

Capítulo 1

Alfabetos, cadenas y lenguajes

De manera muy amplia podría decirse que la computación es la manipulación de entradas (*inputs*) para producir salidas (*outputs*). Pero cualquier mecanismo de cómputo solamente puede manipular un número finito de símbolos en un tiempo finito. Desde el punto de vista teórico esto impone dos restricciones naturales: el conjunto de símbolos (alfabeto) debe ser finito y se deben considerar únicamente cadenas (secuencias de símbolos) de longitud finita. Surgen así los ingredientes esenciales de una teoría abstracta de la computación: *alfabetos* y *cadenas*. Los conjuntos de cadenas (ya sean finitos o infinitos) se denominarán *lenguajes*.

1.1. Alfabetos y cadenas

Alfabetos. Un *alfabeto* es un conjunto finito no vacío cuyos elementos se llaman *símbolos*. Es costumbre usar las letras griegas mayúsculas Σ, Γ, Δ para denotar alfabetos. De ser necesario se utilizan también subíndices: $\Sigma_1, \Sigma_2, \Sigma_3, \dots$

Un alfabeto genérico Σ consta de k símbolos diferentes, s_1, s_2, \dots, s_k , donde $k \geq 1$; esto es, $\Sigma = \{s_1, s_2, \dots, s_k\}$. A continuación presentamos algunos de los alfabetos más conocidos.

Ejemplos

- El alfabeto latino (letras minúsculas, 26 símbolos).

$$\Sigma_1 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z\}.$$

- El alfabeto latino (letras mayúsculas, 26 símbolos).

$$\Sigma_2 = \{A, B, C, D, E, F, G, H, I, J, K, L, M, N, O, P, Q, R, S, T, U, V, W, X, Y, Z\}.$$

- El alfabeto latino (letras minúsculas y mayúsculas, 52 símbolos).
 $\Sigma_3 = \{a, A, b, B, c, C, d, D, e, E, \dots, w, W, x, X, y, Y, z, Z\}.$
- El alfabeto del idioma español o castellano (letras minúsculas) consta de los 26 símbolos del alfabeto latino más los símbolos acentuados *á, é, í, ó, ú, ñ*. Esto es,
 $\Sigma_4 = \{a, b, c, d, e, f, g, h, i, j, k, l, m, n, o, p, q, r, s, t, u, v, w, x, y, z, á, é, í, ó, ú, ñ\}.$
- El alfabeto griego (letras minúsculas, 24 símbolos).
 $\Sigma_5 = \{\alpha, \beta, \gamma, \delta, \epsilon, \eta, \iota, \kappa, \lambda, \mu, \nu, \rho, \sigma, \tau, \theta, \upsilon, \chi, \xi, \zeta, \omega\}.$
- El alfabeto de los dígitos (10 símbolos).
 $\Sigma_6 = \{1, 2, 3, 4, 5, 6, 7, 8, 9, 0\}.$
- El alfabeto $\Sigma = \{0, 1\}$ se conoce como *alfabeto binario*.

El alfabeto binario $\Sigma = \{0, 1\}$ se podría considerar como el más importante ya que cualquier otro alfabeto se puede codificar usando solamente ceros y unos. La noción de *codificación* no es relevante por el momento y se estudiará en detalle en el capítulo 6.

Cadenas. Una *cadena* o *palabra* sobre un alfabeto Σ dado es cualquier sucesión (o secuencia) finita de elementos de Σ . Según las convenciones estándares, una cadena se escribe de izquierda a derecha. Admitimos la existencia de una única cadena que no tiene símbolos, la cual se denomina *cadena vacía* y se denota con λ . La cadena vacía desempeña, en la teoría de la computación, un papel similar al del conjunto vacío \emptyset en la teoría de conjuntos.

Ejemplo Sea $\Sigma = \{a, b\}$ el alfabeto que consta de los dos símbolos a y b . Las siguientes son cadenas sobre Σ :

aba
ababaaa
aaaab.

Obsérvese que $aba \neq aab$ y que $aaab \neq baaa$. En una cadena hay que tener en cuenta el orden en el que aparecen los símbolos ya que las cadenas se definen como *sucesiones*, es decir, conjuntos *secuencialmente ordenados*.

Ejemplo Las cadenas sobre el alfabeto binario $\Sigma = \{0, 1\}$ son secuencias finitas de ceros y unos, llamadas *secuencias binarias*, tales como

001
 1011
 001000001.

En general, dado un alfabeto de referencia Σ , una cadena genérica es una secuencia de la forma $a_1 a_2 \cdots a_n$, donde $a_1, a_2, \dots, a_n \in \Sigma$. Para denotar cadenas concretas se utilizan letras como u, v, w, x, y, z , con subíndices si es necesario; si tales letras hacen parte del

alfabeto Σ , entonces se utilizan letras griegas o símbolos completamente diferentes a los de Σ .

Ejemplo Sea $\Sigma = \{a, b, c, d\}$. Utilizamos u, v, w para denotar (dar un nombre) a ciertas cadenas concretas:

$$u = ddaba.$$

$$v = cababadda.$$

$$w = bbbcc.$$

Longitud de una cadena. La *longitud* de una cadena $u \in \Sigma^*$ se denota $|u|$ y se puede definir descriptivamente como el número de símbolos de u (contando los símbolos repetidos). Es decir,

$$|u| = \begin{cases} 0, & \text{si } u = \lambda, \\ n, & \text{si } u = a_1 a_2 \cdots a_n, \text{ donde } a_1, a_2, \dots, a_n \in \Sigma \text{ y } n \geq 1. \end{cases}$$

Ejemplo Sea $\Sigma = \{a, b, c, d\}$. Si $u = bbcada \implies |u| = 6$. Si $v = ccddd \implies |v| = 5$.


Se puede observar que los símbolos de un alfabeto Σ juegan un doble papel: como símbolos del alfabeto de referencia y como cadenas de longitud uno.


Definición descriptiva de Σ^* . El conjunto de *todas* las cadenas sobre un alfabeto Σ , incluyendo la cadena vacía, se denota por Σ^* .


Ejemplo Sea $\Sigma = \{a, b, c\}$, entonces


$$\Sigma^* = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, \dots\}.$$

En este caso, Σ^* está formado por la cadena vacía λ , las 3 cadenas de longitud uno, a, b, c ; las 9 cadenas de longitud dos, $aa, ab, ac, ba, bb, bc, ca, cb, cc$; las 27 cadenas de longitud tres, $aaa, aab, abc, baa, \dots, ccc$, etc.

 Algunos autores denotan la cadena vacía con la letra griega ε . Preferimos denotarla con λ porque ε tiende a confundirse con el símbolo \in usado para la relación de pertenencia. También es frecuente usar Λ para denotar la cadena vacía.

 Si bien un alfabeto Σ es un conjunto finito, Σ^* es siempre un conjunto infinito (enumerable). En el caso más simple, Σ contiene solo un símbolo, por ejemplo, $\Sigma = \{a\}$, y $\Sigma^* = \{\lambda, a, aa, aaa, aaaa, aaaaa, \dots\}$.

 \emptyset y λ son objetos diferentes: \emptyset es un conjunto (el único conjunto que no tiene elementos) y λ es una cadena (la única cadena que no tiene símbolos).

 La mayor parte de la teoría de la computación se hace con referencia a un alfabeto Σ fijo (pero arbitrario).

1.2. Definiciones y demostraciones recursivas

En la Teoría de la Computación muchos conjuntos se definen recursivamente. A continuación presentamos este tipo de definiciones en un contexto muy amplio.

Definición recursiva de un conjunto. Un conjunto S se define recursivamente por medio de tres cláusulas:

- (1) Cláusula básica. Se especifican los elementos más sencillos (o básicos) de S .
- (2) Cláusula recursiva. Se especifican la regla o reglas para la formación de nuevos elementos de S a partir de elementos ya conocidos de S .
- (3) Cláusula de exclusión. Establece que los elementos de S se pueden obtener únicamente utilizando las cláusulas (1) y (2).

La cláusula de exclusión se suele admitir de manera implícita y es corriente omitirla en las definiciones recursivas concretas, algo que haremos en lo sucesivo.

Ejemplo Dado un alfabeto Σ , el conjunto Σ^* de todas las cadenas, definido descriptivamente en la sección 1.1, se puede definir recursivamente. La idea es que a partir de una cadena $u \in \Sigma^*$ se puede obtener una nueva cadena añadiendo a la derecha un símbolo a perteneciente al alfabeto Σ ; dicha cadena será denotada por ua . Todas las cadenas de Σ^* se pueden obtener de esta manera a partir de la cadena vacía λ ; se entiende que $\lambda a = a$, para cualquier símbolo $a \in \Sigma$.

La definición recursiva de Σ^* consta entonces de las siguientes dos cláusulas:

- (1) $\lambda \in \Sigma^*$.
- (2) Si $u \in \Sigma^*$ entonces $ua \in \Sigma^*$ para cada símbolo $a \in \Sigma$, donde ua es la secuencia de símbolos obtenida a partir de u añadiendo el símbolo a a la derecha.

Definición recursiva de conceptos sobre cadenas. La definición recursiva de Σ^* permite definir recursivamente nociones o conceptos aplicables a todas las cadenas. Sea Σ un alfabeto dado; para definir un concepto C sobre todas las cadenas en Σ^* se utilizan dos cláusulas:

- (1) Se define $C(\lambda)$, es decir, se define el concepto C para la cadena λ .
- (2) Se define $C(ua)$ en términos de $C(u)$ para toda cadena $u \in \Sigma^*$ y todo símbolo $a \in \Sigma$.

Ejemplo Aunque la noción de longitud de una cadena es muy simple y bastaría la descripción presentada en la sección 1.1, también podemos dar una definición recursiva de $|u|$, con $u \in \Sigma^*$:

- (1) $|\lambda| = 0$.
- (2) $|ua| = |u| + 1$ para toda $u \in \Sigma^*$ y todo símbolo $a \in \Sigma$.

Recursión sobre cadenas. Dado un alfabeto Σ , es posible demostrar recursivamente que todas las cadenas en Σ^* satisfacen una cierta propiedad P . A tal tipo de razonamiento recursivo se le conoce como *recursión sobre cadenas*, y consta de dos pasos:

- (1) Se demuestra $P(\lambda)$, es decir, se demuestra que la cadena vacía λ satisface la propiedad P .
- (2) Se demuestra la implicación

$$P(u) \implies P(ua), \text{ para toda } u \in \Sigma^* \text{ y todo } a \in \Sigma.$$

Es decir, se demuestra que si u satisface la propiedad P , entonces la cadena ua también satisface la propiedad P , para todo $a \in \Sigma$. $P(u)$ es la llamada *hipótesis recursiva*, a partir de la cual se demuestra $P(ua)$ para todo $a \in \Sigma$.

La recursión sobre cadenas es un razonamiento demostrativo similar a la llamada *inducción matemática* que se utiliza para demostrar propiedades sobre los números naturales. Estos argumentos son necesarios cuando se quiere demostrar propiedades con toda rigurosidad.

1.3. Concatenación de cadenas

Dado un alfabeto Σ y dos cadenas $u, v \in \Sigma^*$, la *concatenación de u y v* se denota como $u \cdot v$ o simplemente uv y se define descriptivamente así:

- (1) Si $v = \lambda$, entonces $u \cdot \lambda = \lambda \cdot u = u$. Es decir, la concatenación de cualquier cadena u con la cadena vacía, a izquierda o a derecha, es igual a u .
- (2) Si $u = a_1a_2 \cdots a_n$, $v = b_1b_2 \cdots b_m$, donde $a_1, \dots, a_n, b_1, \dots, b_m \in \Sigma$, $n, m \geq 1$, entonces

$$u \cdot v = a_1a_2 \cdots a_nb_1b_2 \cdots b_m.$$

Es decir, $u \cdot v$ es la cadena formada escribiendo los símbolos de u y a continuación los símbolos de v .

Según la definición, se tiene que $|u \cdot v| = |u| + |v|$. En general, $u \cdot v \neq v \cdot u$, es decir, la concatenación no es una operación conmutativa.

Ejemplo Sea $\Sigma = \{a, b, c, d\}$. Si $u = bcaad$ y $v = dcbb$, entonces $u \cdot v = bcaaddcbb$ y $v \cdot u = dcbbbcaad$.

Definición recursiva de la concatenación. Para definir recursivamente la concatenación $u \cdot v$ para todas las cadenas $u, v \in \Sigma^*$ se toma una cadena fija u (pero arbitraria) y se hace recursión sobre v , de la siguiente manera:

- (1) $u \cdot \lambda = \lambda \cdot u = u$.
- (2) $u \cdot (va) = (u \cdot v)a$, para toda $v \in \Sigma^*$, $a \in \Sigma$.

Propiedad asociativa de la concatenación. La concatenación de cadenas es una operación asociativa. Es decir, si $u, v, w \in \Sigma^*$, entonces

$$(u \cdot v) \cdot w = u \cdot (v \cdot w).$$

Demostración. Primero hay que considerar los casos en los que alguna (o algunas) de las cadenas u , v o w sea la cadena vacía λ . Por ejemplo, si $v = \lambda$ tenemos $(u \cdot v) \cdot w = (u \cdot \lambda) \cdot w = u \cdot w$ mientras que $u \cdot (v \cdot w) = u \cdot (\lambda \cdot w) = u \cdot w$.

Luego consideramos el caso en que u , v y w sean diferentes de λ . Entonces podemos escribir

$$\begin{aligned} u &= a_1 \cdots a_n, \text{ con } a_i \in \Sigma, n \geq 1. \\ v &= b_1 \cdots b_m, \text{ con } b_i \in \Sigma, m \geq 1. \\ w &= c_1 \cdots c_k, \text{ con } c_i \in \Sigma, k \geq 1. \end{aligned}$$

Entonces

$$\begin{aligned} (u \cdot v) \cdot w &= (a_1 \cdots a_n b_1 \cdots b_m) \cdot w = a_1 \cdots a_n b_1 \cdots b_m c_1 \cdots c_k. \\ u \cdot (v \cdot w) &= u \cdot (b_1 \cdots b_m c_1 \cdots c_k) = a_1 \cdots a_n b_1 \cdots b_m c_1 \cdots c_k. \end{aligned}$$

Esto demuestra la propiedad asociativa.

La propiedad asociativa también se puede demostrar por recursión sobre cadenas: se mantienen u y v fijas (pero arbitrarias) y se hace recursión sobre w . Para $w = \lambda$, se obtiene fácilmente que

$$(u \cdot v) \cdot w = (u \cdot v) \cdot \lambda = u \cdot v = u \cdot (v \cdot \lambda) = u \cdot (v \cdot w).$$

Para el paso recursivo, suponemos que $(u \cdot v) \cdot w = u \cdot (v \cdot w)$ y demostraremos que $(u \cdot v) \cdot (wa) = u \cdot (v \cdot (wa))$, para todo $a \in \Sigma$.

$$\begin{aligned} (u \cdot v) \cdot (wa) &= ((u \cdot v) \cdot w)a, & (\text{definición recursiva de la concatenación}) \\ &= (u \cdot (v \cdot w))a, & (\text{hipótesis recursiva}) \\ &= u \cdot ((v \cdot w)a), & (\text{definición recursiva de la concatenación}) \\ &= u \cdot (v \cdot (wa)), & (\text{definición recursiva de la concatenación}). \quad \square \end{aligned}$$

La propiedad asociativa permite escribir la concatenación $u \cdot v \cdot w = uvw$ sin necesidad de usar paréntesis. En general, la concatenación de n cadenas u_1, u_2, \dots, u_n de Σ^* (en ese orden) se puede escribir sin ambigüedad como $u_1 u_2 \cdots u_n$.

1.4. Potencias de una cadena

Dado un alfabeto Σ y una cadena $u \in \Sigma^*$, se define u^n , con $n \in \mathbb{N}$, en la siguiente forma:

$$\begin{aligned} u^0 &= \lambda, \\ u^n &= \underbrace{uu \cdots u}_{n \text{ veces}}, \text{ si } n \geq 1. \end{aligned}$$

Lo anterior también se puede escribir como:

$$u^n = \begin{cases} \lambda, & \text{si } n = 0, \\ \underbrace{uu \cdots u}_{n \text{ veces}}, & \text{si } n \geq 1. \end{cases}$$

Es decir, u^n es la concatenación de u consigo misma n veces, si $n \geq 1$. Puesto que n es un número natural, u^n se puede definir inductivamente sobre n :

$$\begin{aligned} u^0 &= \lambda, \\ u^{n+1} &= u^n \cdot u, \text{ para todo } n \geq 0. \end{aligned}$$

Un caso particular importante se presenta cuando la cadena u consta de un solo símbolo, por ejemplo, $u = a$, donde $a \in \Sigma$. Se tiene entonces

$$\begin{aligned} a^0 &= \lambda, \\ a^n &= \underbrace{aa \cdots a}_{n \text{ veces}}, \text{ si } n \geq 1. \end{aligned}$$

Ejemplo Sea $\Sigma = \{a, b, c, d\}$. Usando la propiedad asociativa y la potenciación de cadenas de longitud 1, podemos escribir, por ejemplo:

$$\begin{aligned} baaaddb &= ba^3d^2b. \\ dddbcccccaa &= d^3b^2c^5a^2. \\ ababbbc &= abab^3c = (ab)^2b^2c. \end{aligned}$$

1.5. Reflexión de una cadena

Dado Σ , la *reflexión* (o *inversa* o *reverso*) de una cadena $u \in \Sigma^*$ se denota u^R y se define descriptivamente así:

$$u^R = \begin{cases} \lambda, & \text{si } u = \lambda, \\ a_n \cdots a_2 a_1, & \text{si } u = a_1 a_2 \cdots a_n. \end{cases}$$

Es decir, u^R se obtiene escribiendo los símbolos de u de derecha a izquierda. De la definición se observa claramente que la reflexión de la reflexión de una cadena es la misma cadena, es decir,

$$(u^R)^R = u, \quad \text{para } u \in \Sigma^*.$$

También es fácil verificar que $(uv)^R = v^R u^R$ para todas las cadenas $u, v \in \Sigma^*$.



Algunos autores escriben u^{-1} en lugar de u^R para denotar la reflexión de una cadena u .

Ejercicios de la sección 1.5

- ① Dar una definición recursiva de u^R .
- ② Generalizar la propiedad $(uv)^R = v^R u^R$ a la concatenación de n cadenas, $n \geq 2$.

1.6. Subcadenas, prefijos y sufijos

Una cadena v es una *subcadena* o una *subpalabra* de u si existen cadenas x, y tales que $u = xvy$. Nótese que x o y pueden ser λ y, por lo tanto, la cadena vacía es una subcadena de cualquier cadena y toda cadena es subcadena de sí misma (ya que $u = \lambda u \lambda$).

Un *prefijo* de u es una cadena v tal que $u = vw$ para alguna cadena $w \in \Sigma^*$. Se dice que v es un *prefijo propio* de u si $v \neq u$.

Similarmente, un *sufijo* de u es una cadena v tal que $u = wv$ para alguna cadena $w \in \Sigma^*$. Se dice que v es un *sufijo propio* de u si $v \neq u$.

Obsérvese que λ es un prefijo y un sufijo de toda cadena u ya que $u\lambda = \lambda u = u$. Por la misma razón, toda cadena u es prefijo y sufijo de sí misma.

Ejemplo Sean $\Sigma = \{a, b, c, d\}$ y $u = bcbaadb$.

Prefijos de u :

λ
 b
 bc
 $bc b$
 $bcba$
 $bcbaa$
 $bcbaad$
 $bcbaadb$

Sufijos de u :

λ
 b
 db
 adb
 $aadb$
 $baadb$
 $cbaadb$
 $bcbaadb$

Algunas subcadenas de u : $baad$, aa , cba , db .

1.7. Orden lexicográfico entre cadenas

Si los símbolos de un alfabeto Σ tienen un orden preestablecido, sobre el conjunto Σ^* de todas las cadenas se puede definir un orden total, denominado *orden lexicográfico entre cadenas*. Concretamente, sea $\Sigma = \{s_1, s_2, \dots, s_k\}$ un alfabeto dado en el cual los símbolos tienen un orden predeterminado, $s_1 < s_2 < \dots < s_k$. En el conjunto Σ^* de todas las cadenas se define el orden lexicográfico, también denotado $<$, de la siguiente manera. Sean u, v dos cadenas en Σ^* ,

$$u = a_1 a_2 \cdots a_m, \text{ donde } a_i \in \Sigma, \text{ para } 1 \leq i \leq m.$$

$$v = b_1 b_2 \cdots b_n, \text{ donde } b_i \in \Sigma, \text{ para } 1 \leq i \leq n.$$

Se define $u < v$ si

- (1) $|u| < |v|$ (es decir, si $m < n$) o,
- (2) $|u| = |v|$ (es decir, si $m = n$) y para algún índice i , $1 \leq i \leq m$, se cumple que

$$a_1 = b_1, a_2 = b_2, \dots, a_{i-1} = b_{i-1}, a_i < b_i.$$

Es otros términos, las cadenas de Σ^* se ordenan inicialmente por longitud y si las cadenas u y v tienen la misma longitud, $|u| = |v| = n$, se establece el orden

$$a_1 a_2 \cdots a_{i-1} a_i a_{i+1} \cdots a_n < a_1 a_2 \cdots a_{i-1} b_i b_{i+1} \cdots b_n,$$

cuando $a_i < b_i$. Se define también $\lambda < u$, para toda cadena no vacía u .

El orden lexicográfico entre cadenas es un orden lineal, esto es, para todo par de cadenas diferentes u, v en Σ^* se tiene $u < v$ o $v < u$, y es el orden utilizado para listar las palabras en los diccionarios de los lenguajes naturales.

Ejemplo Sea $\Sigma = \{a, b, c\}$, donde los símbolos tienen el orden preestablecido $a < b < c$. Las primeras cadenas de Σ^* listadas en el orden lexicográfico son:

$\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, acb, acc, baa, bab, bac, bba, bbb, bbc, bca, bcb, bcc, caa, cab, cac, cba, cbb, cbc, cca, ccb, ccc, aaaa, aaab, aaac, \dots$

Ejercicios de la sección 1.7

Sea $\Sigma = \{a, b, c, d\}$, donde los símbolos tienen el orden preestablecido $a < b < c < d$. Para cada cadena dada u en Σ^* hallar la cadena que sigue inmediatamente a u en el orden lexicográfico.

$$(1) \quad u = dbd.$$

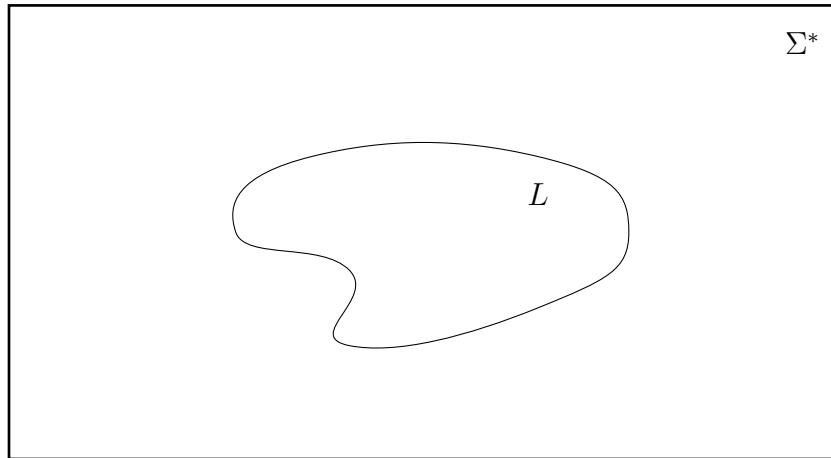
$$(3) \quad u = dabcd.$$

$$(2) \quad u = acbd.$$

$$(4) \quad u = dcadd.$$

1.8. Lenguajes

Un *lenguaje* L sobre un alfabeto Σ es un subconjunto de Σ^* , es decir $L \subseteq \Sigma^*$.



En otras palabras, un lenguaje es un conjunto de cadenas. Hay dos casos extremos:

$$\begin{aligned} L &= \emptyset, & \text{lenguaje vacío.} \\ L &= \Sigma^*, & \text{lenguaje de todas las cadenas sobre } \Sigma. \end{aligned}$$

Todo lenguaje L satisface $\emptyset \subseteq L \subseteq \Sigma^*$, y puede ser finito o infinito. Los lenguajes se denotan corrientemente con letras mayúsculas $A, B, C, \dots, L, M, N, \dots$.

Si P es una propiedad referente a las cadenas de Σ^* , el lenguaje L de todas las cadenas que satisfacen la propiedad P se denotará como $L = \{u \in \Sigma^* : P(u)\}$.

Ejemplos Los siguientes son ejemplos de lenguajes sobre los alfabetos especificados.

- $\Sigma = \{a, b, c\}$. $L = \{a, aba, aca\}$.
- $\Sigma = \{a, b, c\}$. $L = \{\lambda, a, aa, aaa, \dots\} = \{a^n : n \geq 0\}$.
- $\Sigma = \{a, b, c\}$. $L = \{ac, abc, ab^2c, ab^3c, \dots\} = \{ab^n c : n \geq 0\}$.
- $\Sigma = \{a, b\}$. $L =$ lenguaje de todas las cadenas de longitud 2 $= \{u \in \Sigma^* : |u| = 2\} = \{aa, ab, ba, bb\}$.
- $\Sigma = \{a, b, c\}$. $L =$ lenguaje de todas las cadenas de longitud 2 $= \{u \in \Sigma^* : |u| = 2\}$. Este lenguaje tiene nueve cadenas; explícitamente, $L = \{aa, ab, ac, ba, bb, bc, ca, cb, cc\}$.
- $\Sigma = \{a, b, c, \dots, x, y, z, á, é, í, ó, ú, ñ, A, B, C, \dots, X, Y, Z, Á, É, Í, Ó, Ú, Ñ\}$. Sea L el lenguaje de todas las palabras listadas en el diccionario RAE (Real Academia Española). L es un lenguaje finito.
- $\Sigma = \{a, b, c\}$. $L = \{u \in \Sigma^* : u \text{ no contiene el símbolo } c\}$. Por ejemplo, $abbaab \in L$ pero $abbcaa \notin L$.
- $\Sigma = \{0, 1\}$. $L =$ conjunto de todas las secuencias binarias que contienen un número impar de unos.
- $\Sigma = \{0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}$. El conjunto \mathbb{N} de los números naturales se puede definir como un lenguaje sobre Σ , en la siguiente forma:

$$\mathbb{N} = \{u \in \Sigma^* : u \neq \lambda \text{ y } (u = 0 \text{ ó } 0 \text{ no es un prefijo de } u)\}.$$

Como ejercicio, el estudiante puede definir el conjunto de los enteros

$$\mathbb{Z} = \{\dots, -2, -1, 0, 1, 2, \dots\}$$

como un lenguaje sobre un alfabeto adecuado.

1.9. Operaciones entre lenguajes

Puesto que los lenguajes sobre Σ son subconjuntos de Σ^* , las operaciones usuales entre conjuntos son también operaciones válidas entre lenguajes. Así, si A y B son lenguajes sobre Σ (es decir $A, B \subseteq \Sigma^*$), entonces los siguientes también son lenguajes sobre Σ :

$$\begin{aligned} A \cup B &= \{u \in \Sigma^* : u \in A \vee u \in B\} && \text{Unión.} \\ A \cap B &= \{u \in \Sigma^* : u \in A \wedge u \in B\} && \text{Intersección.} \\ A - B &= \{u \in \Sigma^* : u \in A \wedge u \notin B\} && \text{Diferencia.} \\ \overline{A} &= \Sigma^* - A = \{u \in \Sigma^* : u \notin A\} && \text{Complemento.} \end{aligned}$$

La diferencia $A - B$ también se suele escribir como $A \setminus B$. En estas definiciones se utilizan las conectivas \vee (disyunción no-excluyente) y \wedge (conjunción). Otra conectiva útil es \veebar (disyunción excluyente) con la cual se define la llamada diferencia simétrica:

$$\begin{aligned} A \nabla B &= \{u \in \Sigma^* : u \in A \veebar u \in B\} = \{u \in \Sigma^* : u \in A \text{ o } u \in B, \text{ pero no ambas}\} \\ &= \{u \in \Sigma^* : \text{o bien } u \in A, \text{ o bien } u \in B\} \\ &= (A \cup B) - (A \cap B) = (A - B) \cup (B - A). \end{aligned}$$

Estas operaciones entre lenguajes se llaman *operaciones conjuntistas* o *booleanas* para distinguirlas de las *operaciones lingüísticas* (concatenación, potenciación, reflexión, clausura) que son extensiones a los lenguajes de las correspondientes operaciones entre cadenas.

1.10. Concatenación de lenguajes

La *concatenación* de dos lenguajes A y B sobre Σ , notada $A \cdot B$ o simplemente AB se define como

$$A \cdot B = AB = \{uv : u \in A, v \in B\}.$$

En general, $AB \neq BA$.

Ejemplo Si $\Sigma = \{a, b, c\}$, $A = \{a, ab, ac\}$, $B = \{b, b^2\}$, entonces

$$\begin{aligned} AB &= \{ab, ab^2, ab^2, ab^3, acb, acb^2\} = \{ab, ab^2, ab^3, acb, acb^2\}. \\ BA &= \{ba, bab, bac, b^2a, b^2ab, b^2ac\}. \end{aligned}$$

Nótese que AB tiene cinco cadenas en total ya que la cadena ab^2 se obtiene de dos maneras distintas. Por otro lado, BA consta de seis cadenas en total.

Ejemplo Si $\Sigma = \{a, b, c\}$, $A = \{ba, bc\}$, $B = \{b^n : n \geq 0\}$, entonces

$$\begin{aligned} AB &= \{bab^n : n \geq 0\} \cup \{bcb^n : n \geq 0\}. \\ BA &= \{b^n ba : n \geq 0\} \cup \{b^n bc : n \geq 0\} \\ &= \{b^{n+1}a : n \geq 0\} \cup \{b^{n+1}c : n \geq 0\} \\ &= \{b^m a : m \geq 1\} \cup \{b^m c : m \geq 1\}. \end{aligned}$$

Propiedades de la concatenación de lenguajes. Sean A, B, C lenguajes sobre Σ , es decir $A, B, C \subseteq \Sigma^*$. Entonces

$$1. A \cdot \emptyset = \emptyset \cdot A = \emptyset.$$

$$2. A \cdot \{\lambda\} = \{\lambda\} \cdot A = A.$$

3. Propiedad Asociativa,

$$A \cdot (B \cdot C) = (A \cdot B) \cdot C.$$

4. Distributividad de la concatenación con respecto a la unión,

$$\begin{aligned} A \cdot (B \cup C) &= A \cdot B \cup A \cdot C. \\ (B \cup C) \cdot A &= B \cdot A \cup C \cdot A. \end{aligned}$$

5. Propiedad distributiva generalizada. Si $\{B_i\}_{i \in I}$ es una familia indexada de lenguajes sobre Σ (o sea, $B_i \subseteq \Sigma^*$ para todo $i \in I$), entonces

$$\begin{aligned} A \cdot \bigcup_{i \in I} B_i &= \bigcup_{i \in I} (A \cdot B_i), \\ \left(\bigcup_{i \in I} B_i \right) \cdot A &= \bigcup_{i \in I} (B_i \cdot A). \end{aligned}$$

Por ejemplo, si el conjunto de índices I es $\mathbb{N} = \{0, 1, 2, \dots\}$, podemos escribir

$$A \cdot \bigcup_{i \in \mathbb{N}} B_i = A \cdot (B_0 \cup B_1 \cup B_2 \cup \dots) = A \cdot B_0 \cup A \cdot B_1 \cup A \cdot B_2 \cup \dots = \bigcup_{i \in \mathbb{N}} (A \cdot B_i).$$

Demostración.

$$1. A \cdot \emptyset = \{uv : u \in A, v \in \emptyset\} = \emptyset.$$

$$2. A \cdot \{\lambda\} = \{uv : u \in A, v \in \{\lambda\}\} = \{u : u \in A\} = A.$$

3. Se sigue de la asociatividad de la concatenación de cadenas.

4. Caso particular de la propiedad general, demostrada a continuación.

5. Demostración de la igualdad $A \cdot \bigcup_{i \in I} B_i = \bigcup_{i \in I} (A \cdot B_i)$:

$$\begin{aligned} x \in A \cdot \bigcup_{i \in I} B_i &\iff x = u \cdot v, \quad \text{con } u \in A \text{ \& } v \in \bigcup_{i \in I} B_i \\ &\iff x = u \cdot v, \quad \text{con } u \in A \text{ \& } v \in B_j, \text{ para algún } j \in I \\ &\iff x \in A \cdot B_j, \quad \text{para algún } j \in I \\ &\iff x \in \bigcup_{i \in I} (A \cdot B_i). \end{aligned}$$

La igualdad $\left(\bigcup_{i \in I} B_i \right) \cdot A = \bigcup_{i \in I} (B_i \cdot A)$ se demuestra de forma similar. □

- ✎ La propiedad asociativa permite escribir concatenaciones de tres o más lenguajes sin necesidad de usar paréntesis. Por ejemplo, ABC o $ABCD$.
- ✎ En general, no se cumple que $A \cdot (B \cap C) = A \cdot B \cap A \cdot C$. Es decir, la concatenación no es distributiva con respecto a la intersección. Contraejemplo: Sea $\Sigma = \{a\}$, $A = \{a, \lambda\}$, $B = \{\lambda\}$, $C = \{a\}$. Se tiene que $A \cdot (B \cap C) = \{a, \lambda\} \cdot \emptyset = \emptyset$. Por otro lado, $A \cdot B \cap A \cdot C = \{a, \lambda\} \cdot \{\lambda\} \cap \{a, \lambda\} \cdot \{a\} = \{a, \lambda\} \cap \{a^2, a\} = \{a\}$.

Ejercicios de la sección 1.10

- ① Dar un ejemplo de un alfabeto Σ y dos lenguajes diferentes A, B sobre Σ tales que $AB = BA$.
- ② Dar un ejemplo de un alfabeto Σ y tres lenguajes A, B, C sobre Σ , diferentes entre sí, tales que $A \cdot (B \cap C) = A \cdot B \cap A \cdot C$.
- ③ Sea Σ un alfabeto dado. Se ha demostrado en esta sección, mediante un contraejemplo, que la igualdad $A \cdot B \cap A \cdot C = A \cdot (B \cap C)$ no es una identidad válida para todos los lenguajes $A, B, C \subseteq \Sigma^*$. Demostrar, sin embargo, que la contención

$$A \cdot (B \cap C) \subseteq A \cdot B \cap A \cdot C$$

es siempre válida para todos los lenguajes $A, B, C \subseteq \Sigma^*$.

1.11. Potencias de un lenguaje

Dado un lenguaje A sobre Σ , ($A \subseteq \Sigma^*$), y un número natural $n \in \mathbb{N}$, se define A^n en la siguiente forma

$$A^0 = \{\lambda\},$$

$$A^n = \underbrace{AA \cdots A}_{n \text{ veces}} = \{u_1 \cdots u_n : u_i \in A, \text{ para todo } i, 1 \leq i \leq n\}, \text{ para } n \geq 1.$$

Las potencias de A se pueden definir por inducción sobre n :

$$A^0 = \{\lambda\},$$

$$A^{n+1} = A^n \cdot A, \text{ para todo } n \geq 0.$$

Se tiene entonces que $A^1 = A$ y A^2 es el conjunto de las concatenaciones dobles de cadenas de A :

$$A^2 = \{uv : u, v \in A\}.$$

A^3 está formado por las concatenaciones triples:

$$A^3 = \{uvw : u, v, w \in A\}$$

En general, A^n es el conjunto de todas las concatenaciones de n cadenas de A , de todas las formas posibles:

$$A^n = \{u_1 \cdots u_n : (\forall i, 1 \leq i \leq n)(u_i \in A)\}.$$

1.12. La clausura de Kleene de un lenguaje

La *clausura de Kleene* o *estrella de Kleene* o simplemente la *estrella* de un lenguaje A , $A \subseteq \Sigma^*$, es la unión de todas las potencias de A y se denota por A^* .

(Descripción 1)

$$A^* = \bigcup_{n \geq 0} A^n = A^0 \cup A^1 \cup A^2 \cup \cdots \cup A^n \cdots$$

Según la definición de las potencias de un lenguaje, A^* consta de todas las concatenaciones de cadenas de A consigo mismas, de todas las formas posibles. Tenemos así una descripción explícita de A^* :

(Descripción 2)

$$\begin{aligned} A^* &= \text{conjunto de } \textit{todas} \text{ las concatenaciones} \\ &\quad \text{de cadenas de } A, \text{ incluyendo } \lambda \\ &= \{u_1 \cdots u_n : (\forall i, 1 \leq i \leq n)(u_i \in A), \quad n \geq 0\}. \end{aligned}$$

De manera similar se define la *clausura positiva* de un lenguaje A , $A \subseteq \Sigma^*$, denotada por A^+ .

$$A^+ = \bigcup_{n \geq 1} A^n = A^1 \cup A^2 \cup \cdots \cup A^n \cdots$$

A^+ se puede describir explícitamente de la siguiente manera

$$\begin{aligned} A^+ &= \text{conjunto de } \textit{todas} \text{ las concatenaciones de cadenas de } A \\ &= \{u_1 \cdots u_n : (\forall i, 1 \leq i \leq n)(u_i \in A), \quad n \geq 1\}. \end{aligned}$$

También es posible dar una definición recursiva de A^* :

- (1) $\lambda \in A^*$.
- (2) Si $u \in A^*$ y $v \in A$ entonces $u \cdot v \in A^*$.

En definitiva, A^* es todo lo que se puede obtener comenzando con λ y concatenando con cadenas de A . La definición recursiva de A^+ es similar, excepto que sus elementos básicos son las cadenas de A :

- (1) Si $u \in A$ entonces $u \in A^+$.
- (2) Si $u \in A^+$ y $v \in A$ entonces $u \cdot v \in A^+$.


Ejemplos Sea $\Sigma = \{a, b, c\}$.

$$\{a\}^* = \{\lambda, a, a^2, a^3, \dots\}.$$

$$\{a\}^+ = \{a, a^2, a^3, \dots\}.$$

$$\{cb, ba\}^* = \{\lambda, cb, ba, cbba, bacb, (cb)^2, (ba)^2, cbc bba, cbbacb, cbbaba, (cb)^3, \\ bacbba, babacb, bacbcb, (ba)^3, \dots\}.$$

$$\{a, b, c\}^* = \{\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, baa, bab, bac, \dots\} \\ = \text{Lenguaje de todas las cadenas sobre el alfabeto } \Sigma.$$

 Dado un alfabeto Σ , se han presentado dos definiciones de Σ^* :

Σ^* = conjunto de las cadenas sobre el alfabeto Σ .

Σ^* = conjunto de todas las concatenaciones de cadenas de Σ , considerando a Σ como el lenguaje de las cadenas de longitud 1.

No hay conflicto de notaciones porque las dos definiciones anteriores de Σ^* dan lugar al mismo conjunto.

Propiedades de $*$ y $+$. Sea A un lenguaje sobre Σ , es decir, $A \subseteq \Sigma^*$.

1. $A^* = A^+ \cup \{\lambda\}$.
2. $A^* = A^+$ si y solamente si $\lambda \in A$.
3. $A^+ = A^* \cdot A = A \cdot A^*$.
4. $A^* \cdot A^* = A^*$.
5. $(A^*)^n = A^*$, para todo $n \geq 1$.
6. $A^+ \cdot A^+ \subseteq A^+$. En general, $A^+ \cdot A^+ \neq A^+$.
7. $(A^*)^* = A^*$.
8. $(A^*)^+ = A^*$.
9. $(A^+)^* = A^*$.
10. $(A^+)^+ = A^+$.

Demostración.

Las propiedades 1 y 2 se siguen inmediatamente de las definiciones de A^* y A^+ .

$$\begin{aligned} 3. \quad A \cdot A^* &= A \cdot (A^0 \cup A^1 \cup A^2 \cup \dots) \\ &= A^1 \cup A^2 \cup A^3 \cup \dots \\ &= A^+. \end{aligned}$$

Similarmente se demuestra que $A^* \cdot A = A^+$.

4. Si $x \in A^* \cdot A^*$, entonces $x = u \cdot v$, con $u \in A^*$, $v \in A^*$. De modo que, $x = u \cdot v$, con $u = u_1 u_2 \cdots u_n$, donde $u_i \in A$ para $1 \leq i \leq n$, $n \geq 0$, y $v = v_1 v_2 \cdots v_m$, donde $v_i \in A$ para $1 \leq i \leq m$, $m \geq 0$. Se deduce que

$$x = u \cdot v = u_1 \cdot u_2 \cdots u_n \cdot v_1 \cdot v_2 \cdots v_m.$$

Por lo tanto, x es una concatenación de $n + m$ cadenas de A . Así que $x \in A^*$.

Recíprocamente, si $x \in A^*$, entonces $x = x \cdot \lambda \in A^* \cdot A^*$.

Hemos probado que $x \in A^* \cdot A^* \iff x \in A^*$, lo cual establece la igualdad.

5. Se sigue de la propiedad 4.

6. La demostración de $A^+ \cdot A^+ \subseteq A^+$ es similar a la demostración de $A^* \cdot A^* \subseteq A^*$ presentada en la propiedad 4, pero tomando $m, n \geq 1$.

Para exhibir un contraejemplo de $A^+ \cdot A^+ = A^+$, basta considerar $\Sigma = \{a\}$ y $A = \{a\}$. Se tiene que

$$\begin{aligned} A^+ &= A^1 \cup A^2 \cup \cdots = \{a\} \cup \{aa\} \cup \{aaa\} \cup \cdots = \{a^n : n \geq 1\}. \\ A^+ \cdot A^+ &= \{a, a^2, a^3, \dots\} \cdot \{a, a^2, a^3, \dots\} = \{a^2, a^3, a^4, \dots\} = \{a^n : n \geq 2\}. \end{aligned}$$

Se observa que $A^+ \cdot A^+ \neq A^+$.

$$\begin{aligned} 7. \quad (A^*)^* &= (A^*)^0 \cup (A^*)^1 \cup (A^*)^2 \cup \cdots \\ &= \{\lambda\} \cup A^* \cup A^* \cup A^* \cup \cdots \\ &= A^*. \end{aligned}$$

$$\begin{aligned} 8. \quad (A^*)^+ &= (A^*)^1 \cup (A^*)^2 \cup (A^*)^3 \cup \cdots \\ &= A^* \cup A^* \cup A^* \cup \cdots \\ &= A^*. \end{aligned}$$

$$\begin{aligned} 9. \quad (A^+)^* &= (A^+)^0 \cup (A^+)^1 \cup (A^+)^2 \cup \cdots \\ &= \{\lambda\} \cup A^+ \cup A^+ A^+ \cup \cdots \\ &= A^* \cup (\text{conjuntos contenidos en } A^+) \\ &= A^*. \end{aligned}$$

$$\begin{aligned} 10. \quad (A^+)^+ &= (A^+)^1 \cup (A^+)^2 \cup (A^+)^3 \cup \cdots, \\ &= A^+ \cup (\text{conjuntos contenidos en } A^+) \\ &= A^+. \end{aligned}$$

Otras propiedades de $*$. Sean A, B lenguajes sobre Σ , es decir, $A, B \subseteq \Sigma^*$.

1. Si $A \subseteq B$ entonces $A^* \subseteq B^*$.
2. $(A \cup B)^* = (A^*B^*)^* = (B^*A^*)^*$.

Demostración.

1. Se sigue inmediatamente de las definiciones de A^* y B^* .
2. Demostraremos la igualdad $(A \cup B)^* = (A^*B^*)^*$. La contención \subseteq se sigue de

$$\begin{aligned} A \subseteq A^*, \quad B \subseteq B^* &\implies A \subseteq A^*B^*, \quad B \subseteq A^*B^* \\ &\implies A \cup B \subseteq A^*B^* \\ &\implies (A \cup B)^* \subseteq (A^*B^*)^* \quad \text{por la propiedad 1.} \end{aligned}$$

Ahora deduciremos la contención \supseteq . Puesto que $A \subseteq A \cup B$ y $B \subseteq A \cup B$, de la propiedad 1 se obtiene que $A^* \subseteq (A \cup B)^*$ y $B^* \subseteq (A \cup B)^*$; se concluye que $A^*B^* \subseteq (A \cup B)^*(A \cup B)^* = (A \cup B)^*$. Por la propiedad 1 se tiene entonces que

$$(A^*B^*)^* \subseteq ((A \cup B)^*)^* = (A \cup B)^*. \quad \square$$

Ejercicios de la sección 1.12

Sean A, B lenguajes sobre Σ , es decir, $A, B \subseteq \Sigma^*$. Explicar por qué las siguientes igualdades no son válidas en general:

- ① $(A \cup B)^* = A^* \cup B^*$.
- ② $(A \cup B)^* = A^* \cup B^* \cup A^*B^* \cup B^*A^*$.

1.13. Reflexión o inverso de un lenguaje

Dado A un lenguaje sobre Σ , se define A^R de la siguiente forma:

$$A^R = \{u^R : u \in A\}.$$

A^R se denomina la *reflexión*, o el *inverso* de A , o el *reverso* de A .

Propiedades. Sean A y B lenguajes sobre Σ , es decir, $A, B \subseteq \Sigma^*$.

1. $(A \cdot B)^R = B^R \cdot A^R$.
2. $(A \cup B)^R = A^R \cup B^R$.
3. $(A \cap B)^R = A^R \cap B^R$.

$$4. (A^R)^R = A.$$

$$5. (A^*)^R = (A^R)^*.$$

$$6. (A^+)^R = (A^R)^+.$$

Demostración. Demostraremos las propiedades 1 y 5; las demás se dejan como ejercicio (opcional) para el estudiante.

$$\begin{aligned}
 1. \quad x \in (A \cdot B)^R &\iff x = u^R, \text{ donde } u \in A \cdot B \\
 &\iff x = u^R, \text{ donde } u = vw, \ v \in A, w \in B \\
 &\iff x = (vw)^R, \text{ donde } v \in A, w \in B \\
 &\iff x = w^R v^R, \text{ donde } v \in A, w \in B \\
 &\iff x \in B^R \cdot A^R.
 \end{aligned}$$

$$\begin{aligned}
 5. \quad x \in (A^*)^R &\iff x = u^R, \text{ donde } u \in A^* \\
 &\iff x = (u_1 \cdot u_2 \cdots u_n)^R, \text{ donde los } u_i \in A, \ n \geq 0 \\
 &\iff x = u_n^R \cdot u_{n-1}^R \cdots u_1^R, \text{ donde los } u_i \in A, \ n \geq 0 \\
 &\iff x \in (A^R)^*.
 \end{aligned}$$

Ejercicios de la sección 1.13

- ① Demostrar las propiedades 2, 3, 4 y 6 de la reflexión de cadenas.
- ② ¿Se pueden generalizar las propiedades 2 y 3 anteriores para uniones e intersecciones de un número arbitrario de conjuntos, respectivamente?

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 , $\{\lambda\}$, $\{a\}$, $a \in \Sigma$, por medio de 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:

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

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; en la sección 2.13 resolveremos este problema utilizando autómatas.

2.3. Autómatas finitos deterministas (AFD)

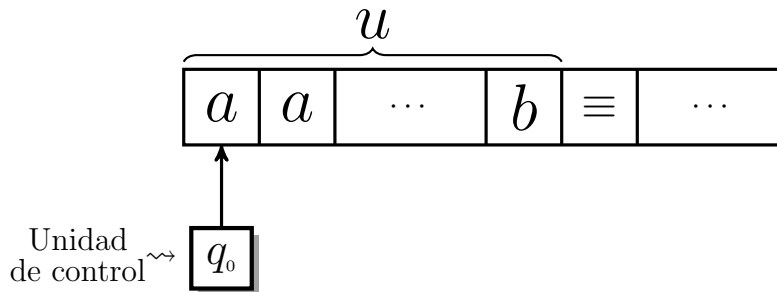
Los *autómatas finitos* son máquinas abstractas que leen de izquierda a derecha cadenas de entrada, las cuales son aceptadas o rechazadas. Un autómata actúa leyendo los símbolos escritos sobre una cinta semi-infinita, dividida en celdas o casillas, sobre la cual se escribe una cadena de entrada u , un 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:

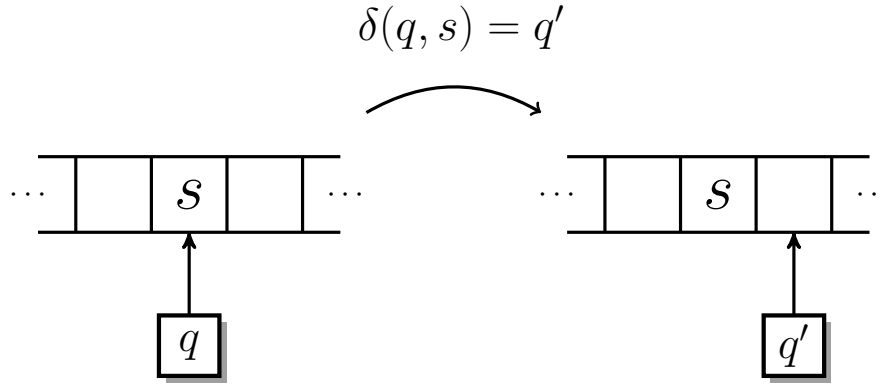
1. Un alfabeto Σ , llamado alfabeto de entrada o alfabeto de cinta. Todas las cadenas que lee M pertenecen a Σ^* .
2. $Q = \{q_0, q_1, \dots, q_n\}$, el conjunto (finito) de los estados internos de la unidad de control.
3. $q_0 \in Q$, estado inicial.
4. $F \subseteq Q$, conjunto de estados finales o 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

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

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

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

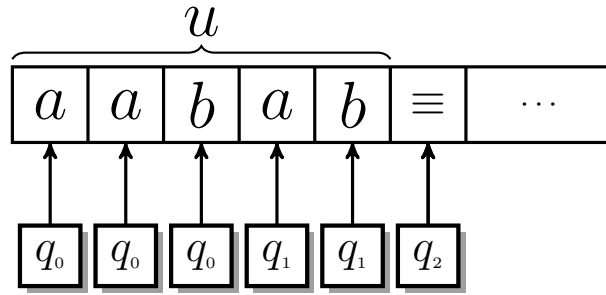
$$\delta(q_1, b) = q_2$$

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

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

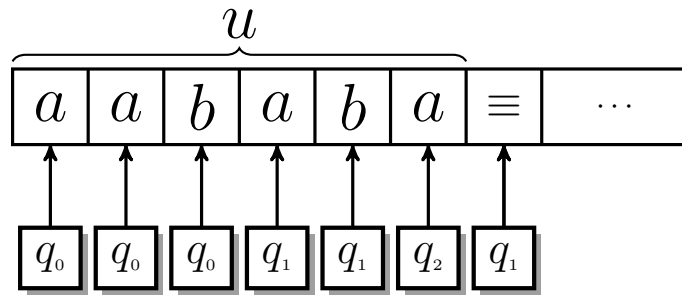
Vamos a ilustrar el procesamiento de tres cadenas de entrada.

1. $u = aabab$.



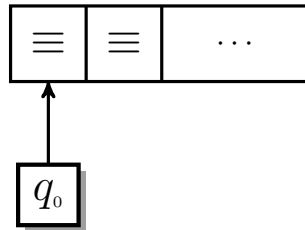
Como q_2 es un estado de aceptación, la cadena de entrada u es aceptada.

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.


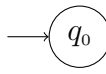

Un autómata M es entonces un mecanismo que clasifica todas las cadenas de Σ^* en dos clases disjuntas: las que son aceptadas y las que son rechazadas. El conjunto de las cadenas aceptadas por M se llama *lenguaje aceptado* o *lenguaje reconocido* por M y se denota por $L(M)$. Es decir,

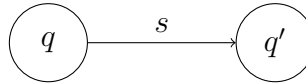
$$\begin{aligned} L(M) &:= \{u \in \Sigma^* : u \text{ es aceptada por } M\} \\ &= \{u \in \Sigma^* : M \text{ termina el procesamiento de } u \text{ en un estado } q \in F\} \end{aligned}$$

2.4. Grafo de un autómata

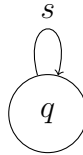
Un autómata finito se puede representar por medio de un grafo dirigido y etiquetado. Recuerdese que un grafo es un conjunto de vértices o nodos unidos por arcos o conectores; si los arcos tienen tanto dirección como etiquetas, el grafo se denomina *grafo dirigido y etiquetado* o *digrafo etiquetado*.

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

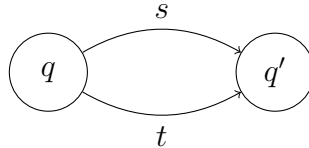
- Los vértices o nodos son los estados del autómata.
- Un estado $q \in Q$ se representa por un círculo con nombre q : 
- El estado inicial q_0 se destaca mediante una “bandera” o “flecha”: 
- Un estado de aceptación q se representa por un círculo doble con nombre q : 
- La transición $\delta(q, s) =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



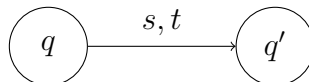
Si $\delta(q, s) = q$ se obtiene lo que se llama un “bucle” (*loop*, en inglés):



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

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

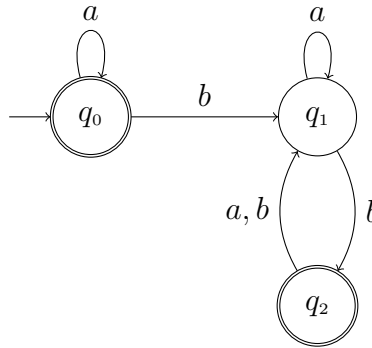
$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_1$

$\delta(q_2, b) = q_1$

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.

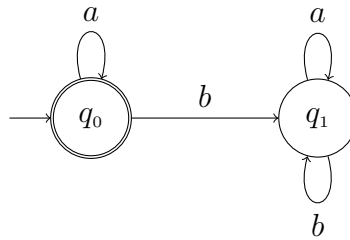
2.5. Diseño de autómatas

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:

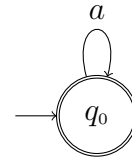
1. Si una cadena u es aceptada por M , entonces u debe pertenecer a L .
2. Recíprocamente, si $u \in L$, entonces u debe ser aceptada por M .

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.

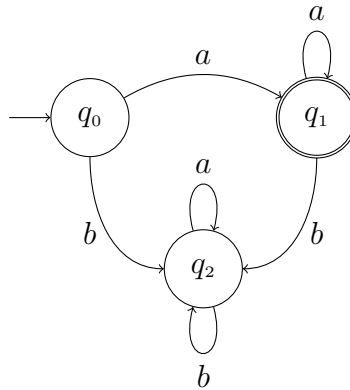
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^* = \{\lambda, a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



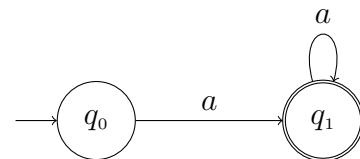
En la versión simplificada omitimos el estado limbo q_1 :



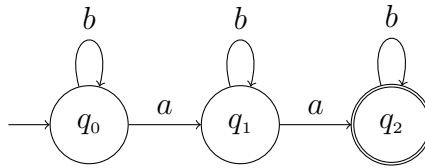
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^+ = \{a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



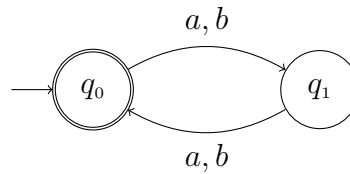
En la versión simplificada omitimos el estado limbo q_2 :



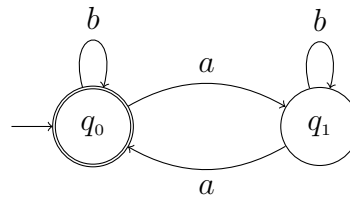
Ejemplo $\Sigma = \{a, b\}$. $L =$ lenguaje de las cadenas que contienen exactamente dos a 's $= b^*ab^*ab^*$. AFD M tal que $L(M) = L$, omitiendo el estado limbo:



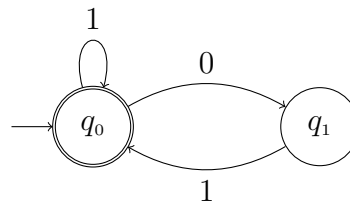
Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que tienen un número par de símbolos (cadenas de longitud par ≥ 0). AFD M tal que $L(M) = L$:



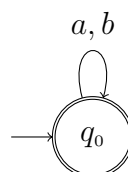
Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que contienen un número par de a 's. AFD M tal que $L(M) = L$:



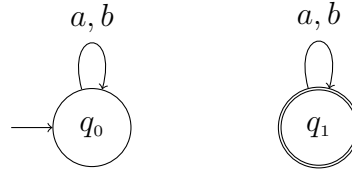
Ejemplo Sean $\Sigma = \{0, 1\}$ y $L = (1 \cup 01)^*$. Construimos un AFD de tal forma que el estado inicial q_0 sea el único estado de aceptación y en él confluyan las trayectorias 1 y 01. Omitiendo el estado limbo obtenemos el siguiente autómata:



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

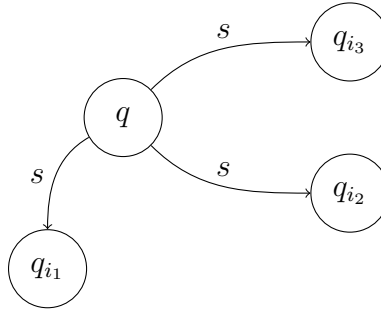
$$\begin{aligned} \Delta : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, s) &\longmapsto \Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\} \end{aligned}$$

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:



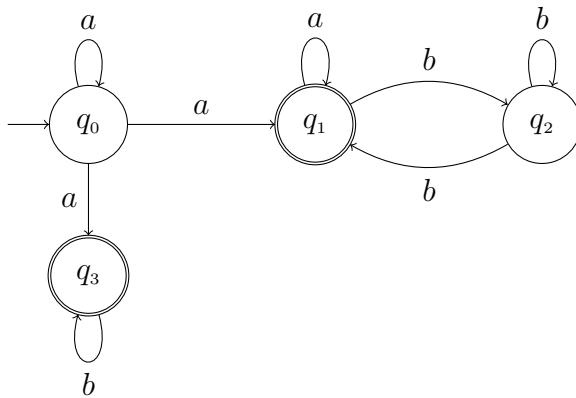
Un AFN M puede procesar una cadena de entrada $u \in \Sigma^*$ de varias maneras. Sobre el grafo del autómata, esto significa que pueden existir varias trayectorias, desde el estado q_0 , etiquetadas con los símbolos de u .

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.

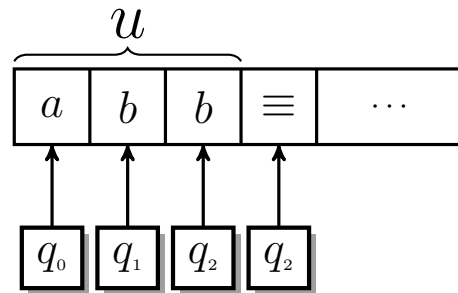
Ejemplo Sea M el siguiente AFN:



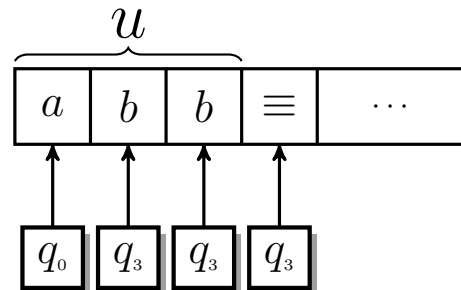
Δ	a	b
q_0	$\{q_0, q_1, q_3\}$	\emptyset
q_1	$\{q_1\}$	$\{q_2\}$
q_2	\emptyset	$\{q_1, q_2\}$
q_3	\emptyset	$\{q_3\}$

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, $u \in L(M)$.

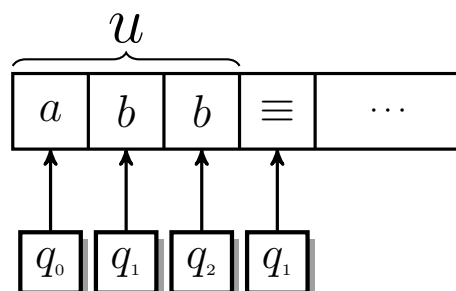
Procesamiento de rechazo:



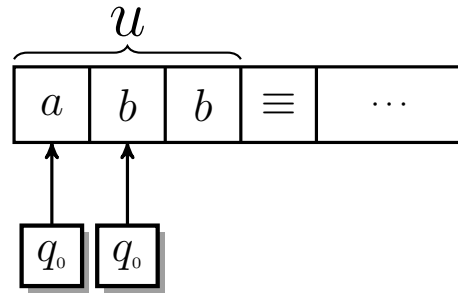
Procesamiento de aceptación:



Otro procesamiento de aceptación:

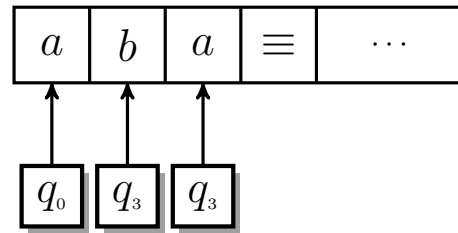


Procesamiento abortado:



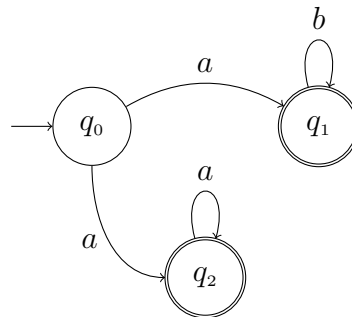
El grafo de M nos permite ver que la cadena $aabbaa$ es aceptada mientras que $aabaa$ es rechazada. Todas las cadenas que comienzan con b son rechazadas. También es rechazada la cadena aba ; uno de sus procesamientos posibles es el siguiente:

Procesamiento abortado:



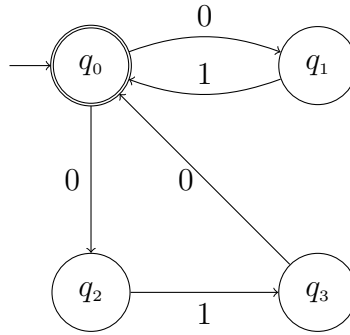
A pesar de que el procesamiento anterior termina en el estado de aceptación q_3 , la entrada no es aceptada porque no se consume completamente.

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

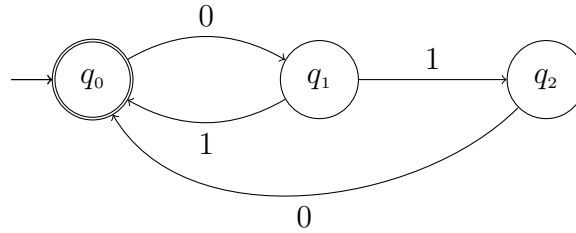


Un AFD que acepte a L requiere como mínimo cuatro estados más un estado limbo (cinco estados en total).

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.

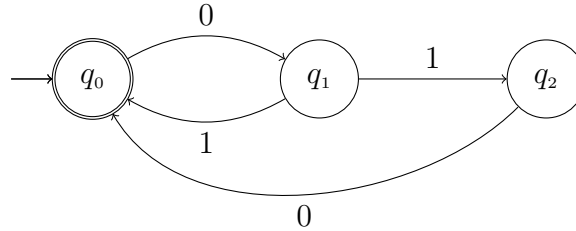
2.7.1 Teorema. Dado un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ se puede construir un AFD M' equivalente a M , es decir, tal que $L(M) = L(M')$.

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 $\Sigma = \{0, 1\}$.

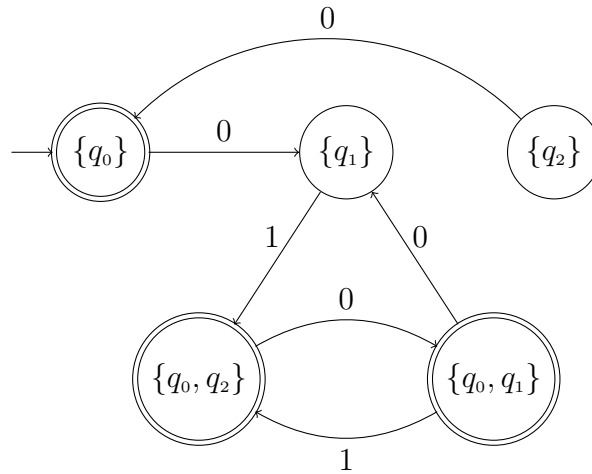


La tabla original de la función de transición Δ de M contiene la combinación $\{q_0, q_2\}$ que adquiere el estatus de nuevo estado en M' . Al extender la tabla de Δ aparece también el nuevo estado $\{q_0, q_1\}$:

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
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$

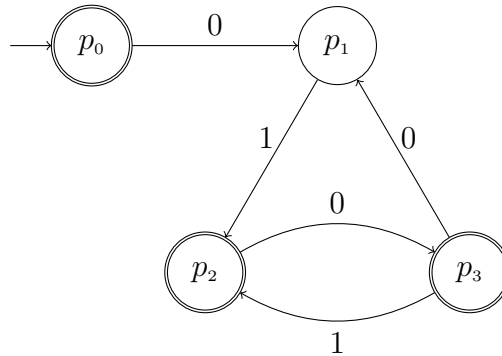
La tabla original de la función Δ y su extensión.

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:

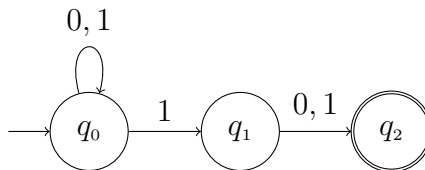


Los estados de aceptación son aquéllos en los que aparezca q_0 ya que q_0 es el único estado de aceptación del autómata original.

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:



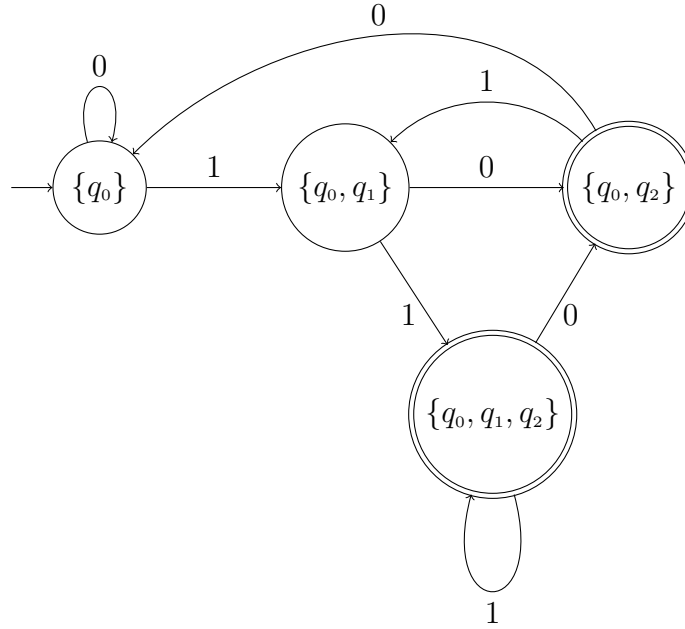
Ejemplo Sean $\Sigma = \{0, 1\}$ y L_2 el lenguaje de todas las cadenas de longitud ≥ 2 en las que el segundo símbolo, de derecha a izquierda es un 1. Una expresión regular para este lenguaje es $(0 \cup 1)^* 1 (0 \cup 1)$ y es fácil diseñar un AFN M que acepte a L_2 :



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 $u \in \Sigma^*$, $\hat{\delta}(q, u)$ es el estado en el que el autómata termina el procesamiento de u a partir del estado q . En particular, $\hat{\delta}(q_0, u)$ 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:

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

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

2.7.3 Definición. Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. La función de transición Δ , $\Delta : Q \times \Sigma \longrightarrow \wp(Q)$ 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 $\widehat{\Delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$, 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

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

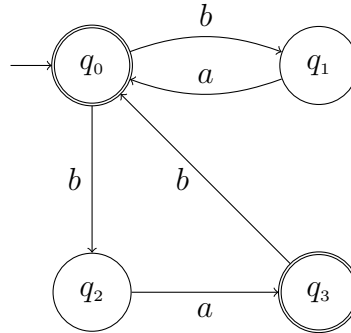
Finalmente podemos demostrar que $L(M') = L(M)$:

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

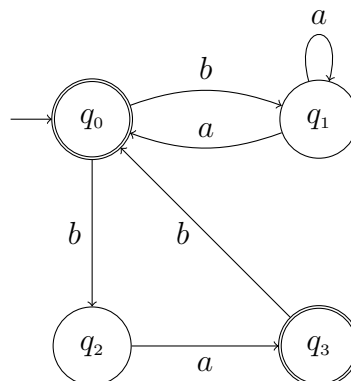
Ejercicios de la sección 2.7

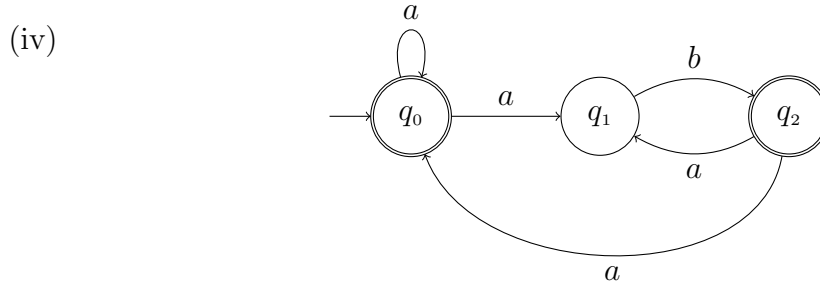
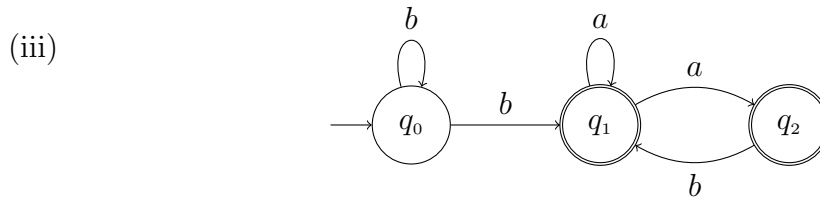
- ① Utilizando el procedimiento de conversión presentado en esta sección, encontrar AFD equivalentes a los siguientes AFN:

(i)



(ii)





- 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

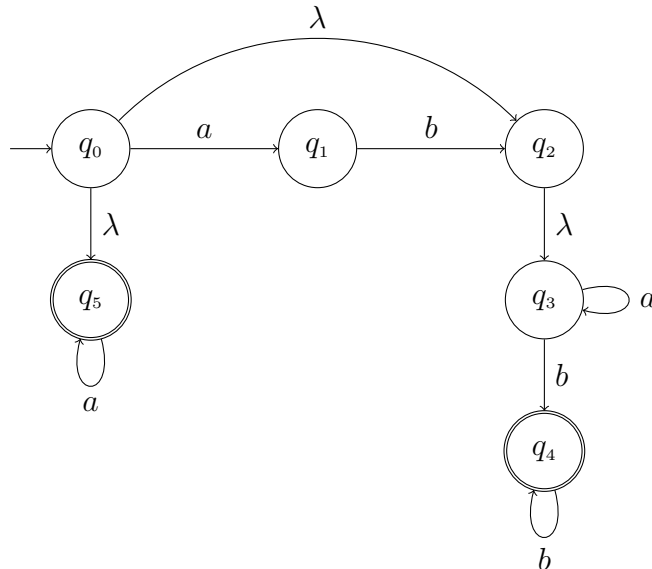
$$\Delta(q, \lambda) = \{q_{i_1}, \dots, q_{i_k}\}$$

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.

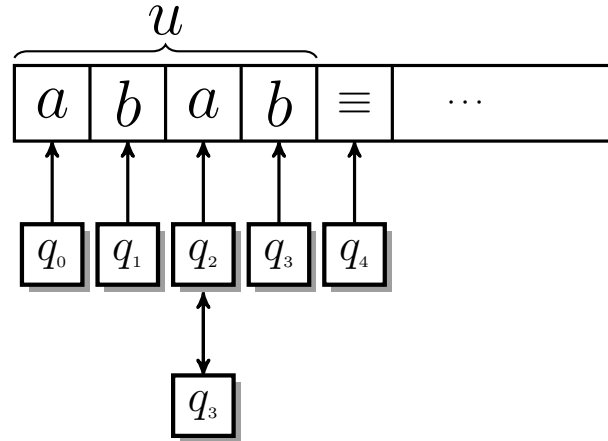
Como sucede en el caso AFN, una cadena de entrada $u \in \Sigma^*$ es aceptada si existe por lo menos un procesamiento completo de u , desde q_0 , que termina en un estado de aceptación. En el grafo del autómata, las transiciones λ dan lugar a arcos con etiquetas λ . Una cadena de entrada u es aceptada 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 procesamiento para una misma cadena de entrada, así como procesamiento abortados.

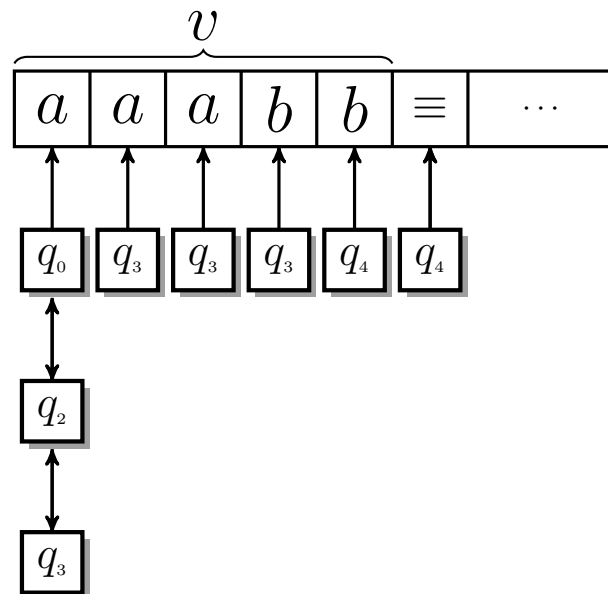
Ejemplo Consideremos el siguiente AFN- λ , M :



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 $\lambda\lambda abbb$.

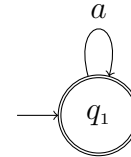
Los AFN- λ permiten aún más libertad en el diseño de autómatas, especialmente cuando hay numerosas uniones y concatenaciones.

Ejemplo Diseñar AFN- λ que acepten los siguientes lenguajes:

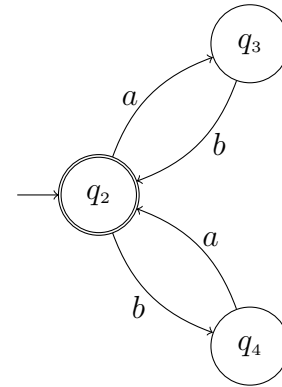
- (1) $a^* \cup (ab \cup ba)^* \cup b^+$.
- (2) $a^*(ab \cup ba)^*b^+$.

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.

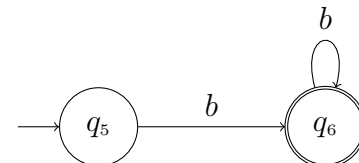
Autómata que acepta a^* :



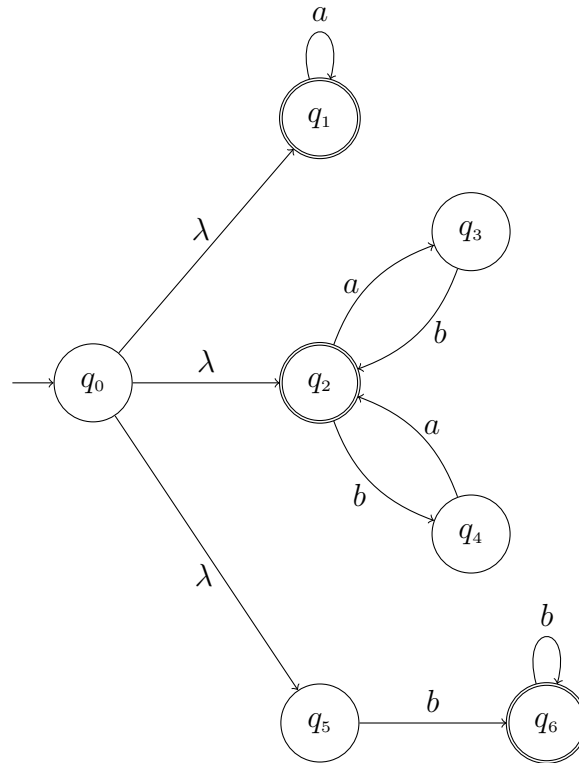
Autómata que acepta $(ab \cup ba)^*$:



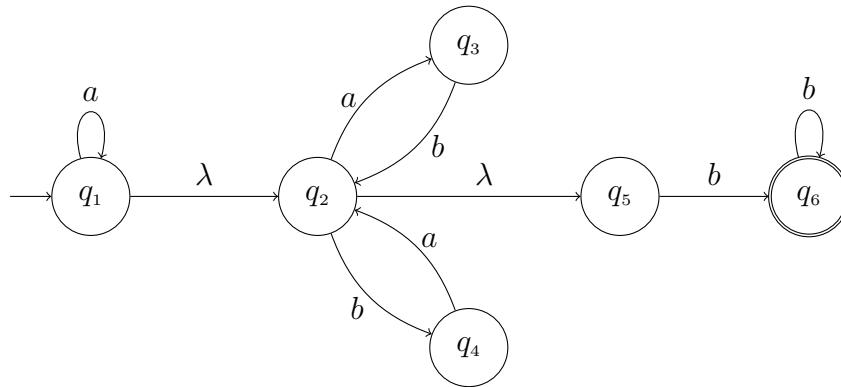
Autómata que acepta b^+ :



- (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.

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:

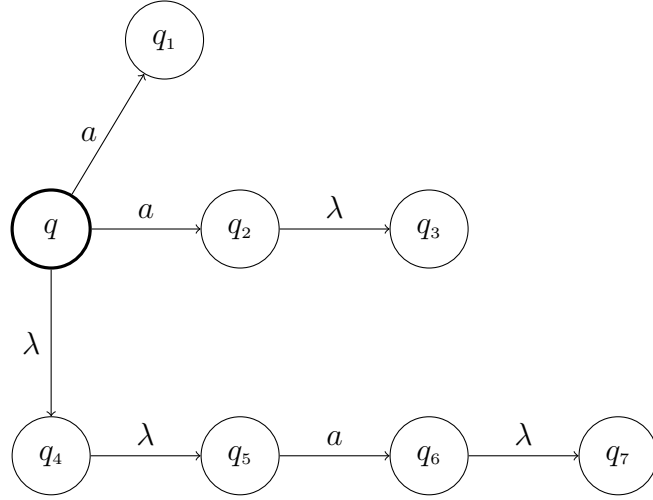
$$\begin{aligned} \Delta' : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, a) &\longmapsto \Delta'(q, a) := \lambda[\Delta(\lambda[q], a)]. \end{aligned}$$

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

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 :



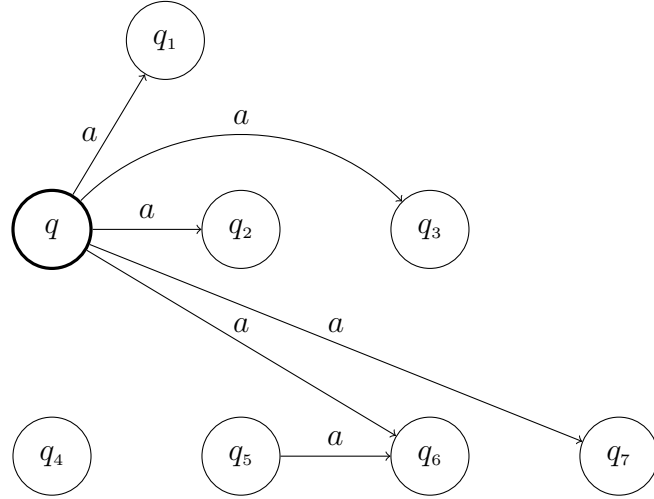
Por simple inspección observamos que, una vez procesada una a , el autómata puede pasar desde el estado q a uno de los siguientes estados: q_1, q_2, q_3, q_6, q_7 . Para obtener esta lista de estados se tienen en cuenta todas las transiciones λ que preceden o prosiguen el procesamiento del símbolo a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

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

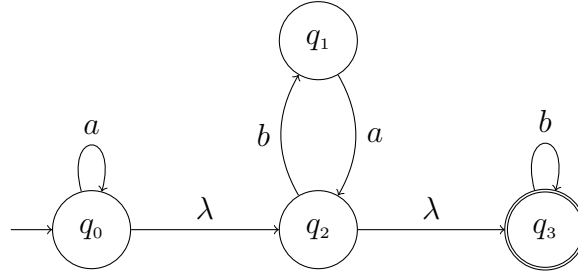
La porción correspondiente del grafo de M' se exhibe en la siguiente gráfica. De esta forma M' simula, sin transiciones λ , todas las transiciones λ de M añadiendo nuevas transiciones con etiqueta a .

Porción de M' :



Ejemplo

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



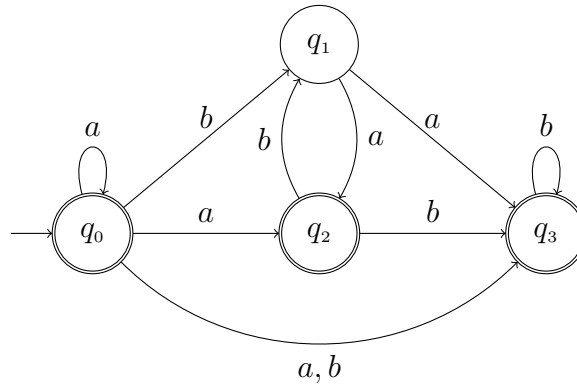
Las λ -clausuras de los estados vienen dadas por:

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

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:



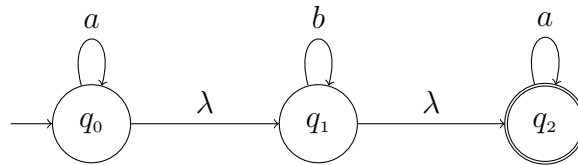
Puesto que q_3 , que es el único estado de aceptación del autómata original M , pertenece a $\lambda[q_0]$, a $\lambda[q_2]$ y a $\lambda[q_3]$, los tres estados q_0 , q_2 y q_3 son estados de aceptación en el autómata M' .

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.

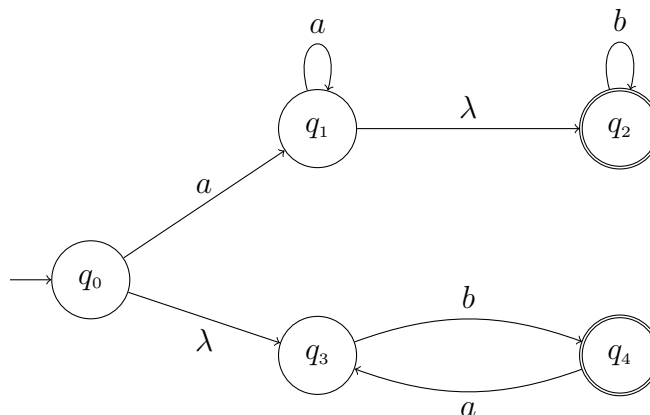
Ejercicios de la sección 2.9

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 $a \in \Sigma$.

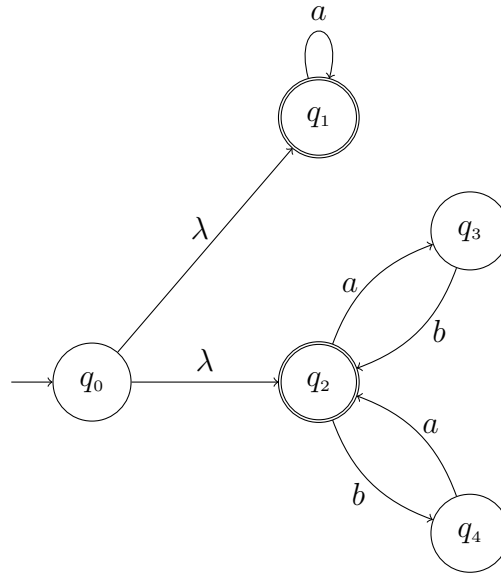
①



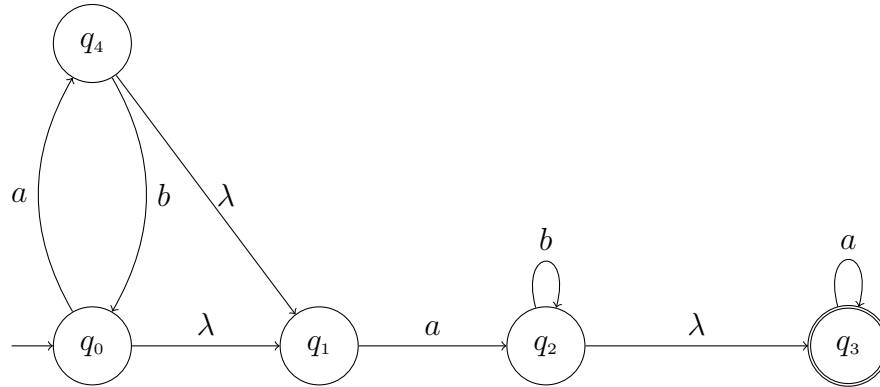
②



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 $L(\overline{M}) = \emptyset$.

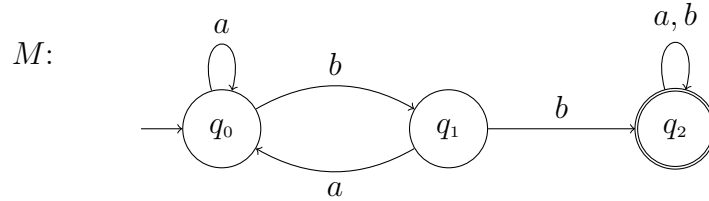
Cuando un lenguaje L está definido por medio de una condición negativa puede ser más fácil diseñar primero un AFD que acepte su complemento \overline{L} .

Ejemplo

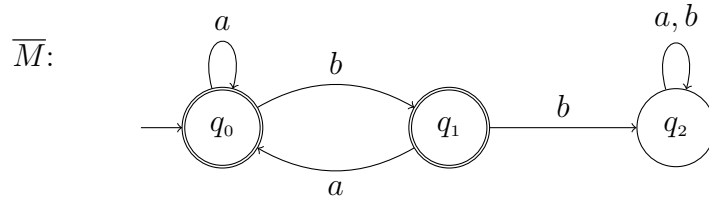
Sea $\Sigma = \{a, b\}$. Encontrar un AFD que acepte el lenguaje L de todas las cadenas que no tienen dos *b*es consecutivas (es decir, no contienen la subcadena

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 :



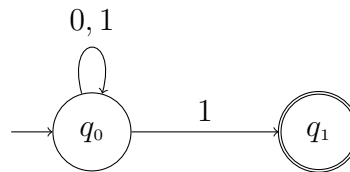
Para aceptar a L formamos el complemento de M , intercambiando aceptación con no-aceptación:



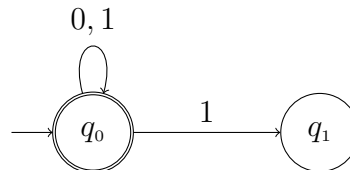
En \overline{M} , q_2 es estado limbo y $L(\overline{M}) = L$.

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

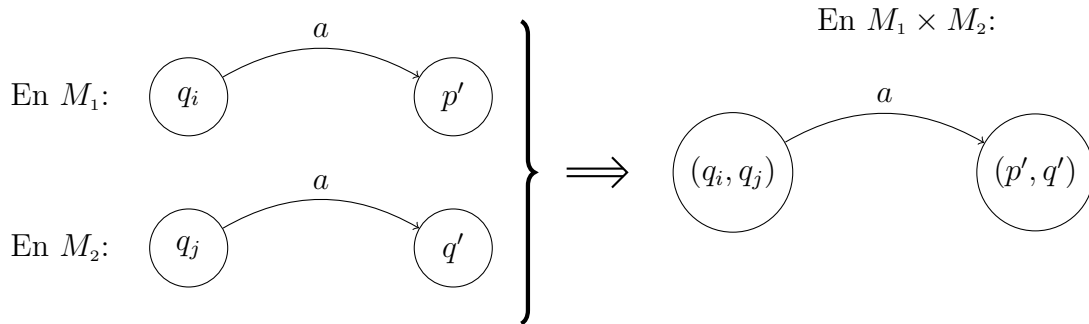
$$\begin{aligned} \delta : (Q_1 \times Q_2) \times \Sigma &\longrightarrow Q_1 \times Q_2 \\ \delta((q_i, q_j), a) &= (\delta_1(q_i, a), \delta_2(q_j, a)). \end{aligned}$$

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 $M_1 \times M_2$ de la siguiente manera:



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 $F = \{(q_i, q_j) : q_i \in F_1 \text{ o } q_j \in F_2\}$ 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 $F = \{(q_i, q_j) : q_i \in F_1 \text{ ó } q_j \notin F_2\}$ 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

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

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

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

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

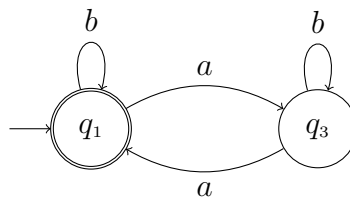
$$\begin{aligned}
 w \in L(M) &\iff (\hat{\delta}_1(q_1, u), \hat{\delta}_2(q_2, u)) \in F \\
 &\iff \hat{\delta}_1(q_1, u) \in F_1 \wedge \hat{\delta}_2(q_2, u) \notin F_2 \\
 &\iff u \in L(M_1) \wedge u \notin L(M_2) \\
 &\iff u \in L(M_1) - L(M_2) = L_1 - L_2.
 \end{aligned}$$

Esto demuestra (iii). □

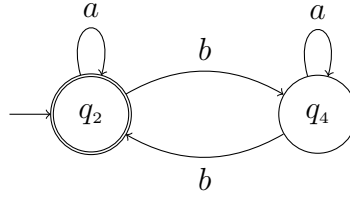
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen un número par de a s y un número par de b s.

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):



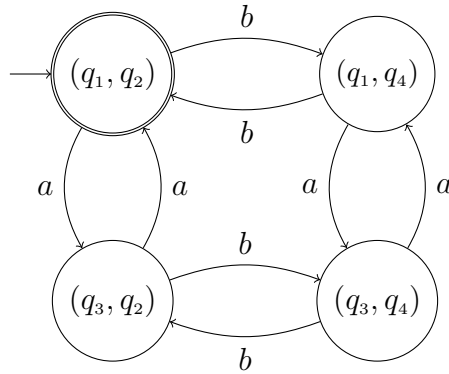
AFD M_2 que acepta L_2 (cadenas con un número par de *bes*):



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 $M_1 \times M_2$.

$$\begin{aligned}
 \delta((q_1, q_2), a) &= (\delta_1(q_1, a), \delta_2(q_2, a)) = (q_3, q_2), \\
 \delta((q_1, q_2), b) &= (\delta_1(q_1, b), \delta_2(q_2, b)) = (q_1, q_4), \\
 \delta((q_1, q_4), a) &= (\delta_1(q_1, a), \delta_2(q_4, a)) = (q_3, q_4), \\
 \delta((q_1, q_4), b) &= (\delta_1(q_1, b), \delta_2(q_4, b)) = (q_1, q_2), \\
 \delta((q_3, q_2), a) &= (\delta_1(q_3, a), \delta_2(q_2, a)) = (q_1, q_2), \\
 \delta((q_3, q_2), b) &= (\delta_1(q_3, b), \delta_2(q_2, b)) = (q_3, q_4), \\
 \delta((q_3, q_4), a) &= (\delta_1(q_3, a), \delta_2(q_4, a)) = (q_1, q_4), \\
 \delta((q_3, q_4), b) &= (\delta_1(q_3, b), \delta_2(q_4, b)) = (q_3, q_2).
 \end{aligned}$$

El grafo del autómata así obtenido es:



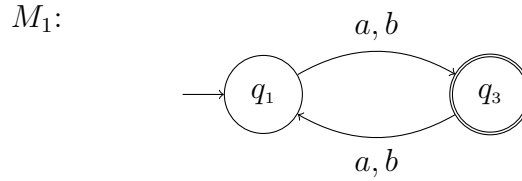
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos *bes* consecutivas, es decir, no contienen la subcadena *bb*.

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

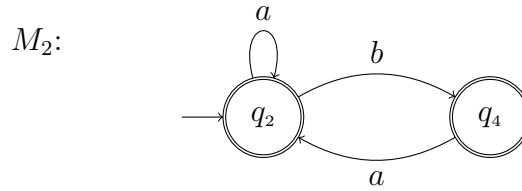
L_1 = lenguaje de todas las cadenas que tienen longitud impar.

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 :

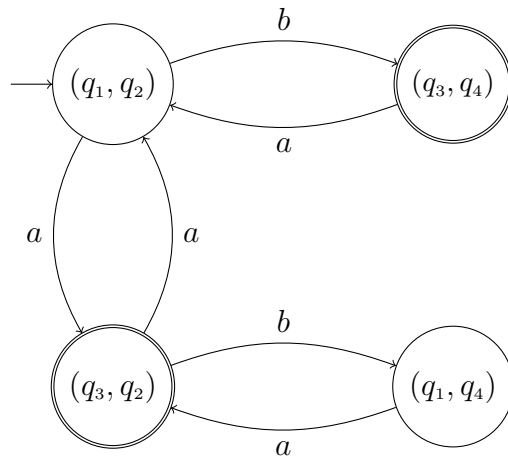


Y un AFD M_2 que acepta L_2 :



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.

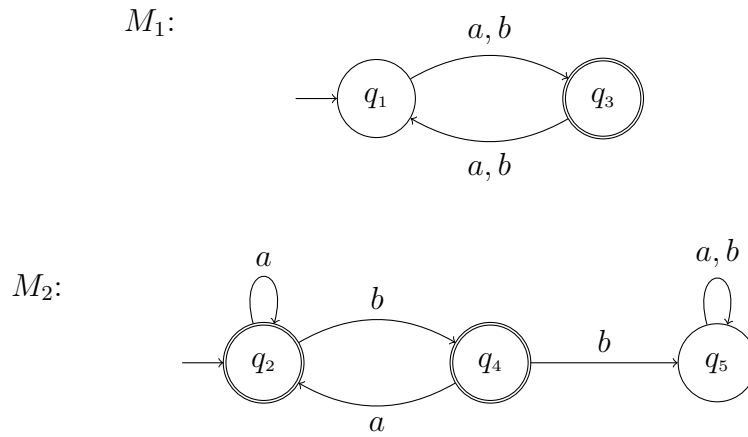
Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene 4 estados: (q_1, q_2) , (q_1, q_4) , (q_3, q_2) y (q_3, q_4) . Los estados de aceptación son (q_3, q_2) y (q_3, q_4) ya que q_3 es de aceptación en M_1 mientras que q_2 y q_4 son de aceptación en M_2 . Utilizando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



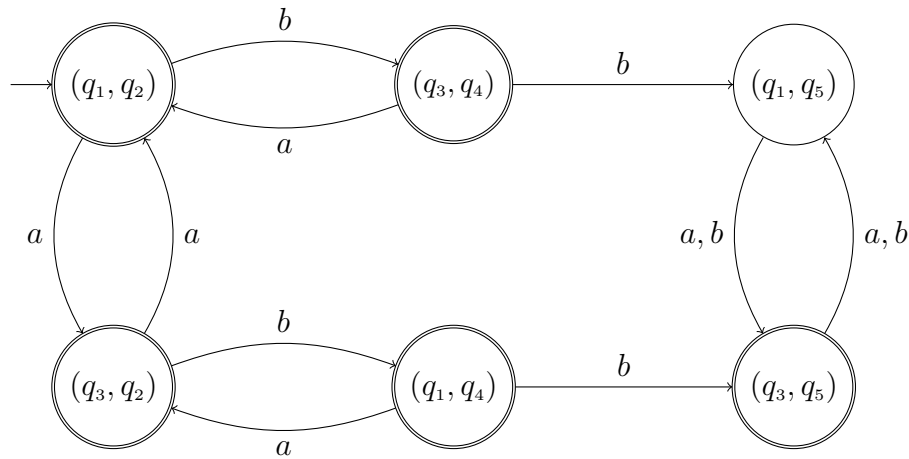
Este problema también se puede resolver expresando el lenguaje L como diferencia de dos lenguajes (véase el Ejercicio ① al final de la presente sección).

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 $M = M_1 \times M_2$ 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

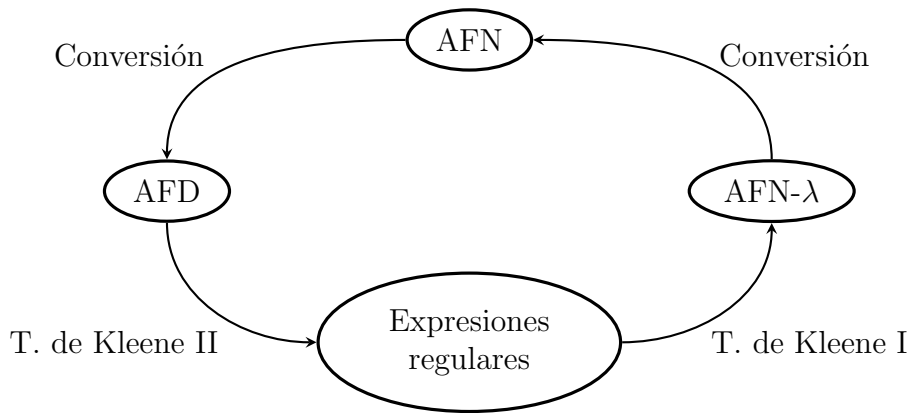
- ① Utilizar el Teorema 2.11.1 (iii) para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos b s consecutivas, expresando L como diferencia de dos lenguajes.
- ② Utilizar el Teorema 2.11.1 para construir 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 s.
 - (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 s.
- 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.

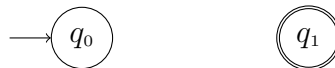


2.12.1. Teorema de Kleene. Sea Σ un alfabeto dado. Un lenguaje es regular (sobre Σ) si y sólo si es aceptado por un autómata finito (AFD o AFN o AFN- λ) con alfabeto de entrada Σ .

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



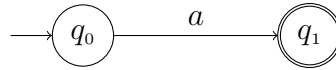
acepta el lenguaje \emptyset , es decir, el lenguaje representado por la expresión regular $R = \emptyset$.

El autómata



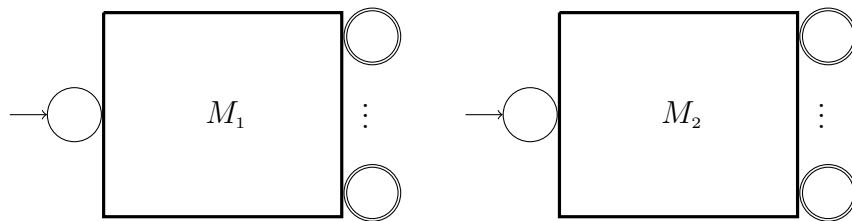
acepta el lenguaje $\{\lambda\}$, es decir, el lenguaje representado por la expresión regular $R = \lambda$.

El autómata



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

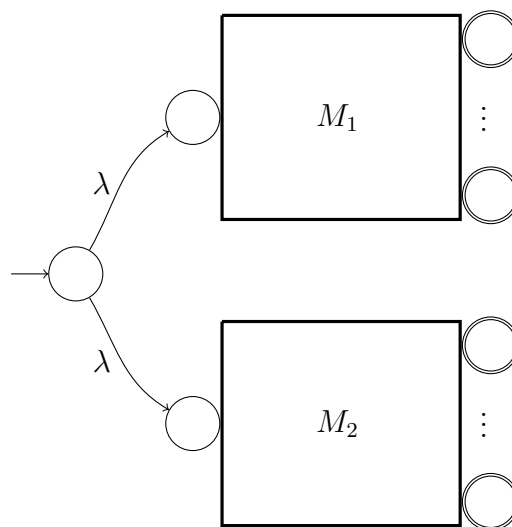
Razonando recursivamente, supóngase que para las expresiones regulares R_1 y R_2 se dispone de AFN- λ M_1 y M_2 tales que $L(M_1) = R_1$ y $L(M_2) = R_2$. Esquemáticamente vamos a presentar los autómatas M_1 y M_2 en la siguiente forma:



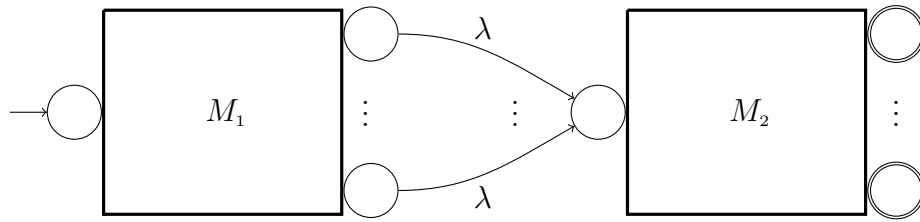
Los estados finales o de aceptación se dibujan a la derecha, pero cabe advertir que el estado inicial puede ser también un estado de aceptación. Podemos ahora obtener AFN- λ que acepten los lenguajes $R_1 \cup R_2$ y $R_1 R_2$.

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 $R_1 \cup R_2$:

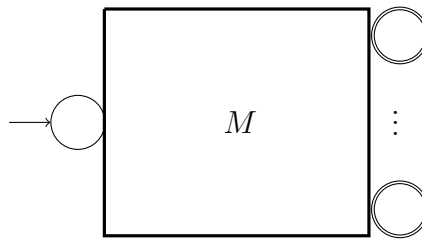


Autómata que acepta $R_1 R_2$:

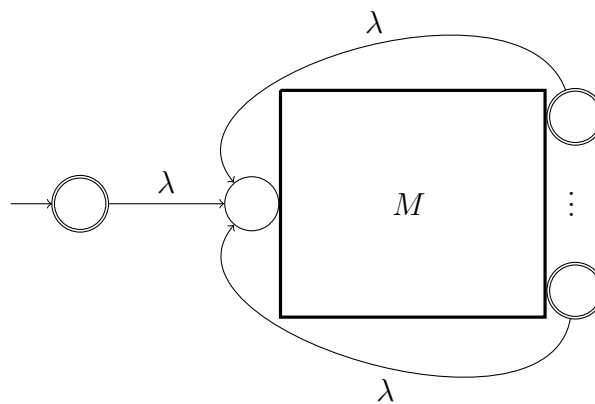


Este tipo de conexión entre dos autómatas M_1 y M_2 se denomina *conexión en serie*. Los estados de aceptación del nuevo autómata son únicamente los estados de aceptación de M_2 .

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

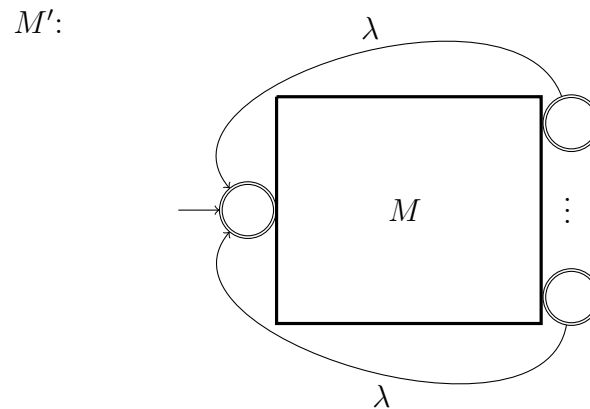


El siguiente autómata acepta R^* :

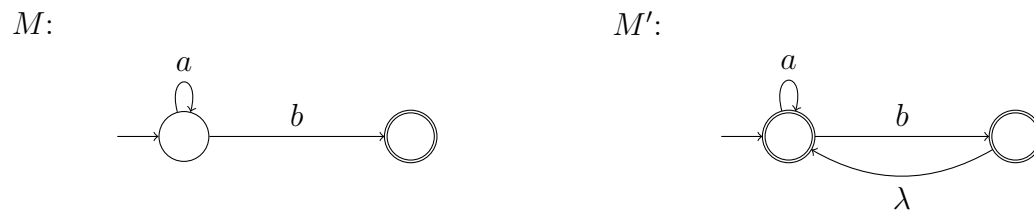


Esto concluye la demostración de la parte I del Teorema de Kleene. □

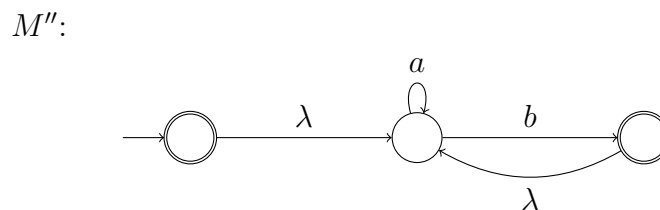
Para aceptar R^* a partir de un autómata M tal que $L(M) = R$, sería incorrecto (en general) utilizar el siguiente autómata M' :



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.



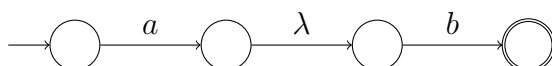
No se cumple que $L(M') = R^* = (a^*b)^*$ ya que M' acepta también a, a^2, a^3, \dots (potencias de a) que no pertenecen a R^* . Para aceptar R^* utilizando la construcción mencionada en la demostración del Teorema 2.12.1 se obtendría el siguiente autómata M'' :



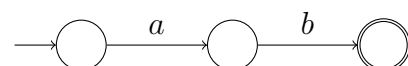
Simplificaciones en el procedimiento. El procedimiento constructivo del Teorema 2.12.1 admite varias simplificaciones, útiles en la práctica.

Para aceptar ab :

Según el procedimiento:

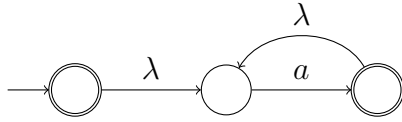


Simplificación:

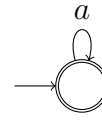


Para aceptar a^* :

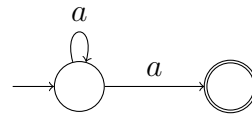
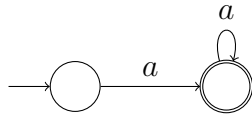
Según el procedimiento:



Simplificación:



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

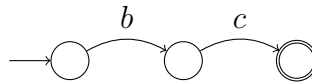


Además, las conexiones en paralelo y en serie de dos autómatas se pueden generalizar fácilmente para aceptar uniones $R_1 \cup R_2 \cup R_3 \cup \dots \cup R_k$, o concatenaciones $R_1 R_2 R_3 \dots R_k$, de k lenguajes ($k \geq 3$).

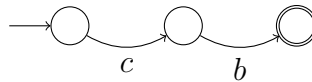
Ejemplo Utilizar el procedimiento del Teorema 2.12.1 para construir un AFN- λ que acepte el lenguaje $(bc \cup cb)^* a^* b \cup (b^* ca)^* c^+$ sobre el alfabeto $\Sigma = \{a, b, c\}$.

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.

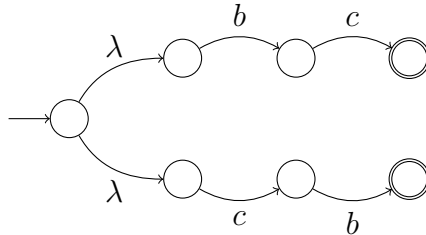
Autómata que acepta bc :



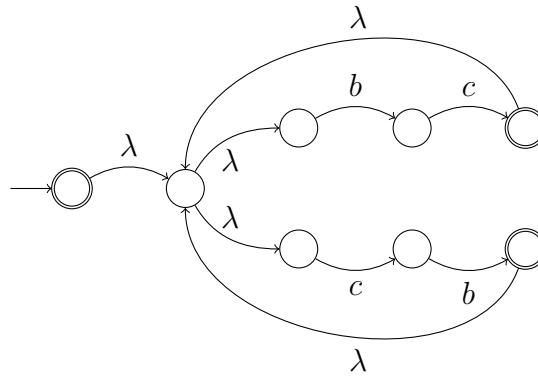
Autómata que acepta cb :



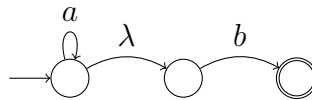
Autómata que acepta $bc \cup cb$:



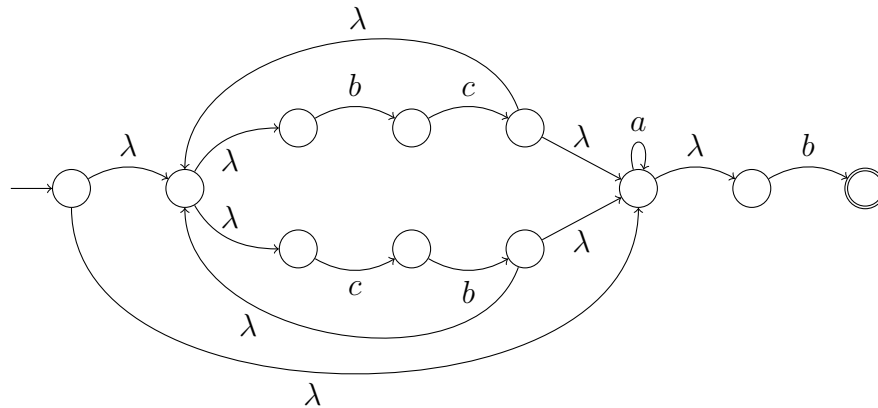
Autómata que acepta $(bc \cup cb)^*$:



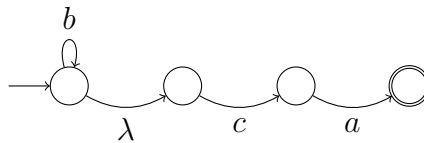
Autómata que acepta a^*b :



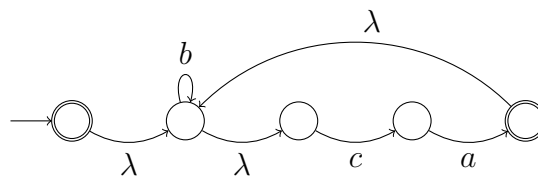
Para aceptar $R_1 = (bc \cup cb)^*a^*b$ conectamos en serie los dos últimos autómatas:



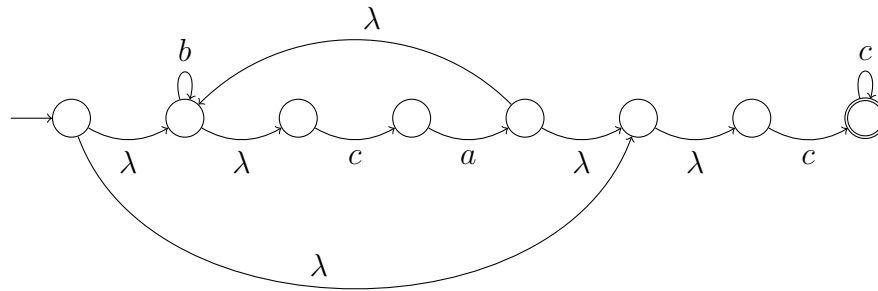
A continuación construimos un autómata que acepte $R_2 = (b^*ca)^*c^+$. Autómata que acepta b^*ca :



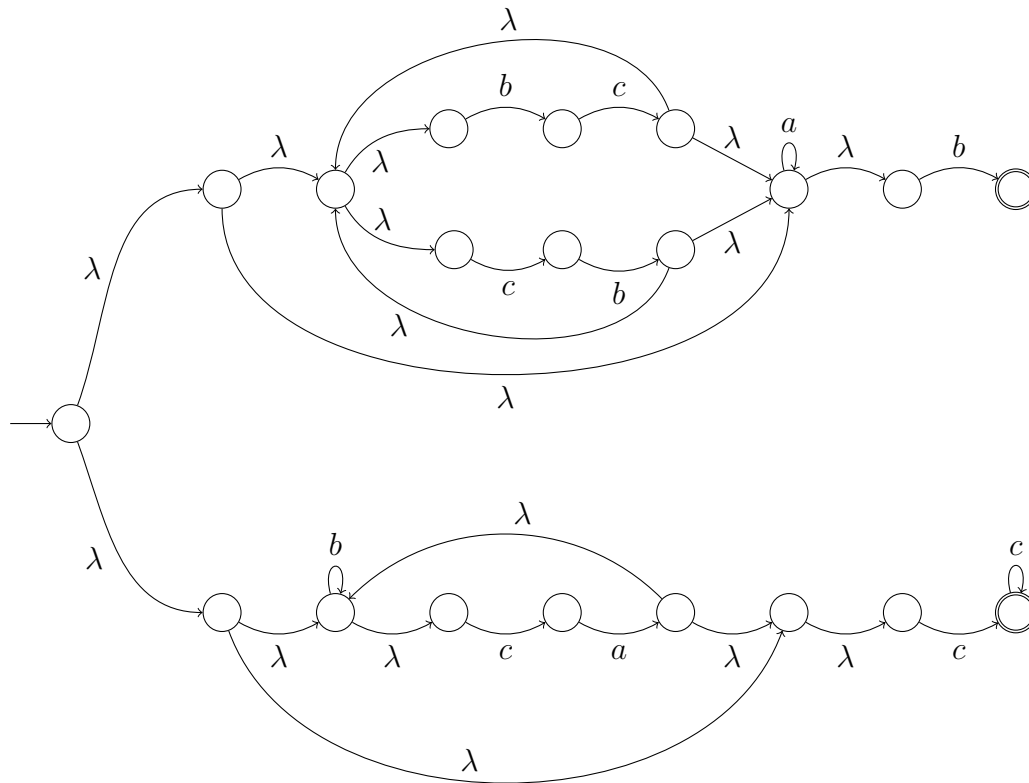
Autómata que acepta $(b^*ca)^*$:



Autómata que acepta $R_2 = (b^*ca)^*c^+$:



Finalmente, conectamos en paralelo los autómatas que aceptan R_1 y R_2 y obtenemos un autómata que acepta $R = R_1 \cup R_2 = (bc \cup cb)^*a^*b \cup (b^*ca)^*c^+$:



Ejercicios de la sección 2.12

Utilizar el procedimiento del Teorema 2.12.1 para construir AFN- λ que acepten los siguientes lenguajes sobre el alfabeto $\{a, b, c\}$.

- ① $(a^*c)^* \cup (b^*a)^*(ca^*)^*b^+$.

- 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 $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^* \cup (b \cup \lambda)(b^*ac)^*$.

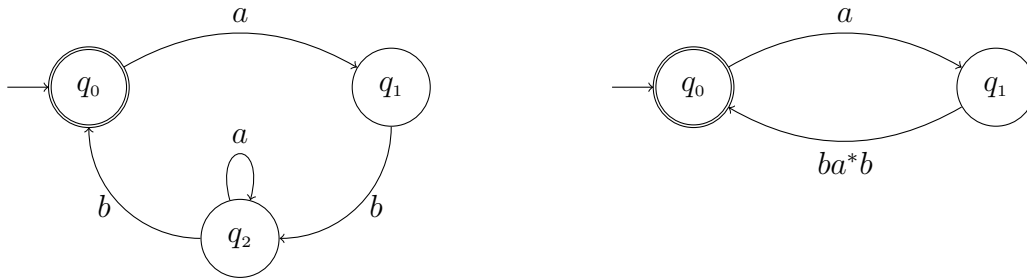
2.13. Teorema de Kleene, parte II

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:

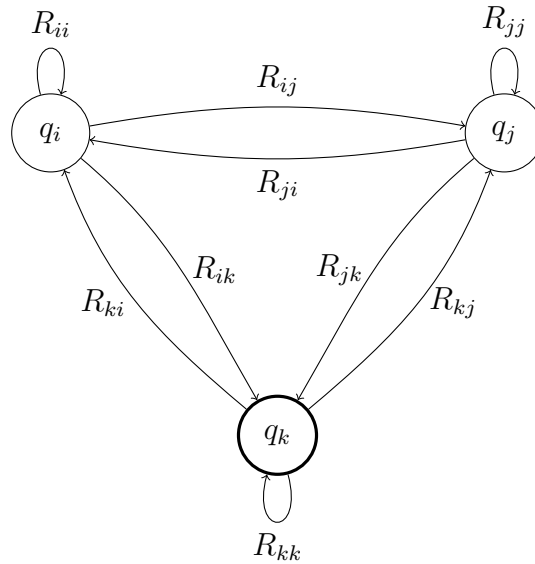
- (1) Convertir el grafo de M en un GEG G reemplazando múltiples arcos etiquetados con símbolos a_1, a_2, \dots, a_k entre dos estados q_i y q_j por un único arco etiquetado $a_1 \cup a_2 \cup \dots \cup a_k$:



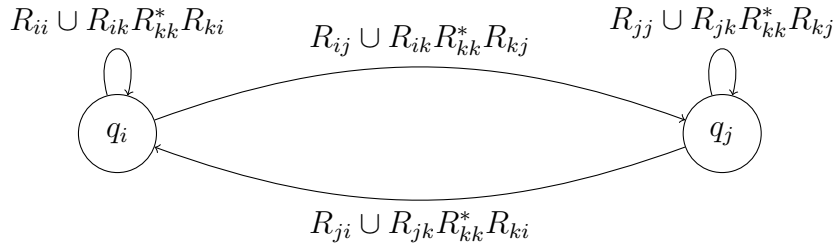
De esta forma, entre dos estados cualesquiera hay, a lo sumo, un arco etiquetado con una expresión regular.

- (2) Modificar el GEG G de tal manera que haya un único estado de aceptación. Esto se puede conseguir añadiendo un nuevo estado q_f , que será el único estado de aceptación, y trazando transiciones λ entre todos y cada uno de los estados en F (los estados de aceptación originales) y q_f . Cuando el autómata original posee un único estado de aceptación, simplemente se mantiene como tal hasta el final.
- (3) Este paso es un proceso 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:

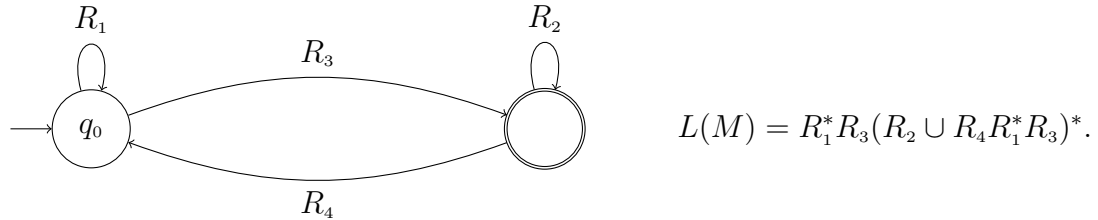
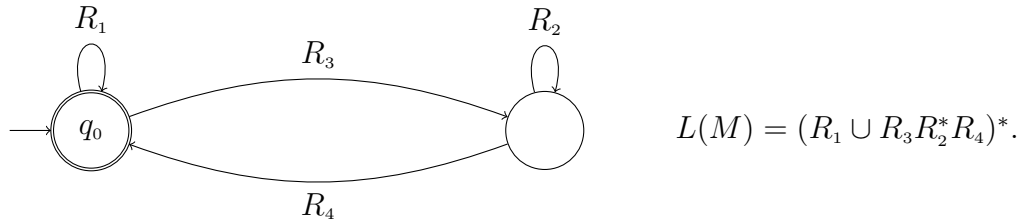


q_k sirve de “puente” entre q_i y q_j , y entre q_j y q_i . Además, a través de q_k hay una trayectoria que conecta q_i consigo mismo, y también una trayectoria que conecta q_j consigo mismo. Teniendo en cuenta tales trayectorias, se procede a eliminar el estado q_k reemplazando las etiquetas entre q_i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

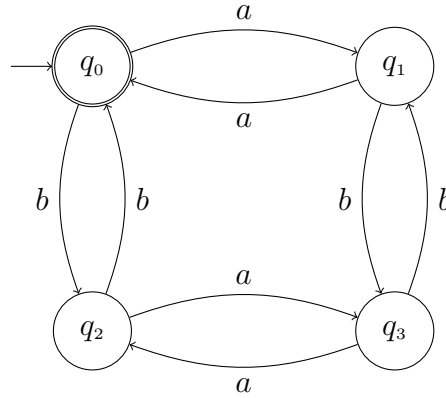


Observaciones:

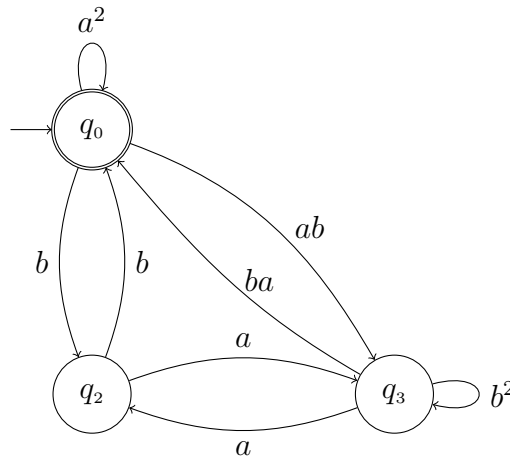
1. El procedimiento anterior es bastante flexible; por ejemplo, el paso (2) (estado de aceptación único) se puede realizar después de haber eliminado uno o más estados (siguiendo las instrucciones del paso (3)).
2. Es importante recalcar que, siempre que se ejecute la subrutina (3), el estado inicial y los estados de aceptación no se pueden eliminar.

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.

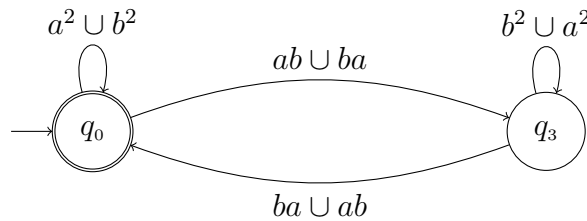
Solución. Conocemos un AFD M que acepta este lenguaje:



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:



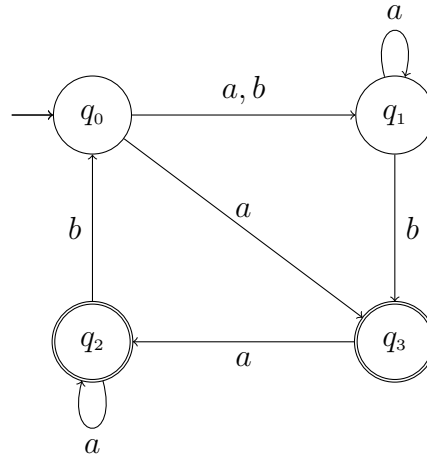
El lenguaje aceptado es entonces

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

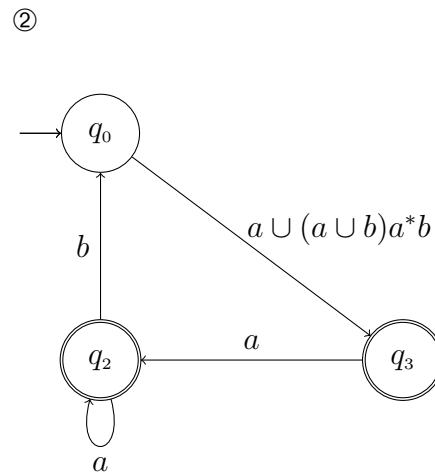
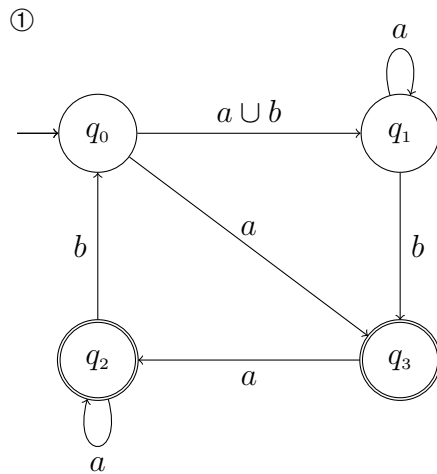
Si en el penúltimo GEG se elimina q_3 en vez de q_2 , se llega finalmente a una expresión regular diferente (y más compleja) que representa el mismo lenguaje.

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

M :

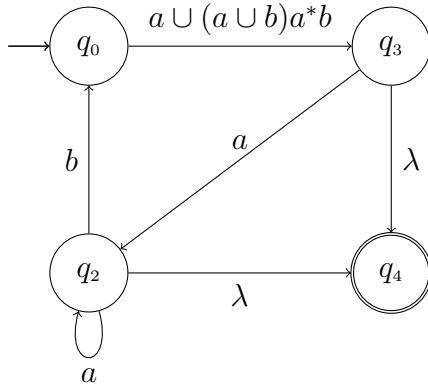


Solución. Primero convertimos el grafo de M en un GEG (grafo ①) y luego eliminamos el estado q_1 (grafo ②).

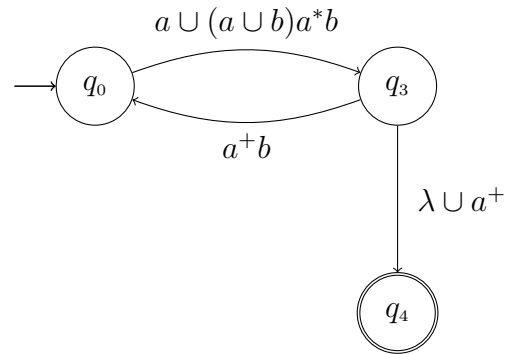


A continuación añadimos el nuevo estado q_4 (que será el único estado de aceptación) y transiciones λ desde q_2 y q_3 hasta q_4 (grafo ③). Luego eliminamos el estado q_2 (grafo ④):

③

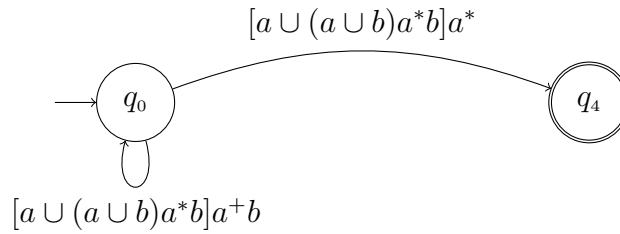


④



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:

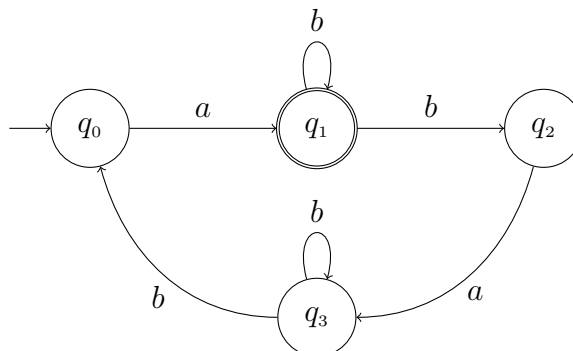
$$L(M) = \left([a \cup (a \cup b)a^*b]a^+b \right)^* [a \cup (a \cup b)a^*b]a^*.$$

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 $L(M)$.

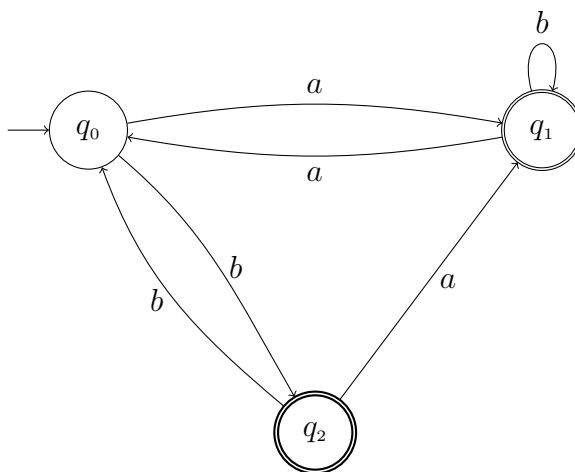
Ejercicios de la sección 2.13

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

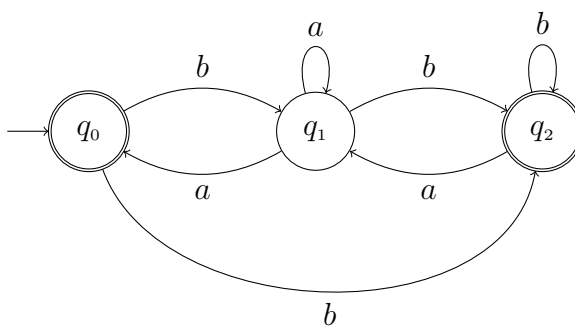
(i)



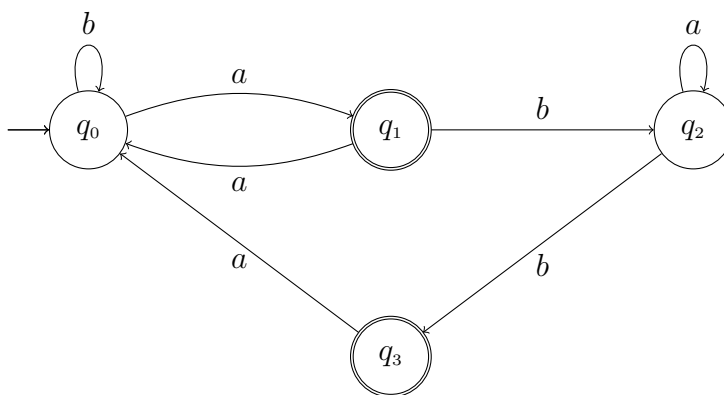
(ii)



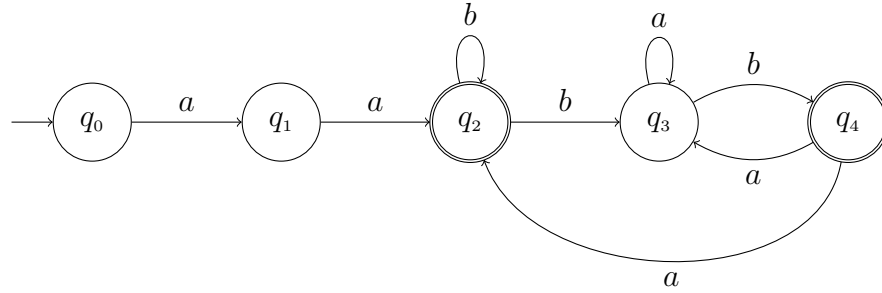
(iii)



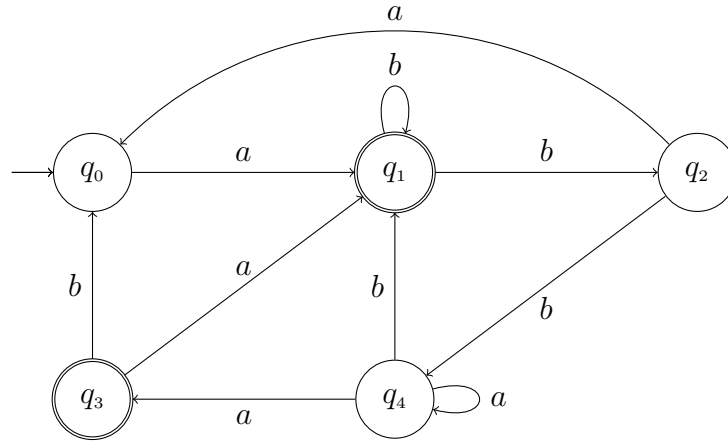
(iv)



(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) $\overline{L} = \Sigma^* - L$. |
| (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 \overline{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) $\overline{L} = \Sigma^* - L$. |
| (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. Minimización de autómatas, parte I

En la presente sección presentaremos un procedimiento general para encontrar un autómata con el menor número de estados posible, equivalente a un AFD dado. Se trata de un procedimiento algorítmico en el que 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.

2.15.1 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:

$$\begin{aligned} Q' &= \{[p] : p \in Q\}, \\ q'_0 &= [q_0], \\ F' &= \{[p] : p \in F\}, \\ \delta'([p], a) &= [\delta(p, a)], \text{ para todo } a \in \Sigma. \end{aligned}$$

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.

2.15.2 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) $\widehat{\delta'}([p], w) = [\widehat{\delta}(p, u)]$ 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

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

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 2.15.2 se puede deducir que M y M' aceptan el mismo lenguaje, tal como se demuestra en el siguiente teorema.

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

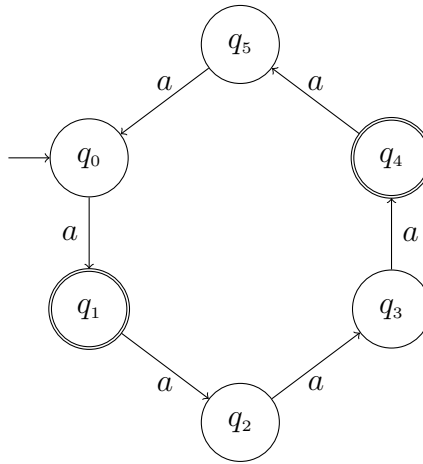
Demostración.

$$\begin{aligned} u \in L(M') &\iff \widehat{\delta'}([q_0], u) \in F' \\ &\iff [\widehat{\delta}(q_0, u)] \in F' \quad (\text{por la Proposición 2.15.2 (iv)}) \\ &\iff \widehat{\delta}(q_0, u) \in F \quad (\text{por la Proposición 2.15.2 (iii)}) \\ &\iff u \in L(M). \quad \square \end{aligned}$$

Dado un AFD M , el autómata cociente M' resulta ser un autómata con el mínimo número de estados posible para aceptar $L(M)$. Esto se demuestra detalladamente en la sección 2.16

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*. Un estado q es accesible si existe una cadena de entrada u 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 deben eliminar previamente. 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”.

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

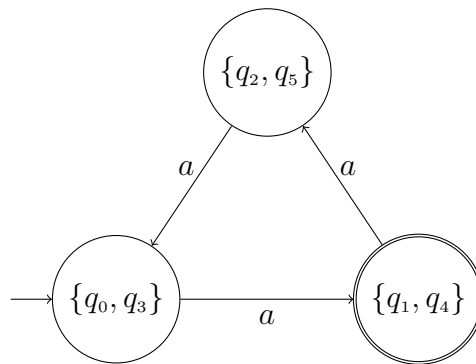
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos el símbolo a . La información necesaria aparece en la siguiente tabla; la columna izquierda corresponde a las casillas no marcadas $\{p, q\}$ y la derecha a las casillas $\{\delta(p, a), \delta(q, a)\}$ obtenidas al procesar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

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

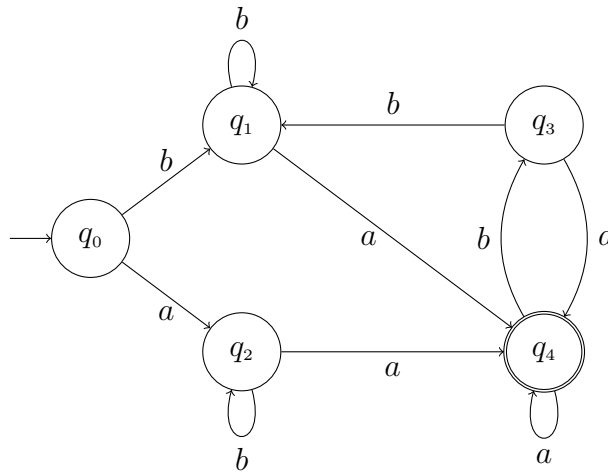
Las marcas \times en la columna derecha representan casillas previamente marcadas; según el algoritmo, las correspondientes casillas en la columna izquierda se marcan con \times . Entonces, al terminar la segunda iteración la tabla triangular adquiere el siguiente aspecto:

q_0					
×	q_1				
×	×	q_2			
	×	×	q_3		
×		×	×	q_4	
×	×		×	×	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	
×	×	×	×	q_4

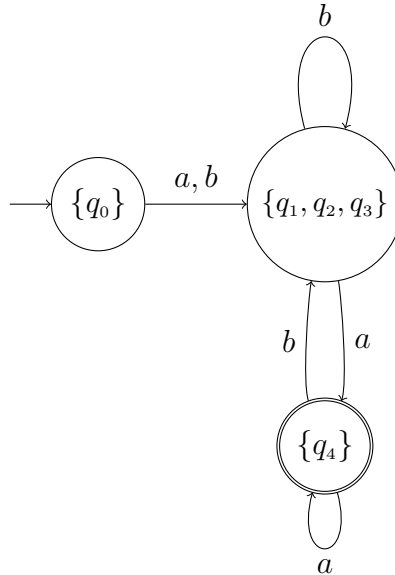
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos con las entradas a y b . Tal información aparece en la siguiente tabla, la cual se utiliza a partir de la segunda iteración.

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

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:

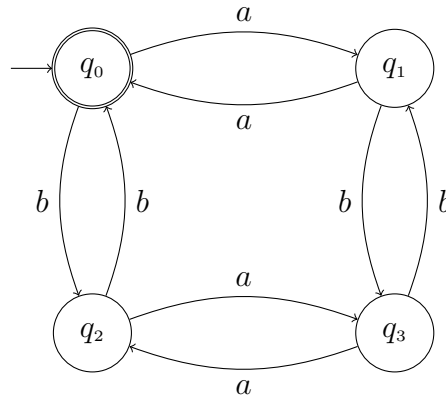
	q_0				
\times		q_1			
\times			q_2		
\times				q_3	
\times	\times	\times	\times	\times	q_4

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 s y un número par de b s 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		
\times	q_1		
\times		q_2	
\times			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

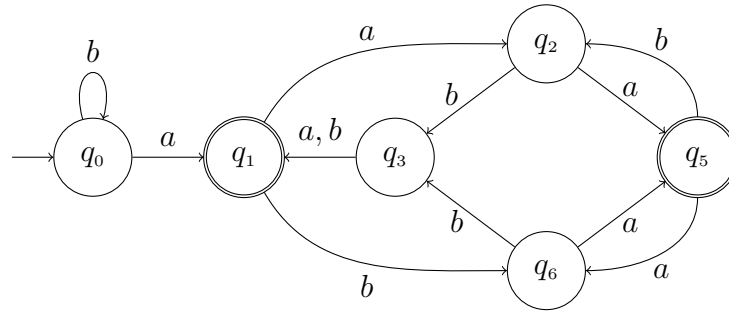
	q_0		
\times	q_1		
\times	\times	q_2	
\times	\times	\times	q_3

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.

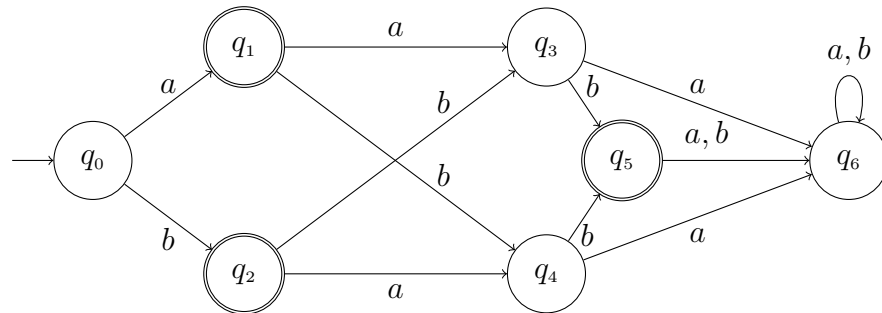
Ejercicios de la sección 2.15

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

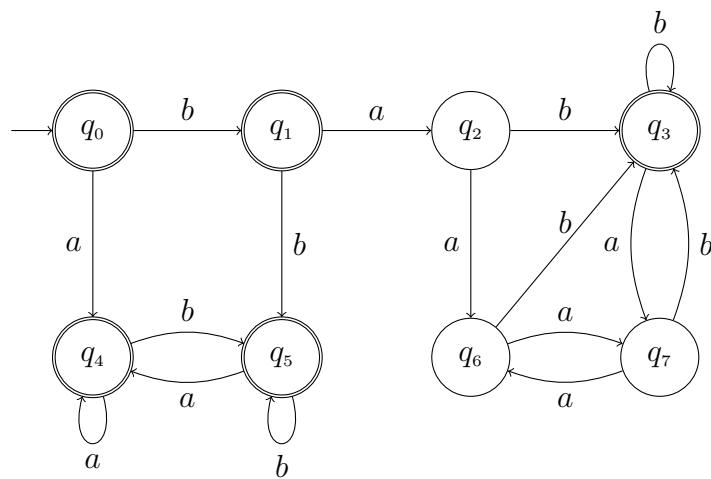
(i) Alfabeto $\Sigma = \{a, b\}$.



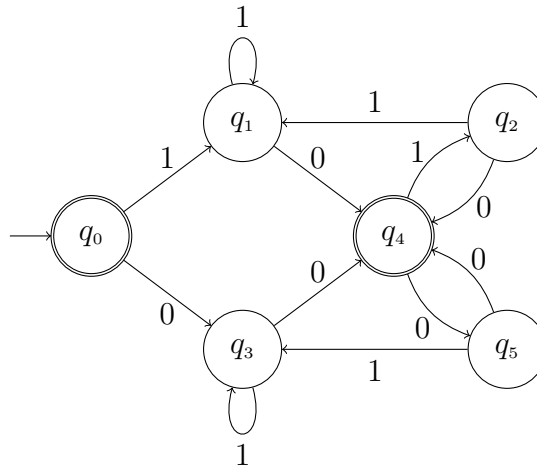
(ii) Alfabeto $\Sigma = \{a, b\}$.



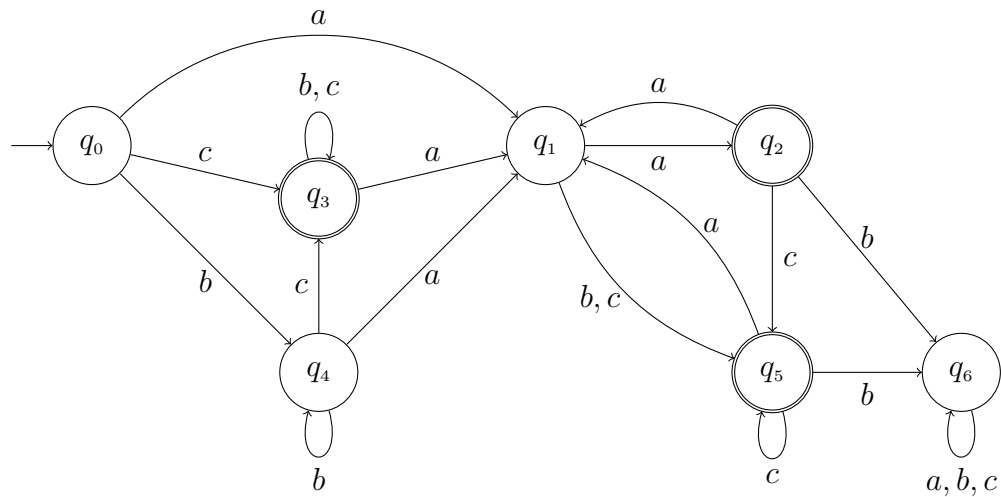
(iii) Alfabeto $\Sigma = \{a, b\}$.



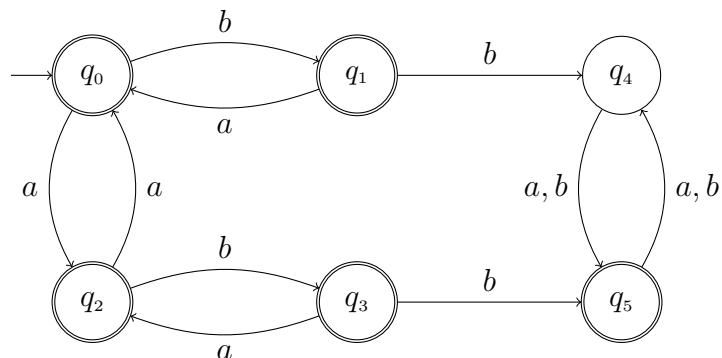
(iv) Alfabeto $\Sigma = \{0, 1\}$.



(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).

2.16. Minimización de autómatas. Parte II

En la presente sección se demuestra que, dado un AFD M , el autómata cociente M' , definido en la sección 2.15, resulta ser un autómata con el mínimo número de estados equivalente a M . La demostración requiere introducir la noción de distinguibilidad entre cadenas, la cual resulta también crucial en el Capítulo 3.

2.16.1 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

$$(2.16.1) \quad (\forall x \in \Sigma^*) [ux \in L \iff vx \in L].$$

Utilizaremos la notación uI_Lv para representar el hecho de que las cadenas u y v son L -indistinguibles.

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 (2.16.1); 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.

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

2.16.2 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.

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

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

Por lo tanto, uI_Lv . □

2.16.3 Corolario. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$. Si las cadenas u y v en Σ^* son L -distinguibiles, entonces $\widehat{\delta}(q_0, u) \neq \widehat{\delta}(q_0, v)$.

Demostración. El enunciado es la implicación contra-recíproca de la Proposición 2.16.2. \square

2.16.4 Proposición. Sea Σ un alfabeto dado y $L \subseteq \Sigma^*$ regular. Si hay n cadenas u_1, u_2, \dots, u_n en Σ^* que sean L -distinguibiles dos a dos (es decir, u_i y u_j son L -distinguibiles para todo $i \neq j$, $i, j = 1, \dots, n$), entonces cualquier AFD M que acepte a L debe tener por lo menos n estados.

Demostración. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD que acepte a L . Por el Corolario 2.16.3, $\widehat{\delta}(q_0, u_i) \neq \widehat{\delta}(q_0, u_j)$ para todo $i \neq j$. Entonces los n estados

$$\widehat{\delta}(q_0, u_1), \widehat{\delta}(q_0, u_2), \dots, \widehat{\delta}(q_0, u_n)$$

son diferentes entre sí (diferentes dos a dos) y, por lo tanto, M tiene por lo menos n estados. \square

El siguiente enunciado es un recíproco parcial de la Proposición 2.16.2.

2.16.5 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 -indistinguibiles, 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

$$(2.16.2) \quad \widehat{\delta}(p, w) = \widehat{\delta}(\widehat{\delta}(q_0, u), w) = \widehat{\delta}(q_0, uw),$$

$$(2.16.3) \quad \widehat{\delta}(q, w) = \widehat{\delta}(\widehat{\delta}(q_0, v), w) = \widehat{\delta}(q_0, vw).$$

Entonces

$$\begin{aligned} \widehat{\delta}(p, w) \in F &\iff \widehat{\delta}(q_0, uw) \in F \quad (\text{por (2.16.2)}) \\ &\iff uw \in L(M) = L \\ &\iff vw \in L \quad (\text{por ser } u \text{ y } v \text{ indistinguibiles}) \\ &\iff \widehat{\delta}(q_0, vw) \in F \\ &\iff \widehat{\delta}(q, w) \in F \quad (\text{por (2.16.3)}). \end{aligned} \quad \square$$

2.16.6 Corolario. Sean $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$, y dos cadenas $u, v \in \Sigma^*$. Si $\widehat{\delta}(q_0, u) \not\approx \widehat{\delta}(q_0, v)$, entonces u y v son L -distinguibiles.

Demostración. Contrarecíproca de la Proposición 2.16.5. \square

2.16.7 Teorema. Sea M un AFD cuyos estados son todos accesibles. El autómata cociente M' (definido en la sección 2.15) es un AFD equivalente a M , con el mínimo número de estados posible para aceptar el lenguaje $L(M)$.

Demostración. Sea M el AFD $M = (\Sigma, Q, q_0, F, \delta)$. Los estados del autómata cociente $M' = (\Sigma, Q', q'_0, F', \delta')$ son todos accesibles. En efecto, dado $q \in Q$, existe u tal que $\widehat{\delta}(q_0, u) = q$ ya que q es accesible en M . Se sigue que $\widehat{\delta'}([q_0], u) = [\widehat{\delta}(q_0, u)] = [q]$ y, por lo tanto $[q]$ es accesible.

Supóngase ahora que M' tiene n estados diferentes, $[q_0], [q_1], \dots, [q_{n-1}]$, formados por clases de equivalencia del autómata M . Como los estados q_0, q_1, \dots, q_{n-1} de M son accesibles, existen cadenas x_0, x_1, \dots, x_{n-1} tales que $\widehat{\delta}(q_0, x_0) = q_0, \widehat{\delta}(q_0, x_1) = q_1, \dots, \widehat{\delta}(q_0, x_{n-1}) = q_{n-1}$. Puesto que $q_i \not\approx q_j$ si $i \neq j$, se sigue del Corolario 2.16.6 que x_i es distinguible de x_j si $i \neq j$. Por lo tanto, x_0, x_1, \dots, x_{n-1} son n cadenas distinguibles dos a dos. Por la Proposición 2.16.4, cualquier AFD que acepte a L debe tener por lo menos n estados. En consecuencia, n es el mínimo número de estados posible. \square

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 las 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:

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

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; en la sección 2.13 resolveremos este problema utilizando autómatas.

2.3. Autómatas finitos deterministas (AFD)

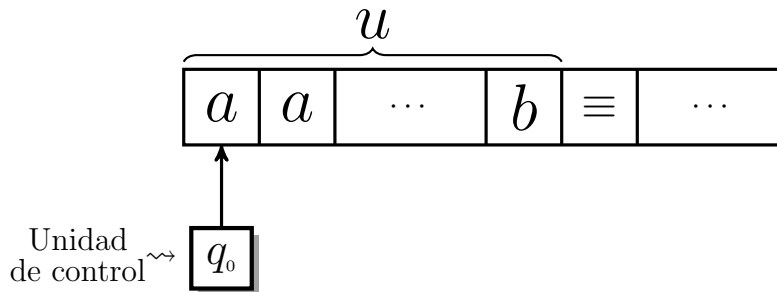
Los *autómatas finitos* son máquinas abstractas que leen de izquierda a derecha cadenas de entrada, las cuales son aceptadas o rechazadas. Un autómata actúa leyendo los símbolos escritos sobre una cinta semi-infinita, dividida en celdas o casillas, sobre la cual se escribe una cadena de entrada u , un 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:

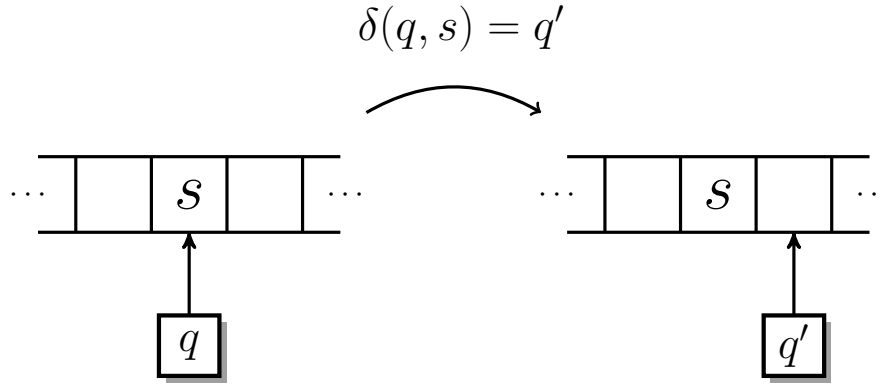
1. Un alfabeto Σ , llamado alfabeto de entrada o alfabeto de cinta. Todas las cadenas que lee M pertenecen a Σ^* .
2. $Q = \{q_0, q_1, \dots, q_n\}$, el conjunto (finito) de los estados internos de la unidad de control.
3. $q_0 \in Q$, estado inicial.
4. $F \subseteq Q$, conjunto de estados finales o 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

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

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

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

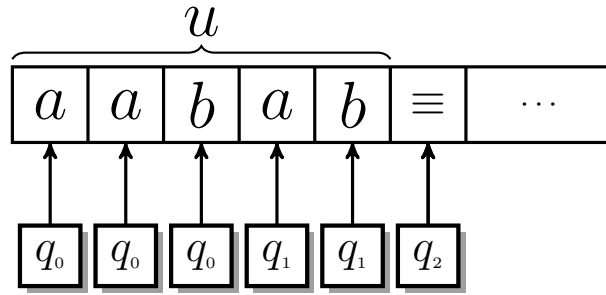
$$\delta(q_1, b) = q_2$$

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

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

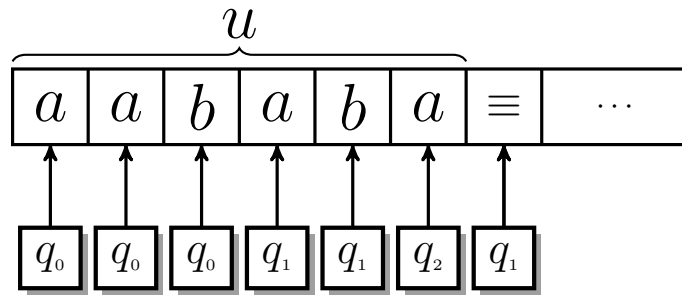
Vamos a ilustrar el procesamiento de tres cadenas de entrada.

1. $u = aabab$.



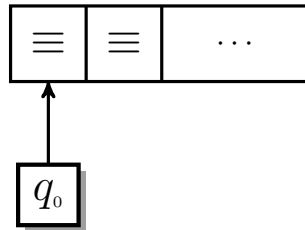
Como q_2 es un estado de aceptación, la cadena de entrada u es aceptada.

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.


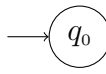

Un autómata M es entonces un mecanismo que clasifica todas las cadenas de Σ^* en dos clases disyuntas: las que son aceptadas y las que son rechazadas. El conjunto de las cadenas aceptadas por M se llama *lenguaje aceptado* o *lenguaje reconocido* por M y se denota por $L(M)$. Es decir,

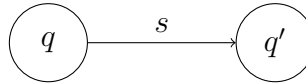
$$\begin{aligned} L(M) &:= \{u \in \Sigma^* : u \text{ es aceptada por } M\} \\ &= \{u \in \Sigma^* : M \text{ termina el procesamiento de } u \text{ en un estado } q \in F\} \end{aligned}$$

2.4. Grafo de un autómata

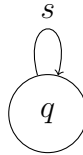
Un autómata finito se puede representar por medio de un grafo dirigido y etiquetado. Recuerdese que un grafo es un conjunto de vértices o nodos unidos por arcos o conectores; si los arcos tienen tanto dirección como etiquetas, el grafo se denomina *grafo dirigido y etiquetado* o *digrafo etiquetado*.

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

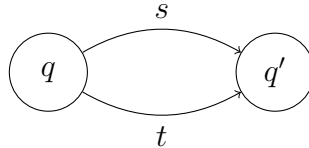
- Los vértices o nodos son los estados del autómata.
- Un estado $q \in Q$ se representa por un círculo con nombre q : 
- El estado inicial q_0 se destaca mediante una “bandera” o “flecha”: 
- Un estado de aceptación q se representa por un círculo doble con nombre q : 
- La transición $\delta(q, s) =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



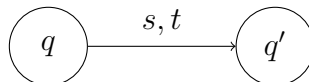
Si $\delta(q, s) = q$ se obtiene lo que se llama un “bucle” (*loop*, en inglés):



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

$\delta(q_0, a) = q_0$

$\delta(q_0, b) = q_1$

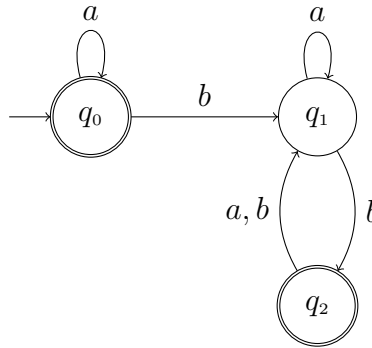
$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_2$

$\delta(q_2, a) = q_1$

$\delta(q_2, b) = q_1$

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.

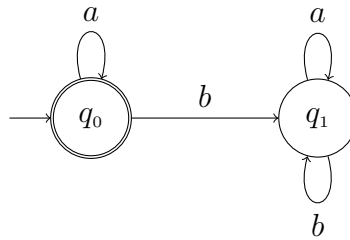
2.5. Diseño de autómatas

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:

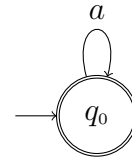
1. Si una cadena u es aceptada por M , entonces u debe pertenecer a L .
2. Recíprocamente, si $u \in L$, entonces u debe ser aceptada por M .

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.

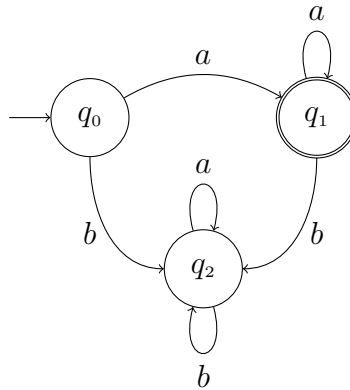
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^* = \{\lambda, a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



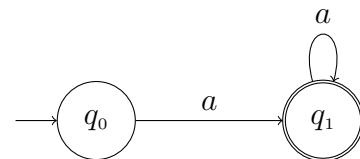
En la versión simplificada omitimos el estado limbo q_1 :



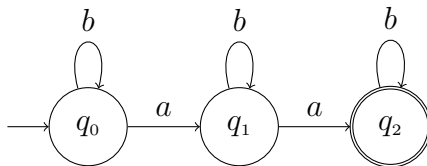
Ejemplo Sea $\Sigma = \{a, b\}$ y $L = a^+ = \{a, a^2, a^3, \dots\}$. AFD M tal que $L(M) = L$:



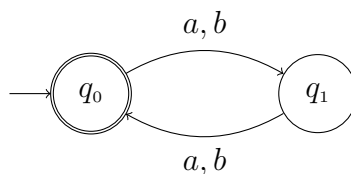
En la versión simplificada omitimos el estado limbo q_2 :



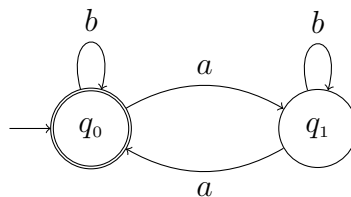
Ejemplo $\Sigma = \{a, b\}$. $L =$ lenguaje de las cadenas que contienen exactamente dos a 's $= b^*ab^*ab^*$. AFD M tal que $L(M) = L$, omitiendo el estado limbo:



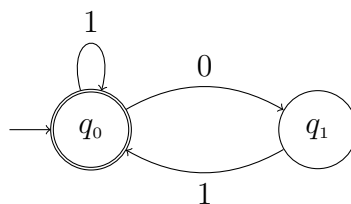
Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que tienen un número par de símbolos (cadenas de longitud par ≥ 0). AFD M tal que $L(M) = L$:



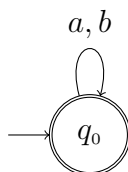
Ejemplo $\Sigma = \{a, b\}$. L = lenguaje de las cadenas sobre Σ que contienen un número par de a 'es. AFD M tal que $L(M) = L$:



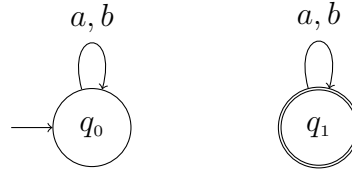
Ejemplo Sean $\Sigma = \{0, 1\}$ y $L = (1 \cup 01)^*$. Construimos un AFD de tal forma que el estado inicial q_0 sea el único estado de aceptación y en él confluyan las trayectorias 1 y 01. Omitiendo el estado limbo obtenemos el siguiente autómata:



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

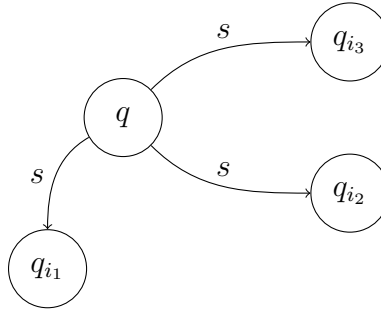
$$\begin{aligned} \Delta : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, s) &\longmapsto \Delta(q, s) = \{q_{i_1}, q_{i_2}, \dots, q_{i_k}\} \end{aligned}$$

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:



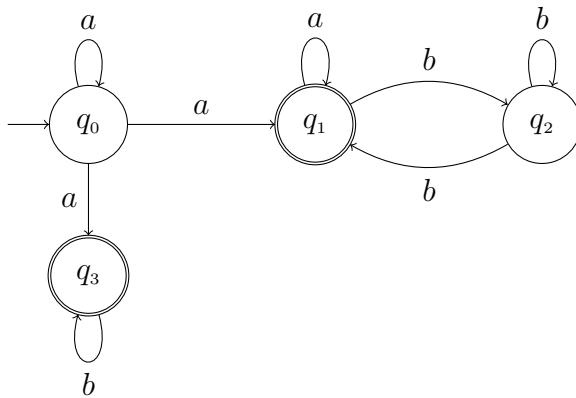
Un AFN M puede procesar una cadena de entrada $u \in \Sigma^*$ de varias maneras. Sobre el grafo del autómata, esto significa que pueden existir varias trayectorias, desde el estado q_0 , etiquetadas con los símbolos de u .

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.

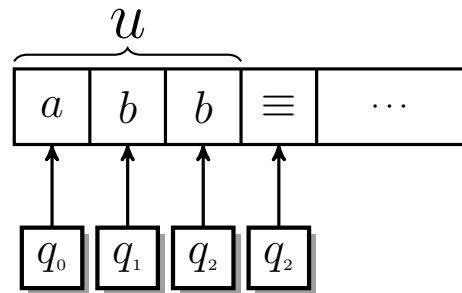
Ejemplo Sea M el siguiente AFN:



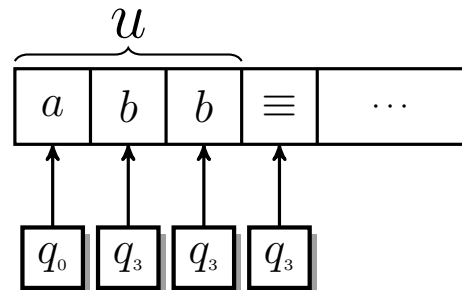
Δ	a	b
q_0	$\{q_0, q_1, q_3\}$	\emptyset
q_1	$\{q_1\}$	$\{q_2\}$
q_2	\emptyset	$\{q_1, q_2\}$
q_3	\emptyset	$\{q_3\}$

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, $u \in L(M)$.

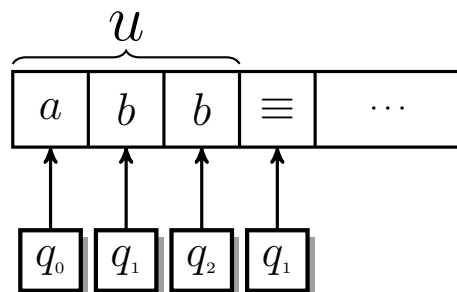
Procesamiento de rechazo:



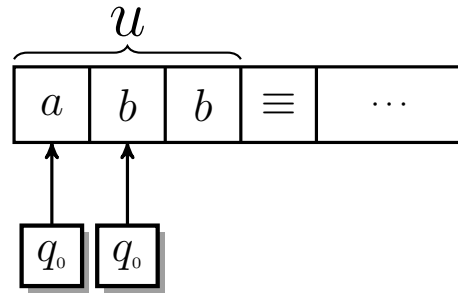
Procesamiento de aceptación:



Otro procesamiento de aceptación:

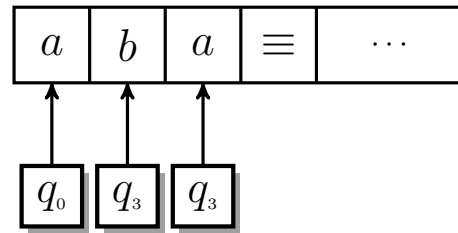


Procesamiento abortado:



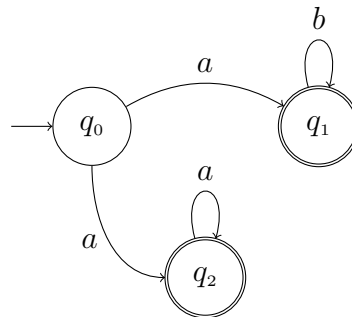
El grafo de M nos permite ver que la cadena $aabbaa$ es aceptada mientras que $aabaa$ es rechazada. Todas las cadenas que comienzan con b son rechazadas. También es rechazada la cadena aba ; uno de sus procesamientos posibles es el siguiente:

Procesamiento abortado:



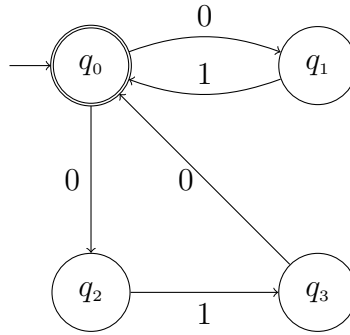
A pesar de que el procesamiento anterior termina en el estado de aceptación q_3 , la entrada no es aceptada porque no se consume completamente.

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

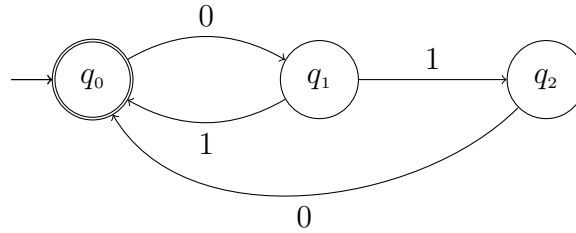


Un AFD que acepte a L requiere como mínimo cuatro estados más un estado limbo (cinco estados en total).

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.

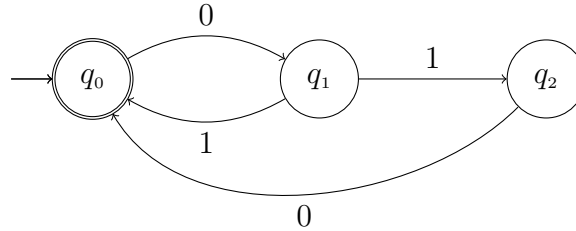
2.7.1 Teorema. Dado un AFN $M = (\Sigma, Q, q_0, F, \Delta)$ se puede construir un AFD M' equivalente a M , es decir, tal que $L(M) = L(M')$.

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 $\Sigma = \{0, 1\}$.

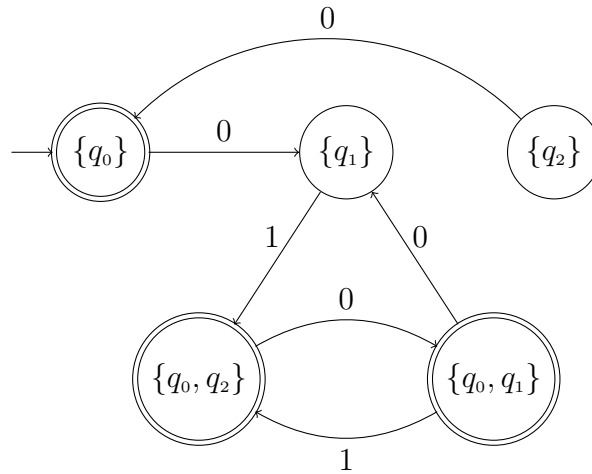


La tabla original de la función de transición Δ de M contiene la combinación $\{q_0, q_2\}$ que adquiere el estatus de nuevo estado en M' . Al extender la tabla de Δ aparece también el nuevo estado $\{q_0, q_1\}$:

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
$\{q_0, q_1\}$	$\{q_1\}$	$\{q_0, q_2\}$

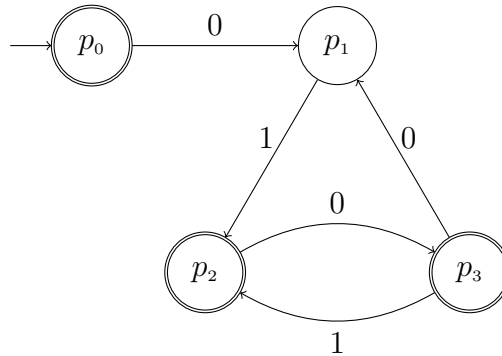
La tabla original de la función Δ y su extensión.

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:

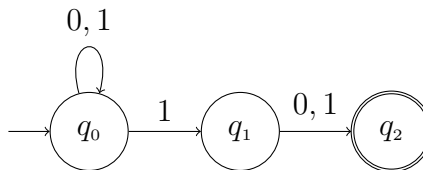


Los estados de aceptación son aquéllos en los que aparezca q_0 ya que q_0 es el único estado de aceptación del autómata original.

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:



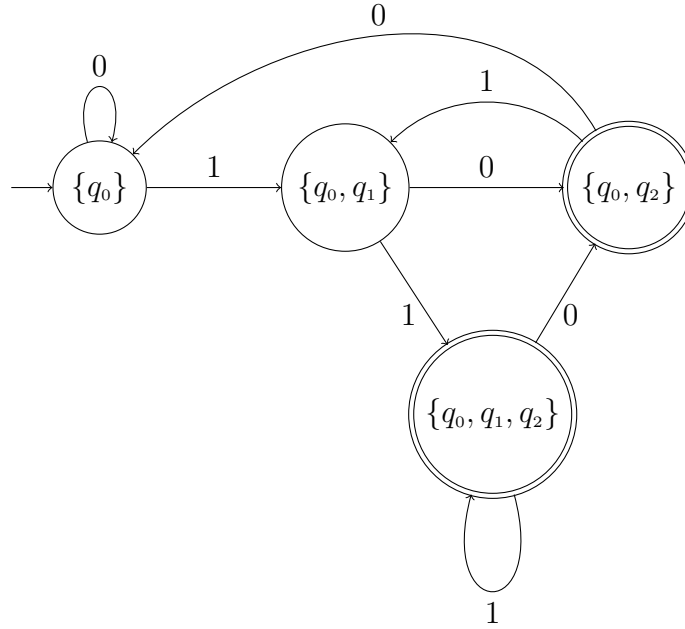
Ejemplo Sean $\Sigma = \{0, 1\}$ y L_2 el lenguaje de todas las cadenas de longitud ≥ 2 en las que el segundo símbolo, de derecha a izquierda es un 1. Una expresión regular para este lenguaje es $(0 \cup 1)^* 1 (0 \cup 1)$ y es fácil diseñar un AFN M que acepte a L_2 :



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 $u \in \Sigma^*$, $\hat{\delta}(q, u)$ es el estado en el que el autómata termina el procesamiento de u a partir del estado q . En particular, $\hat{\delta}(q_0, u)$ 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:

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

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

2.7.3 Definición. Sea $M = (\Sigma, Q, q_0, F, \Delta)$ un AFN. La función de transición Δ , $\Delta : Q \times \Sigma \longrightarrow \wp(Q)$ 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 $\widehat{\Delta} : Q \times \Sigma^* \longrightarrow \wp(Q)$, 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

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

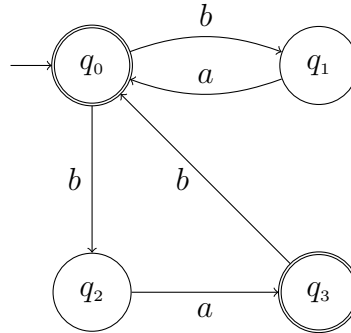
Finalmente podemos demostrar que $L(M') = L(M)$:

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

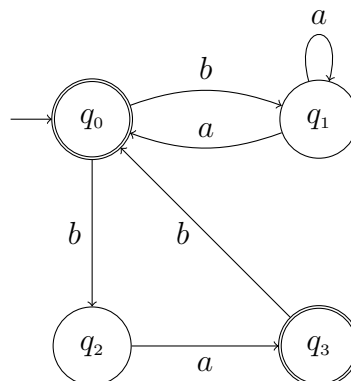
Ejercicios de la sección 2.7

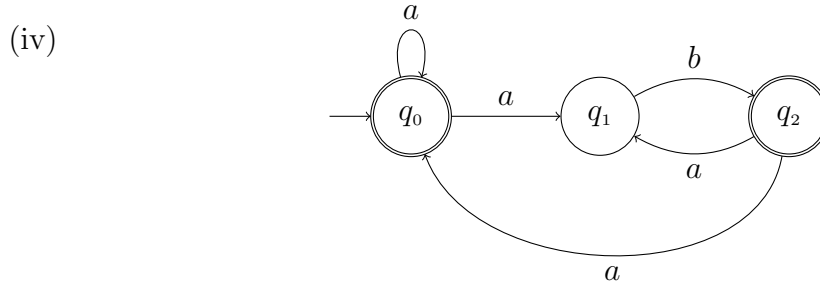
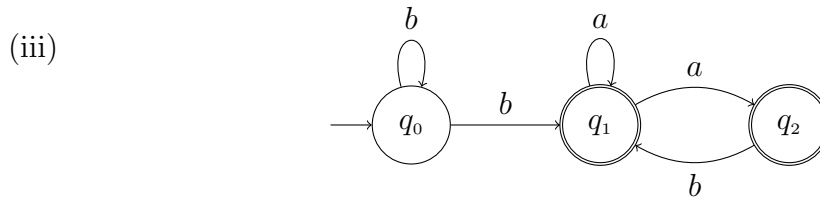
- ① Utilizando el procedimiento de conversión presentado en esta sección, encontrar AFD equivalentes a los siguientes AFN:

(i)



(ii)





- 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

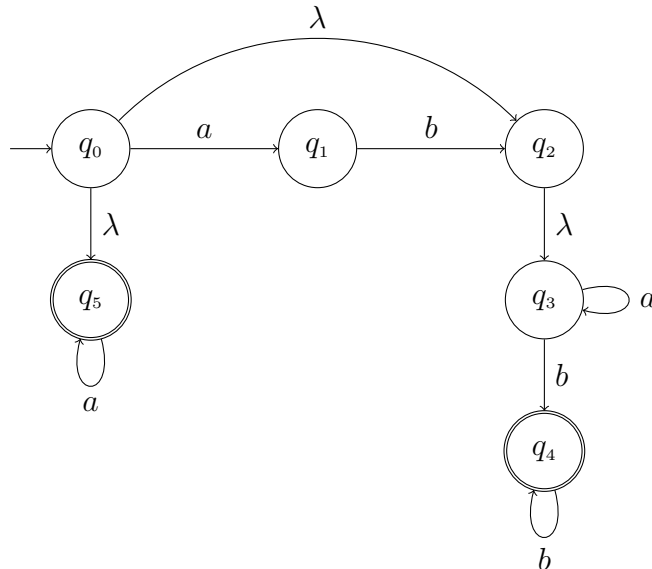
$$\Delta(q, \lambda) = \{q_{i_1}, \dots, q_{i_k}\}$$

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.

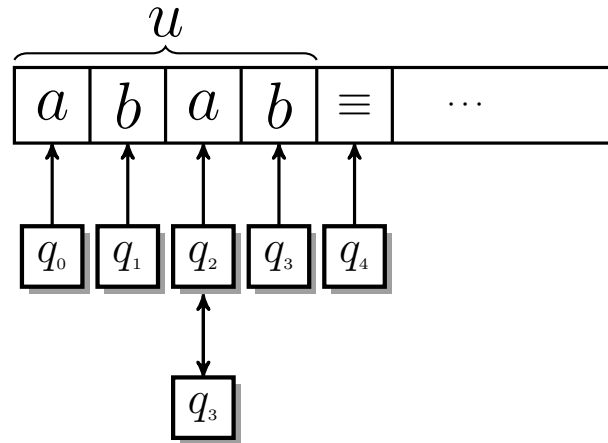
Como sucede en el caso AFN, una cadena de entrada $u \in \Sigma^*$ es aceptada si existe por lo menos un procesamiento completo de u , desde q_0 , que termina en un estado de aceptación. En el grafo del autómata, las transiciones λ dan lugar a arcos con etiquetas λ . Una cadena de entrada u es aceptada 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 procesamiento para una misma cadena de entrada, así como procesamiento abortados.

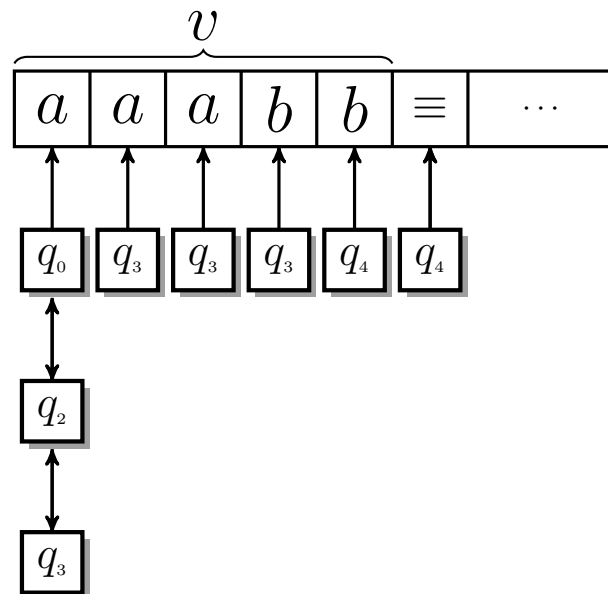
Ejemplo Consideremos el siguiente AFN- λ , M :



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 $\lambda\lambda abbb$.

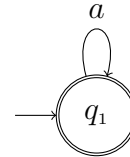
Los AFN- λ permiten aún más libertad en el diseño de autómatas, especialmente cuando hay numerosas uniones y concatenaciones.

Ejemplo Diseñar AFN- λ que acepten los siguientes lenguajes:

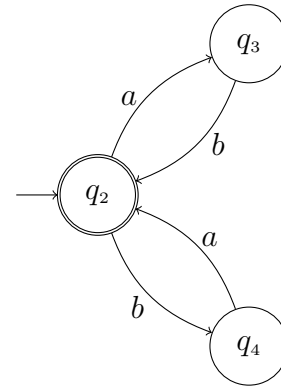
- (1) $a^* \cup (ab \cup ba)^* \cup b^+$.
- (2) $a^*(ab \cup ba)^*b^+$.

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.

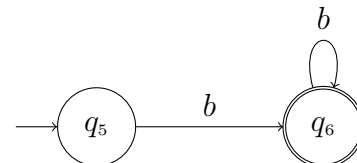
Autómata que acepta a^* :



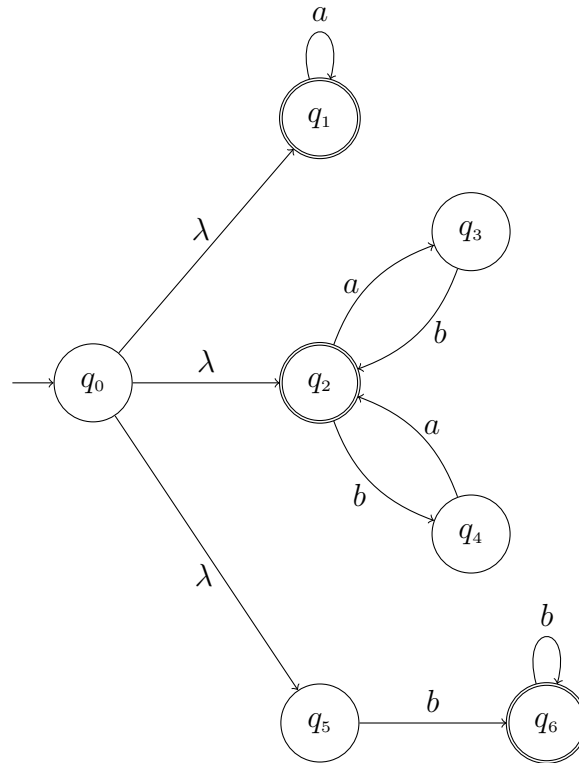
Autómata que acepta $(ab \cup ba)^*$:



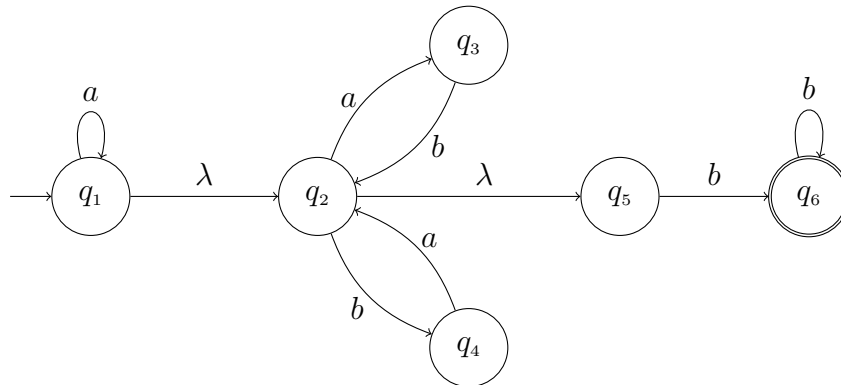
Autómata que acepta b^+ :



- (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.

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:

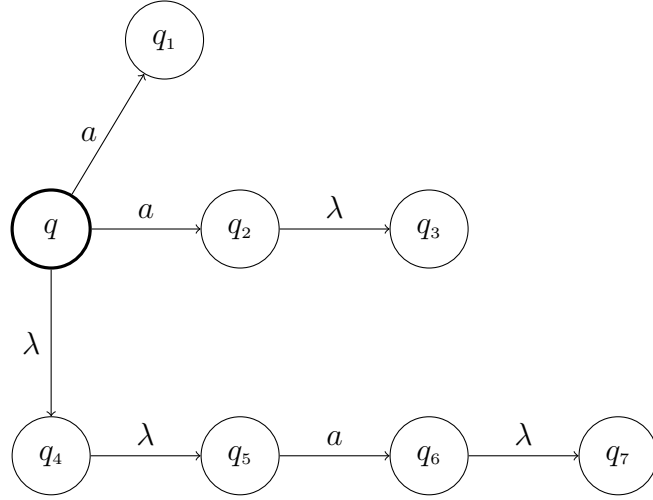
$$\begin{aligned} \Delta' : Q \times \Sigma &\longrightarrow \wp(Q) \\ (q, a) &\longmapsto \Delta'(q, a) := \lambda[\Delta(\lambda[q], a)]. \end{aligned}$$

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

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 :



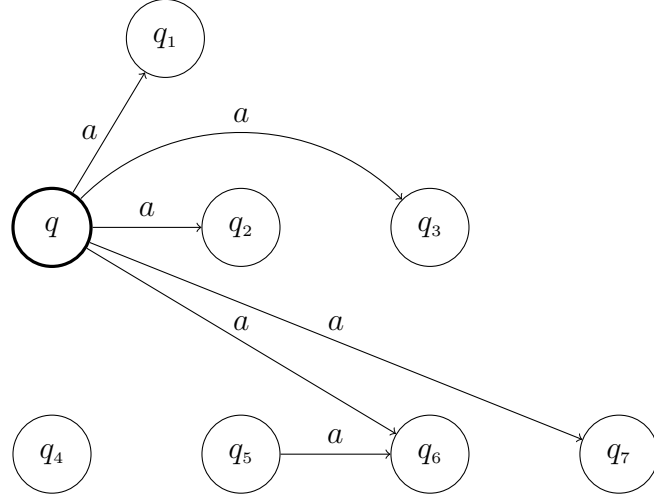
Por simple inspección observamos que, una vez procesada una a , el autómata puede pasar desde el estado q a uno de los siguientes estados: q_1, q_2, q_3, q_6, q_7 . Para obtener esta lista de estados se tienen en cuenta todas las transiciones λ que preceden o prosiguen el procesamiento del símbolo a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

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

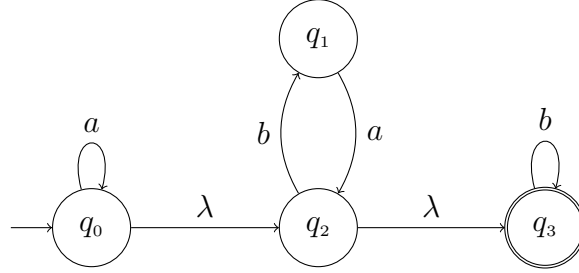
La porción correspondiente del grafo de M' se exhibe en la siguiente gráfica. De esta forma M' simula, sin transiciones λ , todas las transiciones λ de M añadiendo nuevas transiciones con etiqueta a .

Porción de M' :



Ejemplo

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



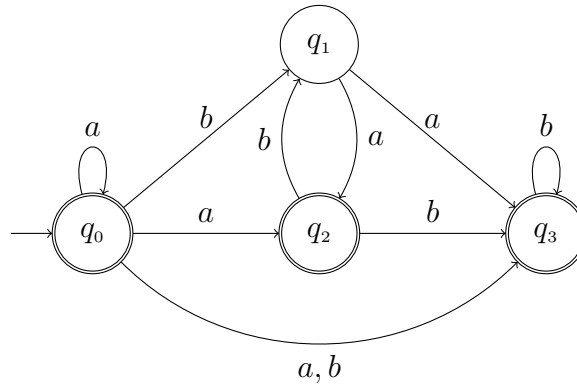
Las λ -clausuras de los estados vienen dadas por:

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

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:



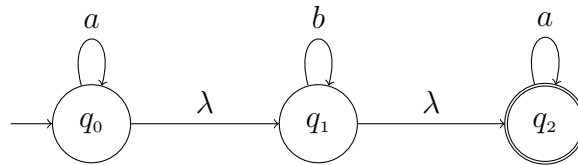
Puesto que q_3 , que es el único estado de aceptación del autómata original M , pertenece a $\lambda[q_0]$, a $\lambda[q_2]$ y a $\lambda[q_3]$, los tres estados q_0 , q_2 y q_3 son estados de aceptación en el autómata M' .

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.

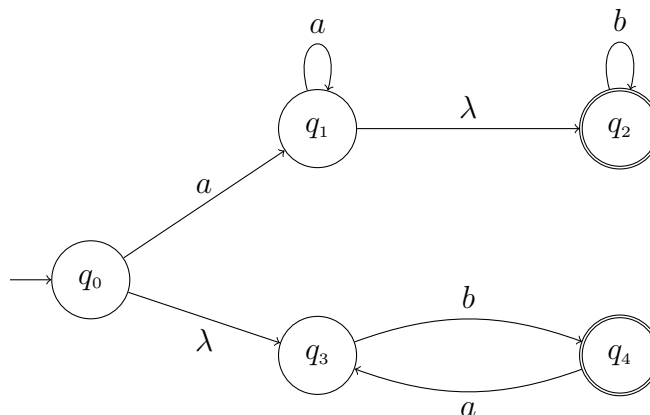
Ejercicios de la sección 2.9

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 $a \in \Sigma$.

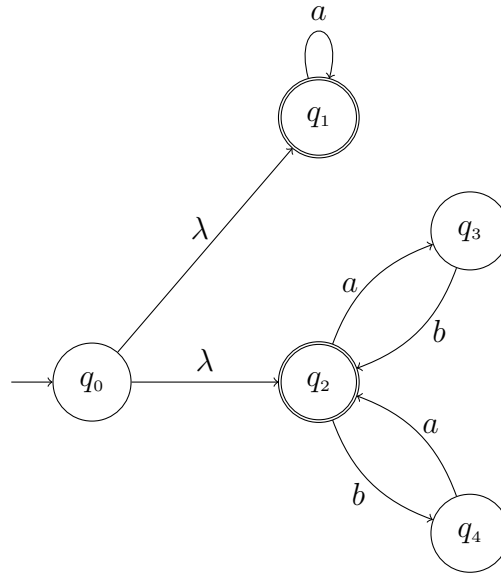
①



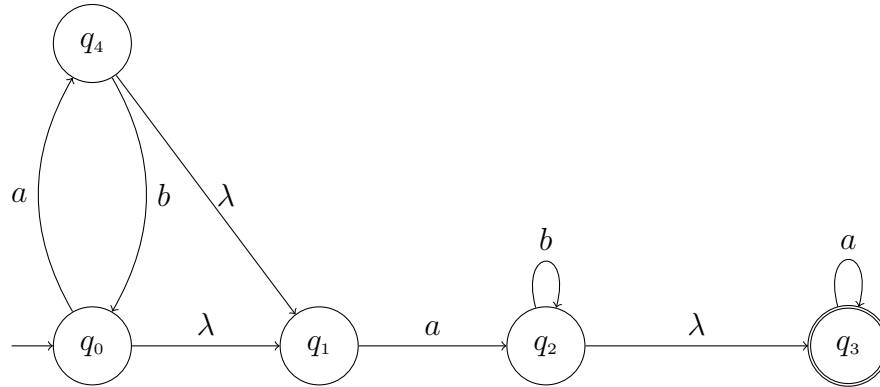
②



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 $L(\overline{M}) = \emptyset$.

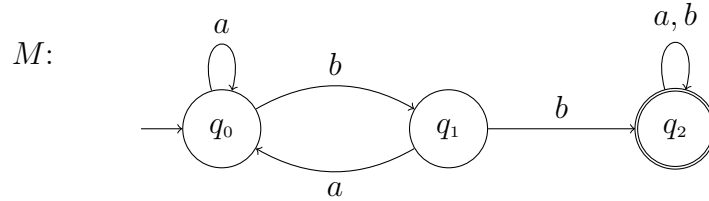
Cuando un lenguaje L está definido por medio de una condición negativa puede ser más fácil diseñar primero un AFD que acepte su complemento \overline{L} .

Ejemplo

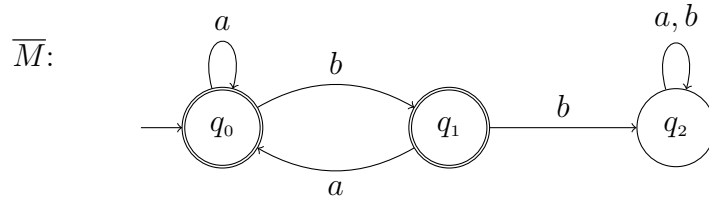
Sea $\Sigma = \{a, b\}$. Encontrar un AFD que acepte el lenguaje L de todas las cadenas que no tienen dos *b*es consecutivas (es decir, no contienen la subcadena

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 :



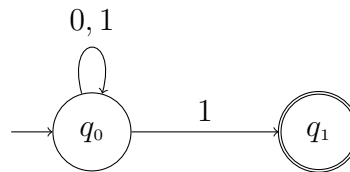
Para aceptar a L formamos el complemento de M , intercambiando aceptación con no-aceptación:



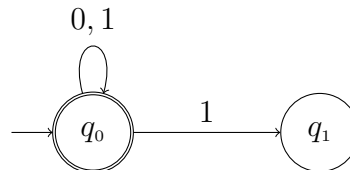
En \overline{M} , q_2 es estado limbo y $L(\overline{M}) = L$.

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

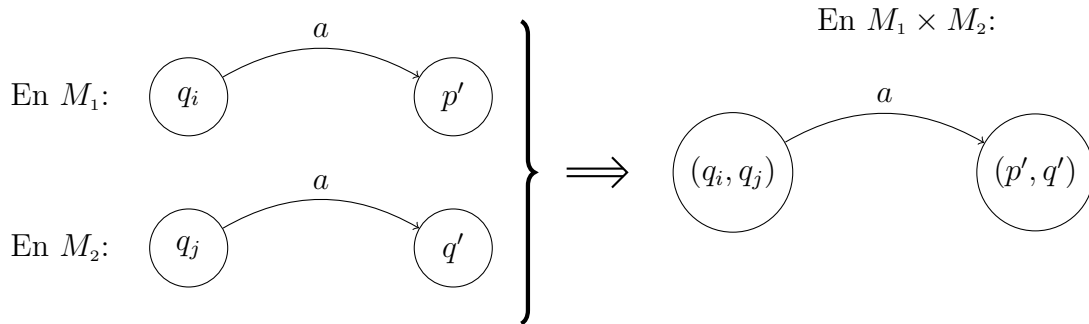
$$\begin{aligned} \delta : (Q_1 \times Q_2) \times \Sigma &\longrightarrow Q_1 \times Q_2 \\ \delta((q_i, q_j), a) &= (\delta_1(q_i, a), \delta_2(q_j, a)). \end{aligned}$$

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 $M_1 \times M_2$ de la siguiente manera:



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 $F = \{(q_i, q_j) : q_i \in F_1 \text{ o } q_j \in F_2\}$ 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 $F = \{(q_i, q_j) : q_i \in F_1 \text{ ó } q_j \notin F_2\}$ 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

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

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

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

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

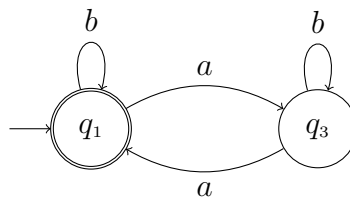
$$\begin{aligned}
 w \in L(M) &\iff (\hat{\delta}_1(q_1, u), \hat{\delta}_2(q_2, u)) \in F \\
 &\iff \hat{\delta}_1(q_1, u) \in F_1 \wedge \hat{\delta}_2(q_2, u) \notin F_2 \\
 &\iff u \in L(M_1) \wedge u \notin L(M_2) \\
 &\iff u \in L(M_1) - L(M_2) = L_1 - L_2.
 \end{aligned}$$

Esto demuestra (iii). □

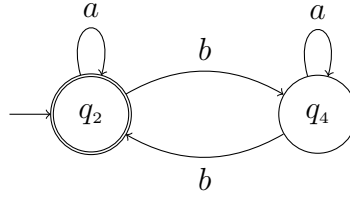
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen un número par de a s y un número par de b s.

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):



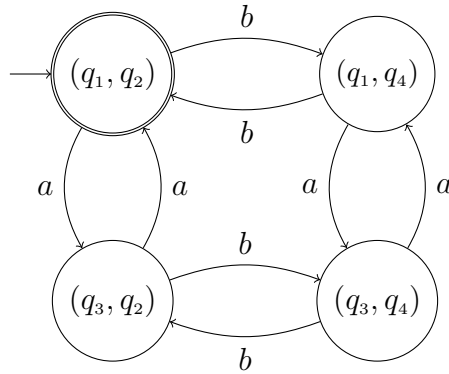
AFD M_2 que acepta L_2 (cadenas con un número par de *bes*):



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 $M_1 \times M_2$.

$$\begin{aligned}
 \delta((q_1, q_2), a) &= (\delta_1(q_1, a), \delta_2(q_2, a)) = (q_3, q_2), \\
 \delta((q_1, q_2), b) &= (\delta_1(q_1, b), \delta_2(q_2, b)) = (q_1, q_4), \\
 \delta((q_1, q_4), a) &= (\delta_1(q_1, a), \delta_2(q_4, a)) = (q_3, q_4), \\
 \delta((q_1, q_4), b) &= (\delta_1(q_1, b), \delta_2(q_4, b)) = (q_1, q_2), \\
 \delta((q_3, q_2), a) &= (\delta_1(q_3, a), \delta_2(q_2, a)) = (q_1, q_2), \\
 \delta((q_3, q_2), b) &= (\delta_1(q_3, b), \delta_2(q_2, b)) = (q_3, q_4), \\
 \delta((q_3, q_4), a) &= (\delta_1(q_3, a), \delta_2(q_4, a)) = (q_1, q_4), \\
 \delta((q_3, q_4), b) &= (\delta_1(q_3, b), \delta_2(q_4, b)) = (q_3, q_2).
 \end{aligned}$$

El grafo del autómata así obtenido es:



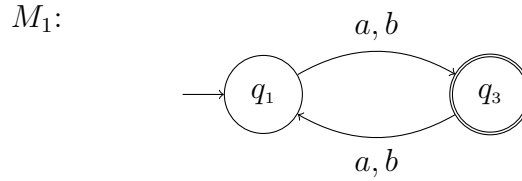
Ejemplo Utilizar el Teorema 2.11.1 para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos *bes* consecutivas, es decir, no contienen la subcadena bb .

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

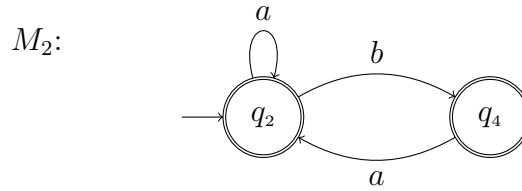
L_1 = lenguaje de todas las cadenas que tienen longitud impar.

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 :

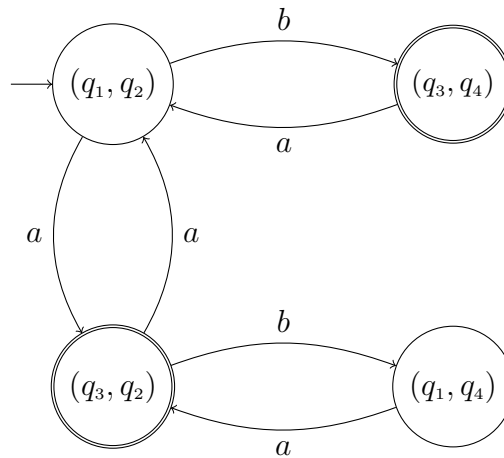


Y un AFD M_2 que acepta L_2 :



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.

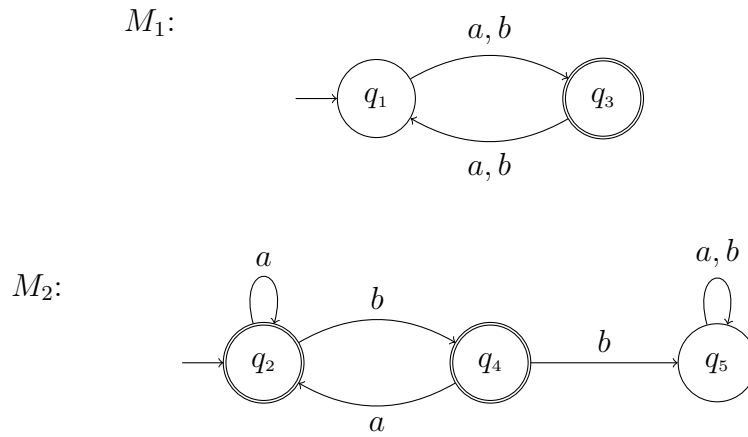
Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene 4 estados: (q_1, q_2) , (q_1, q_4) , (q_3, q_2) y (q_3, q_4) . Los estados de aceptación son (q_3, q_2) y (q_3, q_4) ya que q_3 es de aceptación en M_1 mientras que q_2 y q_4 son de aceptación en M_2 . Utilizando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



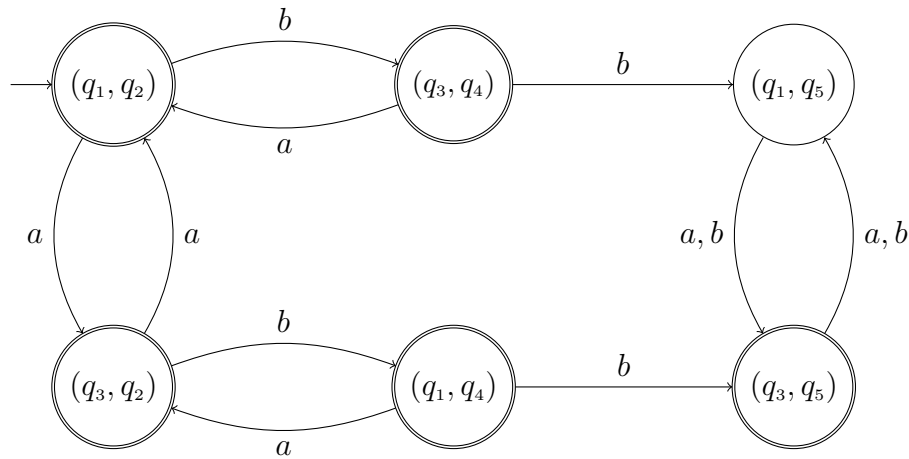
Este problema también se puede resolver expresando el lenguaje L como diferencia de dos lenguajes (véase el Ejercicio ① al final de la presente sección).

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 $M = M_1 \times M_2$ 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

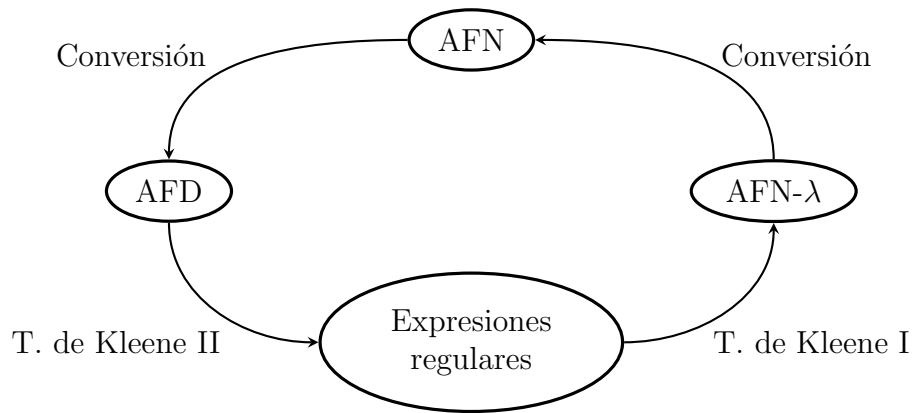
- ① Utilizar el Teorema 2.11.1 (iii) para construir un AFD que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen longitud impar y que no contienen dos *b*es consecutivas, expresando L como diferencia de dos lenguajes.
- ② Utilizar el Teorema 2.11.1 para construir 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.

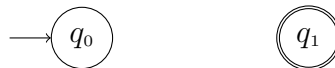


2.12.1. Teorema de Kleene. Sea Σ un alfabeto dado. Un lenguaje es regular (sobre Σ) si y sólo si es aceptado por un autómata finito (AFD o AFN o AFN- λ) con alfabeto de entrada Σ .

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



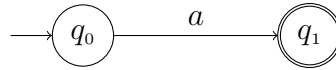
acepta el lenguaje \emptyset , es decir, el lenguaje representado por la expresión regular $R = \emptyset$.

El autómata



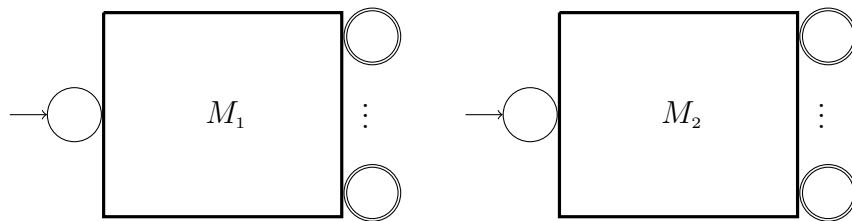
acepta el lenguaje $\{\lambda\}$, es decir, el lenguaje representado por la expresión regular $R = \lambda$.

El autómata



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

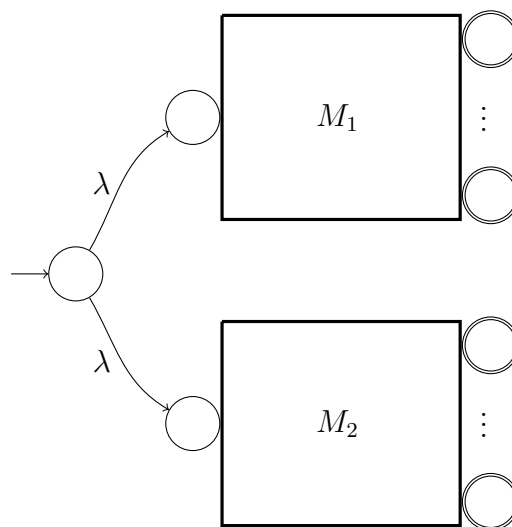
Razonando recursivamente, supóngase que para las expresiones regulares R_1 y R_2 se dispone de AFN- λ M_1 y M_2 tales que $L(M_1) = R_1$ y $L(M_2) = R_2$. Esquemáticamente vamos a presentar los autómatas M_1 y M_2 en la siguiente forma:



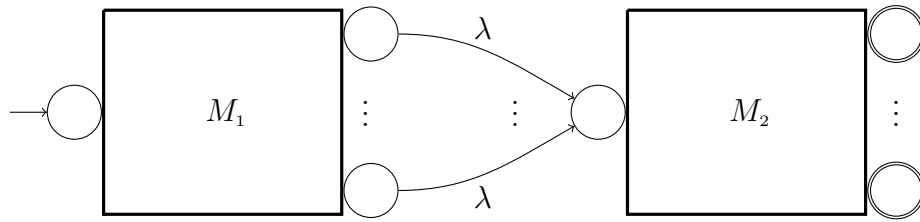
Los estados finales o de aceptación se dibujan a la derecha, pero cabe advertir que el estado inicial puede ser también un estado de aceptación. Podemos ahora obtener AFN- λ que acepten los lenguajes $R_1 \cup R_2$ y $R_1 R_2$.

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 $R_1 \cup R_2$:

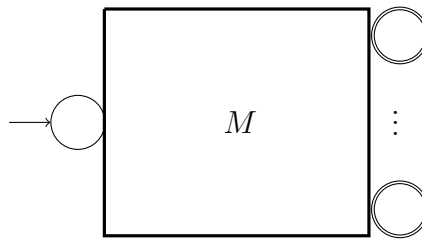


Autómata que acepta $R_1 R_2$:

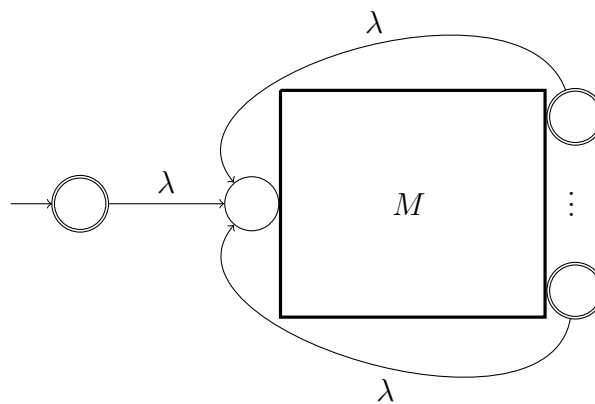


Este tipo de conexión entre dos autómatas M_1 y M_2 se denomina *conexión en serie*. Los estados de aceptación del nuevo autómata son únicamente los estados de aceptación de M_2 .

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

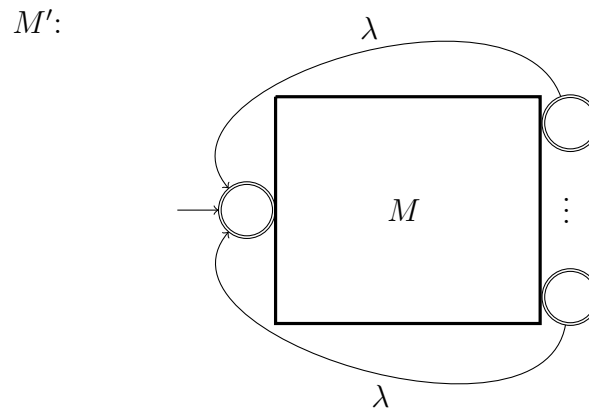


El siguiente autómata acepta R^* :

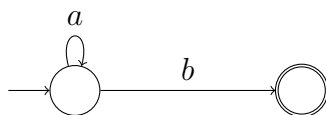
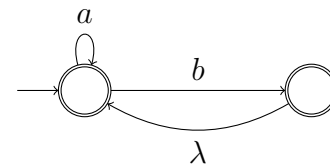


Esto concluye la demostración de la parte I del Teorema de Kleene. □

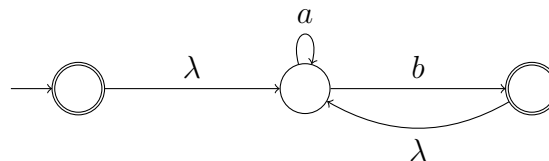
Para aceptar R^* a partir de un autómata M tal que $L(M) = R$, sería incorrecto (en general) utilizar el siguiente autómata M' :



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' :

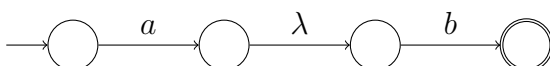
No se cumple que $L(M') = R^* = (a^*b)^*$ ya que M' acepta también a, a^2, a^3, \dots (potencias de a) que no pertenecen a R^* . Para aceptar R^* utilizando la construcción mencionada en la demostración del Teorema 2.12.1 se obtendría el siguiente autómata M'' :

 M'' :

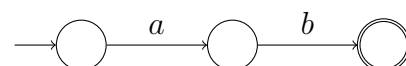
Simplificaciones en el procedimiento. El procedimiento constructivo del Teorema 2.12.1 admite varias simplificaciones, útiles en la práctica.

Para aceptar ab :

Según el procedimiento:

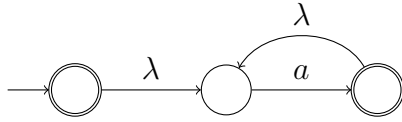


Simplificación:

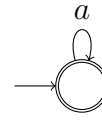


Para aceptar a^* :

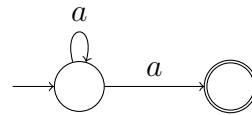
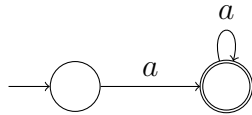
Según el procedimiento:



Simplificación:



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

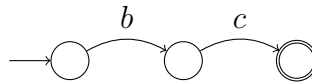


Además, las conexiones en paralelo y en serie de dos autómatas se pueden generalizar fácilmente para aceptar uniones $R_1 \cup R_2 \cup R_3 \cup \dots \cup R_k$, o concatenaciones $R_1 R_2 R_3 \dots R_k$, de k lenguajes ($k \geq 3$).

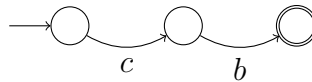
Ejemplo Utilizar el procedimiento del Teorema 2.12.1 para construir un AFN- λ que acepte el lenguaje $(bc \cup cb)^* a^* b \cup (b^* ca)^* c^+$ sobre el alfabeto $\Sigma = \{a, b, c\}$.

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.

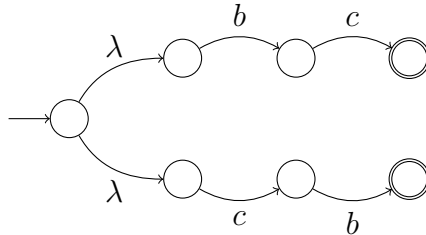
Autómata que acepta bc :



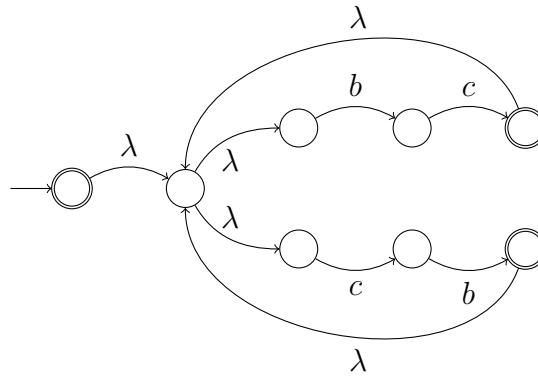
Autómata que acepta cb :



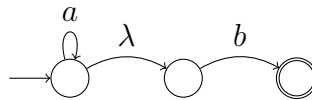
Autómata que acepta $bc \cup cb$:



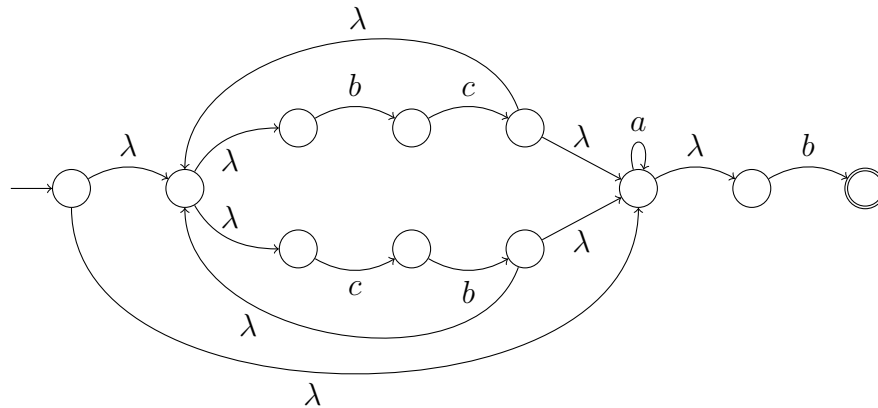
Autómata que acepta $(bc \cup cb)^*$:



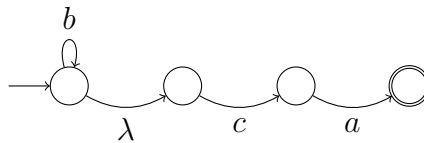
Autómata que acepta a^*b :



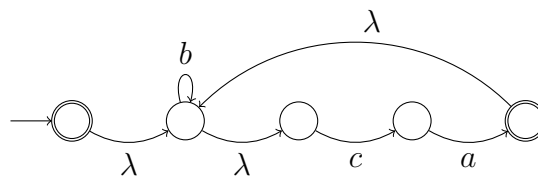
Para aceptar $R_1 = (bc \cup cb)^*a^*b$ conectamos en serie los dos últimos autómatas:



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 $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^* \cup (b \cup \lambda)(b^*ac)^*$.

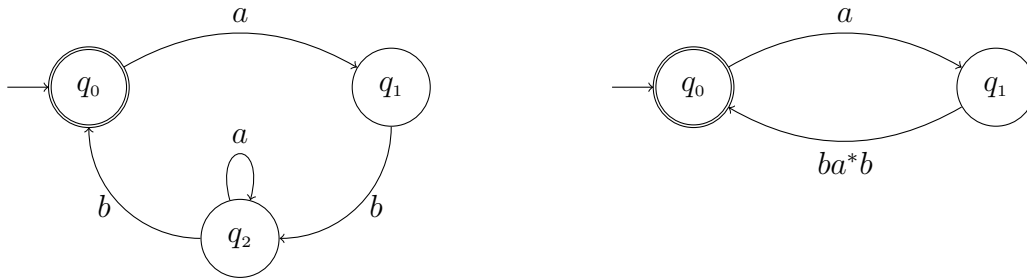
2.13. Teorema de Kleene, parte II

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:

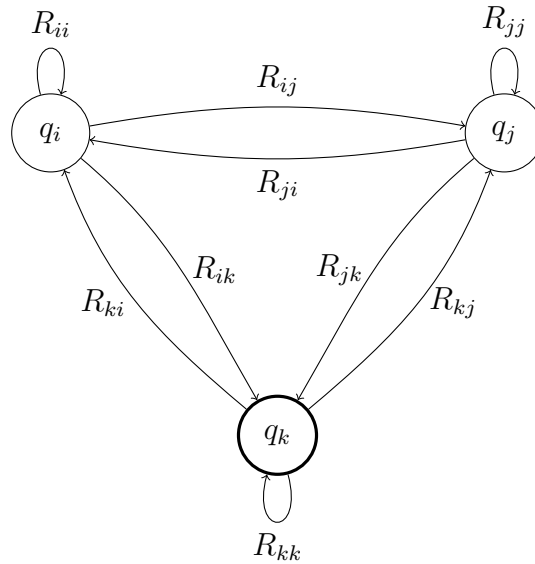
- (1) Convertir el grafo de M en un GEG G reemplazando múltiples arcos etiquetados con símbolos a_1, a_2, \dots, a_k entre dos estados q_i y q_j por un único arco etiquetado $a_1 \cup a_2 \cup \dots \cup a_k$:



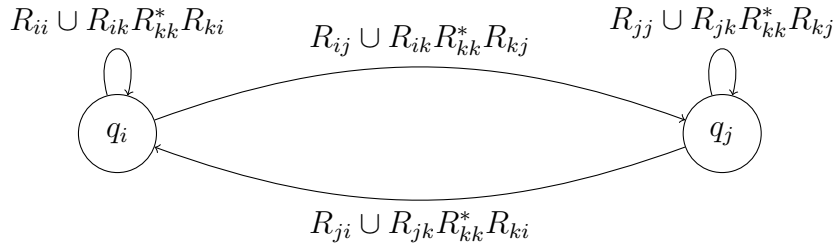
De esta forma, entre dos estados cualesquiera hay, a lo sumo, un arco etiquetado con una expresión regular.

- (2) Modificar el GEG G de tal manera que haya un único estado de aceptación. Esto se puede conseguir añadiendo un nuevo estado q_f , que será el único estado de aceptación, y trazando transiciones λ entre todos y cada uno de los estados en F (los estados de aceptación originales) y q_f . Cuando el autómata original posee un único estado de aceptación, simplemente se mantiene como tal hasta el final.
- (3) Este paso es un proceso 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:

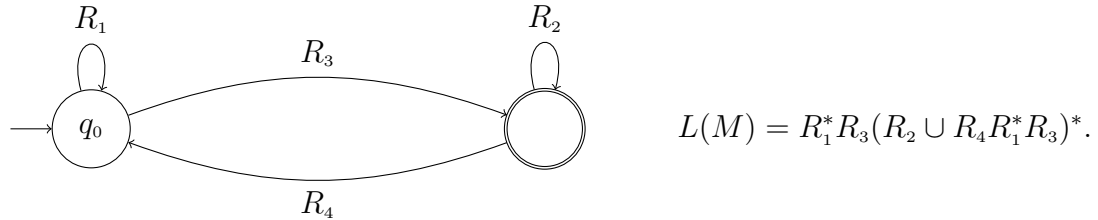
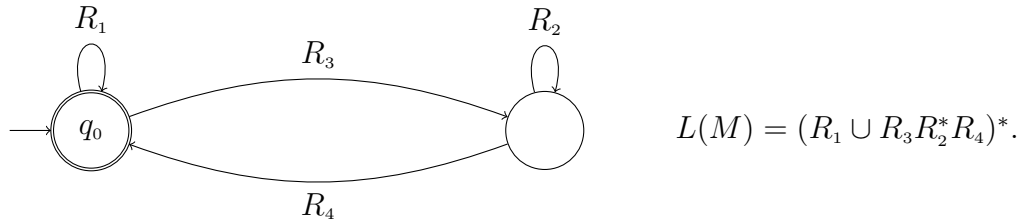


q_k sirve de “puente” entre q_i y q_j , y entre q_j y q_i . Además, a través de q_k hay una trayectoria que conecta q_i consigo mismo, y también una trayectoria que conecta q_j consigo mismo. Teniendo en cuenta tales trayectorias, se procede a eliminar el estado q_k reemplazando las etiquetas entre q_i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

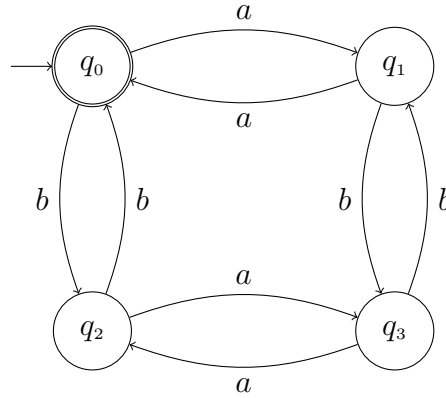


Observaciones:

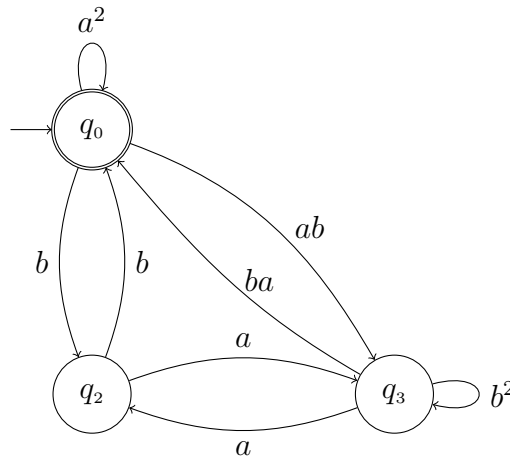
1. El procedimiento anterior es bastante flexible; por ejemplo, el paso (2) (estado de aceptación único) se puede realizar después de haber eliminado uno o más estados (siguiendo las instrucciones del paso (3)).
2. Es importante recalcar que, siempre que se ejecute la subrutina (3), el estado inicial y los estados de aceptación no se pueden eliminar.

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.

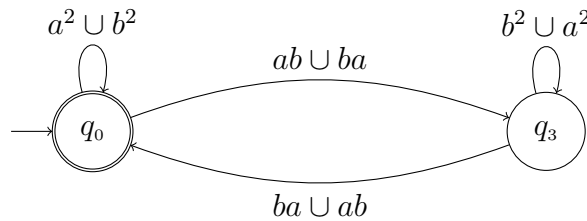
Solución. Conocemos un AFD M que acepta este lenguaje:



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:



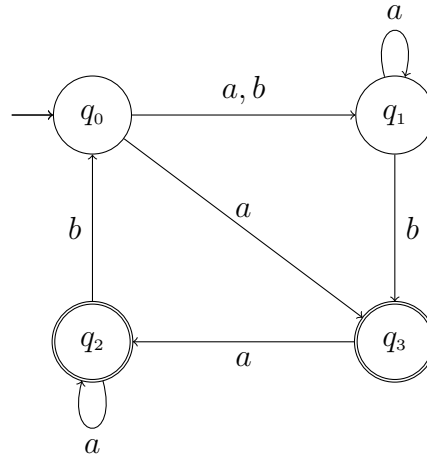
El lenguaje aceptado es entonces

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

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.

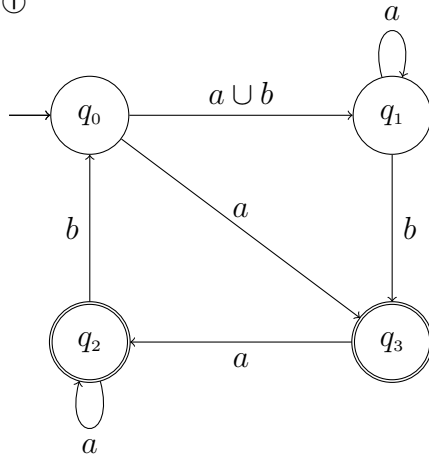
Ejemplo Encontrar una expresión regular para el lenguaje aceptado por el siguiente autómata M .

M :

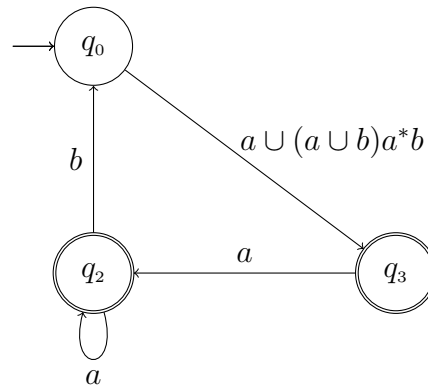


Solución. Primero convertimos el grafo de M en un GEG (grafo ①) y luego eliminamos el estado q_1 (grafo ②).

①

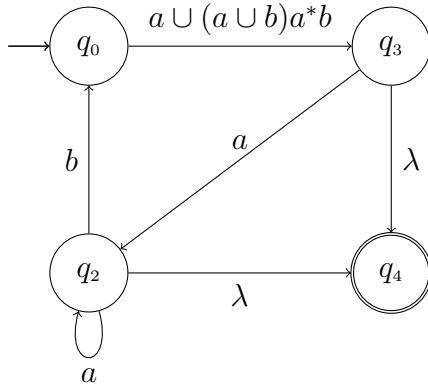


②

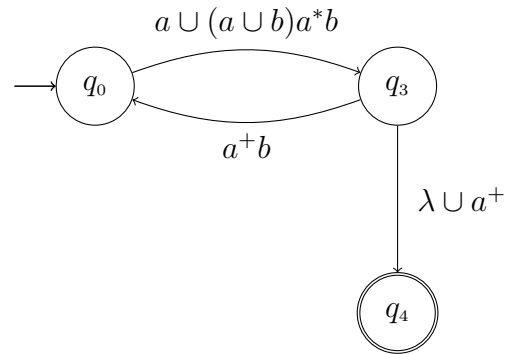


A continuación añadimos el nuevo estado q_4 (que será el único estado de aceptación) y transiciones λ desde q_2 y q_3 hasta q_4 (grafo ③). Luego eliminamos el estado q_2 (grafo ④):

③

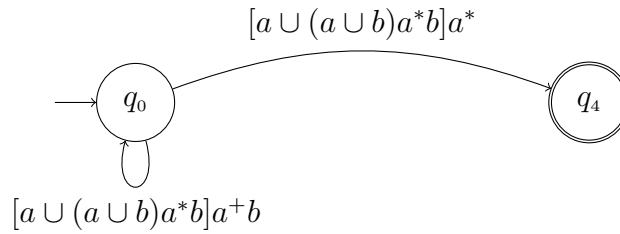


④



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:

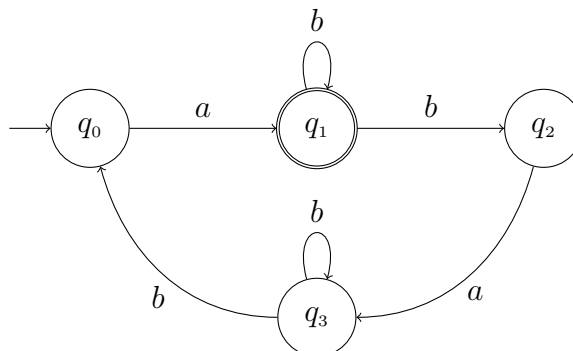
$$L(M) = \left([a \cup (a \cup b)a^*b]a^+b \right)^* [a \cup (a \cup b)a^*b]a^*.$$

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 $L(M)$.

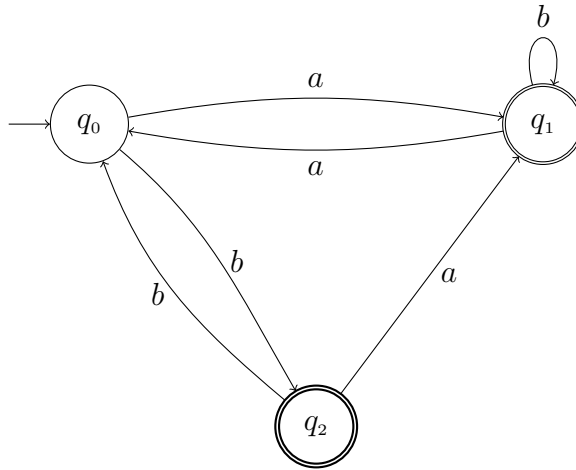
Ejercicios de la sección 2.13

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

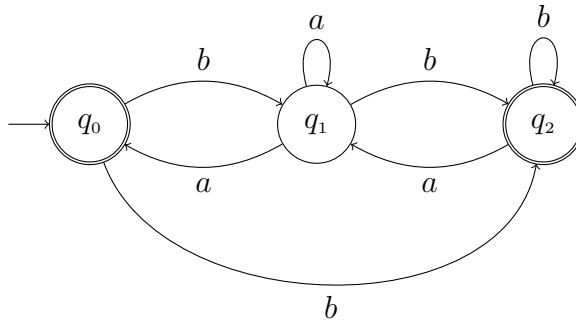
(i)



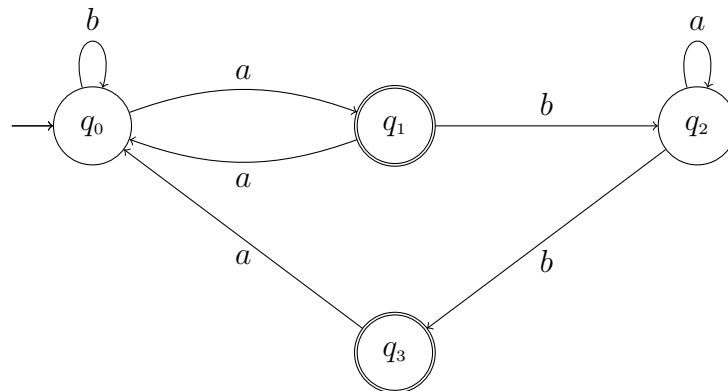
(ii)



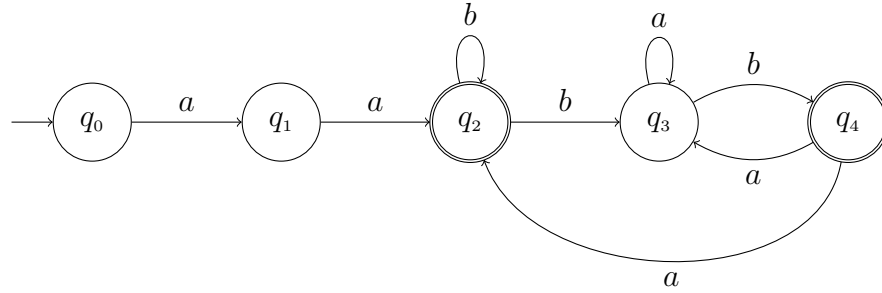
(iii)



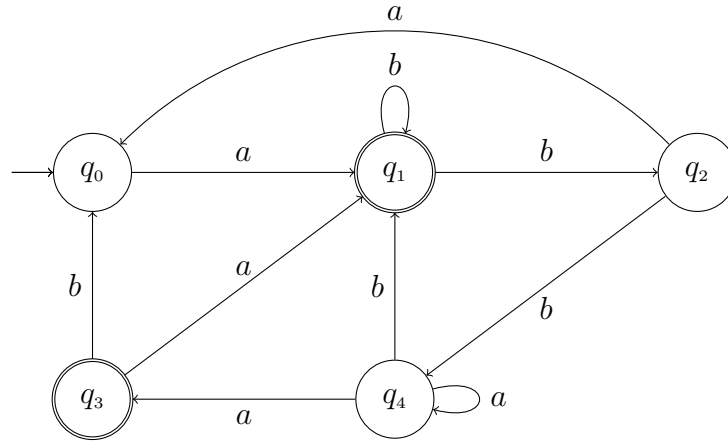
(iv)



(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) $\overline{L} = \Sigma^* - L$. |
| (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 \overline{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) $\overline{L} = \Sigma^* - L$. |
| (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. Minimización de autómatas, parte I

En la presente sección presentaremos un procedimiento general para encontrar un autómata con el menor número de estados posible, equivalente a un AFD dado. Se trata de un procedimiento algorítmico en el que 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.

2.15.1 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:

$$\begin{aligned} Q' &= \{[p] : p \in Q\}, \\ q'_0 &= [q_0], \\ F' &= \{[p] : p \in F\}, \\ \delta'([p], a) &= [\delta(p, a)], \text{ para todo } a \in \Sigma. \end{aligned}$$

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.

2.15.2 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) $\widehat{\delta'}([p], w) = [\widehat{\delta}(p, u)]$ 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

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

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 2.15.2 se puede deducir que M y M' aceptan el mismo lenguaje, tal como se demuestra en el siguiente teorema.

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

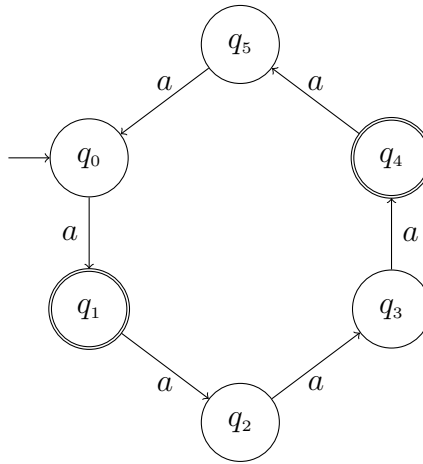
Demostración.

$$\begin{aligned} u \in L(M') &\iff \widehat{\delta'}([q_0], u) \in F' \\ &\iff [\widehat{\delta}(q_0, u)] \in F' \quad (\text{por la Proposición 2.15.2 (iv)}) \\ &\iff \widehat{\delta}(q_0, u) \in F \quad (\text{por la Proposición 2.15.2 (iii)}) \\ &\iff u \in L(M). \quad \square \end{aligned}$$

Dado un AFD M , el autómata cociente M' resulta ser un autómata con el mínimo número de estados posible para aceptar $L(M)$. Esto se demuestra detalladamente en la sección 2.16

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*. Un estado q es accesible si existe una cadena de entrada u 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 deben eliminar previamente. 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”.

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

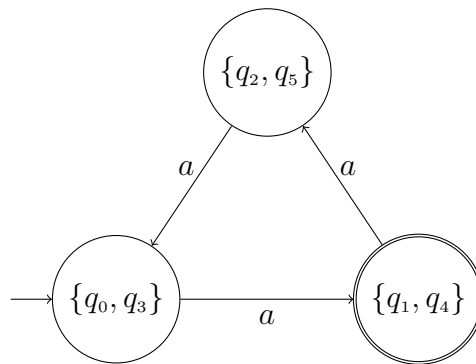
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos el símbolo a . La información necesaria aparece en la siguiente tabla; la columna izquierda corresponde a las casillas no marcadas $\{p, q\}$ y la derecha a las casillas $\{\delta(p, a), \delta(q, a)\}$ obtenidas al procesar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

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

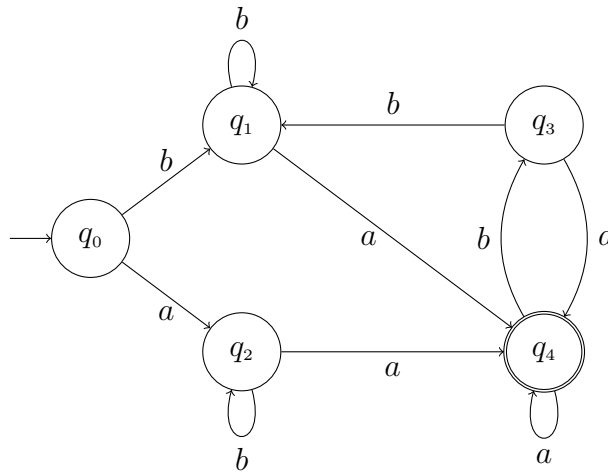
Las marcas \times en la columna derecha representan casillas previamente marcadas; según el algoritmo, las correspondientes casillas en la columna izquierda se marcan con \times . Entonces, al terminar la segunda iteración la tabla triangular adquiere el siguiente aspecto:

q_0					
×	q_1				
×	×	q_2			
	×	×	q_3		
×		×	×	q_4	
×	×		×	×	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	
×	×	×	×	q_4

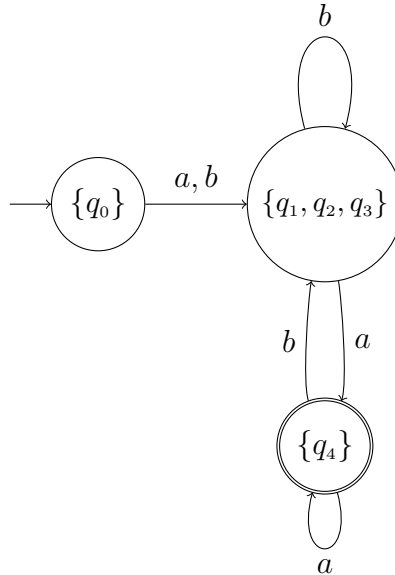
Luego hacemos $i := 2$, examinamos las casillas aún no marcadas y procesamos con las entradas a y b . Tal información aparece en la siguiente tabla, la cual se utiliza a partir de la segunda iteración.

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

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:

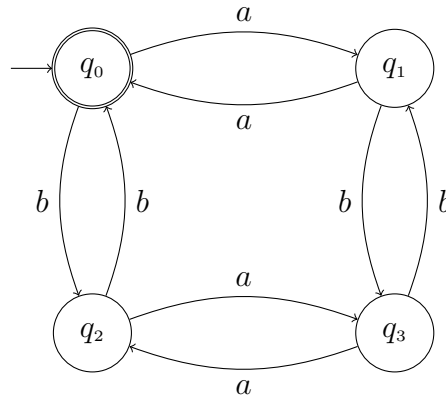
	q_0				
\times		q_1			
\times			q_2		
\times				q_3	
\times	\times	\times	\times	\times	q_4

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 s y un número par de b s 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		
\times	q_1		
\times		q_2	
\times			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

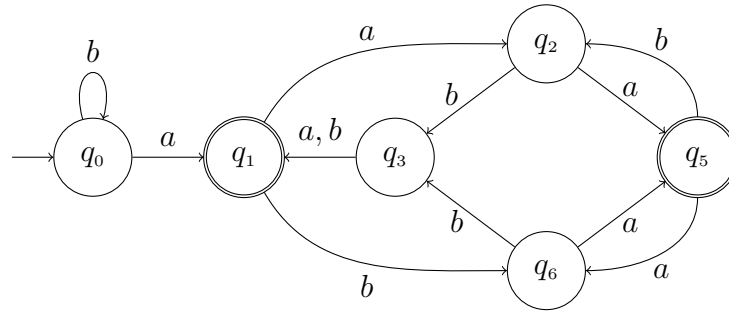
	q_0		
\times	q_1		
\times	\times	q_2	
\times	\times	\times	q_3

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.

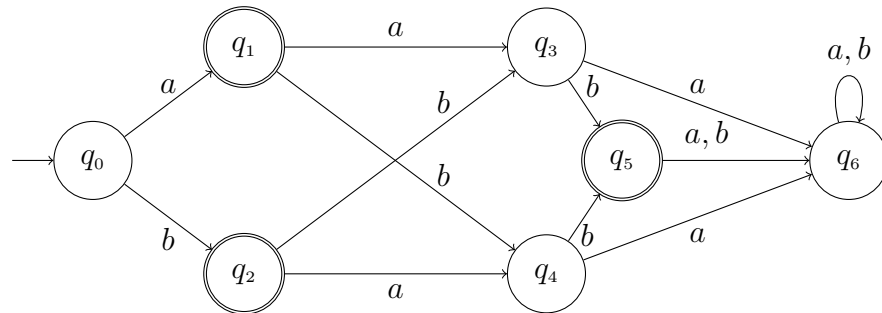
Ejercicios de la sección 2.15

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

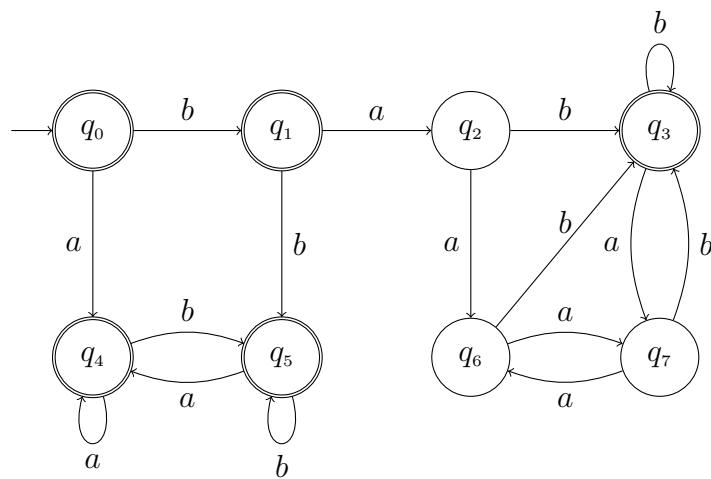
(i) Alfabeto $\Sigma = \{a, b\}$.



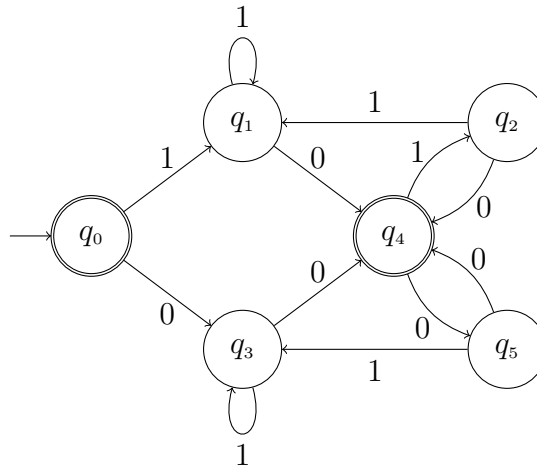
(ii) Alfabeto $\Sigma = \{a, b\}$.



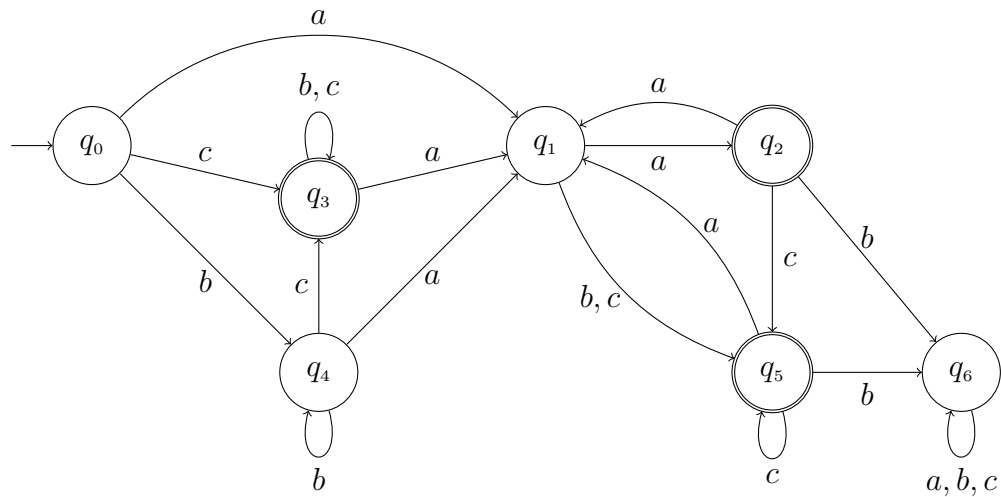
(iii) Alfabeto $\Sigma = \{a, b\}$.



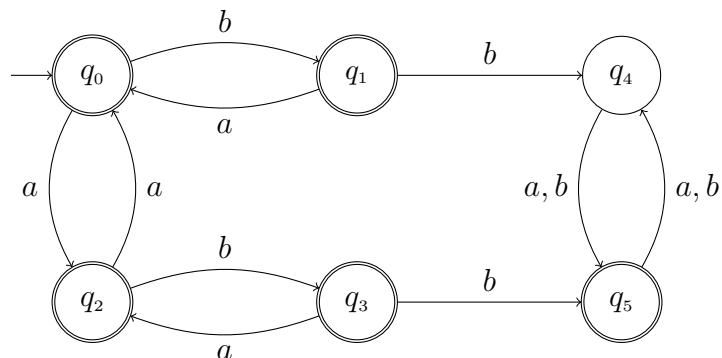
(iv) Alfabeto $\Sigma = \{0, 1\}$.



(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).

2.16. Minimización de autómatas. Parte II

En la presente sección se demuestra que, dado un AFD M , el autómata cociente M' , definido en la sección 2.15, resulta ser un autómata con el mínimo número de estados equivalente a M . La demostración requiere introducir la noción de distinguibilidad entre cadenas, la cual resulta también crucial en el Capítulo 3.

2.16.1 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

$$(2.16.1) \quad (\forall x \in \Sigma^*) [ux \in L \iff vx \in L].$$

Utilizaremos la notación uI_Lv para representar el hecho de que las cadenas u y v son L -indistinguibles.

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 (2.16.1); 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.

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

2.16.2 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.

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

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

Por lo tanto, uI_Lv . □

2.16.3 Corolario. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$. Si las cadenas u y v en Σ^* son L -distinguibiles, entonces $\widehat{\delta}(q_0, u) \neq \widehat{\delta}(q_0, v)$.

Demostración. El enunciado es la implicación contra-recíproca de la Proposición 2.16.2. \square

2.16.4 Proposición. Sea Σ un alfabeto dado y $L \subseteq \Sigma^*$ regular. Si hay n cadenas u_1, u_2, \dots, u_n en Σ^* que sean L -distinguibiles dos a dos (es decir, u_i y u_j son L -distinguibiles para todo $i \neq j$, $i, j = 1, \dots, n$), entonces cualquier AFD M que acepte a L debe tener por lo menos n estados.

Demostración. Sea $M = (\Sigma, Q, q_0, F, \delta)$ un AFD que acepte a L . Por el Corolario 2.16.3, $\widehat{\delta}(q_0, u_i) \neq \widehat{\delta}(q_0, u_j)$ para todo $i \neq j$. Entonces los n estados

$$\widehat{\delta}(q_0, u_1), \widehat{\delta}(q_0, u_2), \dots, \widehat{\delta}(q_0, u_n)$$

son diferentes entre sí (diferentes dos a dos) y, por lo tanto, M tiene por lo menos n estados. \square

El siguiente enunciado es un recíproco parcial de la Proposición 2.16.2.

2.16.5 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 -indistinguibiles, 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

$$(2.16.2) \quad \widehat{\delta}(p, w) = \widehat{\delta}(\widehat{\delta}(q_0, u), w) = \widehat{\delta}(q_0, uw),$$

$$(2.16.3) \quad \widehat{\delta}(q, w) = \widehat{\delta}(\widehat{\delta}(q_0, v), w) = \widehat{\delta}(q_0, vw).$$

Entonces

$$\begin{aligned} \widehat{\delta}(p, w) \in F &\iff \widehat{\delta}(q_0, uw) \in F \quad (\text{por (2.16.2)}) \\ &\iff uw \in L(M) = L \\ &\iff vw \in L \quad (\text{por ser } u \text{ y } v \text{ indistinguibiles}) \\ &\iff \widehat{\delta}(q_0, vw) \in F \\ &\iff \widehat{\delta}(q, w) \in F \quad (\text{por (2.16.3)}). \end{aligned} \quad \square$$

2.16.6 Corolario. Sean $M = (\Sigma, Q, q_0, F, \delta)$ un AFD tal que $L(M) = L$, y dos cadenas $u, v \in \Sigma^*$. Si $\widehat{\delta}(q_0, u) \not\approx \widehat{\delta}(q_0, v)$, entonces u y v son L -distinguibiles.

Demostración. Contrarecíproca de la Proposición 2.16.5. \square

2.16.7 Teorema. Sea M un AFD cuyos estados son todos accesibles. El autómata cociente M' (definido en la sección 2.15) es un AFD equivalente a M , con el mínimo número de estados posible para aceptar el lenguaje $L(M)$.

Demostración. Sea M el AFD $M = (\Sigma, Q, q_0, F, \delta)$. Los estados del autómata cociente $M' = (\Sigma, Q', q'_0, F', \delta')$ son todos accesibles. En efecto, dado $q \in Q$, existe u tal que $\widehat{\delta}(q_0, u) = q$ ya que q es accesible en M . Se sigue que $\widehat{\delta'}([q_0], u) = [\widehat{\delta}(q_0, u)] = [q]$ y, por lo tanto $[q]$ es accesible.

Supóngase ahora que M' tiene n estados diferentes, $[q_0], [q_1], \dots, [q_{n-1}]$, formados por clases de equivalencia del autómata M . Como los estados q_0, q_1, \dots, q_{n-1} de M son accesibles, existen cadenas x_0, x_1, \dots, x_{n-1} tales que $\widehat{\delta}(q_0, x_0) = q_0, \widehat{\delta}(q_0, x_1) = q_1, \dots, \widehat{\delta}(q_0, x_{n-1}) = q_{n-1}$. Puesto que $q_i \not\approx q_j$ si $i \neq j$, se sigue del Corolario 2.16.6 que x_i es distinguible de x_j si $i \neq j$. Por lo tanto, x_0, x_1, \dots, x_{n-1} son n cadenas distinguibles dos a dos. Por la Proposición 2.16.4, cualquier AFD que acepte a L debe tener por lo menos n estados. En consecuencia, n es el mínimo número de estados posible. \square

Capítulo 3

Lenguajes no regulares

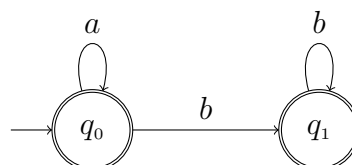
En el capítulo anterior se estudiaron los lenguajes regulares y sus propiedades fundamentales. En particular, el Teorema de Kleene (secciones 2.12 y 2.13) establece que los lenguajes regulares son exactamente los lenguajes aceptados por los autómatas finitos. En este capítulo se demostrará que existen lenguajes no-regulares y, como consecuencia, se concluirá que hay problemas computacionales que no se pueden resolver utilizando los modelos de autómatas finitos (AFD, AFN y AFN- λ) hasta ahora considerados.

3.1. Criterio de no regularidad

El hecho de que todo autómata finito tiene solamente un número finito de estados pero el conjunto Σ^* tiene infinitas cadenas, permite concluir que hay ciertos lenguajes que no pueden ser aceptados por ningún autómata. El razonamiento se basa en una de las versiones del llamado Principio del Palomar: si infinitos objetos se distribuyen en un número finito de compartimientos, entonces hay un compartimiento que contiene dos objetos (por lo menos). A continuación presentamos un par de ejemplos.

Ejemplo Demostrar que el lenguaje $L = \{a^n b^n : n \geq 0\} = \{\lambda, a, a^2, a^3, a^4, \dots\}$, sobre $\Sigma = \{a, b\}$, no es regular.

Solución. En primer lugar, hay que observar que el lenguaje L es diferente de a^*b^* ya que este último lenguaje está formado por las concatenaciones de cualquier número de a s con cualquier número de b s, o sea, $a^*b^* = \{a^m b^n : m, n \geq 0\}$, mientras que cada cadena de L tiene igual número de a s que de b s. a^*b^* es regular y se puede aceptar por medio del siguiente AFD:

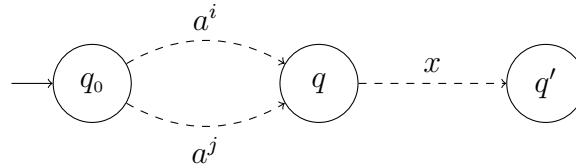


No obstante, para aceptar a L un autómata debe almacenar (de alguna forma) la información sobre el número de a 's y compararla luego con el número de b 's. El único tipo de memoria disponible son los estados y resulta intuitivamente claro que con un número finito de estados es imposible para un autómata almacenar la información necesaria para procesar y aceptar todas y cada una de las cadenas de la forma $a^n b^n$ con $n \geq 0$. Pero se requiere un argumento matemáticamente riguroso para demostrarlo concluyentemente.

Para concluir que L no es regular razonamos por contradicción (o reducción al absurdo): suponemos que L es regular y llegamos a una contradicción. Si L es regular, existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$. Consideramos luego las potencias positivas de a , a saber, a, a^2, a^3, a^4, \dots . Después de leer completamente cada una de estas cadenas, M termina en un determinado estado, pero como hay infinitas potencias de a y solamente un número finito de estados en M , deben existir dos cadenas diferentes a^i y a^j , con $i \neq j$, cuyo procesamiento completo termina en el mismo estado. Esta última afirmación es consecuencia del mencionado Principio del Palomar: si infinitos objetos se distribuyen en un número finito de compartimientos, entonces hay un compartimiento que contiene dos objetos (por lo menos). Utilizando la función $\hat{\delta}$ (Definición 2.7.2), podemos expresar este hecho como $\hat{\delta}(q_0, a^i) = \hat{\delta}(q_0, a^j)$. Como M es un autómata determinista, se concluye que, para toda cadena $x \in \Sigma^*$, el procesamiento de las cadenas $a^i x$ y $a^j x$ termina en el mismo estado; en efecto,

$$\hat{\delta}(q_0, a^i x) = \hat{\delta}(\hat{\delta}(q_0, a^i), x) = \hat{\delta}(\hat{\delta}(q_0, a^j), x) = \hat{\delta}(q_0, a^j x).$$

En el grafo de M se puede visualizar la situación de la siguiente manera:



Esto implica que para toda $x \in \Sigma^*$, las cadenas $a^i x$ y $a^j x$ son ambas aceptadas o ambas rechazadas. La contradicción surge porque existe una cadena $x \in \Sigma^*$ tal que $a^i x$ es aceptada pero $a^j x$ es rechazada. Por simple inspección observamos que tomando $x = b^i$ se tendrá $a^i x = a^i b^i \in L$ pero $a^j x = a^j b^i \notin L$ (porque $i \neq j$). De modo que $a^i b^i$ es aceptada mientras que $a^j b^i$ es rechazada, lo cual es una contradicción.

Ejemplo Demostrar que el lenguaje de los palíndromes sobre $\Sigma = \{0, 1\}$ no es un lenguaje regular. Recuérdese que un palíndromo es una cadena u que coincide con su reflexión, o sea, tal que $u = u^R$.

Solución. Como en el ejemplo anterior, razonamos por contradicción. Suponemos que L es regular; entonces existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$. A continuación consideramos las potencias positivas de 1, a saber, $1, 1^2, 1^3, 1^4, \dots$. Después de leer completamente cada una de estas cadenas, M termina en un determinado estado; pero como hay infinitas potencias de 1 y solamente un número finito de estados en M , deben existir

dos cadenas diferentes 1^i y 1^j , con $i \neq j$, cuyo procesamiento termina en el mismo estado. Es decir, $\widehat{\delta}(q_0, 1^i) = \widehat{\delta}(q_0, 1^j)$. Como M es un autómata determinista, se concluye que, para toda $x \in \Sigma^*$, el procesamiento de las cadenas $1^i x$ y $1^j x$ termina en el mismo estado. En otras palabras, $\widehat{\delta}(q_0, 1^i x) = \widehat{\delta}(q_0, 1^j x)$ para toda cadena $x \in \Sigma^*$. Esto implica que para toda $x \in \Sigma^*$, las cadenas $1^i x$ y $1^j x$ son ambas aceptadas o ambas rechazadas. Se llega a una contradicción si podemos encontrar una cadena $x \in \Sigma^*$ tal que $1^i x$ sea aceptada pero $1^j x$ sea rechazada (o viceversa). Procediendo por inspección (ensayo y error) observamos que tomando $x = 01^i$ se tendrá que $1^i x = 1^i 01^i$ es un palíndromo pero $1^j x = 1^j 01^i$ no lo es (porque $i \neq j$). De modo que $1^i 01^i$ es aceptada mientras que $1^j 01^i$ es rechazada, lo cual es una contradicción.

Nótese que podríamos haber construido un razonamiento completamente similar considerando las infinitas potencias positivas de 0 ($0, 0^2, 0^3, 0^4, \dots$) en vez de las de 1.

Utilizando la noción de distinguibilidad entre cadenas introducida en la sección 2.16, el argumento usado en los dos ejemplos anteriores se puede convertir en un criterio práctico para demostrar que ciertos lenguajes no son regulares. Recordamos la definición de cadenas distinguibles (Definición 2.16.1). Dado un lenguaje L sobre un alfabeto Σ , dos cadenas u y v en Σ^* 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.

3.1.1. Criterio de no regularidad. Sea Σ un alfabeto dado y L un lenguaje sobre Σ . Si en Σ^* hay infinitas cadenas L -distinguibles dos a dos (es decir, dos cadenas diferentes son L -distinguibles entre sí), entonces L no es regular.

Demostración. Razonamos por contradicción (o reducción al absurdo): suponemos que L es regular y llegamos a una contradicción. Si L es regular, existe un AFD $M = (\Sigma, Q, q_0, F, \delta)$ tal que $L(M) = L$. Por hipótesis, existen infinitas cadenas L -distinguibles dos a dos, u_1, u_2, u_3, \dots . Después de leer completamente cada una de estas cadenas, M termina en un determinado estado, pero como solamente hay un número finito de estados en M , por el Principio del Palomar se deduce que deben existir dos cadenas diferentes u_i y u_j , con $i \neq j$, cuyo procesamiento completo termina en el mismo estado, es decir, $\widehat{\delta}(q_0, u_i) = \widehat{\delta}(q_0, u_j)$. Como M es un autómata determinista, se concluye que, para toda cadena $x \in \Sigma^*$, el procesamiento de las cadenas $u_i x$ y $u_j x$ termina en el mismo estado; en efecto,

$$\widehat{\delta}(q_0, u_i x) = \widehat{\delta}(\widehat{\delta}(q_0, u_i), x) = \widehat{\delta}(\widehat{\delta}(q_0, u_j), x) = \widehat{\delta}(q_0, u_j x).$$

Esto implica que para toda $x \in \Sigma^*$, las cadenas $u_i x$ y $u_j x$ son ambas aceptadas o ambas rechazadas, es decir, o bien $(u_i x \in L \text{ y } u_j x \in L)$, o bien $(u_i x \notin L \text{ y } u_j x \notin L)$. Esto contradice el hecho de que u_i y u_j son L -distinguibles. \square

Según este criterio, para concluir que L no es regular basta exhibir infinitas cadenas en Σ^* que sean L -distinguibles dos a dos. No es necesario utilizar explícitamente autómatas

al invocar el criterio de no regularidad. En la mayoría de los casos será suficiente mostrar que a, a^2, a^3, \dots son infinitas cadenas L -distinguibles dos a dos (en caso de que $a \in \Sigma$). En algunos problemas resulta conveniente mostrar que las potencias pares, a^2, a^4, a^6, \dots (o bien, las potencias impares) son infinitas cadenas L -distinguibles dos a dos. Los siguientes ejemplos ilustran el uso de esta técnica.

Ejemplo Utilizar el criterio de no regularidad para demostrar que el lenguaje $L = \{a^n b^n : n \geq 0\}$, sobre $\Sigma = \{a, b\}$, no es regular.

Solución. Vamos a comprobar que a, a^2, a^3, \dots son infinitas cadenas L -distinguibles dos a dos. Sean $i, j \geq 1$, con $i \neq j$. Queremos mostrar que a^i y a^j son L -distinguibles, para lo cual hay que encontrar una cadena x tal que $a^i x \in L$ pero $a^j x \notin L$. La escogencia de x es obvia: $x = b^i$. Se tiene que $a^i x = a^i b^i \in L$ pero $a^j x = a^j b^i \notin L$. Esto muestra que a^i y a^j son L -distinguibles si $i \neq j$ y, por el criterio 3.1.1, L no puede ser regular.

Ejemplo Utilizar el criterio de no regularidad para demostrar que el lenguaje de los palíndromos sobre $\Sigma = \{0, 1\}$ no es un lenguaje regular.

Solución. Vamos a comprobar que $1, 1^2, 1^3, \dots$ son infinitas cadenas L -distinguibles dos a dos. Sean $i, j \geq 1$, con $i \neq j$. Queremos mostrar que 1^i y 1^j son L -distinguibles, para lo cual hay que encontrar una cadena x tal que $1^i x \in L$ pero $1^j x \notin L$. Escogemos $x = 01^i$. Se tendrá entonces que $1^i x = 1^i 01^i \in L$ pero $1^j x = 1^j 01^i \notin L$. Esto muestra que 1^i y 1^j son L -distinguibles si $i \neq j$ y, por criterio 3.1.1, L no puede ser regular.

Ejemplo Utilizar el criterio de no regularidad para demostrar que el lenguaje $L = \{a^{n^2} : n \geq 1\}$ no es regular. L está formado por cadenas de a 's cuya longitud es un cuadrado perfecto, sobre el alfabeto $\Sigma = \{a\}$.

Solución. Vamos a comprobar que las cadenas de L , o sea $a, a^4, a^9, a^{16}, a^{25}, \dots$ son L -distinguibles dos a dos. Sean $i, j \geq 1$, con $i \neq j$; sin pérdida de generalidad podemos suponer que $i < j$. Queremos mostrar que a^{i^2} y a^{j^2} son L -distinguibles. Si escogemos $x = a^{2i+1}$, se tendrá que $a^{i^2} x = a^{i^2} a^{2i+1} = a^{i^2+2i+1} = a^{(i+1)^2} \in L$. De otro lado, $a^{j^2} x = a^{j^2} a^{2i+1} = a^{j^2+2i+1} \notin L$ ya que

$$j^2 < j^2 + 2i + 1 < j^2 + 2j + 1 = (j+1)^2.$$

Esto quiere decir que el número de a 's que hay en a^{j^2+2i+1} está estrictamente comprendido entre dos cuadrados perfectos consecutivos, a saber, j^2 y $(j+1)^2$. Por consiguiente, $a^{j^2+2i+1} \notin L$.

Ejercicios de la sección 3.1

- ① Ya sea utilizando un argumento por contradicción o el criterio de no regularidad, demostrar que los siguientes lenguajes no son regulares:

- (i) $L = \{a^n b^{2n} : n \geq 0\}$, sobre $\Sigma = \{a, b\}$.
- (ii) $L = \{a^{2n} b^n : n \geq 0\}$, sobre $\Sigma = \{a, b\}$.

- (iii) $L = \{a^n b^m : n, m \geq 0, n \neq m\}$, sobre $\Sigma = \{a, b\}$.
- (iv) $L = \{a^m b^n : m \geq n \geq 0\}$, sobre $\Sigma = \{a, b\}$.
- (v) $L = \{0^n 10^n : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (vi) $L = \{0^m 1^n 0^m : m, n \geq 0\}$, sobre $\Sigma = \{0, 1\}$.
- (vii) $L = \{1^n 01^n 0 : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (viii) $L = \{01^n 01^n : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (ix) $L = \{uu : u \in \Sigma^*\}$, sobre $\Sigma = \{a, b\}$.
- (x) $L = \{uu^R : u \in \Sigma^*\}$, sobre $\Sigma = \{a, b\}$.

- ② $\Sigma = \{0, 1\}$. Ya sea utilizando un argumento por contradicción o el criterio de no regularidad, demostrar que el lenguaje $L = \{u \in \Sigma^* : \#_0(u) = \#_1(u)\}$ no es regular.
- ③ Utilizar el criterio de no regularidad para demostrar que el lenguaje

$$L = \{a^n : n \text{ es un número primo}\},$$

sobre $\Sigma = \{a\}$, no es regular. Ayuda: usar el Teorema de Legendre según el cual, si p y q son primos, entonces la progresión aritmética $\{p + kq : k \geq 0\}$ contiene infinitos primos.

- ④ Demostrar que a^*b^* es la unión de dos lenguajes disyuntos no-regulares.
- ⑤ Según la definición de los conjuntos regulares, la unión de un número finito de lenguajes regulares también es regular. Comprobar mediante un contraejemplo que la unión de una familia infinita de conjuntos regulares no necesariamente es regular.
- ⑥ Sea L un lenguaje no-regular y N un subconjunto finito de L . Demostrar que $L - N$ tampoco es regular.
- ⑦ Demostrar o refutar las siguientes afirmaciones:
 - (i) Un lenguaje no-regular debe ser infinito.
 - (ii) Si el lenguaje $L_1 \cup L_2$ es regular, también lo son L_1 y L_2 .
 - (iii) Si los lenguajes L_1 y L_2 no son regulares, el lenguaje $L_1 \cap L_2$ tampoco puede ser regular.
 - (iv) Si el lenguaje L^* es regular, también lo es L .
 - (v) Si L es regular y N es un subconjunto finito de L , entonces $L - N$ es regular.
 - (vi) Un lenguaje regular no-vacío L es infinito si y sólo si en cualquier expresión regular de L aparece por lo menos una $*$ (estrella de Kleene).

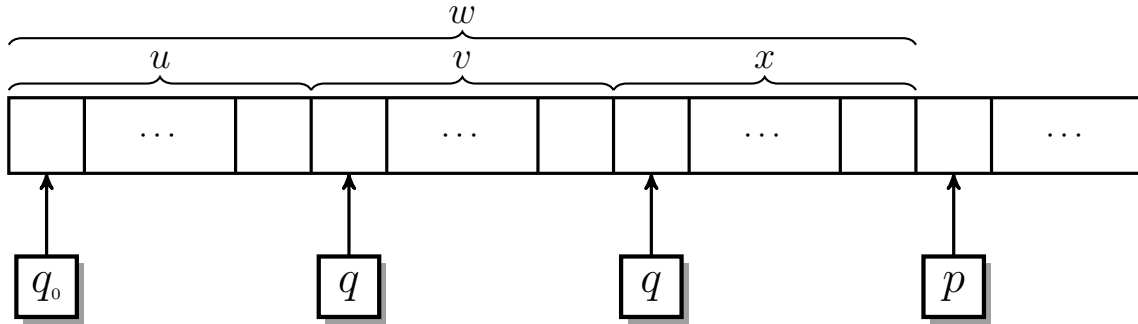
3.2. Lema de bombeo

El llamado “lema de bombeo” (*pumping lemma*, en inglés) es una propiedad de los lenguajes regulares que también se puede usar para demostrar, razonando por contradicción, que ciertos lenguajes no son regulares. En comparación con el criterio de no regularidad, el Lema de bombeo no es una herramienta práctica en casos concretos porque el razonamiento es mucho más complejo. No obstante, se trata de un enunciado clásico en la teoría de lenguajes formales que se puede generalizar para otras colecciones de lenguajes, con importantes consecuencias, tal como veremos en capítulos posteriores.

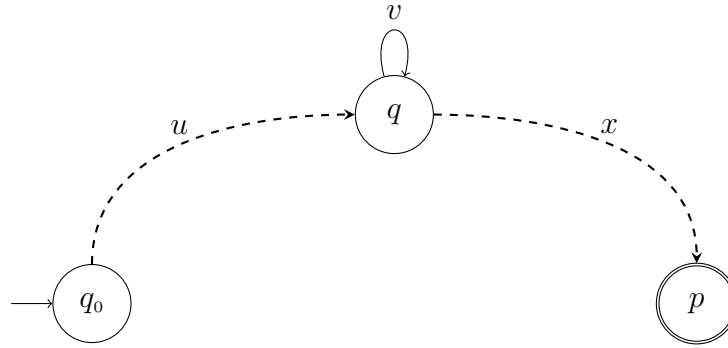
3.2.1. Lema de bombeo. Para todo lenguaje regular L (sobre un alfabeto dado Σ) existe una constante $n \in \mathbb{N}$, llamada constante de bombeo para L , tal que toda cadena $w \in L$, con $|w| \geq n$, satisface la siguiente propiedad:

$$(B) \quad \begin{cases} w \text{ se puede descomponer como } w = uvx, \text{ donde } |uv| \leq n, \\ v \neq \lambda, \text{ y para todo } i \geq 0 \text{ se tiene } uv^i x \in L. \end{cases}$$

Demostración. Por el Teorema de Kleene y por los teoremas de equivalencia de los modelos AFD, AFN y AFN- λ , existe un AFD M tal que $L(M) = L$. Sea n el número de estados de M . Si $w \in L$ y $|w| \geq n$, entonces durante el procesamiento completo de w , hay por lo menos un estado que se repite. Sea q el primer estado que se repite. Tal como se muestra en la siguiente gráfica, w se puede descomponer como concatenación de tres cadenas, esto es, $w = uvx$, donde $|uv| \leq n$, $v \neq \lambda$.



Puesto que $w \in L(M)$, p es un estado de aceptación. Nótese que tanto u como x pueden ser la cadena vacía λ , pero $v \neq \lambda$. Además, la cadena v se puede “bombear”, en el sentido de que $uv^i x$ es aceptada por M para todo $i \geq 0$. La propiedad de bombeo de v se puede apreciar en el grafo del autómata M , como se exhibe la siguiente página.



Uso del lema de bombeo. El lema de bombeo se puede usar para concluir que un cierto lenguaje dado L no es regular, recurriendo a un razonamiento por contradicción (o reducción al absurdo). El razonamiento utilizado tiene la siguiente forma:

1. Si L fuera regular, existiría una constante de bombeo n para L .
2. Se escoge una cadena “adecuada” $w \in L$ y se aplica la propiedad (B) del lema de bombeo, descomponiendo w como

$$w = uvx, \quad v \neq \lambda, \quad |uv| \leq n.$$

3. Se llega a la siguiente contradicción:

- (I) Por el lema de bombeo, $uv^i x \in L$, para todo $i \geq 0$.
- (II) $uv^i x$ no puede estar en L , para algún $i \in I$. Por lo general, basta escoger valores pequeños de i como $i = 0$ ó $i = 2$.

Ejemplo Usar el lema de bombeo para demostrar que el lenguaje $L = \{a^i b^i : i \geq 0\}$ no es regular.

Solución. Si L fuera regular, existiría una constante de bombeo n para L . Sea $w = a^n b^n \in L$. Entonces w se puede descomponer como $w = uvx$, con $|v| \geq 1$ y $|uv| \leq n$. Por lo tanto, u y v constan únicamente de a 's:

$$\begin{aligned} u &= a^r, & \text{para algún } r \geq 0, \\ v &= a^s, & \text{para algún } s \geq 1. \end{aligned}$$

Entonces,

$$x = a^{n-(r+s)} b^n = a^{n-r-s} b^n.$$

Por el lema de bombeo, $uv^i x \in L$ para todo $i \geq 0$. En particular, si $i = 0$, $ux \in L$. Pero $ux = a^r a^{n-r-s} b^n = a^{n-s} b^n$. Como $n-s \neq n$, la cadena $ux \notin L$ lo cual es una contradicción. Se concluye entonces que L no puede ser regular.

Tomando $i = 2$ también se llega a una contradicción: por un lado, $uv^2 x \in L$, pero

$$uv^2 x = a^r a^s a^s a^{n-r-s} b^n = a^{r+2s+n-r-s} b^n = a^{n+s} b^n.$$

Como $s \geq 1$, $a^{n+s}b^n$ no está en L .

El argumento anterior también sirve para demostrar que el lenguaje $L = \{a^i b^i : i \geq 1\}$ no es regular.

Ejemplo Demostrar que el lenguaje de los palíndromos sobre $\{a, b\}$ no es un lenguaje regular.

Solución. Si L fuera regular, existiría una constante de bombeo n para L . Sea $w = a^n b a^n \in L$. Entonces w se puede descomponer como $w = uvx$, con $|v| \geq 1$, $|uv| \leq n$, y para todo $i \geq 0$, $uv^i x \in L$. Por lo tanto, u y v constan únicamente de a 's:

$$\begin{aligned} u &= a^r, & \text{para algún } r \geq 0, \\ v &= a^s, & \text{para algún } s \geq 1. \end{aligned}$$

Entonces,

$$x = a^{n-(r+s)} b a^n = a^{n-r-s} b a^n.$$

Tomando $i = 0$, se concluye que $ux \in L$, pero

$$ux = a^r a^{n-r-s} b a^n = a^{n-s} b a^n.$$

Como $s \geq 1$, $a^{n-s} b a^n$ no es un palíndromo. Esta contradicción muestra que L no puede ser regular.

Ejemplo Demostrar que el lenguaje $L = \{a^{i^2} : i \geq 0\}$ no es regular. L está formado por cadenas de a 's cuya longitud es un cuadrado perfecto.

Solución. Si L fuera regular, existiría una constante de bombeo n para L . Sea $w = a^{n^2} \in L$. Entonces w se puede descomponer como $w = uvx$, con $|v| \geq 1$, $|uv| \leq n$. Por el lema de bombeo, $uv^2x \in L$, pero por otro lado,

$$n^2 < n^2 + |v| = |uvx| + |v| = |uv^2x| \leq n^2 + |uv| \leq n^2 + n < (n+1)^2.$$

Esto quiere decir que el número de símbolos de la cadena uv^2x no es un cuadrado perfecto y, por consiguiente, $uv^2x \notin L$. En conclusión, L no es regular.

Ejercicios de la sección 3.2

① Utilizar el lema de bombeo demostrar que los siguientes lenguajes no son regulares:

- (i) $L = \{a^n b^{2n} : n \geq 0\}$, sobre $\Sigma = \{a, b\}$.
- (ii) $L = \{a^{2n} b^n : n \geq 0\}$, sobre $\Sigma = \{a, b\}$.
- (iii) $L = \{a^n b^m : n, m \geq 0, n \neq m\}$, sobre $\Sigma = \{a, b\}$.
- (iv) $L = \{a^m b^n : m \geq n \geq 0\}$, sobre $\Sigma = \{a, b\}$.
- (v) $L = \{0^n 10^n : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (vi) $L = \{0^m 1^n 0^m : m, n \geq 0\}$, sobre $\Sigma = \{0, 1\}$.

- (vii) $L = \{1^n 0 1^n : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (viii) $L = \{0 1^n 0 1^n : n \geq 1\}$, sobre $\Sigma = \{0, 1\}$.
- (ix) $L = \{uu : u \in \Sigma^*\}$, sobre $\Sigma = \{a, b\}$.
- (x) $L = \{uu^R : u \in \Sigma^*\}$, sobre $\Sigma = \{a, b\}$.

- ② Encontrar la falacia en el siguiente argumento: “Según la propiedad (B) del enunciado del lema de bombeo, si L es un lenguaje regular se tiene que $uv^i x \in L$ para todo $i \geq 0$. Por consiguiente, L posee infinitas cadenas y, en conclusión, todo lenguaje regular es infinito.”

3.3. Problemas insolubles para autómatas finitos

La existencia de lenguajes no-regulares implica que hay ciertos problemas que no pueden ser resueltos por medio de autómatas finitos. Se puede pensar que un autómata es un mecanismo que produce dos salidas para cada entrada $u \in \Sigma^*$: SÍ (cuando u es aceptada) y NO (cuando u es rechazada). Los problemas que puede resolver un autómata son entonces *problemas de decisión*, esto es, problemas para los cuales hay dos únicas salidas, SÍ o NO. Esta idea la precisamos a continuación.

Dado un alfabeto Σ , consideremos una propiedad \mathcal{P} referente a las cadenas de Σ^* . El problema de decisión para \mathcal{P} es el siguiente:

$$(3.3.1) \quad \text{Dada } u \in \Sigma^*, \text{ ¿satisface } u \text{ la propiedad } \mathcal{P}?$$

Denotamos con $L_{\mathcal{P}}$ el lenguaje de todas las cadenas que satisfacen la propiedad \mathcal{P} , es decir,

$$L_{\mathcal{P}} = \{u \in \Sigma^* : u \text{ satisface } \mathcal{P}\} = \{u \in \Sigma^* : \mathcal{P}(u)\}.$$

Decimos que el problema (3.3.1) puede ser resuelto usando autómatas finitos si existe un AFD M tal que $L(M) = L_{\mathcal{P}}$. En tal caso, para cada entrada $u \in \Sigma^*$ se tendrá

$$\begin{cases} \text{Si } u \text{ es aceptada por } M \implies u \in L(M) \implies u \text{ satisface } \mathcal{P}. \\ \text{Si } u \text{ es rechazada por } M \implies u \notin L(M) \implies u \text{ no satisface } \mathcal{P}. \end{cases}$$

Por el Teorema de Kleene se concluye que el problema de decisión (3.3.1) se puede resolver usando autómatas finitos si y sólo si $L_{\mathcal{P}}$ es regular.

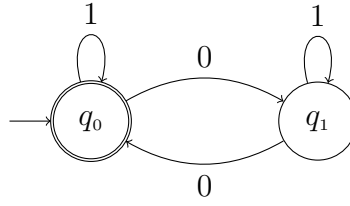
Ejemplo Sea $\Sigma = \{0, 1\}$. El problema de decisión

Dada $u \in \Sigma^*$, ¿tiene u un número par de ceros?

se puede resolver usando autómatas finitos. La propiedad \mathcal{P} en este caso es “tener un número par de ceros” y $L_{\mathcal{P}}$ es el lenguaje

$$L_{\mathcal{P}} = \{u \in \Sigma^* : u \text{ tiene un número par de ceros}\} = \{u \in \Sigma^* : \#_0(u) \text{ es par}\}.$$

Sabemos que $L_{\mathcal{P}}$ es regular y es aceptado por el siguiente AFD M :



Entonces M resuelve el problema de decisión para \mathcal{P} .

Ejemplo Sea $\Sigma = \{0, 1\}$. El problema de decisión

Dada $u \in \Sigma^*$, ¿es u un palíndromo?

no se puede resolver usando autómatas finitos. La propiedad \mathcal{P} en este caso es “ser un palíndromo” y $L_{\mathcal{P}}$ es el lenguaje de todos los palíndromos, el cual se demostró en la sección anterior que no es regular.

Ejemplo Sea $\Sigma = \{0, 1\}$. Demostrar que ningún autómata finito puede resolver el siguiente problema de decisión:

Dada $u \in \Sigma^*$, ¿tiene u el mismo número de ceros que de unos?

Solución. La propiedad \mathcal{P} en este caso es “tener el mismo número de ceros que de unos” y $L_{\mathcal{P}}$ es el lenguaje $L_{\mathcal{P}} = \{u \in \Sigma^* : \#_0(u) = \#_1(u)\}$. Utilizando los argumentos de la sección anterior (ya sea por contradicción o por el criterio de no regularidad), es posible demostrar que $L_{\mathcal{P}}$ no es regular (véase el ejercicio ② de la sección 3.1). Por consiguiente, este problema es insoluble por medio de autómatas finitos.

Ejercicios de la sección 3.3

Determinar si los siguientes problemas de decisión se pueden o no resolver utilizando autómatas finitos. Ya sea la respuesta afirmativa o negativa, se debe presentar una demostración rigurosa.

- ① $\Sigma = \{a, b\}$. Dada $u \in \Sigma^*$, ¿tiene u un número par de a s y un número impar de b s?
- ② $\Sigma = \{a, b\}$. Dada $u \in \Sigma^*$, ¿se cumple que $\#_a(u) = \#_b(u) + 1$?
- ③ $\Sigma = \{a, b\}$. Dada $u \in \Sigma^*$, ¿es el número de a s en u un múltiplo 3? Nota: 0 se considera múltiplo de cualquier otro número natural.
- ④ $\Sigma = \{a, b\}$. Dada $u \in \Sigma^*$, ¿es el número de a s en u un múltiplo del número de b s en u ?
- ⑤ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿el primer símbolo de u (de izquierda a derecha) coincide con el último símbolo de u (de izquierda a derecha)?
- ⑥ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿tiene u por lo menos un cero y por lo menos dos unos?

- ⑦ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿aparece la subcadena 01 una única vez en u ?
- ⑧ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿se cumple $\#_0(u) \neq \#_1(u)$?
- ⑨ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿se cumple $\#_1(u) > \#_0(u)$?
- ⑩ $\Sigma = \{0, 1\}$. Dada $u \in \Sigma^*$, ¿se puede escribir u como la concatenación de dos mitades iguales? Es decir, ¿es $u = ww$, para alguna cadena $w \in \Sigma^*$?

Capítulo 4

Lenguajes y gramáticas independientes del contexto

Como se ha visto, los autómatas son dispositivos que leen cadenas de entrada, símbolo a símbolo. En capítulos posteriores consideraremos modelos de autómatas con mayor poder computacional que el de los modelos AFD, AFN y AFN- λ . En el presente capítulo estudiaremos una noción completamente diferente, aunque relacionada, la de gramática generativa, que es un mecanismo para generar cadenas a partir de un símbolo inicial.

- ☞ Los autómatas *leen* cadenas
- ☞ Las gramáticas *generan* cadenas

4.1. Gramáticas generativas

Las gramáticas generativas fueron introducidas por Noam Chomsky en 1956 como un modelo para la descripción de los lenguajes naturales (español, inglés, etc). Chomsky clasificó las gramáticas en cuatro tipos: 0, 1, 2 y 3. Las gramáticas de tipo 2, también llamadas gramáticas independientes del contexto, se comenzaron a usar en la década de los sesenta del siglo XX para presentar la sintaxis de lenguajes de programación y para el diseño de analizadores sintácticos en compiladores.

Una *gramática generativa* es una cuádrupla, $G = (V, \Sigma, S, P)$ formada por dos alfabetos disyuntos V (alfabeto de *variables* o *no-terminales*) y Σ (alfabeto de *terminales*), una variable especial $S \in V$ (llamada *símbolo inicial*) y un conjunto finito $P \subseteq (V \cup \Sigma)^* \times (V \cup \Sigma)^*$ de *producciones* o *reglas de re-escritura*. Una producción $(u, v) \in P$ se denota por $u \rightarrow v$ y se lee “ u produce v ”; u se denomina la *cabeza* y v el *cuerpo* de la producción. Se exige que la cabeza de la producción tenga por lo menos una variable.

El significado de la producción $u \rightarrow v$ es: la cadena u se puede reemplazar (sobre-escribir) por la cadena v . Comenzando con el símbolo inicial S y aplicando las producciones de la gramática, en uno o más pasos, se obtienen cadenas de terminales y/o no-terminales. Aquellas cadenas que sólo tengan terminales conforman lo que se denomina el *lenguaje generado por G* .

Las gramáticas se clasifican de acuerdo con el tipo de sus producciones:

Gramáticas de tipo 0. No tienen restricciones. También se llaman *gramáticas no-restringidas* o *gramáticas con estructura de frase* en razón de su origen lingüístico.

Gramáticas de tipo 1. Las producciones son de la forma $u_1 A u_2 \rightarrow v_1 v v_2$, donde A es una variable y $v \neq \lambda$. También se llaman *gramáticas sensibles al contexto* o *gramáticas contextuales*.

Gramáticas de tipo 2. Las producciones son de la forma $A \rightarrow w$ donde A es una variable. También se llaman *gramáticas independientes del contexto* o *gramáticas no-contextuales*.

Gramáticas de tipo 3. Las producciones son de la forma $A \rightarrow a$ o de la forma $A \rightarrow aB$, donde A y B son variables y a es un símbolo terminal. También se llaman *gramáticas regulares*.

Se dice que un lenguaje es de tipo i si es generado por una gramática de tipo i . Esta clasificación de lenguajes se conoce como la *jerarquía de Chomsky*.

4.2. Gramáticas independientes del contexto

Una *gramática independiente del contexto* (GIC), también llamada *gramática no-contextual* o *gramática de tipo 2*, es una cuádrupla, $G = (V, \Sigma, S, P)$ formada por:

1. Un alfabeto V cuyos elementos se llaman *variables* o *símbolos no-terminales*.
2. Un alfabeto Σ cuyos elementos se llaman *símbolos terminales*. Se exige que los alfabetos Σ y V sean disyuntos.
3. Una variable especial $S \in V$, llamada *variable inicial* o *símbolo inicial* de la gramática.
4. Un conjunto finito $P \subseteq V \times (V \cup \Sigma)^*$ de *producciones* o *reglas de re-escritura*. Una producción $(A, v) \in P$ de G se denota por $A \rightarrow v$ y se lee “ A produce v ”. A es una variable y se denomina la *cabeza* de la producción; v es una cadena en $(V \cup \Sigma)^*$, y se denomina el *cuerpo* de la producción (formado por concatenaciones de terminales y/o no-terminales). El significado de una producción $A \rightarrow v$ es el siguiente: la variable A se puede reemplazar (o sobre-escribir) por la cadena v .

Notación y definiciones. En ejemplos concretos, las variables se denotan con letras mayúsculas A, B, C, \dots mientras que los elementos de Σ o símbolos terminales se denotan con letras minúsculas a, b, c, \dots . Si $A \rightarrow v$ es una producción, entonces en una cadena como xAy , donde x y $y \in (V \cup \Sigma)^*$, la variable A se puede reemplazar por v para obtener xvy ; esto se denota como

$$xAy \Longrightarrow xvy.$$

Se dice que xvy se *deriva* o se *genera directamente* (o en un paso) de xAy . Si se quiere hacer referencia a la gramática G , se escribe

$$xAy \xRightarrow{G} xvy \quad \text{ó} \quad xAy \Longrightarrow_G xvy.$$

Si u_1, u_2, \dots, u_n son cadenas en $(V \cup \Sigma)^*$ y hay una sucesión de derivaciones directas

$$u_1 \xRightarrow{G} u_2, \quad u_2 \xRightarrow{G} u_3, \quad \dots, \quad u_{n-1} \xRightarrow{G} u_n$$

se dice que u_n se deriva de u_1 y se escribe $u_1 \xRightarrow{*} u_n$. La anterior sucesión de derivaciones directas se representa como

$$u_1 \Longrightarrow u_2 \Longrightarrow u_3 \Longrightarrow \dots \Longrightarrow u_{n-1} \Longrightarrow u_n$$

y se dice que es una *derivación* o una *generación* de u_n a partir de u_1 . Para toda cadena w se asume que $w \xRightarrow{*} w$; por lo tanto, $u \xRightarrow{*} v$ significa que v se obtiene de u utilizando cero, una o más producciones de la gramática. Análogamente, $u \xRightarrow{+} v$ significa que v se obtiene de u utilizando una o más producciones. Nótese que se utilizan flechas simples, \rightarrow , para producciones y flechas dobles, \Longrightarrow , al aplicar las producciones en derivaciones concretas.

El *lenguaje generado por una gramática* G se denota por $L(G)$ y se define como

$$L(G) := \{w \in \Sigma^* : S \xRightarrow{+} w\}.$$

Es decir, el lenguaje generado por G está formado por las cadenas de terminales que se pueden derivar (o generar) en varios pasos a partir de la variable inicial S , aplicando en cada paso una producción. La igualdad $L(G) = L$ es estricta y requiere que se satisfagan las dos contencencias $L(G) \subseteq L$ y $L \subseteq L(G)$; es decir, toda cadena generada por G debe estar en L , y toda cadena de L debe ser generada por G .

Un lenguaje L sobre un alfabeto Σ se dice que es un *lenguaje independiente del contexto* (LIC) o *lenguaje no-contextual* si existe una GIC G tal que $L(G) = L$. Dos GIC G_1 y G_2 son *equivalentes* si $L(G_1) = L(G_2)$.

La denominación “independiente del contexto” proviene del hecho de que en una derivación cada producción o regla de re-escritura $A \rightarrow v$ se aplica a la variable A independientemente de los caracteres que la rodean, es decir, independientemente del contexto en el que aparece A . En inglés, estas gramáticas se denominan *context-free grammars*,

traducido en ocasiones, de manera no muy apropiada, como “gramáticas de contexto libre”.

Ejemplo Sea $G = (V, \Sigma, S, P)$ una gramática dada por:

$$\begin{aligned} V &= \{S, A\} \\ \Sigma &= \{a, b\} \\ P &= \{S \rightarrow aS, S \rightarrow bA, S \rightarrow \lambda, A \rightarrow bA, A \rightarrow b, A \rightarrow \lambda\}. \end{aligned}$$

La manera más conveniente de presentar una gramática es listando sus producciones y separando con una barra $|$ las producciones de una misma variable. Se supone siempre que las letras mayúsculas representan variables y las letras minúsculas representan símbolos terminales. Así la gramática G del presente ejemplo se puede presentar simplemente como:

$$G: \begin{cases} S \rightarrow aS \mid bA \mid \lambda \\ A \rightarrow bA \mid b \mid \lambda \end{cases}$$

Se tiene $S \Rightarrow \lambda$. Todas las demás derivaciones en G comienzan ya sea con la producción $S \rightarrow aS$ o con $S \rightarrow bA$. Por lo tanto, tenemos

$$\begin{aligned} S &\Rightarrow aS \xRightarrow{*} a \cdots aS \Rightarrow a \cdots a. \\ S &\Rightarrow bA \xRightarrow{*} b \cdots bA \Rightarrow b \cdots b. \\ S &\Rightarrow aS \xRightarrow{*} a \cdots aS \Rightarrow a \cdots abA \xRightarrow{*} a \cdots ab \cdots bA \Rightarrow a \cdots ab \cdots b. \end{aligned}$$

Por consiguiente $L(G) = a^*b^*$.

Las siguientes cuatro gramáticas también generan el lenguaje a^*b^* y son, por lo tanto, equivalentes a G :

$$\begin{aligned} G_1: & \begin{cases} S \rightarrow aS \mid bA \mid \lambda \\ A \rightarrow bA \mid \lambda \end{cases} & G_2: & \begin{cases} S \rightarrow aS \mid A \\ A \rightarrow bA \mid \lambda \end{cases} \\ G_3: & \begin{cases} S \rightarrow AB \\ A \rightarrow aA \mid \lambda \\ B \rightarrow bB \mid \lambda \end{cases} & G_4: & \begin{cases} S \rightarrow AB \mid \lambda \\ A \rightarrow aA \mid a \mid \lambda \\ B \rightarrow bB \mid b \mid \lambda \end{cases} \end{aligned}$$

Para generar la cadena vacía λ con la gramática G_3 se requieren tres pasos:

$$S \Rightarrow AB \Rightarrow B \Rightarrow \lambda.$$

Ejemplo La gramática

$$G: \begin{cases} S \rightarrow aS \mid aA \\ A \rightarrow bA \mid b \end{cases}$$

genera el lenguaje a^+b^+ . Otra gramática equivalente es:

$$G' : \begin{cases} S \rightarrow AB \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid b \end{cases}$$

Ejemplo La gramática

$$\begin{cases} S \rightarrow 1A \mid 0 \\ A \rightarrow 0A \mid 1A \mid \lambda \end{cases}$$

genera el lenguaje de los números naturales en numeración binaria. Nótese que la única cadena que comienza con 0, generable con esta gramática, es la cadena 0.

Ejemplo Encontrar una GIC que genere el lenguaje $L = 0^*10^*10^*$ sobre $\Sigma = \{0, 1\}$, es decir, el lenguaje de todas las cadenas con exactamente dos unos.

Solución.

$$G : \begin{cases} S \rightarrow A1A1A \\ A \rightarrow 0A \mid \lambda \end{cases}$$

Una gramática equivalente es

$$G' : \begin{cases} S \rightarrow 0S \mid 1A \\ A \rightarrow 0A \mid 1B \\ B \rightarrow 0B \mid \lambda \end{cases}$$

Esta última gramática G' se puede obtener a partir de un autómata que acepte el lenguaje L , tal como se explicará en la sección 4.3.

Ejemplo Encontrar una GIC que genere el lenguaje $L = \{a^n b^n : n \geq 0\}$ sobre $\Sigma = \{a, b\}$, el cual no es un lenguaje regular.

Solución.

$$S \rightarrow aSb \mid \lambda.$$

Ejemplo Encontrar una GIC que genere el lenguaje de todos los palíndromos sobre $\Sigma = \{a, b\}$, el cual no es lenguaje regular.

Solución.

$$S \rightarrow aSa \mid bSb \mid a \mid b \mid \lambda.$$

Ejemplo Encontrar una GIC que genere el lenguaje L de todas las cadenas sobre $\Sigma = \{a, b\}$ que tienen un número par de símbolos.

Solución. Las cadenas de longitud par (aparte de la cadena vacía λ) se obtienen concatenando los cuatro bloques aa , ab , ba y bb . Por lo tanto, para generar el lenguaje L basta

una sola variable que permita concatenar los cuatro bloques de todas las formas posibles. Las siguientes tres gramáticas generan el lenguaje L :

$$G_1 : \left\{ S \rightarrow aaS \mid abS \mid baS \mid bbS \mid \lambda \right.$$

$$G_2 : \left\{ S \rightarrow Saa \mid Sab \mid Sba \mid Sbb \mid \lambda \right.$$

$$G_3 : \left\{ S \rightarrow aSa \mid aSb \mid bSa \mid bSb \mid \lambda \right.$$

G_1 genera las cadenas de L de izquierda a derecha, G_2 las genera de derecha a izquierda y G_3 por simetría izquierda-derecha. Así por ejemplo, la cadena $baabbb$ se genera de la siguiente manera en las tres gramáticas:

$$\text{En } G_1 : S \Rightarrow baS \Rightarrow baabS \Rightarrow baabbbS \Rightarrow baabbb.$$

$$\text{En } G_2 : S \Rightarrow Sbb \Rightarrow Sabbb \Rightarrow Sbaabbb \Rightarrow baabbb.$$

$$\text{En } G_3 : S \Rightarrow bSb \Rightarrow baSbb \Rightarrow baaSbbb \Rightarrow baabbb.$$

Si se combinan las producciones de G_1 con las de G_2 no necesariamente se genera el lenguaje L . Por ejemplo, la gramática G_4 ,

$$G_4 : \left\{ S \rightarrow aaS \mid Sab \mid baS \mid Sbb \mid \lambda \right.$$

genera ciertamente cadenas de longitud par, pero no las genera todas ya que es imposible generar cadenas como $abbabb$ y $aaabba$ (y muchas otras). Se cumple que $L(G_4) \subseteq L$ pero $L \not\subseteq L(G_4)$ y por ende no se tiene la igualdad $L = L(G_4)$.

Otra gramática que genera el lenguaje L es:

$$G_5 : \left\{ \begin{array}{l} S \rightarrow AAS \mid \lambda \\ A \rightarrow a \mid b \end{array} \right.$$

Ejemplo Encontrar una GIC que genere el lenguaje

$$L = \{a^k b^m c^n : m = k + n, k, m, n \geq 0\}$$

sobre el alfabeto $\Sigma = \{a, b, c\}$.

Solución. Las cadenas de L se pueden escribir como $a^k b^m c^n = a^k b^{k+n} c^n = a^k b^k b^n c^n$. Utilizamos la variable A para generar $a^k b^k$, con $k \geq 0$, y la variable B para generar $b^n c^n$, con $n \geq 0$:

$$G : \left\{ \begin{array}{l} S \rightarrow AB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow bBc \mid \lambda \end{array} \right.$$

Ejemplo Encontrar una GIC que genere el lenguaje

$$L = \{a^k b^m c^n : m > k + n, k, n \geq 0, m \geq 1\}$$

sobre el alfabeto $\Sigma = \{a, b, c\}$.

Solución. Para resolver este problema, utilizamos como punto de partida la gramática G del ejemplo anterior. La condición $m > k + n$ significa que hay estrictamente más *bes* que el total $k + n$; el exceso deseado de *bes* lo podemos obtener por medio de la producción $A \rightarrow Ab$. Obsérvese que el lenguaje L incluye cadenas que solamente tienen *bes* (cuando $k = n = 0$).

$$G_1 : \begin{cases} S \rightarrow AB \\ A \rightarrow aAb \mid Ab \mid b \\ B \rightarrow bBc \mid \lambda \end{cases}$$

Las *bes* adicionales también se pueden generar por medio de la variable B , como se hace en la siguiente gramática:

$$G_2 : \begin{cases} S \rightarrow AB \\ A \rightarrow aAb \mid Ab \mid \lambda \\ B \rightarrow bBc \mid bB \mid b \end{cases}$$

Otra manera de generar este lenguaje es utilizar una nueva variable C , colocada entre A y B , que genere el exceso de *bes*. Obtenemos así la gramática G_3 ,

$$G_3 : \begin{cases} S \rightarrow ACB \\ A \rightarrow aAb \mid \lambda \\ B \rightarrow bBc \mid \lambda \\ C \rightarrow bC \mid b \end{cases}$$

Ejercicios de la sección 4.2

① Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b\}$:

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

② Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b\}$:

- (i) $\{a^{n+1}b^{2n+1} : n \geq 0\}$.
- (ii) $\{a^n b^{n+1} a : n \geq 1\}$.
- (iii) $\{a^n b^m a^{n+1} : n \geq 0, m \geq 1\}$.

- (iv) $\{a^{n+1}b^m a^{2n} : n, m \geq 0\}$.
 - (v) $\{a^m b^n : m > n \geq 0\}$.
 - (vi) $\{a^m b^n : m, n \geq 0, m \neq n\}$.
 - (vii) $\{a^m b^n : m, n \geq 0, n > 2m\}$.
 - (viii) $\{a^m b^n : 0 \leq m \leq n \leq 2m\}$.
- ③ Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b, c\}$:
- (i) $\{a^k b^m a^n : k, m, n \geq 0, n = k + 2m\}$
 - (ii) $\{a^k b^m a^n : k, m \geq 0, n \geq 1, n > k + 2m\}$
 - (iii) $\{a^k b^m a^n : k, m, n \geq 0, k = m + 2n\}$
 - (iv) $\{a^k b^m a^n : m, n \geq 0, k \geq 1, k > m + 2n\}$
 - (v) $\{a^k b^m a^n : k, m, n \geq 0, m = 2k + n\}$
 - (vi) $\{a^k b^m a^n : k, n \geq 0, m \geq 1, m > 2k + n\}$
- ④ Encontrar GIC que generen los siguientes lenguajes sobre $\Sigma = \{a, b, c, d\}$:
- (i) $\{a^m b^{2n} c^n d^{2m} : m, n \geq 1\}$.
 - (ii) $\{a^{2n} b^n c^m d^{2m} : m, n \geq 1\}$.
- ⑤ Sea $\Sigma = \{0, 1\}$. Encontrar una GIC que genere el lenguaje de las cadenas que tienen igual número de ceros que de unos.

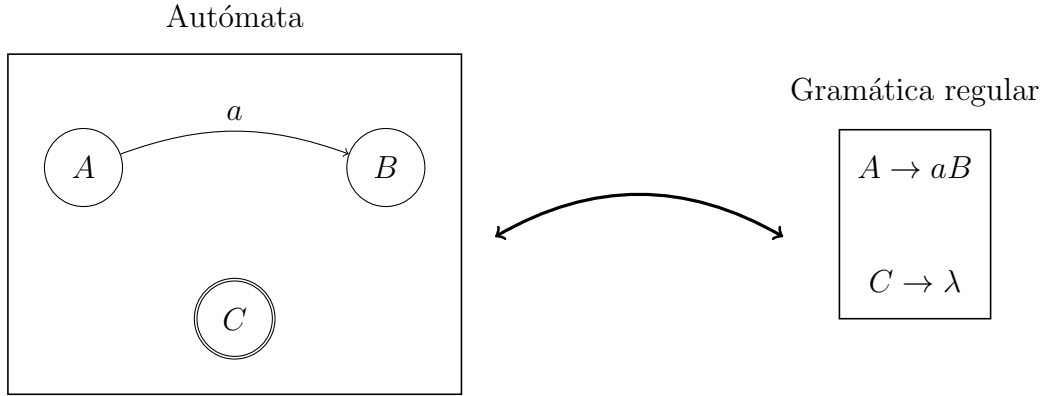
4.3. Gramáticas regulares

4.3.1 Definición. Una GIC $G = (V, \Sigma, S, P)$ se llama *regular* si sus producciones son de la forma

$$\begin{cases} A \rightarrow aB, & a \in \Sigma, A, B \in V. \\ A \rightarrow \lambda & A \in V. \end{cases}$$

En la producción $A \rightarrow aB$, las variables A y B no necesariamente son distintas.

A partir de un autómata AFD o AFN M se puede obtener una gramática regular G , tal que $L(M) = L(G)$, siguiendo el procedimiento esquematizado en el diagrama siguiente. Los estados de M se convierten en las variables de G ; el estado inicial de M pasa a ser la variable inicial de G . Un arco del estado A al estado B con etiqueta a da lugar a la producción $A \rightarrow aB$, y un estado de aceptación C da lugar a la producción $C \rightarrow \lambda$.



El procedimiento es completamente reversible y establece una correspondencia directa entre autómatas y gramáticas regulares. En el Teorema 4.3.2 se demuestra que si M es un AFD, la gramática regular G obtenida satisface $L(M) = L(G)$, y en el Teorema 4.3.3 se extiende este resultado a autómatas no-deterministas.

4.3.2 Teorema. Dado un AFD $M = (\Sigma, Q, q_0, F, \delta)$, existe una GIC regular $G = (V, \Sigma, S, P)$ tal que $L(M) = L(G)$.

Demostración. Sea $V = Q$ y $S = q_0$. Las producciones de G están dadas por

$$\begin{cases} q \rightarrow ap & \text{si y sólo si } \delta(q, a) = p. \\ q \rightarrow \lambda & \text{si y sólo si } q \in F. \end{cases}$$

Demostraremos primero que para toda $w \in \Sigma^*$, $w \neq \lambda$ y para todo $p, q \in Q$ se tiene

(1) Si $\delta(q, w) = p$ entonces $q \xRightarrow{*} wp$.

La demostración de (1) se hace por inducción sobre w . Si $w = a$ y $\delta(q, a) = p$, entonces $q \rightarrow ap$ es una producción de G y obviamente se concluye $q \Rightarrow ap$. Para el paso inductivo, sea $\delta(q, wa) = p'$. Entonces

$$p' = \delta(q, wa) = \delta(\delta(q, w), a) = \delta(p, a)$$

donde $\delta(q, w) = p$. Por hipótesis de inducción $q \xRightarrow{*} wp$ y como $\delta(p, a) = p'$, entonces $p \Rightarrow ap'$. Por lo tanto,

$$q \xRightarrow{*} wp \Rightarrow wap'$$

que era lo que se quería demostrar.

A continuación demostraremos el recíproco de (1): para toda $w \in \Sigma^*$, $w \neq \lambda$ y para todo $p, q \in Q$ se tiene

(2) Si $q \xRightarrow{*} wp$ entonces $\delta(q, w) = p$.

La demostración de (2) se hace por inducción sobre la longitud de la derivación $q \xRightarrow{*} wp$, es decir, por el número de pasos o derivaciones directas que hay en $q \xRightarrow{*} wp$. Si la derivación tiene longitud 1, necesariamente $q \Rightarrow ap$ lo cual significa que $\delta(q, a) = p$. Para

el paso inductivo, supóngase que $q \xRightarrow{*} wp$ tiene longitud $n + 1$, $w = w'a$ y en el último paso se aplica la producción $p' \rightarrow ap$. Entonces

$$q \xRightarrow{*} w'p' \xRightarrow{*} w'ap = wp.$$

Por hipótesis de inducción, $\delta(q, w') = p'$ y por consiguiente

$$\delta(q, w) = \delta(q, w'a) = \delta(\delta(q, w'), a) = \delta(p', a) = p,$$

que era lo que se quería demostrar.

Como consecuencia de (1) y (2) se puede ahora demostrar que

(3) Para toda cadena $w \in \Sigma^*$, $\delta(q_0, w) \in F$ si y sólo si $S \xRightarrow{*}_G w$,

lo cual afirma que $L(M) = L(G)$. En efecto, si $w = \lambda$, $\delta(q_0, w) \in F$ si y sólo si $q_0 \in F$. Por lo tanto, $q_0 \rightarrow \lambda$ es una producción de G . Así que $S \xRightarrow{*} \lambda$. Recíprocamente, si $S \xRightarrow{*} \lambda$, necesariamente $S \xRightarrow{*} \lambda$, $q_0 \in F$ y $\delta(q_0, \lambda) \in F$.

Sea ahora $w \neq \lambda$. Si $\delta(q_0, w) = p \in F$, por (1) se tiene $q_0 \xRightarrow{*} w$, o sea, $S \xRightarrow{*} w$. Recíprocamente, si $S \xRightarrow{*}_G w$, entonces $q_0 \xRightarrow{*}_G wp \xRightarrow{*} w$ donde $p \rightarrow \lambda$. Utilizando (2), se tiene $\delta(q_0, w) = p \in F$. \square

4.3.3 Teorema. Dada una GIC regular $G = (V, \Sigma, S, P)$, existe un AFN $M = (Q, \Sigma, q_0, F, \Delta)$ tal que $L(M) = L(G)$.

Demostración. Se construye $M = (Q, \Sigma, q_0, F, \Delta)$ haciendo $Q = V$, $q_0 = S$ y

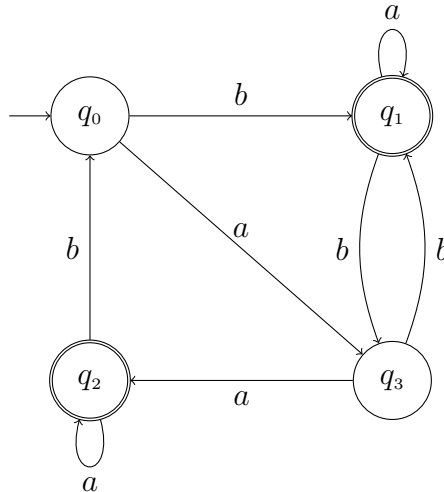
$$\begin{cases} B \in \Delta(A, a) & \text{para cada producción } A \rightarrow aB. \\ A \in F & \text{si } A \rightarrow \lambda. \end{cases}$$

Usando razonamientos similares a los del Teorema 4.3.2, se puede demostrar que

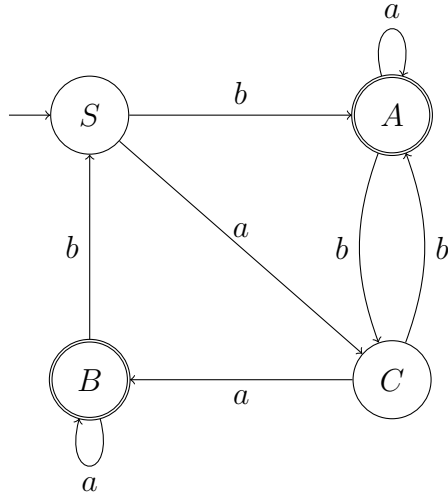
$$A \xRightarrow{*}_G wB \quad \text{si y sólo si} \quad B \in \Delta(A, w), \text{ para todo } w \in \Sigma^*, w \neq \lambda,$$

de donde $L(M) = L(G)$. \square

Ejemplo Dado el siguiente AFD M , encontrar una gramática regular G tal que $L(M) = L(G)$.



Solución. Según la construcción mencionada arriba, los estados del autómata M son las variables de la gramática G . Renombramos los estados de M con las letras mayúsculas S , A , B y C . Toda transición de M da lugar a una producción en G ; los estados de aceptación A y B inducen las producciones $A \rightarrow \lambda$ y $B \rightarrow \lambda$, respectivamente.



$$G : \begin{cases} S \rightarrow bA \mid aC \\ A \rightarrow aA \mid bC \mid \lambda \\ B \rightarrow bS \mid aB \mid \lambda \\ C \rightarrow bA \mid aB \end{cases}$$

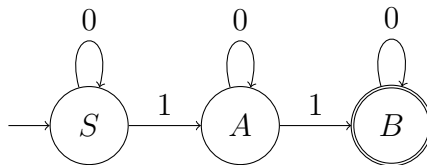
Puesto que el autómata M es determinista, para cada cadena aceptada u existe una única trayectoria etiquetada por los símbolos de u , desde el estado inicial hasta un estado de aceptación; tal trayectoria corresponde a una única derivación $S \xRightarrow{*} u$ en la gramática regular G . Así por ejemplo, la cadena $baabaa$ tiene una única trayectoria de aceptación, a saber, $S \xrightarrow{b} A \xrightarrow{a} A \xrightarrow{a} A \xrightarrow{b} C \xrightarrow{a} B \xrightarrow{a} B$, la cual corresponde a una única derivación en G :

$$S \Rightarrow bA \Rightarrow baA \Rightarrow baaA \Rightarrow baabC \Rightarrow baabaB \Rightarrow baabaaB \Rightarrow baabaa.$$

Ejemplo Para el lenguaje regular $0^*10^*10^*$, sobre $\Sigma = \{0, 1\}$ (el lenguaje de todas las cadenas con exactamente dos unos), vimos en la sección 4.2 una gramática G que lo genera:

$$G : \begin{cases} S \rightarrow A1A1A \\ A \rightarrow 0A \mid \lambda \end{cases}$$

Esta gramática no es regular, pero por medio del AFD



y de la construcción del Teorema 4.3.2 se puede obtener una GIC regular G' que genere $0^*10^*10^*$:

$$G' : \begin{cases} S \rightarrow 0S \mid 1A \\ A \rightarrow 0A \mid 1B \\ B \rightarrow 0B \mid \lambda \end{cases}$$

Los teoremas anteriores permiten concluir que la familia de los lenguajes regulares está estrictamente contenida en la familia de los Lenguajes Independientes del Contexto, tal como se enuncia en el siguiente corolario.

4.3.4 Corolario.

1. Un lenguaje es regular si y solamente si es generado por una gramática regular.
2. Todo lenguaje regular es un LIC (pero no viceversa).

Demostración.

1. Se sigue del Teorema 4.3.2, el Teorema 4.3.3 y del Teorema de Kleene.
2. Se sigue de la parte 1. Por otro lado, tenemos muchos ejemplos de lenguajes LIC que no son regulares, como $\{a^n b^n : n \geq 0\}$ y el lenguaje de los palíndromes sobre el alfabeto $\{a, b\}$. \square

Ejercicios de la sección 4.3

Encontrar gramáticas regulares que generen los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b\}$:

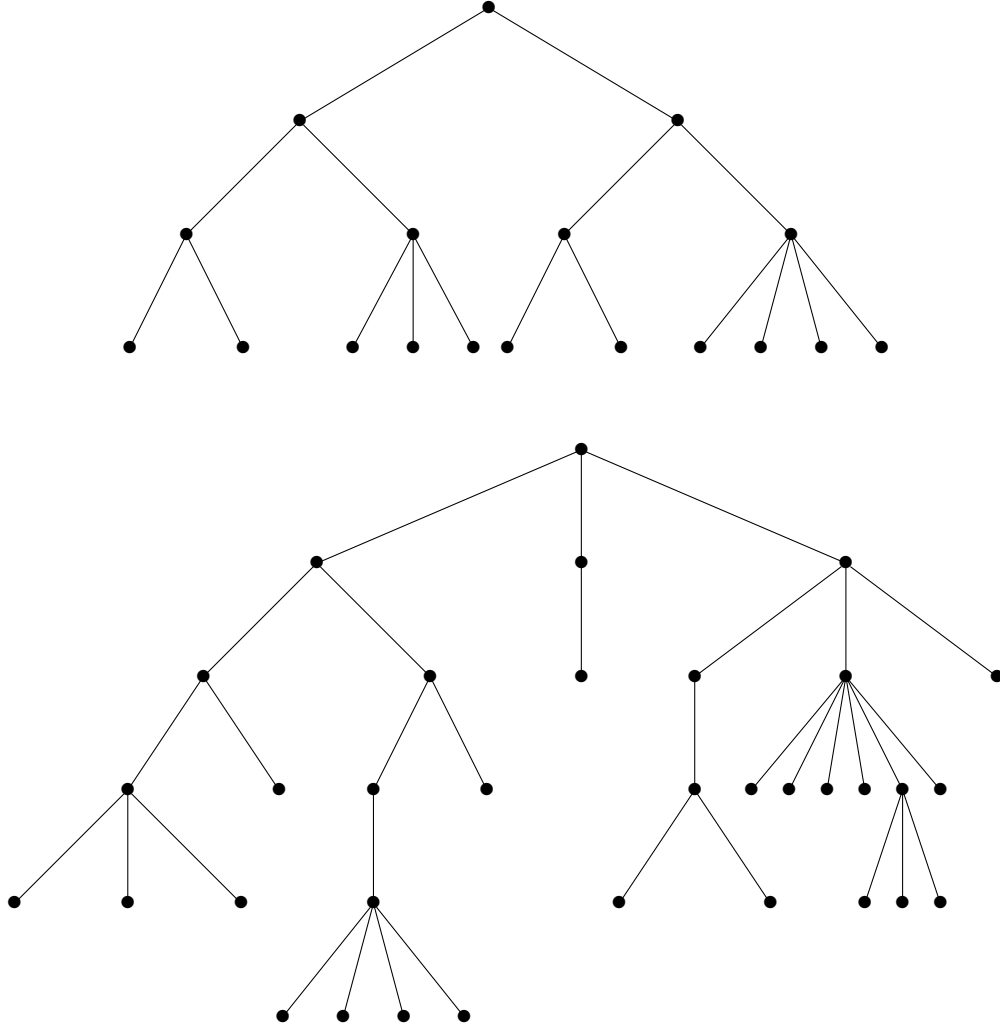
- ① $ba^*b \cup b^+$.
- ② a^+b^*a .
- ③ $a^*b \cup b^*a$.
- ④ El lenguaje de todas las cadenas que tienen un número impar de a 'es y un número par de b 'es.

4.4. Árboles sintácticos

Un *árbol con raíz* es un tipo muy particular de grafo no-dirigido; tiene un nodo especial, llamado la *raíz* del árbol, la cual se ramifica en nodos, llamados *descendientes inmediatos*, cada uno de los cuales puede tener, a su vez, descendientes inmediatos, y así sucesivamente. Un nodo puede tener 0, 1, o más descendientes inmediatos pero tiene un único antecesor inmediato. El único nodo que no tiene antecesores es la raíz. Los nodos que tienen descendientes, excepto la raíz, se denominan *nodos interiores*. En la terminología

usualmente utilizada, los descendientes inmediatos de un nodo también se denominan *hijos*, y los nodos que no tienen descendientes se denominan *hojas*. Un árbol queda caracterizado por la siguiente propiedad: hay una única trayectoria entre la raíz y cualquier otro nodo. Los nodos que aparecen en la única trayectoria entre la raíz y un nodo determinado N se denominan los *ancestros* de N .

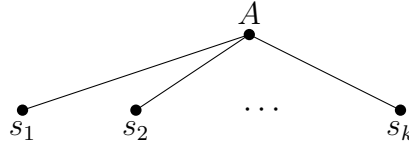
Ejemplo Dos árboles con raíz:



4.4.1 Definición. Dada una GIC $G = (V, \Sigma, S, P)$, el *árbol de una derivación* $S \xRightarrow{+} w$, con $w \in \Sigma^*$, es un árbol con raíz y con nodos etiquetados, definido recursivamente de la siguiente forma:

1. La raíz está etiquetada con el símbolo inicial S .
2. Si en un paso la derivación se aplica la producción $A \rightarrow s_1 s_2 \cdots s_k$, donde cada $s_i \in V \cup \Sigma$, el nodo A tiene k descendientes inmediatos etiquetados con s_1, s_2, \dots ,

s_k , escritos de izquierda a derecha:



Si en un paso la derivación se aplica la producción $A \rightarrow \lambda$, el nodo A tiene un único descendiente etiquetado con λ :



De esta manera, los nodos interiores están etiquetados con símbolos no terminales, y las hojas del árbol están etiquetadas con símbolos terminales o con λ . Si se leen de izquierda a derecha las hojas del árbol de una derivación de $S \xRightarrow{+} w$, se obtiene precisamente la cadena w , con algunos λ intercalados.

Los árboles de derivaciones se suelen llamar *árboles sintácticos*.

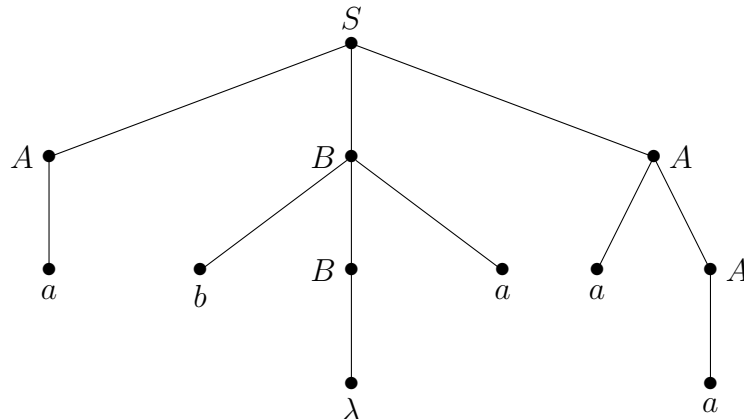
Ejemplo Sea G la gramática:

$$G : \begin{cases} S \rightarrow ABA \mid AaB \\ A \rightarrow aA \mid a \\ B \rightarrow bBa \mid \lambda \end{cases}$$

Consideremos la siguiente derivación de la cadena $abaaa$:

$$(1) \quad S \Rightarrow \underline{A}BA \Rightarrow Ab\underline{B}a\underline{A} \Rightarrow Ab\underline{B}aaA \Rightarrow Abaa\underline{A} \Rightarrow Abaaa \Rightarrow abaaa.$$

En cada paso de la derivación, se ha subrayado la variable para la cual se ha utilizado una producción de la gramática G . El árbol de la derivación (1) es:



Las producciones utilizadas en la derivación (1) se pueden aplicar en diferente orden; obtenemos, por ejemplo, las siguientes derivaciones:

$$(2) \quad S \Rightarrow \underline{A}BA \Rightarrow aB\underline{A} \Rightarrow aBa\underline{A} \Rightarrow aBaa \Rightarrow abBaaa \Rightarrow abaaaa.$$

$$(3) \quad S \Rightarrow \underline{A}BA \Rightarrow a\underline{B}A \Rightarrow ab\underline{B}aA \Rightarrow abaA \Rightarrow abaaA \Rightarrow abaaaa.$$

Las derivaciones (1), (2) y (3) tienen todas seis pasos y ellas se aplican exactamente las mismas producciones pero en diferente orden. Las tres derivaciones tienen todas el mismo árbol, exhibido arriba. Los árboles sintácticos muestran únicamente las producciones utilizadas, no el orden en que se aplican.

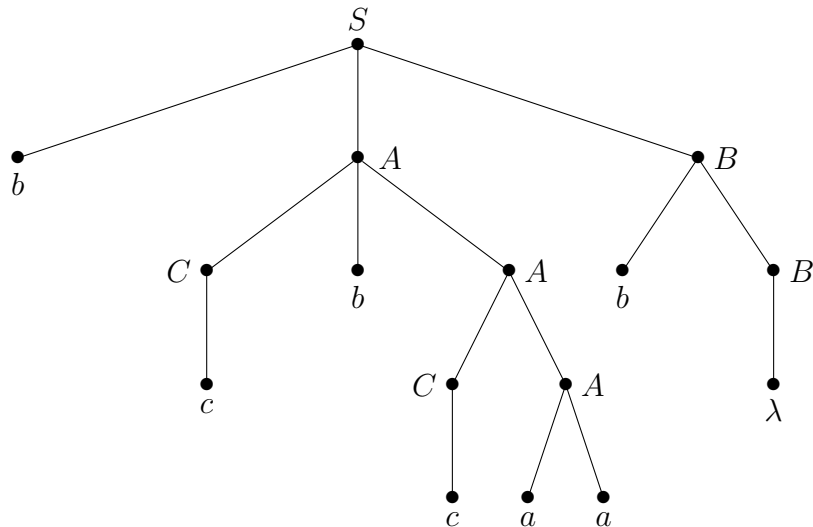
Entre las posibles derivaciones de cadenas se distinguen ciertas derivaciones “estándares”, las llamadas derivaciones a izquierda.

4.4.2 Definición. Una derivación se llama *derivación a izquierda* (o *derivación más a la izquierda*) si en cada paso se aplica una producción a la variable que está más a la izquierda.

En el ejemplo anterior, la derivación (3) es una derivación a izquierda. En general, una derivación cualquiera se puede transformar siempre en una única derivación a izquierda cambiando el orden en que se aplican las producciones, de tal forma que en cada paso se aplica una producción a la variable que esté más a la izquierda. Además, existe una correspondencia biyectiva entre derivaciones a izquierda y árboles sintácticos, tal como se enuncia en la siguiente proposición.

4.4.3 Proposición. Toda derivación a izquierda determina un único árbol sintáctico. Recíprocamente, cualquier árbol sintáctico en una gramática G determina una única derivación a izquierda en G .

Ejemplo Encontrar la única derivación a izquierda determinada por el siguiente árbol sintáctico T proveniente de cierta gramática G con alfabeto de variables $V = \{S, A, B, C\}$ y alfabeto de terminales $\Sigma = \{a, b, c\}$.



Solución.

$$\begin{aligned} S &\Rightarrow b\underline{A}B \Rightarrow b\underline{C}bAB \Rightarrow bcb\underline{A}B \Rightarrow bcb\underline{C}AB \Rightarrow bcbcb\underline{A}B \Rightarrow bcbcbcaabB \\ &\Rightarrow bcbcbcaabB \Rightarrow bcbcbcaab\lambda. \end{aligned}$$

Las hojas del árbol sintáctico forman la cadena generada $bcbcbcaab\lambda = bcbcbcaab$.

Ejercicios de la sección 4.4

① Sea G siguiente gramática:

$$G : \begin{cases} S \rightarrow aS \mid AaB \\ A \rightarrow aA \mid a \\ B \rightarrow bBbB \mid b \end{cases}$$

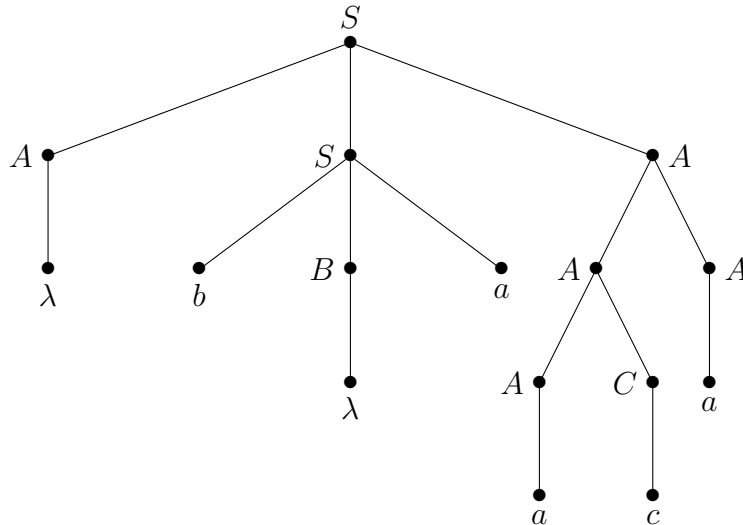
Encontrar una derivación de la cadena $aaaabbbb$ y hallar el árbol de tal derivación.

② Sea G la siguiente gramática:

$$G : \begin{cases} S \rightarrow ABC \mid BaC \mid aB \\ A \rightarrow Aa \mid a \\ B \rightarrow BAB \mid bab \\ C \rightarrow cC \mid \lambda \end{cases}$$

Encontrar derivaciones a izquierda de las cadenas $w_1 = abab$, $w_2 = babacc$, $w_3 = ababababc$ y hallar los árboles de tales derivaciones.

③ Encontrar la única derivación a izquierda determinada por el siguiente árbol sintáctico proveniente de cierta gramática G con alfabeto de variables $V = \{S, A, B, C\}$ y alfabeto de terminales $\Sigma = \{a, b, c\}$.



4.5. Gramáticas ambiguas

4.5.1 Definición. Una GIC $G = (V, \Sigma, S, P)$ es *ambigua* si existe (por lo menos) una cadena $w \in \Sigma^*$ para la cual hay dos derivaciones a izquierda diferentes. Según la Proposición 4.4.3, se puede decir de manera equivalente que una GIC $G = (V, \Sigma, S, P)$ es ambigua si existe una cadena $w \in \Sigma^*$ con dos árboles sintácticos diferentes.

Como consecuencia de esta definición, se concluye que una GIC $G = (V, \Sigma, S, P)$ *no es ambigua* cuando toda cadena $w \in \Sigma^*$ tiene una única derivación a izquierda. Equivalentemente, G no es ambigua cuando toda cadena $w \in \Sigma^*$ tiene un único árbol sintáctico.

Como se mostrará más adelante, las GIC se utilizan para formalizar la sintaxis de los lenguajes de programación, y resulta imperativo que las gramáticas involucradas sean no ambiguas ya que un programa debe tener una única interpretación.

Ejemplo Considérese el alfabeto de terminales $\Sigma = \{x, y, z, 0, 1, +, *\}$ y la gramática G_1 cuya única variable es S y cuyas producciones son:

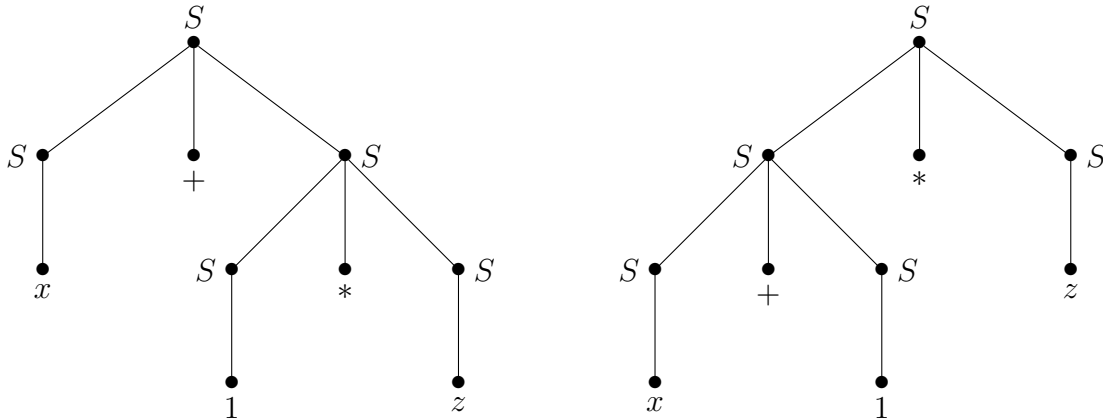
$$G_1 : \{S \rightarrow S + S \mid S * S \mid x \mid y \mid z \mid 0 \mid 1\}$$

Si se interpreta el símbolo $+$ como ‘suma’ y el símbolo $*$ como ‘producto’, G_1 genera expresiones algebraicas sencillas que involucran los terminales $x, y, z, 0$ y 1 . La ambigüedad surge porque algunas expresiones se pueden interpretar ya sea como sumas, o ya sea como productos. Por ejemplo, la cadena $x + 1 * z$ se puede generar como una suma o como un producto por medio de dos derivaciones a izquierda diferentes:

$$S \Rightarrow S + S \Rightarrow x + S \Rightarrow x + S * S \Rightarrow x + 1 * S \Rightarrow x + 1 * z.$$

$$S \Rightarrow S * S \Rightarrow S + S * S \Rightarrow x + S * S \Rightarrow x + 1 * S \Rightarrow x + 1 * z.$$

Los árboles de derivación correspondientes a las dos anteriores derivaciones son:



La ambigüedad en la gramática G_1 se puede eliminar introduciendo paréntesis, como se hace en la siguiente gramática G_2 :

$$G_2 : \{S \rightarrow (S + S) \mid (S * S) \mid x \mid y \mid z \mid 0 \mid 1\}$$

En G_2 el alfabeto de terminales sería $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$. En esta gramática se pueden generar las expresiones $((x + 1) * z)$ y $(x + (1 * z))$ que eliminan la ambigüedad de la expresión $x + 1 * z$.

Precedencia de los operadores. Aunque la introducción de paréntesis elimina la ambigüedad, las expresiones algebraicas generadas en la gramática G_2 de la página anterior tienen un excesivo número de paréntesis ya que cada aplicación de los operadores $+$ y $*$ produce un par de ellos. Esto dificulta el análisis sintáctico (en un compilador, por ejemplo). Lo más corriente en estas situaciones es utilizar gramáticas que establezcan un orden de precedencia para los operadores. Lo usual es establecer que $*$ tenga un mayor orden de precedencia que $+$, es decir, por convención $*$ actúa antes que $+$. Por ejemplo, la expresión $x + 1 * z$ se interpreta como $x + (1 * z)$ sin necesidad de usar paréntesis. En la siguiente gramática G_3 se implementa la precedencia del operador $*$ sobre el operador $+$ para generar expresiones algebraicas sobre el alfabeto de terminales $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$. El alfabeto de variables de G_3 es $V = \{S, T, F\}$.

$$G_3 : \begin{cases} S \rightarrow S + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow x \mid y \mid z \mid 0 \mid 1 \mid (S) \end{cases}$$

En G_3 toda expresión algebraica se genera como una suma de “términos”:

$$S \Rightarrow S + T \Rightarrow S + T + T \xRightarrow{*} S + T + \dots + T \Rightarrow T + T + \dots + T,$$

y todo término generado con la variable T es un producto de “factores”:

$$T \Rightarrow T * F \Rightarrow T * F * F \xRightarrow{*} T * F * \dots * F \Rightarrow F * F * \dots * F.$$

Uno de los posibles factores generados con la variable F es (S) , que se utiliza para generar factores encerrados entre paréntesis. Vamos a ilustrar cómo se puede generar en G_3 la expresión algebraica $x^2 + y(x + z^3 + 1)$. Puesto que G_3 únicamente posee los operadores aritméticos $+$ y $*$, dicha expresión se debe generar como $x * x + y * (x + z * z * z + 1)$.

$$\begin{aligned} S &\Rightarrow S + T \Rightarrow T + T \xRightarrow{2} T * F + T * F \xRightarrow{2} F * F + F * F \xRightarrow{4} x * x + y * (S) \\ &\xRightarrow{3} x * x + y * (T + T + T) \xRightarrow{4} x * x + y * (x + T + 1) \\ &\xRightarrow{3} x * x + y * (x + F * F * F + 1) \xRightarrow{3} x * x + y * (x + z * z * z + 1). \end{aligned}$$

Eliminación de la ambigüedad. La ambigüedad es un asunto delicado porque no existe un algoritmo o procedimiento general que permita decidir si una GIC dada G es o no ambigua. En otras palabras, el problema de decisión, “Dada una GIC G , ¿es G ambigua?” es matemáticamente insoluble. La inexistencia de un algoritmo para este problema de decisión es una imposibilidad matemática cuya demostración se realizará en un capítulo posterior. Por consiguiente, en cada caso concreto hay que proceder por

inspección o por ensayo y error, recurriendo a las definiciones, para poder concluir si una GIC dada es o no ambigua.

Ejemplo Demostrar que la siguiente gramática G es ambigua y encontrar una gramática G' no ambigua equivalente a G , es decir, tal que $L(G) = L(G')$.

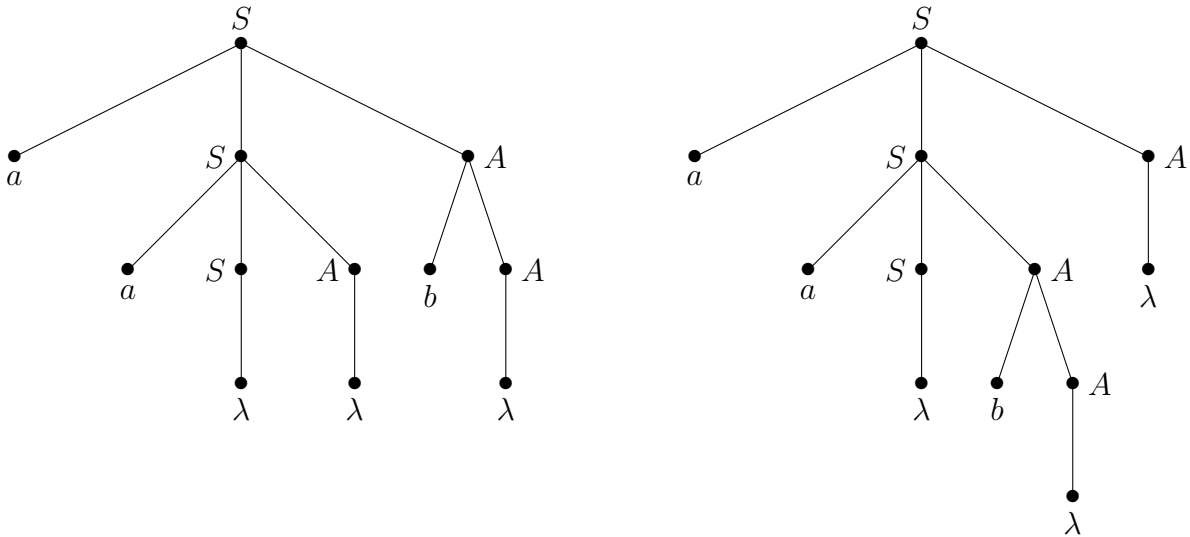
$$G : \begin{cases} S \rightarrow aSA \mid \lambda \\ A \rightarrow bA \mid \lambda \end{cases}$$

Solución. G es ambigua porque hay cadenas con dos derivaciones a izquierda diferentes. Por ejemplo, para la cadena aab tenemos:

$$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aaA \Rightarrow aabA \Rightarrow aab.$$

$$S \Rightarrow aSA \Rightarrow aaSAA \Rightarrow aaAA \Rightarrow aabAA \Rightarrow aabA \Rightarrow aab.$$

Los árboles sintácticos de estas dos derivaciones son:

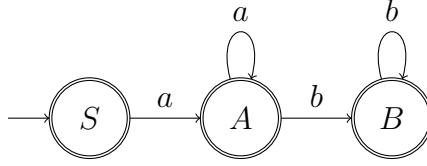


Por simple inspección, observamos que el lenguaje generado por esta gramática es $a^+b^*\cup\lambda$. Se puede construir una gramática no-ambigua que genere el mismo lenguaje:

$$G' : \begin{cases} S \rightarrow AB \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid \lambda \end{cases}$$

Para ver que la gramática G' no es ambigua se puede razonar de la siguiente manera: la cadena λ se puede generar de manera única con la derivación $S \Rightarrow \lambda$. Una derivación de una cadena no vacía debe comenzar aplicando la producción $S \rightarrow AB$; la variable A genera cadenas de a 'es de manera única (iterando la producción $A \rightarrow aA$), y B genera cadenas de b 'es también de manera única (iterando la producción $B \rightarrow bB$). Por consiguiente, toda cadena tiene una única derivación a izquierda.

Como el lenguaje $L = a^+b^* \cup \lambda$ es regular, podemos encontrar otra gramática G'' no ambigua equivalente a G a partir de un AFD M tal que $L(M) = L$.



La gramática regular inducida por M , según el procedimiento presentado en la sección 4.3 es G'' :

$$G'' : \begin{cases} S \rightarrow aA \mid \lambda \\ A \rightarrow aA \mid bB \mid \lambda \\ B \rightarrow bB \mid \lambda \end{cases}$$

Puesto que M es un autómata determinista, para cada cadena de L existe una única trayectoria de aceptación en M , y en consecuencia, cada cadena en $L(G'') = L$ se puede generar de manera única. Esto muestra que G'' no es ambigua.

Ejercicios de la sección 4.5

① Demostrar que las siguientes gramáticas son ambiguas:

- (i) $G : \{S \rightarrow aSb \mid aaSb \mid \lambda\}.$
- (ii) $G : \{S \rightarrow aSb \mid aSbb \mid \lambda\}.$
- (iii) $G : \{S \rightarrow aSb \mid abS \mid \lambda\}.$
- (iv) $G : \{S \rightarrow aaS \mid aaaS \mid \lambda\}$

② Para cada una de las siguientes gramáticas G , demostrar que G es ambigua, hallar $L(G)$ y encontrar una gramática no-ambigua que genere el mismo lenguaje $L(G)$.

- (i) $G : \begin{cases} S \rightarrow aaSB \mid b \mid \lambda \\ B \rightarrow Bb \mid \lambda \end{cases}$
- (ii) $G : \begin{cases} S \rightarrow ABA \mid b \mid \lambda \\ A \rightarrow aA \mid \lambda \\ B \rightarrow bB \mid \lambda \end{cases}$
- (iii) $G : \begin{cases} S \rightarrow AaSbB \mid \lambda \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid \lambda \end{cases}$
- (iv) $G : \begin{cases} S \rightarrow ASB \mid AB \\ A \rightarrow aA \mid a \\ B \rightarrow bB \mid \lambda \end{cases}$

4.6. Gramáticas para lenguajes de programación

Para presentar rigurosamente la sintaxis de los lenguajes de programación se utilizan con frecuencia gramáticas independientes del contexto en una notación denominada *Forma Backus-Naur* (FBN). En la FBN las variables de la gramática se representan mediante palabras descriptivas encerradas entre paréntesis angulares $\langle \rangle$, en la forma $\langle \text{Variable} \rangle$. Además, se utiliza el símbolo $::=$ para las producciones, en vez de la flecha \rightarrow .

Ejemplo A continuación presentamos la gramática G_3 de la sección 4.5, primero en la notación simple y luego en la FBN. Puesto que G_3 genera expresiones algebraicas, la variable inicial S se representa en la FBN como $\langle \text{Expresión} \rangle$. Por otro lado, las variables T y F se representan mediante $\langle \text{Término} \rangle$ y $\langle \text{Factor} \rangle$, respectivamente. En ambos casos, el alfabeto de terminales es $\Sigma = \{x, y, z, 0, 1, +, *, (,)\}$.

$$\text{Notación simple.} \quad G_3 : \begin{cases} S \rightarrow S + T \mid T \\ T \rightarrow T * F \mid F \\ F \rightarrow x \mid y \mid z \mid 0 \mid 1 \mid (S) \end{cases}$$

$$\text{Forma Backus-Naur.} \quad G_3 : \begin{cases} \langle \text{Expresión} \rangle ::= \langle \text{Expresión} \rangle + \langle \text{Término} \rangle \mid \langle \text{Término} \rangle \\ \langle \text{Término} \rangle ::= \langle \text{Término} \rangle * \langle \text{Factor} \rangle \mid \langle \text{Factor} \rangle \\ \langle \text{Factor} \rangle ::= x \mid y \mid z \mid 0 \mid 1 \mid (\langle \text{Expresión} \rangle) \end{cases}$$

Ejemplo En el recuadro de la siguiente página se presenta la gramática G_C que genera comandos para un lenguaje de programación genérico. G_C está escrita en la Forma Backus-Naur y su variable inicial es $\langle \text{Comando} \rangle$, por medio de la cual se pueden generar cuatro tipos de comandos (condicionales, asignaciones, comandos *while* y comandos *begin*). La sintaxis de los comandos condicionales utiliza las expresiones en negrilla **if**, **then** y **else**. Tales expresiones hacen parte del alfabeto de terminales de la gramática. También son terminales las expresiones en negrilla **while**, **do**, **begin** y **end**, así como los símbolos que no aparecen explícitamente encerrados entre paréntesis angulares $\langle \rangle$.

El símbolo de la igualdad tiene tres usos claramente diferenciados; $::=$ se utiliza para presentar las producciones, $:=$ se utiliza en los comandos de asignación y $=$ es uno de los operadores de comparación generados por la variable $\langle \text{Op. Comparación} \rangle$. Nótese también que la variable $\langle \text{Lista-Comandos} \rangle$ genera listas de comandos, separados por el símbolo ‘punto y coma’ (dicho símbolo es un terminal).

Gramática G_C

$\langle \text{Comando} \rangle ::= \langle \text{Condicional} \rangle \mid \langle \text{Comando While} \rangle \mid \langle \text{Asignación} \rangle \mid \langle \text{Comando Begin} \rangle$
 $\langle \text{Condicional} \rangle ::= \text{if } \langle \text{Expresión Booleana} \rangle \text{ then } \langle \text{Comando} \rangle \text{ else } \langle \text{Comando} \rangle$
 $\langle \text{Comando While} \rangle ::= \text{while } \langle \text{Expresión Booleana} \rangle \text{ do } \langle \text{Comando} \rangle$
 $\langle \text{Asignación} \rangle ::= \langle \text{Variable} \rangle := \langle \text{Expres. Aritmética} \rangle$
 $\langle \text{Comando Begin} \rangle ::= \text{begin } \langle \text{Lista-Comandos} \rangle \text{ end}$
 $\langle \text{Lista-Comandos} \rangle ::= \langle \text{Comando} \rangle \mid \langle \text{Comando} \rangle ; \langle \text{Lista-Comandos} \rangle$
 $\langle \text{Expresión Booleana} \rangle ::= \langle \text{Expres. Aritmética} \rangle \langle \text{Op. Comparación} \rangle \langle \text{Expres. Aritmética} \rangle$
 $\langle \text{Op. Comparación} \rangle ::= < \mid > \mid \leq \mid \geq \mid = \mid \neq$
 $\langle \text{Expres. Aritmética} \rangle ::= \langle \text{Variable} \rangle \mid \langle \text{Numeral} \rangle \mid$
 $\quad (\langle \text{Expres. Aritmética} \rangle \langle \text{Op. Aritmético} \rangle \langle \text{Expres. Aritmética} \rangle)$
 $\langle \text{Numeral} \rangle ::= \langle \text{Numeral} \rangle \langle \text{Numeral} \rangle \mid 0 \mid 1 \mid 2 \mid 3 \mid 4 \mid 5 \mid 6 \mid 7 \mid 8 \mid 9$
 $\langle \text{Op. Aritmético} \rangle ::= + \mid - \mid * \mid /$
 $\langle \text{Variable} \rangle ::= a \mid b \mid c \mid d \mid e \mid f \mid g \mid i \mid j \mid m \mid n \mid p \mid q \mid r \mid s \mid t \mid u \mid v \mid w \mid x \mid y \mid z$

Como ilustración, generamos a continuación el comando: **while** $x > 1$ **do** $y := (x + 2)$.

$\langle \text{Comando} \rangle \implies \langle \text{Comando While} \rangle \implies \text{while } \langle \text{Expresión Booleana} \rangle \text{ do } \langle \text{Comando} \rangle$
 $\xRightarrow{2} \text{while } \langle \text{Expres. Aritmética} \rangle \langle \text{Op. Comparación} \rangle \langle \text{Expres. Aritmética} \rangle \text{ do } \langle \text{Asignación} \rangle$
 $\xRightarrow{4} \text{while } \langle \text{Variable} \rangle > \langle \text{Numeral} \rangle \text{ do } \langle \text{Variable} \rangle := \langle \text{Expres. Aritmética} \rangle$
 $\xRightarrow{4} \text{while } x > 1 \text{ do } y := (\langle \text{Expres. Aritmética} \rangle \langle \text{Op. Aritmético} \rangle \langle \text{Expres. Aritmética} \rangle)$
 $\xRightarrow{3} \text{while } x > 1 \text{ do } y := (\langle \text{Variable} \rangle + \langle \text{Numeral} \rangle) \xRightarrow{2} \text{while } x > 1 \text{ do } y := (x + 2).$

Ejercicios de la sección 4.6

① Generar los siguientes comandos con la gramática G_C :

- (i) **if** $x > (y + 1)$ **then** $z := (x + 1)$ **else** $z := (x - 1)$.
- (ii) **begin if** $z = (y + 5)$ **then** $x := z$ **else** $z := (x + 1)$ **end**.
- (iii) **while** $x \leq y$ **do begin** $x := (x + 1); y := (y - 1)$ **end**.
- (iv) **if** $x > 1$ **then while** $y < 0$ **do** $z := (x + y)$ **else begin** $z := (x + 2); y := (x - 2)$ **end**.
- (v) **begin if** $y \neq 1$ **then while** $x \geq 0$ **do** $z := (x + y)$ **else begin** $u := x; v := y$ **end; z := (u/(v + y)) end**.

② La siguiente gramática G' es una modificación de la gramática G_C presentada en la presente sección, con dos tipos de comandos condicionales.

Gramática G'

```

⟨Comando⟩ ::= ⟨Comando Begin⟩ | ⟨Condicional⟩ | ⟨Condicional Alternativo⟩ | ⟨Asignación⟩
⟨Comando Begin⟩ ::= begin ⟨Lista-Comandos⟩ end
⟨Lista-Comandos⟩ ::= ⟨Comando⟩ | ⟨Comando⟩;⟨Lista-Comandos⟩
⟨Condicional⟩ ::= if T then ⟨Comando⟩
⟨Condicional Alternativo⟩ ::= if T then ⟨Comando⟩ else ⟨Comando⟩
⟨Asignación⟩ ::=  $x :=$  ⟨Numeral⟩
⟨Numeral⟩ ::= 0 | 1 | 2 | 3 | 4 | 5 | 6 | 7 | 8 | 9

```

En esta gramática el alfabeto Σ de terminales contiene todos los símbolos que no aparecen encerrados entre paréntesis angulares $\langle \rangle$; en particular, el símbolo $:=$ y el símbolo ‘punto y coma’. Explícitamente,

$$\Sigma = \{\mathbf{begin}, \mathbf{end}, \mathbf{if\ T\ then}, \mathbf{else}, x, :=, ;, 0, 1, 2, 3, 4, 5, 6, 7, 8, 9\}.$$

- (i) Demostrar que la gramática G' es ambigua.
- (ii) Encontrar una gramática no-ambigua que genere el mismo lenguaje generado por G' .

Capítulo 5

Autómatas con pila

En el presente capítulo presentamos el modelo de autómata con pila; distinguiremos las versiones determinista y no-determinista, pero, a diferencia de lo que sucede con los modelos AFD y AFN, los autómatas con pila deterministas y no-deterministas no resultan ser computacionalmente equivalentes.

5.1. Definición de los autómatas con pila

5.1.1. Autómatas con Pila Deterministas (AFPD)

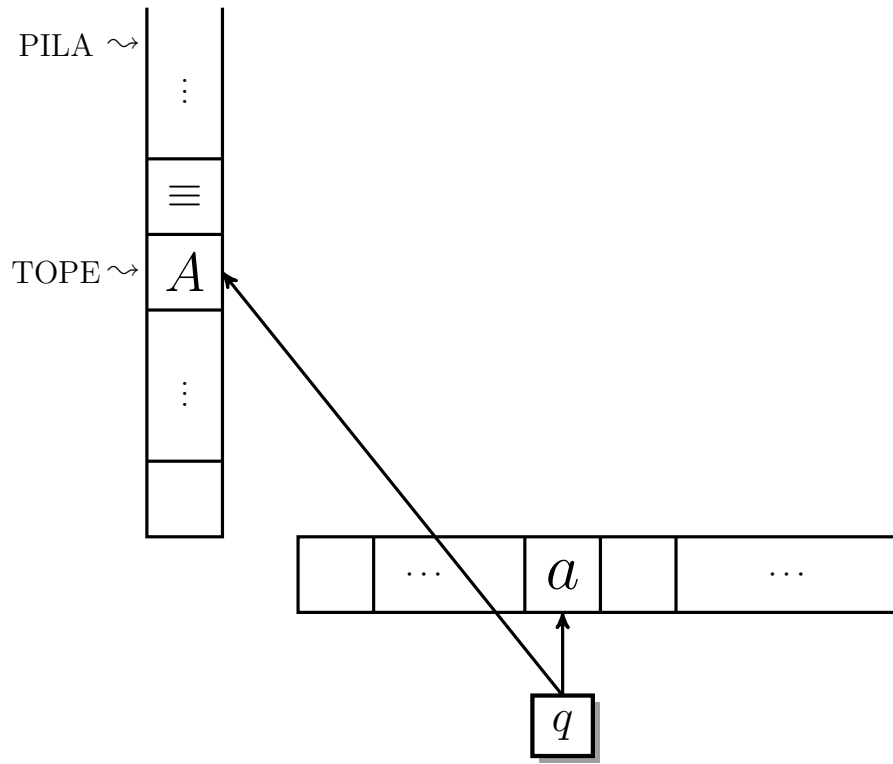
Un *Autómata Finito con Pila Determinista* (AFPD) es una séxtupla, $M = (Q, q_0, F, \Sigma, \Gamma, \Delta)$, con los siguientes componentes:

1. Q es el conjunto (finito) de estados internos de la unidad de control.
2. $q_0 \in Q$ es el estado inicial.
3. F es el conjunto de estados finales o de aceptación, $\emptyset \neq F \subseteq Q$.
4. Σ es el alfabeto de entrada, también llamado alfabeto de cinta.
5. Γ es el alfabeto de pila.
6. Δ es la función de transición del autómata:

$$\Delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \longrightarrow (Q \times (\Gamma \cup \{\lambda\})).$$

Como en los modelos de autómatas básicos considerados en Capítulo 2, un autómata con pila lee cadenas sobre una cinta de entrada semi-infinita; estas entradas están escritas con el alfabeto Σ . Pero hay una cinta adicional, llamada pila, también semi-infinita, que es utilizada por el autómata como lugar de almacenamiento o memoria temporal. Los símbolos que el autómata puede colocar en la pila pertenecen al alfabeto Γ . Los alfabetos Σ y Γ pueden tener símbolos comunes, es decir, no son necesariamente disyuntos. Inicialmente, la pila está vacía y la unidad de control está escaneando el fondo de la pila.

En un momento determinado, la unidad de control del autómata está en el estado q escaneando un símbolo a sobre la cinta de entrada, y el símbolo A en el tope o cima de la pila, como lo muestra la siguiente gráfica:



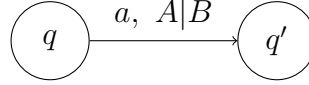
En todo momento la unidad de control solo tiene acceso al símbolo colocado en el tope de la pila (esa es la razón por la cual la pila se dibuja verticalmente). Al leer el símbolo a , la unidad de control se desplaza una casilla a la derecha, cambia al estado q' (que puede ser el mismo q) y realiza sobre la pila una de las siguientes cuatro acciones: o reemplaza el tope de la pila por otro símbolo, o añade un nuevo símbolo a la pila, o borra el tope de la pila, o no altera la pila. Las instrucciones permitidas están definidas en términos de la función de transición

$$\Delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \longrightarrow (Q \times (\Gamma \cup \{\lambda\}))$$

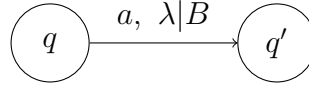
$$\Delta(q, a, A) = (q', B).$$

Hay que tener en cuenta los casos $a \in \Sigma$, $a = \lambda$, $A, B \in \Gamma$, $A = \lambda$ o $B = \lambda$, que se detallan a continuación.

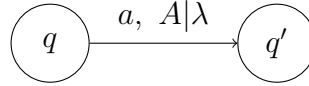
- (1) Caso $a \in \Sigma$; $A, B \in \Gamma$. Instrucción $\Delta(q, a, A) = (q', B)$. El tope de la pila A se reemplaza por B . En otras palabras, el símbolo B sobre-escribe a A . En el grafo del autómata, esta instrucción la representamos como



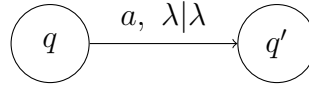
- (2) Caso $a \in \Sigma$, $A = \lambda$, $B \in \Gamma$. Instrucción $\Delta(q, a, \lambda) = (q', B)$. Independientemente del tope actual de la pila, se inserta o añade un nuevo símbolo B a la pila. B pasa a ser el nuevo tope de la pila. En el grafo del autómata, esta instrucción la representamos como



- (3) Caso $a \in \Sigma$, $A \in \Gamma$, $B = \lambda$. Instrucción $\Delta(q, a, A) = (q', \lambda)$. Se borra el tope de la pila A . La unidad de control pasa a escanear el símbolo ubicado en la casilla inmediatamente debajo, que es el nuevo tope de la pila. Si la A está inicialmente colocada en el fondo de la pila, entonces la pila se vacía y la unidad de control queda escaneando el fondo vacío. En el grafo del autómata, esta instrucción la representamos como



- (4) Caso $a \in \Sigma$, $A = B = \lambda$. Instrucción $\Delta(q, a, \lambda) = (q', \lambda)$. El contenido de la pila no se altera. En el grafo del autómata, esta instrucción la representamos como



En las instrucciones (1) a (4) el símbolo a se consume, pero también se permiten transiciones λ o transiciones espontáneas, que se ejecutan independientemente del símbolo escaneado en la cinta de entrada. Con las transiciones λ , la unidad de control no se desplaza a la derecha. Las siguientes son las transiciones λ permitidas; las acciones realizadas por el autómata sobre la pila son las mismas que con las instrucciones homólogas (1) a (4):

$$(1') \quad \Delta(q, \lambda, A) = (q', B).$$

$$(2') \quad \Delta(q, \lambda, \lambda) = (q', B).$$

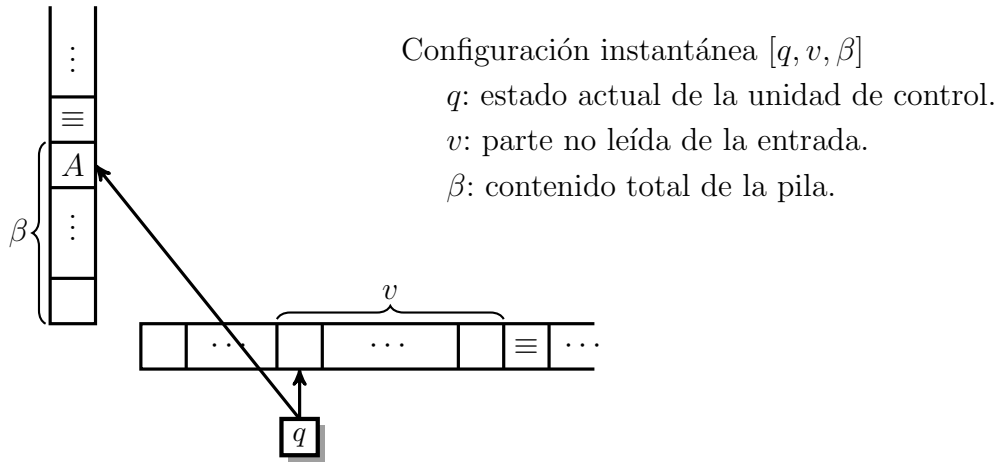
$$(3') \Delta(q, \lambda, A) = (q', \lambda).$$

$$(4') \Delta(q, \lambda, \lambda) = (q', \lambda).$$

Las transiciones λ o espontáneas permiten que el autómata cambie el contenido de la pila sin consumir símbolos sobre la cinta de entrada.

Es importante aclarar que la función Δ puede estar parcialmente definida; es decir, $\Delta(q, a, A)$ puede no estar definida, para algunos valores $q \in Q$, $a \in (\Sigma \cup \{\lambda\})$, $A \in (\Gamma \cup \{\lambda\})$. Se infiere que la lectura de algunas cadenas de entrada puede abortarse o detenerse sin que las entradas se lean completamente.

Configuración o descripción instantánea. Es una tripla $[q, v, \beta]$ que representa lo siguiente: la unidad de control está en el estado q , v es la parte no procesada de la cadena de entrada (la unidad de control está escaneando el primer símbolo de v), y la cadena β es el contenido total de la pila, tal como se exhibe en la siguiente gráfica:



Se adopta la convención de que la cadena β se lee de arriba hacia abajo. Así, en la gráfica anterior el primer símbolo de β es A , que es el tope de la pila.

La notación

$$[q, v, \beta] \vdash [q', w, \gamma].$$

representa un *paso computacional*, es decir, el autómata pasa de la configuración instantánea $[q, v, \beta]$ a la configuración $[q', w, \gamma]$ al ejecutar una de las instrucciones definidas por la función de transición Δ . Similarmente, la notación

$$[q, v, \beta] \vdash^* [p, w, \gamma]$$

significa que el autómata pasa de la configuración instantánea $[q, v, \beta]$ a la configuración instantánea $[p, w, \gamma]$ en uno o más pasos computacionales. También utilizaremos la notación $[q, v, \beta] \vdash^k [p, w, \gamma]$ para indicar que el autómata pasa de la configuración $[q, v, \beta]$ a la configuración $[p, w, \gamma]$ en k pasos.

En el modelo de autómatas con pila determinista (AFPD) se debe cumplir la siguiente restricción adicional: dada cualquier configuración instantánea $[q, v, \beta]$ solamente hay una instrucción posible (a lo sumo) que se puede aplicar. Para garantizar el determinismo, si la instrucción $\Delta(q, a, A)$ está definida (con $a \in \Sigma$ y $A \in \Gamma$), entonces ni $\Delta(q, \lambda, A)$ ni $\Delta(q, a, \lambda)$ pueden estar definidas simultáneamente; de lo contrario, en una configuración instantánea de la forma $[p, av, A\gamma]$ el autómata tendría varias opciones de proseguir el procesamiento de la entrada. En el modelo determinista se permiten transiciones λ siempre y cuando satisfagan la restricción anterior. Esto implica que un AFPD lee cada cadena de entrada $u \in \Sigma^*$ de manera única, aunque es posible que el procesamiento de u se aborte (o se detenga) sin consumir toda la entrada.

Configuración inicial y configuración de aceptación. Hay dos casos particulares importantes de configuraciones instantáneas. En primer lugar, la *configuración inicial* es $[q_0, u, \lambda]$ para una cadena de entrada $u \in \Sigma^*$. Al comenzar a leer una entrada, la pila está vacía y la unidad de control está escaneando el fondo de la pila.

La configuración $[p, \lambda, \lambda]$, siendo p un estado final o de aceptación, se llama *configuración de aceptación*. Esto significa que, para ser aceptada, una cadena de entrada debe ser leída completamente, la pila debe estar vacía y la unidad de control en un estado de aceptación.

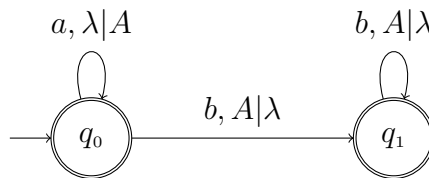
Lenguaje aceptado por un AFPD. El lenguaje aceptado por un AFPD M se define como

$$L(M) := \{u \in \Sigma^* : [q_0, u, \lambda] \vdash^* [p, \lambda, \lambda], p \in F\}.$$

O sea, una cadena u es aceptada si el único procesamiento posible de u desde la configuración inicial $[q_0, u, \lambda]$ termina en una configuración de aceptación. Como caso particular se deduce que si el estado inicial q_0 es un estado de aceptación, la cadena vacía λ es aceptada, ya que $[q_0, \lambda, \lambda]$ sería una configuración tanto inicial como de aceptación.

Ejemplo Diseñar un AFPD M que acepte el lenguaje $L = \{a^n b^n : n \geq 0\}$, sobre el alfabeto $\Sigma = \{a, b\}$. Recordemos que L no es regular y no puede ser aceptado por ningún autómata básico (sin pila).

Solución. La idea es almacenar las a s en la pila y borrar luego una a del tope de la pila por cada b que sea leída sobre la cinta. Una cadena será aceptada si es leída completamente y la pila se vacía. Para distinguir los símbolos de la cinta de entrada de los que hay en la pila, cada a se apilará como A ; es decir, el alfabeto de pila es $\Gamma = \{A\}$. El grafo de M es:



Podemos ilustrar el procesamiento de varias cadenas de entrada $u \in \Sigma^*$. Sea, inicialmente, $u = aaabbb$.

$$\begin{aligned} [q_0, aaabbb, \lambda] &\vdash [q_0, aabbb, A] \vdash [q_0, abbb, AA] \vdash [q_0, bbb, AAA] \vdash [q_1, bb, AA] \\ &\vdash [q_1, b, A] \vdash [q_1, \lambda, \lambda]. \end{aligned}$$

La última es una configuración de aceptación; por lo tanto la cadena $u = aaabbb$ es aceptada.

Si una cadena tiene más *bes* que *aes*, el autómata no la lee completamente. Por ejemplo, para la cadena de entrada $u = aabbb$, se obtiene el siguiente procesamiento:

$$\begin{aligned} [q_0, aabbb, \lambda] &\vdash [q_0, abbb, A] \vdash [q_0, bbb, AA] \vdash [q_1, bb, A] \\ &\vdash [q_1, b, \lambda] \quad (\text{Procesamiento abortado}). \end{aligned}$$

Obsérvese que el autómata termina en el estado de aceptación q_1 , con la pila vacía, pero la cadena de entrada no es aceptada debido a que no se ha leído completamente; $[q_1, b, \lambda]$ no es una configuración de aceptación.

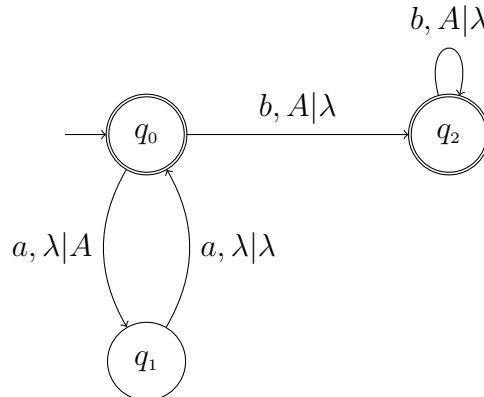
Si una cadena tiene más *aes* que *bes*, la pila no se vacía. Por ejemplo, para la cadena de entrada $u = aaabb$, se tiene:

$$[q_0, aaabb, \lambda] \vdash^3 [q_0, bb, AAA] \vdash [q_1, b, AA] \vdash [q_1, \lambda, A].$$

A pesar de que la cadena de entrada w ha sido leída completamente y el procesamiento termina en el estado de aceptación q_1 , la configuración $[q_1, \lambda, A]$ no es de aceptación. Por lo tanto, $u = aaabb$ no es aceptada.

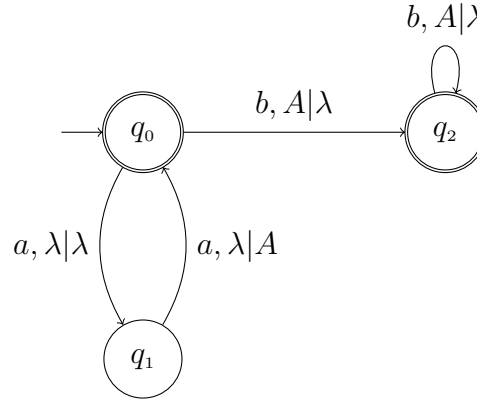
Ejemplo Diseñar un AFPD M que acepte el lenguaje $L = \{a^{2n}b^n : n \geq 0\}$, sobre el alfabeto $\Sigma = \{a, b\}$. L no es un regular y no puede ser aceptado por ningún autómata básico (sin pila).

Solución. La idea es apilar una A por cada dos *aes* leídas en la cinta y luego borrar una A de la pila por cada *b* que sea leída sobre la cinta. El alfabeto de pila es $\Gamma = \{A\}$ y el grafo de M es:



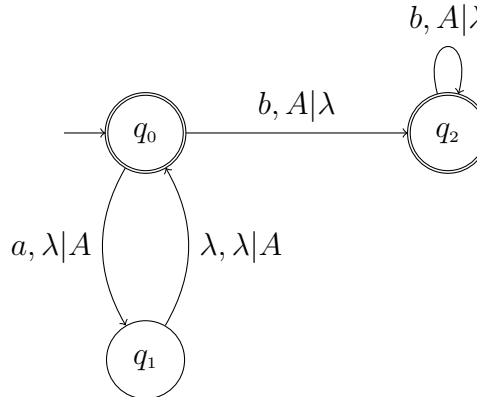
Obsérvese que las cadenas aceptadas por M tienen un número par de a 's. Además, si una entrada solamente tiene a 's, no es aceptada porque la pila no se vacía.

Las dos transiciones que unen a q_0 con q_1 se pueden invertir para aceptar el mismo lenguaje:



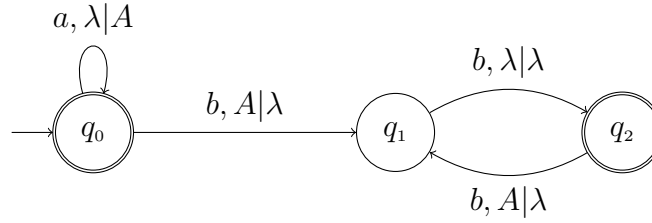
Ejemplo Diseñar un AFPD M que acepte el lenguaje $L = \{a^n b^{2n} : n \geq 0\}$, sobre el alfabeto $\Sigma = \{a, b\}$. L no es un regular y no puede ser aceptado por ningún autómata básico (sin pila).

Solución 1. Se apilan dos A 's por cada a leída en la cinta, y luego se borra una A de la pila por cada b que sea leída sobre la cinta. El alfabeto de pila es $\Gamma = \{A\}$ y el grafo de M es:



Se utiliza una transición λ de q_1 a q_0 para almacenar una A en la pila sin consumir ningún símbolo en la cinta de entrada. Nótese además que, cuando aparece la primera b el número de A 's en la pila es necesariamente par. Es importante resaltar que este autómata es determinista (AFPD), a pesar de la transición λ que hay entre q_1 y q_0 .

Solución 2. Se almacenan todas las a 's en la pila y luego se borra una A de la pila por cada dos b 's leídas en la cinta de entrada. El alfabeto de pila es $\Gamma = \{A\}$ y el grafo de M es:



Obsérvese que una cadena aceptada por M necesariamente tiene un número par de *bes*, y tal número es exactamente el doble del número de *aes*.

Nota. En los tres ejemplos anteriores utilizamos $\Gamma = \{A\}$ como alfabeto de pila pero podemos perfectamente hacer $\Gamma = \{a\}$ y apilar *aes* en vez de *Aes*.

5.1.2. Autómatas con Pila No-Deterministas (AFPND)

Un *Autómata Finito con Pila No-Determinista* (AFPND) consta de los mismos seis componentes de un AFPD, $M = (Q, q_0, F, \Sigma, \Gamma, \Delta)$, pero la función de transición Δ está definida como:

$$\Delta : Q \times (\Sigma \cup \{\lambda\}) \times (\Gamma \cup \{\lambda\}) \longrightarrow \wp(Q \times (\Gamma \cup \{\lambda\})),$$

donde $\wp(Q \times (\Gamma \cup \{\lambda\}))$ es el conjunto de subconjuntos de $Q \times (\Gamma \cup \{\lambda\})$. Para $q \in Q$, $a \in \Sigma \cup \{\lambda\}$ y $A \in (\Gamma \cup \{\lambda\})$, $\Delta(q, a, A)$ es de la forma

$$\Delta(q, a, A) = \{(p_1, B_1), (p_2, B_2), \dots, (p_k, B_k)\},$$

donde $p_1, p_2, \dots, p_k \in Q$, y $B_1, B_2, \dots, B_k \in (\Gamma \cup \{\lambda\})$. El significado de esta transición es: cuando la unidad de control escanea el símbolo a sobre la cinta de entrada y el símbolo A en el tope de la pila, la unidad de control puede ejecutar (aleatoriamente) una de las instrucciones (p_k, B_k) ($1 \leq i \leq k$). A diferencia de lo que sucede con los AFPD, en el modelo AFPND las instrucciones $\Delta(q, a, A)$ y $\Delta(q, a, \lambda)$ pueden estar definidas simultáneamente, y las transiciones λ , $\Delta(q, \lambda, A)$, no tienen restricción alguna. De esta manera, una cadena de entrada puede tener muchos procesamientos diferentes. También se permite que $\Delta(q, a, A) = \emptyset$, lo que da lugar a procesamientos abortados que no consumen completamente la entrada.

El lenguaje aceptado por un AFPND M se define como:

$$L(M) := \{u \in \Sigma^* : \text{existe un procesamiento } [q_0, u, \lambda] \vdash^* [p, \lambda, \lambda], p \in F\}.$$

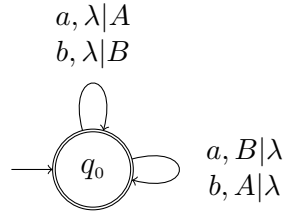
O sea, una cadena u es aceptada si existe por lo menos un procesamiento de u desde la configuración inicial $[q_0, u, \lambda]$ hasta una configuración de aceptación. Para ser aceptada, la entrada u debe ser leída completamente, el procesamiento debe terminar con la pila vacía y la unidad de control en un estado de aceptación.

Ejemplo

Diseñar un Autómata Finito con Pila No-Determinista (AFPND) que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen

igual número de *aes* que de *bes*. Es decir, $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u)\}$.

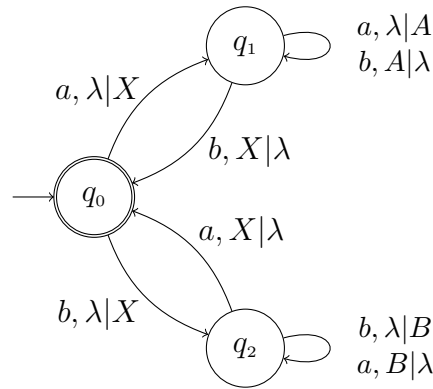
Solución. Para resolver este problema utilizamos el alfabeto de pila $\Gamma = \{A, B\}$. La idea es almacenar en la pila *aes* consecutivas o *bes* consecutivas. Si en el tope de la pila hay una *A* y el autómata lee una *b*, se borra la *A*; similarmente, si en el tope de la pila hay una *B* y el autómata lee una *a*, se borra la *B*. De esta manera, cada *a* de la entrada se empareja con una *b*, y viceversa. Para implementar esta idea solamente se requiere un estado:



El autómata M es no-determinista porque en un momento determinado las dos instrucciones $a, \lambda|A$ y $a, B|\lambda$ le otorgan a M dos opciones al leer una *a* en la cinta de entrada: almacenar *A* o borrar el tope de la pila *B*. Algo similar sucede con las dos instrucciones $b, \lambda|B$ y $b, A|\lambda$. De todas maneras, si una cadena tiene igual número de *aes* que de *bes*, existe un procesamiento que consume toda entrada y desocupa la pila. Por otro lado, si una entrada u tiene un número diferente de *aes* que de *bes*, entonces cualquier procesamiento completo de u , consume la entrada sin vaciar la pila.

Ejemplo Diseñar un Autómata Finito con Pila Determinista (AFPD) que acepte el lenguaje L de todas las cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que tienen igual número de *aes* que de *bes*. Es decir, $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u)\}$.

Solución. En el ejemplo anterior se presentó un AFPN que acepta a L , pero es también posible diseñar un AFPD. Vamos a utilizar un recurso que es útil en muchas situaciones: colocar un marcador de fondo X en la pila al iniciar el procesamiento de una entrada; de esta forma el autómata sabrá, al vaciar la pila en pasos subsiguientes, cuándo se ha llegado al fondo. Para el presente problema el alfabeto de pila es $\Gamma = \{X, A, B\}$; al consumir el primer símbolo de la entrada, ya sea *a* o *b*, el autómata coloca X en el fondo.



A continuación ilustramos el funcionamiento de M procesando la entrada $aababbbba$, la cual debe ser aceptada porque tiene cuatro a s y cuatro b s.

$$\begin{aligned} [q_0, aababbbba, \lambda] &\vdash [q_1, ababbbba, X] \vdash [q_1, babbbba, AX] \vdash [q_1, abbbba, X] \vdash [q_1, bbba, AX] \\ &\vdash [q_1, bba, X] \vdash [q_0, ba, \lambda] \vdash [q_2, a, X] \vdash [q_0, \lambda, \lambda]. \end{aligned}$$

Ejercicios de la sección 5.1

- ① Diseñar un AFPD que acepte el lenguaje $L = \{a^m b^n : n > m \geq 1\}$, sobre el alfabeto $\Sigma = \{a, b\}$. Utilizando la notación de configuración (o descripción) instantánea, procesar paso a paso las cadenas $a^3 b^4$ (aceptada), ab^5 (aceptada), $a^4 b^3$ (rechazada) y $a^3 b^3$ (rechazada).
- ② Diseñar un AFPN que acepte el lenguaje $L = \{a^m b^n : m > n \geq 0\}$, sobre el alfabeto $\Sigma = \{a, b\}$. Utilizando la notación de configuración (o descripción) instantánea, presentar procesamientos de aceptación para las entradas $a^4 b^3$ y $a^5 b$. Explicar por qué las cadenas $a^3 b^4$ y $a^3 b^3$ son rechazadas.
NOTA: Con las técnicas de la sección 5.3 se puede demostrar que el lenguaje L no puede ser aceptado por ningún AFPD.
- ③ Diseñar AFPD que acepten los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b\}$.
 - (i) $\{a^n b^{n+1} a : n \geq 1\}$.
 - (ii) $\{a^n b^m a^{n+1} : n \geq 0, m \geq 1\}$.
 - (iii) $\{a^n b^k a^m : k \geq 1, m > n \geq 1\}$.
- ④ Diseñar autómatas con pila, deterministas o no-deterministas, que acepten los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b, c\}$.
 - (i) $\{a^k b^m c^n : k, m, n \geq 0, k + m = n\}$.
 - (ii) $\{a^k b^m c^n : k, m, n \geq 0, k + n = m\}$.
 - (iii) $\{a^k b^m c^n : k, m, n \geq 0, m + n = k\}$.
 - (iv) $\{a^k b^m c^n : k, m, n \geq 0, 2n = k + m\}$.
 - (v) $\{a^k b^m c^n : k, m \geq 0, n \geq 1, n > k + m\}$.
 - (vi) $\{a^k b^m c^n : k, m \geq 1, n \geq 0, n < k + m\}$.
- ⑤ Sea $\Sigma = \{a, b\}$. Diseñar autómatas con pila, deterministas o no-deterministas, que acepten los siguientes lenguajes sobre el alfabeto $\Sigma = \{a, b\}$.
 - (i) $\{a^m b^n : m, n \geq 0, m \neq n\}$.
 - (ii) $\{u \in \Sigma^* : \#_a(u) > \#_b(u)\}$.

- (iii) $\{u \in \Sigma^* : \#_a(u) \geq \#_b(u)\}.$
- (iv) $\{u \in \Sigma^* : \#_a(u) \neq \#_b(u)\}.$
- (v) $\{u \in \Sigma^* : \#_a(u) = 2\#_b(u)\}.$

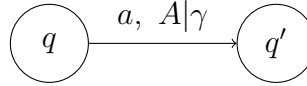
5.2. Inserción de cadenas en la pila

En los autómatas con pila la instrucción básica es $\Delta(q, a, A) = (p, B)$, que incluye varios casos: $a \in \Sigma$, $a = \lambda$, $A, B \in \Gamma$, $A = \lambda$ o $B = \lambda$. Al ejecutar una instrucción básica, el autómata solo puede añadir (a lo sumo) un símbolo a la pila. No obstante, las instrucciones por medio de las cuales se insertan en la pila cadenas de longitud arbitraria se pueden simular con las transiciones básicas y, por consiguiente, se pueden permitir en el diseño de autómatas. Esto se precisa en la siguiente proposición.

5.2.1 Proposición. En los autómatas con pila se permiten instrucciones de los siguientes tipos:

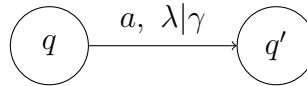
1. $\Delta(q, a, A) = (q', \gamma)$. Instrucción por medio de la cual el tope de la pila A se sustituye por la cadena $\gamma \in \Gamma^*$, o sea, γ sobre-escribe a A . El nuevo tope de la pila pasa a ser el primer símbolo de γ . Aquí se incluyen los casos $a \in \Sigma$ (se consume a en la cinta y) y $a = \lambda$ (transición λ).

$$\Delta(q, a, A) = (q', \gamma)$$



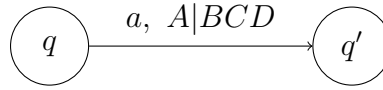
2. $\Delta(q, a, \lambda) = (q', \gamma)$. Instrucción por medio de la cual se inserta en pila la cadena $\gamma \in \Gamma^*$, independientemente del tope actual de la pila (este último se mantiene en la pila). El nuevo tope de la pila pasa a ser el primer símbolo de γ . Aquí se incluyen los casos $a \in \Sigma$ (se consume a en la cinta y) y $a = \lambda$ (transición λ).

$$\Delta(q, a, \lambda) = (q', \gamma)$$

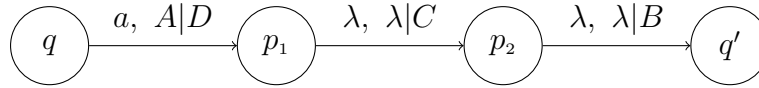


Bosquejo de la demostración. Cada una de las instrucciones $\Delta(q, a, A) = (q', \gamma)$ y $\Delta(q, a, \lambda) = (q', \gamma)$ se puede simular (añadiendo estados auxiliares) por medio de una secuencia de k transiciones básicas, donde k es la longitud de γ . \square

Ejemplo La instrucción $\Delta(q, a, A) = (q', BCD)$ consume a en cinta y reemplaza (o sustituye) el tope de la pila A por la cadena de tres símbolos BCD ; el nuevo tope de la pila pasa a ser B :

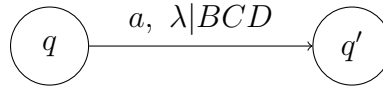


Esta instrucción se puede simular con una secuencia de tres transiciones básicas, como se muestra a continuación:

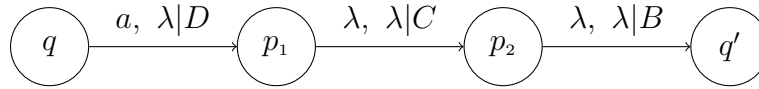


La secuencia es: primero sobre-escribir A por D , luego añadir C a la pila y luego añadir B . En la primera transición se consume la a , y las demás son transiciones λ . Los estados p_1 y p_2 son estados auxiliares (nuevos) que se introducen con el único propósito de hacer la simulación anterior.

La instrucción $\Delta(q, a, \lambda) = (q', BCD)$ consume a en cinta e inserta la cadena de tres símbolos BCD encima del tope actual de la pila (este último se mantiene en la pila); el nuevo tope de la pila pasa a ser B :



Esta instrucción se puede simular con una secuencia de tres transiciones básicas, como se muestra a continuación:

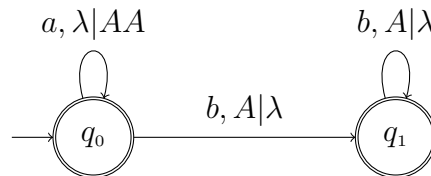


La secuencia es: primero añadir D encima del tope actual de la pila, luego añadir C y luego añadir B . En la primera transición se consume la a , y las demás son transiciones λ .

Las instrucciones de la Proposición 5.2.1 se pueden usar en ejercicios concretos de diseño de autómatas.

Ejemplo Diseñar un AFPD M que acepte el lenguaje $L = \{a^n b^{2n} : n \geq 0\}$, sobre el alfabeto $\Sigma = \{a, b\}$.

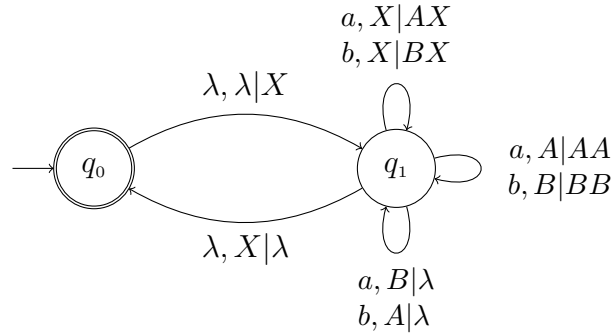
Solución. Se almacenan en la pila dos A s por cada a leída en la cinta, y luego se borra una A de la pila por cada b que sea leída sobre la cinta.



En la sección 5.1 se utilizó esta misma idea para resolver este problema, pero añadir dos Aes a la pila cada a leída en la cinta, usando transiciones básicas, requiere un estado auxiliar adicional.

Ejemplo En la sección 5.1.2 presentamos dos autómatas con pila, uno determinista y otro no-determinista, para aceptar el lenguaje $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u)\}$ formado por todas las cadenas que tienen igual número de a s que de b s. A continuación presentamos otro autómata para aceptar este mismo lenguaje. El alfabeto de pila es $\Gamma = \{A, B, X\}$; el símbolo X se utiliza como marcador de fondo.

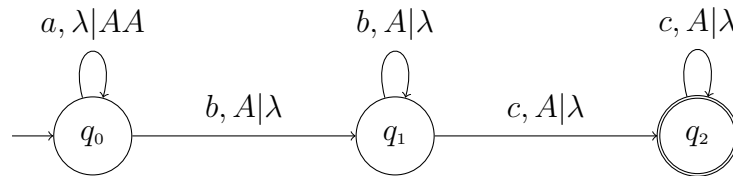
La instrucción $\Delta(q_1, \lambda, X) = (q_0, \lambda)$ es una transición λ que sirve para borrar el marcador de fondo y aceptar, pero introduce una opción no-determinista con respecto a los bucles $\Delta(q_1, a, X) = (q_1, AX)$ y $\Delta(q_1, b, X) = (q_1, BX)$. Por consiguiente, este es un AFPN.



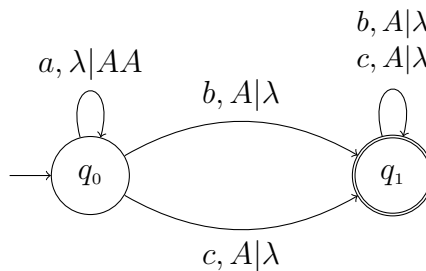
Ejercicios de la sección 5.2

- ① Sea $\Sigma = \{a, b, c\}$. Describir explícitamente los lenguajes aceptados por los siguientes autómatas con pila:

(i)



(ii)



② Diseñar autómatas con pila, deterministas o no-deterministas, que acepten los siguientes lenguajes.

- (i) $\{a^{n+1}b^{2n+1} : n \geq 0\}$ sobre el alfabeto $\Sigma = \{a, b\}$.
- (ii) $\{a^{2n}b^{3n} : n \geq 0\}$ sobre el alfabeto $\Sigma = \{a, b\}$.
- (iii) $\{a^{3n}b^{2n} : n \geq 0\}$ sobre el alfabeto $\Sigma = \{a, b\}$.
- (iv) $\{a^k b^m c^n : k, m, n \geq 0, n = 2k + m\}$ sobre el alfabeto $\Sigma = \{a, b, c\}$.
- (v) $\{a^k b^m c^n : k, m, n \geq 0, m = k + 2n\}$ sobre el alfabeto $\Sigma = \{a, b, c\}$.
- (vi) $\{a^k b^m c^n : k, m, n \geq 0, k = 2m + n\}$ sobre el alfabeto $\Sigma = \{a, b, c\}$.
- (vii) $\{a^k b^m c^n : k, m, n \geq 0, 2k = m + n\}$ sobre el alfabeto $\Sigma = \{a, b, c\}$.
- (viii) $\{a^m b^n : 0 \leq n \leq 2m\}$ sobre el alfabeto $\Sigma = \{a, b\}$.

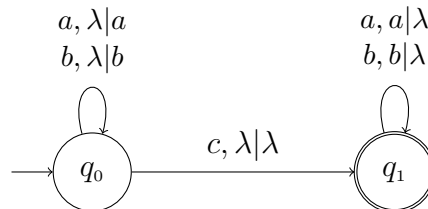
5.3. Los modelos AFPD y AFPN no son equivalentes

En contraste con lo que sucede con los modelos AFD y AFN, los modelos de autómata con pila determinista (AFPD) y no-determinista (AFPN) no resultan ser computacionalmente equivalentes: existen lenguajes aceptados por autómatas AFPN que no pueden ser aceptados por ningún AFPD. Un ejemplo concreto es el lenguaje $L = \{ww^R : w \in \Sigma^*\}$ sobre el alfabeto $\Sigma = \{a, b\}$. Las cadenas de la forma ww^R son palíndromes de longitud par; en efecto,

$$(ww^R)^R = (w^R)^R w^R = ww^R.$$

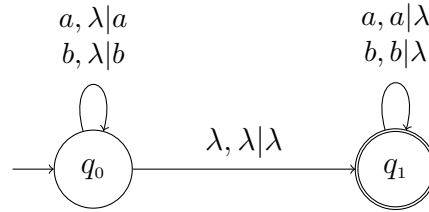
Como se mostrará en la presente sección, se puede construir un autómata con pila no-determinista para aceptar a L , pero no es posible diseñar ningún autómata con pila determinista que lo haga. La demostración de esta imposibilidad se presenta en el Teorema 5.3.1

Ejemplo Primero consideramos el lenguaje $L' = \{w c w^R : w \in \{a, b\}^*\}$ sobre el alfabeto $\Sigma = \{a, b, c\}$. Nótese que las cadenas w y w^R sólo poseen *aes* y/o *bes*. Es muy fácil diseñar un AFPD que acepte el lenguaje L' porque el símbolo c actúa como un separador entre las dos mitades w y w^R . Para aceptar una cadena de la forma $w c w^R$, la idea es almacenar en la pila los símbolos que preceden a la c central. El contenido de la pila, leído de arriba hacia abajo será precisamente w^R . El autómata consume la c (sin alterar la pila) y luego procesa el resto de la entrada borrando en cada paso el tope de la pila. El alfabeto de pila para este autómata es $\Gamma = \{a, b\}$ y su grafo se exhibe a continuación.



Ejemplo Diseñar un AFPN que acepte el lenguaje $L = \{ww^R : w \in \Sigma^*\}$, donde $\Sigma = \{a, b\}$. Como se señaló arriba, L es exactamente el lenguaje de los palíndromes de longitud par.

Solución. El lenguaje L del presente ejemplo es similar al lenguaje $\{wcw^R : w \in \{a, b\}^*\}$ del ejemplo anterior, excepto que ya no aparece el separador c entre w y w^R . El no-determinismo se utiliza para permitirle al autómata la opción de “adivinar” cuál es la mitad de la cadena de entrada. Si acierta, procederá a comparar el resto de la cadena de entrada con los símbolos almacenados en la pila. Si no acierta entonces o bien, no se consume toda la entrada, o bien, no se vacía la pila. Si la cadena de entrada tiene la forma deseada, entre todos los cómputos posibles estará aquél en el que el autómata adivina correctamente cuándo ha llegado a la mitad de la cadena.



La transición λ de q_0 a q_1 le permite al autómata una opción no-determinista: conjeturar que se ha llegado a la mitad de la cadena de entrada. En este último caso, la unidad de control pasa al estado q_1 y comienza a borrar los símbolos ya almacenados en la pila, que deben coincidir con los que se leen en la cinta.

5.3.1 Proposición. El lenguaje $L = \{ww^R : w \in \Sigma^*\}$, donde $\Sigma = \{a, b\}$, no puede ser aceptado por ningún AFPD (Autómata Finito con Pila Determinista).

Demostración. Razonamiento por contradicción. Se supone que existe un AFPD M tal que $L(M) = L$. Las potencias a^2, a^4, a^6, \dots son palíndromes de longitud par y, por tanto, son aceptadas por M . Pero como hay infinitas cadenas de la forma a^{2n} , con $n \geq 0$, y solamente hay un número finito de configuraciones de aceptación (ya que M tiene un número finito de estados), por el Principio del Palomar existen dos cadenas a^{2i} y a^{2j} , con $i \neq j$, cuyo procesamiento termina en la misma configuración de aceptación $[p, \lambda, \lambda]$, con $p \in F$. Esto es, $[q_0, a^{2i}, \lambda] \vdash^* [p, \lambda, \lambda]$ y $[q_0, a^{2j}, \lambda] \vdash^* [p, \lambda, \lambda]$. Como M es determinista, se sigue entonces que, para toda cadena $x \in \Sigma^*$, M leerá las cadenas $a^{2i}x$ y $a^{2j}x$ exactamente de la misma forma ya que $[q_0, a^{2i}x, \lambda] \vdash^* [p, x, \lambda]$ y $[q_0, a^{2j}x, \lambda] \vdash^* [p, x, \lambda]$. Esto implica que, para toda cadena $x \in \Sigma^*$, $a^{2i}x$ y $a^{2j}x$ son ambas aceptadas o ambas rechazadas por M ; en otras palabras,

$$(\forall x \in \Sigma^*)[(a^{2i}x \in L \text{ y } a^{2j}x \in L) \text{ ó } (a^{2i}x \notin L \text{ y } a^{2j}x \notin L)].$$

La contradicción surge porque existe una cadena $x \in \Sigma^*$ tal que $a^{2i}x$ es aceptada pero $a^{2j}x$ es rechazada. Por simple inspección observamos que tomando $x = bba^{2i}$ se tendrá $a^{2i}x = a^{2i}bba^{2i} \in L$ pero $a^{2j}x = a^{2j}bba^{2i} \notin L$ (porque $i \neq j$). De esta manera se llega a una contradicción. \square

Utilizando la noción de distinguibilidad entre cadenas, el argumento de la Proposición 5.3.1 se puede convertir en una técnica práctica para demostrar que ciertos lenguajes no pueden ser aceptados por autómatas con pila deterministas. Eso se hace en la siguiente proposición, muy similar al criterio de no regularidad (Teorema 3.1.1).

5.3.2 Proposición. Sea Σ un alfabeto dado y L un lenguaje sobre Σ . Si L contiene infinitas cadenas L -distinguibles dos a dos, entonces L no puede ser aceptado por ningún AFPD (Autómata Finito con Pila Determinista).

Demostración. Razonamiento por contradicción. Se supone que existe un AFPD M tal que $L(M) = L$. Por hipótesis existen en L infinitas cadenas u_1, u_2, u_3, \dots que son L -distinguibles dos a dos. Estas infinitas cadenas son aceptadas por M ; puesto que solamente hay un número finito de configuraciones de aceptación (ya que M tiene un número finito de estados), por el Principio del Palomar existen dos cadenas u_i y u_j , con $i \neq j$, cuyo procesamiento termina en la misma configuración de aceptación $[p, \lambda, \lambda]$, con $p \in F$. Esto es, $[q_0, u_i, \lambda] \vdash^* [p, \lambda, \lambda]$ y $[q_0, u_j, \lambda] \vdash^* [p, \lambda, \lambda]$. Como M es determinista, se sigue entonces que, para toda cadena $x \in \Sigma^*$, M leerá las cadenas $u_i x$ y $u_j x$ exactamente de la misma forma ya que $[q_0, u_i x, \lambda] \vdash^* [p, x, \lambda]$ y $[q_0, u_j x, \lambda] \vdash^* [p, x, \lambda]$. Esto implica que, para toda cadena $x \in \Sigma^*$, $u_i x$ y $u_j x$ son ambas aceptadas o ambas rechazadas por M ; en otras palabras,

$$(\forall x \in \Sigma^*)[(u_i x \in L \text{ y } u_j x \in L) \text{ ó } (u_i x \notin L \text{ y } u_j x \notin L)].$$

Esto contradice que u_i y u_j son L -distinguibles. \square

Ejemplo Utilizar la Proposición 5.3.2 para demostrar que el lenguaje L de todos los palíndromes de longitud par sobre $\Sigma = \{a, b\}$, o sea, $L = \{ww^R : w \in \Sigma^*\}$, no puede ser aceptado por ningún AFPD (Autómata Finito con Pila Determinista).

Solución. Las infinitas cadenas a^2, a^4, a^6, \dots son palíndromes de longitud par y, por tanto, pertenecen a L . Vamos a comprobar que son L -distinguibles dos a dos. Sean $i, j \geq 1$, con $i \neq j$. Queremos mostrar que a^{2i} y a^{2j} son L -distinguibles, para lo cual hay que encontrar una cadena x tal que $a^{2i}x \in L$ pero $a^{2j}x \notin L$ (o viceversa). La escogencia más sencilla de x es $x = bba^{2i}$. Se tiene que $a^{2i}x = a^{2i}bba^{2i} \in L$ pero $a^{2j}x = a^{2j}bba^{2i} \notin L$ (porque $i \neq j$). Esto muestra que a^{2i} y a^{2j} son L -distinguibles si $i \neq j$ y, por la Proposición 5.3.2, L no puede ser aceptado por ningún AFPD.

Ejercicios de la sección 5.3

- ① Sea $\Sigma = \{a, b\}$. Diseñar autómatas con pila no-deterministas que acepten los siguientes lenguajes.
 - (i) El lenguaje de todos los palíndromes.
 - (ii) El lenguaje de todos los palíndromes de longitud impar.
 - (iii) $\{a^m b^n : m > n \geq 0\}$.
 - (iv) $\{a^m b^n : m, n \geq 0, m \neq n\}$.

- (v) $\{u \in \Sigma^* : \#_a(u) > \#_b(u) \geq 0\}$.
- (vi) $a^* \cup \{a^n b^n : n \geq 0\}$.
- (vii) $\{a^n b^n : n \geq 0\} \cup \{a^n b^{2n} : n \geq 0\}$.
- (viii) $\{a^{2n} b^n : n \geq 0\} \cup \{a^n b^{2n} : n \geq 0\}$.

- ② Demostrar que ninguno de los lenguajes del problema ① puede ser aceptado por un AFPD (Autómata Finito con Pila Determinista).
- ③ Sea $\Sigma = \{a, b, c\}$. Diseñar un autómata con pila no-determinista que acepte el lenguaje $L = \{a^k b^m c^n : k, m, n \geq 0, k = m \text{ o } k = n\}$. Demostrar que L no puede ser aceptado por ningún AFPD (Autómata Finito con Pila Determinista).

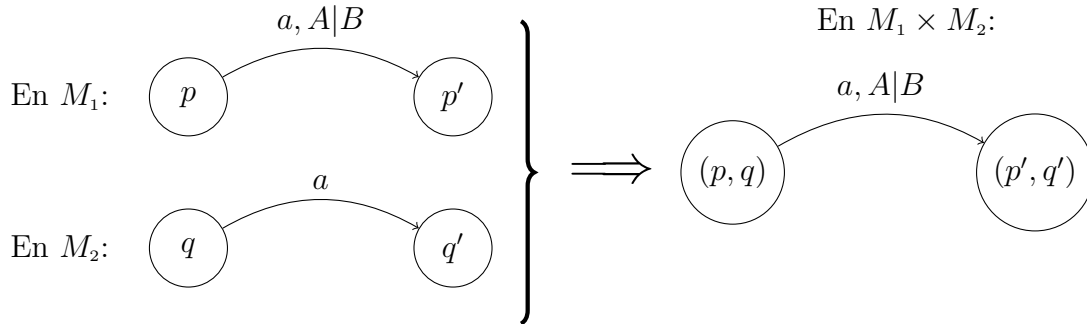
5.4. Producto cartesiano de un autómata con pila y un AFD

Sea $M_1 = (Q_1, q_1, F_1, \Sigma, \Gamma, \Delta)$ un autómata con pila, ya sea AFPD o AFPN, y sea $M_2 = (\Sigma, Q_2, q_2, F_2, \delta)$ un AFD. Los dos autómatas tiene el mismo alfabeto de entrada Σ . El Producto Cartesiano $M_1 \times M_2$ es un autómata con pila definido como

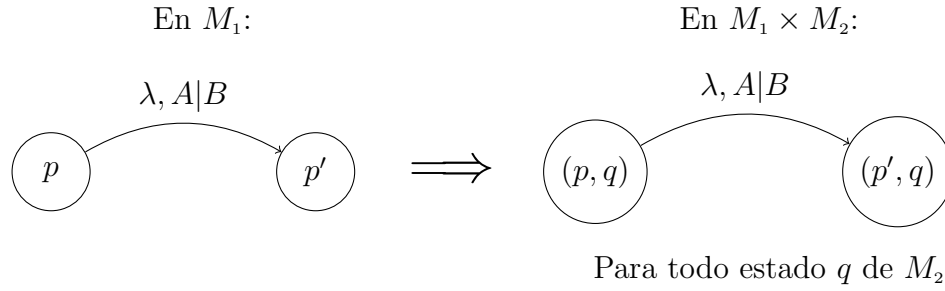
$$M_1 \times M_2 = (Q_1 \times Q_2, (q_1, q_2), F_1 \times F_2, \Sigma, \Gamma, \Delta')$$

donde la función de transición Δ' consta de las siguientes instrucciones:

- (1) Si $(p', B) \in \Delta(p, a, A)$ y $\delta(q, a) = q'$, entonces $((p', q'), B) \in \Delta'((p, q), a, A)$; aquí se tienen en cuenta todas las transiciones de M_1 , con $a \in \Sigma$, $A, B \in \Gamma \cup \{\lambda\}$. Considerando los grafos de los autómatas, las anteriores instrucciones se pueden presentar esquemáticamente de la siguiente forma:



- (2) Una transición λ de p a p' en M_1 da lugar a la misma transición λ de (p, q) a (p', q) en $M_1 \times M_2$, para todos los estados q de M_2 . De esta forma, una transición λ en M_1 origina k transiciones λ en $M_1 \times M_2$, donde k es el número de estados de M_2 . Esquemáticamente,



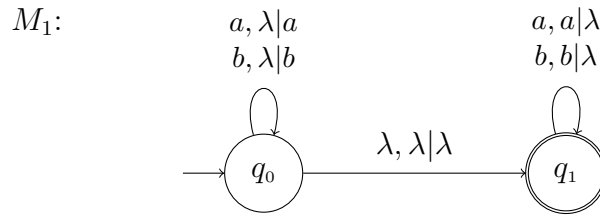
En términos de las funciones de transición, se tiene que si $(p', B) \in \Delta(p, \lambda, A)$ entonces $((p', q), B) \in \Delta'((p, q), \lambda, A)$ para todos los estados q de M_2 , y considerando $A, B \in \Gamma \cup \{\lambda\}$.

Al examinar detenidamente la anterior construcción, se observa que el producto cartesiano $M_1 \times M_2$ simula los autómatas M_1 y M_2 “en paralelo”: con las primeras componentes de $M_1 \times M_2$ se simula el funcionamiento de M_1 mientras que con las segundas componentes se simula a M_2 . Un símbolo de $a \in \Sigma$ se consume solamente si tanto M_1 como M_2 lo consumen simultáneamente, y un estado (p, q) es de aceptación si tanto p como q son estados de aceptación. Por este argumento, el siguiente teorema resulta intuitivamente claro.

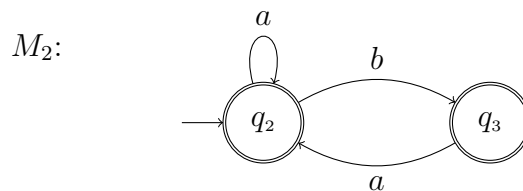
5.4.1 Teorema. Sea $M_1 = (Q_1, q_1, F_1, \Sigma, \Gamma, \Delta)$ un autómata con pila (AFPD o AFPN) tal que $L(M_1) = L_1$ y sea $M_2 = (\Sigma, Q_2, q_2, F_2, \delta)$ un AFD tal que $L(M_2) = L_2$. El autómata con pila $M_1 \times M_2$ definido arriba satisface $L(M_1 \times M_2) = L_1 \cap L_2$.

Ejemplo Utilizar el Teorema 5.4.1 para construir un autómata con pila que acepte el lenguaje L de todos los palíndromes de longitud par que no tienen dos *bes* consecutivas, sobre el alfabeto $\Sigma = \{a, b\}$.

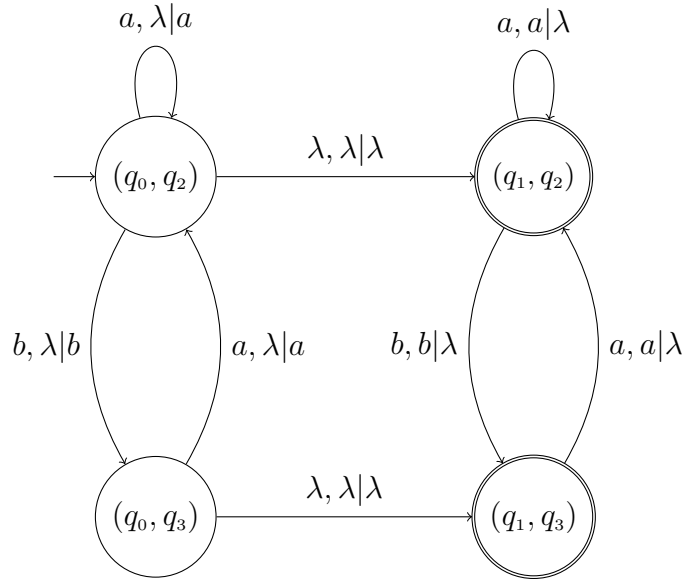
Solución. El lenguaje L se puede escribir como $L = L_1 \cap L_2$ donde L_1 es el lenguaje de todos los palíndromes de longitud par y L_2 es el lenguaje de las cadenas no tienen dos *bes* consecutivas. Para aceptar a L_1 tenemos el AFPN M_1 diseñado en la sección anterior:



Para aceptar a L_2 tenemos el siguiente AFD M_2 . Se ha omitido el estado limbo porque solamente se necesitan las trayectorias de aceptación si se desea aceptar $L_1 \cap L_2$.



Al formar el producto cartesiano $M_1 \times M_2$ se obtiene el siguiente AFPN:



La transición λ en M_1 da lugar a dos transiciones λ en $M_1 \times M_2$, manteniendo fijas las segundas componentes. Al examinar el autómata $M_1 \times M_2$ se observa que los estados de la izquierda se utilizan para almacenar símbolos en la pila pero impidiendo que aparezcan dos *bes* consecutivas. Los estados de la derecha se utilizan para desocupar la pila impidiendo también dos *bes* consecutivas.

Ejercicios de la sección 5.4

Utilizando la técnica del producto cartesiano de un AFPN con un AFD construir autómatas con pila que acepten los siguientes lenguajes, definidos sobre el alfabeto $\Sigma = \{0, 1\}$.

- ① El lenguaje de todos los palíndromes que no terminan en 01.
- ② El lenguaje de todos los palíndromes que no contienen la subcadena 101.
- ③ El lenguaje de todas las cadenas de longitud impar que tienen más ceros que unos.
- ④ $L = \{u \in \Sigma^* : \#_0(u) = \#_1(u) = \text{impar}\}$.
- ⑤ $L = \{u \in \Sigma^* : \#_0(u) > \#_1(u) \text{ y } \#_0(u) \text{ es impar}\}$.
- ⑥ $L = \{u \in \Sigma^* : \#_0(u) > \#_1(u) \text{ y } \#_1(u) \text{ es impar}\}$.

Capítulo 6

Máquinas de Turing

6.1. El modelo estándar de Máquina de Turing

Una *máquina de Turing* (MT), $M = (Q, q_0, F, \Sigma, \Gamma, \delta)$, consta de seis componentes:

1. Q es el conjunto (finito) de estados internos de la unidad de control.
2. $q_0 \in Q$ es el estado inicial.
3. F es el conjunto de estados finales o de aceptación, $\emptyset \neq F \subseteq Q$.
4. Σ es el alfabeto de entrada.
5. Γ es el alfabeto de cinta, que incluye a Σ , es decir, $\Sigma \subseteq \Gamma$. Los símbolos de Γ que no hacen parte de Σ (o sea, los símbolos en $\Gamma - \Sigma$) se utilizan como símbolos auxiliares.
6. δ es la función de transición de la máquina:

$$\delta : Q \times (\Gamma \cup \{\square\}) \longrightarrow Q \times (\Gamma \cup \{\square\}) \times \{\leftarrow, \rightarrow, -\}.$$

δ es una función parcial, es decir, puede no estar definida en algunos elementos del dominio. El símbolo \square no hace parte del alfabeto de cinta Γ sino que es un símbolo “externo” que representa una casilla en blanco. Por simplicidad, $\Gamma \cup \{\square\}$ se denotará como Γ_{\square} .

Una máquina de Turing M lee cadenas de entrada $u \in \Sigma^*$ colocadas sobre una cinta infinita en ambas direcciones. Una entrada u se coloca en una porción cualquiera de la cinta y M comienza a leerla escaneando el primer símbolo de u , con la unidad de control

en el estado inicial q_0 . Las demás celdas o casillas de la cinta están completamente en blanco.

La transición (o instrucción)

$$\delta(q, s) = (q', s', D),$$

donde $s, s' \in \Gamma$, significa: estando en el estado q , escaneando el símbolo s , la unidad de control sobre-escribe s por s' , cambia al estado q' , y realiza un desplazamiento D , que puede ser \rightarrow o \leftarrow o $-$. Los desplazamientos permitidos y la representación de las instrucciones $\delta(q, s) = (q', s', D)$ en el grafo de estados de la Máquina de Turing se exhiben en la siguiente tabla.

Instrucción	Acción de la unidad de control	Grafo
$\delta(q, s) = (q', s', \rightarrow)$	Se desplaza a la derecha	
$\delta(q, s) = (q', s', \leftarrow)$	Se desplaza a la izquierda	
$\delta(q, s) = (q', s', -)$	Permanece estacionaria	

Hay otros dos tipos de instrucciones permitidas que son: borrar el símbolo actualmente escaneado y escribir un símbolo en una casilla vacía que está siendo escaneada. Se detallan en la siguiente tabla.

Instrucción	Acción de la unidad de control	Grafo
$\delta(q, s) = (q', \square, D)$	Borra el símbolo s y la unidad de control realiza el desplazamiento D , que puede ser \rightarrow o \leftarrow o $-$.	
$\delta(q, \square) = (q', s, D)$	Escribe el símbolo s en la casilla vacía escaneada, y la unidad de control realiza el desplazamiento D , que puede ser \rightarrow o \leftarrow o $-$.	

Como se dijo arriba, \square no hace parte del alfabeto de cinta Γ sino que es un símbolo externo utilizado para presentar las instrucciones $\delta(q, s) = (q', \square, D)$ y $\delta(q, \square) = (q', s, D)$, cuyo significado se acaba de precisar.

En definitiva, toda instrucción básica en una MT es de la forma

$$\delta(q, s) = (q', s', D), \text{ donde } s, s' \in \Gamma \cup \square; \ q, q' \in Q \text{ y } D \in \{\rightarrow, \leftarrow, -\}.$$

Un detalle importante es que no es necesario definir explícitamente las transiciones λ para máquinas de Turing ya que una instrucción de la forma $\delta(q, s) = (q', s, -)$ permite

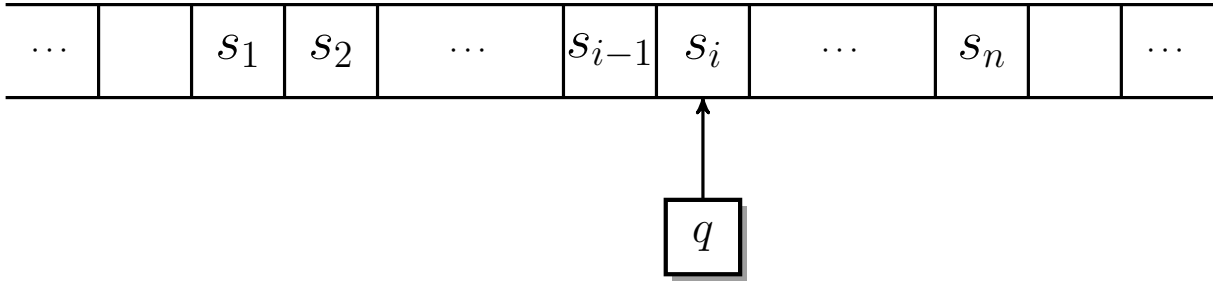
un cambio de estado en la unidad de control sin consumir ningún símbolo ni realizar ningún desplazamiento.

Tal como se ha definido, el modelo estándar de máquina de Turing es determinista y, por tanto, toda cadena de entrada tiene un único procesamiento.

Descripción o configuración instantánea. Es una secuencia de la forma

$$s_1 s_2 \cdots s_{i-1} q s_i \cdots s_n$$

donde los símbolos s_1, \dots, s_n pertenecen a Γ_\square y $q \in Q$. Indica que la unidad de control de M está en el estado q escaneando el símbolo s_i , y todas las casillas a la izquierda s_1 y a la derecha de s_n están enteramente en blanco, tal como se muestra en la siguiente gráfica:



Puesto que toda cadena de entrada es finita, después de que M haya ejecutado un número finito de instrucciones solamente hay una porción finita de la cinta que no está enteramente en blanco; tal porción finita queda representada por la configuración instantánea.

Ejemplos concretos de configuraciones instantáneas son:

$$\begin{aligned} & aabq_2bABa \\ & q_5ababca \\ & ab\square\square aabq_0BbA \\ & \square aX\square Ycq_3Bb\square \end{aligned}$$

Una configuración instantánea como $aX\square Ycq_3Bb$ también se puede escribir en la forma $\square aX\square Ycq_3Bb\square$, destacando los símbolos \square en los extremos.

En general, una configuración instantánea es una secuencia de símbolos de la forma uqv donde $u, v \in (\Gamma_\square)^*$ y $q \in Q$. Como caso particular, la configuración instantánea inicial, o simplemente *configuración inicial*, es q_0u , donde $u \in \Sigma^*$ es la cadena de entrada. Otro caso particular importante es el de *configuración de aceptación*, definida más adelante.

Paso computacional. El paso de una configuración instantánea a otra, por medio de una transición definida por δ , se denomina un *paso computacional* y se denota por

$$u_1qu_2 \vdash v_1pv_2,$$

donde $u_1, u_2, v_1, v_2 \in (\Gamma_\square)^*$ y $p, q \in Q$. Un ejemplo concreto es

$$abbaq_2ba \vdash\!\!\vdash abbq_1aca$$

en la cual la máquina ejecutó la instrucción $\delta(q_2, b) = (q_1, c, \leftarrow)$.

La notación

$$u_1qu_2 \vdash^* v_1pv_2$$

significa que M puede pasar de la configuración instantánea u_1qu_2 a la configuración instantánea v_1pv_2 en uno o más pasos computacionales. También se utiliza la notación $u_1qu_2 \vdash^k v_1pv_2$ para indicar que M pasa de la configuración u_1qu_2 a la configuración v_1pv_2 en exactamente k pasos.

Lenguaje aceptado por una MT. Para simplificar los argumentos sobre máquinas de Turing, en el modelo estándar se supone que la unidad de control siempre se detiene al ingresar a un estado de aceptación. Es decir, no se permiten transiciones $\delta(q, s)$ cuando $q \in F$. La noción de aceptación para máquinas de Turing es más flexible que para autómatas: una cadena de entrada no tiene que ser leída en su totalidad para que sea aceptada; sólo se requiere que la máquina ingrese a un estado de aceptación, y al hacerlo, la unidad de control se detiene inmediatamente. En definitiva, el lenguaje aceptado por una MT $M = (Q, q_0, F, \Sigma, \Gamma, \delta)$ se define como

$$L(M) := \{u \in \Sigma^* : q_0u \vdash^* vpw, p \in F, v, w \in (\Gamma_\square)^*\}.$$

Dicho explícitamente, una cadena de entrada u es aceptada por una MT M si el procesamiento que se inicia en la configuración inicial q_0u termina en una *configuración de aceptación*, es decir, en una configuración de la forma vpw , donde p estado de aceptación. Las cadenas $v, w \in (\Gamma_\square)^*$ que quedan escritas en la cinta cuando M ingresa al estado de aceptación p son irrelevantes.

Si $u \in L(M)$, al leer u , M se detiene en un estado de aceptación. Si $u \notin L(M)$, hay dos situaciones que se pueden presentar al leer u :

1. M se detiene en un estado que no es de aceptación; esto sucede cuando la función de transición $\delta(q, s)$ no está definida para cierta combinación $q \in Q, s \in \Gamma_\square$, donde $q \notin F$. Estos son los llamados procesamientos o cálculos abortados.
2. M no se detiene; esto es lo que se denomina un “bucle infinito” o un “ciclo infinito”. Esta situación se representa con la notación

$$q_0u \vdash^* \infty$$

la cual indica que el procesamiento que se inicia en la configuración inicial q_0u no se detiene nunca.

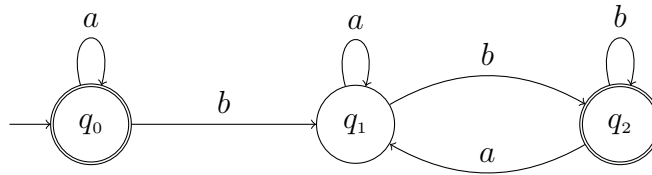
Familias de lenguajes aceptados. Las máquinas de Turing originan las siguientes clases de lenguajes:

1. L es un lenguaje *Turing-aceptable* si existe una MT M tal que $L(M) = L$.
2. L es un lenguaje *Turing-decidible* si existe una MT M tal que $L(M) = L$ y M se detiene con todas las cadenas de entrada.

Obviamente, todo lenguaje Turing-decidible es Turing-aceptable, pero la afirmación recíproca no es (en general) válida. En otras palabras, existen lenguajes que son Turing-aceptables pero no Turing-decidibles como se demostrará en la sección 6.10. Esto permite concluir que el fenómeno de máquinas que nunca se detienen no se puede eliminar de la Teoría de la Computación.

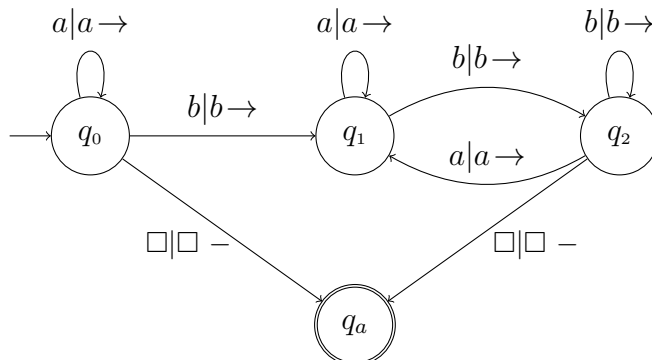
A continuación demostraremos que los lenguajes regulares son Turing-decidibles. Intuitivamente, parece claro que un autómata determinista AFD se puede simular con una MT estándar. No obstante, en un AFD M se permiten transiciones desde estados de aceptación pero en una MT no. La simulación de M por medio de una MT M' consiste en añadir a M un nuevo estado q_a , que será el único estado de aceptación de M' , y transiciones con casilla en blanco desde los estados de aceptación originales hasta q_a . Consideremos, por ejemplo, el siguiente AFD M cuyo alfabeto de entrada es $\Sigma = \{a, b\}$.

M :



Las transiciones de la MT M' que simula a M deben declarar explícitamente el desplazamiento a la derecha. Cuando termina de leer una entrada, M detecta una casilla en blanco y se detiene, aceptando si la unidad de control está en los estados q_0 o q_2 ; la MT M' acepta exactamente las mismas entradas siguiendo las transiciones que llegan a q_a desde q_0 y q_2 .

M' :



La simulación utilizada en el anterior ejemplo se puede generalizar; esto se hace en el siguiente teorema.

6.1.1 Teorema. Todo autómata finito se puede simular con una MT estándar. En consecuencia, todo lenguaje regular es Turing-decidible.

Demostración. Todo autómata finito (modelo AFD, AFN o AFN- λ) se puede convertir en un AFD $M = (Q, q_0, F, \Sigma, \delta)$ equivalente, usando las técnicas del Capítulo 2. Luego se construye una MT M' que simula a M , añadiendo un nuevo estado q_a , que será el único estado de aceptación de M' , y transiciones con casilla en blanco desde los estados de F hasta q_a . En concreto, $M' = (Q', q_0, F', \Sigma, \Gamma, \delta')$ donde

$$\begin{aligned} Q' &= Q \cup \{q_a\}, & q_a \text{ es un estado nuevo,} \\ \Gamma &= \Sigma, \\ F' &= \{q_a\}, \\ \delta'(q, s) &= (\delta(q, s), s, \rightarrow), & \text{para } q \in Q, s \in \Sigma, \\ \delta'(q, \square) &= (q_a, \square, -), & \text{para todo } q \in F. \end{aligned}$$

La MT M' así construida se detiene con cualquier entrada $u \in \Sigma^*$ y $L(M) = L(M')$. Por lo tanto, $L(M)$ es Turing-decidible. \square

6.2. Ejemplos de Máquinas de Turing

En esta sección se presentan varios ejemplos concretos de máquinas de Turing. En cada ejemplo se diseña la MT implementando un determinado algoritmo, descrito explícitamente en la forma usual, es decir, como un conjunto de instrucciones que se pueden ejecutar secuencialmente de manera sistemática. En la sección 6.7 se explora a fondo la conexión existente entre la noción intuitiva de algoritmo y los modelos teóricos de computación secuencial, en particular, la máquina de Turing.

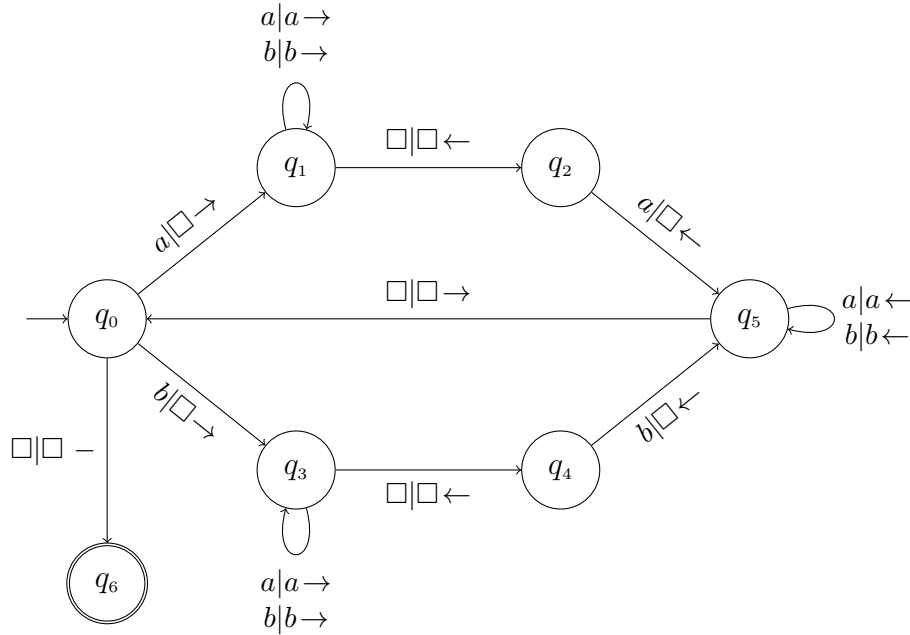
Ejemplo $\Sigma = \{a, b\}$. Diseñar una Máquina de Turing que acepte el lenguaje L de todos los palíndromos de longitud par ≥ 0 . $L = \{ww^R : w \in \Sigma^*\}$.

Solución. En el capítulo 4 se construyó un autómata con pila no-determinista para aceptar a L . Es posible aceptar L de manera determinista utilizando el modelo estándar de MT implementado el siguiente sencillo algoritmo.

Algoritmo: verificar que el primer símbolo de la entrada coincide con el último, el segundo con el penúltimo, y así sucesivamente. Aceptar únicamente cuando la entrada tiene longitud par.

La MT M exhibida a continuación hace la comparación de los símbolos ubicados en extremos simétricos y los va borrando cuando estos coinciden. Para implementar esta idea M no utiliza símbolos auxiliares; en otras palabras, el alfabeto de cinta es simplemente $\Gamma =$

$\{a, b\}$. Es importante resaltar que M está diseñada de tal forma que acepta únicamente palíndromes de longitud par. Cuando una entrada u no es un palíndromo de longitud par, la lectura de u se detiene en el estado q_2 o en el estado q_4 . Como M se detiene al leer cualquier entrada, L es un lenguaje Turing-decidible.

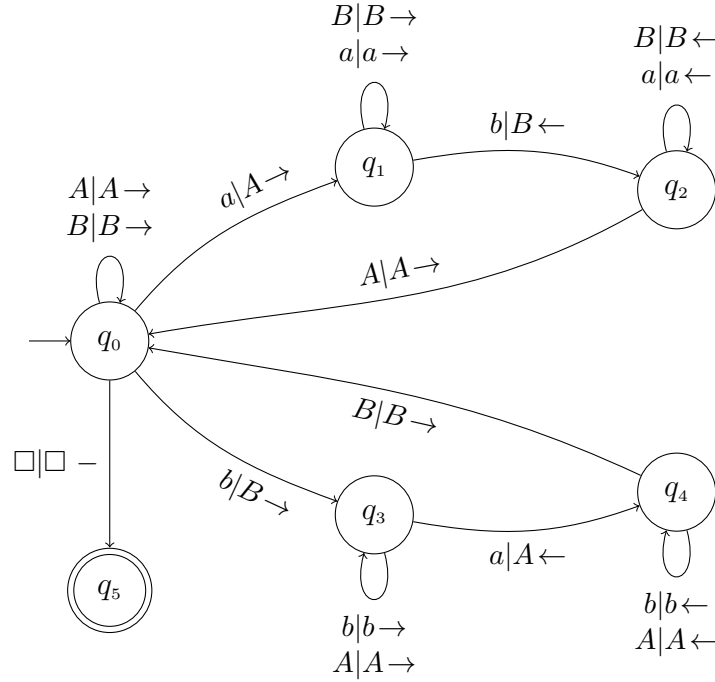


Ejemplo Sea $\Sigma = \{a, b\}$. Diseñar una MT M que acepte el lenguaje de todas las cadenas que tienen igual número de a es que de b es. Este lenguaje también se ha presentado previamente como $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u)\}$. En el capítulo 4 se diseñaron autómatas con pila para aceptar a L .

Solución. M debe verificar que toda a se corresponde con una única b y viceversa, lo cual se puede conseguir implementando el siguiente algoritmo.

Algoritmo: si el primer símbolo a la izquierda es a , sobre-escribirla por A y buscar la primera b a la derecha, sobre-escribiéndola por B . Si el primer símbolo a la izquierda es b , sobre-escribirla por B y buscar la primera a a la derecha, sobre-escribiéndola por A . Repetir este ciclo, recorriendo la cadena múltiples veces de izquierda a derecha, hasta que todas las letras minúsculas hayan sido reemplazadas por mayúsculas.

La MT M exhibida a continuación utiliza los símbolos auxiliares A y B , es decir, el alfabeto de cinta es $\Gamma = \{a, b, A, B\}$, y consta de dos ciclos simétricos, dependiendo de si el primer símbolo de la entrada es una a o una b . Al seguir la rama superior de M , la primera a se sobre-escribe por A y M busca la primera b a la derecha, la cual se cambia por B . De manera similar, la rama inferior de M empareja una b (sobre-escribiéndola por B) con la primera a que encuentra a la derecha (la cual cambia por A). Puesto que M se detiene al procesar todas las entradas (algunas aceptadas, otras rechazadas), L es un lenguaje Turing-decidible.



A continuación se presenta el procesamiento completo de la entrada $aababb$ utilizando la notación de configuración instantánea.

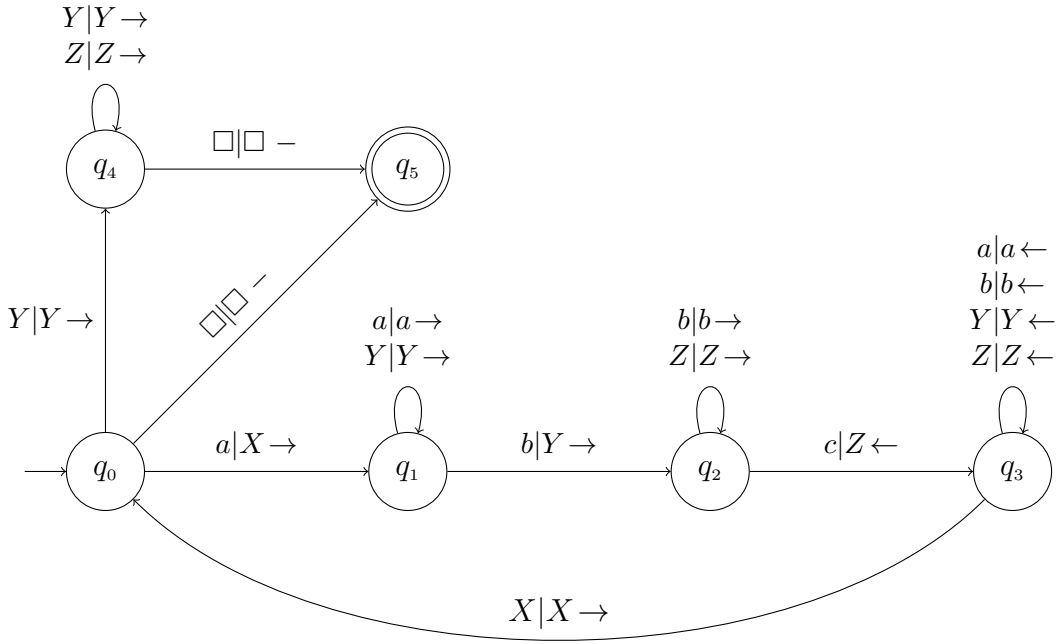
$$\begin{aligned}
 q_0 aababb &\vdash Aq_1 ababb \vdash Aa q_1 babb \vdash Aq_2 aBabb \vdash q_2 AaBabb \vdash Aq_0 aBabb \vdash AAq_1 Babb \\
 &\vdash^2 AABa q_1 bb \vdash AABq_2 aBb \vdash AAq_2 BaBb \vdash Aq_2 ABaBb \vdash AAq_0 BaBb \\
 &\vdash AABq_0 aBb \vdash AABA q_1 Bb \vdash AABABq_1 b \vdash AABAq_2 BB \vdash AABq_2 ABB \\
 &\vdash^2 AABAq_0 BB \vdash AABABBq_0 \square \vdash AABABBq_5 \square \text{ (configuración de aceptación).}
 \end{aligned}$$

Ejemplo Sea $\Sigma = \{a, b, c\}$. En este ejemplo se construye una MT M que acepta el lenguaje $L = \{a^n b^n c^n : n \geq 0\}$ y que se detiene al leer todas las entradas. Por consiguiente, L es un lenguaje Turing-decidible aunque no es un LIC, como se demostró en el Capítulo 5; es decir, L no puede ser aceptado por ningún autómata con pila.

La MT M exhibida en la siguiente página implementa el siguiente algoritmo.

Algoritmo: recorrer la cadena múltiples veces de izquierda a derecha. En cada ciclo completo reemplazar una a por X , una b por Y y una c por Z . Aceptar únicamente cuando todas las letras minúsculas hayan sido reemplazadas por mayúsculas, después de terminar el último ciclo completo.

La MT M utiliza los tres símbolos auxiliares X , Y y Z ; es decir, el alfabeto de cinta es $\Gamma = \{a, b, c, X, Y, Z\}$.



La unidad de control cambia la primera a por X y se mueve a la derecha hasta encontrar la primera b , la cual sobre-escribe por una Y . Luego se mueve hacia la derecha hasta encontrar la primera c , la cual se cambia por Z . La unidad de control retrocede entonces hacia la izquierda en busca de la primera X que encuentre en su camino; este retorno se hace en el estado q_3 . La máquina avanza luego una casilla hacia la derecha hasta la primera a que quede en la cinta, y todo el ciclo anterior se repite. Si la cadena de entrada tiene la forma requerida, todas las a s serán reemplazadas por X s, las b s por Y s y las c s por Z s. Al agotarse las a s, M detecta la primera Y en el estado q_0 , y se mueve hacia la derecha en el estado q_4 . Todavía no puede aceptar porque debe verificar que no hay más b s, c s ni símbolos adicionales en la cadena de entrada. En el estado q_5 M se salta todas las Y s y todas las Z s hasta encontrar la primera casilla en blanco, en cuyo caso ingresa al estado de aceptación q_5 .

M está diseñada de tal forma que si una cadena de entrada u no tiene la forma deseada, el procesamiento de u terminará en un estado diferente del estado de aceptación q_5 . A continuación procesamos paso a paso la cadena de entrada $w = aabbcc \in L$.

$$\begin{aligned}
 q_0 a a b b c c &\vdash X q_1 a b b c c \vdash X a q_1 b b c c \vdash X a Y q_2 b c c \vdash X a Y b q_2 c c \vdash X a Y b q_3 b Z c \\
 &\vdash^* q_3 X a Y b Z c \vdash X q_0 a Y b Z c \vdash X X q_1 Y b Z c \vdash X X Y q_1 b Z c \\
 &\vdash X X Y Y q_2 Z c \vdash X X Y Z q_2 c \vdash X X Y q_3 Z Z \vdash^* X q_3 X Y Z Z \\
 &\vdash X X q_0 Y Y Z Z \vdash X X Y q_4 Y Z Z \vdash^* X X Y Y Z Z q_4 \square \\
 &\vdash X X Y Y Z Z \square q_5 \square \text{ (configuración de aceptación).}
 \end{aligned}$$

La cadena de entrada $w = aaabbcc$, que no está en L , se procesa así:

$$\begin{aligned}
 q_0aaabbcc &\vdash Xq_1aabbcc \vdash Xaaq_1bbcc \vdash XaaYq_2bcc \vdash XaaYbq_2cc \\
 &\vdash XaaYbq_3bZc \vdash^* q_3XaaYbZc \vdash Xq_0aaYbZc \vdash XXaq_1YbZc \\
 &\vdash XXaYq_1bZc \vdash XXaYYq_2Zc \vdash XXaYZq_2c \vdash XXaYq_3ZZ \\
 &\vdash^* Xq_3XaYZZ \vdash XXq_0aYZZ \vdash XXXq_1YZZ \\
 &\vdash^* XXXYq_1ZZ \text{ (procesamiento abortado)}.
 \end{aligned}$$

Por otro lado, la cadena $abbcc$, que tampoco está en L , se procesaría así:

$$\begin{aligned}
 q_0abbcc &\vdash Xq_1bbcc \vdash XYq_2bcc \vdash XYbq_2cc \vdash XYq_3bZc \vdash^* q_3XYbZc \\
 &\vdash Xq_0YbZc \vdash XYq_4bZc \text{ (procesamiento abortado)}.
 \end{aligned}$$

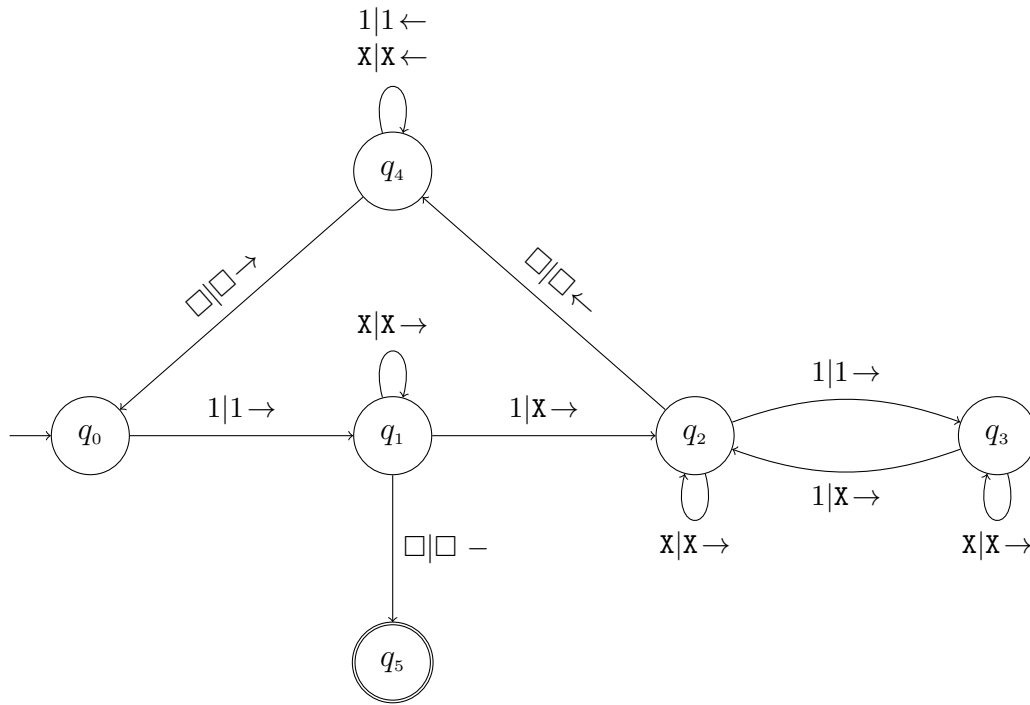
Ejemplo Sea $\Sigma = \{1\}$. Diseñar una máquina de Turing que acepte el lenguaje

$$\{1^{2^n} : n \geq 0\} = \{1, 1^2, 1^4, 1^8, 1^{16}, 1^{32}, \dots\}.$$

Solución. La idea que hay detrás del algoritmo que se utilizará para construir la MT M es que si un número natural de la forma 2^n se divide sucesivamente por 2 (n veces) se obtiene 1. Esto implica que si en una cadena de entrada de la forma 1^{2^n} se va reduciendo el número de unos a la mitad en n pasos consecutivos se obtiene finalmente la cadena 1. En el siguiente algoritmo los unos que se eliminan en cada iteración son reemplazados por una X cada uno.

Algoritmo: recorrer sucesivamente la entrada de izquierda a derecha reemplazando cada 1 por X alternadamente, es decir, un 1 se reemplaza por X pero el siguiente no. De esta manera, al completar cada ciclo exactamente la mitad de los unos existentes se han reemplazado por una X cada uno. Aceptar cuando quede un único 1 en la cinta.

En la página siguiente se exhibe una MT M que implementa este algoritmo. En un ciclo que comienza en el estado q_0 y retorna a q_0 , exactamente la mitad de los unos son reemplazados por X. Tal sustitución se hace en el bucle q_2 - q_3 . Si la entrada tiene longitud impar o si tiene longitud par que no es de la forma 2^n , el procesamiento se aborta en algún momento en el estado q_3 y la entrada no será aceptada. M se detiene al leer todas las entradas y, en consecuencia, L es un lenguaje Turing-decidible.



A continuación procesamos paso a paso la cadena 1^{16} destacando las configuraciones instantáneas que completan cada iteración que comienza en q_0 y retorna a q_0 . El procesamiento termina en la configuración de aceptación $1XXXXXXXXXXXXXXXXXq_5\Box$

```

      q01111111111111111
├*  q01X1X1X1X1X1X1X
├*  q01XXX1XXX1XXX1XXX
├*  q01XXXXXXXX1XXXXXXXX
├*  q01XXXXXXXXXXXXXXXXX
├*  1XXXXXXXXXXXXXXXXXq5□

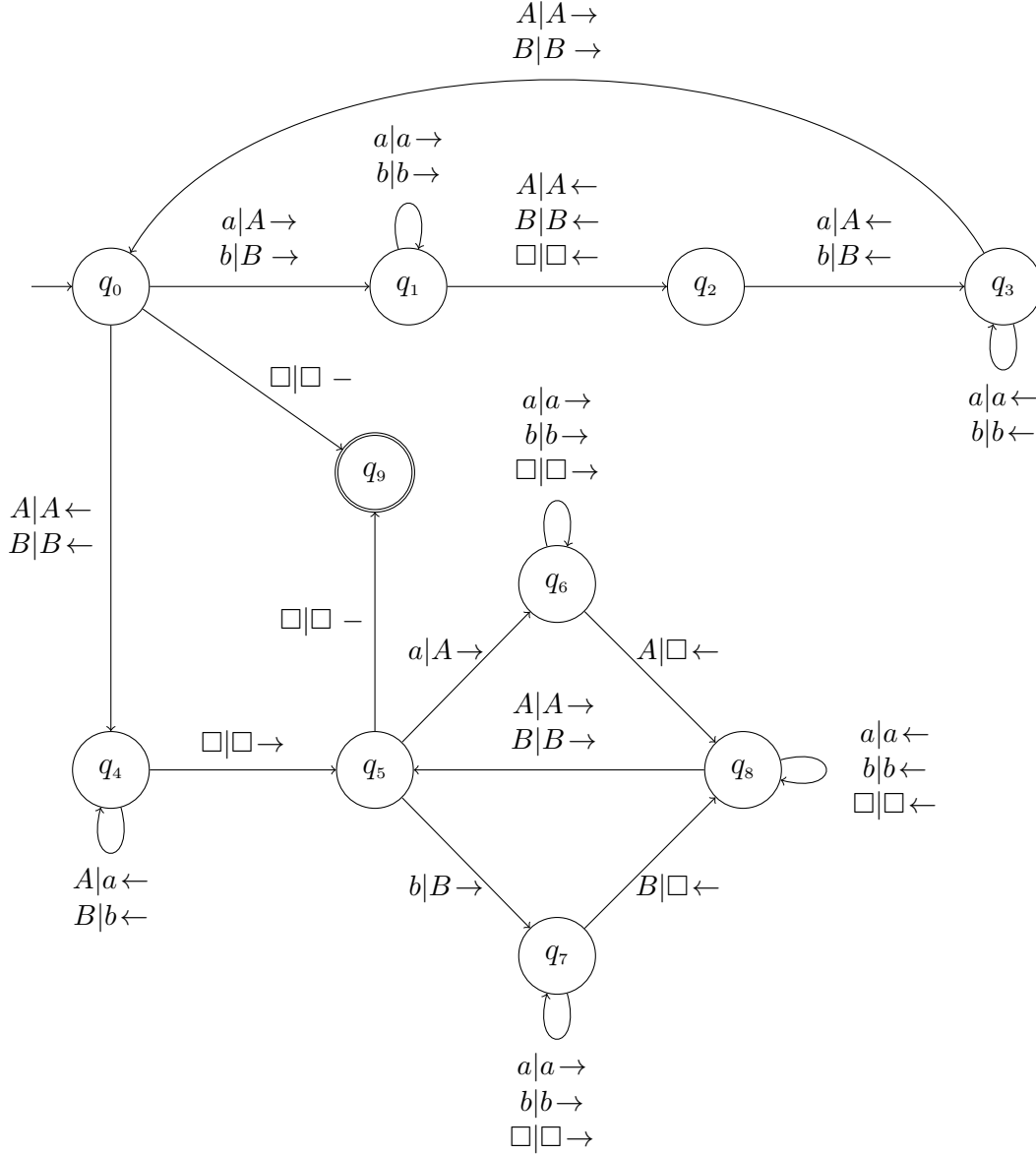
```

Ejemplo Sea $\Sigma = \{a, b\}$. Diseñar una máquina de Turing que acepte el lenguaje $L = \{ww : w \in \Sigma^*\}$.

Solución. La máquina de Turing M exhibida en la página siguiente implementa el algoritmo descrito a continuación.

Algoritmo: chequear la paridad de las entradas; una cadena de longitud impar debe ser rechazada. Escribir toda entrada u de longitud par como la concatenación de dos subcadenas de la misma longitud; es decir, escribir u en la forma $u = vw$ donde $|v| = |w|$ (esto equivale a ubicar la mitad de la cadena de entrada u). Comparar las dos mitades v y w símbolo a símbolo, leyéndolas de izquierda a derecha, y aceptar únicamente si $v = w$.

Para explicar el funcionamiento de la MT M adoptaremos la siguiente notación: si u es una cadena en Σ^* , entonces U denota la cadena obtenida a partir de u cambiando minúsculas por mayúsculas, o sea a por A y b por B . El alfabeto de cinta utilizado por M es $\Gamma = \{a, b, A, B\}$.



M procesa una entrada $u \in \Sigma^*$ en tres etapas:

1. El ciclo q_0, q_1, q_2, q_3, q_0 sobre-escribe a por A y b por B , ubicando la mitad de la entrada u , cuando esta tiene longitud par: $q_0 u \vdash^* V q_0 W$. Este ciclo sirve además para chequear la paridad de la cadena de entrada u ; si u tiene longitud impar, el ciclo se aborta en el estado q_2 .

2. En el estado q_4 se sobre-escribe de nuevo la primera mitad V por v y luego M ingresa al estado q_5 .
3. El ciclo q_5, q_6, q_7, q_8, q_5 compara las dos mitades v y W , símbolo a símbolo, convirtiendo los símbolos de v en mayúsculas y borrando los correspondientes símbolos de W , si estos coinciden. M acepta únicamente si las dos mitades V y W coinciden, y esto sucede si, estando en el estado q_5 , M detecta una casilla en blanco, en cuyo caso ingresa al estado de aceptación q_9 .

M se detiene al leer todas las entradas así que L es un lenguaje Turing-decidible. Una cadena de entrada que sea de la forma ww se procesa de la siguiente forma:

$$q_0ww \vdash^* Wq_0W \vdash^* q_4\Box wW \vdash^* q_5wW \vdash^* Wq_5\Box\Box\Box\cdots \vdash Wq_9\Box.$$

Por ejemplo, la cadena de entrada $abaababab$, que es de la forma ww , con $w = abaab$, se procesaría de la siguiente forma:

$$\begin{aligned} q_0abaababab &\vdash^* ABAABq_0ABAAB \vdash ABAq_4BABAAB \vdash^* q_4\Box abaabABAAB \\ &\vdash q_5abaabABAAB \vdash^* Aq_5baab\Box BAAB \vdash^* ABq_5aab\Box\Box AAB \\ &\vdash^* ABAq_5ab\Box\Box\Box AB \vdash^* ABAq_5b\Box\Box\Box B \vdash^* ABAABq_5\Box\Box\Box\Box \\ &\vdash ABAABq_9\Box\Box\Box\Box \text{ (configuración de aceptación).} \end{aligned}$$

Ejercicios de la sección 6.2

Sea $\Sigma = \{a, b, c\}$. Diseñar Máquinas de Turing, modelo estándar, que acepten los siguientes lenguajes. Presentar cada MT por medio de un grafo de estados.

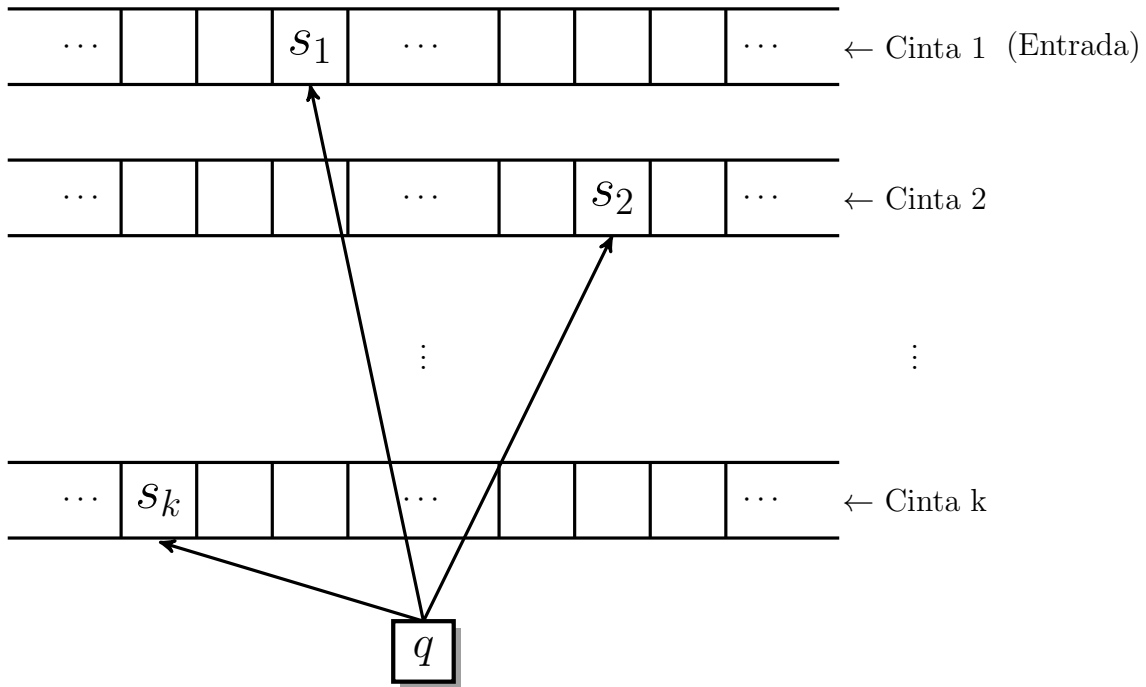
- ① $L = \{a^n b^{n+1} c^{n+2} : n \geq 0\}$.
- ② $L = \{a^n b^{2n} c^n : n \geq 0\}$.
- ③ $L = \{a^k b^m c^n : 0 \leq k \leq m \leq n\}$.
- ④ $L = \{a^k b^m c^n : 0 \leq k < m < n\}$.
- ⑤ $L = \{a^k b^m c^n : k \geq m \geq n \geq 0\}$.
- ⑥ $L = \{a^m b^n c^n : m, n \geq 1, m > n\}$.
- ⑦ $L = \{a^m b^n c^n : m, n \geq 1, n > m\}$.
- ⑧ $L = \{a^m b^n c^n : m, n \geq 1, m \neq n\}$.
- ⑨ $L = \{a^m b^n a^m c^n : m, n \geq 1\}$.

Simulación. Las máquinas de Turing que actúan sobre una cinta dividida en k pistas aceptan los mismos lenguajes que las MT estándares. Para concluir tal afirmación, basta considerar el modelo multi-pista como una MT normal en la que el alfabeto de cinta está formado por el conjunto de k -uplas (s_1, s_2, \dots, s_k) , donde los $s_i \in \Gamma_{\square}$. Es decir, el nuevo alfabeto de cinta es el producto cartesiano $(\Gamma_{\square})^k = \Gamma_{\square} \times \Gamma_{\square} \times \dots \times \Gamma_{\square}$ (k veces).

Ejemplo Las pistas se usan por lo general para señalar con “marcas” o “marcadores” ciertas posiciones en la cinta. La MT diseñada en el ejemplo de la sección 6.1 para aceptar el lenguaje $L = \{a^n b^n c^n : n \geq 0\}$ utiliza implícitamente la idea de marcadores: reemplazar las *as* por *Xs*, las *bes* por *Ys* y las *ces* por *Zs* no es otra cosa que colocar las marcas *X*, *Y* y *Z* en las posiciones deseadas. Estas marcas se pueden colocar en una pista diferente, sin necesidad de sobre-escribir los símbolos de la cadena de entrada.

6.3.3. Máquina de Turing con múltiples cintas

En el modelo multi-cinta hay k cintas diferentes (k finito, $k \geq 2$), cada una dividida en celdas o casillas, como se muestra en la siguiente figura.



La unidad de control tiene k “visores” que escanean una casilla en cada cinta. Inicialmente, la cadena de entrada se coloca en la primera cinta, llamada cinta de entrada, y las demás cintas están enteramente en blanco. En un paso computacional, la unidad de control cambia el contenido de la casilla escaneada en cada cinta y realiza luego uno de los desplazamientos \rightarrow , \leftarrow o $-$. Esto se hace de manera independiente en cada cinta; los k visores actúan independientemente en cada cinta. La función de transición tiene la

siguiente forma:

$$\delta(q, (s_1, s_2, s_3, \dots, s_k)) = (q', (s'_1, D_1), (s'_2, D_2), (s'_3, D_3), \dots, (s'_k, D_k)),$$

donde los s_i y los s'_i son símbolos pertenecientes a Γ_\square , y cada D_i es un desplazamiento \rightarrow , \leftarrow ó $-$.

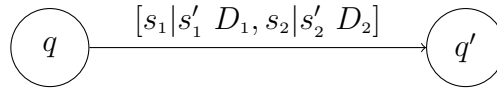
Simulación. A pesar de que el modelo multi-cinta parece, a primera vista, más poderoso que el modelo estándar, resulta que una MT con múltiples cintas se puede simular con una MT estándar. En otras palabras, si una MT multi-cinta acepta un lenguaje L , es posible construir una MT estándar que acepta el mismo lenguaje L . A continuación bosquejaremos el procedimiento de simulación.

Una MT con k cintas se simula con una MT que actúa sobre una única cinta dividida en $2k + 1$ pistas. Cada cinta de la máquina multi-cinta da lugar a dos pistas en la máquina simuladora: la primera simula la cinta propiamente dicha y la segunda tiene todas sus celdas en blanco, excepto una, marcada con un símbolo especial X , que indica la posición actual del visor de la máquina original en dicha cinta. La pista adicional de la máquina simuladora se utiliza para marcar, con los símbolos especiales Y y Y' , las posiciones más a la izquierda y más a la derecha de la unidad de control en la máquina original. Para simular un solo paso computacional, la nueva máquina requiere hacer múltiples recorridos a izquierda y a derecha, actualizando el contenido de las pistas y la posición de los marcadores X , Y y Y' .

Grafos. Es posible utilizar grafos de estados para MT multi-cintas. Por ejemplo, para una máquina con dos cintas, una instrucción de la forma

$$\delta(q, (s_1, s_2)) = (q', (s'_1, D_1), (s'_2, D_2)),$$

se presenta en el grafo como

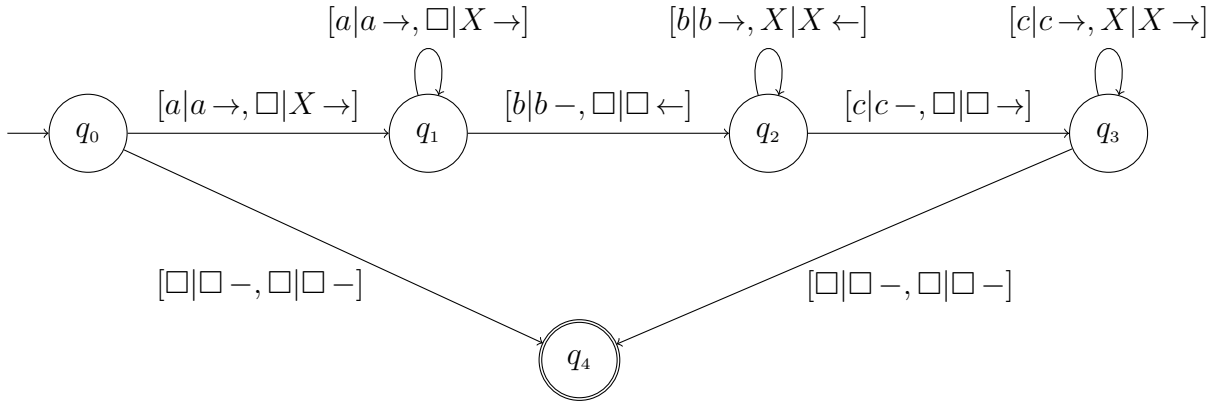


Cada instrucción se encierra entre paréntesis angulares, y las acciones en cada cinta se separan con comas, con la primera cinta a la izquierda y la segunda a la derecha.

Ejemplo Diseñar una MT con dos cintas que acepte el lenguaje $L = \{a^n b^n c^n : n \geq 0\}$.

Solución. Para aceptar este lenguaje se puede diseñar una MT con dos cintas más sencilla y eficiente que la MT estándar presentada en el tercer ejemplo de la sección 6.2 (página 166). La idea es copiar en la segunda cinta una X por cada a leída, y verificar luego hay tantas X s como b es y c es. Cuando lee las b es, la unidad de control avanza hacia la derecha en la primera cinta y hacia la izquierda en la segunda. Cuando lee las c es se avanza hacia la derecha en ambas cintas. Si la cadena de entrada tiene la forma $a^n b^n c^n$, se detectará simultáneamente el símbolo en blanco \square en ambas cintas, y la cadena se aceptará.

Se utilizan cinco estados para implementar esta idea; el grafo de M es:



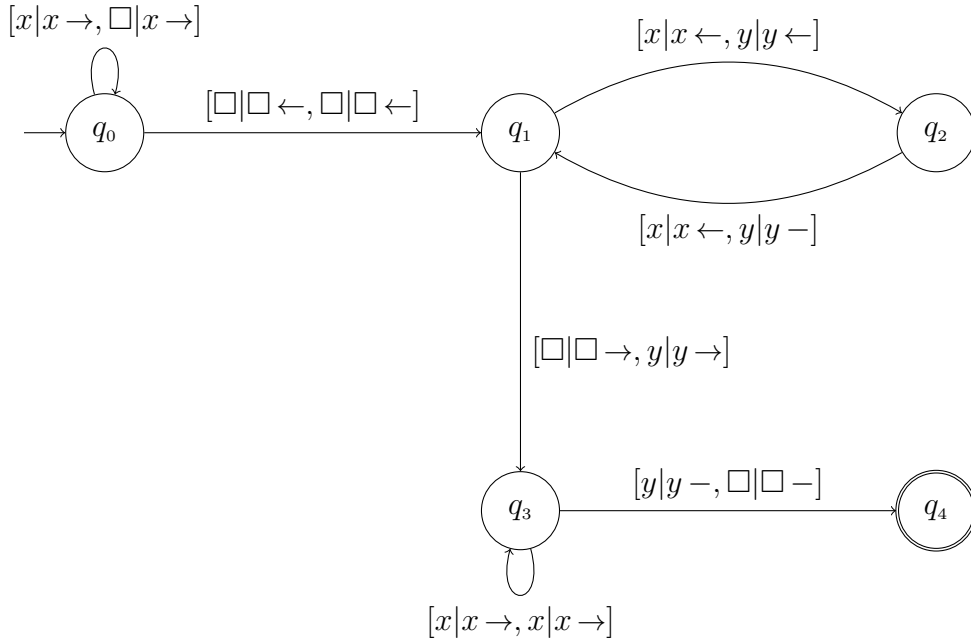
M también se puede presentar por medio de su función de transición:

$$\begin{aligned}
 \delta(q_0, (a, \square)) &= (q_1, (a, \rightarrow), (X, \rightarrow)), \\
 \delta(q_1, (a, \square)) &= (q_1, (a, \rightarrow), (X, \rightarrow)), \\
 \delta(q_1, (b, \square)) &= (q_2, (b, -), (\square, \leftarrow)), \\
 \delta(q_2, (b, X)) &= (q_2, (b, \rightarrow), (X, \leftarrow)), \\
 \delta(q_2, (c, \square)) &= (q_3, (c, -), (\square, \rightarrow)), \\
 \delta(q_3, (c, X)) &= (q_3, (c, \rightarrow), (X, \rightarrow)), \\
 \delta(q_3, (\square, \square)) &= (q_4, (\square, -), (\square, -)), \\
 \delta(q_0, (\square, \square)) &= (q_4, (\square, -), (\square, -)).
 \end{aligned}$$

Ejemplo Diseñar una MT con dos cintas que acepta el lenguaje $L = \{ww : w \in \Sigma^*\}$ sobre el alfabeto $\Sigma = \{a, b\}$.

En la sección 6.2 se presentó una MT estándar para aceptar este lenguaje. Con dos cintas se puede diseñar una máquina más sencilla y eficiente. La idea es copiar la entrada en la segunda cinta y luego retroceder a la izquierda avanzando dos casillas en la primera cinta pero solo una casilla en la segunda. Cuando la unidad de control lee una casilla en blanco en el extremo izquierdo de la primera cinta, se detecta la mitad de la entrada en la segunda cinta. Finalmente, se avanza a la derecha simultáneamente en ambas cintas verificando que los símbolos coinciden uno por uno.

Para presentar la MT se utilizará la siguiente convención: $x, y \in \{a, b\}$. Las letras x, y no son símbolos auxiliares de máquina de Turing sino son variables externas que representan los símbolos de entrada a y b . Con esta convención, el bucle sobre el estado q_0 representa dos instrucciones, una cuando x es a y otra cuando x es b . La transición $[x|x \leftarrow, y|y \leftarrow]$ entre los estados q_1 y q_2 representa cuatro instrucciones teniendo en cuenta las posibles combinaciones de valores para x y y ; lo mismo sucede con la instrucción $[x|x \leftarrow, y|y -]$ entre q_2 y q_1 . Cuando entrada tiene longitud impar, la máquina se detiene en algún momento estando en el estado q_2 . El alfabeto de cinta de esta máquina es simplemente $\{a, b\}$.



A continuación ilustramos el procesamiento de la cadena de entrada $baabbaab$ (aceptada) utilizando la notación de configuración instantánea para máquinas de Turing con dos cintas, y resaltando el funcionamiento del bucle q_1 - q_2 .

$$\begin{aligned}
 (q_0, \underline{b}aabbaab, \underline{\square}) &\stackrel{*}{\vdash} (q_1, baabbaab, \underline{baabbaab}) \stackrel{2}{\vdash} (q_1, baabbaab, \underline{baabbaab}) \\
 &\stackrel{2}{\vdash} (q_1, baabbaab, \underline{baabbaab}) \\
 &\stackrel{2}{\vdash} (q_1, baabbaab, \underline{baabbaab}) \\
 &\stackrel{2}{\vdash} (q_1, \underline{\square}baabbaab, baabbaab) \\
 &\vdash (q_3, \underline{b}aabbaab, baabbaab) \\
 &\stackrel{*}{\vdash} (q_4, baabbaab, baabbaab\underline{\square}).
 \end{aligned}$$

Ejercicios de la sección 6.3

- ① Sea $\Sigma = \{a, b, c\}$. Diseñar Máquinas de Turing con dos o más cintas que acepten los siguientes lenguajes. En cada caso, explicar brevemente el plan utilizado en el diseño y presentar la MT ya sea por medio de una lista de transiciones o por medio de un grafo de estados.

- (i) $L = \{a^n b^{2n} c^n : n \geq 0\}$.
- (ii) $L = \{a^k b^m c^n : 0 \leq k \leq m \leq n\}$.
- (iii) $L = \{a^k b^m c^n : k \geq m \geq n \geq 0\}$.
- (iv) $L = \{a^m b^n c^n : m, n \geq 1, m > n\}$.

- (v) $L = \{a^m b^n c^n : m, n \geq 1, n > m\}.$
- (vi) $L = \{a^m b^n a^m c^n : m \neq n\}.$
- (vii) $L = \{a^m b^n a^m c^n : m, n \geq 1\}.$
- (viii) $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u) = \#_c(u)\}.$

- ② Sea $\Sigma = \{a, b, c, d\}$. Diseñar una Máquina de Turing con dos o más cintas que acepte el lenguaje $L = \{a^n b^n c^n d^n : n \geq 0\}$.

6.4. Funciones Turing-computables

Como las máquinas de Turing tienen la capacidad de transformar entradas en salidas se pueden utilizar como mecanismos para calcular funciones. Esta noción se precisa en las siguientes definiciones.

6.4.1 Definiciones. Sean Σ y Γ dos alfabetos. Una función $f : \Sigma^* \rightarrow \Gamma^*$ cuyo dominio, denotado $\text{dom}(f)$, es un subconjunto propio de Σ^* se dice que está parcialmente definida, o que es una función parcial. Si $\text{dom}(f) = \Sigma^*$ se dice que f está totalmente definida o que es una función total.

Una función parcial o total $f : \Sigma^* \rightarrow \Gamma^*$, con dominio $\text{dom}(f)$, es *Turing-computable* o *Turing-calculable* si existe una MT estándar $M = (Q, q_0, q_a, \Sigma, \Gamma', \delta)$ tal que, para toda cadena $u \in \Sigma^*$,

$$u \in \text{dom}(f) \implies q_0 u \vdash^* q_a v, \quad \text{donde } v = f(u).$$

Es decir, con entrada u , M produce la salida $f(u)$. El estado q_a es un estado de aceptación en el sentido usual, pero aquí representa el estado en el cual se detiene M para producir salidas. Como M es una MT estándar, no se permiten transiciones desde el estado q_a . Nótese que M debe detenerse en la configuración $q_a v$, o sea, la unidad de control debe estar escaneando el primer símbolo de la salida v .

El alfabeto de cinta Γ' de M incluye los alfabetos Σ y Γ , esto es, $\Sigma \cup \Gamma \subseteq \Gamma'$. También se dice que M *calcula* o *computa* la función f .

Para funciones numéricas se puede tomar $\Sigma = \{1\}$ y usar el sistema de numeración unitario: si n es un número natural, n se escribe como la secuencia de n unos 1^n , y la cadena vacía representa el número 0. Con esta representación el conjunto \mathbb{N} de los números naturales se identifica con

$$\mathbb{N} = \{1^n : n \geq 0\} = \{\lambda, 1, 11, 111, 1111, 11111, \dots\}$$

Como caso particular, una función numérica $f : \mathbb{N} \rightarrow \mathbb{N}$ es Turing-computable si existe una MT estándar $M = (Q, q_0, q_a, \Sigma, \Gamma, \delta)$ tal que $q_0 1^m \vdash^* q_a 1^n$ siempre que $f(m) = n$, con $m, n \in \mathbb{N}$.

La noción de función Turing-computable se puede extender fácilmente a funciones de varios argumentos. Sea $(\Sigma^*)^k = \Sigma^* \times \dots \times \Sigma^*$ (k veces). Una función $f : (\Sigma^*)^k \rightarrow \Gamma^*$

parcial o total, con dominio $\text{dom}(f)$, es Turing-computable si existe una MT estándar $M = (Q, q_0, q_a, \Sigma, \Gamma, \delta)$ tal que, para toda k -upla $(u_1, u_2, \dots, u_k) \in (\Sigma^*)^k$ se tiene

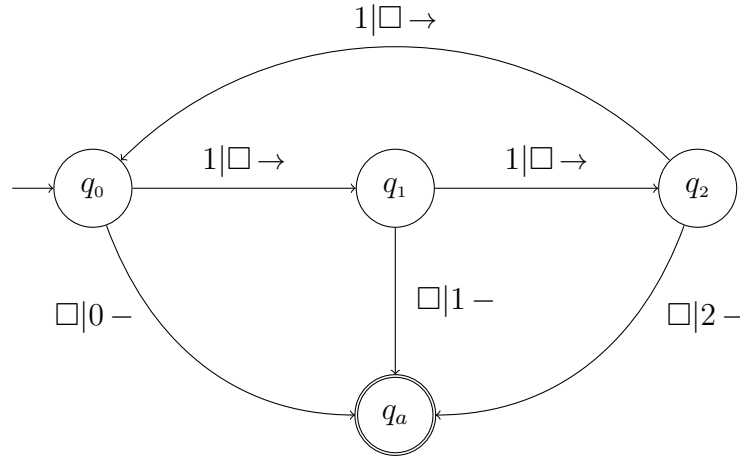
$$(u_1, u_2, \dots, u_k) \in \text{dom}(f) \implies q_0 u_1 \square u_2 \square \dots \square u_k \vdash^* q_a v, \quad \text{donde } v = f(u_1, u_2, \dots, u_k).$$

Nótese que para escribir la entrada en la cinta, los k argumentos u_1, u_2, \dots, u_k están separados entre sí por una casilla en blanco. Igual que antes, no se permiten transiciones desde el estado final q_a .

Ejemplo Diseñar una MT M que calcule el residuo de división de n por 3, para cualquier número natural $n \geq 0$, escrito en el sistema de numeración unitario.

Solución. Aquí $\Sigma = \{1\}$, $\Gamma = \{0, 1, 2\}$ y f es la función total $f : \Sigma^* \longrightarrow \Gamma^*$ definida por $f(n) = \text{residuo de división de } n \text{ por } 3$, para todo $n \in \Sigma^*$.

Los posibles residuos de división por 3 son 0, 1 y 2, por lo cual bastan 3 estados, aparte de q_a , para calcular esta función. M recorre de izquierda a derecha la secuencia de entrada borrando los unos y pasando alternadamente por los estados q_0 (que representa el residuo 0), q_1 (residuo 1) y q_2 (residuo 2). El grafo de M se muestra a continuación.



Orden lexicográfico. Sea $\Sigma = \{s_1, s_2, \dots, s_k\}$ un alfabeto dado en el cual los símbolos tienen un orden preestablecido, $s_1 < s_2 < \dots < s_k$. En el conjunto Σ^* de todas las cadenas se define el *orden lexicográfico*, también denotado $<$, de la siguiente manera. Sean u, v dos cadenas en Σ^* ,

$$u = a_1 a_2 \dots a_m, \quad \text{donde } a_i \in \Sigma, \text{ para } 1 \leq i \leq m.$$

$$v = b_1 b_2 \dots b_n, \quad \text{donde } b_i \in \Sigma, \text{ para } 1 \leq i \leq n.$$

Se tiene $u < v$ si

(1) $|u| < |v|$ (es decir, si $m < n$) o,

(2) $|u| = |v|$ (es decir, si $m = n$) y para algún índice i , $1 \leq i \leq m$, se cumple que

$$a_1 = b_1, a_2 = b_2, \dots, a_{i-1} = b_{i-1}, a_i < b_i.$$

El orden lexicográfico entre cadenas es un orden lineal, o sea, para todo par de cadenas diferentes u, v en Σ^* se tiene $u < v$ o $v < u$, y es el orden utilizado para listar las palabras en los diccionarios de los lenguajes naturales.

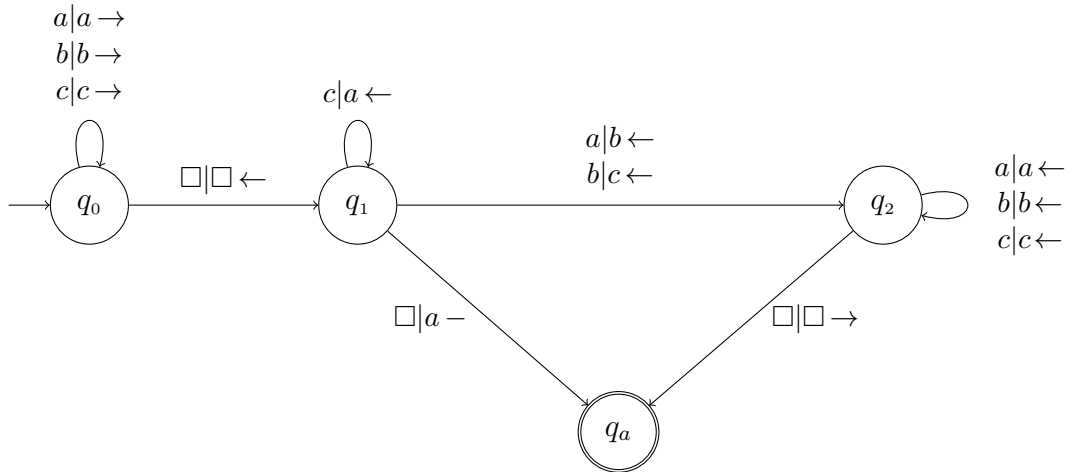
Ejemplo Sea $\Sigma = \{a, b, c\}$, donde los símbolos tienen el orden preestablecido $a < b < c$. La MT M exhibida abajo calcula la función $f : \Sigma^* \rightarrow \Sigma^*$ definida por

$$f(u) = \text{cadena que sigue a } u \text{ en el orden lexicográfico.}$$

Las primeras cadenas de Σ^* ordenadas lexicográficamente son:

$\lambda, a, b, c, aa, ab, ac, ba, bb, bc, ca, cb, cc, aaa, aab, aac, aba, abb, abc, aca, acb, acc, baa, bab, \dots$

Hay que tener presente que la cadena que sigue a una cadena no-vacía u en el orden lexicográfico tiene la misma longitud que u , a menos que u solamente tenga c s. Así por ejemplo, $f(bab) = bac$, $f(babc) = baca$, $f(babccc) = bacaaa$, $f(cccc) = aaaaa$.



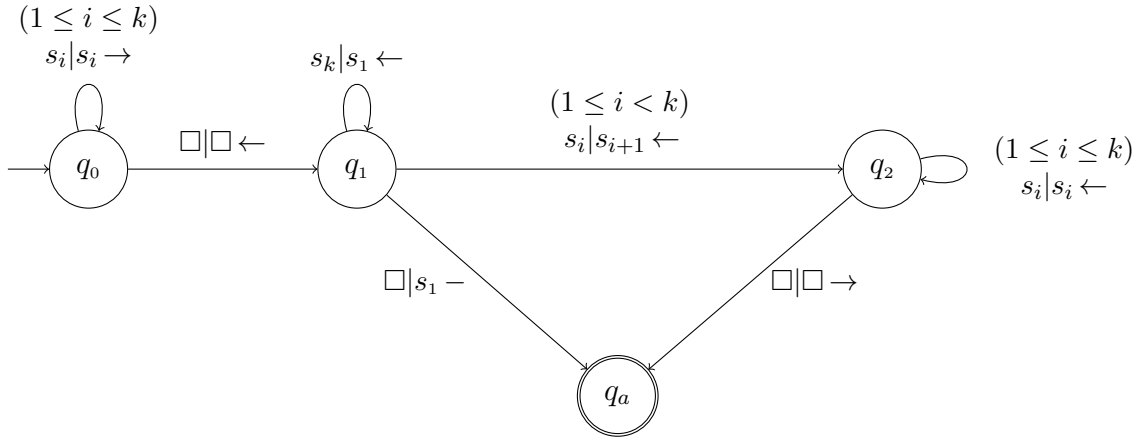
La MT M calcula $f(babccc) = bacaaa$ de la siguiente manera:

$$q_0 babccc \vdash^6 babccq_0 \square \vdash babccq_1 c \vdash^3 baq_1 baaa \vdash bq_2 acaaa \vdash^3 q_a bacaaa.$$

Esta MT se puede modificar, sin necesidad de añadir más estados, para el caso de alfabetos con un número arbitrario de símbolos. Sea $\Sigma = \{s_1, s_2, \dots, s_k\}$ un alfabeto dado en el cual los k símbolos tienen el orden preestablecido $s_1 < s_2 < \dots < s_k$. La MT exhibida en la siguiente página calcula la función $f : \Sigma^* \rightarrow \Sigma^*$ definida por

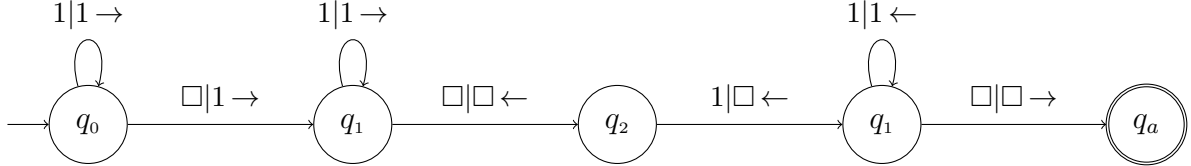
$$f(u) = \text{cadena que sigue a } u \text{ en el orden lexicográfico.}$$

Los bucles de los estados q_0 y q_2 tienen k instrucciones cada uno, con $1 \leq i \leq k$. Entre q_1 y q_2 hay $k - 1$ instrucciones, con $1 \leq i < k$.



Ejemplo Sea $\Sigma = \{1\}$. Diseñar una MT M que calcule la función parcial suma, en el sistema de numeración unitario, definida por $f(m, n) = m + n$, donde $n, m \geq 1$. Con entrada $1^m \square 1^n$, M debe producir como salida 1^{m+n} . Las secuencias de unos, 1^m y 1^n , representan los números naturales m y n , respectivamente.

Solución. La MT M que se exhibe a continuación transforma la entrada $1^m \square 1^n$ en la salida 1^{m+n} trasladando la cadena 1^n una casilla hacia la izquierda. Esto se consigue escribiendo un 1 en la casilla en blanco que hay entre 1^m y 1^n , y borrando el último 1 de 1^n .



Esta MT se puede modificar para calcular la función total suma, $f : \Sigma^* \rightarrow \Sigma^*$ definida por $f(m, n) = m + n$, donde $m, n \geq 0$, es decir, incluyendo los casos $m = 0$ o $n = 0$ (véase el Ejercicio ① de la presente sección).

Ejercicios de la sección 6.4

Diseñar MT que calculen las siguientes funciones numéricas, utilizando para \mathbb{N} el sistema de numeración unitario.

① $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, donde $f(m, n) = m + n$.

② $f : \mathbb{N} \rightarrow \mathbb{N}$, donde $f(n) = 2n$.

③ $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, donde

$$f(m, n) = \begin{cases} 1, & \text{si } m \geq n, \\ 0, & \text{si } m < n. \end{cases}$$

④ $f : \mathbb{N} \times \mathbb{N} \rightarrow \mathbb{N}$, donde $f(m, n) = \max(m, n)$.

⑤ $f : \mathbb{N} \times \mathbb{N} \times \mathbb{N} \longrightarrow \mathbb{N}$, donde $f(i, j, k) = j$. Esta función se llama “segunda proyección”.

⑥ $f : \mathbb{N}^k \longrightarrow \mathbb{N}$ donde

$$f(m_1, \dots, m_i, \dots, m_k) = m_i.$$

f es la llamada i -ésima proyección.

⑦ $f : \mathbb{N} \longrightarrow \mathbb{N}$, donde $f(n) = 2^n$.

6.5. Máquinas de Turing no-deterministas (MTN)

El modelo no-determinista de máquina de Turing tiene los mismos componentes del modelo estándar, $M = (Q, q_0, q_a, \Sigma, \Gamma, \delta)$, con un único estado de aceptación q_a , pero se permite que, en un paso computacional, la unidad de control escoja aleatoriamente entre varias transiciones posibles. La función δ tiene la siguiente forma:

$$\delta(q, s) = \{(p_1, s_1, D_1), (p_2, s_2, D_2), \dots, (p_k, s_k, D_k)\},$$

donde s y los s_i son símbolos pertenecientes a Γ_\square , los p_i son estados y cada D_i es un desplazamiento.

La noción de aceptación en una MTN es similar a la de los modelos no-deterministas de autómatas considerados antes: una cadena de entrada u es aceptada si existe por lo menos un procesamiento, a partir de la configuración inicial q_0u , que termine en una configuración de aceptación, vq_aw , con $v, w \in (\Gamma_\square)^*$.

Si $u \in \Sigma^*$, una secuencia de configuraciones instantáneas C_0, C_1, \dots, C_m tal que

$$C_0 \vdash C_1 \vdash \dots \vdash C_m,$$

donde C_0 es la configuración inicial q_0u , se denomina una *computación* (de m pasos). Si C_m es una configuración de aceptación, la secuencia es llamada una *computación de aceptación de u* . El lenguaje aceptado por M se puede describir como

$$L(M) := \{u \in \Sigma^* : \text{existe una computación de aceptación } C_0 \vdash C_1 \vdash \dots \vdash C_m\}.$$

El no determinismo no amplía la familia de lenguajes aceptados por máquinas de Turing; en otras palabras, una MTN se puede simular con una MT estándar, como se demuestra en el siguiente teorema.

6.5.1 Teorema. Todo lenguaje aceptado por una máquina de Turing no-determinista M puede ser aceptado por una MT estándar M' .

Demostración. Sea $M = (Q, q_0, q_a, \Sigma, \Gamma, \delta)$ una máquina de Turing no-determinista dada. La idea de la demostración es que si una cadena $u \in \Sigma^*$ es aceptada por M , la MT simuladora M' también va a aceptar a u después de realizar una búsqueda exhaustiva

entre todas las computaciones $C_0 \vdash C_1 \vdash \dots \vdash C_m$, donde C_0 es la configuración inicial q_0u y $m \geq 1$. M' primero examina las computaciones de un paso, luego las computaciones de dos pasos, y así sucesivamente hasta encontrar una computación de aceptación.

Sea n el número máximo de transiciones en los conjuntos $\delta(q, s)$, considerando todo símbolo $s \in \Gamma_\square$ y todo estado $q \in Q$. Para cada $s \in \Gamma_\square$ y $q \in Q$, las transiciones contenidas en $\delta(q, s)$ se pueden enumerar entre 1 y n . Si hay menos de n transiciones en un $\delta(q, s)$ particular, se repite arbitrariamente una de ellas hasta completar n . De esta manera, cada una de las transiciones $\delta(q, s)$ se puede considerar como un conjunto ordenado, con exactamente n opciones indexadas:

$$\delta(q, s) = \left\{ \underbrace{(p_1, b_1, D_1)}_1, \underbrace{(p_2, b_2, D_2)}_2, \dots, \underbrace{(p_n, b_n, D_n)}_n \right\} \leftarrow \text{índice}$$

En algunos conjuntos $\delta(q, s)$ habrá opciones repetidas, pero esta repetición no altera el lenguaje aceptado.

Sea $\Phi = \{\varphi_1, \varphi_2, \dots, \varphi_n\}$ un alfabeto con n símbolos nuevos (o sea, Γ y Φ son alfabetos disyuntos). Se puede pensar que el alfabeto $\{\varphi_1, \varphi_2, \dots, \varphi_n\}$ es una “codificación” del conjunto de números naturales $\{1, 2, \dots, n\}$ (esto es, φ_1 representa a 1, φ_2 representa a 2 y así sucesivamente). La MT M' que simulará a M tendrá tres cintas. La primera cinta almacena la entrada, la segunda cinta contiene en todo momento una cadena de Φ^* , que representa una secuencia finita de números naturales entre 1 y n , y en la tercera cinta se hace la simulación de M propiamente dicha.

Concretamente, con entrada $u \in \Sigma^*$, M' realiza el siguiente procedimiento de 3 etapas:

- (1) Copia la cadena de entrada u (que está escrita en la primera cinta) en la tercera cinta, y escribe la cadena $x = \varphi_1$ en la segunda cinta.
- (2) Si $x = \varphi_{i_1}\varphi_{i_2} \dots \varphi_{i_k}$ es la cadena escrita en la segunda cinta, M' simula en la cinta 3 la computación de k pasos que hace sobre u la máquina original M , utilizando en el paso j la opción con índice i_j del conjunto $\delta(q, s)$ correspondiente. Durante la simulación, el visor de la segunda cinta se va desplazando una casilla a la derecha de tal manera que en el j -ésimo paso está leyendo el símbolo φ_{i_j} . Si M' se detiene en algún momento en el estado de aceptación q_a , entonces M' acepta la entrada (como lo ha hecho M). Si después de estos k pasos la computación simulada no termina en aceptación, se pasa a la etapa (3).
- (3) M' reemplaza la cadena x que está escrita en la segunda cinta por la cadena que sigue a x en el orden lexicográfico, sobre el alfabeto $\Phi = \{\varphi_1, \varphi_1, \dots, \varphi_n\}$. Para ello, M' utiliza como subrutina la función Turing-computable presentada en la sección 6.4. A continuación, M' retorna a la etapa (2).

Si la MT original M acepta una cadena $u \in \Sigma^*$ siguiendo una computación de m pasos que corresponde a la sucesión i_1, i_2, \dots, i_m de enteros del conjunto $\{1, 2, \dots, n\}$ (en el sentido

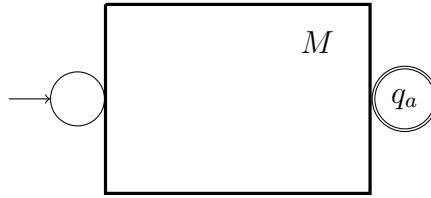
de que M utiliza en el paso r la opción i_r entre todas las opciones disponibles), entonces la MT M' simulará dicha computación cuando la cadena $x = \varphi_{i_1}\varphi_{i_2}\cdots\varphi_{i_m}$ esté escrita en la segunda cinta, y la cadena u será aceptada por M' . Esto demuestra que $L(M) \subseteq L(M')$. Recíprocamente, se tiene $L(M') \subseteq L(M)$ ya que M' solo puede aceptar cadenas que son aceptadas por M . Se concluye entonces que $L(M) = L(M')$.

Un detalle importante en la anterior simulación es que si la cadena $u \in \Sigma^*$ no es aceptada por la MT original M , entonces, con entrada u , la MT M' nunca se detendrá (entrará en un bucle infinito) porque regresará una y otra vez a la etapa (3), chequeando sin éxito todas las cadenas de Φ^* , en el orden lexicográfico. \square

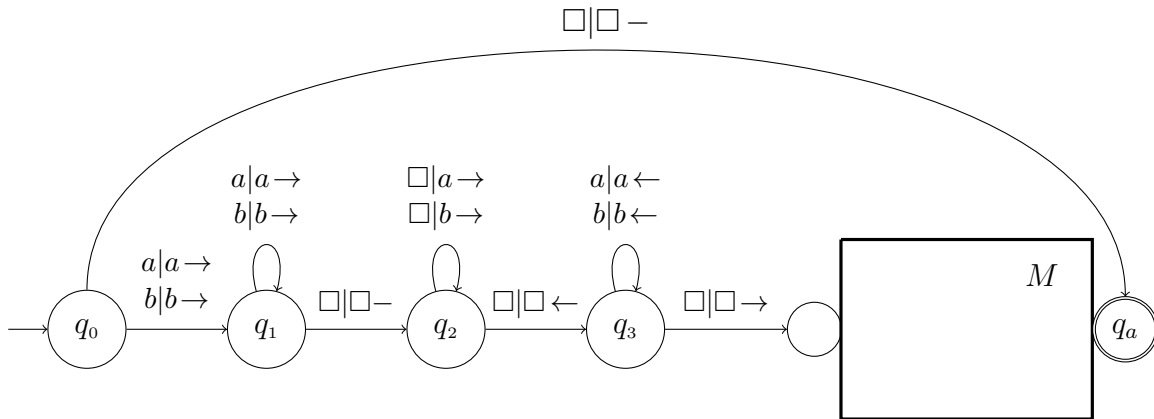
Ejemplo Para un lenguaje L sobre Σ se define P_L como el lenguaje de todos los prefijos de cadenas de L :

$$P_L = \{x \in \Sigma^* : (\exists y \in \Sigma^*)(xy \in L)\}.$$

Es claro que $\lambda \in P_L$ porque λ es prefijo de cualquier cadena, y $L \subseteq P_L$ porque toda cadena es prefijo de sí misma. Supóngase que el lenguaje L es Turing-aceptable; existe entonces una MT M , modelo estándar, con un único estado de aceptación q_a , tal que $L(M) = L$:



Una MTN M' tal que $L(M') = P_L$ se muestra en la siguiente gráfica.



M' actúa así: con una entrada $x \in \Sigma^*$ sobre la cinta, su cabeza se mueve, en el estado q_1 , hacia el extremo derecho de x y escribe aleatoriamente una cadena $y \in \Sigma^*$, en el estado q_2 . Las transiciones que salen de q_2 constituyen la parte no-determinista del funcionamiento de M' . A continuación, M' retorna al extremo izquierdo, en el estado q_3 , y procesa la

cadena xy escrita en cinta utilizando la máquina original M . Si realmente $x \in P_L$, entre todos los cómputos posibles con entrada x , M' encontrará eventualmente una cadena $y \in \Sigma^*$ tal que $xy \in L$ y aceptará a x . Si $x \notin P_L$, ninguna de las elecciones de y conducirá a la aceptación.

La MTN M' está diseñada para implementar el siguiente algoritmo no-determinista:

Algoritmo. Con entrada $x \in \Sigma^*$:

- (1) Escribir aleatoriamente una cadena y perteneciente a Σ^* .
- (2) Correr la MT M con entrada xy . Si M acepta la cadena xy , aceptar la entrada x .

La parte (1) es la etapa no-determinista del algoritmo y constituye una “conjetura”. La parte (2) es una etapa de “verificación”, que es completamente determinista. Muchos algoritmos no-deterministas tienen esta estructura de dos etapas que se puede sintetizar en la fórmula

$$\text{NO-DETERMINISMO} = \text{CONJETURA} + \text{VERIFICACIÓN}.$$

Según el Teorema 6.5.1 para la MTN M' se puede construir una MT determinista M'' tal que $L(M') = L(M'') = P_L$. Con entrada x , la máquina simuladora M'' reemplaza la etapa de conjetura por la búsqueda exhaustiva de un sufijo y tal que $xy \in L$. Si la cadena x no está en P_L , esta búsqueda es infructuosa y la máquina M'' nunca se detendrá con entrada x .

Una máquina de Turing multicinta no-determinista se define como el modelo multicinta de la sección 6.3.3 pero permitiendo un número finito de opciones en cada transición. Utilizando un argumento similar al del Teorema 6.5.1, se puede demostrar que este modelo es equivalente al modelo estándar. Este hecho se enuncia en el siguiente teorema.

6.5.2 Teorema. Todo lenguaje aceptado por una máquina de Turing multicinta no-determinista puede ser aceptado por una máquina de Turing estándar.

Ejercicios de la sección 6.5

- ① Sea L un lenguaje sobre Σ y M una MT, modelo estándar, tal que $L(M) = L$. Construir MTN (Máquinas de Turing no-deterministas) que acepten los siguientes lenguajes:

- (i) El lenguaje S_L de todos los sufijos de cadenas de L ; es decir,

$$S_L = \{x \in \Sigma^* : (\exists y \in \Sigma^*)(yx \in L)\}.$$

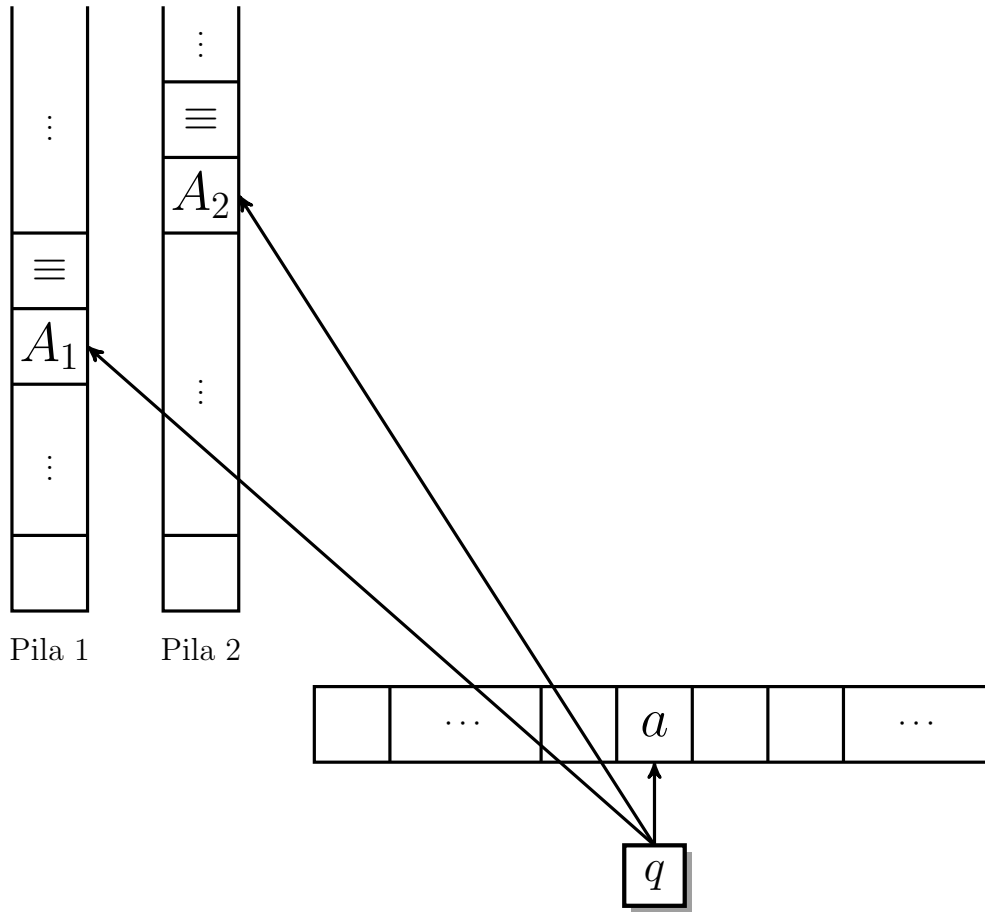
- (ii) El lenguaje C_L de todas las subcadenas de cadenas de L ; es decir,

$$C_L = \{x \in \Sigma^* : (\exists y, z \in \Sigma^*)(yxz \in L)\}.$$

- ② Sea $\Sigma = \{a, b\}$ y $L = \{ww : w \in \Sigma^*\}$. Diseñar una Máquina de Turing no-determinista (con una o varias cintas) que acepte el lenguaje L .

6.6. Autómatas con dos pilas (AF2P)

En esta sección presentaremos el modelo de autómata con dos pilas (AF2P) que resulta también equivalente al modelo estándar de máquina de Turing. Un autómata con dos pilas es esencialmente un AFPN, tal como se definió en el capítulo 4, con la adición de una pila más. Podríamos también definir autómatas con k pilas, $k \geq 3$, pero tal modelo no aumenta la capacidad computacional que se consigue con dos pilas. Las pilas tienen la misma restricción que antes: el autómata sólo tiene acceso al símbolo que está en el tope de cada pila. Un paso computacional depende del estado actual de la unidad de control, del símbolo escaneado en la cinta y de los dos topes de pila, como se muestra en la siguiente gráfica.

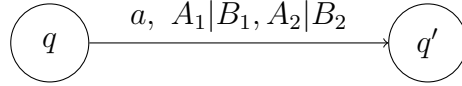


Al procesar un símbolo $a \in \Sigma$, la unidad de control se desplaza una casilla a la derecha, cambia al estado q' (que puede ser el mismo q) y realiza sobre cada pila una de las siguientes cuatro acciones: o reemplaza el tope de la pila por otro símbolo, o añade un nuevo símbolo a la pila, o borra el tope de la pila, o no altera la pila. Estas acciones permitidas en las pilas son las mismas que en el modelo AFPN y están definidas en términos de la función de transición Δ en la que las instrucciones básicas tienen la forma

$$\Delta(q, a, A_1, A_2) = (q', B_1, B_2).$$

Se tienen en cuenta todos los casos $a \in \Sigma \cup \{\lambda\}$ y $A_1, A_2, B_1, B_2 \in \Gamma \cup \{\lambda\}$. El caso $a = \lambda$ corresponde a las transiciones λ .

La instrucción $\Delta(q, a, A_1, A_2) = (q', B_1, B_2)$ se representa en el grafo de estados en la siguiente forma:



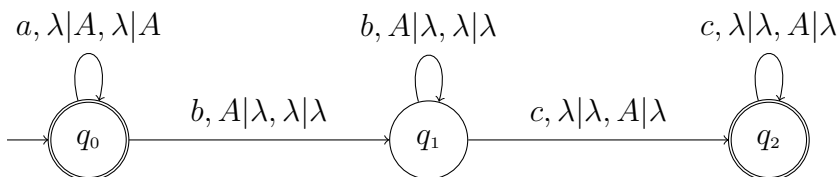
El modelo AF2P es no-determinista, de manera que cada $\Delta(q, a, A_1, A_2)$ es, en general, un conjunto finito de opciones, cada una de las cuales es una instrucción básica. Inicialmente las dos pilas están vacías y la unidad de control está escaneando el fondo de cada pila. La cadena de entrada se coloca en la cinta de entrada, en la forma usual. Nótese que sólo es necesario exigir que la cinta de entrada sea infinita en una dirección ya que la unidad de control no puede nunca retornar a la izquierda.

Una configuración instantánea es una cuádrupla de la forma $[q, v, \beta_1, \beta_2]$ que representa lo siguiente: la unidad de control está en el estado q , v es la parte no procesada de la cadena de entrada (la unidad de control está escaneando el primer símbolo de v), y las cadenas β_1 y β_2 son el contenido total de la pila 1 y de la pila 2, respectivamente. Tanto β_1 como β_2 se leen en la pila de arriba hacia abajo, así que el tope de la pila i es el primer símbolo de β_i , $i = 1, 2$. Si $w \in \Sigma^*$ es una entrada, la configuración inicial es $[q_0, w, \lambda, \lambda]$ y una configuración de aceptación es de la forma $[p, \lambda, \lambda, \lambda]$ donde $p \in F$; es decir, para ser aceptada, una entrada debe ser procesada completamente, el procesamiento debe terminar con las dos pilas vacías y la unidad de control en un estado de aceptación. El lenguaje aceptado por un AF2P M se define entonces como:

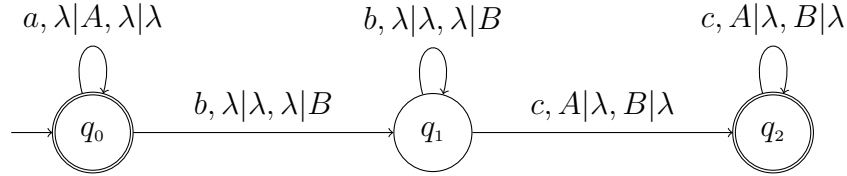
$$L(M) := \{w \in \Sigma^* : \text{existe un procesamiento } [q_0, w, \lambda, \lambda] \xrightarrow{*} [p, \lambda, \lambda, \lambda], p \in F\}.$$

Ejemplo Diseñar un AF2P que acepte el lenguaje $L = \{a^n b^n c^n : n \geq 0\}$.

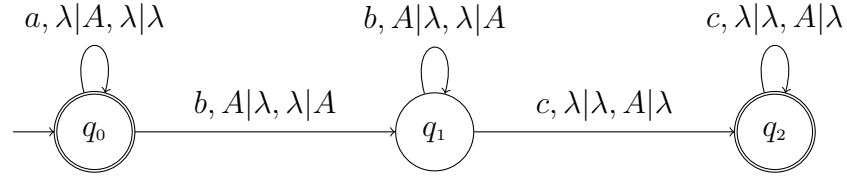
Solución 1. Un plan natural es apilar las a es en ambas pilas. Al aparecer las b es se va vaciando la primera pila (sin alterar la segunda) y al aparecer las c es se va vaciando la segunda pila (sin alterar la primera). Si la entrada tiene la forma $a^n b^n c^n$, se consume completamente la entrada y se vacían ambas pilas. Por el contrario, si la entrada no es de la forma $a^n b^n c^n$, o bien no se consume toda la entrada, o bien no desocupa alguna de las pilas. Para implementar esta idea utilizamos tres estados, siendo q_2 el único estado de aceptación. El estado inicial q_0 debe ser de aceptación para aceptar la cadena λ . El alfabeto de pila es $\Gamma = \{A\}$.



Solución 2. También es posible apilar las *aes* (como *Aes*) en la primera pila y las *bes* (como *Bes*) en la segunda pila, y luego vaciar simultáneamente ambas pilas cuando aparecen las *ces* en cinta. En este caso, el alfabeto de pila es $\Gamma = \{A, B\}$.



Solución 3. Otra manera de aceptar a L es apilar las *aes* en la primera pila únicamente, dejando la segunda pila vacía. Al aparecer las *bes*, se va vaciando la primera pila y se inserta en la segunda pila una *A* por cada *b* leída. Finalmente, al aparecer las *ces* se va vaciando la segunda pila sin alterar la primera. El alfabeto de pila es $\Gamma = \{A\}$.



Ejercicios de la sección 6.6

- ① Sea $\Sigma = \{a, b, c\}$. Diseñar AF2P (Autómatas Finitos con dos pilas) que acepten los siguientes lenguajes. En cada caso, explicar brevemente el plan utilizado en el diseño y presentar el autómata por medio de un grafo de estados.
 - (i) $L = \{a^n b^{2n} c^n : n \geq 0\}$.
 - (ii) $L = \{a^k b^m c^n : 0 \leq k \leq m \leq n\}$.
 - (iii) $L = \{a^k b^m c^n : k \geq m \geq n \geq 0\}$.
 - (iv) $L = \{a^m b^n c^n : m, n \geq 1, m > n\}$.
 - (v) $L = \{a^m b^n c^n : m, n \geq 1, n > m\}$.
 - (vi) $L = \{a^m b^n a^m c^n : m \neq n\}$.
 - (vii) $L = \{a^m b^n a^m c^n : m, n \geq 1\}$.
 - (viii) $L = \{u \in \Sigma^* : \#_a(u) = \#_b(u) = \#_c(u)\}$.
- ② Sea $\Sigma = \{a, b\}$ y $L = \{ww : w \in \Sigma^*\}$. Diseñar un AF2P (Autómata Finito con dos pilas) que acepte el lenguaje L .
- ③ Sea $\Sigma = \{a, b, c, d\}$ y $L = \{a^n b^n c^n d^n : n \geq 0\}$. Diseñar un AF2P (Autómata Finito con dos pilas) que acepte el lenguaje L .

6.7. La tesis de Church-Turing

Si bien la máquina de Turing antecedió en varias décadas a la implementación física de los computadores actuales, ha resultado ser un modelo muy conveniente para representar “lo computable” (lo que es capaz de hacer *cualquier* dispositivo físico de computación secuencial). La declaración conocida como “tesis de Church-Turing” afirma precisamente que la Máquina de Turing es un modelo general de computación secuencial: cualquier procedimiento algorítmico que pueda realizar un ser humano o un computador digital se puede llevar a cabo mediante una Máquina de Turing. Se establece así una conexión intuitiva directa entre máquinas de Turing y algoritmos.

6.7.1. Tesis de Church-Turing. *Todo algoritmo puede ser descrito por medio de una máquina de Turing.*

En su formulación más amplia, la tesis de Church-Turing abarca tanto los algoritmos que producen una salida para cada entrada como aquéllos que no terminan (ingresan en bucles infinitos) para algunas entradas.

Para apreciar su significado y su alcance, hay que enfatizar que la Tesis de Church-Turing no es un enunciado matemático susceptible de demostración, ya que involucra la noción intuitiva (no definida rigurosamente) de *algoritmo*. Por tal razón la tesis no se puede demostrar como un teorema pero sí se podría refutar exhibiendo un procedimiento efectivo, que todo el mundo acepte que es un verdadero algoritmo y que no pueda ser descrito por medio de una máquina de Turing. Pero tal refutación no se ha producido hasta la fecha; de hecho, la experiencia acumulada durante décadas de investigación ha corroborado una y otra vez la tesis de Church-Turing.

Hay dos hechos más que contribuyen a apoyar la tesis:

1. La adición de recursos computacionales a las máquinas de Turing (múltiples pistas o cintas, no determinismo, etc) no incrementa el poder computacional del modelo básico. Esto es un indicativo de que la máquina de Turing no sólo es un modelo extremadamente robusto sino que representa el límite de lo que un dispositivo de computación secuencial puede hacer.
2. Todos los modelos o mecanismos computacionales propuestos para describir formalmente la noción de algoritmo han resultado ser equivalentes a la máquina de Turing, en el sentido de que lo que se puede hacer con ellos también se puede hacer con una MT adecuada, y viceversa. Entre los modelos de computación equivalentes a la máquina de Turing podemos citar:
 - Las funciones parciales recursivas (modelo de Gödel y Kleene, 1936).
 - El cálculo- λ (modelo de Church, 1936).
 - Sistemas de deducción canónica (modelo de Post, 1943).
 - Lógica combinatoria (modelos de Schönfinkel, 1924; Curry, 1929)

- Algoritmos de Markov (modelo de Markov, 1951).
- Las gramáticas irrestrictas (modelo de Chomsky, 1956).
- Las máquinas de registro (modelo de Shepherdson-Sturgis, 1963).

La equivalencia mutua de todos los modelos propuestos refuerza la idea que ellos capturan la esencia de lo que significa “computación efectiva”.

6.8. Codificación de Máquinas de Turing

Toda MT se puede codificar como una secuencia binaria finita, es decir, una secuencia finita de ceros y unos. En esta sección presentaremos un esquema de codificación para todas las MT (modelo estándar) que actúen sobre un alfabeto de entrada $\Sigma = \{s_2, s_3, \dots, s_m\}$ predeterminado. Para definir una máquina de Turing se utilizan los siguientes símbolos:

- (1) Estados disponibles: $q_1, q_2, q_3, q_4, \dots$
 q_1 es siempre el estado inicial y q_2 es el único estado de aceptación.
- (2) Símbolos de cinta disponibles: $s_1, s_2, s_3, \dots, s_m, s_{m+1}, s_{m+2}, \dots, \dots$
 Aquí, $s_1 = \square$ (símbolo que representa una casilla en blanco), $\Sigma = \{s_2, s_3, \dots, s_m\}$ es el alfabeto de entrada común (fijo) de todas las MT, y s_k , con $k \geq m+1$, son los símbolos de cinta disponibles (cada MT utiliza su propia colección finita de símbolos auxiliares).

Cada MT posee únicamente un número finito de estados y un número finito de símbolos de cinta, pero hay una secuencia infinita de símbolos de ambos tipos disponibles para todas las MT.

Una instrucción determinada I en una MT, $\delta(q_i, s_j) = (q_k, s_\ell, D_t)$, se codifica por medio de la cadena binaria

$$01^i 01^j 01^k 01^\ell 01^t 0.$$

Es decir, los estados y los símbolos de cinta se codifican como secuencias de unos, utilizando ceros como separadores. La codificación de una instrucción utiliza exactamente seis ceros separados por secuencias de unos. El exponente final t asume uno de los siguientes valores: 1 (desplazamiento a la derecha, \rightarrow), 2 (desplazamiento a la izquierda, \leftarrow) o 3 (estacionario, $-$). La codificación de una instrucción I se denota como $\langle I \rangle$.

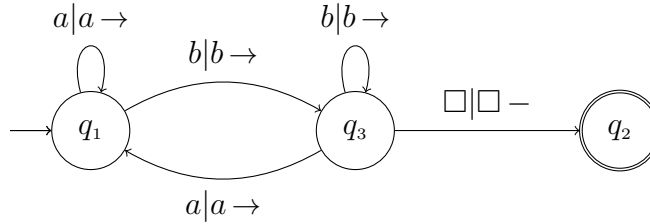
Una MT M está definida completamente por su función de transición, es decir, por su lista de instrucciones: $\{I_1, I_2, \dots, I_r\}$ (un número finito de instrucciones). Una codificación de M , denotada $\langle M \rangle$, se obtiene concatenando las codificaciones de todas sus instrucciones (o transiciones). Más precisamente,

$$\langle M \rangle = \langle I_1 \rangle \langle I_2 \rangle \cdots \langle I_r \rangle 00.$$

En una codificación de M las instrucciones aparecen separadas por 00 y $\langle M \rangle$ termina en 000. Puesto que el orden en que se presentan las transiciones de una MT no es relevante,

una misma MT tiene varias codificaciones diferentes. De hecho, como se puede cambiar arbitrariamente el nombre de los estados (excepto q_1 y q_2), toda máquina de Turing con tres o más estados tiene infinitas codificaciones. Esto no representa ninguna desventaja práctica ni conceptual ya que no se pretende que las codificaciones sean únicas.

Ejemplo Sea $\Sigma = \{a, b\}$ y M la siguiente MT que acepta el lenguaje de todas las cadenas que terminan en b :



Como el alfabeto de entrada es $\Sigma = \{a, b\}$, entonces $s_1 = \square$, $s_2 = a$ y $s_3 = b$. Las cinco instrucciones de M (en orden arbitrario) y sus respectivas codificaciones son:

$I_1 : \delta(q_1, s_2) = (q_1, s_2, \rightarrow).$	Codificación de I_1 :	0101101011010.
$I_2 : \delta(q_1, s_3) = (q_3, s_3, \rightarrow).$	Codificación de I_2 :	01011101110111010.
$I_3 : \delta(q_3, s_2) = (q_1, s_2, \rightarrow).$	Codificación de I_3 :	011101101011010.
$I_4 : \delta(q_3, s_3) = (q_3, s_3, \rightarrow).$	Codificación de I_4 :	0111011101110111010.
$I_5 : \delta(q_3, s_1) = (q_2, s_1, -).$	Codificación de I_5 :	0111010110101110.

Manteniendo este orden, una codificación de M sería $\langle M \rangle = \langle I_1 \rangle \langle I_2 \rangle \langle I_3 \rangle \langle I_4 \rangle \langle I_5 \rangle 00$. Explícitamente,

$$\langle M \rangle = 010110101101001011101110111010011101101011010011101110111010011101011010111000,$$

la cual se puede escribir también como

$$0101^2 0101^2 0100101^3 01^3 01^3 01001^3 01^2 0101^2 01001^3 01^3 01^3 01^3 01001^3 0101^2 0101^3 000.$$

Cambiando el orden de las cinco transiciones de M obtendríamos en total $5! = 120$ codificaciones diferentes para M .

Es claro que no todas las secuencias binarias representan máquinas de Turing: la codificación de una MT no puede comenzar con 1 ni pueden aparecer cuatro ceros consecutivos. Así, las secuencias 01010000110, 0100001110 y 1011010110111010 no codifican ninguna MT. Una cadena binaria que codifique una MT se denomina un *código válido* (o una *codificación válida*) de una MT. Podemos concebir un algoritmo que determine si una secuencia binaria finita codifica o no una MT.

Algoritmo para verificar si una cadena binaria es un código válido de una MT.

Una cadena en $u \in \{0, 1\}^*$ es un código válido de una MT si

- (1) $u \in (01^+01^+01^+01^+0)^+00$. Esta expresión regular representa las concatenaciones de bloques con exactamente seis ceros que separan secuencias de unos. Cada uno de esos bloques contiene cadenas de la forma $01^i01^j01^k01^\ell01^t0$, que representan instrucciones individuales. La cadena u debe entonces terminar en 000 (los dos ceros finales requeridos más el cero proveniente de la última instrucción).
- (2) En cada bloque (instrucción) $01^i01^j01^k01^\ell01^t0$, $t \in \{1, 2, 3\}$. Es decir, la unidad de control solo tiene tres posibles desplazamientos (derecha, izquierda, estacionario).
- (3) No hay dos bloques (instrucciones) que comiencen con la misma subcadena 01^i01^j0 . Es decir, la máquina de Turing es determinista.
- (4) No hay ningún bloque (instrucción) que comience con 01^20 . Es decir, no hay transiciones desde el estado de aceptación.

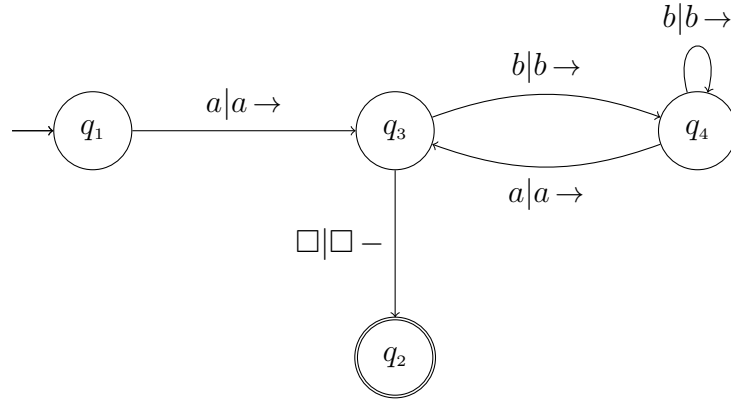
Una MT puede tener muchas codificaciones pero un código válido se puede decodificar y representa una única MT. En otras palabras, la codificación no es única pero la decodificación sí lo es. Si una cadena binaria no es un código válido de una MT, supondremos que codifica una MT sin transiciones que no acepta ninguna cadena, es decir, acepta el lenguaje $\emptyset = \{ \}$. De esta forma, *todas las cadenas binarias representan máquinas de Turing*. Puesto que $\{0, 1\}^*$ se puede ordenar lexicográficamente, podemos entonces hablar de la i -ésima máquina de Turing, la cual denotaremos por M_i . Nótese que en la enumeración M_1, M_2, M_3, \dots , cada MT aparece varias veces (porque al cambiar el orden de las transiciones se obtiene una codificación diferente). Las MT que aceptan el lenguaje \emptyset aparecen infinitas veces en la enumeración.

La enumeración M_1, M_2, M_3, \dots de todas las máquinas de Turing sobre un alfabeto de entrada Σ induce una enumeración de todos los lenguajes Turing-aceptables, a saber, $L_1 = L(M_1)$, $L_2 = L(M_2)$, $L_3 = L(M_3)$, \dots . Como conclusión podemos enunciar el siguiente teorema.

6.8.1 Teorema. Dado Σ , el conjunto de todas las máquinas de Turing con alfabeto de entrada Σ es infinito enumerable. También lo es el conjunto de todos los lenguajes Turing-aceptables.

Ejercicios de la sección 6.8

- ① Sea M la siguiente MT cuyo alfabeto de entrada es $\Sigma = \{a, b\}$:



Determinar el lenguaje aceptado por M y codificar la máquina M siguiendo el esquema presentado en la presente sección ($s_1 = \square$, $s_2 = a$ y $s_3 = b$).

- ② Las siguientes cadenas binarias son códigos válidos de MT que actúan sobre el alfabeto de entrada $\Sigma = \{a, b\}$, siguiendo el esquema de codificación presentado en la presente sección ($s_1 = \square$, $s_2 = a$ y $s_3 = b$). Decodificar las máquinas y determinar en cada caso el lenguaje aceptado.

- (i) 010110111011010011101110111011010011101110 $\xrightarrow{\text{sigue}}$
1111011101001111011011110110100111101011010111000.
- (ii) 0101³0101³0100101²01³01²01001⁴01²01⁵01²01001³01²01⁴01²010 $\xrightarrow{\text{sigue}}$
01⁴01³0101³01001⁵0101²0101³000.

- ③ Escribir las codificaciones de las siguientes MT: M_7 , M_{14} , M_{65} , M_{150} y M_{255} . ¿Cuál es el lenguaje aceptado por tales máquinas?

6.9. Lenguajes que no son Turing-aceptables

Utilizando argumentos de cardinalidad, es posible concluir rápidamente que existen lenguajes que no son Turing-aceptables. Dado un alfabeto Σ , el conjunto de todos los subconjuntos de Σ^* , es decir, $\mathcal{P}(\Sigma^*)$, es el universo de todos los lenguajes sobre Σ . Por el llamado Teorema de Cantor, $\mathcal{P}(\Sigma^*)$ no es enumerable, pero según el Teorema 6.8.1, el conjunto de todos los lenguajes Turing-aceptables sí es enumerable. Por consiguiente, existen infinitos lenguajes que no pueden ser aceptados por ninguna máquina de Turing.

En esta sección exhibiremos un lenguaje concreto que no es Turing-aceptable. Recordemos que para concluir que ciertos lenguajes no son regulares o no son LIC utilizamos razonamientos por contradicción. Aquí también razonaremos por contradicción, aunque el lenguaje definido es completamente artificial. Dado un alfabeto de entrada $\Sigma = \{s_2, \dots, s_m\}$, las cadenas de Σ^* se pueden ordenar lexicográficamente estableciendo primero un orden arbitrario (pero fijo) para los símbolos de Σ , por ejemplo, $s_2 < s_3 < \dots < s_m$. Obtenemos así una enumeración w_1, w_2, w_3, \dots de todas las cadenas de Σ^* . Esto nos permite hablar de

la i -ésima cadena de entrada, w_i . La interacción entre estas dos enumeraciones diferentes w_1, w_2, w_3, \dots (entradas) y M_1, M_2, M_3, \dots (máquinas de Turing) produce un lenguaje que no es Turing-aceptable.

6.9.1 Teorema. Sea Σ un alfabeto dado. El lenguaje

$$L = \{w_i \in \Sigma^* : w_i \text{ no es aceptada por } M_i\}$$

no es Turing-aceptable.

Demostración. Razonamos por contradicción (o reducción al absurdo). Si L fuera Turing-aceptable, existiría una MT M_k (para algún $k \geq 1$), con respecto a la enumeración de máquinas de Turing descrita en la sección 6.8, tal que $L = L(M_k)$. Se tendría entonces

$$w_k \in L \implies w_k \text{ no es aceptada por } M_k \implies w_k \notin L(M_k) = L.$$

$$w_k \notin L \implies w_k \text{ es aceptada por } M_k \implies w_k \in L(M_k) \implies w_k \in L.$$

Por lo tanto, $w_k \in L \iff w_k \notin L$, lo cual es una contradicción. \square

El lenguaje L del Teorema 6.9.1 se denota como L_d y se denomina *lenguaje de diagonalización*; esta terminología proviene del hecho de que la demostración contiene un “argumento diagonal” que se puede visualizar de la siguiente manera. En la siguiente matriz infinita hay un 1 en la posición (i, j) si M_i acepta a w_j , y un 0 si M_i no acepta a w_j :

	w_1	w_2	w_3	w_4	\dots
M_1	0	0	1	0	\dots
M_2	1	0	0	0	\dots
M_3	0	1	1	0	\dots
\vdots	\vdots				

Las filas de la matriz representan *todos* los lenguajes Turing-aceptables. El lenguaje L_d se construye “complementando” la diagonal (cambiando ceros por unos y unos por ceros). Tal lenguaje no puede estar en esa lista y, por lo tanto, no puede ser Turing-aceptable. Se trata del mismo argumento utilizado para demostrar que el conjunto de todas las sucesiones infinitas de ceros y unos no es enumerable.

6.10. Máquina de Turing universal

En la sección 6.8 se presentó un esquema de codificación para todas las máquinas de Turing con un alfabeto de entrada predeterminado $\Sigma = \{s_2, s_3, \dots, s_m\}$ (El símbolo s_1 ha sido reservado para \square). Es necesario ahora codificar todas las cadenas de Σ^* para lo cual utilizaremos un esquema muy simple, compatible con la codificación de MT: el símbolo s_i se codifica como 1^i y una cadena $s_{i_1}s_{i_2}\dots s_{i_k}$ se codifica como $1^{i_1}01^{i_2}0\dots 01^{i_k}0$.

Ejemplo Sea $\Sigma = \{a, b, c, d\}$. Podemos asignar los símbolos así: $s_2 = a$, $s_3 = b$, $s_4 = c$, y $s_5 = d$. La cadena $dbabc$ se codifica como $1^501^301^201^301^40$.

Una codificación válida de una cadena en Σ^* debe constar de bloques de unos separados por ceros individuales, es decir, deber ser de la forma $1^{i_1}01^{i_2}0 \cdots 01^{i_k}0$, con la restricción $2 \leq i_j \leq m$ para todo exponente i_j . En otras palabras, una cadena binaria u es una codificación válida si $u \in (11^+0)^+$. A diferencia de la codificación de máquinas de Turing, la codificación de cadenas en este esquema es única (una vez que a cada símbolo de Σ le sea asignado un símbolo s_i). La codificación de una cadena $w \in \Sigma^*$ se denota como $\langle w \rangle$.

Una máquina de Turing universal M_U simula el funcionamiento de todas las MT (sobre el alfabeto de entrada Σ). M_U está diseñada para procesar cadenas binarias de la forma $\langle M \rangle \langle w \rangle$, siendo M una MT determinada y w una cadena de entrada perteneciente a Σ^* .

M_U es una MT con tres cintas cuyo alfabeto de cinta es $\{0, 1\}$. M_U utiliza el símbolo externo \square (que representa una casilla en blanco) como se hace en el modelo estándar. Hay que aclarar que las demás máquinas de Turing utilizan el símbolo blanco codificado como 1, por la asignación previa $s_1 = \square$. Las entradas que lee M_U son cadenas binarias que se escriben en la primera cinta (cinta de entrada); las otras dos cintas están inicialmente en blanco. M_U no escribe nunca sobre la primera cinta (que es entonces una cinta de lectura).

A continuación se detalla el funcionamiento de la máquina M_U , invocando apropiadamente la tesis de Church-Turing:

1. M_U copia la entrada en la cinta 2 y verifica que sea de la forma $\langle M \rangle \langle w \rangle$. Puesto que el código de una MT M termina en 000, en la cadena $\langle M \rangle \langle w \rangle$ aparecen tres ceros consecutivos únicamente en el sitio que separa los códigos de M y w . Chequea que $\langle M \rangle$ sea una codificación válida de una MT, y $\langle w \rangle$ sea una codificación válida de una cadena de Σ^* . Si la entrada no es de la forma $\langle M \rangle \langle w \rangle$, M_U se detiene sin aceptar.
2. Si la entrada es de la forma $\langle M \rangle \langle w \rangle$, M_U copia la cadena w , o sea, la decodificación de $\langle w \rangle$, en la segunda cinta, borrando todo lo que haya escrito previamente en esa cinta. M_U utilizará la segunda cinta para simular el procesamiento que hace M con entrada w .
3. La cadena 1, que representa el estado inicial q_1 , se coloca en la tercera cinta. La unidad de control pasa a escanear el primer símbolo de cada cadena binaria, en cada una de las tres cintas. La tercera cinta se usa para almacenar el estado actual de M , también codificado: q_1 se codifica como 1, q_2 se codifica como 11, q_3 como 111, etc.
4. M_U utiliza la información de las cintas 2 y 3 para buscar en la cinta 1 la transición que sea aplicable. Si encuentra una transición aplicable, M_U simula en la cinta 2 lo que haría M y cambia el estado escrito en la cinta 3. La simulación continúa de esta forma, si hay transiciones aplicables. Después de ejecutar una transición, la unidad de control regresa, en la primera y tercera cintas, al primer símbolo de la cadena.

Si al procesar una entrada w , M_U se detiene en el único estado de aceptación de M (que es q_2 y está codificado como 11), entonces la cadena w será aceptada. Por

consiguiente, M_U tiene también un único estado de aceptación, q_2 , que es el mismo estado de aceptación de cualquier otra MT.

Si durante la simulación M_U no encuentra una transición aplicable o se detiene en un estado que no es de aceptación, se detiene sin aceptar, como lo haría M .

Se tiene entonces que M_U acepta la entrada $\langle M \rangle \langle w \rangle$ si y solamente si M acepta a w . De modo que el lenguaje aceptado por la máquina de Turing universal M_U se puede describir explícitamente; este lenguaje se denomina corrientemente el *lenguaje universal* y se denota con L_U :

$$L_U = \{ \langle M \rangle \langle w \rangle : \text{la MT } M \text{ acepta la cadena } w \in \Sigma^* \}.$$

El lenguaje universal L_U es, por consiguiente, un lenguaje Turing-aceptable. Sin embargo, este lenguaje no es Turing-decidible, como se establece en el siguiente teorema.

6.10.1 Teorema. El lenguaje universal, $L_U = \{ \langle M \rangle \langle w \rangle : M \text{ acepta a } w \}$, es Turing-aceptable pero no es Turing-decidible.

Demostración. Para demostrar que L_U no es Turing-decidible razonamos por contradicción: suponemos que existe una MT \mathcal{M} que procesa todas las entradas $\langle M \rangle \langle w \rangle$ y se detiene siempre en un estado de aceptación (si M acepta a w) o en uno de rechazo (si M no acepta a w). Esta suposición permitirá construir una MT \mathcal{M}' que acepte el lenguaje L_d ,

$$L_d = \{ w_i \in \Sigma^* : w_i \text{ no es aceptada por } M_i \},$$

de lo cual se deduciría que L_d es Turing-aceptable, contradiciendo así la conclusión del Teorema 6.9.1.

Con una entrada $w \in \Sigma^*$, la máquina \mathcal{M}' procede así: examina consecutivamente las cadenas w_1, w_2, w_3, \dots , en el orden lexicográfico (utilizando como subrutina la función Turing-computable presentada en la sección 6.4) hasta que encuentra un k tal que $w = w_k$. Luego simula (o invoca) a \mathcal{M} con entrada $\langle M_k \rangle \langle w_k \rangle$: si \mathcal{M} se detiene en un estado que no es de aceptación, significa que M_k no acepta a w_k ; en tal caso, \mathcal{M}' acepta la entrada $w = w_k$. Por consiguiente, $L(\mathcal{M}') = L_d$. Esto significa que L_d es Turing-aceptable, lo cual contradice el Teorema 6.9.1. \square

Este teorema implica que existen lenguajes que pueden ser aceptados por MT específicas pero en cualquier MT que los acepte habrá cómputos que nunca terminan (obviamente, los cómputos de las cadenas aceptadas siempre terminan). De este hecho extraemos la siguiente importante conclusión: los cómputos interminables, o bucles infinitos, son inherentes a la computación y no se pueden eliminar de la Teoría de la Computación.