

Capítulo 2

Lenguajes Regulares y Autómatas Finitos

La definición de lenguaje presentada en el Capítulo 1 es muy amplia: cualquier conjunto de cadenas (finito o infinito) es un lenguaje. Es de esperarse que no todos los lenguajes tengan la misma importancia o relevancia. En este capítulo se estudia la primera colección o familia realmente importante de lenguajes, los llamados lenguajes regulares. Estos lenguajes pueden ser *reconocidos* o *aceptados*, en un sentido que se precisará más adelante, por máquinas abstractas muy simples, llamadas autómatas finitos.

2.1. Lenguajes regulares

Los *lenguajes regulares* sobre un alfabeto dado Σ son todos los lenguajes que se pueden formar a partir de los lenguajes básicos \emptyset , $\{\lambda\}$, $\{a\}$, $a \in \Sigma$, aplicando un número finito de veces las operaciones de unión, concatenación y estrella de Kleene.

A continuación presentamos la definición recursiva de la colección de todos los lenguajes regulares sobre un alfabeto dado Σ .

- (1) \emptyset , $\{\lambda\}$ y $\{a\}$, para cada $a \in \Sigma$, son lenguajes regulares sobre Σ . Estos son los denominados lenguajes regulares básicos.
- (2) Si A y B son lenguajes regulares sobre Σ , también lo son

$$\begin{array}{ll} A \cup B & \text{(unión),} \\ A \cdot B & \text{(concatenación),} \\ A^* & \text{(estrella de Kleene).} \end{array}$$

La unión, la concatenación y la estrella de Kleene se denominan *operaciones regulares*.

Ejemplos Sea $\Sigma = \{a, b\}$. Los siguientes son lenguajes regulares sobre Σ porque los podemos obtener a partir de los lenguajes básicos $\{\lambda\}$, $\{a\}$ y $\{b\}$ por medio de un número finito de uniones, concatenaciones o estrellas de Kleene:

1. $\{a\}^* = \{\lambda, a, a^2, a^3, a^4, \dots\}$.
2. $\{a\}^+ = \{a\}^* \cdot \{a\} = \{a, a^2, a^3, a^4, \dots\}$.
3. $\{a, b\}^* = (\{a\} \cup \{b\})^*$. Es decir, el lenguaje de todas las cadenas sobre Σ es regular.
4. El lenguaje A de todas las cadenas que tienen exactamente una a :

$$A = \{b\}^* \cdot \{a\} \cdot \{b\}^*.$$


5. El lenguaje B de todas las cadenas que comienzan con b :

$$B = \{b\} \cdot \{a, b\}^* = \{b\} \cdot (\{a\} \cup \{b\})^*.$$

6. El lenguaje C de todas las cadenas que contienen la subcadena ba :


$$C = (\{a\} \cup \{b\})^* \cdot \{b\} \cdot \{a\} \cdot (\{a\} \cup \{b\})^*.$$

7. $(\{a\} \cup \{b\})^* \cdot \{a\}$.
8. $(\{\lambda\} \cup \{a\}) \cdot (\{b\} \cup \{b\} \cdot \{a\})^*$.

 Es importante observar que *todo lenguaje finito* $L = \{w_1, w_2, \dots, w_n\}$ es regular ya que L se puede obtener como una unión finita:

$$L = \{w_1\} \cup \{w_2\} \cup \dots \cup \{w_n\},$$

y cada cadena w_i de L es la concatenación de un número finito de símbolos; si $w_i = a_1 a_2 \dots a_k$, entonces $\{w_i\} = \{a_1\} \cdot \{a_2\} \dots \{a_k\}$.

 Según la definición recursiva de los lenguajes regulares, si A_1, A_2, \dots, A_k son k lenguajes regulares, entonces $\bigcup_{i=1}^k A_i$ es regular. Pero si $\{A_i\}_{i \in I}$ es una familia *infinita* de lenguajes regulares, la unión $\bigcup_{i \in I} A_i$ no necesariamente es regular, como se verá en el Capítulo 2.

2.2. Expresiones regulares

Los ejemplos de la sección 2.1 muestran que en la presentación de los lenguajes regulares abundan las llaves o corchetes $\{ \}$ y los paréntesis. Con el propósito de hacer más legible la representación de los lenguajes regulares, se introducen las denominadas expresiones regulares.

La siguiente es la definición recursiva de las *expresiones regulares* sobre un alfabeto Σ dado.

(1) Expresiones regulares básicas:

- \emptyset es una expresión regular que representa al lenguaje \emptyset .
- λ es una expresión regular que representa al lenguaje $\{\lambda\}$.
- a es una expresión regular que representa al lenguaje $\{a\}$, para cada $a \in \Sigma$.

(2) Si R y S son expresiones regulares sobre Σ que representan los lenguajes regulares A y B , respectivamente, entonces

- $(R \cup S)$ es una expresión regular que representa al lenguaje $A \cup B$.
- (RS) es una expresión regular que representa al lenguaje $A \cdot B$.
- $(R)^*$ es una expresión regular que representa al lenguaje A^* .

Los paréntesis (y) son símbolos de agrupación y se pueden omitir si no hay peligro de ambigüedad. Es decir, podemos escribir simplemente $R \cup S$, RS y R^* para la unión, la concatenación y la estrella de Kleene, respectivamente, siempre y cuando no haya confusiones ni ambigüedades. Los paréntesis son inevitables en expresiones grandes para delimitar el alcance de la operaciones involucradas.

La anterior definición se puede escribir de manera más compacta utilizando la siguiente notación:

$$L[R] := \text{lenguaje representado por } R.$$

Con esta notación la definición recursiva de las expresiones regulares se puede presentar en la siguiente forma.

(1) \emptyset , λ y a (donde $a \in \Sigma$) son expresiones regulares tales que

$$\begin{aligned} L[\emptyset] &= \emptyset. \\ L[\lambda] &= \{\lambda\}. \\ L[a] &= \{a\}, \text{ para cada } a \in \Sigma. \end{aligned}$$

(2) Si R y S son expresiones regulares, entonces

$$\begin{aligned} L[(R \cup S)] &= L[R] \cup L[S]. \\ L[(RS)] &= L[R] \cdot L[S]. \\ L[(R)^*] &= (L[R])^*. \end{aligned}$$

Ejemplo Dado el alfabeto $\Sigma = \{a, b, c\}$,

$$(a \cup b^*)a^*(bc)^*$$

es una expresión regular que representa al lenguaje

$$(\{a\} \cup \{b\}^*) \cdot \{a\}^* \cdot (\{b\} \cdot \{c\})^*.$$

Ejemplo Dado el alfabeto $\Sigma = \{a, b\}$,

$$(\lambda \cup ab)(a \cup b)^*(ba)^*$$

es una expresión regular que representa al lenguaje

$$(\{\lambda\} \cup \{a\} \cdot \{b\}) \cdot (\{a\} \cup \{b\})^* \cdot (\{b\} \cdot \{a\})^*.$$

Ejemplos Sea $\Sigma = \{a, b\}$. Podemos obtener expresiones regulares para algunos de los lenguajes mencionados de la sección 2.1:

1. El lenguaje de todas las cadenas sobre Σ :

$$(a \cup b)^*.$$

2. El lenguaje A de todas las cadenas que tienen exactamente una a :

$$b^*ab^*.$$

3. El lenguaje B de todas las cadenas que comienzan con b :

$$b(a \cup b)^*.$$

4. El lenguaje C de todas las cadenas que contienen la subcadena ba :

$$(a \cup b)^*ba(a \cup b)^*.$$

La representación de lenguajes regulares por medio de expresiones regulares no es única. Es posible que haya varias expresiones regulares diferentes para el mismo lenguaje. Por ejemplo, $b(a \cup b)^*$ y $b(b \cup a)^*$ representan el mismo lenguaje. Otro ejemplo: las dos expresiones regulares $(a \cup b)^*$ y $(a^*b^*)^*$ representan el mismo lenguaje por la propiedad $(A \cup B)^* = (A^*B^*)^*$ mencionada en la página 16.

Por la propiedad $A^+ = A^* \cdot A = A \cdot A^*$, la clausura positiva $+$ se puede expresar en términos de $*$ y concatenación. Por tal razón, se permite el uso de $+$ en expresiones regulares.

Ejemplo Las tres expresiones $(a^+ \cup b)ab^+$, $(aa^* \cup b)abb^*$ y $(a^*a \cup b)ab^*b$ representan el mismo lenguaje. Análogamente, las tres siguientes expresiones

$$\begin{aligned} &(a \cup b)^+ \cup a^+b^+. \\ &(a \cup b)(a \cup b)^* \cup aa^*bb^*. \\ &(a \cup b)^*(a \cup b) \cup a^*ab^*b. \end{aligned}$$

representan el mismo lenguaje.

- ✎ En las expresiones regulares se usan reglas de precedencia para la unión, la concatenación y la estrella de Kleene similares a las que se utilizan en las expresiones algebraicas para la suma, la multiplicación y la exponenciación. El orden de precedencia es $*$, \cdot , \cup ; es decir, la estrella actúa antes que la concatenación y ésta antes que la unión. Por ejemplo, la expresión $ba^* \cup c$ se interpreta como $[b(a)^*] \cup c$.
- ✎ La propiedad distributiva $A \cdot (B \cup C) = A \cdot B \cup A \cdot C$, leída de izquierda a derecha sirve para distribuir A con respecto a una unión, y leída de derecha a izquierda sirve para “factorizar” A . Se deduce que si R , S y T son expresiones regulares, entonces

$$L[R(S \cup T)] = L[RS \cup RT].$$

Esto permite obtener nuevas expresiones regulares ya sea distribuyendo o factorizando.

- ✎ En las expresiones regulares también se permiten potencias. Podemos escribir, por ejemplo, a^2 en vez de aa y a^3 en vez de a^2a , aa^2 o aaa .

Los ejemplos y ejercicios que aparecen a continuación son del siguiente tipo: dado un lenguaje L sobre un determinado alfabeto encontrar una expresión regular R tal que $L[R] = L$. Para resolver estos problemas hay que recalcar que la igualdad $L[R] = L$ es estricta en el sentido de que toda cadena de $L[R]$ debe pertenecer a L , y recíprocamente, toda cadena de L debe estar incluida en $L[R]$.

Ejemplo Sea $\Sigma = \{0, 1\}$. Encontrar una expresión regular para el lenguaje de todas las cadenas cuyo penúltimo símbolo, de izquierda a derecha, es un 0.

Soluciones: El penúltimo símbolo debe ser un cero pero para el último hay dos posibilidades: 0 o 1. Estos casos se representan como una unión:

$$(0 \cup 1)^*00 \cup (0 \cup 1)^*01.$$

Factorizando podemos obtener otras expresiones regulares:

$$\begin{aligned} &(0 \cup 1)^*(00 \cup 01). \\ &(0 \cup 1)^*0(0 \cup 1). \end{aligned}$$

Ejemplo Sea $\Sigma = \{a, b\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que tienen un número par de símbolos (cadenas de longitud par ≥ 0).

Soluciones: Aparte de la cadena vacía λ , todas las cadenas de longitud par ≥ 2 se obtienen concatenando de todas las formas posibles los cuatro bloques aa , ab , ba y bb . Así llegamos a la expresión

$$(aa \cup ab \cup ba \cup bb)^*.$$

Otra expresión correcta es $(a^2 \cup b^2 \cup ab \cup ba)^*$. Utilizando la propiedad distributiva de la concatenación con respecto a la unión, encontramos otra expresión regular para este lenguaje:

$$[(a \cup b)(a \cup b)]^*.$$

Ejemplo Sea $\Sigma = \{a, b\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que tienen un número par ≥ 0 de a 's.

Soluciones: Si la cadena tiene cero a 's, significa que solamente posee b 's. Todas las demás cadenas tienen un número par de a 's (2, 4, 6, ...) que aparecen rodeadas a izquierda o a derecha por cualquier cantidad de b 's (posiblemente 0 b 's). Encontramos así varias posibles soluciones:

$$\begin{aligned} &b^*(b^*ab^*ab^*)^*. \\ &(b^*ab^*ab^*)^*b^*. \\ &b^*(b^*ab^*ab^*)^*b^*. \\ &(b^*ab^*ab^*)^* \cup b^*. \\ &(ab^*a \cup b)^*. \\ &(ab^*a \cup b^*)^*. \end{aligned}$$

La expresión regular $(b^*ab^*ab^*)^*$ no es una solución correcta para este problema porque no incluye las cadenas con cero a 's (aparte de λ). La expresión regular $R = b^*(ab^*a)^*b^*$ tampoco es correcta porque en medio de dos bloques de la forma ab^*a no se podrían insertar b 's. De modo que cadenas como $abab^2aba$ y $ab^3ab^4ab^5a$ no harían parte de $L[R]$, a pesar de que poseen un número par de a 's.

Ejemplo Sea $\Sigma = \{0, 1\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que no contienen dos ceros consecutivos.

Soluciones: La condición de que no haya dos ceros consecutivos implica que todo cero debe estar seguido necesariamente de un uno, excepto un cero al final de la cadena. Por lo tanto, las cadenas de este lenguaje se obtienen concatenando unos con bloques 01, de todas las formas posibles. Hay que tener en cuenta, además, que la cadena puede terminar ya sea en 1 o en 0. A partir de este análisis, llegamos a la expresión regular

$$(1 \cup 01)^* \cup (1 \cup 01)^*0.$$

Factorizando $(1 \cup 01)^*$, obtenemos otra expresión para este lenguaje: $(1 \cup 01)^*(\lambda \cup 0)$. Otras dos soluciones análogas son:

$$\begin{aligned} &(1 \cup 10)^* \cup 0(1 \cup 10)^*. \\ &(\lambda \cup 0)(1 \cup 10)^*. \end{aligned}$$

Ejemplo Sea $\Sigma = \{a, b, c\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que no contienen la subcadena bc .

Solución. Tanto a 's como c 's pueden concatenarse entre sí sin ninguna restricción. Pensamos entonces en una expresión de la forma

$$(a \cup c \cup ?)^*.$$

Una b puede estar seguida solamente de otra b o de una a ; por lo tanto, conjeturamos con la expresión $(a \cup c \cup b^*a)^*$. Pero debemos permitir también cadenas que terminen en bes ; haciendo el ajuste correspondiente llegamos a la primera solución:

$$(a \cup c \cup b^*a)^*b^*.$$

Esta expresión puede simplificarse omitiendo la a inicial:

$$(c \cup b^*a)^*b^*.$$

ya que el bloque b^*a permite obtener cualquier cadena de aes .

También podemos atacar el problema en la forma $(a \cup b \cup ?)^*$. Teniendo en cuenta que debemos permitir ces iniciales e impedir la subcadena bc , llegamos a la solución

$$c^*(a \cup b \cup ac^*)^*,$$

la cual se puede simplificar como $c^*(b \cup ac^*)^*$.

Otras soluciones válidas para este problema son:

$$(a \cup c \cup b^+a)^*b^*.$$

$$c^*(a \cup b \cup ac^+)^*.$$

Ejercicios de la sección 2.2

- ① Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{a, b\}$:
 - (i) Lenguaje de todas las cadenas que comienzan con el símbolo b y terminan con el símbolo a .
 - (ii) Lenguaje de todas las cadenas de longitud impar.
 - (iii) Lenguaje de todas las cadenas que tienen un número impar de aes .
 - (iv) Lenguaje de todas las cadenas en las que el número de bes es un múltiplo ≥ 0 de 3.
 - (v) Lenguaje de todas las cadenas que no comienzan con la subcadena ba ni terminan en b .
 - (vi) $\Sigma = \{a, b\}$. Lenguaje de todas las cadenas que no contienen la subcadena bba .
- ② Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{0, 1, 2\}$:
 - (i) Lenguaje de todas las cadenas que comienzan con 2 y terminan con 1.
 - (ii) Lenguaje de todas las cadenas que no comienzan con 2 ni terminan en 1.

- (iii) Lenguaje de todas las cadenas que tienen exactamente dos ceros.
 - (iv) Lenguaje de todas las cadenas que tienen un número par de símbolos.
 - (v) Lenguaje de todas las cadenas que tienen un número impar de símbolos.
 - (vi) Lenguaje de todas las cadenas que no contienen dos unos consecutivos.
- ③ Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{0, 1\}$:
- (i) Lenguaje de todas las cadenas que tienen por lo menos un 0 y por lo menos un 1.
 - (ii) Lenguaje de todas las cadenas que no contienen tres ceros consecutivos.
 - (iii) Lenguaje de todas las cadenas cuya longitud es ≥ 4 .
 - (iv) Lenguaje de todas las cadenas cuya longitud es ≥ 5 y cuyo quinto símbolo, de izquierda a derecha, es un 1.
 - (v) Lenguaje de todas las cadenas que no terminan en 01.
 - (vi) Lenguaje de todas las cadenas de longitud par ≥ 2 formadas por ceros y unos alternados.
 - (vii) Lenguaje de todas las cadenas de longitud ≥ 2 formadas por ceros y unos alternados.
 - (viii) Lenguaje de todas las cadenas que no contienen dos ceros consecutivos ni dos unos consecutivos.
 - (ix) Lenguaje de todas las cadenas de longitud impar que tienen unos en todas y cada una de las posiciones impares; en las posiciones pares pueden aparecer ceros o unos.
 - (x) Lenguaje de todas las cadenas cuya longitud es un múltiplo de tres.
 - (xi) Lenguaje de todas las cadenas que no contienen cuatro ceros consecutivos.
 - (xii) Lenguaje de todas las cadenas que no comienzan con 00 ni terminan en 11.
 - (xiii) Lenguaje de todas las cadenas que no contienen la subcadena 101.
- ④ Sea $\Sigma = \{a, b\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que tienen un número par ≥ 0 de *aes* y un número par ≥ 0 de *bes*.
NOTA: Encontrar por inspección una expresión regular correcta para este lenguaje no es tan fácil; más adelante resolveremos este problema utilizando autómatas.

2.3. Autómatas finitos deterministas (AFD)

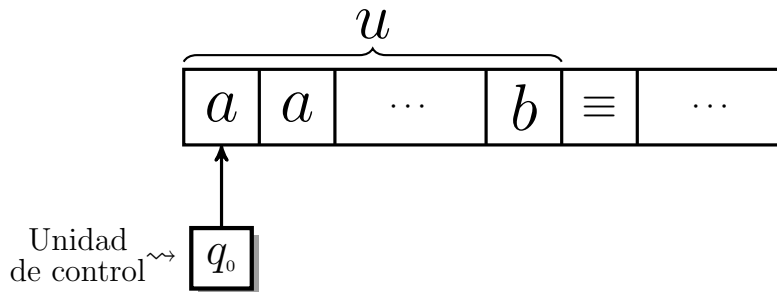
Los *autómatas finitos* son máquinas abstractas que leen de izquierda a derecha cadenas de entrada, las cuales son aceptadas o rechazadas. Un autómata actúa leyendo los símbolos escritos sobre una cinta semi-infinita, dividida en celdas o casillas, sobre la cual se escribe una cadena de entrada u , un símbolo por casilla. El autómata posee una *unidad de control* (también llamada *cabeza lectora*, *control finito* o *unidad de memoria*) que tiene un número finito de configuraciones internas, llamadas *estados del autómata*. Entre los estados de un autómata se destacan el *estado inicial* y los *estados finales* o *estados de aceptación*.

Formalmente, un autómata finito M posee cinco componentes, $M = (\Sigma, Q, q_0, F, \delta)$, a saber:

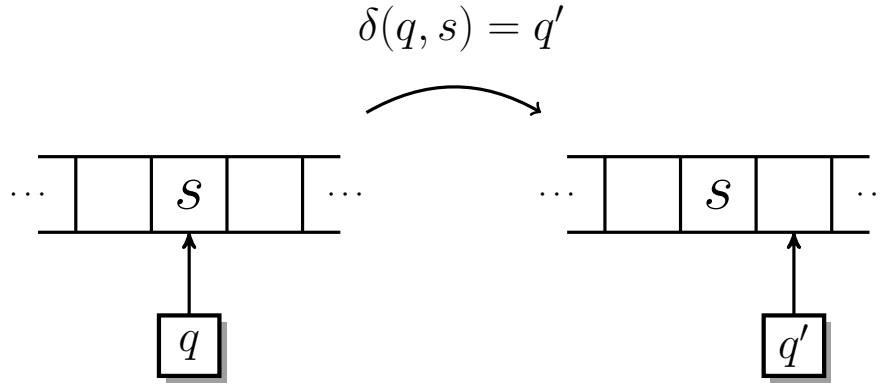
1. Un alfabeto Σ , llamado alfabeto de entrada o alfabeto de cinta. Todas las cadenas que lee M pertenecen a Σ^* .
2. $Q = \{q_0, q_1, \dots, q_n\}$, el conjunto (finito) de los estados internos de la unidad de control.
3. $q_0 \in Q$, estado inicial.
4. $F \subseteq Q$, conjunto de estados finales o de aceptación. F debe ser distinto de \emptyset ; es decir, debe haber por lo menos un estado de aceptación.
5. La función δ de transición del autómata

$$\begin{aligned} \delta : Q \times \Sigma &\longrightarrow Q \\ (q, s) &\longmapsto \delta(q, s) \end{aligned}$$

Una cadena de entrada u se coloca en la cinta de tal manera que el primer símbolo de u ocupa la primera casilla de la cinta. La unidad de control está inicialmente en el estado q_0 escaneando la primera casilla:



La función de transición δ , también llamada *dinámica del autómata*, es la lista de instrucciones que utiliza M para procesar todas las entradas. Las únicas instrucciones son de la forma $\delta(q, s) = q'$, la cual tiene el siguiente significado: en presencia del símbolo s , la unidad de control pasa del estado q al estado q' y se desplaza hacia la casilla situada inmediatamente a la derecha. Esta acción constituye un *paso computacional*:



La unidad de control de un autómata siempre se desplaza hacia la derecha una vez escanea o “consume” un símbolo; no puede retornar ni tampoco sobre-escribir símbolos sobre la cinta.

Puesto que δ está definida para toda combinación estado-símbolo, una cadena de entrada u cualquiera es leída completamente, hasta que la unidad de control encuentra la primera casilla vacía; en ese momento la unidad de control se detiene. Si el estado en el cual termina el procesamiento de u pertenece a F , se dice que la entrada u es *aceptada* (o reconocida) por M ; de lo contrario, se dice que u es *rechazada* (o no aceptada o no reconocida).

Ejemplo

Consideremos el autómata $M = (\Sigma, Q, q_0, F, \delta)$ definido por los siguientes cinco componentes:

$$\Sigma = \{a, b\}.$$

$$Q = \{q_0, q_1, q_2\}.$$

q_0 : estado inicial.

$F = \{q_0, q_2\}$, estados de aceptación.

Función de transición δ :

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

$$\delta(q_0, a) = q_0$$

$$\delta(q_0, b) = q_1$$

$$\delta(q_1, a) = q_1$$

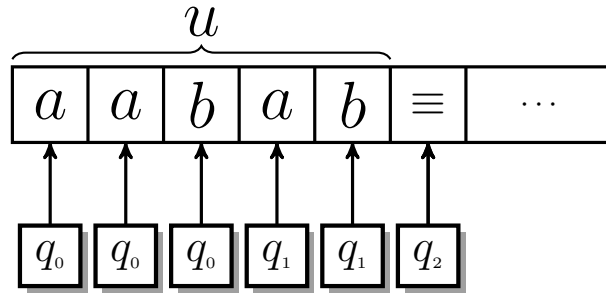
$$\delta(q_1, b) = q_2$$

$$\delta(q_2, a) = q_1$$

$$\delta(q_2, b) = q_1.$$

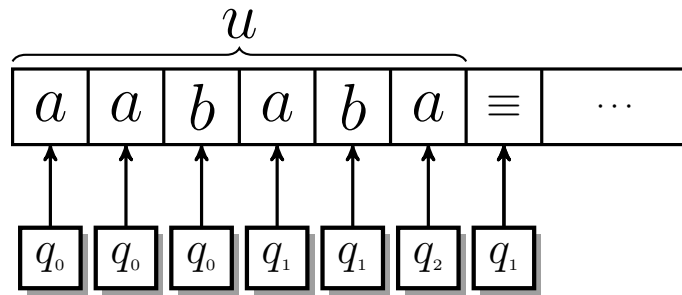
Vamos a ilustrar el procesamiento de tres cadenas de entrada.

1. $u = aabab$.



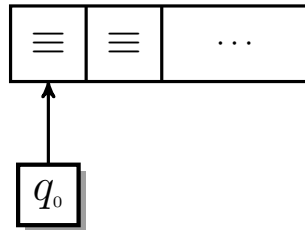
Como q_2 es un estado de aceptación, la cadena de entrada u es aceptada.

2. $v = aababa$.



Puesto que q_1 no es un estado de aceptación, la entrada v es rechazada.

3. Caso especial: la cadena λ es la cadena de entrada.



Como q_0 es un estado de aceptación, la cadena λ es aceptada.

En general se tiene lo siguiente: la cadena vacía λ es aceptada por un autómata M si y solamente si el estado inicial q_0 también es un estado de aceptación.

Los autómatas finitos descritos anteriormente se denominan *autómatas finitos deterministas* (AFD) ya que para cada estado q y para cada símbolo $s \in \Sigma$, la función de transición $\delta(q, s)$ siempre está definida y es un único estado. Esto implica que cualquier cadena de entrada se procesa completamente y de manera única.


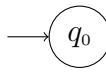

Un autómata M es entonces un mecanismo que clasifica todas las cadenas de Σ^* en dos clases disjuntas: las que son aceptadas y las que son rechazadas. El conjunto de las cadenas aceptadas por M se llama *lenguaje aceptado* o *lenguaje reconocido* por M y se denota por $L(M)$. Es decir,

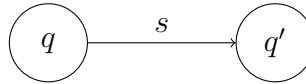
$$\begin{aligned} L(M) &:= \{u \in \Sigma^* : u \text{ es aceptada por } M\} \\ &= \{u \in \Sigma^* : M \text{ termina el procesamiento de } u \text{ en un estado } q \in F\} \end{aligned}$$

2.4. Grafo de un autómata

Un autómata finito se puede representar por medio de un grafo dirigido y etiquetado. Recuerdese que un grafo es un conjunto de vértices o nodos unidos por arcos o conectores; si los arcos tienen tanto dirección como etiquetas, el grafo se denomina *grafo dirigido y etiquetado* o *digrafo etiquetado*.

El digrafo etiquetado de un autómata $M = (\Sigma, Q, q_0, F, \delta)$ se obtiene siguiendo las siguientes convenciones:

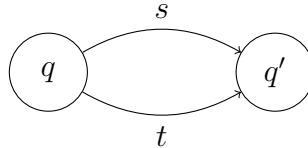
- Los vértices o nodos son los estados del autómata.
- Un estado $q \in Q$ se representa por un círculo con nombre q : 
- El estado inicial q_0 se destaca mediante una “bandera” o “flecha”: 
- Un estado de aceptación q se representa por un círculo doble con nombre q : 
- La transición $\delta(q, s) = q'$ se representa por medio de un arco entre el estado q y el estado q' con etiqueta s :



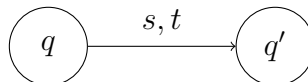
Si $\delta(q, s) = q$ se obtiene lo que se llama un “bucle” (*loop*, en inglés):



Las etiquetas de los arcos entre estados son entonces símbolos del alfabeto de entrada Σ . Si dos estados están unidos por dos arcos que tienen la misma dirección, basta trazar un solo arco con doble etiqueta, separando los símbolos con comas. Así por ejemplo,



donde $s, t \in \Sigma$, se representa simplemente como



El grafo de un autómata es muy útil para hacer el seguimiento o rastreo completo del procesamiento de una cadena de entrada. Una cadena $u \in \Sigma^*$ es aceptada si existe una trayectoria etiquetada con los símbolos de u , que comienza en el estado q_0 y termina en un estado de aceptación.

El grafo de un autómata determinista $M = (\Sigma, Q, q_0, F, \delta)$ tiene la siguiente característica: desde cada estado salen tantos arcos como símbolos tiene el alfabeto de entrada Σ . Esto se debe al hecho de que la función de transición δ está definida para cada combinación estado-símbolo (q, s) , con $q \in Q$ y $s \in \Sigma$.

Ejemplo En la sección anterior se presentó el siguiente autómata $M = (\Sigma, Q, q_0, F, \delta)$:

$\Sigma = \{a, b\}$.

$Q = \{q_0, q_1, q_2\}$.

q_0 : estado inicial.

$F = \{q_0, q_2\}$, estados de aceptación.

Función de transición δ :

δ	a	b
q_0	q_0	q_1
q_1	q_1	q_2
q_2	q_1	q_1

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

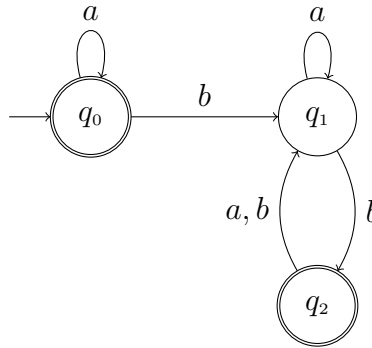
$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_1$

$\delta(q_2, b) = q_1$

El grafo de M es:



Examinando directamente el grafo podemos observar fácilmente que, por ejemplo, las entradas $bbab$ y $aaababbb$ son aceptadas mientras que $baabb$ y $aabaaba$ son rechazadas.

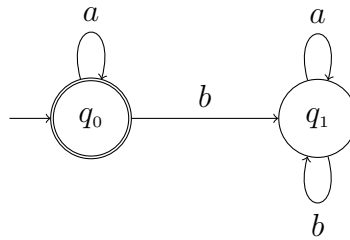
2.5. Diseño de autómatas

En esta sección abordaremos el siguiente tipo de problemas: dado un lenguaje L , diseñar un autómata finito M que acepte o reconozca a L , es decir, tal que $L(M) = L$. Más adelante se demostrará en toda su generalidad que, si L es regular, estos problemas siempre tienen solución. Para encontrar un autómata M tal que $L(M) = L$ hay que tener en cuenta que esta igualdad es estricta y, por tanto, se deben cumplir las dos siguientes condiciones:

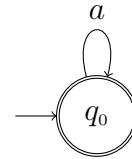
1. Si una cadena u es aceptada por M , entonces u debe pertenecer a L .
2. Recíprocamente, si $u \in L$, entonces u debe ser aceptada por M .

Un estado q en un autómata M se llama *estado limbo* si q no es un estado de aceptación y desde q no salen trayectorias que conduzcan a estados de aceptación. Puesto que los estados limbo no hacen parte de las trayectorias de aceptación se suprimen, por conveniencia, al presentar un autómata. No obstante, los estados limbo (si los hay) hacen parte integrante del autómata y solo se omiten para simplificar los grafos.

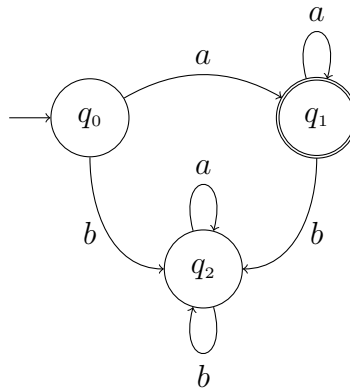
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^* = \{\lambda, a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



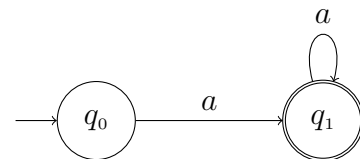
En la versión simplificada omitimos el estado limbo q_1 :



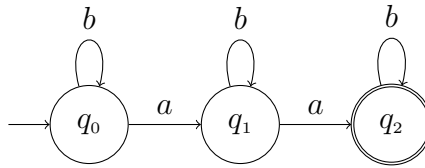
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^+ = \{a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



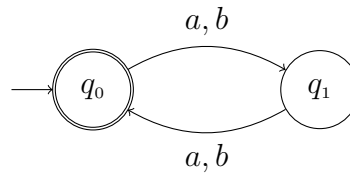
En la versión simplificada omitimos el estado limbo q_2 :



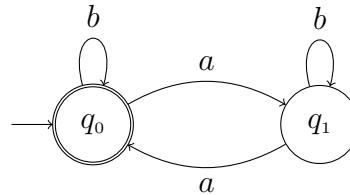
Ejemplo $\Sigma = \{a, b\}$. $L =$ lenguaje de las cadenas que contienen exactamente dos a 's $= b^*ab^*ab^*$. AFD M tal que $L(M) = L$, omitiendo el estado limbo:

**Ejemplo**

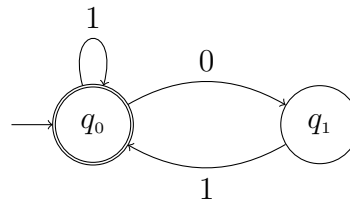
$\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que tienen un número par de símbolos (cadenas de longitud par ≥ 0). AFD M tal que $L(M) = L$:

**Ejemplo**

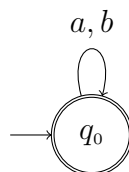
$\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que contienen un número par de a 'es. AFD M tal que $L(M) = L$:

**Ejemplo**

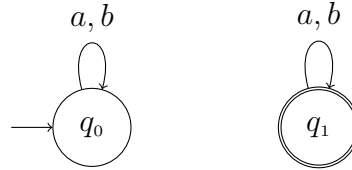
Sean $\Sigma = \{0, 1\}$ y $L = (1 \cup 01)^*$. Construimos un AFD de tal forma que el estado inicial q_0 sea el único estado de aceptación y en él confluyan las trayectorias 1 y 01. Omitiendo el estado limbo obtenemos el siguiente autómata:

**Ejemplo**

¿Cuál es el autómata más sencillo que se puede construir con el alfabeto de entrada $\Sigma = \{a, b\}$? Tal autómata M tiene un único estado que debe ser tanto estado inicial como estado de aceptación. Se tiene entonces que M acepta todas las cadenas, es decir, $L(M) = (a \cup b)^*$



Por otro lado, podemos construir un autómata M' que no acepte ninguna cadena, es decir, tal que $L(M') = \emptyset$. El estado inicial de M' no puede ser estado de aceptación (ya que aceptaría λ), pero como todo autómata debe tener por lo menos un estado de aceptación, M' debe tener dos estados:



Ejercicios de la sección 2.5

- ① Sea $\Sigma = \{a, b\}$. Diseñar AFD (autómatas finitos deterministas) que acepten los siguientes lenguajes:

- | | |
|------------------------|-------------------------|
| (i) a^*b^* . | (ii) $a^* \cup b^*$. |
| (iii) $(ab)^*$. | (iv) $(ab)^+$. |
| (v) $ab^* \cup b^*a$. | (vi) $a(a \cup ab)^*$. |

(vii) a^+b^*a . Un AFD que acepte este lenguaje requiere como mínimo 5 estados más un estado limbo (6 estados en total).

(viii) $a^*b \cup b^*a$. Un AFD que acepte este lenguaje requiere como mínimo 6 estados más un estado limbo (7 estados en total).

(ix) $b^*(ab \cup ba)$.

(x) $b^*(ab \cup ba)a^+$.

- ② Sea $\Sigma = \{a, b\}$.

(i) Diseñar un AFD que acepte el lenguaje de todas las cadenas que contienen un número par de a 'es y un número par de b 'es. Se entiende que par incluye a 0. Ayuda: utilizar 4 estados.

(ii) Para cada combinación de las condiciones “par” e “impar” y de las conectivas “o” e “y”, diseñar un AFD que acepte el lenguaje L definido por

$L =$ lenguaje de las cadenas con un número par/impar de a 'es
y/o un número par/impar de b 'es.

Ayuda: utilizar el autómata de 4 estados diseñado en la parte (i), modificando adecuadamente el conjunto de estados de aceptación.

- ③ Sea $\Sigma = \{0, 1\}$. Diseñar AFD (autómatas finitos deterministas) que acepten los siguientes lenguajes:

- (i) El lenguaje de todas las cadenas que terminan en 01.
 - (ii) El lenguaje de todas las cadenas que tienen un número par ≥ 2 de unos.
 - (iii) El lenguaje de todas las cadenas con longitud ≥ 4 .
 - (iv) El lenguaje de todas las cadenas que contienen por lo menos dos unos consecutivos.
 - (v) El lenguaje de todas las cadenas que tienen un número par de ceros pero no tienen dos ceros consecutivos.
 - (vi) $(01 \cup 101)^*$.
 - (vii) $1^+(10 \cup 01^+)^*$.
- ④ Sea $\Sigma = \{a, b, c\}$. Diseñar AFD (autómatas finitos deterministas) que acepten los siguientes lenguajes:
- (i) $a^*b^*c^*$.
 - (ii) $a^+b^* \cup ac^* \cup b^*ca^*$.
 - (iii) $a^*(ba \cup ca)^+$.
- ⑤ Sea $L = \{a^{2i}b^{3j} : i, j \geq 0\}$ definido sobre el alfabeto $\Sigma = \{a, b\}$. Encontrar una expresión regular para L y un AFD M tal que $L(M) = L$.

2.6. Autómatas finitos no-deterministas (AFN)

En el modelo determinista una entrada u se procesa de manera única y el estado en el cual finaliza tal procesamiento determina si u es o no aceptada. En contraste, en el modelo no-determinista es posible que una entrada se pueda procesar de varias formas o que haya procesamientos abortados.

Concretamente, un AFN (*Autómata Finito No-determinista*) $M = (\Sigma, Q, q_0, F, \Delta)$ posee cinco componentes, los primeros cuatro con el mismo significado que en el caso determinista:

1. Un alfabeto Σ , llamado alfabeto de entrada o alfabeto de cinta. Todas las cadenas que procesa M pertenecen a Σ^* .
2. $Q = \{q_0, q_1, \dots, q_n\}$, el conjunto (finito) de los estados internos de la unidad de control.
3. $q_0 \in Q$, estado inicial.
4. $F \subseteq Q$, conjunto de estados finales o de aceptación. F debe ser distinto de \emptyset ; es decir, debe haber por lo menos un estado de aceptación.

5. La función Δ de transición del autómata

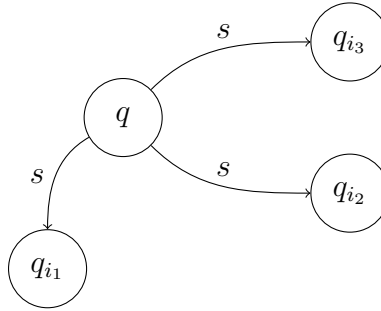
$$\begin{aligned} \Delta : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, s) &\longmapsto \Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\} \end{aligned}$$

donde $\wp(Q)$ es el conjunto de subconjunto de Q .

El significado de $\Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}$ es el siguiente: estando en el estado q , en presencia del símbolo s , la unidad de control puede pasar (aleatoriamente) a uno cualquiera de los estados $q_{i_1}, q_{i_2}, \dots, q_{i_k}$, después de lo cual se desplaza a la derecha.

Puede suceder que $\Delta(q, s) = \emptyset$, lo cual significa que, si durante la lectura de una cadena de entrada u , la cabeza lectora de M ingresa al estado q leyendo sobre la cinta el símbolo s , el procesamiento se aborta.

La noción de digrafo etiquetado para un AFN se define de manera análoga al caso AFD, pero puede suceder que desde un mismo nodo (estado) salgan dos o más arcos con la misma etiqueta:



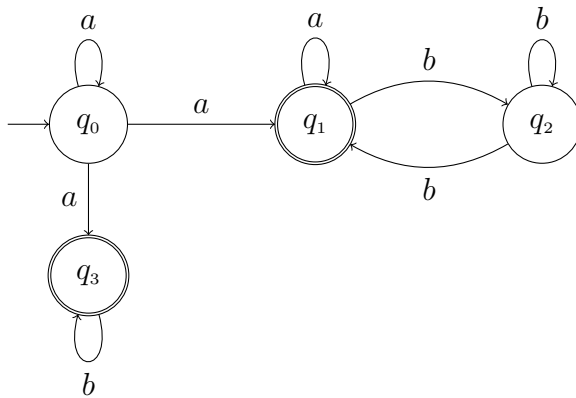
Un AFN M puede procesar una cadena de entrada $u \in \Sigma^*$ de varias maneras. Sobre el grafo del autómata, esto significa que pueden existir varias trayectorias, desde el estado q_0 , etiquetadas con los símbolos de u .

Igual que en el caso determinista, $L(M)$ denota el lenguaje aceptado o reconocido por M . La siguiente es la noción de aceptación para autómatas no-deterministas:

$$L(M) = \{u \in \Sigma^* : \text{existe por lo menos un procesamiento completo de } u \text{ desde } q_0, \text{ que termina en un estado } q \in F\}$$

Es decir, para que una cadena u sea aceptada, debe existir algún procesamiento en el que u sea procesada completamente y que finalice estando M en un estado de aceptación. En términos del grafo del AFN, una cadena de entrada u es aceptada si existe por lo menos una trayectoria etiquetada con los símbolos de u , desde el estado q_0 hasta un estado de aceptación.

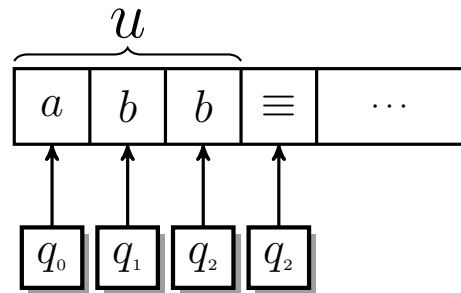
Ejemplo Sea M el siguiente AFN:



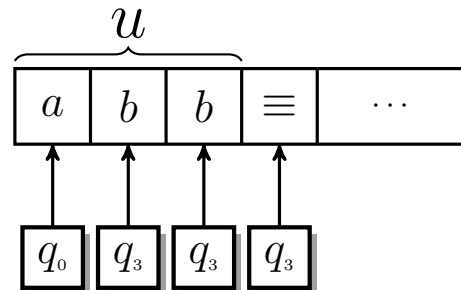
Δ	a	b
q_0	$\{q_0, q_1, q_3\}$	\emptyset
q_1	$\{q_1\}$	$\{q_2\}$
q_2	\emptyset	$\{q_1, q_2\}$
q_3	\emptyset	$\{q_3\}$

Para la cadena de entrada $u = abb$, existen procesamientos que conducen al rechazo, procesamientos abortados y procesamientos que terminan en estados de aceptación. Según la definición de lenguaje aceptado, $u \in L(M)$.

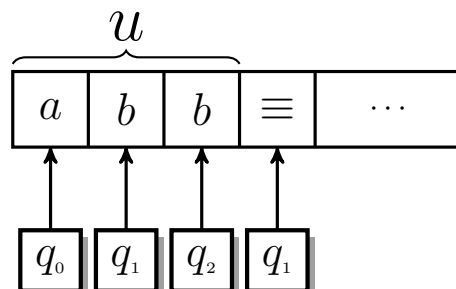
Procesamiento de rechazo:



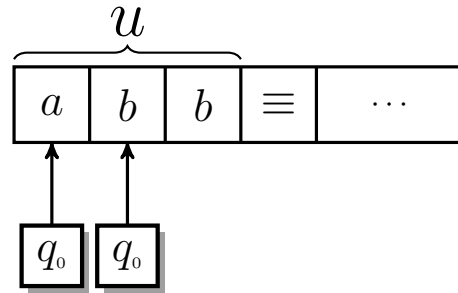
Procesamiento de aceptación:



Otro procesamiento de aceptación:

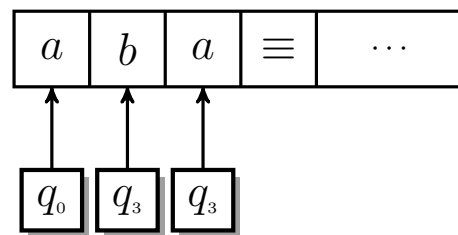


Procesamiento abortado:



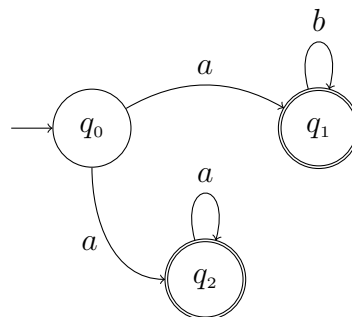
El grafo de M nos permite ver que la cadena $aabbaa$ es aceptada mientras que $aabaa$ es rechazada. Todas las cadenas que comienzan con b son rechazadas. También es rechazada la cadena aba ; uno de sus procesamientos posibles es el siguiente:

Procesamiento abortado:



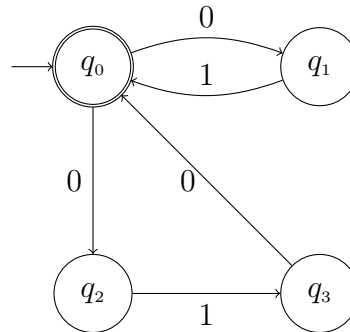
A pesar de que el procesamiento anterior termina en el estado de aceptación q_3 , la entrada no es aceptada porque no se consume completamente.

Ejemplo Considérese el lenguaje $L = ab^* \cup a^+$ sobre el alfabeto $\Sigma = \{a, b\}$. El siguiente AFN M satisface $L(M) = L$.

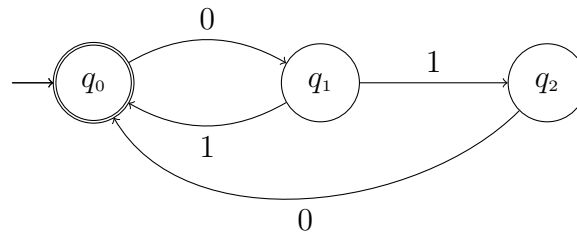


Un AFD que acepte a L requiere como mínimo cuatro estados más un estado limbo (cinco estados en total).

Ejemplo $\Sigma = \{0, 1\}$, $L = (01 \cup 010)^*$. El siguiente AFN acepta a L .



Otro AFN que acepta el mismo lenguaje y que tiene sólo tres estados es el siguiente:



Ejercicios de la sección 2.6

① Sea $\Sigma = \{a, b\}$. Diseñar AFN (autómatas finitos no-deterministas) que acepten los siguientes lenguajes:

- (i) $a(a \cup ab)^*$.
- (ii) a^+b^*a .
- (iii) $a^*b \cup b^*a$.
- (iv) $b^*(ab \cup ba)^*$.
- (v) $b^*(ab \cup ba)^*a^*$.

② Sea $\Sigma = \{0, 1\}$. Diseñar AFN (autómatas finitos no-deterministas) que acepten los siguientes lenguajes:

- (i) $(01 \cup 001)^*$.
- (ii) $(01^*0 \cup 10^*)^*$.
- (iii) $1^*01 \cup 10^*1$.

2.7. Equivalencia computacional entre los AFD y los AFN

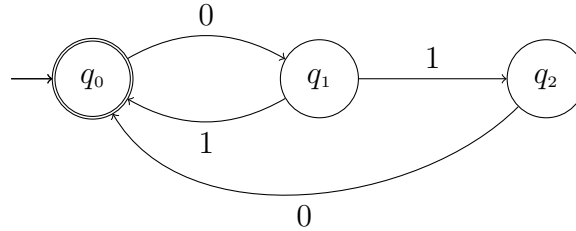
En esta sección se mostrará que los modelos AFD y AFN son computacionalmente equivalentes en el sentido de que aceptan los mismos lenguajes. En primer lugar, es fácil ver que un AFD $M = (\Sigma, Q, q_0, F, \delta)$ puede ser considerado como un AFN $M' = (\Sigma, Q, q_0, F, \Delta)$ definiendo $\Delta(q, a) = \{\delta(q, a)\}$ para cada $q \in Q$ y cada $a \in \Sigma$. Para la afirmación recíproca tenemos el siguiente teorema.

2.7.1 Teorema. Dado un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ se puede construir un AFD M' equivalente a M , es decir, tal que $L(M) = L(M')$.

La demostración detallada de este teorema se presentará más adelante. La idea de la demostración consiste en considerar cada conjunto de estados $\Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\}$ que aparezca en la tabla de la función Δ del autómata no-determinista M como un *único* estado del nuevo autómata determinista M' . La tabla de Δ se completa hasta que no aparezcan nuevas combinaciones de estados. Los estados de aceptación del nuevo autómata son los conjuntos de estados en los que aparece *por lo menos* un estado de aceptación del autómata original.

Se obtiene así un procedimiento algorítmico que convierte un AFN dado en un AFD equivalente. En los siguientes dos ejemplos ilustramos el procedimiento.

Ejemplo El siguiente AFN M , presentado en el último ejemplo de la sección 2.6, acepta el lenguaje $L = (01 \cup 010)^*$ sobre $\Sigma = \{0, 1\}$.

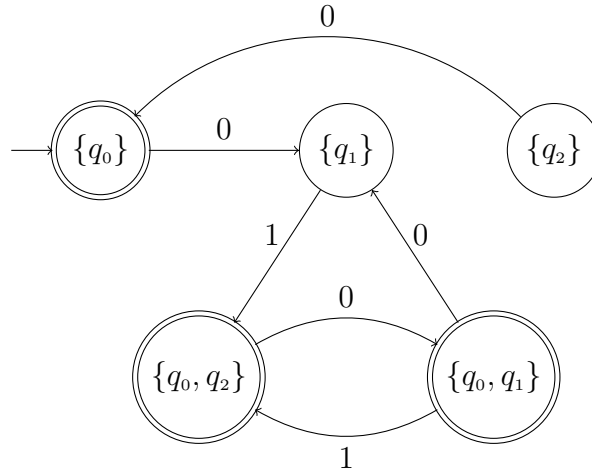


La tabla original de la función de transición Δ de M contiene la combinación $\{q_0, q_2\}$ que adquiere el estatus de nuevo estado en M' . Al extender la tabla de Δ aparece también el nuevo estado $\{q_0, q_1\}$:

Δ	0	1
q_0	$\{q_1\}$	\emptyset
q_1	\emptyset	$\{q_0, q_2\}$
q_2	$\{q_0\}$	\emptyset
$\{q_0, q_2\}$	$\{q_0, q_1\}$	\emptyset
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$

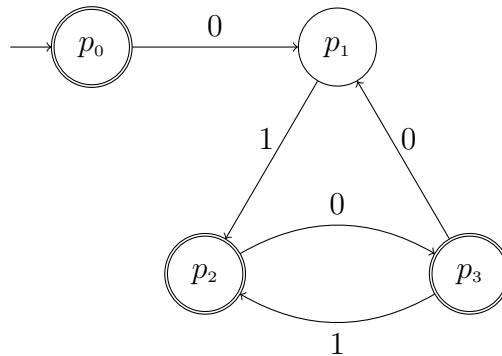
La tabla original de la función Δ y su extensión.

La tabla de la derecha corresponde a un AFD ya que cada combinación de estados de M se considera ahora como un único estado en M' . El grafo del nuevo autómata M' es:

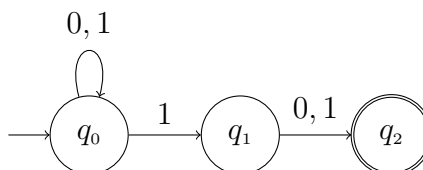


Los estados de aceptación son aquéllos en los que aparezca q_0 ya que q_0 es el único estado de aceptación del autómata original.

Puesto que el estado $\{q_2\}$ en el nuevo autómata no interviene en la aceptación de cadenas, es inútil y puede ser eliminado. El autómata M' se puede simplificar en la siguiente forma:



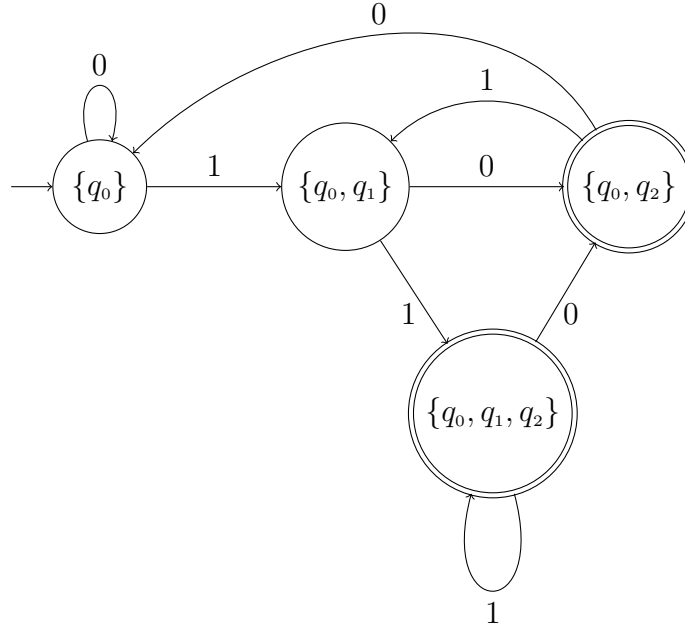
Ejemplo Sean $\Sigma = \{0, 1\}$ y L_2 el lenguaje de todas las cadenas de longitud ≥ 2 en las que el segundo símbolo, de derecha a izquierda es un 1. Una expresión regular para este lenguaje es $(0 \cup 1)^* 1 (0 \cup 1)$ y es fácil diseñar un AFN M que acepte a L_2 :



Por simple inspección no es tan fácil diseñar un AFD que acepte a L_2 , pero aplicando el procedimiento de conversión podemos encontrar uno. Hacemos la tabla de la función de transición Δ y la extendemos con las nuevas combinaciones de estados.

Δ	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	\emptyset	\emptyset
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

En el nuevo autómata los estados $\{q_1\}$ y $\{q_2\}$ resultan inútiles; una vez eliminados obtenemos el siguiente AFD equivalente al AFN M :



Para la demostración del Teorema 2.7.1, conviene extender la definición de la función de transición, tanto de los autómatas deterministas como de los no-deterministas.

2.7.2 Definición. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD. La función de transición δ , $\delta : Q \times \Sigma \rightarrow Q$ se extiende a una función $\hat{\delta} : Q \times \Sigma^* \rightarrow Q$ por medio de la siguiente definición recursiva:

$$\begin{cases} \hat{\delta}(q, \lambda) = q, & q \in Q, \\ \hat{\delta}(q, a) = \delta(q, a), & q \in Q, a \in \Sigma, \\ \hat{\delta}(q, ua) = \delta(\hat{\delta}(q, u), a), & q \in Q, a \in \Sigma, u \in \Sigma^*. \end{cases}$$

Según esta definición, para una cadena cualquiera $u \in \Sigma^*$, $\widehat{\delta}(q, u)$ es el estado en el que el autómata termina el procesamiento de u a partir del estado q . En particular, $\widehat{\delta}(q_0, u)$ es el estado en el que el autómata termina el procesamiento de la entrada u desde el estado inicial q_0 . Por lo tanto, podemos describir el lenguaje aceptado por M de la siguiente forma:

$$L(M) = \{u \in \Sigma^* : \widehat{\delta}(q_0, u) \in F\}.$$

La función de transición extendida $\widehat{\delta}$ permite definir de manera precisa las nociones de estado accesible y estado inaccesible en un AFD.

2.7.3 Definición. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD. Un estado $q \in Q$ es accesible si existe una cadena de entrada $u \in \Sigma^*$ tal que $\widehat{\delta}(q_0, u) = q$.

Un estado inaccesible q (o sea, no accesible) es completamente inútil ya que la unidad de control del autómata nunca ingresará a q al procesar una cadena cualquiera desde el estado inicial q_0 . Los estados inaccesibles se pueden eliminar del autómata sin alterar el lenguaje aceptado.

No se debe confundir la noción de estado inaccesible con la de estado limbo. Los estados limbo son necesarios para rechazar cadenas en un AFD mientras que los estados inaccesibles son inútiles.

Notación. La función extendida $\widehat{\delta}(q, u)$ se puede escribir simplemente $\delta(q, u)$. Esto no crea confusión ni ambigüedad.

2.7.4 Definición. Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. La función de transición Δ , $\Delta : Q \times \Sigma \rightarrow \wp(Q)$ se extiende inicialmente a conjuntos de estados. Para $a \in \Sigma$ y $S \subseteq Q$ se define

$$\Delta(S, a) := \bigcup_{q \in S} \Delta(q, a).$$

Se tendría $\Delta(S, a) = \emptyset$ en el caso en que $\Delta(q, a) = \emptyset$ para todo $q \in S$.

Luego se extiende Δ a una función $\widehat{\Delta} : Q \times \Sigma^* \rightarrow \wp(Q)$, de manera similar a como se hace para los AFD. Recursivamente,

$$\begin{cases} \widehat{\Delta}(q, \lambda) = \{q\}, & q \in Q, \\ \widehat{\Delta}(q, a) = \Delta(q, a), & q \in Q, a \in \Sigma, \\ \widehat{\Delta}(q, ua) = \Delta(\widehat{\Delta}(q, u), a) = \bigcup_{p \in \widehat{\Delta}(q, u)} \Delta(p, a), & q \in Q, a \in \Sigma, u \in \Sigma^*. \end{cases}$$

Según esta definición, para una cadena cualquiera $u \in \Sigma^*$, $\widehat{\Delta}(q, u)$ es el conjunto de los posibles estados en los que terminan los procesamientos *completos* de u a partir del estado q . En particular, para una cadena de entrada $u \in \Sigma^*$, $\widehat{\Delta}(q_0, u)$ es el conjunto de los posibles estados en los que terminan los procesamientos *completos* de u desde el estado inicial q_0 . Si todos los procesamientos de u se abortan en algún momento, se tendría $\widehat{\Delta}(q_0, u) = \emptyset$.

Usando la función extendida $\widehat{\Delta}$, el lenguaje aceptado por M se puede describir de la siguiente forma:

$$L(M) = \{u \in \Sigma^* : \widehat{\Delta}(q_0, u) \text{ contiene por lo menos un estado de aceptación}\}.$$

Notación. La función extendida $\widehat{\Delta}(q, u)$ se puede escribir simplemente $\Delta(q, u)$. Esto no crea confusión ni ambigüedad.

Demostración del Teorema 2.7.1:

Dado el AFN $M = (\Sigma, Q, q_0, F, \Delta)$, construimos el AFD M' así:

$$M' = (\Sigma, \wp(Q), \{q_0\}, F', \delta)$$

donde

$$\begin{aligned} \delta : \wp(Q) \times \Sigma &\longrightarrow \wp(Q) \\ (S, a) &\longmapsto \delta(S, a) := \Delta(S, a). \end{aligned}$$

$$F' = \{S \subseteq Q : S \text{ contiene por lo menos un estado de aceptación de } M\}.$$

Razonando por recursión sobre u , se demostrará para toda cadena $u \in \Sigma^*$, $\delta(\{q_0\}, u) = \Delta(q_0, u)$. Para $u = \lambda$, claramente se tiene $\delta(\{q_0\}, \lambda) = \{q_0\} = \Delta(q_0, \lambda)$. Para $u = a$, $a \in \Sigma$, se tiene

$$\delta(\{q_0\}, a) = \Delta(\{q_0\}, a) = \Delta(q_0, a).$$

Supóngase (hipótesis recursiva) que $\delta(\{q_0\}, u) = \Delta(q_0, u)$, y que $a \in \Sigma$. Entonces

$$\begin{aligned} \delta(\{q_0\}, ua) &= \delta(\delta(\{q_0\}, u), a) && \text{(definición de la extensión de } \delta) \\ &= \delta(\Delta(q_0, u), a) && \text{(hipótesis recursiva)} \\ &= \Delta(\Delta(q_0, u), a) && \text{(definición de } \delta) \\ &= \Delta(q_0, ua) && \text{(definición de la extensión de } \Delta). \end{aligned}$$

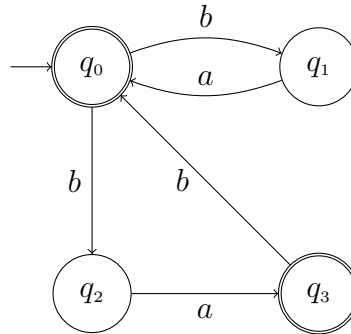
Finalmente podemos demostrar que $L(M') = L(M)$:

$$\begin{aligned} u \in L(M') &\iff \delta(\{q_0\}, u) \in F' \\ &\iff \Delta(q_0, u) \in F' \\ &\iff \Delta(q_0, u) \text{ contiene un estado de aceptación de } M \\ &\iff u \in L(M). \quad \square \end{aligned}$$

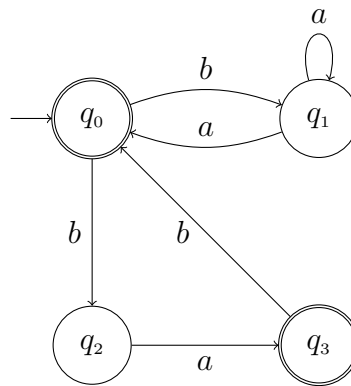
Ejercicios de la sección 2.7

- ① Utilizando el procedimiento de conversión presentado en esta sección, encontrar AFD equivalentes a los siguientes AFN:

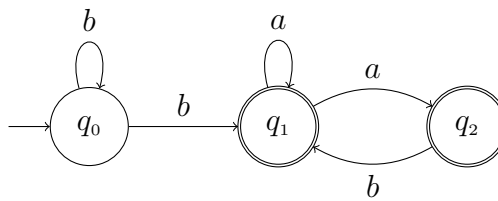
(i)



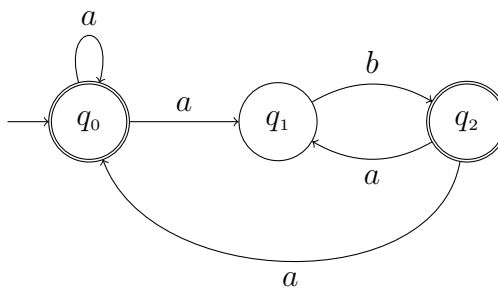
(ii)



(iii)



(iv)



- ② Sean $\Sigma = \{0, 1\}$ y L_3 el lenguaje de todas las cadenas de longitud ≥ 3 en las que el tercer símbolo, de derecha a izquierda es un 1. Diseñar un AFN con cuatro estados que acepte a L_3 y aplicar luego el procedimiento de conversión para encontrar un AFD equivalente.

- ③ Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD. Demostrar por recursión sobre cadenas que la extensión de δ satisface

$$\delta(q, uv) = \delta(\delta(q, u), v),$$

para todo estado $q \in Q$, y todas las cadenas $u, v \in \Sigma^*$.

- ④ Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. Demostrar por recursión sobre cadenas que la extensión de Δ satisface

$$\Delta(q, uv) = \Delta(\Delta(q, u), v),$$

para todo estado $q \in Q$, y todas las cadenas $u, v \in \Sigma^*$.

2.8. Autómatas con transiciones λ (AFN- λ)

Un *autómata finito con transiciones λ* (AFN- λ) es un autómata no-determinista $M = (\Sigma, Q, q_0, F, \Delta)$ en el que la función de transición está definida como:

$$\Delta : Q \times (\Sigma \cup \{\lambda\}) \rightarrow \wp(Q).$$

Δ permite, además de las instrucciones no-deterministas usuales, transiciones de la forma $\Delta(q, \lambda) = \{q_{i_1}, \dots, q_{i_k}\}$, llamadas *transiciones λ* , *transiciones nulas* o *transiciones espontáneas*. Sobre la cinta de entrada, el significado computacional de la instrucción

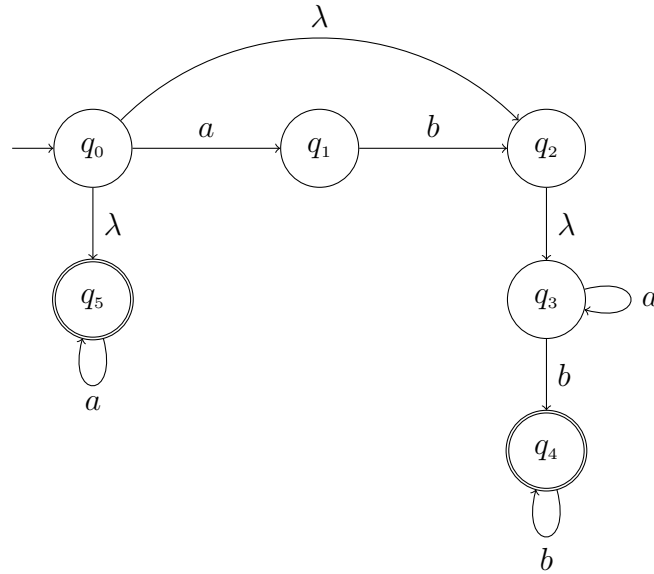
$$\Delta(q, \lambda) = \{q_{i_1}, \dots, q_{i_k}\}$$

es el siguiente: estando en el estado q , el autómata puede cambiar aleatoriamente a uno cualquiera de los estados q_{i_1}, \dots, q_{i_k} , independientemente del símbolo leído y sin mover la unidad de control a la derecha. Dicho de otra manera, las transiciones λ permiten a la unidad de control del autómata cambiar internamente de estado sin procesar o “consumir” el símbolo leído sobre la cinta.

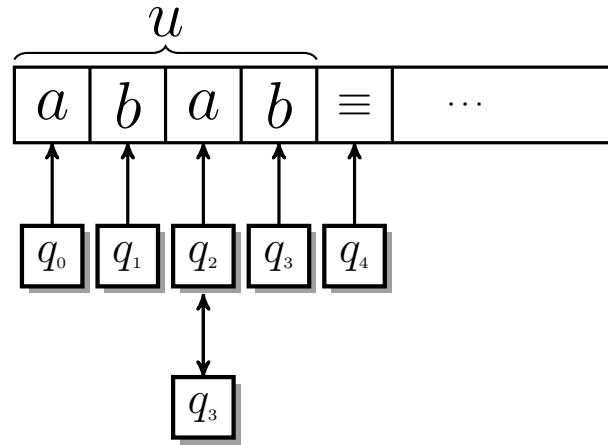
Como sucede en el caso AFN, una cadena de entrada $u \in \Sigma^*$ es aceptada si existe por lo menos un procesamiento completo de u , desde q_0 , que termina en un estado de aceptación. En el grafo del autómata, las transiciones λ dan lugar a arcos con etiquetas λ . Una cadena de entrada u es aceptada por un AFN- λ si existe por lo menos una trayectoria, desde el estado q_0 , cuyas etiquetas son exactamente los símbolos de u , intercalados con cero, uno o más λ s.

En los autómatas AFN- λ , al igual que en los AFN, puede haber múltiples procesamientos para una misma cadena de entrada, así como procesamientos abortados.

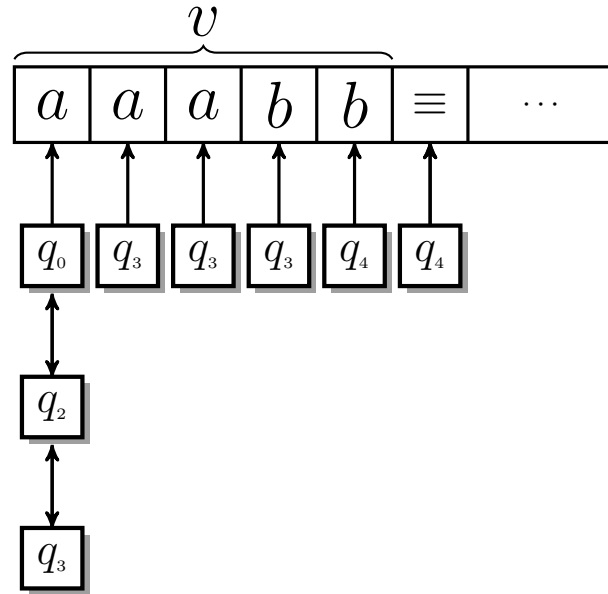
Ejemplo Consideremos el siguiente AFN- λ , M :



La entrada $u = abab$ es aceptada siguiendo sobre el grafo de M la trayectoria $ab\lambda ab$. Si miramos este procesamiento sobre la cinta de entrada, M utiliza una transición λ para cambiar internamente del estado q_2 al estado q_3 , sin desplazar la cabeza lectora a la derecha.



La entrada $v = aaabb$ es aceptada siguiendo sobre el grafo de M la trayectoria $\lambda\lambda aaabb$. Sobre la cinta de entrada, este procesamiento de v corresponde a dos transiciones espontáneas consecutivas: de q_0 a q_2 y luego de q_2 a q_3 . Al utilizar estas transiciones λ , la cabeza lectora no se desplaza a la derecha.



También puede observarse sobre el grafo de M que para la cadena $abbb$ hay dos trayectorias de aceptación diferentes, a saber, $ab\lambda bb$ y $\lambda\lambda abbb$.

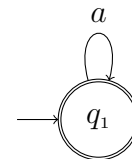
Los AFN- λ permiten aún más libertad en el diseño de autómatas, especialmente cuando hay numerosas uniones y concatenaciones.

Ejemplo Diseñar AFN- λ que acepten los siguientes lenguajes:

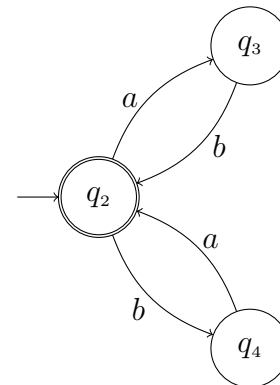
- (1) $a^* \cup (ab \cup ba)^* \cup b^+$.
- (2) $a^*(ab \cup ba)^*b^+$.

Las expresiones regulares para estos dos lenguajes se pueden obtener a partir de las tres sub-expresiones a^* , $(ab \cup ba)^*$ y b^+ , para las cuales es fácil diseñar autómatas.

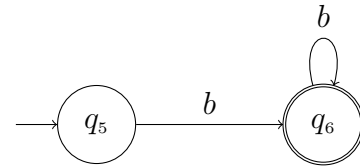
Autómata que acepta a^* :



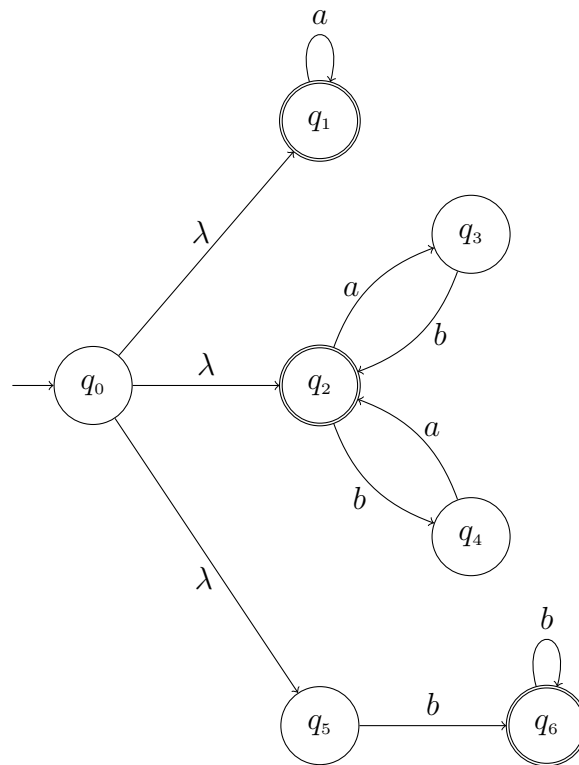
Autómata que acepta $(ab \cup ba)^*$:



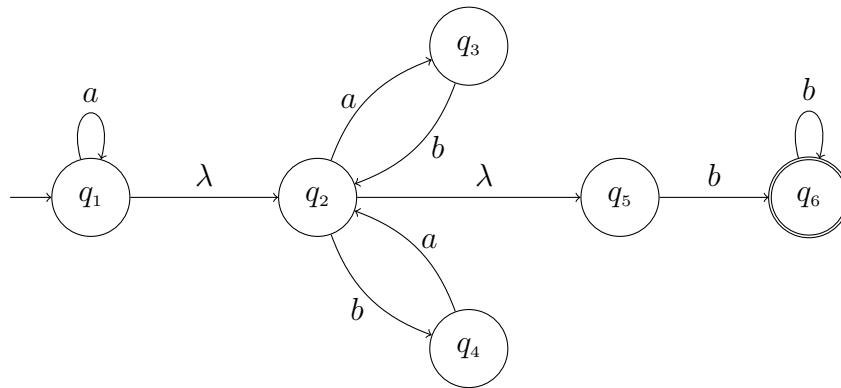
Autómata que acepta b^+ :



- (1) Para aceptar $a^* \cup (ab \cup ba)^* \cup b^+$ utilizamos un nuevo estado inicial q_0 y tres transiciones λ que lo conectan con los tres autómatas anteriores. Los estados de aceptación se mantienen. Esta manera de conectar autómatas la llamaremos “conexión en paralelo”. Desde el estado inicial q_0 el autómata puede proseguir el procesamiento de una entrada por tres caminos diferentes para aceptar a^* , $(ab \cup ba)^*$ y b^+ , respectivamente:

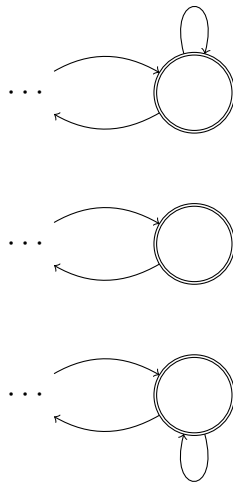


- (2) Para aceptar $a^*(ab \cup ba)^*b^+$ conectamos linealmente los tres autómatas mediante transiciones λ . Esta manera de conectar autómatas la llamaremos “conexión en serie”. Nótese que hay un único estado de aceptación correspondiente al bucle final b^+ . Los estados q_1 y q_2 no pueden ser de aceptación en el nuevo autómata:

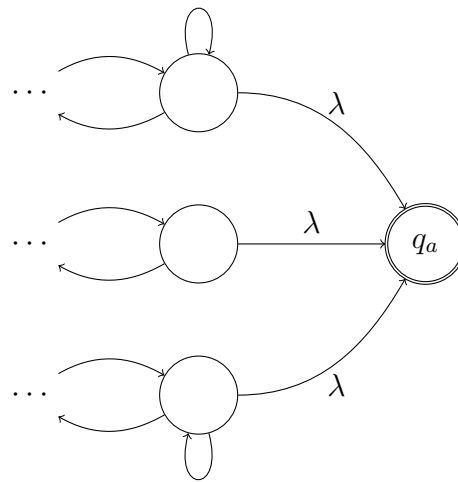


En la sección 2.12 veremos que los procedimientos de conexión en paralelo y en serie se pueden sistematizar para diseñar algorítmicamente un AFN- λ que acepte el lenguaje representado por una expresión regular dada.

Estado de aceptación único. Un autómata cualquiera M siempre se puede modificar, sin alterar el lenguaje aceptado, de tal manera que tenga un único estado de aceptación. Esto se consigue añadiendo un nuevo estado q_a , que será el único estado de aceptación, y trazando transiciones λ entre todos y cada uno de los estados de aceptación originales y q_a . Todas las demás transiciones de M se mantienen sin alteraciones. El siguiente diagrama ilustra la situación cuando el autómata original tiene tres estados de aceptación.



Tres estados de aceptación

Único estado de aceptación q_a

Que un autómata tenga un único estado de aceptación será un detalle relevante en secciones posteriores.

Ejercicios de la sección 2.8

- ① Sea $\Sigma = \{a, b\}$. Diseñar AFN- λ (autómatas finitos no-deterministas con transiciones λ) que acepten los siguientes lenguajes:
- (i) $(ab \cup b)^* ab^* a^*$.
 - (ii) $ab^* \cup ba^* \cup b(ab \cup ba)^*$.
 - (iii) $(a \cup aba)^* b^* (ab \cup ba)^* a^*$.
- ② Sea $\Sigma = \{0, 1\}$. Diseñar AFN- λ (autómatas finitos no-deterministas con transiciones λ) que acepten los siguientes lenguajes:
- (i) $(1 \cup 01 \cup 001)^* 0^* 1^* 0^+$.
 - (ii) $0^+ 1(010)^* (01 \cup 10)^* 1^+$.
 - (iii) $(101)^* \cup 1^* (1 \cup 10)^* 0^+ (01 \cup 10)^*$.

2.9. Equivalencia computacional entre los AFN- λ y los AFN

En esta sección se mostrará que el modelo AFN- λ es computacionalmente equivalente al modelo AFN. En primer lugar, un AFN puede ser considerado como un AFN- λ en el que, simplemente, hay cero transiciones λ . Recíprocamente, vamos a presentar un procedimiento algorítmico de conversión de un AFN- λ en un AFN que consiste en eliminar las transiciones λ añadiendo transiciones que las simulen, sin alterar el lenguaje aceptado. El procedimiento se basa en la noción de λ -clausura de un estado. Dado un AFN- λ M y un estado q de M , la λ -clausura de q , notada $\lambda[q]$, es el conjunto de estados de M a los que se puede llegar desde q por 0, 1 o más transiciones λ . Según esta definición, un estado q siempre pertenece a su λ -clausura, es decir, $q \in \lambda[q]$. Si desde q no hay transiciones λ , se tendrá $\lambda[q] = \{q\}$. La λ -clausura de un conjunto de estados $\{q_1, \dots, q_k\}$ se define como la unión de las λ -clausuras, esto es,

$$\lambda[\{q_1, \dots, q_k\}] := \lambda[q_1] \cup \dots \cup \lambda[q_k].$$

También se define $\lambda[\emptyset] := \emptyset$.

2.9.1 Teorema. Dado un AFN- λ $M = (\Sigma, Q, q_0, F, \Delta)$, se puede construir un AFN M' (sin transiciones λ) equivalente a M , es decir, tal que $L(M) = L(M')$.

Bosquejo de la demostración. Se construye $M' = (\Sigma, Q, q_0, F', \Delta')$ a partir de M manteniendo el conjunto de estados Q y el estado inicial q_0 . M' tiene una nueva función de transición Δ' y un nuevo conjunto de estados de aceptación F' definidos por:

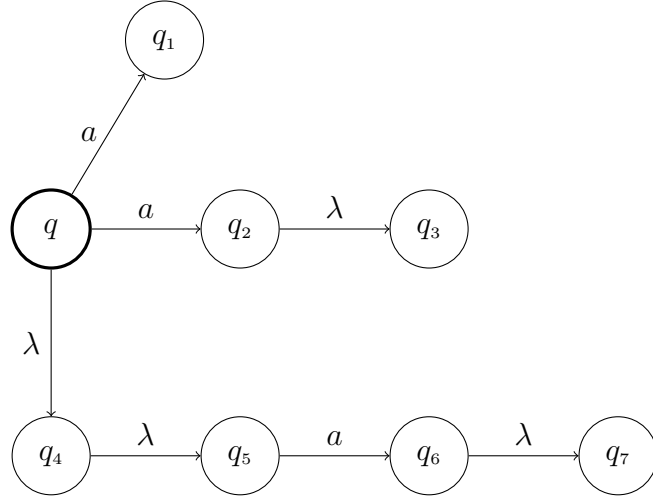
$$\begin{aligned} \Delta' : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, a) &\longmapsto \Delta'(q, a) := \lambda[\Delta(\lambda[q], a)]. \end{aligned}$$

$$F' = \{q \in Q : \lambda[q] \text{ contiene al menos un estado de aceptación}\}.$$

Es decir, los estados de aceptación de M' incluyen los estados de aceptación de M y aquellos estados desde los cuales se puede llegar a un estado de aceptación por medio de una o más transiciones λ . \square

La construcción de M' a partir de M es puramente algorítmica. El significado de la fórmula $\Delta'(q, a) = \lambda[\Delta(\lambda[q], a)]$ se puede apreciar considerando el grafo que aparece a continuación, que es una porción de un AFN- λ M .

Porción de M :



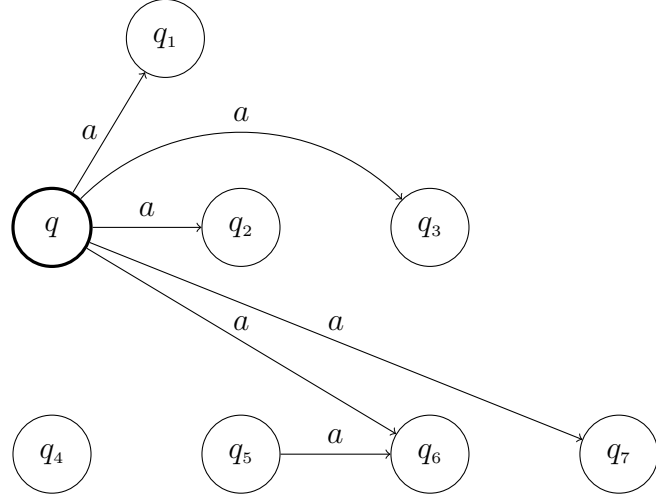
Por simple inspección observamos que, una vez procesada una a , el autómata puede pasar desde el estado q a uno de los siguientes estados: q_1, q_2, q_3, q_6, q_7 . Para obtener esta lista de estados se tienen en cuenta todas las transiciones λ que preceden o prosiguen el procesamiento del símbolo a desde el estado q .

Al aplicar la fórmula $\Delta'(q, a) = \lambda[\Delta(\lambda[q], a)]$ se llega a esta misma lista de estados. En efecto, por la definición de λ -clausura se tiene que $\lambda[q] = \{q, q_4, q_5\}$, y se obtiene que

$$\begin{aligned} \Delta'(q, a) &= \lambda[\Delta(\lambda[q], a)] = \lambda[\Delta(\{q, q_4, q_5\}, a)] \\ &= \lambda[\{q_1, q_2, q_6\}] = \lambda[q_1] \cup \lambda[q_2] \cup \lambda[q_6] \\ &= \{q_1\} \cup \{q_2, q_3\} \cup \{q_6, q_7\} = \{q_1, q_2, q_3, q_6, q_7\}. \end{aligned}$$

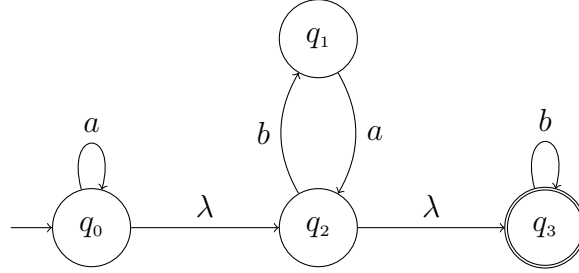
La porción correspondiente del grafo de M' se exhibe en la siguiente gráfica. De esta forma M' simula, sin transiciones λ , todas las transiciones λ de M añadiendo nuevas transiciones con etiqueta a .

Porción de M' :



Ejemplo

Utilizar la construcción del Teorema 2.9.1 para encontrar un AFN equivalente al siguiente AFN- λ M .



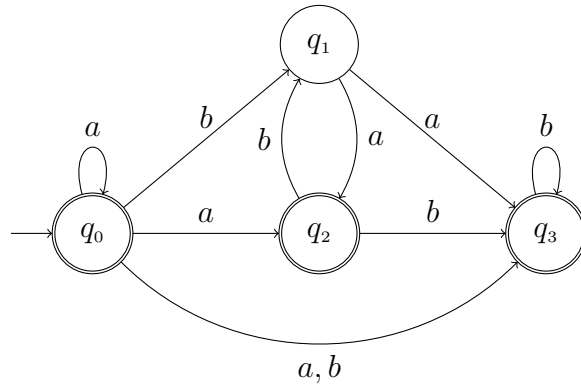
Las λ -clausuras de los estados vienen dadas por:

$$\begin{aligned}\lambda[q_0] &= \{q_0, q_2, q_3\}. \\ \lambda[q_1] &= \{q_1\}. \\ \lambda[q_2] &= \{q_2, q_3\}. \\ \lambda[q_3] &= \{q_3\}.\end{aligned}$$

La función de transición $\Delta' : Q \times \{a, b\} \rightarrow \mathcal{P}(\{q_0, q_1, q_2, q_3\})$ es:

$$\begin{aligned}\Delta'(q_0, a) &= \lambda[\Delta(\lambda[q_0], a)] = \lambda[\Delta(\{q_0, q_2, q_3\}, a)] = \lambda[\{q_0\}] = \{q_0, q_2, q_3\}. \\ \Delta'(q_0, b) &= \lambda[\Delta(\lambda[q_0], b)] = \lambda[\Delta(\{q_0, q_2, q_3\}, b)] = \lambda[\{q_1, q_3\}] = \{q_1, q_3\}. \\ \Delta'(q_1, a) &= \lambda[\Delta(\lambda[q_1], a)] = \lambda[\Delta(\{q_1\}, a)] = \lambda[\{q_2\}] = \{q_2, q_3\}. \\ \Delta'(q_1, b) &= \lambda[\Delta(\lambda[q_1], b)] = \lambda[\Delta(\{q_1\}, b)] = \lambda[\emptyset] = \emptyset. \\ \Delta'(q_2, a) &= \lambda[\Delta(\lambda[q_2], a)] = \lambda[\Delta(\{q_2, q_3\}, a)] = \lambda[\emptyset] = \emptyset. \\ \Delta'(q_2, b) &= \lambda[\Delta(\lambda[q_2], b)] = \lambda[\Delta(\{q_2, q_3\}, b)] = \lambda[\{q_1, q_3\}] = \lambda[\{q_1\}] \cup \lambda[\{q_3\}] = \{q_1, q_3\}. \\ \Delta'(q_3, a) &= \lambda[\Delta(\lambda[q_3], a)] = \lambda[\Delta(\{q_3\}, a)] = \lambda[\emptyset] = \emptyset. \\ \Delta'(q_3, b) &= \lambda[\Delta(\lambda[q_3], b)] = \lambda[\Delta(\{q_3\}, b)] = \lambda[\{q_3\}] = \{q_3\}.\end{aligned}$$

El autómata M' así obtenido es el siguiente:



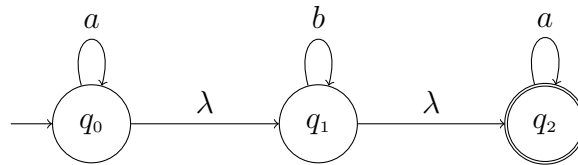
Puesto que q_3 , que es el único estado de aceptación del autómata original M , pertenece a $\lambda[q_0]$, a $\lambda[q_2]$ y a $\lambda[q_3]$, los tres estados q_0 , q_2 y q_3 son estados de aceptación en el autómata M' .

Es importante recalcar que para autómatas sencillos como el autómata M de este ejemplo, es posible obtener M' procediendo por simple inspección. Para ello es necesario tener en cuenta todas las transiciones λ que preceden o prosiguen el procesamiento de cada símbolo de entrada, desde cada estado.

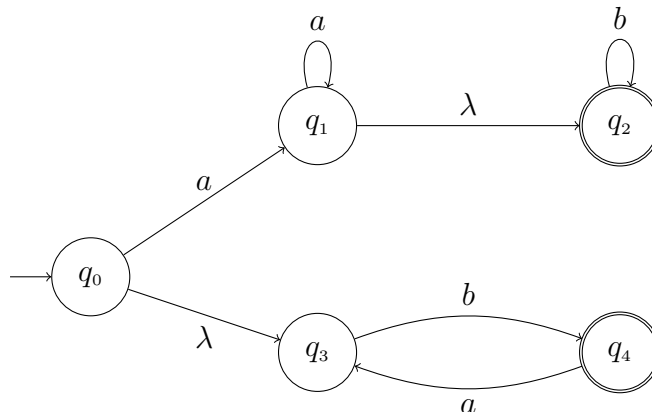
Ejercicios de la sección 2.9

Utilizando el procedimiento presentado en esta sección, construir AFN equivalentes a los siguientes AFN- λ . Proceder ya sea por simple inspección o aplicando explícitamente la fórmula $\Delta'(q, a) = \lambda[\Delta(\lambda[q], a)]$ para todo $q \in Q$ y todo $a \in \Sigma$.

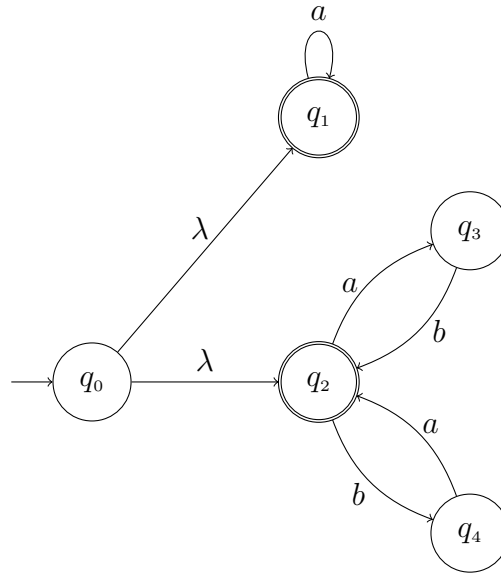
①



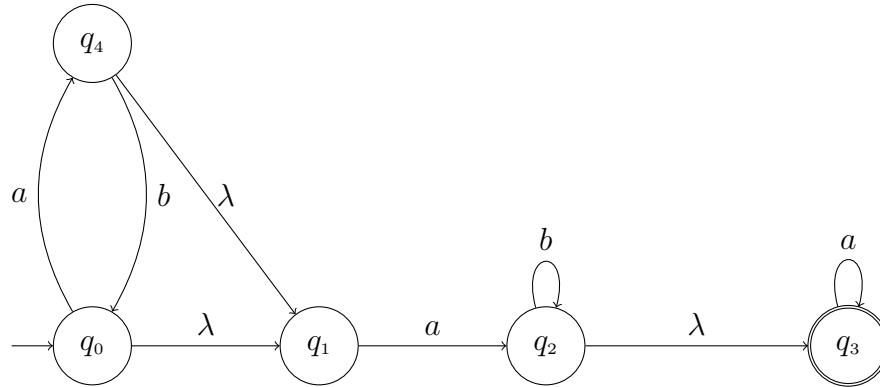
②



③



④



2.10. Complemento de un autómata determinista

El *complemento* de un AFD $M = (\Sigma, Q, q_0, F, \delta)$ es el AFD $\overline{M} = (\Sigma, Q, q_0, \overline{F}, \delta)$ donde $\overline{F} = Q - F$. Es decir, el complemento de M se obtiene intercambiando los estados de aceptación con los de no-aceptación, manteniendo los demás componentes de M . \overline{M} acepta lo que M rechaza y viceversa; se concluye que si $L(M) = L$ entonces $L(\overline{M}) = \overline{L} = \Sigma^* - L$.

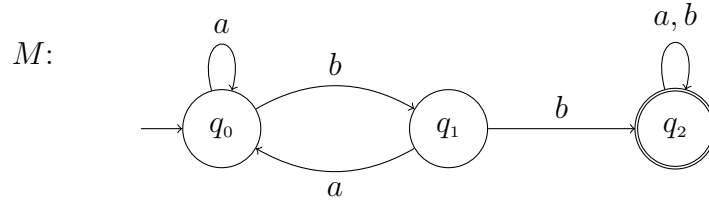
NOTA: Si en M todos los estados son de aceptación, entonces $L(M) = \Sigma^*$. En tal caso, se define el complemento de M como el AFD \overline{M} con dos estados tal que $L(\overline{M}) = \emptyset$.

Cuando un lenguaje L está definido por medio de una condición negativa puede ser más fácil diseñar primero un AFD que acepte su complemento \overline{L} .

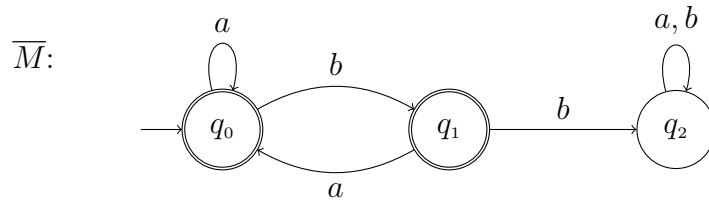
Ejemplo Sea $\Sigma = \{a, b\}$. Encontrar un AFD que acepte el lenguaje L de todas las cadenas que no tienen dos *b*es consecutivas (es decir, no contienen la subcadena

bb).

Diseñamos primero un AFD M que acepte el lenguaje de todas las cadenas que tienen dos b s consecutivas. Esto lo conseguimos forzando la trayectoria de aceptación bb :



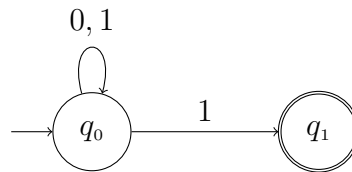
Para aceptar a L formamos el complemento de M , intercambiando aceptación con no-aceptación:



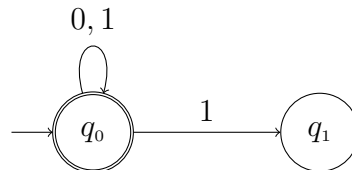
En \overline{M} , q_2 es estado limbo y $L(\overline{M}) = L$.

La noción de complemento no es útil para AFN ya que si M es un AFN tal que $L(M) = L$, no necesariamente se tendrá que $L(\overline{M}) = \overline{L}$, como se aprecia en el siguiente ejemplo.

Ejemplo Sea $\Sigma = \{0, 1\}$ y L el lenguaje de todas las cadenas que terminan en 1. El siguiente AFN acepta a L :



Pero al intercambiar aceptación con no-aceptación se obtiene el AFN:



cuyo lenguaje aceptado es $(0 \cup 1)^*$, diferente de \overline{L} .

Ejercicios de la sección 2.10

Utilizar la noción de complemento de un AFD para diseñar AFD que acepten los siguientes lenguajes:

- ① El lenguaje de todas las cadenas que no contienen la subcadena bc . Alfabeto: $\{a, b, c\}$.
- ② El lenguaje de todas las cadenas que no tienen tres unos consecutivos. Alfabeto: $\{0, 1\}$.
- ③ El lenguaje de todas las cadenas que no terminan en 01 . Alfabeto: $\{0, 1\}$.
- ④ El lenguaje de todas las cadenas que no terminan en 22 . Alfabeto: $\{0, 1, 2\}$.

2.11. Producto cartesiano de autómatas deterministas

Dados dos autómatas deterministas $M_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ se puede formar un nuevo autómata determinista cuyos estados son todas las parejas de la forma (q_i, q_j) , donde $q_i \in Q_1$ y $q_j \in Q_2$. Este nuevo autómata se denomina *producto cartesiano* de M_1 y M_2 y se denota por $M_1 \times M_2$. Concretamente,

$$M_1 \times M_2 = (\Sigma, Q_1 \times Q_2, (q_1, q_2), F, \delta)$$

donde el estado inicial (q_1, q_2) está conformado por los estados iniciales de los dos autómatas, y la función de transición δ está dada por

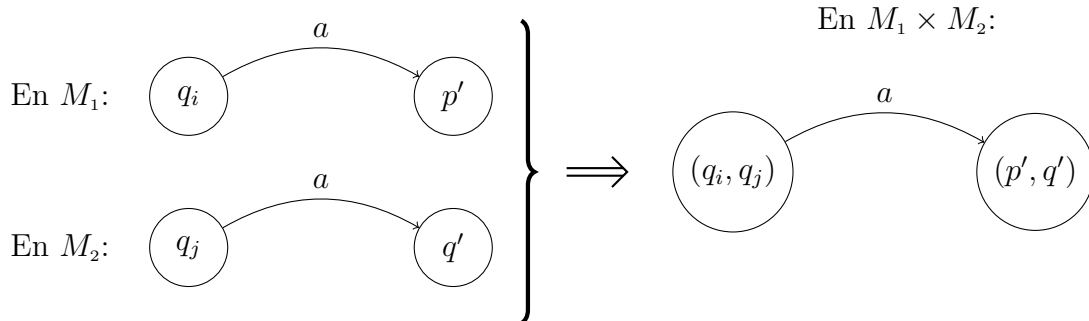
$$\begin{aligned} \delta : (Q_1 \times Q_2) \times \Sigma &\longrightarrow Q_1 \times Q_2 \\ \delta((q_i, q_j), a) &= (\delta_1(q_i, a), \delta_2(q_j, a)). \end{aligned}$$

El conjunto F de estados de aceptación se puede escoger según la conveniencia de la situación. En el siguiente teorema se muestra que es posible escoger F adecuadamente para que $M_1 \times M_2$ acepte ya sea $L_1 \cup L_2$ o $L_1 \cap L_2$ o $L_1 - L_2$.

Según la definición de la función de transición δ , se tiene que

$$\text{Si } \delta_1(q_i, a) = p' \text{ y } \delta_2(q_j, a) = q' \text{ entonces } \delta((q_i, q_j), a) = (p', q'),$$

lo cual se puede visualizar en los grafos de M_1 , M_2 y $M_1 \times M_2$ mediante el siguiente esquema gráfico:



2.11.1 Teorema. Sean $M_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ dos AFD tales que $L(M_1) = L_1$ y $L(M_2) = L_2$, y sea $M = M_1 \times M_2 = (\Sigma, Q_1 \times Q_2, q_1, q_2), F, \delta)$ el producto cartesiano definido arriba.

- (i) Si $F = \{(q_i, q_j) : q_i \in F_1 \text{ ó } q_j \in F_2\}$ entonces $L(M_1 \times M_2) = L_1 \cup L_2$. Es decir, para aceptar $L_1 \cup L_2$, en el autómata $M_1 \times M_2$ se escogen como estados de aceptación los pares de estados (q_i, q_j) en los que alguno de los dos es de aceptación. Formalmente, $F = (F_1 \times Q_2) \cup (Q_1 \times F_2)$.
- (ii) Si $F = \{(q_i, q_j) : q_i \in F_1 \text{ y } q_j \in F_2\}$ entonces $L(M_1 \times M_2) = L_1 \cap L_2$. Es decir, para aceptar $L_1 \cap L_2$, en el autómata $M_1 \times M_2$ se escogen como estados de aceptación los pares de estados (q_i, q_j) en los que ambos son estados de aceptación. Formalmente, $F = F_1 \times F_2$.
- (iii) Si $F = \{(q_i, q_j) : q_i \in F_1 \text{ y } q_j \notin F_2\}$ entonces $L(M_1 \times M_2) = L_1 - L_2$. Es decir, para aceptar $L_1 - L_2$, en el autómata $M_1 \times M_2$ se escogen como estados de aceptación los pares de estados (q_i, q_j) en los que el primero es de aceptación en M_1 y el segundo no lo es en M_2 . Formalmente, $F = F_1 \times (Q_2 - F_2)$.

Demostración. Las conclusiones del teorema se obtienen demostrando primero que la definición de la función δ de $M = M_1 \times M_2$ se puede extender a cadenas arbitrarias:

$$(2.11.1) \quad \widehat{\delta}((q_i, q_j), u) = (\widehat{\delta}_1(q_i, u), \widehat{\delta}_2(q_j, u)) \text{ para toda cadena } u \in \Sigma^*, q_i \in Q_1, q_j \in Q_2.$$

Aquí se usan las funciones extendidas de δ , δ_1 y δ_2 , según la definición 2.7.2. La igualdad (2.11.1) se puede demostrar por recursión sobre u tal como se hace a continuación. Para $u = \lambda$, el resultado es inmediato, y para $u = a$, la igualdad se reduce a la definición de la función δ de $M = M_1 \times M_2$. Para el paso recursivo, suponemos como hipótesis recursiva que (2.11.1) se cumple para una cadena arbitraria u ; se pretende establecer la igualdad para la cadena de entrada ua , donde $a \in \Sigma$. Se tiene

$$\begin{aligned} \widehat{\delta}((q_i, q_j), ua) &= \delta(\widehat{\delta}((q_i, q_j), u), a) && \text{(definición de } \widehat{\delta}) \\ &= \delta((\widehat{\delta}_1(q_i, u), \widehat{\delta}_2(q_j, u)), a) && \text{(hipótesis recursiva)} \\ &= (\delta_1(\widehat{\delta}_1(q_i, u), a), \delta_2(\widehat{\delta}_2(q_j, u), a)) && \text{(definición de } \delta) \\ &= (\widehat{\delta}_1(q_i, ua), \widehat{\delta}_2(q_j, ua)) && \text{(definición de } \widehat{\delta}_1 \text{ y } \widehat{\delta}_2). \end{aligned}$$

Este razonamiento por recursión sobre cadenas concluye la demostración de (2.11.1).

Procedemos ahora a demostrar las afirmaciones (i), (ii) y (iii) del teorema. Usando la igualdad (2.11.1) se tiene que, para toda cadena $u \in \Sigma^*$,

$$u \in L(M) \iff \widehat{\delta}((q_1, q_2), u) \in F \iff (\widehat{\delta}_1(q_1, u), \widehat{\delta}_2(q_2, u)) \in F.$$

Por consiguiente, si $F = \{(q_i, q_j) : q_i \in F_1 \text{ ó } q_j \in F_2\}$, entonces para toda cadena $u \in \Sigma^*$, se tendrá

$$\begin{aligned} u \in L(M) &\iff (\widehat{\delta}_1(q_1, u), \widehat{\delta}_2(q_2, u)) \in F \\ &\iff \widehat{\delta}_1(q_1, u) \in F_1 \vee \widehat{\delta}_2(q_2, u) \in F_2 \\ &\iff u \in L(M_1) \vee u \in L(M_2) \\ &\iff u \in L(M_1) \cup L(M_2) = L_1 \cup L_2. \end{aligned}$$

Esto demuestra (i).

Ahora bien, si $F = \{(q_i, q_j) : q_i \in F_1 \text{ y } q_j \in F_2\}$, entonces para toda cadena $u \in \Sigma^*$, se tendrá

$$\begin{aligned} u \in L(M) &\iff (\widehat{\delta}_1(q_1, u), \widehat{\delta}_2(q_2, u)) \in F \\ &\iff \widehat{\delta}_1(q_1, u) \in F_1 \wedge \widehat{\delta}_2(q_2, u) \in F_2 \\ &\iff u \in L(M_1) \wedge u \in L(M_2) \\ &\iff u \in L(M_1) \cap L(M_2) = L_1 \cap L_2. \end{aligned}$$

Esto demuestra (iii).

Finalmente, si $F = \{(q_i, q_j) : q_i \in F_1 \text{ y } q_j \notin F_2\}$, entonces para toda cadena $u \in \Sigma^*$, se tendrá

$$\begin{aligned} w \in L(M) &\iff (\widehat{\delta}_1(q_1, u), \widehat{\delta}_2(q_2, u)) \in F \\ &\iff \widehat{\delta}_1(q_1, u) \in F_1 \wedge \widehat{\delta}_2(q_2, u) \notin F_2 \\ &\iff u \in L(M_1) \wedge u \notin L(M_2) \\ &\iff u \in L(M_1) - L(M_2) = L_1 - L_2. \end{aligned}$$

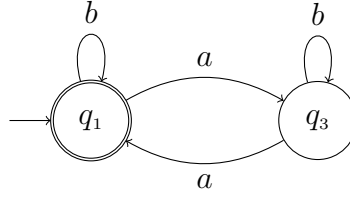
Esto demuestra (iii). □

La construcción del Producto Cartesiano es útil para resolver problemas prácticos porque el esquema gráfico mostrado en la página 57 permite obtener las transiciones de $M_1 \times M_2$ por simple inspección.

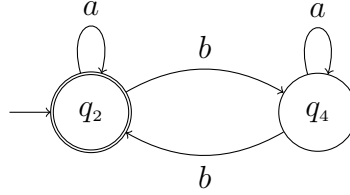
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen un número par de a 's y un número par de b 's.

Solución. En el ejercicio ② de la sección 2.5 se pidió diseñar, por ensayo y error, un AFD para aceptar este lenguaje. Ahora podemos proceder sistemáticamente siguiendo el método del teorema Teorema 2.11.1 ya que el lenguaje L se puede escribir como $L = L_1 \cap L_2$ donde L_1 es el lenguaje de las cadenas con un número par de a 's y L_2 es el lenguaje de las cadenas con un número de par de b 's. Esto nos permite utilizar la parte (ii) del Teorema a partir de autómatas que acepten a L_1 y L_2 , respectivamente.

AFD M_1 que acepta L_1 (cadenas con un número par de a 's):



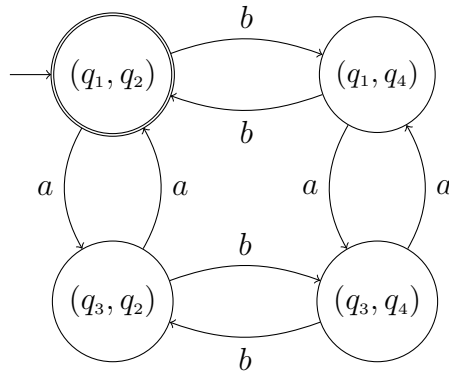
AFD M_2 que acepta L_2 (cadenas con un número par de *bes*):



Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene 4 estados: (q_1, q_2) , (q_1, q_4) , (q_3, q_2) y (q_3, q_4) ; el único estado de aceptación es (q_1, q_2) ya que es el único par de estados en el cual ambos estados son de aceptación. Su función de transición δ se obtiene utilizando la definición de $M_1 \times M_2$.

$$(2.11.2) \quad \left\{ \begin{array}{l} \delta((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a)) = (q_3, q_2), \\ \delta((q_1, q_2), b) = (\delta_1(q_1, b), \delta_2(q_2, b)) = (q_1, q_4), \\ \delta((q_1, q_4), a) = (\delta_1(q_1, a), \delta_2(q_4, a)) = (q_3, q_4), \\ \delta((q_1, q_4), b) = (\delta_1(q_1, b), \delta_2(q_4, b)) = (q_1, q_2), \\ \delta((q_3, q_2), a) = (\delta_1(q_3, a), \delta_2(q_2, a)) = (q_1, q_2), \\ \delta((q_3, q_2), b) = (\delta_1(q_3, b), \delta_2(q_2, b)) = (q_3, q_4), \\ \delta((q_3, q_4), a) = (\delta_1(q_3, a), \delta_2(q_4, a)) = (q_1, q_4), \\ \delta((q_3, q_4), b) = (\delta_1(q_3, b), \delta_2(q_4, b)) = (q_3, q_2). \end{array} \right.$$

El grafo del autómata así obtenido es:



Como se señaló arriba, las transiciones de $M_1 \times M_2$ se pueden obtener también por simple inspección utilizando el esquema gráfico de la página 57, sin necesidad de escribir explícitamente las igualdades en (2.11.2).

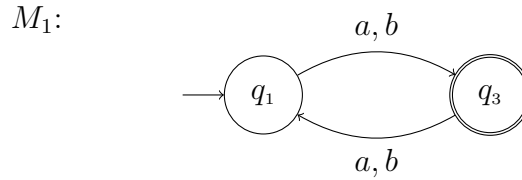
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos b s consecutivas, es decir, no contienen la subcadena bb .

Solución. Utilizamos la parte (ii) del Teorema 2.11.1 expresando L como $L = L_1 \cap L_2$, donde

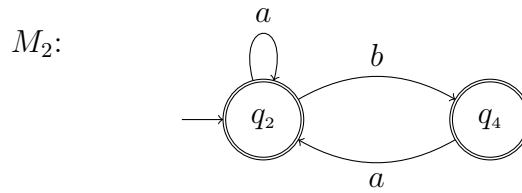
$L_1 =$ lenguaje de todas las cadenas que tienen longitud impar.

$L_2 =$ lenguaje de todas las cadenas que no contienen la subcadena bb .

Encontramos fácilmente un AFD M_1 que acepta el lenguaje L_1 :

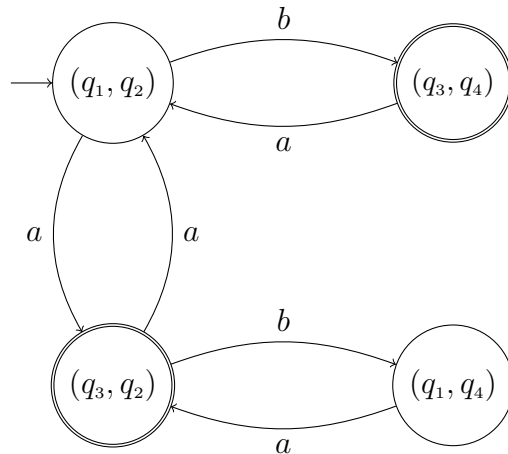


Y un AFD M_2 que acepta L_2 :



El autómata M_2 fue obtenido a partir de su complemento en el primer ejemplo de la sección 2.10. Aquí hemos suprimido el estado limbo ya que no interviene en la aceptación de cadenas, y en el producto cartesiano los estados de aceptación para el lenguaje $L_1 \cap L_2$ son los pares de estados (q_i, q_j) en los que ambos son estados de aceptación.

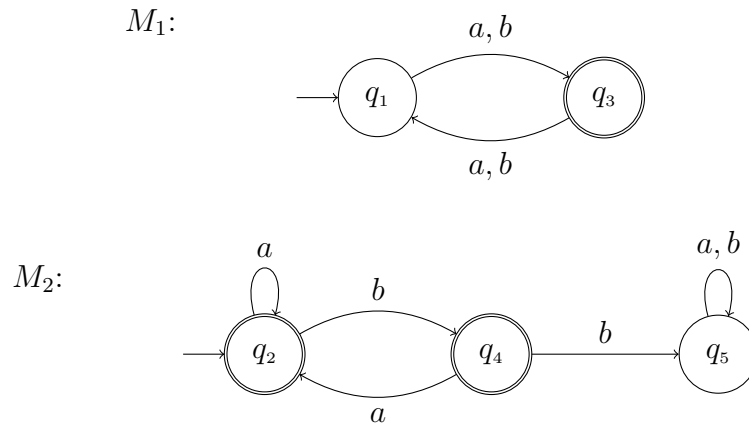
Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene 4 estados: (q_1, q_2) , (q_1, q_4) , (q_3, q_2) y (q_3, q_4) . Los estados de aceptación son (q_3, q_2) y (q_3, q_4) ya que q_3 es de aceptación en M_1 mientras que q_2 y q_4 son de aceptación en M_2 . Utilizando la definición de la función de transición δ de $M_1 \times M_2$ se obtiene el siguiente AFD:



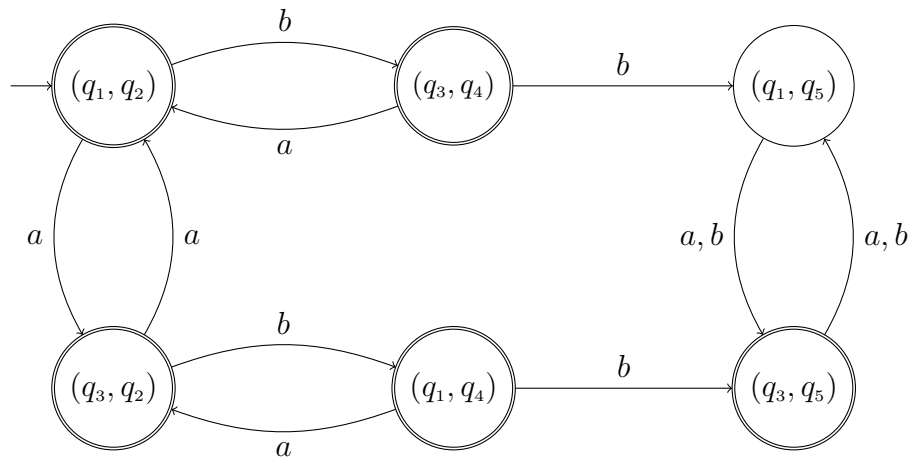
Este problema también se puede resolver expresando el lenguaje L como diferencia de dos lenguajes (véase el Ejercicio ① al final de la presente sección).

Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar o que no contienen dos b s consecutivas.

Solución. Se tiene que $L = L_1 \cup L_2$ donde L_1 y L_2 son los lenguajes definidos en el ejemplo anterior. Utilizamos la parte (i) del Teorema 2.11.1: en el producto cartesiano los estados de aceptación para el lenguaje $L_1 \cup L_2$ son los pares (q_i, q_j) en los que alguno de los dos es un estado de aceptación. Por lo tanto, hay que tener en cuenta los estados limbo de M_1 y M_2 , si los hay:



El producto cartesiano $M = M_1 \times M_3$ tiene seis estados y los estados de aceptación son (q_1, q_2) , (q_3, q_2) , (q_1, q_4) , (q_3, q_4) y (q_3, q_5) .



Así que M requiere seis estados y no hay estado limbo, a pesar de que q_5 es un estado limbo en el autómata M_2 .

Este último ejemplo ilustra que, en general, para construir el producto cartesiano $M_1 \times M_2$, los AFD originales M_1 y M_2 deben ser completos, es decir, deben incluir los

estados limbo, si los hay. Los estados limbo en los autómatas M_1 y M_2 se pueden omitir únicamente cuando se desea aceptar el lenguaje $L_1 \cap L_2$.

Ejercicios de la sección 2.11

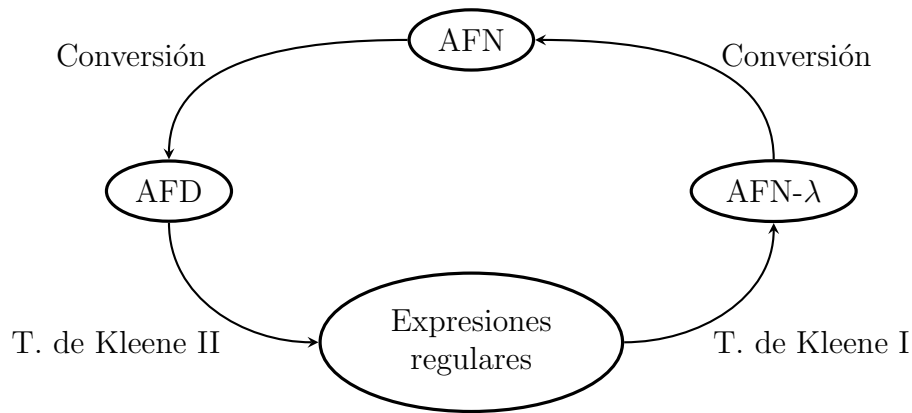
- ① Utilizar el Teorema 2.11.1 (iii) para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos b es consecutivas, expresando L como diferencia de dos lenguajes.
- ② Utilizar el Teorema 2.11.1 para construir AFD que acepten los siguientes lenguajes sobre el alfabeto $\{0, 1\}$:
 - (i) El lenguaje L de todas las cadenas que tienen longitud par o que terminan en 10.
 - (ii) El lenguaje L de todas las cadenas que tienen longitud impar y que terminan en 01.
 - (iii) El lenguaje L de todas las cadenas que tienen longitud impar y que no terminan en 11.
 - (i) El lenguaje L de todas las cadenas que tienen un número par de ceros o que no tienen dos ceros consecutivos.
- ③ Utilizar el Teorema 2.11.1 para construir AFD que acepten los siguientes lenguajes sobre el alfabeto $\{a, b, c\}$:
 - (i) El lenguaje L de todas las cadenas que tienen longitud par y terminan en a .
 - (ii) El lenguaje L de todas las cadenas que tienen longitud par o que tienen un número impar de c 's.
 - (iii) El lenguaje L de todas las cadenas que tienen longitud impar y que tienen un número par de c es.
 - (iv) El lenguaje L de todas las cadenas que tienen longitud impar y que no terminan en c .
 - (v) El lenguaje L de todas las cadenas de longitud impar que tengan exactamente dos a es.
- ④ En el contexto del Teorema 2.11.1, dados dos AFD, $M_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ tales que $L(M_1) = L_1$ y $L(M_2) = L_2$, escoger adecuadamente el conjunto de estados de aceptación F para que el producto cartesiano $M_1 \times M_2 = (\Sigma, Q_1 \times Q_2, (q_1, q_2), F, \delta)$ acepte la diferencia simétrica $L_1 \triangleleft L_2$. Recuerdese que la diferencia simétrica se define como

$$L_1 \triangleleft L_2 = (L_1 \cup L_2) - (L_1 \cap L_2) = (L_1 - L_2) \cup (L_2 - L_1).$$

2.12. Teorema de Kleene, parte I

En las secciones anteriores se ha mostrado la equivalencia computacional de los modelos AFD, AFN y AFN- λ , lo cual quiere decir que para cada autómata de uno de estos tres modelos se pueden construir autómatas equivalentes en los otros modelos. Por lo tanto, los autómatas AFD, AFN y AFN- λ aceptan exactamente la misma colección de lenguajes. El Teorema de Kleene establece que tal colección de lenguajes la conforman precisamente los lenguajes regulares, representados por las expresiones regulares.

El siguiente diagrama esboza los procedimientos constructivos de conversión entre los modelos de autómatas y las expresiones regulares.



2.12.1. Teorema de Kleene. Sea Σ un alfabeto dado. Un lenguaje es regular (sobre Σ) si y sólo si es aceptado por un autómata finito (AFD o AFN o AFN- λ) con alfabeto de entrada Σ .

Para demostrar el teorema consideraremos las dos direcciones por separado.

Parte I del Teorema de Kleene. Para un lenguaje regular, representado por una expresión regular R dada, se puede construir un AFN- λ M tal que el lenguaje aceptado por M sea exactamente el lenguaje representado por R , es decir, $L(M) = L[R]$. Por simplicidad escribiremos simplemente $L(M) = R$.

Demostración. Puesto que se ha dado una definición recursiva de las expresiones regulares, la demostración se lleva a cabo razonando recursivamente sobre R . Para las expresiones regulares básicas, podemos construir fácilmente autómatas que acepten los lenguajes representados. Así, el autómata



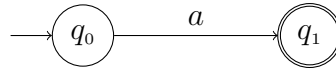
acepta el lenguaje \emptyset , es decir, el lenguaje representado por la expresión regular $R = \emptyset$.

El autómata



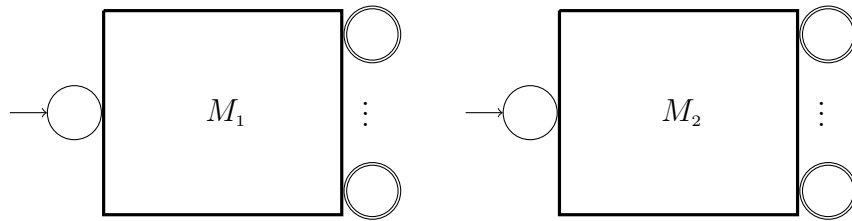
acepta el lenguaje $\{\lambda\}$, es decir, el lenguaje representado por la expresión regular $R = \lambda$.

El autómata



acepta el lenguaje $\{a\}$, $a \in \Sigma$, es decir, el lenguaje representado por la expresión regular $R = a$.

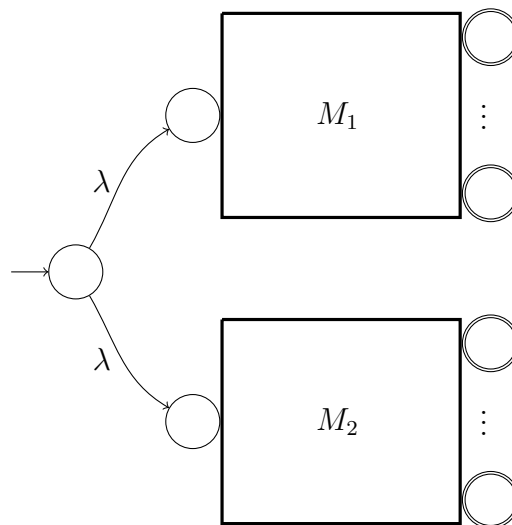
Razonando recursivamente, supóngase que para las expresiones regulares R_1 y R_2 se dispone de AFN- λ M_1 y M_2 tales que $L(M_1) = R_1$ y $L(M_2) = R_2$. Esquemáticamente vamos a presentar los autómatas M_1 y M_2 en la siguiente forma:



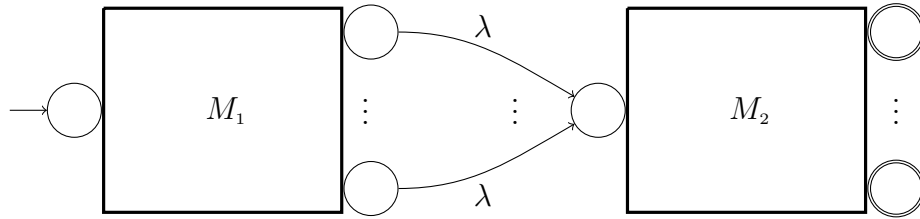
Los estados finales o de aceptación se dibujan a la derecha, pero cabe advertir que el estado inicial puede ser también un estado de aceptación. Podemos ahora obtener AFN- λ que acepten los lenguajes $R_1 \cup R_2$ y $R_1 R_2$.

Para aceptar $R_1 \cup R_2$ los autómatas M_1 y M_2 se conectan mediante lo que se denomina una *conexión en paralelo*. Hay un nuevo estado inicial y los estados de aceptación del nuevo autómata son los estados de aceptación de M_1 , junto con los de M_2 .

Autómata que acepta $R_1 \cup R_2$:

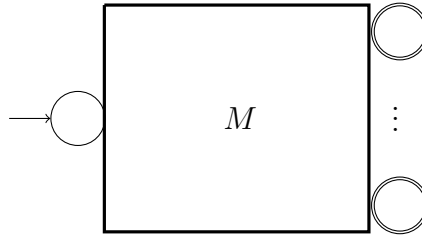


Autómata que acepta R_1R_2 :

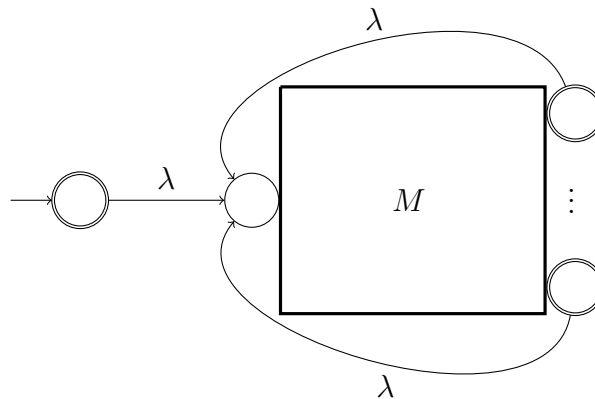


Este tipo de conexión entre dos autómatas M_1 y M_2 se denomina *conexión en serie*. Los estados de aceptación del nuevo autómata son únicamente los estados de aceptación de M_2 .

Supóngase ahora R es una expresión regular y M es un AFN- λ tal que $L(M) = R$:

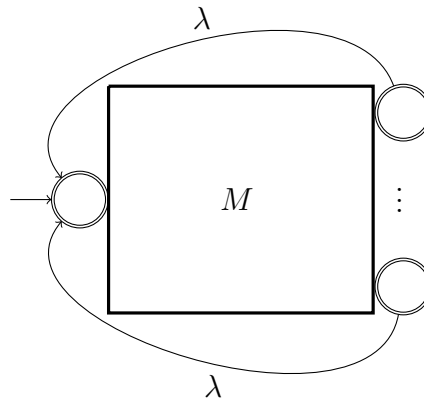


El siguiente autómata acepta R^* :

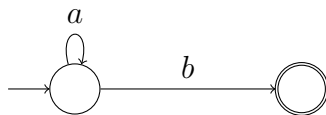
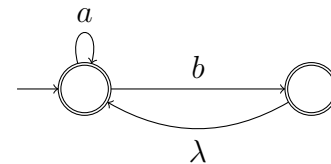


Esto concluye la demostración de la parte I del Teorema de Kleene. □

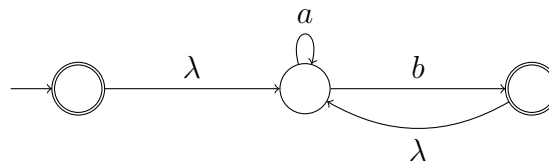
Para aceptar R^* a partir de un autómata M tal que $L(M) = R$, sería incorrecto (en general) utilizar el siguiente autómata M' :

M' :

A primera vista esta construcción parece razonable, pero al convertir el estado inicial en estado de aceptación, M' podría aceptar cadenas adicionales no pertenecientes a R^* . Como contraejemplo consideremos la expresión regular $R = a^*b$. En la gráfica siguiente se exhibe a la izquierda un AFN M tal que $L(M) = a^*b$ y a la derecha el autómata M' obtenido realizando la construcción esbozada arriba.

 M : M' :

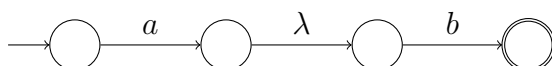
No se cumple que $L(M') = R^* = (a^*b)^*$ ya que M' acepta también a, a^2, a^3, \dots (potencias de a) que no pertenecen a R^* . Para aceptar R^* utilizando la construcción mencionada en la demostración del Teorema 2.12.1 se obtendría el siguiente autómata M'' :

 M'' :

Simplificaciones en el procedimiento. El procedimiento constructivo del Teorema 2.12.1 admite varias simplificaciones, útiles en la práctica.

Para aceptar ab :

Según el procedimiento:

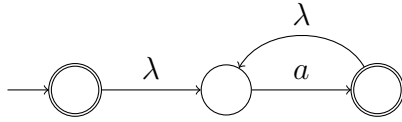


Simplificación:

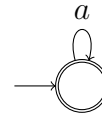


Para aceptar a^* :

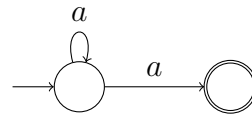
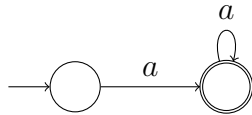
Según el procedimiento:



Simplificación:



Para aceptar a^+ podemos usar una cualquiera de las siguientes dos simplificaciones:

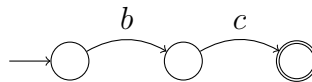


Además, las conexiones en paralelo y en serie de dos autómatas se pueden generalizar fácilmente para aceptar uniones $R_1 \cup R_2 \cup R_3 \cup \dots \cup R_k$, o concatenaciones $R_1 R_2 R_3 \dots R_k$, de k lenguajes ($k \geq 3$).

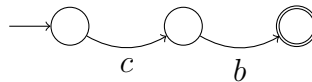
Ejemplo Utilizar el procedimiento del Teorema 2.12.1 para construir un AFN- λ que acepte el lenguaje $(bc \cup cb)^* a^* b \cup (b^* ca)^* c^+$ sobre el alfabeto $\Sigma = \{a, b, c\}$.

Solución. Escribimos la expresión regular $R = (bc \cup cb)^* a^* b \cup (b^* ca)^* c^+$ como $R = R_1 \cup R_2$ donde $R_1 = (bc \cup cb)^* a^* b$ y $R_2 = (b^* ca)^* c^+$. Construimos dos autómatas que acepten R_1 y R_2 , respectivamente, y luego los conectamos en paralelo.

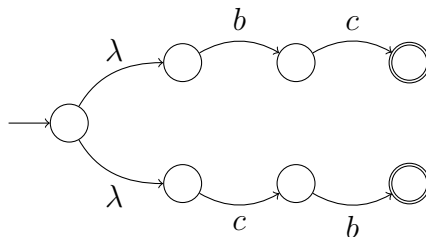
Autómata que acepta bc :



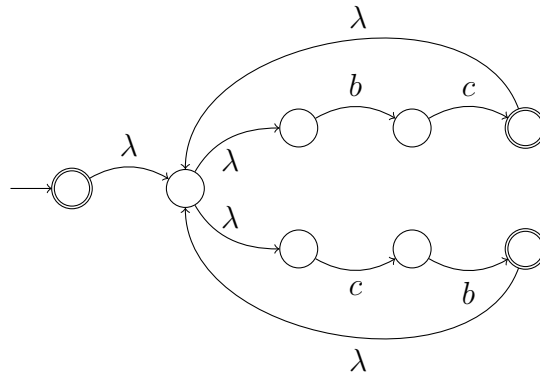
Autómata que acepta cb :



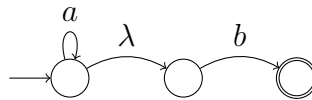
Autómata que acepta $bc \cup cb$:



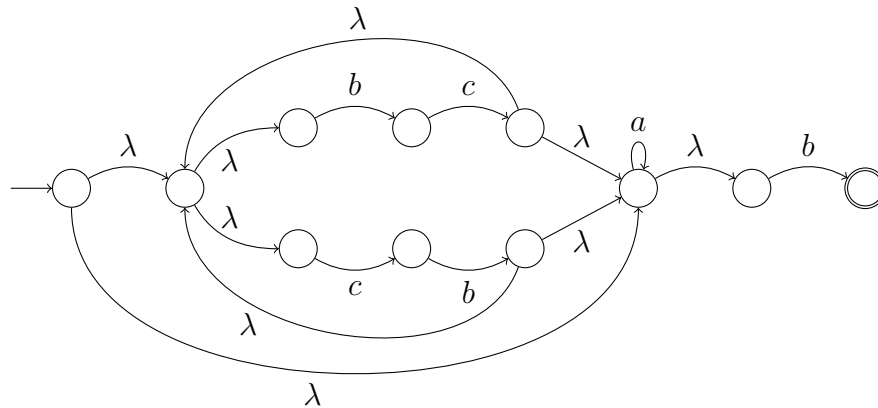
Autómata que acepta $(bc \cup cb)^*$:



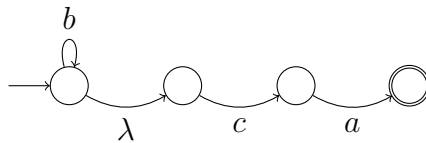
Autómata que acepta a^*b :



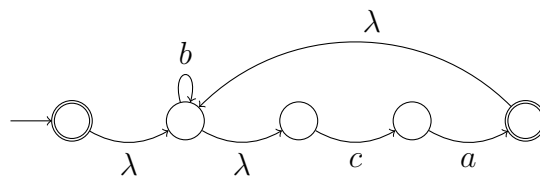
Para aceptar $R_1 = (bc \cup cb)^*a^*b$ conectamos en serie los dos últimos autómatas:



A continuación construimos un autómata que acepte $R_2 = (b^*ca)^*c^+$. Autómata que acepta b^*ca :



Autómata que acepta $(b^*ca)^*$:



- ② $(a^*cb)^*(a \cup b)(a \cup bc)^*$.
- ③ $(a \cup ba \cup ca)^*(\lambda \cup a)b^+c^+$.
- ④ $(a^*bc)^* \cup (cb^*a)^+ \cup (ca \cup cb \cup c^2)^*a^*b^+$.
- ⑤ $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^* \cup (b \cup \lambda)(b^*ac)^*$.

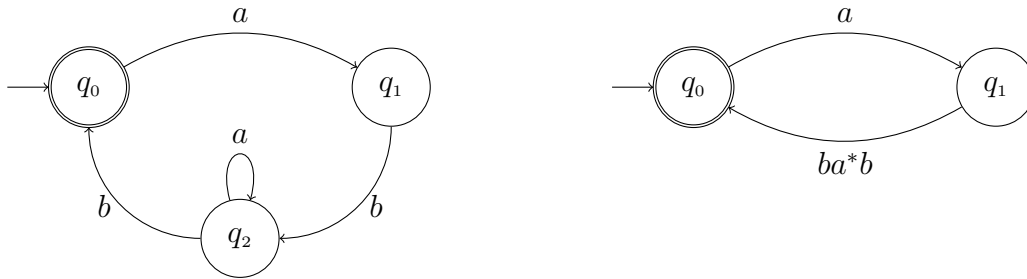
2.13. Teorema de Kleene, parte II

Parte II del Teorema de Kleene. Dado un autómata M , ya sea AFD o AFN o AFN- λ , se puede encontrar una expresión regular R tal que $L(M) = R$.

La demostración de este enunciado será también constructiva y se basa en la noción de grafo etiquetado generalizado, o GEG, que es un grafo como el de un autómata excepto que las etiquetas de los arcos entre estados pueden ser expresiones regulares en lugar de simplemente símbolos del alfabeto. El procedimiento consiste en eliminar uno a uno los estados del autómata original M , obteniendo en cada paso un GEG cuyo lenguaje aceptado coincide con $L(M)$. Cuando el grafo se reduce a dos estados (uno de ellos debe ser el estado inicial), el lenguaje aceptado se puede obtener por simple inspección.

Antes de presentar el procedimiento en todo detalle, consideraremos un ejemplo sencillo.

Ejemplo A la izquierda aparece el grafo de un AFD M dado. Se observa que el estado q_2 solamente sirve de “puente” o “pivote” entre q_1 y q_0 y, por consiguiente, se puede eliminar añadiendo un arco entre q_1 y q_0 con etiqueta ba^*b . Se obtiene el GEG (grafo derecho) cuyo lenguaje aceptado (por inspección) es $(aba^*b)^*$.



El procedimiento general para encontrar una expresión regular R que represente el lenguaje aceptado por un autómata M dado consta de los siguientes pasos:

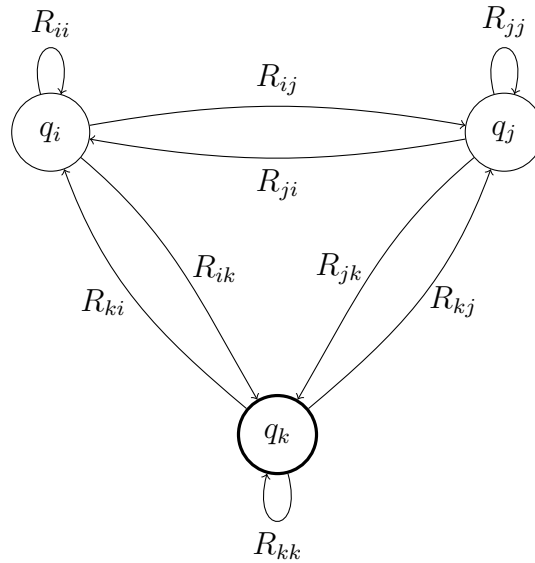
- (1) Convertir el grafo de M en un GEG G reemplazando múltiples arcos etiquetados con símbolos a_1, a_2, \dots, a_k entre dos estados q_i y q_j por un único arco etiquetado $a_1 \cup a_2 \cup \dots \cup a_k$:



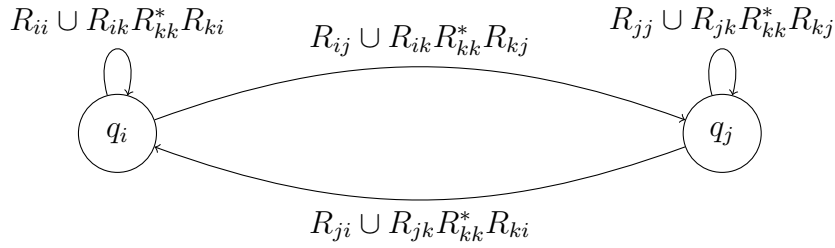
De esta forma, entre dos estados cualesquiera hay, a lo sumo, un arco etiquetado con una expresión regular.

- (2) Modificar el GEG G de tal manera que haya un único estado de aceptación. Esto se hace trazando transiciones λ en la forma indicada en la sección 2.8, página 50. Cuando el autómata original posee un único estado de aceptación, simplemente se mantiene como tal hasta el final.
- (3) Este paso es un ciclo iterativo por medio del cual se van eliminando uno a uno los estados de G hasta que permanezcan únicamente dos estados (uno de ellos debe ser el estado inicial q_0). Para presentar el procedimiento en forma general utilizaremos la siguiente notación: se denota con R_{ij} la etiqueta (expresión regular) entre dos estados q_i y q_j ; si no existe un arco entre q_i y q_j , se considera que $R_{ij} = \emptyset$.

Si hay tres o más estados, escoger un estado cualquiera q_k , diferente de q_0 y que no sea un estado de aceptación. Se pretende eliminar q_k añadiendo adecuadamente transiciones entre los estados restantes de tal manera que el lenguaje aceptado no se altere. Sean q_i y q_j dos estados, diferentes de q_k , con arcos etiquetados por expresiones regulares, en la siguiente forma:

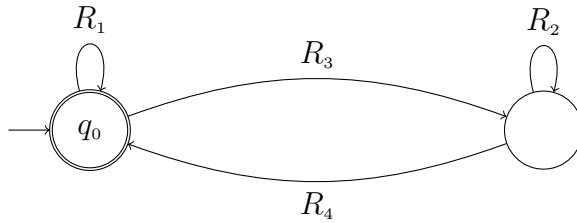


q_k sirve de “puente” entre q_i y q_j , y entre q_j y q_i . Además, a través de q_k hay una trayectoria que conecta q_i consigo mismo, y también una trayectoria que conecta q_j consigo mismo. Teniendo en cuenta tales trayectorias, se procede a eliminar el estado q_k reemplazando las etiquetas entre q_i y q_j por las siguientes:

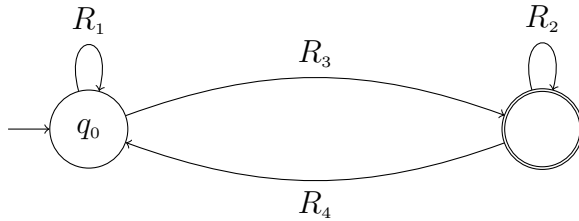


Lo anterior se debe realizar para todos los pares de estados q_i y q_j (diferentes de q_k). Para tener en cuenta arcos inexistentes entre estados, en todo momento se deben usar las simplificaciones: $R \cup \emptyset = R$, $R\emptyset = \emptyset$ y $\emptyset^* = \lambda$. Eliminar posteriormente el estado q_k junto con todos los arcos que entra o salen de él.

- (4) Finalmente, cuando haya solamente dos estados, se puede obtener una expresión regular para $L(M)$ considerando los siguientes dos casos:



$$L(M) = (R_1 \cup R_3 R_2^* R_4)^*.$$



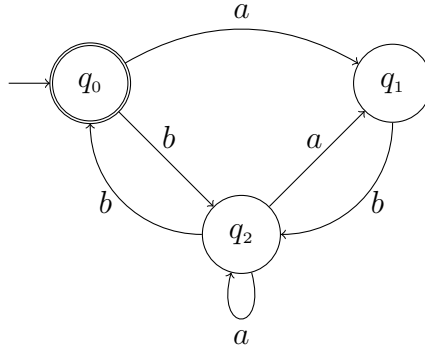
$$L(M) = R_1^* R_3 (R_2 \cup R_4 R_1^* R_3)^*.$$

Observaciones:

1. El procedimiento anterior es bastante flexible; por ejemplo, el paso (2) (estado de aceptación único) se puede realizar después de haber eliminado uno o más estados (siguiendo las instrucciones del paso (3)).
2. Es importante recalcar que, siempre que se ejecute la subrutina (3), el estado inicial y los estados de aceptación no se pueden eliminar.
3. Para la solución de problemas concretos no es necesario memorizar las expresiones regulares para q_i y q_j que se muestran en el grafo del paso (3), ni las expresiones para $L(M)$ que aparecen en el paso (4). Tales expresiones se pueden deducir por simple inspección en autómatas concretos.

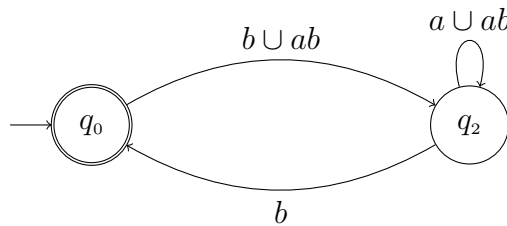
Ejemplo

Encontrar una expresión regular para el lenguaje aceptado por el siguiente autómata M .



Solución. El estado inicial q_0 se puede mantener hasta el final como el único estado de aceptación. A continuación se puede eliminar el estado q_1 , o bien el estado q_2 . En general, siempre resulta más conveniente eliminar primero los estados que tengan el menor número de arcos entrantes/salientes, ya que se obtienen así expresiones regulares más sencillas. Con este criterio, procedemos primero a eliminar el estado q_1 .

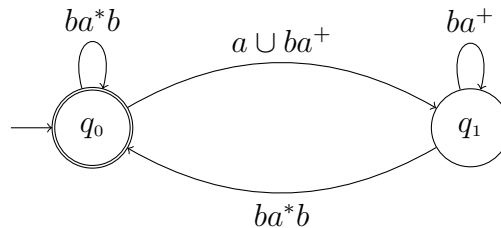
Eliminación del estado q_1 :



Por inspección se obtiene que $L(M) = [(b \cup ab)(a \cup ab)^*b]^*$.

Si en el autómata original se elimina q_2 en vez de q_1 , se llega a una expresión regular más compleja.

Eliminación del estado q_2 :

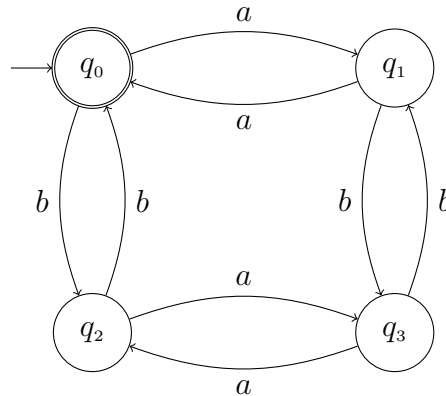


Por inspección obtenemos

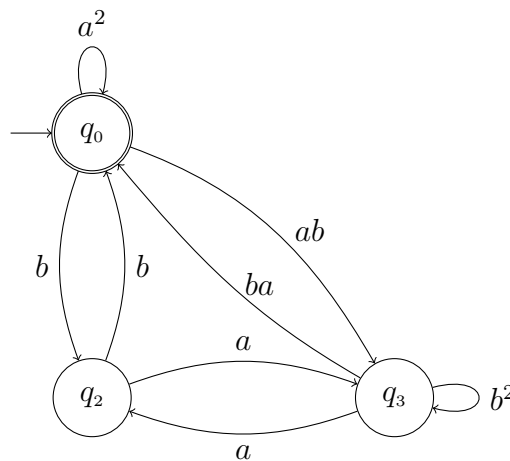
$$L(M) = [ba^*b \cup (a \cup ba^+)(ba^+)^*ba^*b]^*.$$

Ejemplo Utilizar el procedimiento presentado en la presente sección para encontrar una expresión regular que represente el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen un número par de a s y un número par de b s.

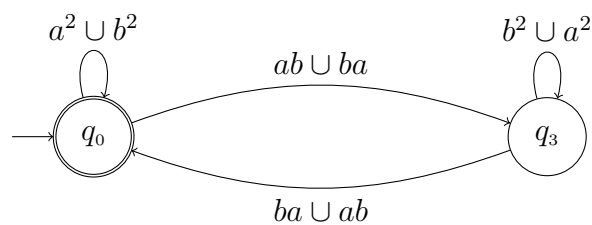
Solución. Conocemos un AFD M que acepta este lenguaje:



El estado inicial q_0 se puede mantener hasta el final como el único estado de aceptación. Procedemos primero a eliminar el estado q_1 (debido a la simetría del grafo de M , también podríamos eliminar primero q_2 , o bien q_3).



A continuación podemos eliminar ya sea q_2 o q_3 . Puesto que q_2 no tiene bucles es más sencillo eliminarlo:



El lenguaje aceptado es entonces

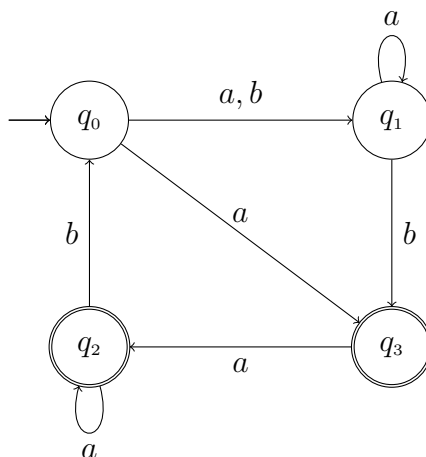
$$L(M) = [a^2 \cup b^2 \cup (ab \cup ba)(a^2 \cup b^2)^*(ab \cup ba)]^*.$$

Si en el penúltimo GEG se elimina q_3 en vez de q_2 , se llega finalmente a una expresión regular diferente (y más compleja) que representa el mismo lenguaje.

Ejemplo

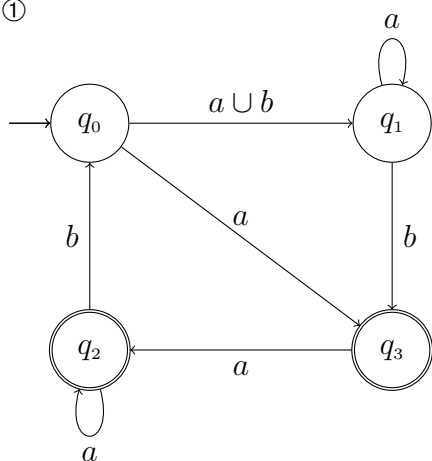
Encontrar una expresión regular para el lenguaje aceptado por el siguiente autómata M .

M :

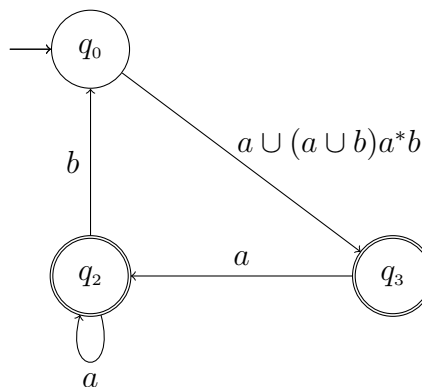


Solución. Primero convertimos el grafo de M en un GEG (grafo ①) y luego eliminamos el estado q_1 (grafo ②).

①

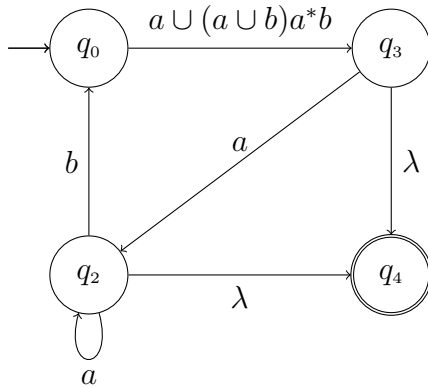


②

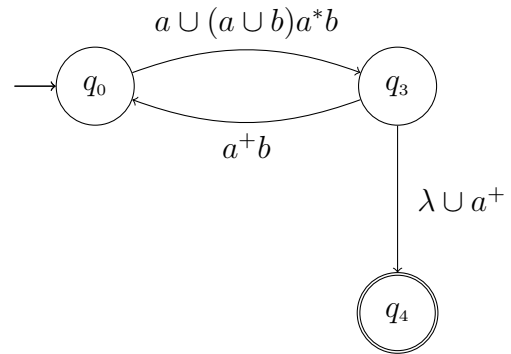


A continuación añadimos el nuevo estado q_4 (que será el único estado de aceptación) y transiciones λ desde q_2 y q_3 hasta q_4 (grafo ③). Luego eliminamos el estado q_2 (grafo ④):

③

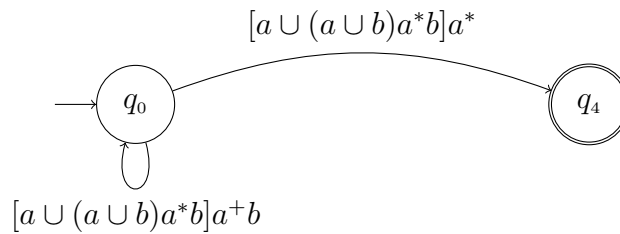


④



Finalmente, eliminamos el estado q_3 ; teniendo en cuenta que $\lambda \cup a^+ = a^*$, obtenemos:

⑤



Con el grafo ⑤ obtenemos el lenguaje aceptado por simple inspección:

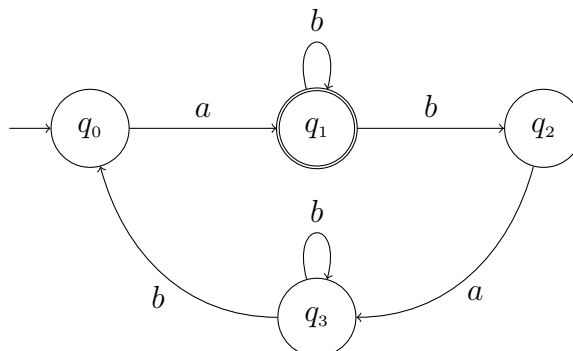
$$L(M) = \left([a \cup (a \cup b)a^*b]a^+b \right)^* [a \cup (a \cup b)a^*b]a^*.$$

Podemos observar que en el grafo ③ es también posible eliminar q_3 en vez de q_2 ; procediendo así se llega a una expresión regular mucho más compleja para $L(M)$.

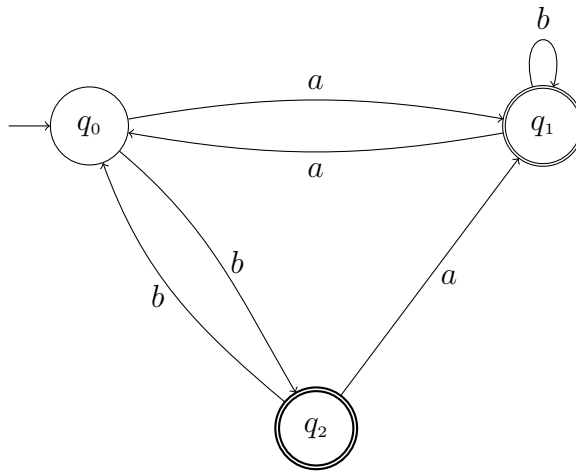
Ejercicios de la sección 2.13

- ① Utilizar el procedimiento presentado en la presente sección para encontrar expresiones regulares para los lenguaje aceptados por los siguientes autómatas:

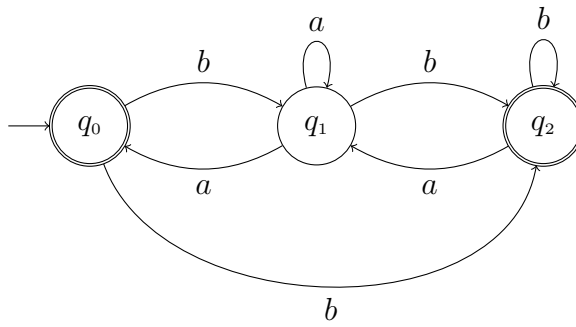
(i)



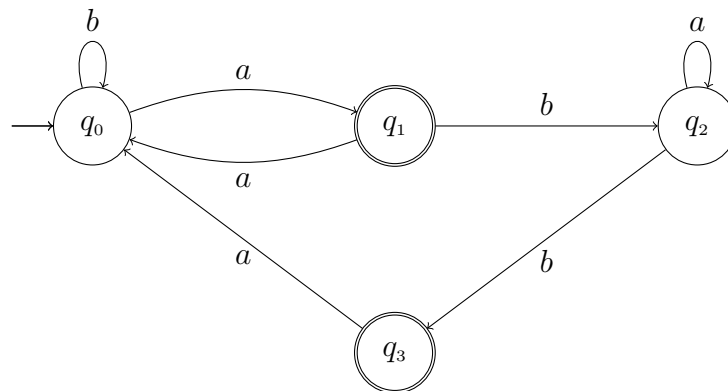
(ii)



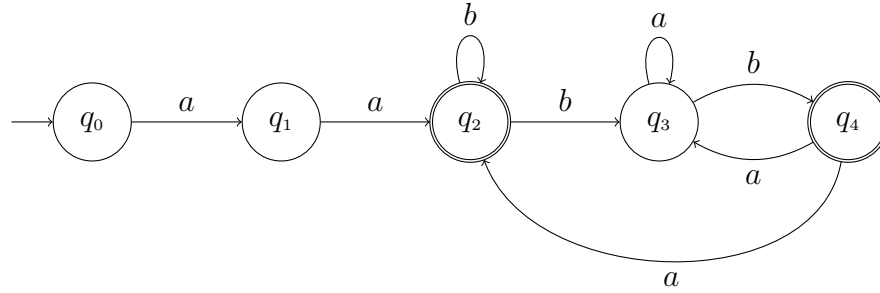
(iii)



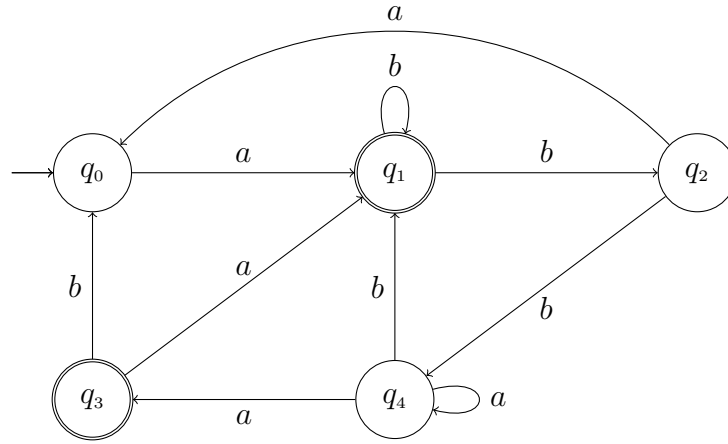
(iv)



(v)



(vi)



- ② Sea $\Sigma = \{a, b, c\}$ y L el lenguaje de todas las cadenas que no contienen la subcadena bc . Diseñar un autómata M que acepte el lenguaje L , y luego utilizar el procedimiento presentado en la presente sección para encontrar una expresión regular para L . Esta expresión regular se puede comparar con las obtenidas en el último ejemplo de la sección 2.2, página 24.
- ③ Sea $\Sigma = \{a, b\}$ y $L = \{u \in \Sigma^* : [\#_a(u) - \#_b(u)] \equiv 1 \pmod{3}\}$. La notación $\#_a(u)$ representa el número de a s en la cadena u mientras que $\#_b(u)$ es el número de b s. Diseñar un AFD M con tres estados que acepte el lenguaje L , y luego utilizar el procedimiento presentado en la presente sección para encontrar una expresión regular para L .

2.14. Propiedades de clausura de los lenguajes regulares

Las propiedades de clausura afirman que a partir de lenguajes regulares se pueden obtener otros lenguajes regulares por medio de ciertas operaciones entre lenguajes. Es decir, la regularidad es preservada por ciertas operaciones entre lenguajes; en tales casos se dice que *los lenguajes regulares son cerrados bajo las operaciones*.

Inicialmente presentamos las propiedades de clausura para autómatas. El siguiente teorema resume los procedimientos algorítmicos que han sido presentados en secciones anteriores para la construcción de nuevos autómatas finitos.

2.14.1 Teorema. Sean M , M_1 y M_2 autómatas finitos (ya sean AFD o AFN o AFN- λ) tales que $L(M) = L$, $L(M_1) = L_1$, $L(M_2) = L_2$. Se pueden construir autómatas finitos que acepten los siguientes lenguajes:

- | | |
|----------------------|--------------------------------|
| (1) $L_1 \cup L_2$. | (5) $\bar{L} = \Sigma^* - L$. |
| (2) $L_1 L_2$. | (6) $L_1 \cap L_2$. |
| (3) L^* . | (7) $L_1 - L_2$. |
| (4) L^+ . | (8) $L_1 \triangleleft L_2$. |

Demostración. La construcción de autómatas que acepten $L_1 \cup L_2$, $L_1 L_2$ y L^* se presentó en la demostración de la parte I del Teorema de Kleene. Como $L^+ = L^* L = L L^*$, también se pueden utilizar tales construcciones para (4).

Para construir un autómata que acepte \bar{L} , se construye primero un AFD que acepte a L y se intercambian luego los estados de aceptación con los de no aceptación. Se obtiene así el complemento de M , tal como se explicó en la sección 2.10.

Para construir autómatas que acepten $L_1 \cap L_2$ y $L_1 - L_2$, basta formar el producto cartesiano de dos AFDs que acepten a L_1 y L_2 , y escoger adecuadamente los estados de aceptación, tal como se indicó en la sección 2.11. También se puede usar el producto cartesiano para aceptar $L_1 \cup L_2$.

Finalmente, los procedimientos de construcción de (1), (6) y (7) se pueden combinar para obtener un autómata que acepte el lenguaje $L_1 \triangleleft L_2 = (L_1 - L_2) \cup (L_2 - L_1)$. \square

Puesto que, según el Teorema de Kleene, los lenguajes regulares son precisamente los lenguajes aceptados por autómatas finitos, el Teorema 2.14.1 se puede presentar en términos de lenguajes regulares.

2.14.2 Teorema. Si L , L_1 y L_2 son lenguajes regulares sobre un alfabeto Σ , también son regulares los siguientes lenguajes:

- | | |
|----------------------|--------------------------------|
| (1) $L_1 \cup L_2$. | (5) $\bar{L} = \Sigma^* - L$. |
| (2) $L_1 L_2$. | (6) $L_1 \cap L_2$. |
| (3) L^* . | (7) $L_1 - L_2$. |
| (4) L^+ . | (8) $L_1 \triangleleft L_2$. |

2.15. Teorema de Myhill-Nerode

El Teorema de Kleene (Teorema 2.12.1) caracteriza los lenguajes regulares como los lenguajes que son aceptados por autómatas finitos. Otra caracterización importante de los lenguajes regulares se establece en el Teorema de Myhill-Nerode, resultado con muchas consecuencias importantes. Este teorema se enuncia en términos de la relación de indistinguibilidad entre cadenas, noción definida a continuación.

2.15.1 Definición. Sea Σ un alfabeto dado y L un lenguaje sobre Σ (o sea, $L \subseteq \Sigma^*$). Dos cadenas $u, v \in \Sigma^*$ son *indistinguibles con respecto a L* (o *L -indistinguibles*) si

$$(2.15.1) \quad (\forall x \in \Sigma^*) [ux \in L \iff vx \in L].$$

Utilizaremos la notación uI_Lv para representar el hecho de que las cadenas u y v son L -indistinguibles. Es necesario recalcar que esta definición se da para un lenguaje L cualquiera, no necesariamente regular. La relación de indistinguibilidad I_L también se llama relación de Myhill-Nerode.

Si dos cadenas u, v no son indistinguibles con respecto a L , se dice que son *distinguibles con respecto a L* o *L -distinguibles*. Afirmar que u y v son L -distinguibles equivale a negar (2.15.1); así que $u, v \in \Sigma^*$ son L -distinguibles si

$$(\exists x \in \Sigma^*) [(ux \in L \text{ y } vx \notin L) \text{ ó } (ux \notin L \text{ y } vx \in L)].$$

Es decir, u y v son L -distinguibles si existe x en Σ^* tal que una de las cadenas ux ó vx está en L y la otra no.

Se deduce fácilmente de la definición 2.15.1 que I_L es una relación de equivalencia sobre Σ^* , es decir, se cumplen las siguientes propiedades para todas las cadenas $u, v, w \in \Sigma^*$:

- Reflexividad. uI_Lu .
- Simetría. Si uI_Lv entonces vI_Lu .
- Transitividad. Si uI_Lv y vI_Lw entonces uI_Lw .

Con el siguiente resultado podemos obtener muchos ejemplos concretos de cadenas L -indistinguibles, cuando L es un lenguaje regular.

2.15.2 Proposición. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$. Si para dos cadenas u y v en Σ^* se tiene que $\hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$, entonces u y v son L -indistinguibles, esto es uI_Lv . En otras palabras, dos cadenas que terminan de leerse en el mismo estado son L -indistinguibles.

Demostración. Hay que demostrar que $(\forall x \in \Sigma^*) [ux \in L \iff vx \in L]$. Sea $x \in \Sigma^*$; puesto que $\hat{\delta}(q_0, u) = \hat{\delta}(q_0, v)$, se tendrá que

$$\hat{\delta}(q_0, ux) = \hat{\delta}(\hat{\delta}(q_0, u), x) = \hat{\delta}(\hat{\delta}(q_0, v), x) = \hat{\delta}(q_0, vx).$$

Entonces

$$ux \in L = L(M) \iff \widehat{\delta}(q_0, ux) \in F \iff \widehat{\delta}(q_0, vx) \in F \iff vx \in L(M) = L.$$

Por lo tanto, uI_Lv . □

Recordamos que una relación de equivalencia R definida sobre un conjunto cualquiera A , particiona a A en clases disyuntas llamadas clases de equivalencia. El conjunto de todas las clases de equivalencia se denota A/R y se denomina el *conjunto cociente inducido por la relación R* . La cardinalidad del conjunto cociente A/R se conoce como el *índice de la relación R* . La relación de equivalencia R tiene índice finito si el conjunto cociente A/R es finito, es decir, si R particiona a A en un número finito de clases de equivalencia.

2.15.3. Teorema de Myhill-Nerode. Sea Σ un alfabeto dado y L un lenguaje sobre Σ . L es regular si y sólo si la relación de indistinguibilidad I_L es de índice finito, es decir, si I_L determina un número finito de clases de equivalencia sobre Σ^* .

Demostración. (\implies) Si L es regular, existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$, con todos sus estados accesibles; es decir, para cada $q \in Q$ existe $u \in \Sigma^*$ tal que $\widehat{\delta}(q_0, u) = q$. Se define la relación R_M sobre Σ^* de la siguiente manera:

$$uR_Mv \text{ si y solo si } \widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v).$$

Esto es, las cadenas u y v están relacionadas si terminan de leerse en el mismo estado. Es bastante claro que R_M es una relación de equivalencia, o sea, R_M es reflexiva, simétrica y transitiva. Cada clase de equivalencia está formada por cadenas que terminan de leerse en un mismo estado. Por consiguiente, hay tantas clases de equivalencia en Σ^*/R_M como estados hay en Q , o sea, $|\Sigma^*/R_M| = |Q|$. En consecuencia, R_M es de índice finito.

La conexión entre las relaciones de equivalencia I_L y R_M es la siguiente: para todo $u \in \Sigma^*$ se cumple

$$(2.15.2) \quad [u]_{R_M} \subseteq [u]_{I_L}.$$

En palabras: la clase de u según R_M está contenida en la clase de u según I_L . Para demostrar (2.15.2) observamos que la contención es equivalente a la implicación

$$uR_Mv \implies uI_Lv,$$

o sea,

$$\widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v) \implies u, v \text{ son } L\text{-indistinguibles},$$

y esto es precisamente lo que afirma la Proposición 2.15.2, demostrada arriba.

La contención (2.15.2) se expresa usualmente diciendo que R_M es un refinamiento de I_L e implica que toda I_L -clase de equivalencia es la unión de R_M -clases de equivalencia. Se deduce entonces que

$$(2.15.3) \quad |\Sigma^*/I_L| \leq |\Sigma^*/R_M| = |Q|.$$

Esto muestra que I_L es de índice finito.

(\Leftarrow) Si I_L es de índice finito se puede construir un AFD M_L que acepte a L . Los estados de M_L son precisamente las clases de equivalencia $[u]_{I_L}$ determinadas por las relación de indistinguibilidad I_L . Para simplificar la notación, escribiremos simplemente $[u]$ en vez de $[u]_{I_L}$. Formalmente se define $M_L = (\Sigma, Q_L, q_i, F_L, \delta_L)$ estableciendo

$$\begin{aligned} Q_L &= \{[u] : u \in \Sigma^*\} = \Sigma^*/I_L, \\ q_i &= [\lambda] \text{ es el estado inicial de } M_L, \\ F_L &= \{[u] : u \in L\}, \\ \delta_L([u], a) &= [ua], \text{ para todo } u \in \Sigma^* \text{ y todo } a \in \Sigma. \end{aligned}$$

Hay que verificar que la función de transición δ_L está bien definida, es decir, que no depende del representante escogido en la clase de equivalencia, lo cual equivale a demostrar que

$$uI_Lv \implies uaI_Lva, \text{ para todo } a \in \Sigma.$$

Esto se deduce de la definición de I_L (Ejercicio ① al final de la sección). También hay que demostrar que F_L está bien definido, lo cual equivale a demostrar que

$$u \in L \text{ y } v \in [u] \implies v \in L.$$

Esto se obtiene también de la definición de I_L (véase el Ejercicio ①). Finalmente, hay que demostrar que $L(M_L) = L$, para lo cual se demuestra primero, por recursión sobre u , que

$$(2.15.4) \quad \widehat{\delta}_L([\lambda], u) = [u], \text{ para toda cadena } u \in \Sigma^*,$$

(véase el Ejercicio ②).

Se tiene entonces, para toda $u \in \Sigma^*$,

$$\begin{aligned} u \in L(M_L) &\implies \widehat{\delta}_L([\lambda], u) \in F_L \quad (\text{por definición de aceptación}) \\ &\implies [u] \in F_L \quad (\text{por la propiedad (2.15.4)}) \\ &\implies u \in L \quad (\text{por definición de } F_L). \quad \square \end{aligned}$$

Con el Teorema de Kleene se puede encontrar un autómata AFN- λ para aceptar un lenguaje regular L , a partir de una expresión regular para L , mientras que en la demostración del Teorema de Myhill-Nerode se construye el AFD “abstracto” M_L para aceptar a L , utilizando la relación de equivalencia I_L . M_L resulta ser un AFD con el mínimo número de estados posible para aceptar a L . Esta es la consecuencia más importante del Teorema de Myhill-Nerode y se demuestra en el siguiente teorema.

2.15.4 Teorema. Sea L un lenguaje regular sobre Σ . El autómata $M_L = (\Sigma, Q_L, q_i, F_L, \delta_L)$ construido en la demostración del Teorema de Myhill-Nerode es un AFD con el mínimo número de estados posible para aceptar a L . Además, un autómata mínimo para aceptar a L es único salvo isomorfismo (es decir, salvo los nombres de los estados).

Para demostrar la última afirmación de este teorema se necesita definir de manera precisa la noción de autómatas isomorfos.

2.15.5 Definición. Dos AFD $M_1 = (\Sigma, Q_1, q_1, F_1, \delta_1)$ y $M_2 = (\Sigma, Q_2, q_2, F_2, \delta_2)$ son isomorfos si existe una biyección $f : Q_1 \longrightarrow Q_2$ tal que

- (1) $f(q_1) = q_2$.
- (2) $q \in F_1 \iff f(q) \in F_2$.
- (3) $\delta_2(f(q), a) = f(\delta_1(q, a))$, para todo $q \in Q_1$ y todo $a \in \Sigma$.

La propiedad (3) afirma que las funciones de transición δ_1 y δ_2 coinciden, una vez se renombran los estados.

Demostración del Teorema 2.15.4. En la demostración del Teorema de Myhill-Nerode se considera un AFD *cualquiera* que acepta a L y se define la relación de equivalencia R_M sobre Σ^* para concluir la desigualdad (2.15.3) que afirma que el número de estados de M_L es menor o igual que el número de estados de M . Esto demuestra que M_L es un AFD con el mínimo número de estados posible para aceptar a L .

Para demostrar que $M_L = (\Sigma, Q_L, q_i, F_L, \delta_L)$ es único, salvo isomorfismo, consideramos un AFD arbitrario $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$, con el mínimo número de estados posible, es decir, $|Q| = |Q_L|$, y definimos la función $f : Q \longrightarrow Q_L$ de la siguiente manera:

$$(2.15.5) \quad f(q) = [u]_{I_L}, \text{ donde } \widehat{\delta}(q_0, u) = q, \text{ para alguna cadena } u \in \Sigma^*.$$

Hay que verificar que f está bien definida. En primer lugar, un estado cualquiera $q \in Q$ es accesible (de lo contrario, q se podría eliminar para obtener un AFD con menos estados para aceptar a L). Por lo tanto, dado $q \in Q$ existe $u \in \Sigma^*$ tal que $\widehat{\delta}(q_0, u) = q$. En segundo lugar, la definición de f no depende del representante $u \in \Sigma^*$. En efecto, si $\widehat{\delta}(q_0, u) = \widehat{\delta}(q_0, v) = q$, entonces uI_Lv por la Proposición 2.15.2; de donde $[u]_{I_L} = [v]_{I_L}$.

Según la definición, f es claramente sobreyectiva, y como $|Q| = |Q_L|$, se concluye que f es biyectiva. Las propiedades (1) y (2) en la definición de isomorfismo son fáciles de verificar para f (véase el Ejercicio ③) mientras que la propiedad (3) se convierte en

$$\delta_L(f(q), a) = f(\delta_1(q, a)), \text{ con } q \in Q_1 \text{ y } a \in \Sigma.$$

Para demostrarla, tomamos $u \in \Sigma^*$ tal que $\widehat{\delta}(q_0, u) = q$. Entonces $\widehat{\delta}(q_0, ua) = \delta(\widehat{\delta}(q_0, u), a) = \delta(q, a)$, lo cual implica, según la definición de la función f , que

$$(2.15.6) \quad f(\delta(q, a)) = [ua]_{I_L},$$

Por consiguiente, se tiene

$$\begin{aligned} \delta_L(f(q), a) &= \delta_L([u]_{I_L}, a), && \text{por la definición de } f(q) \\ &= [ua]_{I_L}, && \text{por la definición de } \delta_L \\ &= f(\delta(q, a)), && \text{por (2.15.6)} \quad \square \end{aligned}$$

Ejercicios de la sección 2.15

- ① Sea Σ un alfabeto dado, $L \subseteq \Sigma^*$ y I_L la relación de indistinguibilidad definida sobre Σ^* . Para cada $u \in \Sigma^*$, $[u]$ denota la clase de equivalencia de u , determinada por I_L . Demostrar que para todo $u, v \in \Sigma^*$ se cumple
- (i) Si $u \in L$ y $v \in [u]$ entonces $v \in L$.
 - (ii) Si uI_Lv entonces uaI_Lva , para todo $a \in \Sigma$.
- ② Demostrar la afirmación (2.15.4) de la demostración del Teorema de Myhill-Nerode, esto es, demostrar por recursión sobre u , que

$$\widehat{\delta}_L([\lambda], u) = [u], \text{ para toda cadena } u \in \Sigma^*.$$

- ③ Demostrar que la función $f : Q \rightarrow Q_L$ definida en la demostración del Teorema 2.15.4 satisface las propiedades (1) y (2) de la Definición 2.15.5.

2.16. Algoritmo de minimización de AFDs

En la presente sección abordaremos el siguiente problema: dado un AFD M encontrar un AFD con el mínimo número de estados posible que acepte el mismo lenguaje L aceptado por M . Se presentará un algoritmo general para resolver este problema y se demostrará que el autómata obtenido M' coincide (salvo los nombres de los estados) con el AFD M_L definido en el Teorema de Myhill-Nerode, el cual es, según el Teorema 2.15.4, un autómata mínimo para aceptar a L .

En el procedimiento de minimización se identifican “estados equivalentes”, en un sentido que se precisará detalladamente, lo cual permite “colapsar estados” en el autómata original y de esta manera reducir el número de estados hasta el mínimo posible.

2.16.1 Definición. Dado un AFD $M = (\Sigma, Q, q_0, F, \delta)$ y dos estados $p, q \in Q$, se dice que p es equivalente a q , notado $p \approx q$, si:

$$p \approx q \text{ si y sólo si } (\forall u \in \Sigma^*) [\widehat{\delta}(p, u) \in F \iff \widehat{\delta}(q, u) \in F].$$

Es fácil comprobar que la relación \approx es reflexiva, simétrica y transitiva; es decir, para todos los estados p, q, r de Q se cumple:

- Reflexividad. $p \approx p$.
- Simetría. Si $p \approx q$ entonces $q \approx p$.
- Transitividad. Si $p \approx q$ y $q \approx r$ entonces $p \approx r$.

Por lo tanto, \approx es una relación de equivalencia sobre el conjunto de estados Q . Si $p \approx q$ se dice que p y q son *estados equivalentes*. La clase de equivalencia de un estado p se denotará con $[p]$; es decir,

$$[p] := \{q \in Q : p \approx q\}.$$

Se define el *autómata cociente* M' identificando entre sí los estados equivalentes según la relación \approx . Formalmente, $M' = (\Sigma, Q', q'_0, F', \delta')$ donde:

$$Q' = \{[p] : p \in Q\},$$

$$q'_0 = [q_0],$$

$$F' = \{[p] : p \in F\},$$

$$\delta'([p], a) = [\delta(p, a)], \text{ para todo } a \in \Sigma.$$

Hay que verificar que tanto F' como la función de transición δ' están bien definidos, es decir, que no dependen del representante escogido en la clase de equivalencia. Esto se hace en la siguiente proposición.

2.16.2 Proposición.

- (i) δ' está bien definida, es decir, si $[p] = [q]$ (o sea, si $p \approx q$) entonces $\delta(p, a) \approx \delta(q, a)$ para todo $a \in \Sigma$.
- (ii) F' está bien definido, es decir, si $q \in F$ y $p \approx q$ entonces $p \in F$.
- (iii) $p \in F \iff [p] \in F'$.
- (iv) $\widehat{\delta'}([p], u) = [\widehat{\delta}(p, u)]$ para toda cadena $u \in \Sigma^*$.

Demostración.

- (i) Si $p \approx q$, entonces

$$(\forall u \in \Sigma^*)(\forall a \in \Sigma)[\widehat{\delta}(p, au) \in F \iff \widehat{\delta}(q, au) \in F],$$

de donde

$$(\forall u \in \Sigma^*)[\widehat{\delta}(\widehat{\delta}(p, a), u) \in F \iff \widehat{\delta}(\delta(q, a), u) \in F],$$

para todo $a \in \Sigma$. Por la definición de la relación \approx , se concluye que $\delta(p, a) \approx \delta(q, a)$.

- (ii) Tomando $u = \lambda$ en la definición de $p \approx q$, se tiene que $p = \widehat{\delta}(p, \lambda) \in F$ si y solo si $q = \widehat{\delta}(q, \lambda) \in F$. Puesto que $q \in F$, se concluye que $p \in F$.
- (iii) La dirección (\implies) se sigue de la definición de F' . Para demostrar la otra dirección sea $[p] \in F'$. Entonces $[p] = [q]$, con $q \in F$; de donde $p \approx q$. De (ii) se sigue que $p \in F$.
- (iv) Se demuestra por recursión sobre u . □

Usando las propiedades de la Proposición 2.16.2 se puede deducir que M y M' aceptan el mismo lenguaje, tal como se demuestra en el siguiente teorema.

2.16.3 Teorema. El autómata M y el autómata cociente M' aceptan el mismo lenguaje, es decir, $L(M) = L(M') = L$.

Demostración.

$$\begin{aligned}
 u \in L(M') &\iff \widehat{\delta}'([q_0], u) \in F' \\
 &\iff [\widehat{\delta}(q_0, u)] \in F' \quad (\text{por la Proposición 2.16.2 (iv)}) \\
 &\iff \widehat{\delta}(q_0, u) \in F \quad (\text{por la Proposición 2.16.2 (iii)}) \\
 &\iff u \in L(M). \quad \square
 \end{aligned}$$

Para demostrar que M' es isomorfo a M_L , el autómata definido en el Teorema de Myhill-Nerode, establecemos primero un recíproco parcial de la Proposición 2.15.2.

2.16.4 Proposición. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$. Si dos cadenas u y $v \in \Sigma^*$ son L -indistinguibles, o sea, si $uI_L v$, entonces $\widehat{\delta}(q_0, u) \approx \widehat{\delta}(q_0, v)$.

Demostración. Sean $p = \widehat{\delta}(q_0, u)$ y $q = \widehat{\delta}(q_0, v)$. Hay que demostrar que

$$(\forall w \in \Sigma^*) [\widehat{\delta}(p, w) \in F \iff \widehat{\delta}(q, w) \in F].$$

Sea $w \in \Sigma^*$. Se tiene que

$$(2.16.1) \quad \widehat{\delta}(p, w) = \widehat{\delta}(\widehat{\delta}(q_0, u), w) = \widehat{\delta}(q_0, uw),$$

$$(2.16.2) \quad \widehat{\delta}(q, w) = \widehat{\delta}(\widehat{\delta}(q_0, v), w) = \widehat{\delta}(q_0, vw).$$

Entonces

$$\begin{aligned}
 \widehat{\delta}(p, w) \in F &\iff \widehat{\delta}(q_0, uw) \in F \quad (\text{por (2.16.1)}) \\
 &\iff uw \in L(M) = L \\
 &\iff vw \in L \quad (\text{por ser } u \text{ y } v \text{ indistinguibles}) \\
 &\iff \widehat{\delta}(q_0, vw) \in F \\
 &\iff \widehat{\delta}(q, w) \in F \quad (\text{por (2.16.2)}). \quad \square
 \end{aligned}$$

2.16.5 Teorema. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$, con todos sus estados accesibles. El autómata cociente M' y el autómata M_L definido en el Teorema de Myhill-Nerode son isomorfos. Por consiguiente, M' es un AFD con el mínimo número de estados posible para aceptar a L , y es único salvo los nombres de los estados.

Demostración. Puesto que $L(M') = L$, es suficiente demostrar que $|Q'| = |Q_L|$ e invocar el Teorema 2.15.4. Como M_L es un AFD mínimo para aceptar a L , se tiene que $|Q_L| \leq |Q'|$;

entonces basta demostrar que $|Q'| \leq |Q_L|$. Para ello definimos la función $g : Q_L \longrightarrow Q'$ por medio de

$$g([u]_{I_L}) = [q], \text{ donde } q = \widehat{\delta}(q_0, u).$$

$[q]$ denota la clase de equivalencia del estado q según la relación \approx . La función g está bien definida ya que si $[u]_{I_L} = [v]_{I_L}$, entonces $uI_L v$ y se tendrá $\widehat{\delta}(q_0, u) \approx \widehat{\delta}(q_0, v)$ por la Proposición 2.16.4.

Como los estados de M son accesibles, g es sobreyectiva. Se sigue que $|Q'| \leq |Q_L|$, que era lo que se quería demostrar. \square

Dado un AFD M , se dispone de un algoritmo para encontrar el autómata cociente M' , y se le conoce como *algoritmo de minimización por llenado de tabla*. Es muy importante tener presente que para aplicar este algoritmo se requiere que todos los estados de M dado sean accesibles, según la definición 2.7.3. Los estados inaccesibles se deben eliminar previamente ya que son inútiles. Además, siendo un autómata determinista, M debe ser *completo*, es decir, para cada estado q y cada símbolo $a \in \Sigma$, la transición $\delta(q, a)$ debe estar definida. Por consiguiente, en el grafo de M se deben mostrar todos los estados, incluyendo los llamados “estados limbo”.

**Algoritmo por llenado de tabla para determinar la
equivalencia de estados en un AFD**

ENTRADA:

AFD $M = (\Sigma, Q, q_0, F, \delta)$ completo (incluyendo estados limbo) cuyos estados son todos accesibles y tabla triangular que muestra todos los pares $\{p, q\}$ de estados $p, q \in Q$.

INICIALIZAR:

$i := 1$. Se marca con \times la casilla $\{p, q\}$ si $p \in F$ y $q \notin F$ (o viceversa).

REPETIR:

$i := i + 1$. Para cada casilla no marcada $\{p, q\}$ y cada $a \in \Sigma$, hallar $\{\delta(p, a), \delta(q, a)\}$. Si para algún $a \in \Sigma$, la casilla $\{\delta(p, a), \delta(q, a)\}$ ha sido marcada previamente, entonces se marca la casilla $\{p, q\}$ con \times .

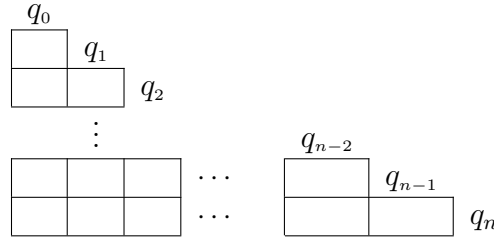
HASTA:

No se puedan marcar más casillas en la tabla.

SALIDA:

Si la casilla $\{p, q\}$ está marcada con \times , entonces $p \not\approx q$. Si la casilla $\{p, q\}$ no está marcada, entonces $p \approx q$.

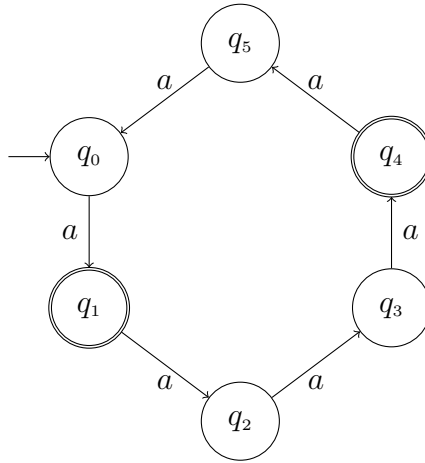
El algoritmo se ejecuta llenando una tabla que muestra todos los pares de estados $\{p, q\}$; el propósito es determinar cuáles son equivalentes y cuáles no. Puesto que en el par $\{p, q\}$ no importa el orden de los estados, basta trabajar en una tabla de formato triangular como la siguiente:



En cada iteración la tabla se recorre por columnas, q_0, q_1, \dots, q_n . La casilla $\{p, q\}$ es marcada por el algoritmo de minimización si y sólo si existe $u \in \Sigma^*$ tal que $\widehat{\delta}(q, u) \in F$ y $\widehat{\delta}(p, u) \notin F$, o viceversa, es decir, si y solo si $p \approx q$. Por consiguiente, las casillas no marcadas al finalizar el algoritmo representan estados equivalentes.

Si en la definición de la relación \approx se toma $u = \lambda$, se deduce que si $p \approx q$, entonces $\widehat{\delta}(p, \lambda) \in F \iff \widehat{\delta}(q, \lambda) \in F$, o sea, $p \in F \iff q \in F$. Esto implica que si $p \approx q$, entonces ambos estados son de aceptación o ninguno de los dos lo es. De manera que si p es estado de aceptación y q no lo es (o viceversa), se tendrá $p \not\approx q$. Esta es la razón por la cual el algoritmo de minimización comienza marcando las casillas $\{p, q\}$ para las cuales $p \in F$ y $q \notin F$ (o viceversa).

Ejemplo Aplicar el algoritmo de minimización para encontrar un AFD con el menor número de estados posible, equivalente al siguiente AFD, cuyo alfabeto de entrada es $\{a\}$.



Solución. Primero marcamos con \times las casillas $\{p, q\}$ para las cuales p es un estado de aceptación y q no lo es, o viceversa:

q_0					
\times	q_1				
	\times	q_2			
	\times		q_3		
\times		\times	\times	q_4	
	\times			\times	q_5

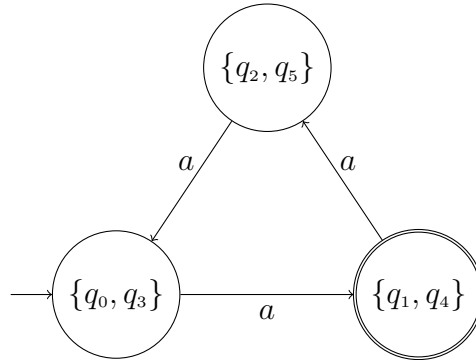
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos el símbolo a . La información necesaria aparece en la siguiente tabla; la columna izquierda corresponde a las casillas no marcadas $\{p, q\}$ y la derecha a las casillas $\{\delta(p, a), \delta(q, a)\}$ obtenidas al procesar a . Esta tabla se utiliza a partir de la segunda iteración.

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$
$\{q_0, q_2\}$	$\{q_1, q_3\} \times$
$\{q_0, q_3\}$	$\{q_1, q_4\}$
$\{q_0, q_5\}$	$\{q_1, q_0\} \times$
$\{q_1, q_4\}$	$\{q_2, q_5\}$
$\{q_2, q_3\}$	$\{q_3, q_4\} \times$
$\{q_2, q_5\}$	$\{q_3, q_0\}$
$\{q_3, q_5\}$	$\{q_4, q_0\} \times$

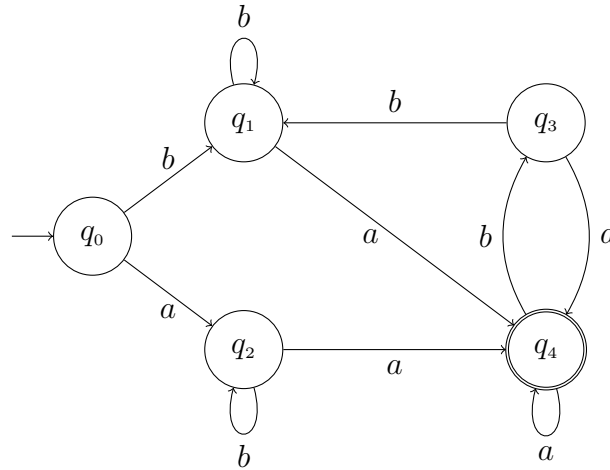
Las marcas \times en la columna derecha representan casillas previamente marcadas; según el algoritmo, las correspondientes casillas en la columna izquierda se marcan con \times . Entonces, al terminar la segunda iteración la tabla triangular adquiere el siguiente aspecto:

q_0						
\times	q_1					
\times	\times	q_2				
	\times	\times	q_3			
\times		\times	\times	q_4		
\times	\times		\times	\times	q_5	

En la tercera iteración ya no se pueden marcar más casillas y el algoritmo termina. Las casillas vacías representan estados equivalentes; así que $q_0 \approx q_3$, $q_1 \approx q_4$ y $q_2 \approx q_5$. El autómata cociente M' tiene entonces tres estados (las tres clases de equivalencia): $\{q_0, q_3\}$, $\{q_1, q_4\}$ y $\{q_2, q_5\}$; el grafo obtenido es:


Ejemplo

Aplicar el algoritmo de minimización para encontrar un AFD con el menor número de estados posible, equivalente al siguiente AFD.



Al marcar con \times las casillas $\{p, q\}$ para las cuales p es un estado de aceptación y q no lo es, o viceversa, obtenemos la tabla:

q_0				
	q_1			
		q_2		
			q_3	
\times	\times	\times	\times	q_4

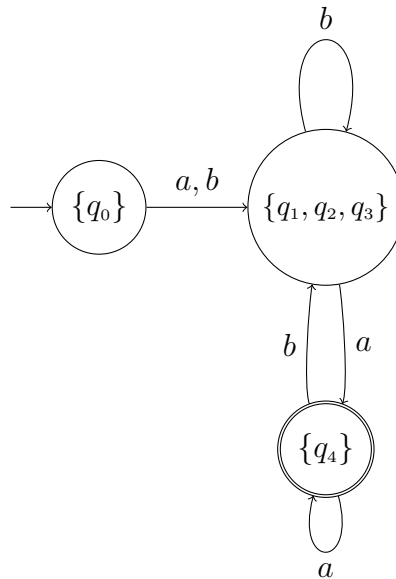
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos con las entradas a y b . Tal información aparece en la siguiente tabla, la cual se utiliza a partir de la segunda iteración.

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_0, q_1\}$	$\{q_2, q_4\} \times$	$\{q_1, q_1\}$
$\{q_0, q_2\}$	$\{q_2, q_4\} \times$	$\{q_1, q_2\}$
$\{q_0, q_3\}$	$\{q_2, q_4\} \times$	$\{q_1, q_1\}$
$\{q_1, q_2\}$	$\{q_4, q_4\}$	$\{q_1, q_2\}$
$\{q_1, q_3\}$	$\{q_4, q_4\}$	$\{q_1, q_1\}$
$\{q_2, q_3\}$	$\{q_4, q_4\}$	$\{q_2, q_1\}$

Las marcas \times representan casillas previamente marcadas; según el algoritmo, las correspondientes casillas en la columna izquierda se marcan con \times . Entonces, al terminar la segunda iteración obtenemos la tabla triangular:

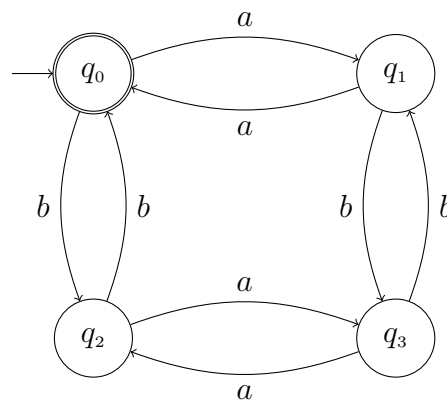
q_0				
\times	q_1			
\times		q_2		
\times			q_3	
\times	\times	\times	\times	q_4

En la tercera iteración ya no se marcan más casillas y el algoritmo termina. Se deduce que los tres estados q_1 , q_2 y q_3 son equivalentes entre sí ($q_1 \approx q_2 \approx q_3$). El autómata cociente posee entonces tres estados, a saber, $\{q_0\}$, $\{q_1, q_2, q_3\}$ y $\{q_4\}$. Su grafo es:



Ejemplo Sea $\Sigma = \{a, b\}$. Demostrar que el lenguaje L de todas las cadenas que tienen un número par de a 'es y un número par de b 'es no puede ser aceptado por ningún AFD con menos de cuatro estados.

Solución. Ya conocemos un AFD M que acepta este lenguaje:



El problema se reduce a minimizar este AFD con el objeto de determinar si o no es posible construir uno equivalente con menos de cuatro estados. Al aplicar el algoritmo de minimización tenemos inicialmente las siguientes marcas sobre la tabla:

	q_0		
×	q_1		
×		q_2	
×			q_3

Consideramos luego las casillas no marcadas y procesamos con a y con b :

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_1, q_2\}$	$\{q_0, q_3\} \times$	$\{q_3, q_0\} \times$
$\{q_1, q_3\}$	$\{q_0, q_3\} \times$	$\{q_3, q_1\}$
$\{q_2, q_3\}$	$\{q_3, q_2\}$	$\{q_0, q_1\} \times$

Llegamos entonces a la tabla triangular

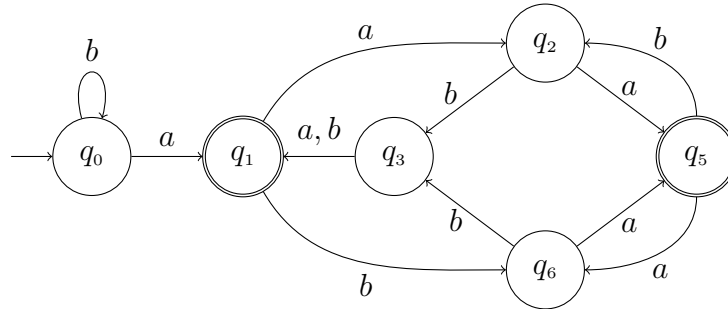
	q_0		
\times		q_1	
\times	\times		q_2
\times	\times	\times	q_3

en la cual todas las casillas han sido marcadas. Esto quiere decir que no hay pares de estados diferentes que sean equivalentes entre sí, o lo que es lo mismo, todo estado es equivalente solamente a sí mismo. Por lo tanto, el autómata no se puede minimizar más y no es posible aceptar el lenguaje L con menos de cuatro estados.

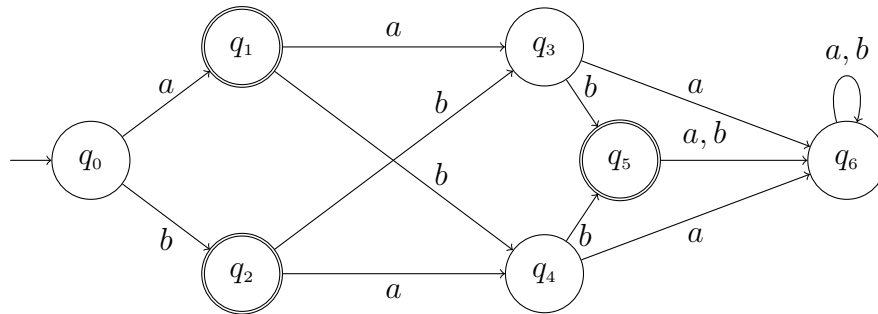
Ejercicios de la sección 2.16

- ① Minimizar los siguientes AFD, es decir, encontrar autómatas deterministas con el mínimo número de estados posible, equivalentes a los autómatas dados.

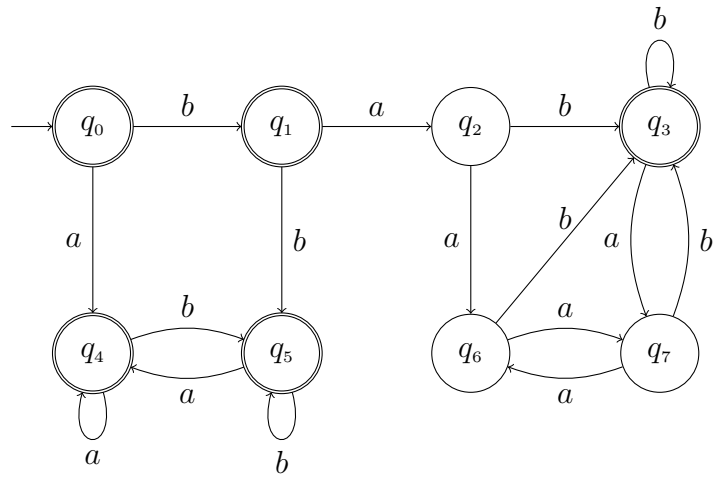
(i) Alfabeto $\Sigma = \{a, b\}$.



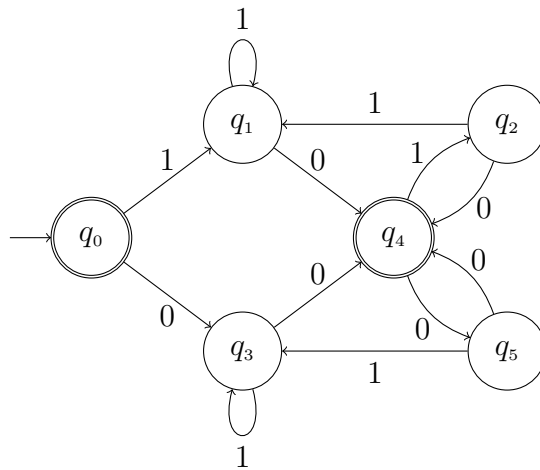
(ii) Alfabeto $\Sigma = \{a, b\}$.



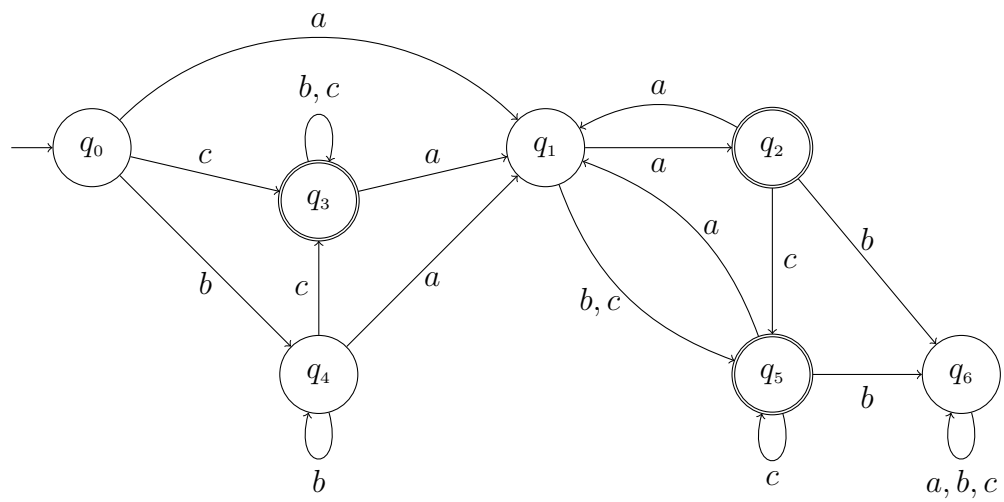
(iii) Alfabeto $\Sigma = \{a, b\}$.



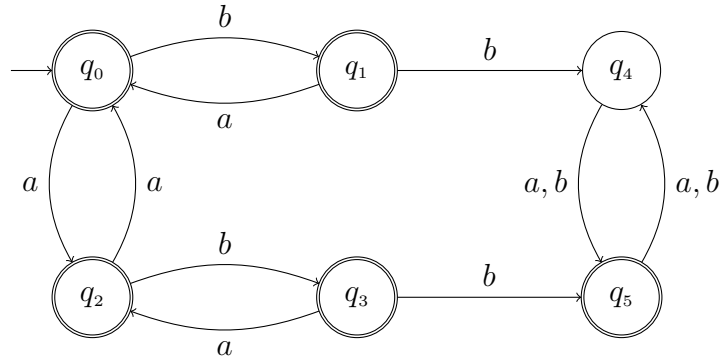
(iv) Alfabeto $\Sigma = \{0, 1\}$.



(v) Alfabeto $\Sigma = \{a, b, c\}$.



- (vi) El siguiente autómata fue obtenido utilizando el producto cartesiano para aceptar el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar o que no contienen dos b es consecutivas (sección 2.11).



- ② Sea $\Sigma = \{a, b\}$. Demostrar que el lenguaje $L = a^+b^*a$ no puede ser aceptado por ningún AFD con menos de seis estados (incluyendo el estado limbo).
- ③ Sea $\Sigma = \{a, b\}$. Demostrar que el lenguaje $L = a^*b \cup b^*a$ no puede ser aceptado por ningún AFD con menos de siete estados (incluyendo el estado limbo).