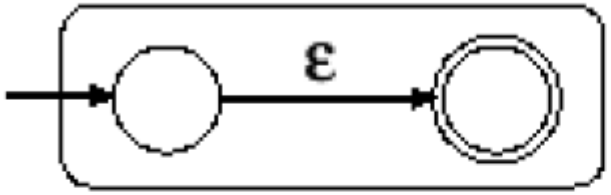
- Se mostrará que un NFA con transiciones-ϵ puede aceptar el lenguaje de una RE.
- Después, se mostrará que un *RE* puede describir el lenguaje de un DFA (la misma construcción funciona para un NFA).
- Los lenguajes aceptados por DFA, NFA, ϵ-NFA, RE son llamados lenguajes regulares.

Convirtiendo una RE a un Autómata

Aplicación del teorema de Kleene

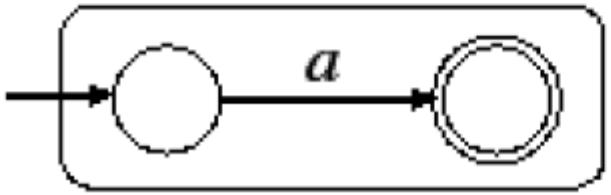
1. Exactamente un estado de aceptación
2. Sin arcos que lleguen al estado inicial
3. Sin arcos que salgan del estado de aceptación

Base: cumpliendo las condiciones 1, 2, y 3.



El lenguaje es $\in$

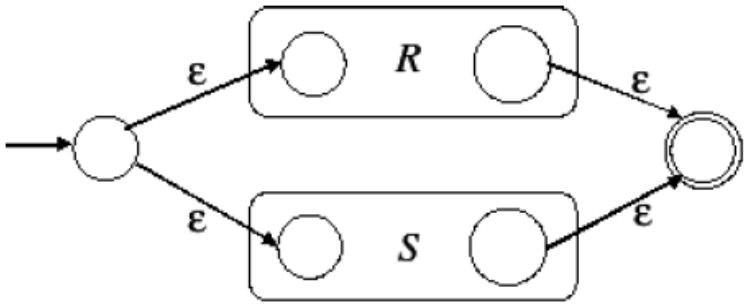
El lenguaje es ϕ



El lenguaje es la RE *a*, y sólo contiene la cadena *a*

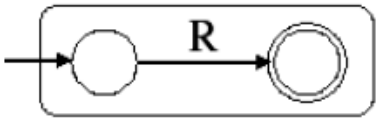
Inducción:

a)



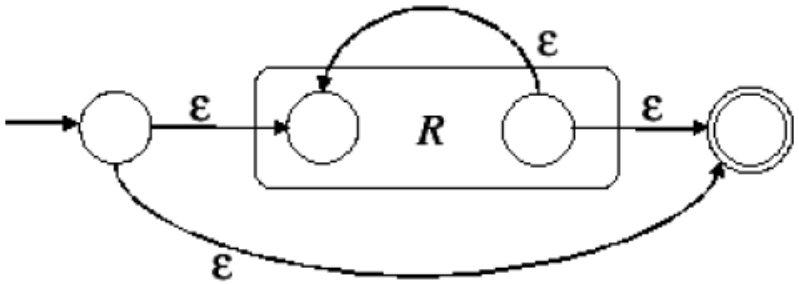
Lenguaje: $L(R) \cup L(S)$

d)



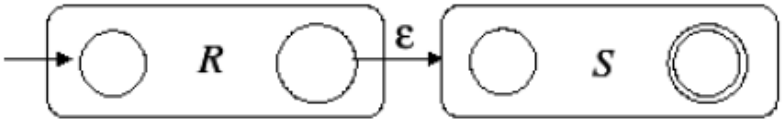
El lenguaje es R ó (R)

c)



El lenguaje es la R^*

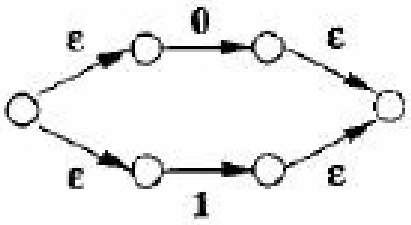
b)



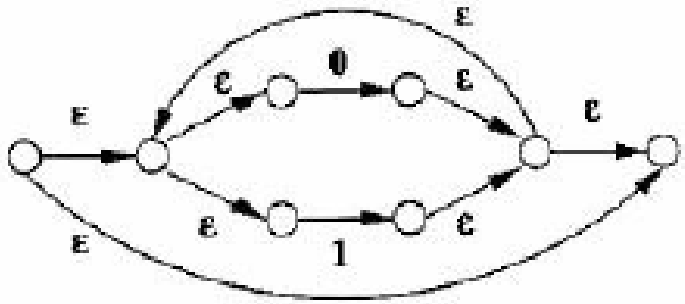
Lenguaje: $L(R)L(S)$

Ejemplo

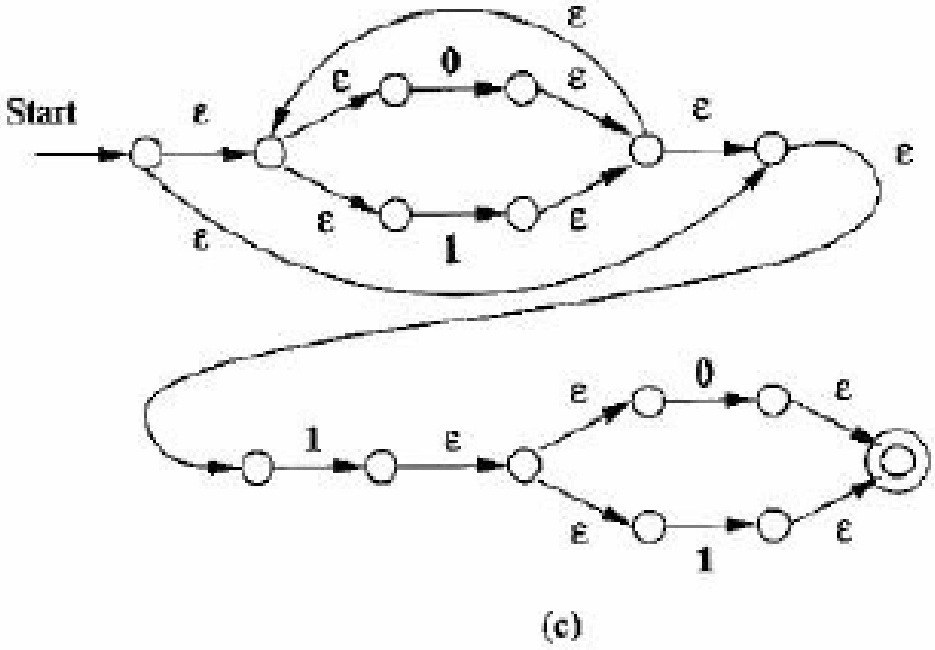
Convertir la RE $(0 + 1)^*1(0 + 1)$ a un ϵ-NFA.



(a)



(b)



(c)

Método para obtener la Expresión regular que denota a un AF dado.

Cada ecuación de un sistema de ecuaciones lineales en expresiones regulares tiene la siguiente forma general

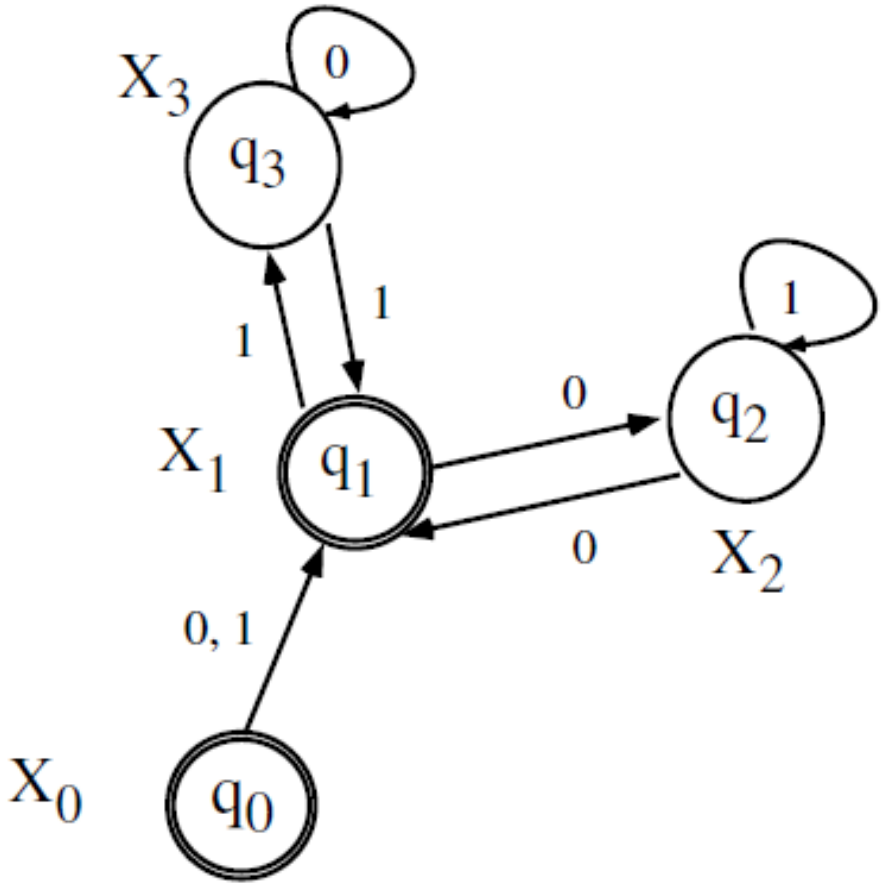
$$X = rX + s$$

en la que r y s son expresiones regulares sobre un alfabeto Σ.

Solucion: $X = r^* s$.

1. Se asocia una variable a cada estado: $\forall q_i \in Q$, se le asocia X_i.
2. Las ecuaciones se construyen en función de las transiciones: si $q_j \in f(q_i, a)$, entonces en la ecuación de la variable X_i aparece el término aX_j en su parte derecha: $X_i = ... + aX_j + ...$
3. Además, se asocia el término λ a los estados finales: si $q_i \in F$, entonces en la ecuación de la variable X_i aparece λ en su parte derecha: $X_i = ... + \lambda + ...$

El lenguaje reconocido por el AF es la *expresión regular de la variable asociada a su estado inicial*.



$$X_0 = 0X_1 + 1X_1 + \lambda = (0+1)X_1 + \lambda$$
$$X_1 = 0X_2 + 1X_3 + \lambda$$
$$X_2 = 1X_2 + 0X_1$$
$$X_3 = 0X_3 + 1X_1$$

$$X_0 = 0X_1 + 1X_1 + \lambda = (0+1)X_1 + \lambda$$
$$X_1 = 0X_2 + 1X_3 + \lambda$$
$$X_2 = 1X_2 + 0X_1$$
$$X_3 = 0X_3 + 1X_1$$

la solución de la ecuación general, $X = rX + s$ es $X = r^*s$.

$$X_3 = \underbrace{0}_{r} X_3 + \underbrace{1X_1}_{s}. \quad \Rightarrow \quad X_3 = 0^*1X_1.$$

$$X_2 = \underbrace{1}_{r} X_2 + \underbrace{0X_1}_{s}, \quad \Rightarrow \quad X_2 = 1^*0X_1$$

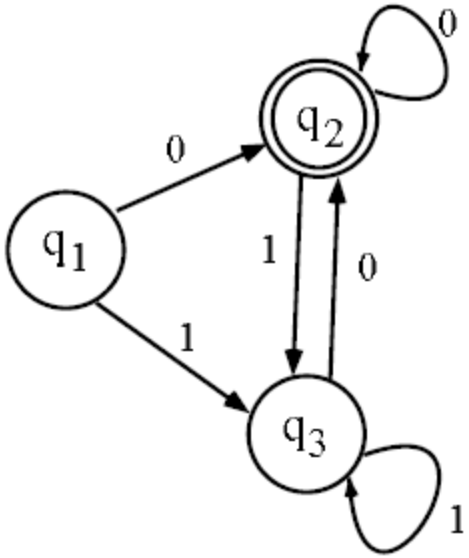
Se tienen X_2 y X_3 en función de X_1. Al substituir en la ecuación de X_1 se obtiene

$$X_1 = 0X_2 + 1X_3 + \lambda = 01^*0X_1 + 10^*1X_1 + \lambda = (01^*0 + 10^*1)X_1 + \lambda$$

$$X_1 = \underbrace{(01^*0 + 10^*1)}_{r} X_1 + \underbrace{\lambda}_{s}, \quad \Rightarrow \quad X_1 = (01^*0 + 10^*1)^*\lambda = (01^*0 + 10^*1)^*$$

al substituir en la ecuación de X_0 se obtiene finalmente,

$$X_0 = (0+1)X_1 + \lambda = (0+1)(01^*0 + 10^*1)^* + \lambda$$



$$X_1 = 0X_2 + 1X_3$$
$$X_2 = 0X_2 + 1X_3 + \lambda$$
$$X_3 = 1X_3 + 0X_2$$

$$X_3 = \underbrace{1}_{r} X_3 + \underbrace{0X_2}_{s}, \Rightarrow X_3 = 1^*0X_2.$$

$$X_2 = 0X_2 + 1X_3 + \lambda = 0X_2 + 11^*0X_2 + \lambda = \underbrace{0 + 11^*0}\, X_2 + \underbrace{\lambda}, \Rightarrow X_2 = (0 + 11^*0)^*.$$

$$X_1 = 0X_2 + 1X_3 = 0(0 + 11^*0)^* + 11^*0X_2 = 0(0 + 11^*0)^* + 11^*0(0 + 11^*0)^*$$

$$X_1 = (0 + 11^*0)(0 + 11^*0)^* = ((\lambda + 11^*)0)((\lambda + 11^*)0)^* = 1^*0(1^*0)^* = (1^*0)^*1^*0 = (1 + 0)^*0.$$

Lema de Pumping

Si L es un lenguaje regular, entonces existe una constante n tal que cada cadena $w \in L$, de longitud n o más, puede ser escrita como $w = xyz$, donde:

1. $y \neq \epsilon$
2. $|xy| \leq n$
3. Para toda $i \geq 0$, wy^iz también está en L. Notese que $y^i = y$ repetida i veces; $y^0 = \epsilon$.

Lo que dice es que si tenemos una cadena con una longitud mayor al número de estados del autómata, entonces una cadena no vacía y puede ser repetida (*pumped*) un número arbitrario de veces.

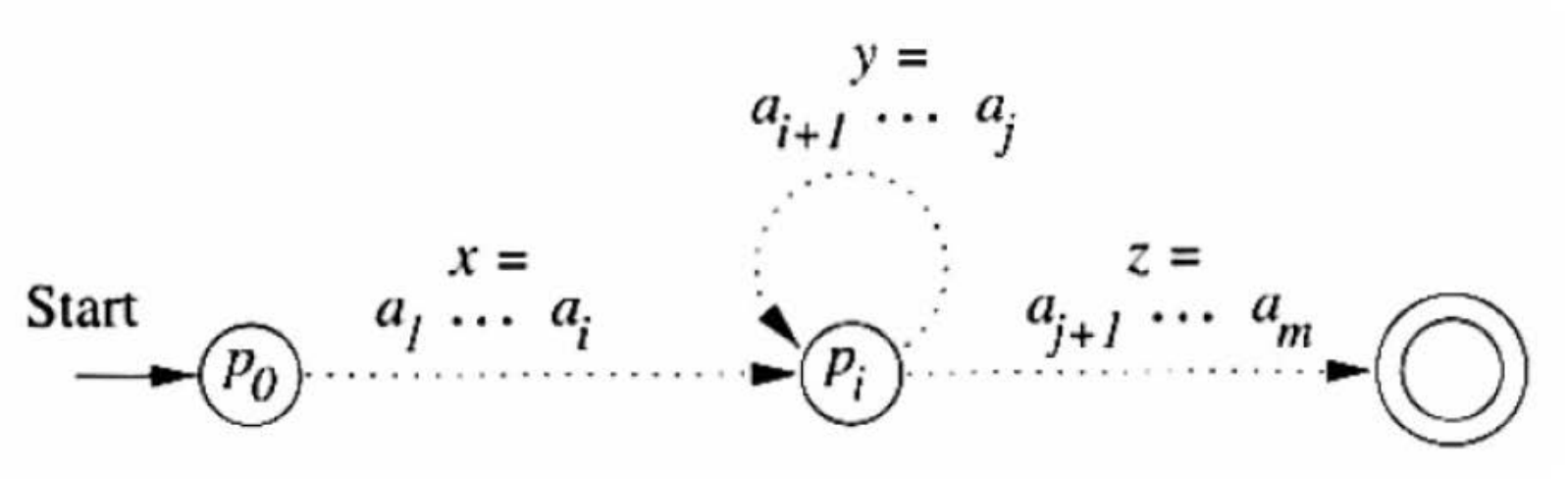
Prueba del Lema de Pumping

- Como se da por hecho que L es regular, debe existir un DFA A tal que $L = L(A)$. Si A tiene n estados; escogemos esta n para el lema de pumping.
- Sea w una cadena de longitud $\geq n$ en L, e.g., $w = a_1 a_2 \ldots a_m$, donde $m \geq n$.
- Sea q_i el estado en que A esta después de leer los primeros i símbolos de w.
- q_0 = estado de inicio, $q_1 = \delta(q_0, a_1), q2 = \delta'(q_0, a_1 a_2)$, etc.

Prueba del Lema de Pumping

- Como sólo hay n estados diferentes, dos de $q_0, q_1, \ldots, q_n$ deben ser los mismos; digamos $q_i = q_j$, donde $0 \leq i < j \leq n$.
- Sea $x = a_1 \ldots a_i$; $y = a_{i+1} \ldots a_j$; $z = a_{j+1} \ldots a_m$. Entonces, si repetimos el ciclo desde q_i a q_i con la etiqueta $a_{i+1} \ldots a_j$ cero o más veces, se puede probar que xy^iz es aceptado por A.

Lema de Pumping



Uso del Lema de Pumping

El Lema de Pumping se utiliza para mostrar que un lenguaje L no es regular.

- Se inicia supomiendo que L es regular.
- Luego, debe haber alguna n que sirva como la constante de PL (puede que no sepamos el valor de n)
- Escogemos una w que sabemos que está en L (normalmente w depende de n)
- Aplicando el PL, sabemos que w puede descomponerse en xyz, satisfaciendo las propiedades del PL (de nuevo, puede que no sepamos como descomponer w, así que utilizamos x, y, z como parámetros)
- Derivamos una contradicción escogiendo i (la cual puede depender de n, x, y, y/ó z) tal que $xy^i z$ no está en L.

Ejemplo

- Considere el lenguaje de cadenas con el mismo número de 0's y 1's. Por el pumping lemma, $w = xyz, |xy| \leq n, y \neq \epsilon$ y $xy^k z \in L$

$$w = \underbrace{000\ldots}_{x}\underbrace{\ldots 0}_{y}\underbrace{0111\ldots 11}_{z}$$

- En particular, $xz \in L$, pero xz tiene menos 0's que 1's

Ejemplo 2

- Supongamos que $L = 1^p : p$ es primo es regular
- Sea n el parámetro del pumping lemma y seleccionemos un primo $p \geq n + 2$.

$$w = \overbrace{\underbrace{111\ldots}_{x}\underbrace{\ldots 1}_{y}\underbrace{111\ldots 11}_{z}}^{p}$$

- Sea $|y| = m$ y $|xz| = p - m$. Ahora $xy^{p-m}z \in L$ $|xp^{p-m}z| = |xz| + (p-m)|y| = p - m + (p-m)m = (1+m)(p-m)$
- Que no es un número primo, a menos que uno de los factores sea 1
- Pero: $y \neq \epsilon \Rightarrow 1 + m > 1$ y $m = |y| \leq |xy| \leq n, p \geq n + 2 \Rightarrow p - m \geq n + 2 - n = 2$

Problemas

Problema 1: Considere el problema $0^n 1 0^n$ y demuestre que no es regular.

Problemas

Problema 2: Considere que el conjunto de cadenas de 0's cuya longitud es un cuadrado perfecto; formalmente $L = \{0^i | i \text{ es un cuadrado}\}$.

- Suponemos que L es regular. Entonces hay una n constante que satisface las condiciones del PL.
- Considere $w = 0^{n^2}$, que seguramente estará en L.
- Entonces $w = xyz$, donde $|xy| \leq n$ y $y \neq \epsilon$
- Por PL $xyyz$ está en L. Pero la longitud de $xyyz$ es más grande que n^2 y no más grande que $n^2 + n$.
- Sin embargo, el próximo cuadrado perfecto después de n^2 es $(n+1)^2 = n^2 + 2n + 1$.
- Así, $xyyz$ no es de longitud cuadrada y no está en L, por lo que por "prueba por contradicción" L no es regular.

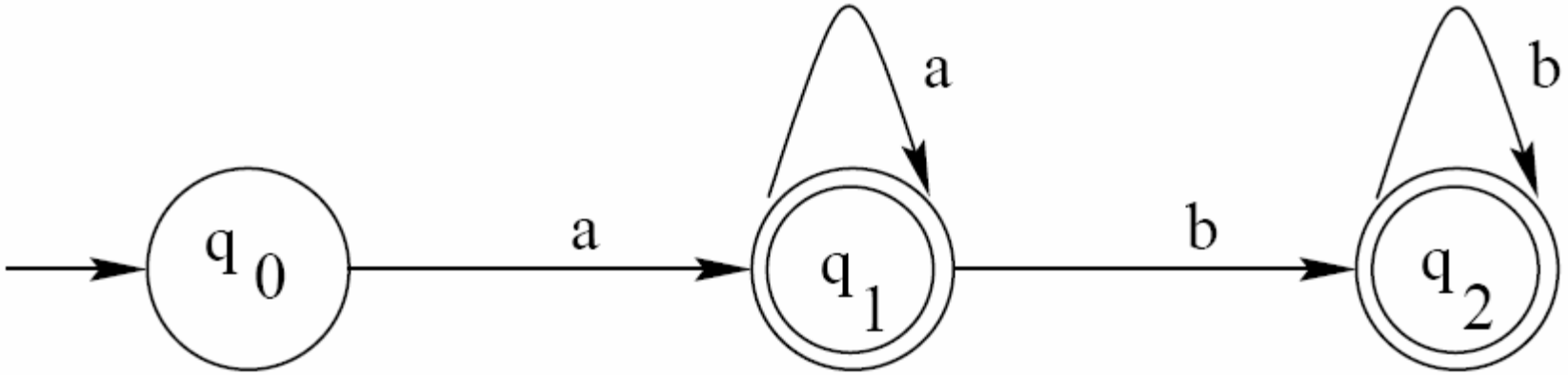
Propiedades de Cerradura

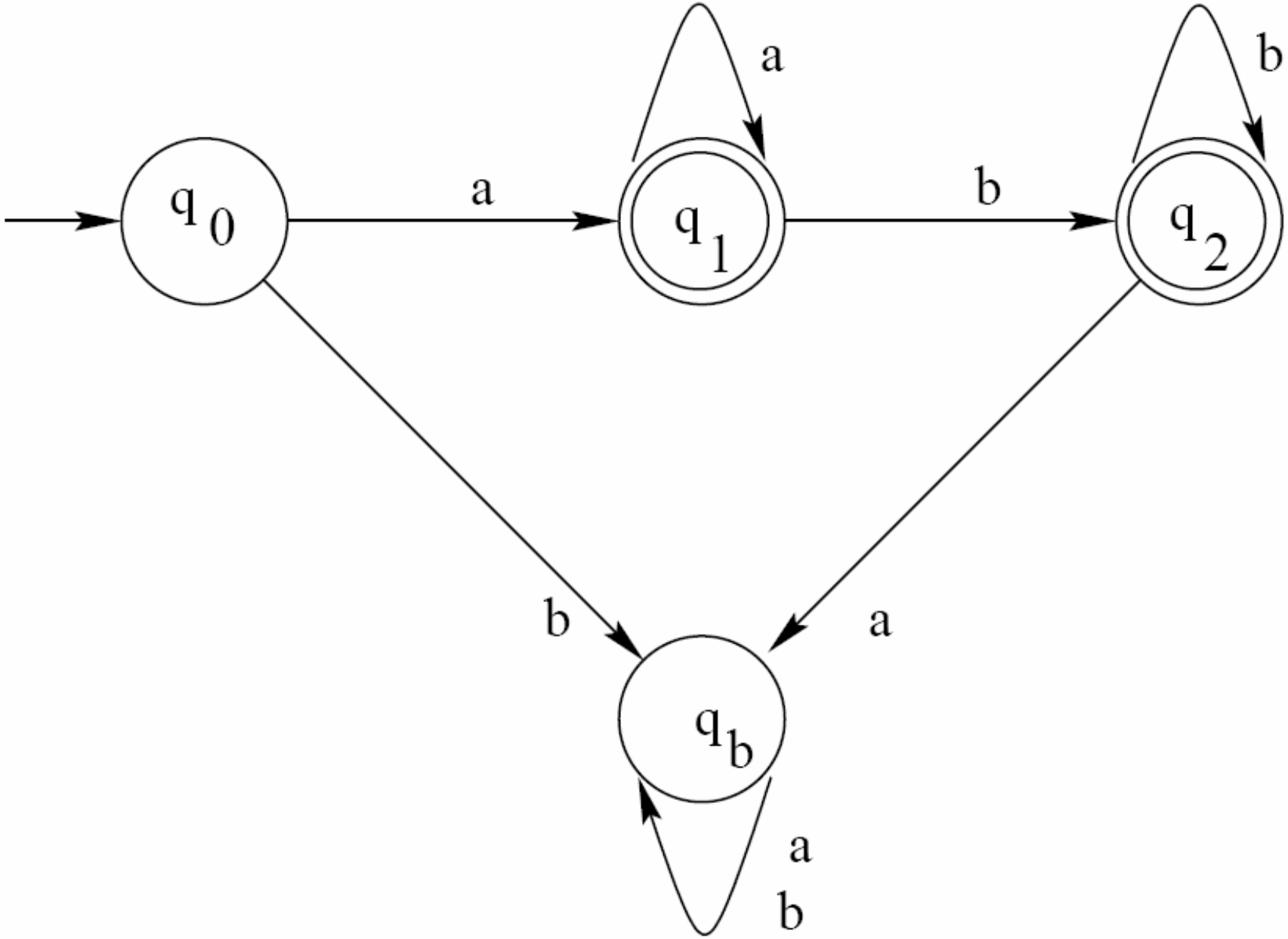
Algunas operaciones sobre lenguajes regulares garantizan producir lenguajes regulares:

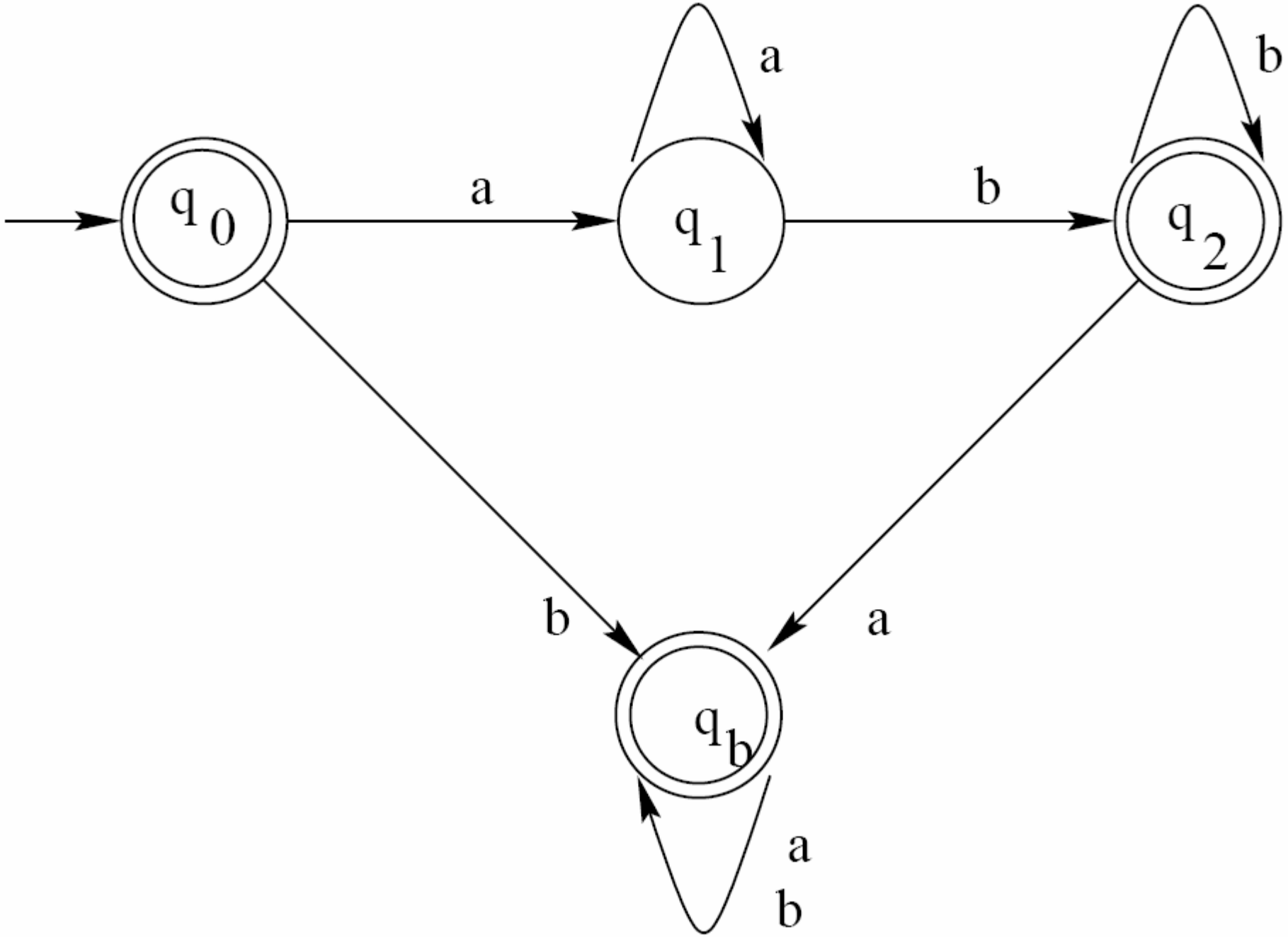
- Unión: $L \cup M$ lenguajes con cadenas en L, M o en ambos.
- Intersección: $L \cap M$ lenguajes con cadenas en ambos.
- Complemento: $\overline{L}$ cadenas que no están en L.
- Diferencia: $L \backslash M$ o $L - M$.
- Inversión: $L^R = \{w^R : w \in L\}$ (R = al revés)
- Cerradura: L^*
- Concatenación: $L.M$
- Homomorfismo (substitución): $h(L) = \{h(w) \in L\} h$ es un homomorfismo.
- Homomorfismo inverso (substitución inversa): $h^{-1}(L) = \{w \in \Sigma : h(w) \in L, h : \Sigma \rightarrow\}$ es un homomorfismo.

- **Unión**: la unión de lenguajes regulares es regular. Sea $L = L(E)$ y $M = L(F)$. Entonces $L(E + F) = L \cup M$, por la definición de "+" en RE.
- **Complemento**: Si L es un lenguaje regular sobre Σ, entonces tambi´en lo es $\overline{L} = \Sigma^*$ - L. Todos los estados son de aceptaci´on excepto los F.

- **Ejemplo**: Sea L definido por el siguiente DFA


q 0
a
q 1
a
b
q 2
b

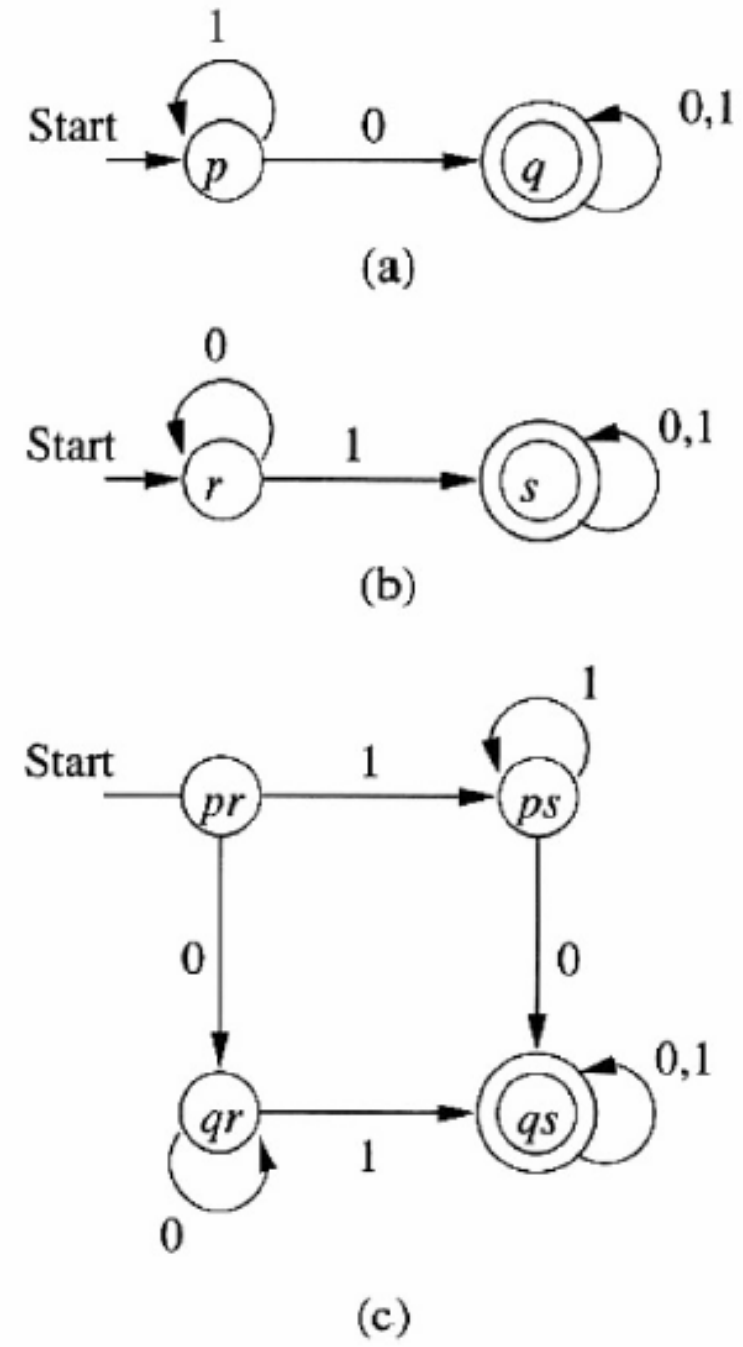

q 0
a
q 1
a
b
q 2
b
b
a
q b
a
b


q 0
a
q 1
a
b
q 2
b
b
a
q b
a
b

- **Intersección**: Si L y M son regulares, entonces también $L \cap M$. Usando las leyes de Morgan: $L \cap M = \overline{\overline{L} \cup \overline{M}}$.
- Para esto también se puede construir un autómata que simula A_L y A_M en paralelo y llega a un estado de aceptación si A_L y A_M también lo hacen.

L_1, L_2 regulares $\Rightarrow L_1 = L(A_1), L_2 = L(A_2)$ con
$A_i = (Q_i, \Sigma, \delta_i, q_i, F_i)$, $i = 1, 2$.
Construimos $A = (Q, \Sigma, \delta, q_0, F)$ con:

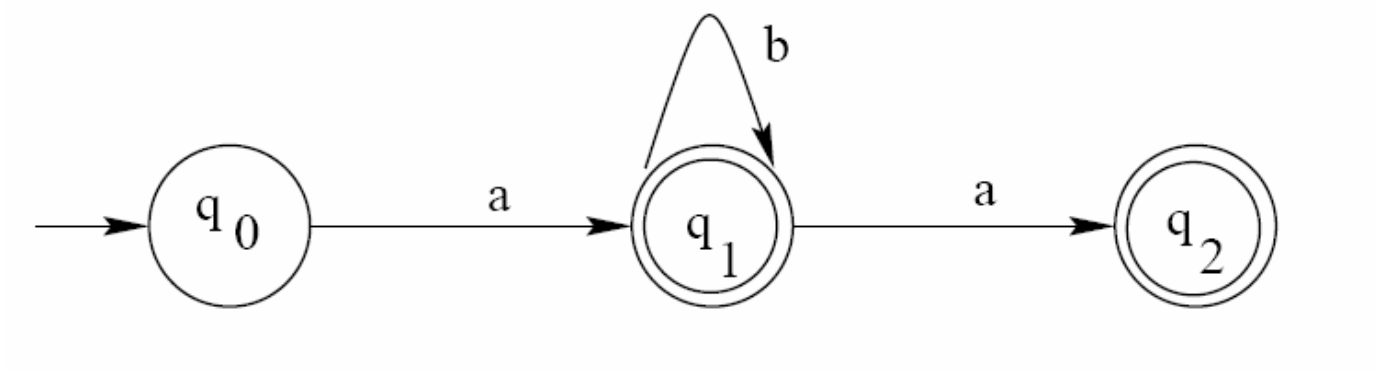
- $Q = Q_1 \times Q_2$
- $q_0 = [q_1, q_2]$
- $F = F_1 \times F_2$
- $\delta([p_1, p_2], a) = [\delta_1(p_1, a), \delta_2(p_2, a)]$,
  $\forall\, p_1 \in Q_1, \forall\, p_2 \in Q_2, \forall\, a \in \Sigma$

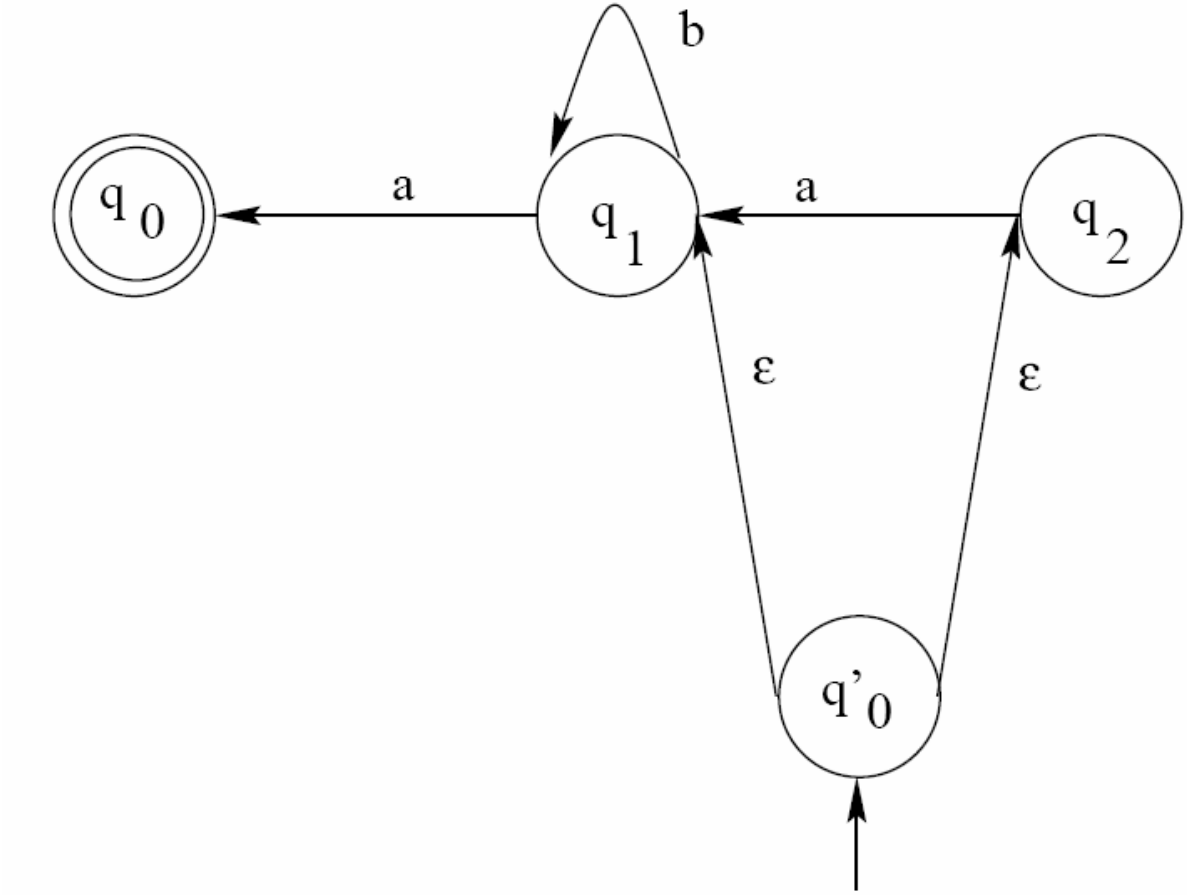

1
Start
0
0,1
p
q
(a)
0
Start
1
0,1
r
s
(b)
1
Start
1
pr
ps
0
0
0,1
qr
qs
1
0
(c)

- **Diferencia**: $L \backslash M$ lenguaje en L pero no en M, $L \backslash M = L \cap \overline{M}$.
- **Inversión**: w^R es la cadena invertida w. Ejemplo: $0010^R = 0100$.
- Inversión de un lenguaje L es el lenguaje con todas las cadenas de L invertidas. Por ejemplo:
  $L = \{001, 10, 111\}$, $L^R = \{100, 01, 111\}$.
  Se puede probar que $L(E^R) = (L(E))^R$.
  En particular, $E^R = E$ para $\emptyset$ conjunto vacío y a.
  Si $E = F + G$, entonces, $E^R = F^R + G^R$
  Si $E = F \cdot G$, entonces $E^R = G^R \cdot F^R$

Ejemplo

- Por ejemplo, $L(E_1) = \{01, 111\}$ y $L(E_2) = \{00, 10\}$.
- $L(E_1)L(E_2) = \{0100, 0110, 11100, 11110\}$
- $L^R(E_1) = \{10, 111\}$ y $L^R(E_2) = \{00, 01\}$
- $L^R(E_1)L^R(E_2) = \{0010, 00111, 0110, 01111\} = (L(E_1)L(E_2))^R$
- Si $L = (0 + 1)0^*$, $L^R = (0^*)^R(0 + 1)^R = 0^*(0 + 1)$
- Si $E = F^*$, entonces $E^R = (F^R)^*$

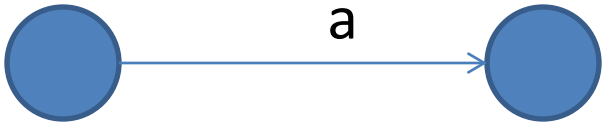

b
q 0
a
q 1
a
q 2

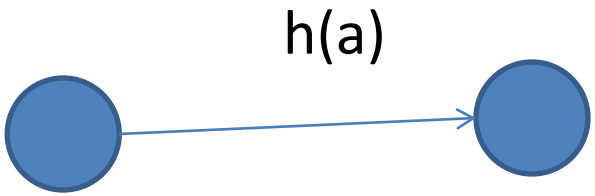

b
q 0
a
q 1
a
q 2
ε
ε
q'0

Homomorfismo

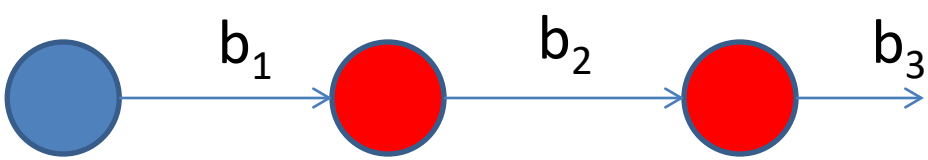
- Un *homomorfismo* sobre Σ es una función $h : \Sigma^* \to \Theta^*$, donde Σ y Θ son alfabetos.
- Sea $w = a_1 a_2 \ldots a_n \in \Sigma$. Entonces $h(w) = h(a_1)h(a_2)\ldots h(a_n)$ y $h(L) = \{h(w) \in L\}$.
- **Ejemplo**: sea $h : \{0,1\}^* \to \{a,b\}^*$ definido como $h(0) = ab$, y $h(1) = \epsilon$. Entonces $h(0011) = abab$ y $h(L(10^*1)) = L((ab)^*)$.
  $h(L)$ es regular si L es regular.
  Sea $L = L(A)$ para un FA A, queremos probar que $\mathrm{h}(L(A))$ es regular.

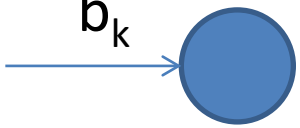
Homomorfismo





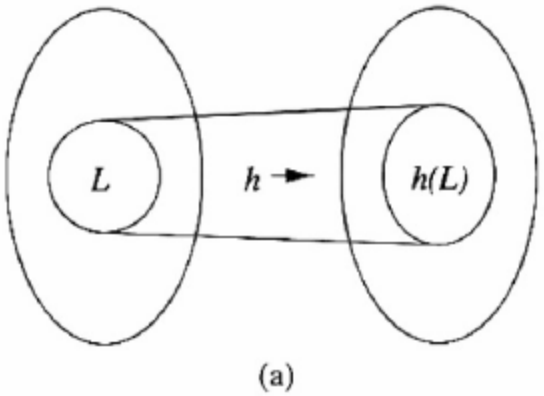
$h(a) = b_1 b_2 \ldots b_k$



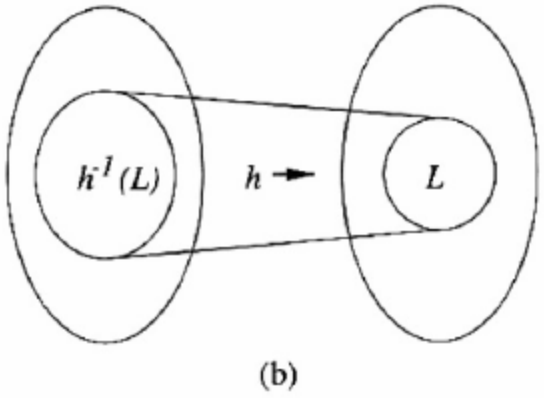


Homomorfismo Inverso

Sea $h : \Sigma^* \to \Theta^*$ un homomorfismo. Sea $L \subseteq \Theta^*$, entonces $h^{-1}(L) = \{w | w \in \Sigma^* : h(w) \in L\}$.



(a)



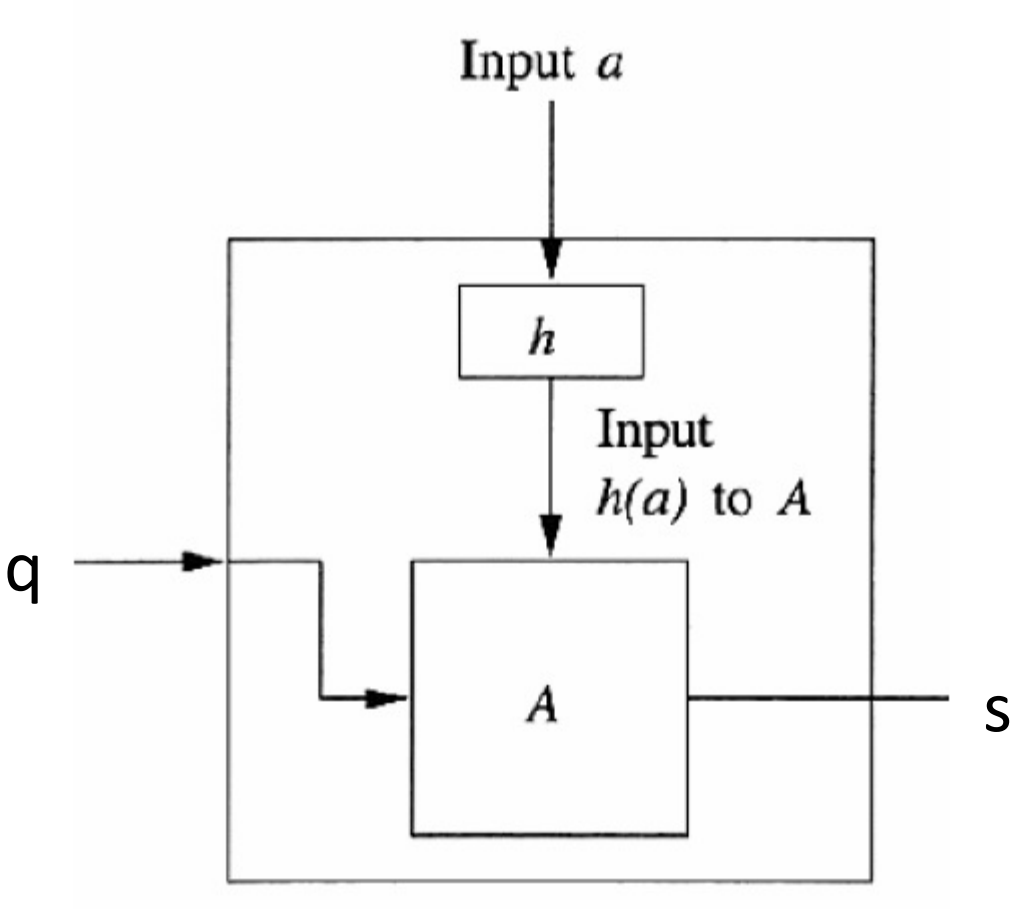
(b)

Ejemplo

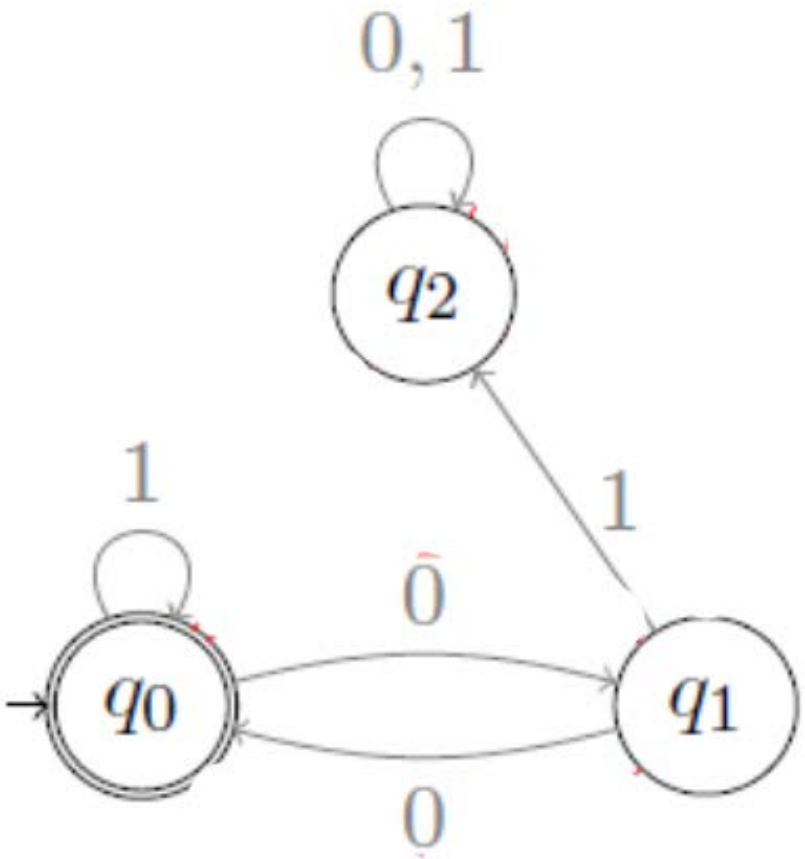
- Sea L el lenguaje de la expresión regular $(00 + 1)^*$. Sea h un homomorfismo definido por: $h(a) = 01$ y $h(b) = 10$.
- En particular: $h^{-1}(L) = (ba)^*$ (que combinación de a's y b's me dan el mismo lenguaje?)
- $h(ba) = 1001$ y $h(w)$ que son n repeticiones de 1001 están en L.
- Por otro lado, si w empieza solo con a (01), termina con b (10), tiene dos a's seguidas (0101) o dos b's seguidas (1010), no se cumple

Homomorfismo Inverso

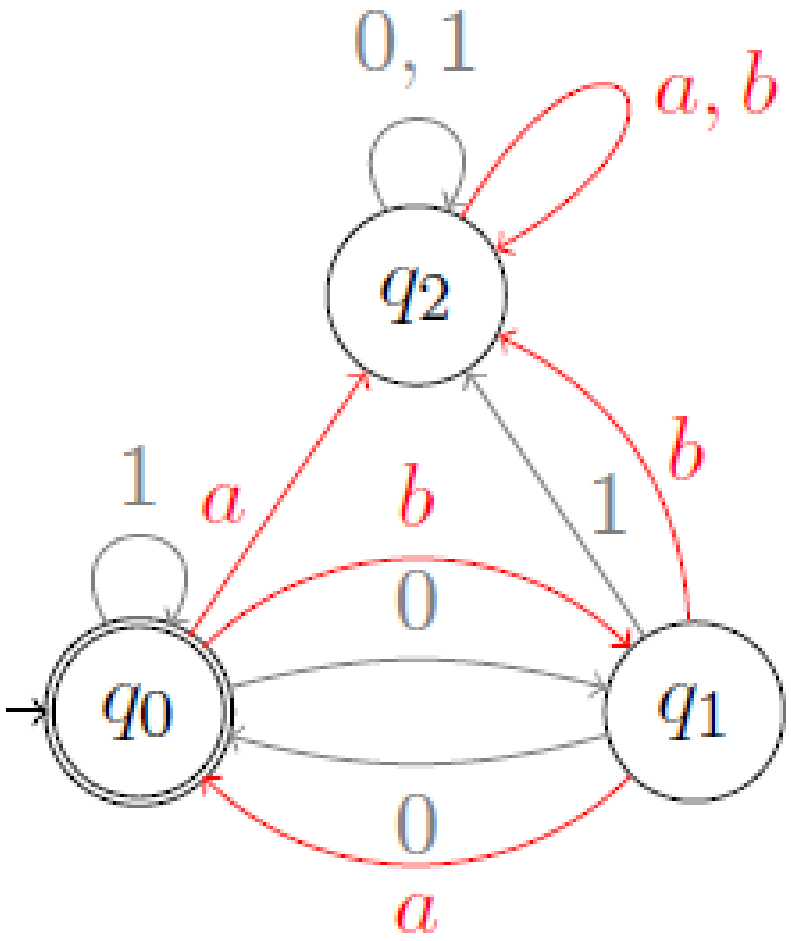
- Sea $h : \Sigma^* \to \Theta^*$ un homomorfismo. Sea $L \subseteq \Theta^*$ un lenguaje regular, entonces $h^{-1}(L)$ es regular.
- El DFA del homomorfismo inverso usa los estados del DFA original, pero cambia los símbolos de entrada de acuerdo a la función h antes de decidir el siguiente estado.


Input a
h
Input
$h(a)$ to A
q
A
s

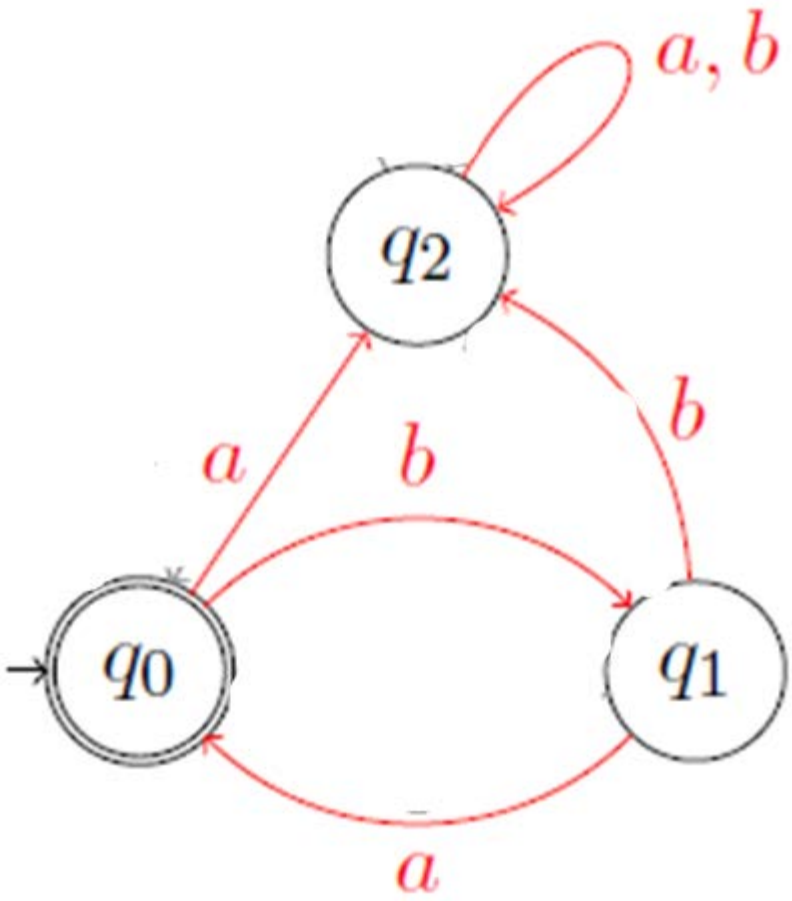
$L = L\left((00 \cup 1)^*\right).\ h(a) = 01,\ h(b) = 10.$



$L = L\left((00 \cup 1)^*\right).\ h(a) = 01,\ h(b) = 10.$



$L = L\left((00 \cup 1)^*\right).\ h(a) = 01,\ h(b) = 10.$



Propiedades de decisión

- Algunas de las preguntas que nos podemos hacer acerca de lenguajes son si el lenguaje es vacío, si una cadena particular pertenece al lenguaje o si dos descripciones definen el mismo lenguaje.
- También podemos cuestionarnos acerca del costo computacional requerido para resolver estas preguntas o por ejemplo el requerido para hacer la conversión entre una representación a otra.

Análisis de Complejidad

Transformar un ϵ-NFA a un DFA:

- Para cada símbolo y cada subconjunto el calcular la función de transición para todos los estados, requiere a lo más n^3 pasos, lo cual nos da una complejidad total de $O(n^3 2^n)$.

Análisis de Complejidad

Otras Transformaciones:

- **Transformar un DFA a un NFA**: Sólo se requiere poner corchetes a los estados, lo cual nos da O(n).
- **Transformar un FA a una RE**: $O(n^3 4^n)$. Es todavía peor si el FA es NFA. Si lo convertimos primero a un DFA nos da: $O(n^3 4^{n^3 2^n})$.
- **Transformar de una RE a un FA**: se puede construir un autómata en n pasos. Si eliminamos transiciones ϵ toma $O(n^3)$. Si se requiere un DFA puede tomar un número exponencial de pasos.

Análisis de Complejidad

- **Decidir si un lenguaje es vacío**: el probar si existe un camino entre un estado inicial a uno final o de aceptación, es simplemente un problema de ver si un nodo está conectado en un grafo, lo cual tiene una complejidad de $O(n^2)$.
- **Probar por pertenencia a un lenguaje regular**: ver si una cadena es miembro del lenguaje. Si la cadena w es de longitud n para un DFA, esto es de complejidad $O(n)$. Si es un NFA de s estados, entonces la complejidad es: $O(ns^2)$. Si es un ϵ-NFA entonces la complejidad es $O(ns^3)$.

Detección de ciclos

Algoritmo Detectar-ciclos (primera parte)

[1] inicio

[2] Alcanzables ← $\{q_0\}$ y q_0 *no marcado*

[3] Visitados ← $\varnothing$

[4] Ciclo ← *falso*

[5] mientras (Ciclo = falso) y (haya un estado no marcado $q \in$ *Alcanzables)*

[6] hacer

[7] *Marcar el estado q*

[8] *Visitados* ← *Visitados* $\cup \{q\}$

[9] *Auxiliar* ← $\{q' \mid \exists \sigma \in \Sigma \wedge \delta(q, \sigma) = q'\}$

[10] *si Visitados* $\cap$ *Auxiliar* $\neq \varnothing$

[11] *entonces Ciclo* ← *verdadero*

[12] *si no*

[13] *para* $q \in$ *Auxiliar* − *Alcanzables hacer*

[14] *Alcanzables* ← *Alcanzables* $\cup \{q\}$ *y q no marcado*

[15] *fin para*

[16] *fin si*

[17] fin mientras

[18] si (Ciclo = falso)

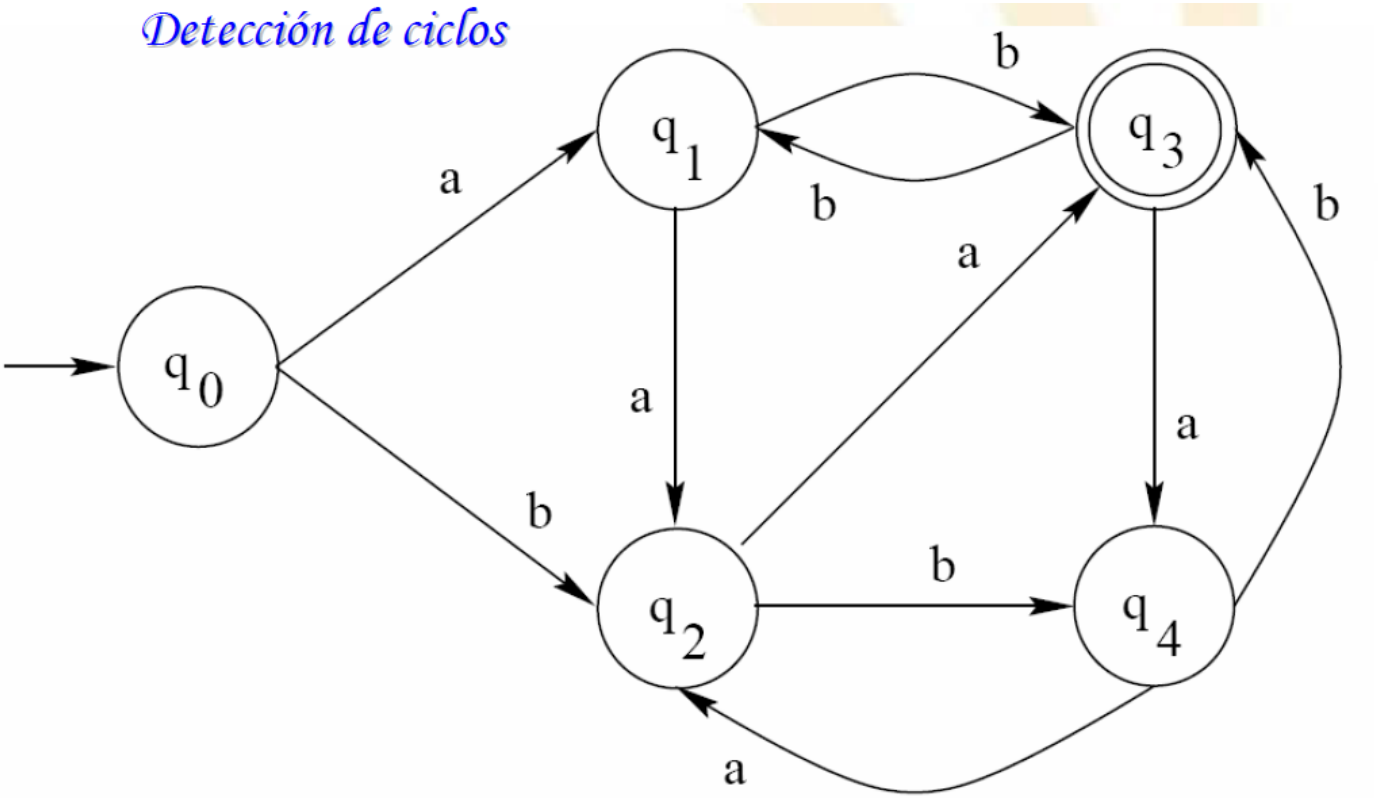
[19] *entonces escribir "Autómata sin ciclos"*

[20] *si no escribir "Autómata con ciclos"*

[21] fin si

[22] fin

Detección de ciclos



Paso	*Alcanzables*	*Visitados*	*Auxiliar*	*Visitados* $\cap$ *Auxiliar*
0	$\{q_0\}$	$\emptyset$	$\emptyset$	$\emptyset$
1	$\{\underline{q_0}\}$	$\{q_0\}$	$\{q_1, q_2\}$	$\emptyset$
2	$\{\underline{q_0}, \underline{q_1}, q_2\}$	$\{q_0, q_1\}$	$\{q_2, q_3\}$	$\emptyset$
3	$\{\underline{q_0}, \underline{q_1}, \underline{q_2}\}$	$\{q_0, q_1, q_2\}$	$\{q_3, q_4\}$	$\emptyset$
4	$\{\underline{q_0}, \underline{q_1}, \underline{q_2}, \underline{q_3}\}$	$\{q_0, q_1, q_2, q_3\}$	$\{q_1, q_4\}$	$\{q_1\}$

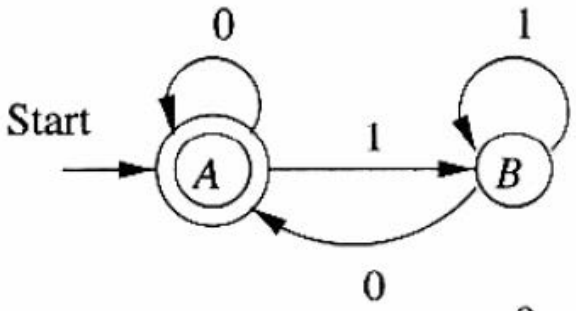
Equivalencia y minimización de autómatas

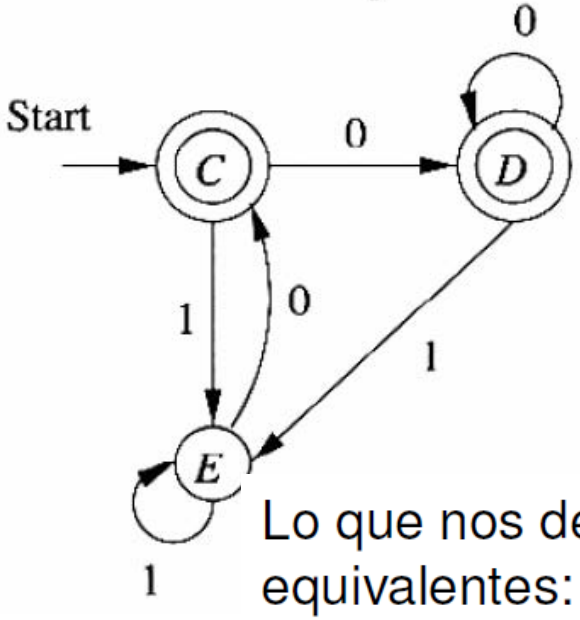
- Lo que queremos saber es si dos autómatas diferentes definen el mismo lenguaje.
- Primero definiremos lo que son estados equivalentes.
- Dos estados p y q dentro de un autómata son *equivalentes*:
  $p \equiv q \Leftrightarrow \forall w \in \Sigma^* : \hat{\delta}(p, w) \in F \Leftrightarrow \hat{\delta}(q, w) \in F$.
- Si no, entonces se dice que son *distinguibles*. Osea que p y q son distinguibles si:
  $\exists w : \hat{\delta}(p, w) \in F \wedge \hat{\delta}(q, w) \notin F$ o viceversa.

Prueba de equivalencia entre lenguajes regulares

- Sea L y M dos lenguajes regulares (dados en alguna forma).
- Convierte L y M a DFA's
- Junta los dos DFA's
- Prueba si los dos estados iniciales son equivalentes, en cuyo caso $L = M$. Si no son equivalentes, entonces $L \neq M$.

Ejemplo





B	x			
C		x		
D		x		
E	x		x	x
	A	B	C	D

Lo que nos deja los siguientes pares de estados equivalentes: $\{A, C\}, \{A, D\}, \{C, D\}$ y $\{B, E\}$. Como A y C son equivalentes, entonces los dos autómatas son equivalentes.