Ejercicios de Introducción a la Teoría de la Computación

Edgar Santiago Ochoa
eochoaq@unal.edu.co

Mateo Andrés Manosalva
mmanosalva@unal.edu.co

Contenido

1 | Alfabetos, lenguajes y cadenas

2 | Lenguajes Regulares y Autómatas Finitos

Alfabetos, lenguajes y cadenas

Este manual de solución de las notas de introducción a la teoría de la computación tiene como objetivo ser de ayuda para estudiantes que se encuentren viendo la materia en futuros semestres, a la fecha (2023-1) no hay ningún manual de solución de los ejercicios de este curso y nos pareció importante poder dar un poco de ayuda o soporte a las personas que en el futuro vean el curso, todo ha sido separado en secciones para que sea más sencillo encontrar los ejercicios específicos que el estudiante pueda necesitar.

Las soluciones que presentemos no necesariamente son correctas pero trabajaremos para que lo sean y ante cualquier error nos pueden escribir a los correos que están en la portada.

1.1 Reflexión de una cadena

Punto 1: Dar una definicion recursiva de u^R.

Primero consideramos la cadena vacía, es claro que

- $\lambda^R = \lambda$

Esto por definición de reflexión de cadenas, ahora consideremos la cadena ua, por definición de reflexión se tiene que:

- $(ua)^R = au^R$

Punto 2: Generalizar la propiedad $(uv)^R = v^R u^R$ a la concatenacion de n cadenas, $n \geq 2$.

Para generalizar la propiedad consideramos la cadena:

$$(u_1 u_2 ... u_n)$$

luego:

$$\begin{aligned}(u_1 u_2 ... u_n)^R &= (u_{n-1} u_n)^R (u_1 u_2 ... u_{n-2})^R \\ &= u_n^R u_{n-1}^R (u_1 u_2 ... u_{n-2})^R\end{aligned}$$

Podemos repetir este proceso 2 a 2 paso a paso y llegamos a que:

$$(u_1 u_2 ... u_n)^R = u_n^R u_{n-1}^R ... u_2^R u_1^R$$

Uno podría pensar algo como $(u_1u_2...u_n)^R = u_nu_{n-1}...u_2u_1$, pero esta generalización falla ya que recordemos que los u_i son cadenas y no elementos del lenguaje, es decir pueden representar una cadena de varios elementos y se hace necesario reflejarlos también, por ejemplo:

Sea $\Sigma = \{a, b, c\}$ y considere $u_1 = ab$ y $u_2 = ac$, $v = u_1u_2 = abac$, entonces:

$$v^R = caba$$

Mientras que $(u_1u_2)^R = acab$ si tomamos esa definición, note que $v^R \neq (u_1u_2)^R$, por tanto se hace evidente porqué esa definición no sirve.

♦

1.2 Orden lexicográfico entre cadenas

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

(1) $u = dbd$

Como en el alfabeto solo hay a,b,c,d y $a < b < c < d$, entonces la cadena que buscamos es $v = dca$ ya que e no existe en el alfabeto y el siguiente a b es c y cambiamos la d por a ya que necesitamos la cadena que sigue y $dca < dcb < dcc < dcd$.

Similarmente analizamos los siguientes ejercicios:

(2) $u = acbd$ aplicando el análisis anterior encontramos $v = acca$

(3) $u = dabcdd$, nuevamente encontramos $v = dabdaa$

(4) $dcaddd$, finalmente $v = dcbaaa$

♦

1.3 Concatenación de lenguajes

Punto 1: Dar un ejemplo de un alfabeto Σ y dos lenguajes diferentes A, B sobre Σ tales que $AB = BA$.

Considere $\Sigma = \{a, b\}$ los lenguajes $A =\emptyset$ y $B = \{a\}$, por definición de concatenación de lenguajes:

$$AB = \emptyset = BA$$

Punto 2: Dar un ejemplo de un alfabeto Σ y tres lenguajes A, B, C sobre Σ diferentes entre si, tales que $A \cdot (B \cap C) = A \cdot B \cap A \cdot C$.

Sea $\Sigma = \{a, b, c\}$, considere los lenguajes $A = \{a\}$, $B = \{b\}$ y $C = \{c\}$, entonces:

$$
\begin{aligned}
A \cdot (B \cap C) &= \emptyset && \longleftarrow (A \cdot \emptyset = \emptyset) \\
&= \{ab\} \cap \{bc\}
\end{aligned}
$$

La intersección es vacía porque ab como cadena es diferente de bc

Punto 3: Sea Σ un alfabeto dado. Se ha demostrado en esta seccion, mediante un contrajemplo, que la igualdad $A \cdot (B \cap C) = A \cdot B \cap A \cdot C$ no es una identidad valida para todos los lenguajes $A, B, C \subseteq \Sigma^*$. Demostrar, sin embargo, que la contenencia

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

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

Demostración. Sea $u \in A \cdot (B \cap C)$, entonces $u = a_i x_i$ para algún $a_i \in A$ y $x_i \in B \cap C$, luego como $x_i \in B \cap C$ $x_i \in B$ y $x_i \in C$, por tanto $u \in A \cdot B$ y $u \in A \cdot C$, así $u \in A \cdot B \cap A \cdot C$, concluimos que $A \cdot (B \cap C) \subseteq A \cdot B \cap A \cdot C$

□

♦

1.4 La clausura de Kleene de un lenguaje

Sean A, B lenguajes sobre Σ, es decir $A, B \subseteq \Sigma^*$. Explicar por que las siguientes igualdades no son validas en general:
Punto 1: $(A \cup B)^* = A^* \cup B^*$
Esto ocurre ya que la clausura de Kleene son todas las posibles concatenaciones de elementos de un lenguaje, por tanto unir las de dos lenguajes contempla las clausuras de ambos por separado y luego las une, mientras que unir los lenguajes y luego hacer la clausura contempla todas las posibles concatenaciones de elementos que se encontraban en ambos lenguajes, de hecho se podría ver con un argumento similar que $(A \cup B)^* \supset A^* \cup B^*$.

Punto 2: $(A \cup B)^* = A^* \cup B^* \cup A^*B^* \cup B^*A^*$
En este caso ocurre algo similar y es que faltan cadenas de $(A \cup B)^*$, por ejemplo considere $A = \{a\}$ y $B = \{b\}$, entonces por ejemplo es imposible obtener la cadena $(ba)^2$ a traves de $A^* \cup B^* \cup A^*B^* \cup B^*A^*$, estos mismos lenguajes sirven para darnos cuenta en el punto 1 que $(A \cup B)^* \supset A^* \cup B^*$, es este caso es lo mismo, $(A \cup B)^* \supset A^* \cup B^* \cup A^*B^* \cup B^*A^*$.

Para ver la falsedad de estas afirmaciones recomendamos siempre usar lenguajes pequeños (con pocas cadenas) de tal manera que no nos gastemos mucho tiempo haciendo cuentas.

♦

1.5 Reflexión o inverso de un lenguaje

Los ejercicios de esta sección son más bien opcionales ya que Korgi no suele evaluar demostraciones en este curso, sin embargo para las personas que quizá se les dificulte demostrar estas afirmaciones y esté interesado en aprender se hace la solución de los mismos.

Punto 1: Demostrar las propiedades 2,3,4 y 6 de la reflexion de cadenas.

Demostración. Usando la definición de unión y de reflexión de un lenguaje tenemos que:

$$\begin{aligned}(A \cup B)^R &= \{u^R : u \in A \text{ o } u \in B\} \\ &= \{x^R : x \in A\} \cup \{y^R : y \in B\} \\ &= A^R \cup B^R\end{aligned}$$

□

Demostración. En este caso es similar el argumento solo que cambiamos la ”o” por una ”y” ya que es una intersección y por tanto los elementos deben estar en ambos conjuntos:

$$\begin{aligned}(A \cap B)^R &= \{u^R : u \in A \text{ y } u \in B\} \\ &= \{x^R : x \in A\} \cap \{y^R : y \in B\} \\ &= A^R \cap B^R\end{aligned}$$

□

Demostración. Usando la definición de reflexión de un lenguaje tenemos que:

$$(A^R)^R = \{(u^R)^R : u \in A\}$$

Es decir es la reflexión de la reflexión de todas las cadenas de A, y nosotros ya sabemos que la reflexión de la reflexión de una cadena es la misma cadena luego:

$$(A^R)^R = \{u : u \in A\} = A$$

□

Demostración. En este caso seguiremos el mismo modelo de prueba que se usa en las notas de clase para la propiedad 5

$$\begin{aligned}x \in (A^+)^R &\iff x = u^R, \text{ donde } u \in A^+ \\ &\iff x = (u_1 \cdot u_2 \cdots u_n)^R, \text{ donde los } u_i \in A, n \geq 1 \\ &\iff x = u_n^R \cdot u_2^R \cdots u_1^R, \text{ donde los } u_i \in A, n \geq 1 \\ &\iff x \in \left(A^R\right)^+.\end{aligned}$$

□

Punto 2: ¿Se pueden generalizar las propiedades 2 y 3 anteriores para uniones e intersecciones de un número arbitrario de conjuntos, respectivamente?

CLARAmente se pueden generalizar las propiedades 2 y 3 ya que la unión y la intersección se comportan bien 2 a 2, es decir si tenemos que calcular la reflexión de la unión de n conjuntos, podemos hacerlo con los primeros dos y luego con los siguientes dos y así hasta acabar o que nos quede solo uno y pues en ese caso calculamos la reflexión y unimos todo o en el otro caso intersectamos.

$$\begin{aligned}\left(\bigcup_{i\geq 0} A_i\right)^R &= (A_1 \cup A_2)^R \cup (A_3 \cup A_4 \cup A_5 \cup \ldots)^R \\ &= A_1^R \cup A_2^R \cup (A_3 \cup A_4)^R \cup (A_5 \cup A_6 \cup A_7 \cup \ldots)^R\end{aligned}$$

Y continuando así llegamos a:

$$\left(\bigcup_{i\geq 0} A_i\right)^R = A_1^R \cup A_2^R \cup A_3^R \cup \ldots \cup A_n^R \ldots$$

De manera análoga se ve la intersección.

♦

Lenguajes Regulares y Autómatas Finitos

En este capítulo comenzamos la segunda parte de las notas de clase, en estas secciones resaltamos que no hay una única solución a los ejercicios y que algunas de las soluciones pueden llegar a ser redundantes, sin embargo son funcionales y esto es lo que más nos interesa.

2.1 Expresiones regulares

Punto 1: Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{a, b\}$:

- Lenguaje de todas las cadena que comienzan con el símbolo b y terminan con el símbolo a.

  La solución más evidente es la siguiente: $b(a \cup b)^* a$, con $(a \cup b)^*$ consideramos todas las cadenas y lo que hacemos es forzar que las cadenas comiencen con b y terminen en a concatenando.

- Lenguaje de todas las cadenas de longitud impar.

  Sabemos que para generar todas las cadenas de longitud par usamos $(aa \cup bb \cup ab \cup ba)^* = ((a \cup b)(a \cup b))^*$, luego para generar las impares debemos considerar 4 casos y unirlos:

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

  Esto convierte las cadenas pares en impares siempre y considera los casos en que comience por a o por b o termine por a o b (se puede llegar a una solución mejor quizá).

- Lenguaje de todas las cadenas que tienen un numero impar de aes.

  Sabemos que $(b^* a b^* a b^*)^*$ genera todas las cadenas con un número par (mayor que 0) de aes, luego usando este hecho construimos:

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

  y acabamos.

- Lenguaje de todas las cadenas en las que el numero de bes es un múltiplo ≥ 0 de 3.

  Sabemos generar las cadenas de a y b que contienen un número par de a o b, la solución está propuesta en las notas de clase, luego cambiamos un poco la expresión de esta forma:

$$a^*(a^*ba^*ba^*ba^*)^*$$

La expresión $a^*(ba^*ba^*b)^*a^*$ también es una solución.

- Lenguaje de todas las cadenas que no comienzan con la subcadena ba ni terminan en b.

  Para este caso las cadenas pueden comenzar por a, ab, b^2, a^2, luego obtenemos la expresión:

$$(a \cup ab \cup b^2 \cup a^2)(a \cup b)^*a$$

  Ya que tampoco pueden acabar en b, ahora no falta notas que por agregar esta a al final y la expresión $(a \cup ab \cup b^2 \cup a^2)$ es imposible obtener la cadena vacía y la cadena a, pues las añadimos y nos queda finalmente:

$$(a \cup ab \cup b^2 \cup a^2)(a \cup b)^*a \cup a \cup \lambda$$

- Lenguaje de todas las cadenas que no contienen la subcadena bba.

  Para la expresión regular en este caso notemos que toda cadena tiene bloques de la forma ba donde estos van intercalados con un numero de a arbitrarias así obtenemos la expresión:

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

  Observe que las b están restringidas ya que para $b \geq 2$ las cadenas de este estilo no pueden tener una a luego de la cantidad arbitraria de $b$es, pero esta expresión no contempla las cadenas del estilo $bb\dots b$, para esto basta concatenar estas cadenas al final, obteniendo finalmente la expresión:

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

Punto 2: Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{0, 1, 2\}$:

- Lenguaje de todas las cadenas que comienzan con 2 y terminan con 1.

  Al igual que en el primer ítem del punto anterior, lo mas natural es forzar que la cadena empiece en con 2 y termine en 1, concatenando respectivamente obtenemos la expresión $2(0 \cup 1 \cup 2)^*1$.

- Lenguaje de todas las cadenas que no comienzan con 2 ni terminan en 1.

  Similar a la construcción anterior podemos forzar a que las cadenas no empiecen con 2 ni terminen con 1 concatenando $(0 \cup 1)$ y $(0 \cup 2)$ respectivamente. De esta forma obtenemos la expresión:

$$(0 \cup 1)(0 \cup 1 \cup 2)^*(0 \cup 2)$$

Note que en el lenguaje propuesto las cadenas λ y 0 también cumplen la condición, pero es imposible generarlas por medio de la expresión dada. Afortunadamente arreglar esto es sencillo ya que podemos agregarlas por medio de uniones, obteniendo así:

$$(0 \cup 1)(0 \cup 1 \cup 2)^*(0 \cup 2) \cup \lambda \cup 0$$

- Lenguaje de todas las cadenas que tienen exactamente dos ceros.

  Nuevamente la forma mas natural de construir la expresión que represente al lenguaje es forzando que aparezcan solo 2 ceros, tenga en cuenta que los ceros pueden estar en cualquier posición y por tanto la expresión es la siguiente $(1 \cup 2)^*0(1 \cup 2)^*0(1 \cup 2)^*$.

- Lenguaje de todas las cadenas que tienen un numero par de símbolos.

  Ya sabemos como generar los bloques de dos elementos de un lenguaje, para este caso $(0 \cup 1 \cup 2)(0 \cup 1 \cup 2) = (0 \cup 1 \cup 2)^2$, luego de esto basta concatenar estos bloques de todas la formas posibles, obteniendo así la expresión:

$$\left((0 \cup 1 \cup 2)^2\right)^*$$

- Lenguaje de todas las cadenas que tienen un numero impar de símbolos.

  Usando la expresión del ítem anterior, si concatenamos al final 0,1 o 2 obtenemos las cadenas de longitud impar, es decir:

$$\left((0 \cup 1 \cup 2)^2\right)^* (0 \cup 1 \cup 2)$$

- Lenguaje de todas las cadenas que no contienen dos unos consecutivos.

  Como no pueden aparecer dos unos consecutivos, las cadenas contienen bloques de la forma $(0 \cup 2)1(0 \cup 2)$, y junto a ellas cantidades arbitrarias de ceros y dos alternados:

$$(0 \cup 2 \cup (0 \cup 2)1(0 \cup 2))^*$$

  Observe que si bien esta expresión nos da múltiples cadenas aun hay varias que no genera. Por ejemplo no genera cadenas que empiecen o terminen en 1. Esto podemos agregarlo concatenado $1 \cup \lambda$ al inicio y final de la expresión:

$$(1 \cup \lambda)(0 \cup 2 \cup (0 \cup 2)1(0 \cup 2))^+(1 \cup \lambda)$$

  La cadena λ es de vital importancia en la expresión ya que esta nos permite concatenar sin perder las cadenas que ya teníamos previamente. Además note que en la expresión cambiamos la $*$ por un $+$, esto se debe a que si no realizamos este cambio generaríamos la cadena 11 y esta no cumple los criterios del lenguaje, por ultimo las cadenas λ y 1 cumplen las condiciones, mas no pueden ser generadas por lo que solo queda agregarlas y así obtener la expresión final:

$$(1 \cup \lambda)(0 \cup 2 \cup (0 \cup 2)1(0 \cup 2))^+(1 \cup \lambda) \cup \lambda \cup 1$$

Punto 3: Encontrar expresiones regulares para los siguientes lenguajes definidos sobre el alfabeto $\Sigma = \{0,1\}$:

- Lenguaje de todas las cadenas que tienen por lo menos un 0 y por lo menos un 1.

  Para que una cadena tenga al menos un 0 y un 1 debe ser mínimo un bloque 01 o un bloque 10, luego simplemente fijamos esas dos posibilidades para que la solución sea $(0 \cup 1)^*(01 \cup 10)(0 \cup 1)^*$.

- Lenguaje de todas las cadenas que no contienen tres ceros consecutivos.

  La condición nos indica que en las cadenas solo pueden haber uno o dos ceros consecutivos, es decir las forman bloque de la forma 01 o 001, luego podemos generar la expresión:

  $$(1 \cup 01 \cup 001)^*$$

  Note que esta no contempla cadenas que terminen en uno o dos ceros, pero esto lo podemos arreglar fácilmente concatenando lo necesario:

  $$(1 \cup 01 \cup 001)^*(\lambda \cup 0 \cup 00)$$

- Lenguaje de todas las cadenas cuya longitud es ≥ 4.

  Para esta solución simplemente consideremos las cadenas de longitud 4 es decir las que son generadas por $(0 \cup 1)^4$, luego como estas son las mínimas cadenas que acepta el lenguaje solo queda hacer que aparezcan las demás posibilidades así $(0 \cup 1)^4(0 \cup 1)^*$.

- Lenguaje de todas las cadenas cuya longitud es ≥ 5 y cuyo quinto símbolo, de izquierda a derecha, es un 1.

  Note que al forzar que el quinto símbolo de izquierda a derecha sea un 1 en todas las cadenas básicamente podemos rehusar la solución anterior forzando la condición de esta forma $(0 \cup 1)^4 1 (0 \cup 1)^*$.

- Lenguaje de todas las cadenas que no terminan en 01.

  Si la cadena no puede terminar en 01 forzosamente tiene que terminar en $00, 10$ o 11, forzando estas obtenemos:

  $$(0 \cup 1)^*(00 \cup 10 \cup 11)$$

  Observe que esta expresión solo genera cadenas de longitud ≥ 2 pero las cadenas $\lambda, 0$ y 1 cumplen la condición, entonces:

  $$(0 \cup 1)^*(00 \cup 10 \cup 11) \cup \lambda \cup 0 \cup 1$$

  De esta forma terminamos.

- Lenguaje de todas las cadenas de longitud par ≥ 2 formadas por ceros y unos alternados.

  Como son cadenas de longitud par, pueden ser formadas por bloques de la forma 01 o de la forma 10 luego la solución luce de esta forma $(01)^+ \cup (10)^+$. Note que usamos $+$ ya que la cadena λ no es aceptada en este lenguaje.

- Lenguaje de todas las cadenas de longitud ≥ 2 formadas por ceros y unos alternados.

  Como en el ejercicio anterior ya construimos las cadenas pares, solo nos queda construir todas las impares. Esto lo logramos por medio de concatenar un elemento mas a las expresiones que ya tenemos. La solución luce de la siguiente forma:

$$(1 \cup \lambda)(01)^+ \cup (0 \cup \lambda)(10)^+$$

- Lenguaje de todas las cadenas que no contienen dos ceros consecutivos ni dos unos consecutivos.

  Note que si no pueden haber dos ceros seguidos ni dos unos seguidos, los ceros y los unos deben de ir alternados forzosamente, es decir que la solución es la misma que la del ejercicio previo, exceptuando un detalle:

$$(1 \cup \lambda)(01)^* \cup (0 \cup \lambda)(10)^*$$

  Observe que cambiamos el $+$ por una $*$, esto se debe a que las cadenas $\lambda, 0$ y 1 si son validas en este lenguaje.

- 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.

  Como la cadena tiene que ser impar la de mínima aceptación es la cadena 1. Ahora note que como nos restringimos a que en la posición impar siempre hayan unos, la cadena solo va a tener bloques de la forma 01 o 11. Así la expresión seria $1(01 \cup 11)^*$.

- Lenguaje de todas las cadenas cuya longitud es un múltiplo de tres.

  Para generar cadenas cuya longitud es un múltiplo de 3, necesitamos todos los bloques de longitud 3 y posteriormente los concatenamos de todas las formas posibles, es decir tenemos la expresión $\left((0 \cup 1)^3\right)^*$. Recuerde que 0 es múltiplo de 3 por eso usamos el $*$ para asegurar la cadena λ.

- Lenguaje de todas las cadenas que no contienen cuatro ceros consecutivos.

  Esta expresión sigue un análisis muy similar al de las cadenas donde no podían haber tres ceros consecutivos, de esta forma solo falta agregar los bloques 0001 y 000 respectivamente a la expresión que habíamos obtenido:

$$(1 \cup 01 \cup 001 \cup 0001)^*(\lambda \cup 0 \cup 00 \cup 000)$$

- Lenguaje de todas las cadenas que no comienzan con 00 ni terminan en 11.

  Observe que la cadena tiene que empezar por 1 o 01 y de forma similar tiene que acabar en 0 o en 01. Forzando estos símbolos obtenemos:

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

  Ahora como usualmente ha ocurrido a lo largo de esta sección, al forzar cadenas en la expresión, no generamos cadenas que si son aceptadas dentro del lenguaje, pero basta con simplemente agregarlas:

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

- Lenguaje de todas las cadenas que no contienen la subcadena 101.

  Para que no contengan la subcadena 101 note que se debe forzar que en todas las expresiones aparezcan al menos dos ceros entre dos unos, las cadenas de este estilo se consiguen por medio de la expresión:

$$(1 \cup 00^+)^*$$

  Uno podría verse tentado en pensar que esta es la solución, pero observe que esta expresión no contempla cadenas que empiecen por 01 y que son totalmente validas, además tampoco contempla cadenas que terminen en un solo cero:

$$(01 \cup \lambda)(1 \cup 00^+)^*(0 \cup \lambda)$$

  Luego de este arreglo si podemos asegurar que están todas las cadenas.

Punto 4: Sea $\Sigma = \{a, b\}$. Encontrar una expresión regular que represente el lenguaje de todas las cadenas que tienen un numero para ≥ 0 de aes y un numero par ≥ 0 de bes.

Para este ejercicio no logramos hallar la expresión por mera inspección, pero para no dejar un mal sabor de boca mostraremos la expresión obtenida usando uno de los procedimientos presentados en secciones futuras:

$$[a^2 \cup b^2 \cup (ab \cup ba)(a^2 \cup b^2)^*(ba \cup ab)]^*$$

♦

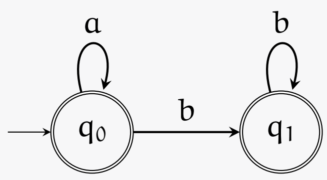
2.2 Diseño de autómatas

Al igual que en la sección anterior las soluciones de este capitulo no son únicas y puede que algunas sean redundantes, además otra aclaración de vital importancia es que todos los autómatas presentados no muestran sus estados limbo, es decir presentaremos AFD simplificados.

Punto 1: Sea $\Sigma = \{a, b\}$. Diseñar AFD que acepten los siguientes lenguajes:

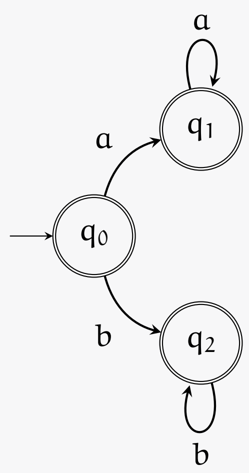
- a^*b^*

Como la cadena mínima es λ entonces el estado inicial tiene que ser de aceptación, luego como también se aceptan $a$es arbitrarias estas pueden ser aceptadas por medio de un bucle. Apenas aparezca una b el autómata cambiara de estado pero ese seria también de aceptación, incluyendo un bucle para las $b$es arbitrarias:



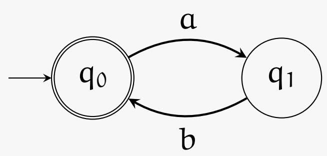
- $a^* \cup b^*$

Nuevamente el estado inicial es de aceptación ya que λ pertenece a el lenguaje, luego basta con tomar dos caminos para el caso donde sean cadenas de $a$es y el de cadenas de $b$es:



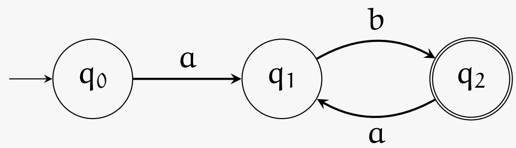
- $(ab)^*$

Note que todas las cadenas de este lenguaje son de la forma $ab \dots ab$, es decir siempre son bloques ab y todas las cadenas empiezan en a y terminan en b:



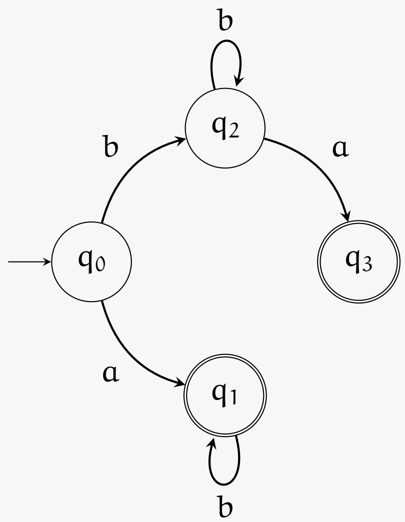
- $(ab)^+$

  Bastante similar a la anterior excepto que la cadena mínima aceptada es ab debido al $+$, así que forzamos esa cadena:

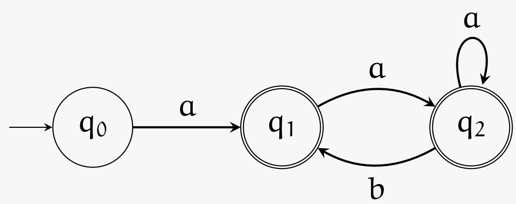


- $ab^* \cup b^*a$

  Note que debido a la expresión se forman dos caminos, uno son las cadenas que empiezan por a y luego tienen una cantidad de $b$es arbitrarias. El otro son aquellas que comienzan por un numero de $b$es arbitrarias pero están forzadas a terminar en a para ser aceptadas:

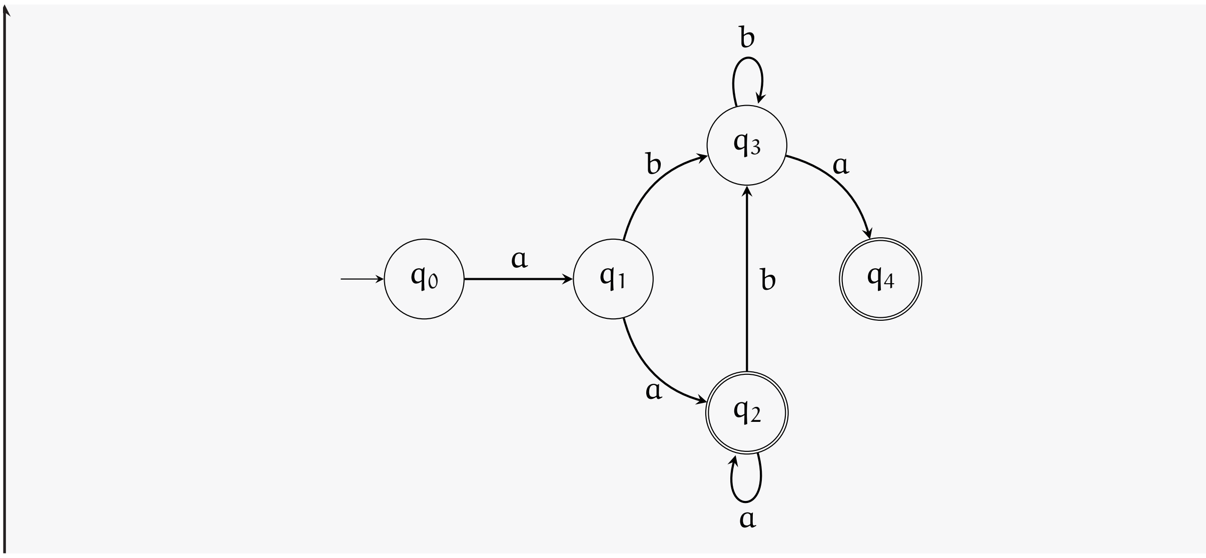


- $a(a \cup ab)^*$ Observe que las cadena mínima de este lenguaje tiene una sola a así que el estado inicial no es de aceptación, luego note que en ambos casos de la $*$ el elemento es una a, por lo que forzamos dos $a$es y posteriormente realizamos respectivamente el bucle de $a$es y de abes:



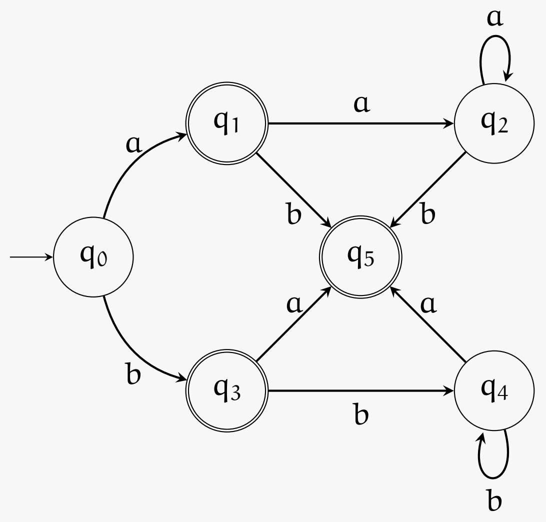
- a^+b^*a. Un AFD que acepte este lenguaje requiere como mínimo 5 estados mas un estado limbo.

  Observe que la cadena mínima es aa entonces debemos forzar ese camino. Ahora note que en ese camino podemos generar el bucle para a^+. Note que hay dos posibilidades para las $b$es, o inicia la cadena con solo una a y luego tiene una cantidad arbitraria de $b$es o empieza con $a$es arbitrarias y luego tiene $b$es arbitrarias, pero note que siempre que aparece al menos una b la cadena siempre acaba en a:



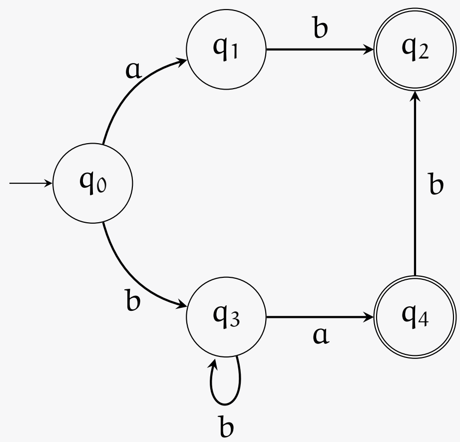
- $a^*b \cup b^*a$. Un AFD que acepte este lenguaje requiere como mínimo 6 estados mas un estado limbo.

  Primero notemos que las cadenas mínimas son a y b, así que debemos forzar dos caminos que las acepten. Ahora observe que si sale una a existen dos posibilidades, es seguida inmediatamente de una b o salen $a$es arbitrarias y luego si una b. Note que este análisis también es valido respectivamente con b:



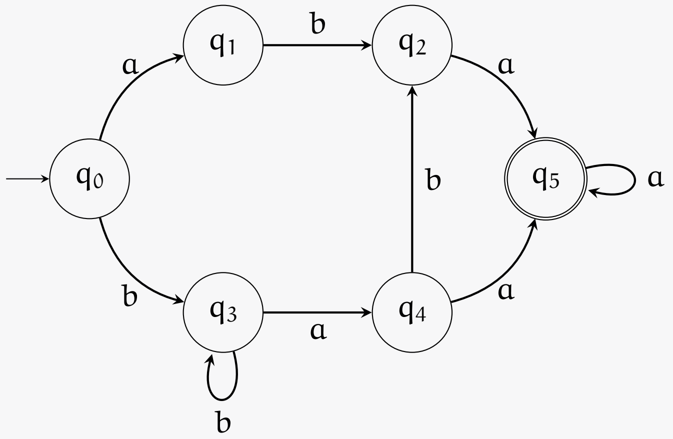
- $b^*(ab \cup ba)$

Observe que nuestras dos cadenas mínimas de aceptación son ab y ba así que forzamos esos dos caminos. Posteriormente para hacer que aparezcan las $b$es arbitrarias podemos agregar un bucle en el camino donde generamos ba. Si lo dejáramos hasta ahí nos faltarían las cadenas de la forma b^*ab pero esto se arregla fácilmente conectando el camino de ba al de ab colocando una b entre estados de aceptación:



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

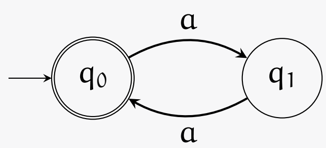
Note que el lenguaje es muy parecido a excepción de que las cadenas forzosamente acaban en por lo menos una a, pero lo que podemos hacer es usar el autómata construido previamente y aquellos estados de aceptación forzamos que salga una a hacia un nuevo estado de aceptación que generara la concatenación de a^+:



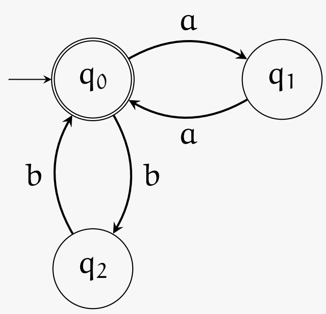
Punto 2: Sea $\Sigma = \{a, b\}$

- Diseñar un AFD que acepte el lenguaje de todas las cadenas que contienen un numero par de aes y un numero par de bes.

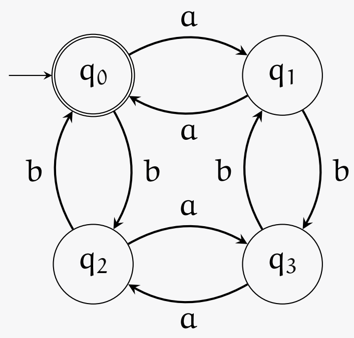
  Recordemos que si por ejemplo queremos construir un autómata que acepte el lenguaje de las cadenas que tienen una cantidad par de aes hacemos la siguiente construcción:



  Note como la construcción para cantidad par de bes es la misma entonces basta simplemente con agregar un camino para estas en el autómata ya diseñado hacia un nuevo estado:



  Observe que este autómata no acepta por ejemplo las cadenas $abab$ o $abba$. Aquí entra en juego la recomendación de usar 4 estados para la construcción y de esta forma podremos crear estos caminos para aceptar aquellas cadenas que antes no y que si son generadas por el lenguaje:

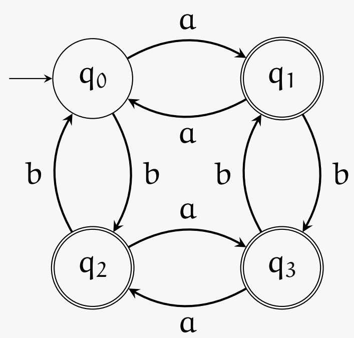


Algo importante a destacar aquí es que el único estado de aceptación es q_0 ya que en los demás siempre hay cantidad impar de aes o bes. Esto es relevante ya que es la esencia del siguiente ejercicio.

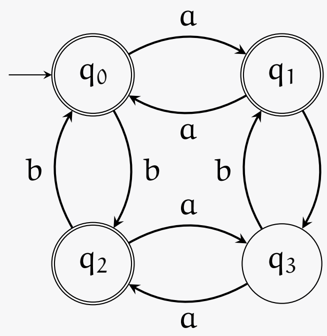
- Para cada combinación de condiciones 'par' e 'impar' y de las conectivas 'o' e 'y', diseñar un AFD que acepte el lenguaje L definido por L=lenguaje de las cadenas con un numero par/impar de aes y/o un num

  Observe que para este ejercicio tenemos ocho posibilidades en total, bueno para ser exactos simplemente tenemos que hacer 7 ya que en el anterior ya cubrimos el caso par de aes y par de bes. La forma en que determinaremos que estados se vuelven de aceptación es considerando cadenas básicas que cumplen las condiciones y en que estado terminan.

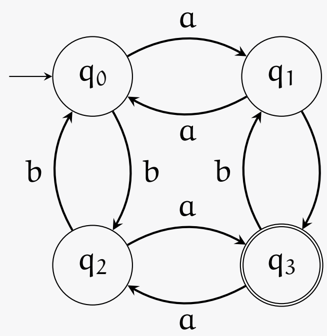
Caso 1: Impar de aes o impar de bes. Note que en el autómata original q_1, q_2 y q_3 no son de aceptación ya que en todos aes o bes son impares, luego en este caso estos tres se vuelven de aceptación mientras que q_0 ya no lo es.



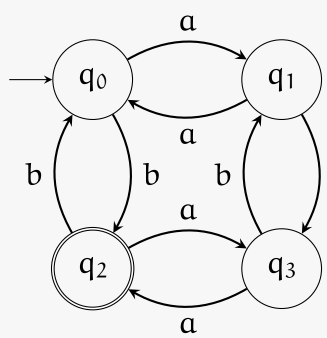
Caso 2: Par de aes o par de bes. Observe que el estado q_0 lo mantenemos como aceptación, luego note que la cadena abb es aceptada por la condición y si seguimos este camino en el autómata q_1 debe ser de aceptación. De la misma manera q_2 tiene que ser de aceptación ya que la cadena baa también es aceptada.



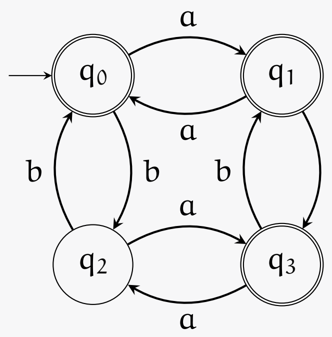
Caso 3: Impar de aes e impar de bes. Note que en la construcción anterior el estado q_3 no es de aceptación ya que es el único lugar donde aes y bes son impares, entonces para este caso ese va a ser nuestro único estado de aceptación.



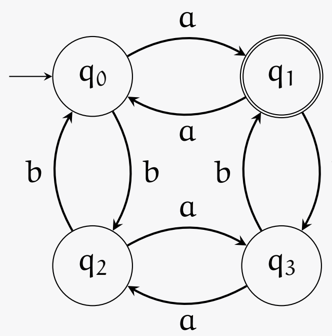
Caso 4: Par de aes e impar de bes. Observe que una cadena aceptada por estas condiciones es la cadena baa y esta es aceptada en q_2 luego este tiene que ser de aceptación. Ahora note que ninguno de los demás es de aceptación ya que q_0, q_1 aceptan cadenas con un numero par de bes y q_3 acepta cadenas con un numero impar de aes.



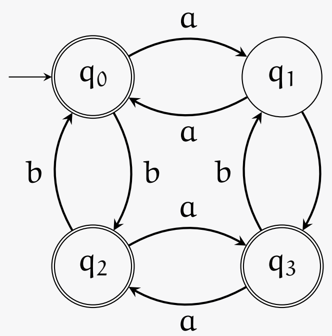
Caso 5: Impar de aes o par de bes. Por las bes pares q_0 es de aceptación. Por el impar de aes, q_1 y q_3 son de aceptación.



Caso 6: Impar de $a$es y par de $b$es. Una cadena que debe ser aceptada es por ejemplo abb, luego el estado que la acepta es q_1, ademas note que los demás estados no son de aceptación ya que aceptan las cadenas λ, b y ab que no cumplen las condiciones.



Caso 7: Par de $a$es o Impar de $b$es.

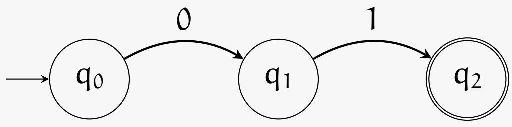


Observe que por ejemplo la condición del caso 7 es la negación de la condición del caso 6. Ademas los estados de aceptación del caso 6 son de no aceptación en el caso 7 y viceversa. Esto quedaría mas que nada como una mera curiosidad sino fuera por que posteriormente se mostrara que esta es una forma de construir autómatas para lenguajes que tienen una condición negativa. Primero se construirá el autómata que acepta la condición positiva y luego para aceptar la negación se cambian los estados de aceptación a no aceptación y viceversa.

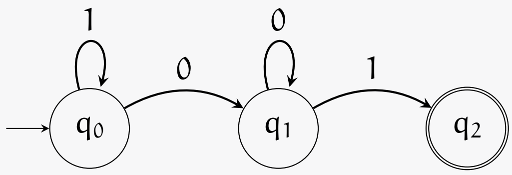
Punto 3: Sea $\Sigma = \{0, 1\}$. Diseñar AFD que acepten los siguientes lenguajes:

- El lenguaje de todas las cadenas que terminan en 01.

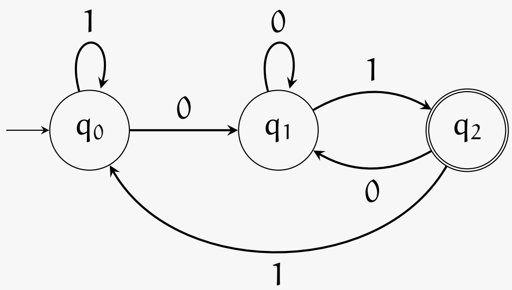
  Primero debemos forzar el camino para aceptar 01 que es la cadena mínima.



  Ahora observe la cadena puede empezar por unos arbitrarios, estos lo generamos por medio de un bucle en el estado inicial. Luego lo mismo pasa en q_1 pero con ceros. Por ultimo al estar ya en el estado de aceptación note que si sale un 0 o un 1 la cadena dejaría de ser de aceptada y uno estaría tentado a pensar que van a un estado limbo y que el autómata quedaría de la siguiente forma:

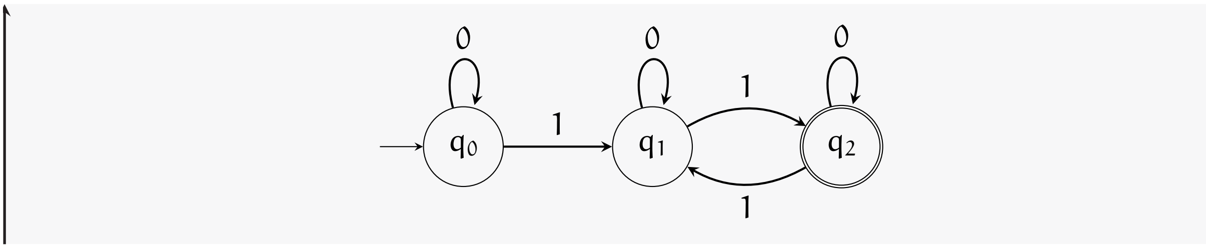


  Pero note que las cadenas 0101 y 101101 son cadenas validas pero no son aceptadas por el autómata. Esto se arregla fácilmente con añadir los bucles correspondientes desde el ultimo estado.



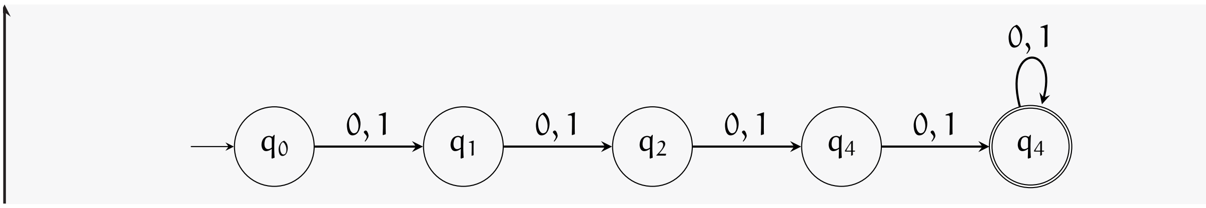
- El lenguaje de todas las cadenas que tienen un numero par ≥ 2 de unos.

  Primero forzamos la mínima cadena par de unos que es 11, luego realizamos la construcción para obtener cantidad par de unos y añadimos bucles de 0 en todos los estados ya que estos no tienen restricción alguna.



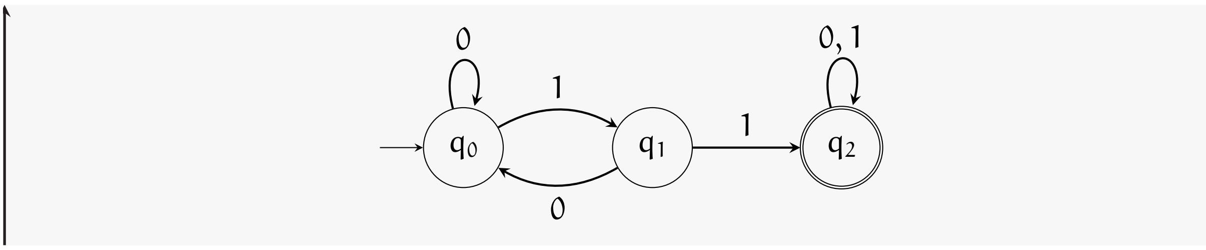
- El lenguaje de todas las cadenas con longitud ≥ 4.

  Al igual que hacíamos con las expresiones regulares, hacemos primero que se acepten todas las cadenas de longitud 4 que son las mínimas y posteriormente por medio de un bucle se generaran todas las demás.



- El lenguaje de todas las cadenas que contienen por lo menos dos unos consecutivos.

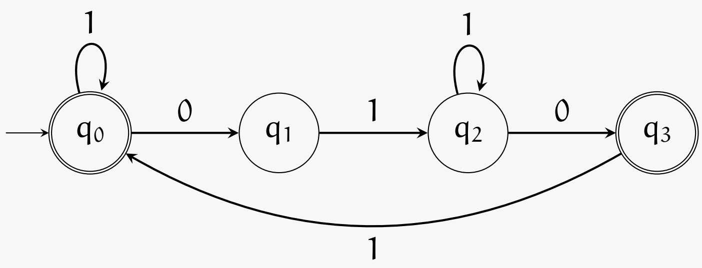
  Nuevamente forzaremos el hecho de que hayan al menos dos unos seguidos, en el primer estado agregaremos un bucle de 0 ya que estos pueden ser arbitrarios y en el ultimo un bucle $0, 1$ ya que ahi ya se cumple la condición entonces da igual que salga después



  Note que agregamos una flecha de q_1 a q_0 con etiqueta 0 ya que esto permite aceptar cadenas como 101011001 que sin este no serian aceptadas.

- El lenguaje de todas las cadenas que tienen un numero par de ceros pero no tienen dos ceros consecutivos.

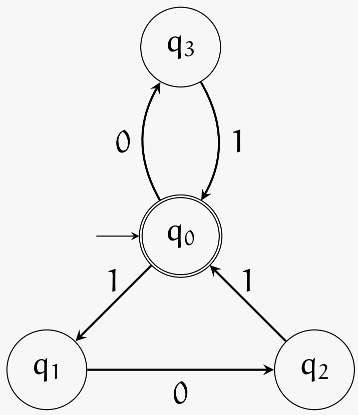
  Observe que debemos construir el camino forzando a que halla por lo menos un 1 entre dos ceros (no olvide que la cadena λ es de aceptación en este lenguaje).



Note que estamos acostumbrados a que cuando hacemos la construcción de pares el bucle se hace con los que queremos forzar a ser pares, pero en este caso se vuelve al estado inicial por medio de un 1 es debido a que la condición de que no pueden haber 2 ceros consecutivos.

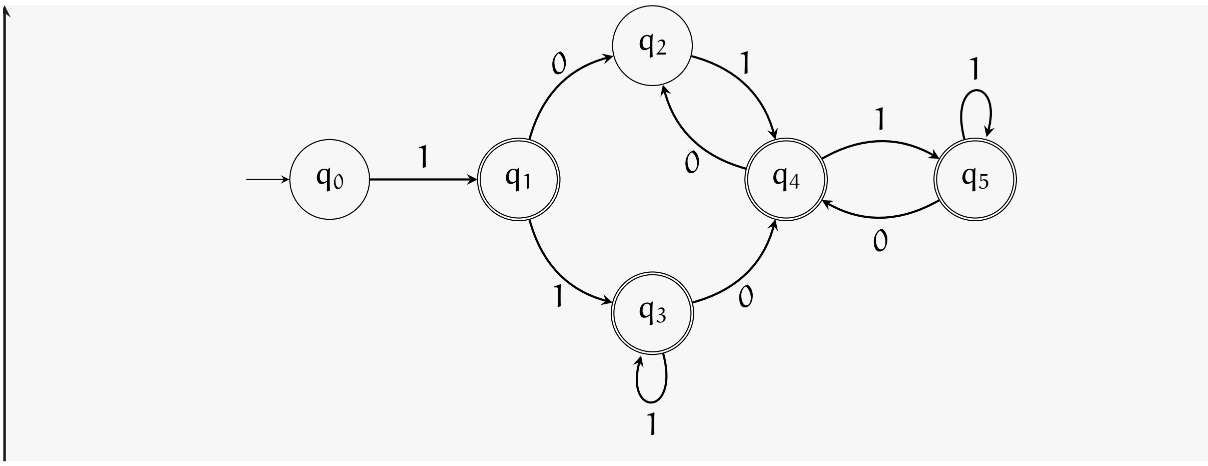
- $(01 \cup 101)^*$

Note que podemos formar los respectivos bucles por separado ya que la cadena empieza por 01 o por 101 y luego estas se concatenan en cualquier orden pero siempre son esos dos bloques.



- $1^+(10 \cup 01^+)^*$

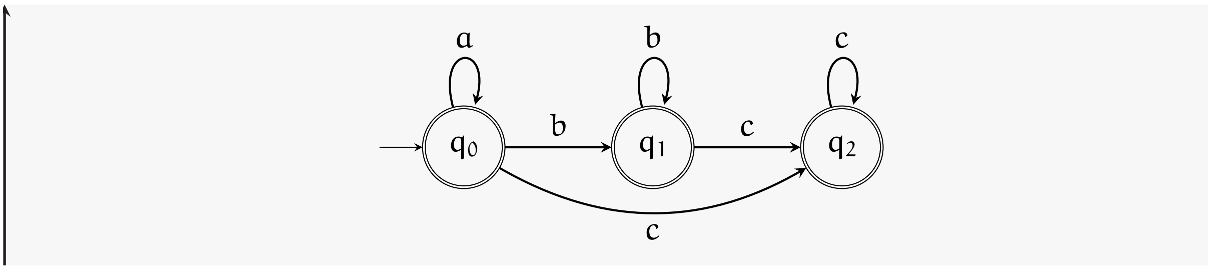
Este AFD fue construido por medio de un algoritmo que sera presentado en un capitulo posterior. Siendo sincero me fue imposible construir este AFD por mera inspección de la expresión regular.



Punto 4: Sea $\Sigma = \{a, b, c\}$. Diseñar AFD que acepten los siguientes lenguajes:

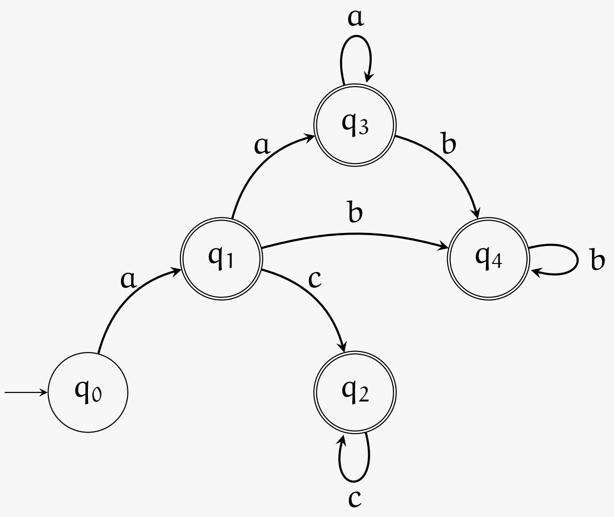
- $a^*b^*c^*$

  Basta con forzar los bucles para cada elemento y el hecho de que todas las cadenas tienen la misma secuencia de elementos, $a$es arbitrarias luego $b$es arbitrarias y por ultimo $c$es arbitrarias.

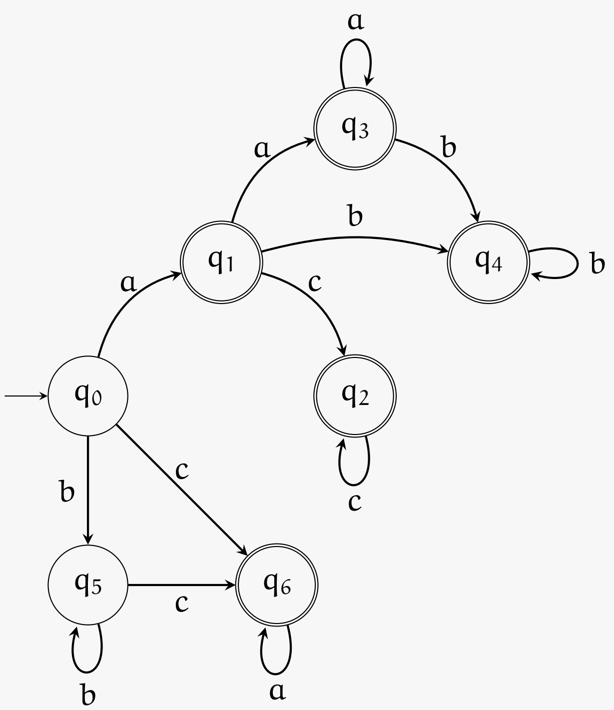


- $a^+b^* \cup ac^* \cup b^*ca^*$

  Haremos una construcción por dos caminos distintos a partir de las cadenas mínimas. La primera es el camino que se forma con la cadena a.



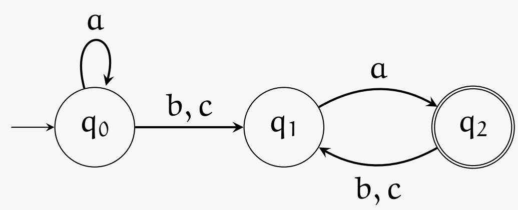
Esta construcción se puede ver un tanto enredada pero si nos fijamos detalladamente simplemente se están considerando tres casos, primero las cadenas de la forma $\mathtt{a}\mathtt{c}^*$, segundo las cadenas de la forma $\mathtt{a}\mathtt{b}^*$ y por ultimo las $\mathtt{a}\mathtt{a}^+\mathtt{b}^*$ (note que es necesario dividir estos dos últimos caminos así debido a la definición de los AFD). Ahora solo nos falta construir el camino para aceptar cadenas de la forma $\mathtt{b}^*\mathtt{c}\mathtt{a}^*$.



Note como forzamos la cadena mínima $\mathtt{c}$ y usamos un estado extra para generar el bucle de $\mathtt{b}$es ya que si no lo hacemos y colocamos ese bucle en q_0 alteraríamos toda la parte superior del autómata. De esta forma ya estamos considerando todas las cadenas que son dadas aceptadas en el lenguaje.

- $a^*(ba \cup ca)^+$

Observe que las cadenas mínimas son ba y ca luego basta con forzarlas y notar que ademas de las $a$es arbitrarias iniciales siempre luego de una b o c va una a y viceversa.

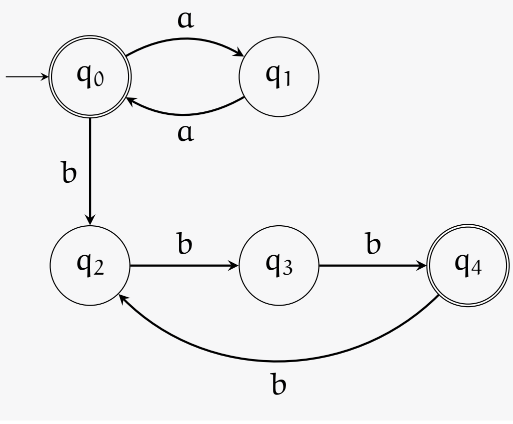


Punto 5: Sea $L = \{a^{2i}b^{3j} : i, j \geq 0\}$ definido sobre el alfabeto $\Sigma = \{a, b\}$. Encontrar una expresión regular para L y un AFD M tal que $L(M) = L$.

Primero es bastante evidente por la definición que se nos da que la expresión regular que determina a L es:

$$(aa)^*(bbb)^* = \left(a^2\right)^* \left(b^3\right)^*$$

Ahora para la construcción del AFD M tal que $L(M) = L$, basta con primero realizar la construcción para obtener una cantidad par de $a$es y posteriormente agregamos un camino para aceptar cantidad de $b$es que sean múltiplos de 3 (esta construcción es muy parecida a la de pares solo que en vez de concatenar bloques bb estamos concatenando bloques bbb).



Uno podría pensar que para hacer que se aceptaran esos bloques de bbb sobra ese estado final y se puede ahorrar haciendo que vaya una flecha de q_3 a q_0 con la etiqueta b, pero note que eso causaría que las cadenas de la forma $(bbb)^*(aa)^*$ fueran aceptadas lo cual no es contemplado dentro del lenguaje.

♦

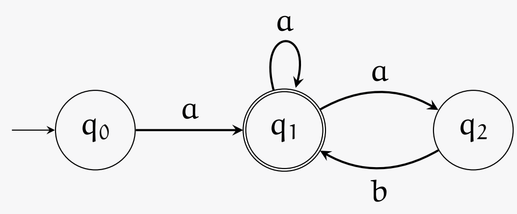
2.3 Autómatas finitos no-deterministas (AFN)

A continuación construiremos AFN los cuales no tienen las mismas restricciones que los AFD y por tanto son bastante mas sencillos de construir, eso si algo de vital importancia es que al igual que en los AFD debemos tener cuidado de no construir caminos que acepten cadenas que no son contempladas por el lenguaje.

Punto 1: Sea $\Sigma = \{a, b\}$. Diseñar AFN que acepten los siguientes lenguajes:

- $a(a \cup ab)^*$

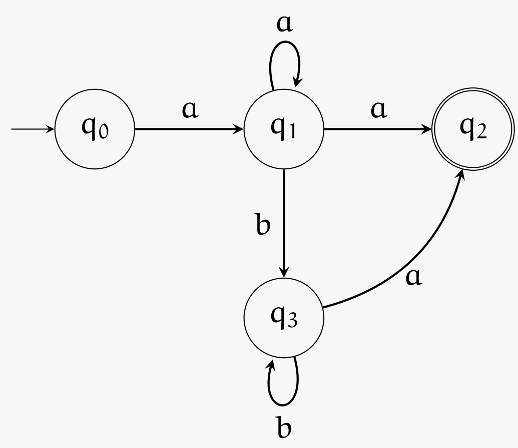
  Tenemos que asegurar que toda cadena empiece por una a, luego simplemente generamos los dos bucles necesarios uno para $a$es arbitrarias y el otro para bloques ab.

  Observe que este es un AFN ya que hay dos flechas con etiqueta a saliendo desde q_1.

- a^+b^*a

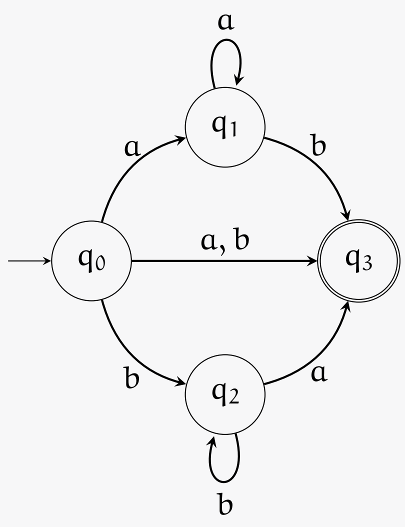
  Para la construcción forzamos la cadena mínima aa junto a el bucle de $a$es inicial.

  Note que las $b$es intermedias las agregamos por medio de un estado extra generando un camino nuevo.

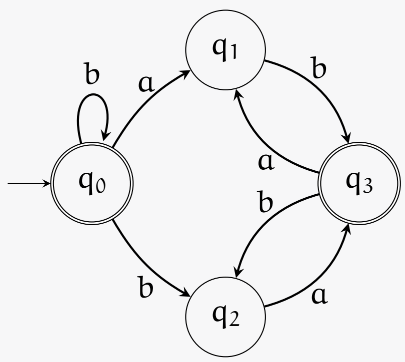
- $a^*b \cup b^*a$

Basta con construir un camino para las cadenas mínimas a y b, luego construimos dos caminos extra para aceptar a^*b y b^*a respectivamente.



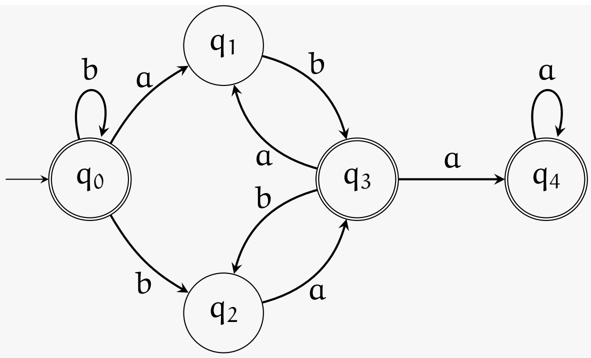
- $b^*(ab \cup ba)^*$

Forzaremos las $b$es arbitrarias en el primer estado y luego generamos los caminos correspondientes para la concatenación de ab y ba.



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

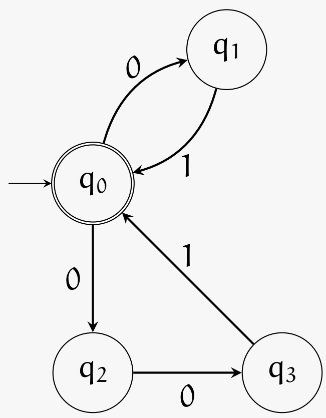
Observe que el lenguaje es exactamente igual al anterior a excepción de que ahora las cadenas pueden terminar en un numero arbitrario de $a$es, pero como estamos construyendo un AFN basta con notar que agregando un nuevo estado que sea de aceptación con un bucle de $a$es es suficiente.



Punto 2: Sea $\Sigma = \{0, 1\}$. Diseñar AFN que acepten los siguientes lenguajes:

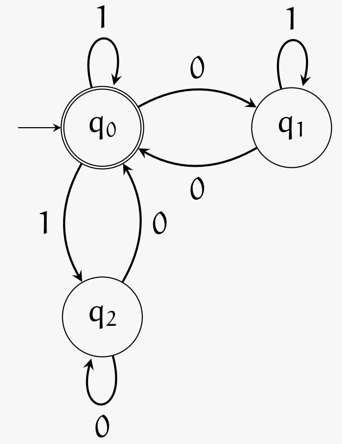
- $(01 \cup 001)^*$

  Simplemente construimos los caminos para aceptar 01 y 001 por separado.



- $(01^*0 \cup 10^*)^*$

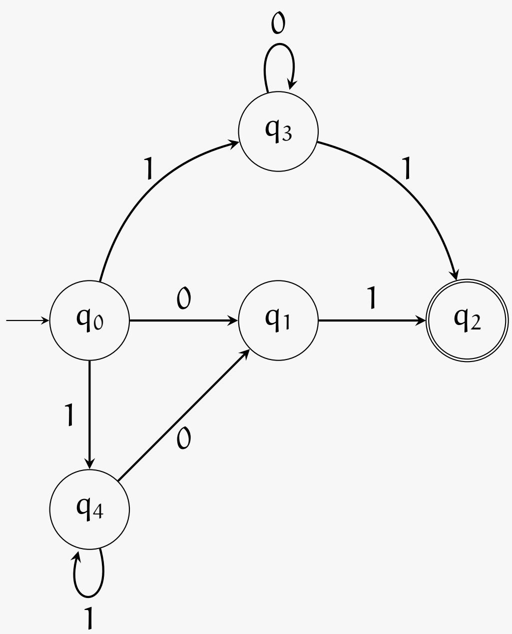
  Nuevamente basta con construir los caminos correspondientes a 01^*0 y 10^*.



Observe que es necesario incluir ese bucle con etiqueta 1 para aceptar cuando solo hay unos en la expresión 10^* (es decir en el caso donde 1 se concatena con λ).

- $1^*01 \cup 10^*1$

Construimos los caminos de las cadenas 10^*1 y 01.



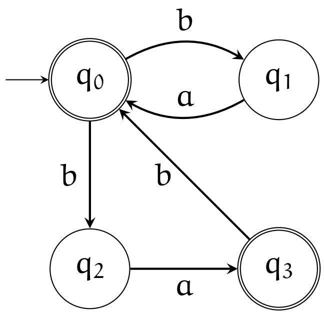
Note que usamos un estado adicional para construir el camino para aceptar 1^*01.

♦

2.4 Equivalencia computacional entre los AFD y los AFN

En la anterior sección se hizo bastante notorio como la construcción de AFN es mucho mas sencilla que la de un AFD. En esta sección se nos enseña que por medio de un proceso podemos convertir cualquier AFN en un AFD, algo que resulta extremadamente conveniente pero aveces el proceso es algo tedioso.

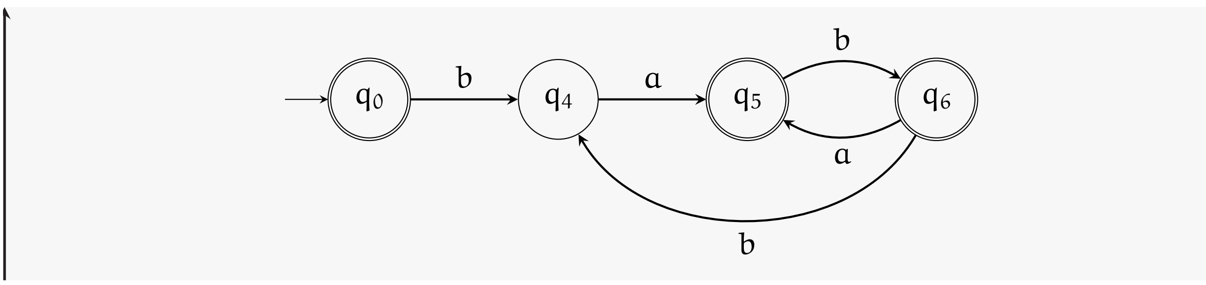
Punto 1: Utilizando el procedimiento de conversión presentado en esta sección, encontrar AFD equivalentes a los siguientes AFN:
A continuación en cada AFD que construyamos omitiremos los estados inútiles.

- 

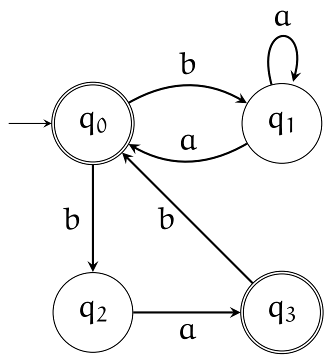
  Primero realicemos la extensión de la tabla.

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

  En el AFN original teníamos que los estados de aceptación eran q_0 y q_3 luego en el AFD los estados de aceptación son todos aquellos que contengan alguno de estos dos. Por estética definiremos $q_4 := \{q_1, q_2\}$, $q_5 := \{q_0, q_3\}$ y $q_6 := \{q_0, q_1, q_2\}$.

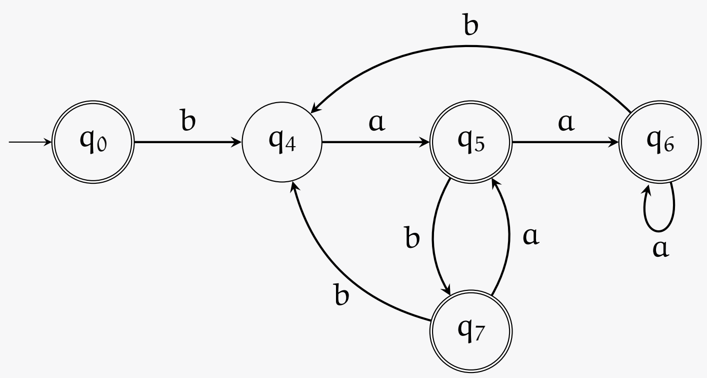
-



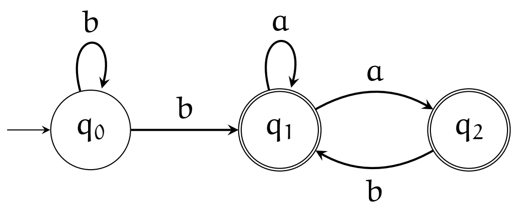
Realicemos la extensión de la tabla.

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

Nuevamente los estados de aceptación del AFD son todos aquellos que contienen q_0 o q_3. Ahora definimos $q_4 := \{q_1, q_2\}$, $q_5 := \{q_0, q_1, q_3\}$, $q_6 := \{q_0, q_1\}$ y $q_7 := \{q_0, q_1, q_2\}$.



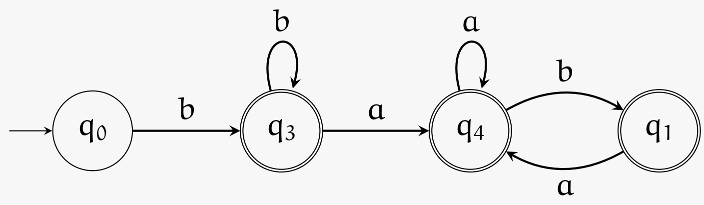
•



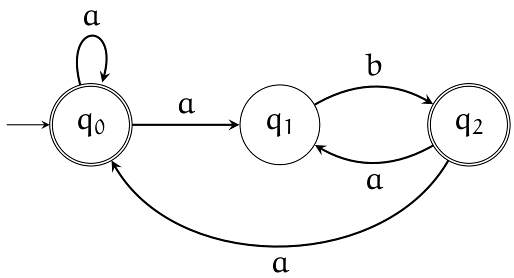
Realicemos la extensión de la tabla.

Δ	a	b
q_0	$\varnothing$	$\{q_0, q_1\}$
q_1	$\{q_1, q_2\}$	$\varnothing$
q_2	$\varnothing$	$\{q_1\}$
$\{q_0, q_1\}$	$\{q_1, q_2\}$	$\{q_0, q_1\}$
$\{q_1, q_2\}$	$\{q_1, q_2\}$	$\{q_1\}$

Los estados de aceptación serán aquellos que contengan q_1 y q_2, ademas definimos $q_3 := \{q_0, q_1\}$ y $q_4 := \{q_1, q_2\}$.



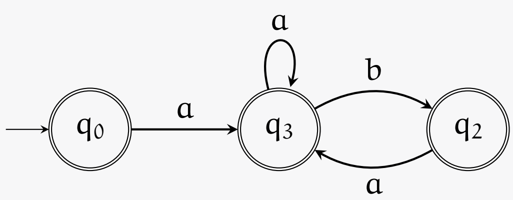
•



Creo que ya es evidente que es lo primero que haremos.

Δ	a	b
q_0	$\{q_0, q_1\}$	$\varnothing$
q_1	$\varnothing$	$\{q_2\}$
q_2	$\{q_0, q_1\}$	$\varnothing$
$\{q_0, q_1\}$	$\{q_0, q_1\}$	$\{q_2\}$

Los estados de aceptación son aquellos que contienen q_0 y q_2, ademas definimos $q_3 := \{q_0, q_1\}$.

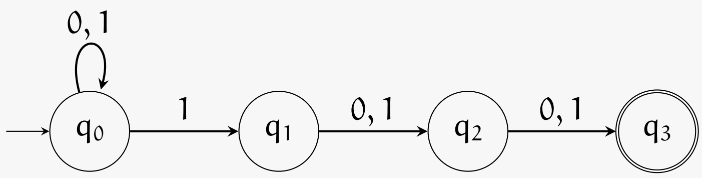


Punto 2: Sean $\Sigma = \{0, 1\}$ y L_3 el lenguaje de todas las cadenas de longitud ≥ 3 en las que el tercer símbolo, de derecha a izquierda es un 1. Diseñar un AFN con cuatro estados que acepte a L_3 y aplicar luego el procedimiento de conversión para encontrar un AFD equivalente.

Note primero como ayuda que la expresión regular que describe L_3 es:

$$(0 \cup 1)^*1(0 \cup 1)^2$$

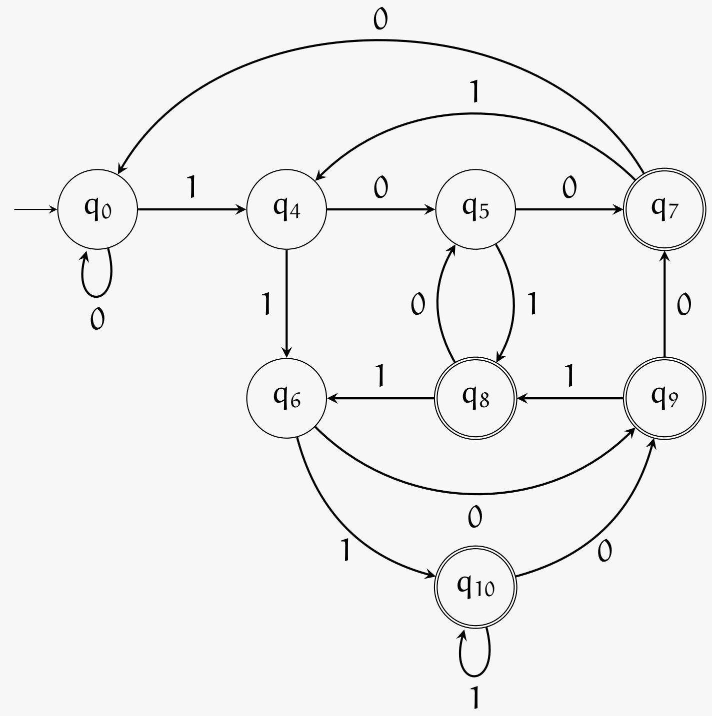
Luego el AFN que describe este lenguaje surge de una construcción bastante natural solo observando la expresión regular.



Ahora realicemos la extensión de la tabla.

Δ	0	1
q_0	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{q_3\}$
q_3	$\varnothing$	$\varnothing$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_1, q_2\}$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$
$\{q_0, q_3\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_1, q_3\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$\{q_0, q_2, q_3\}$	$\{q_0, q_3\}$	$\{q_0, q_1, q_3\}$
$\{q_0, q_1, q_2, q_3\}$	$\{q_0, q_2, q_3\}$	$\{q_0, q_1, q_2, q_3\}$

Ahora observe que en el AFD los estados de aceptación serán aquellos que contengan a q_3, ademas definimos $q_4 := \{q_0, q_1\}$, $q_5 := \{q_0, q_2\}$, $q_6 := \{q_0, q_1, q_2\}$, $q_7 := \{q_0, q_3\}$, $q_8 := \{q_0, q_1, q_3\}$, $q_9 := \{q_0, q_2, q_3\}$ y $q_{10} := \{q_0, q_1, q_2, q_3\}$.



Punto 3: Sea $M = \{\Sigma, Q, q_0, F, \delta\}$ un AFD. Demostrar por recursión sobre cadenas que la extensión δ satisface

$$\delta(q_0, uv) = \delta(\delta(q, u), v),$$

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

Demostración. Dados $q \in Q$ y $u, v \in \Sigma^*$ realizaremos recursión sobre la cadena v.
Si $v = \lambda$ es evidente por la definición de la extensión de δ que:

$$\delta(\delta(q, u), \lambda) = \delta(q, u) = \delta(q, u\lambda)$$

Como tenemos el caso base consideremos que nuestra hipótesis recursiva se cumple es decir:

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

Ahora veamos que ocurre con va donde $a \in \Sigma$

$$\begin{aligned}
\delta(q, uva) &= \delta(\delta(q, uv), a) && \text{(Definición de la extensión de } \delta) \\
&= \delta(\delta(\delta(q, u), v), a) && \text{(Hipótesis recursiva)} \\
&= \delta(\delta(q, u), va) && \text{(Definición de la extensión de } \delta)
\end{aligned}$$

De esta manera concluimos que la propiedad se cumple.

□

Punto 4: Sea $M = \{\Sigma, Q, q_0, F, \Delta\}$ un AFN. Demostrar por recursión sobre cadenas que la extensión Δ satisface

$$\Delta(q_0, uv) = \Delta(\Delta(q, u), v),$$

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

Demostración. Dados $q \in Q$ y $u, v \in \Sigma^*$ procederemos igual que en el punto anterior.
Si $v = \lambda$ por la definición de la extensión de Δ se tiene que:

$$\Delta(\Delta(q,u),\lambda) = \bigcup_{p \in \Delta(q,u)} \Delta(p,\lambda) = \bigcup_{p \in \Delta(q,u)} \{p\} = \Delta(q,u) = \Delta(q,u\lambda)$$

Como tenemos el caso base consideremos que nuestra hipótesis recursiva se cumple es decir:

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

Ahora veamos que ocurre con va donde $a \in \Sigma$

$$\begin{aligned} \Delta(q, uva) &= \Delta(\Delta(q, uv), a) && \text{(Definición de la extensión de } \Delta) \\ &= \Delta(\Delta(\Delta(q,u), v), a) && \text{(Hipótesis recursiva)} \\ &= \Delta(\Delta(q,u), va) && \text{(Definición de la extensión de } \Delta) \end{aligned}$$

De esta manera concluimos que la propiedad se cumple.

□

♦

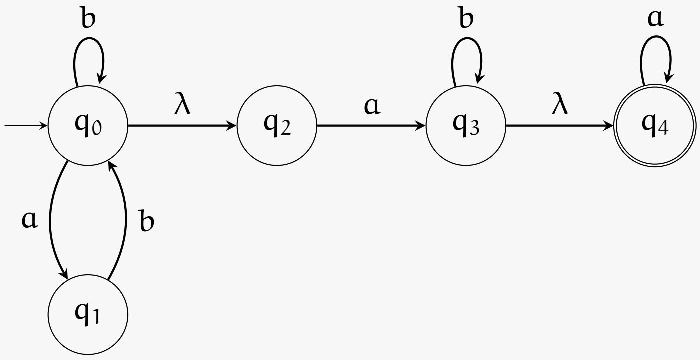
2.5 Autómatas con transiciones λ (AFN-λ)

Si bien previamente habíamos observado como los AFN eran bastante fáciles de construir, en esta sección nos daremos cuenta de que existe incluso construcciones aun mas sencillas, que son los AFN-λ. En esta sección veremos como construirlos y luego observaremos su relevancia.

Punto 1: Sea $\Sigma = \{a, b\}$. Diseñar AFN-λ que acepten los siguientes lenguajes:

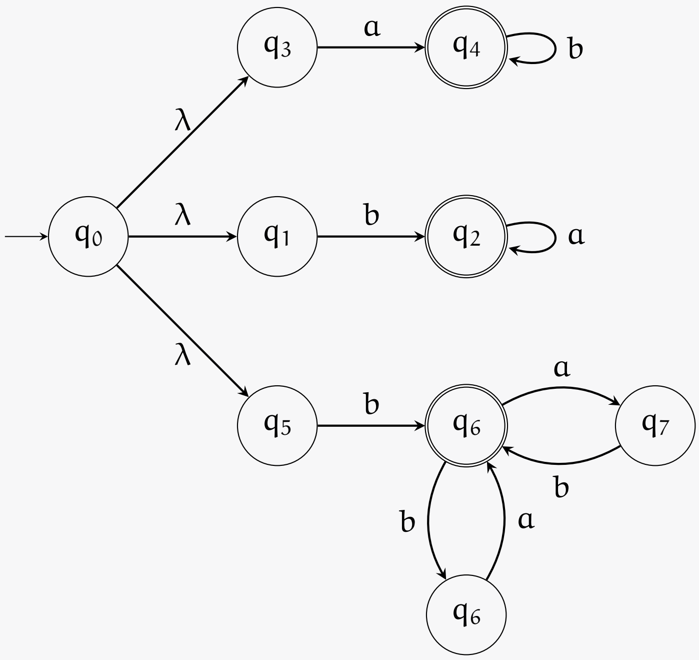
- $(ab \cup b)^* ab^* a^*$.

  Para este autómata realizaremos una construcción en serie, concatenando los autómatas respectivos de cada parte del lenguaje, note que a es la mínima cadena de aceptación.



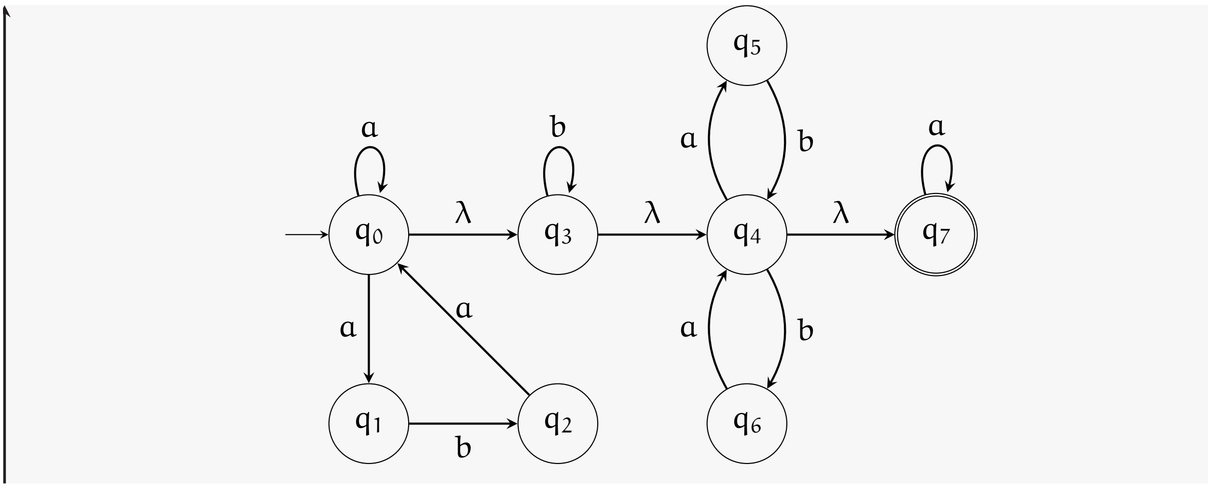
- $ab^* \cup ba^* \cup b(ab \cup ba)^*$.

  Basta con realizar una construcción en paralelo para cada expresión de la unión. Es decir construimos un AFD para ab^*, ba^* y $b(ab \cup ba)^*$ respectivamente y luego realizamos la construcción en paralelo.



- $(a \cup aba)^* b^* (ab \cup ba)^* a^*$.

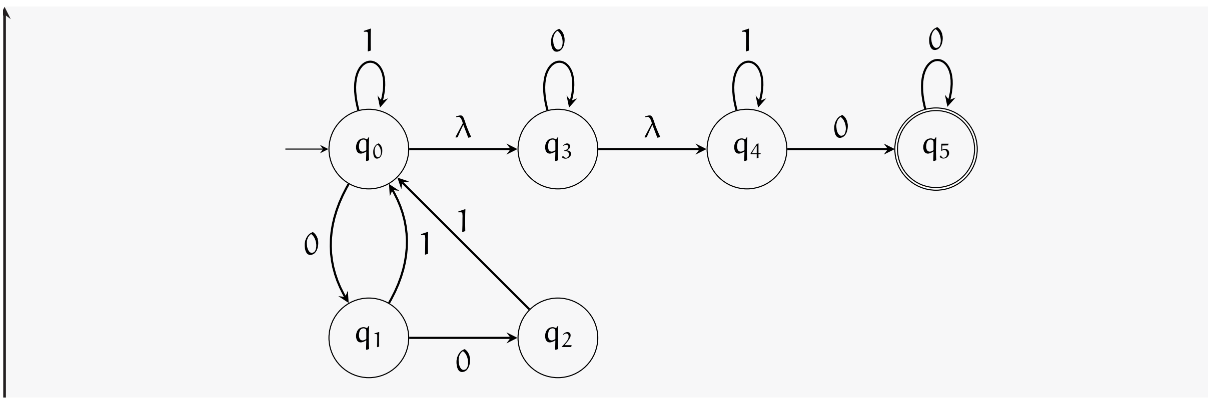
  Nuevamente realizaremos una construcción en serie, note que la cadena λ es la mínima que se acepta.



Punto 2: Sea $\Sigma = \{0, 1\}$. Diseñar AFN-λ que acepten los siguientes lenguajes:

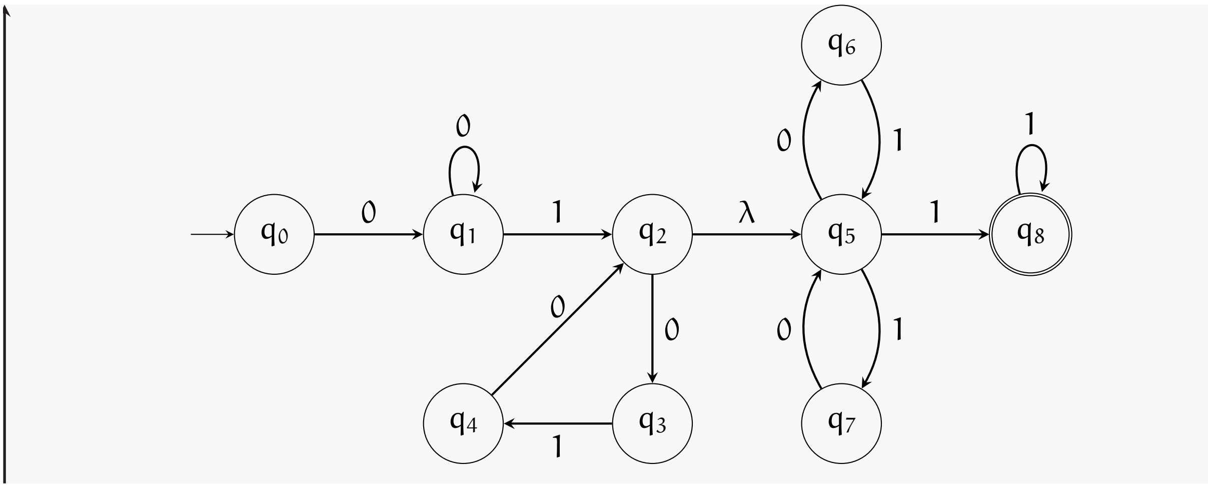
- $(1 \cup 01 \cup 001)^*0^*1^*0^+$.

  El único autómata que puede resultar un poco complicado de construir es el de la expresión inicial $(1 \cup 01 \cup 001)^*$ pero basta con 3 estados, luego simplemente es realizar una construcción en serie.



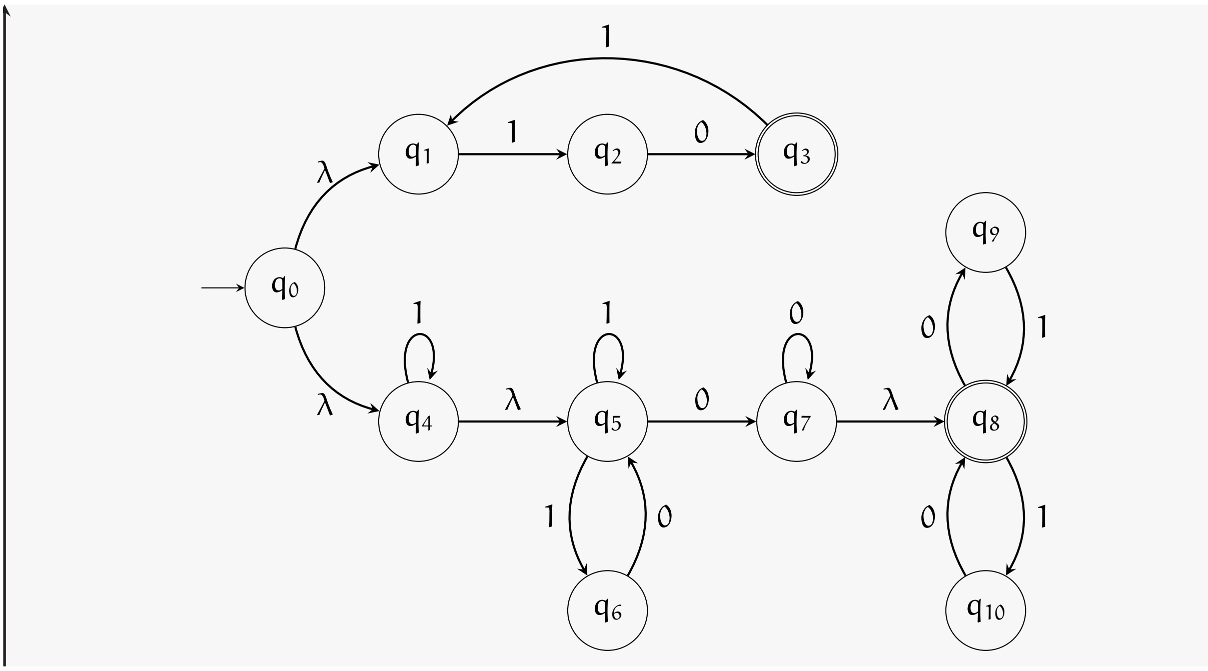
- $0^+1(010)^*(01 \cup 10)^*1^+$.

  Nuevamente una construcción en serie basta.



- $(101)^* \cup 1^*(1 \cup 10)^*0^+(01 \cup 10)^*$.

  Observe que en este caso usaremos tanto la construcción en paralelo como la construcción en serie ya que uno de los caminos es relativamente complicado de construir pero con esta construcción es mucho mas sencillo.

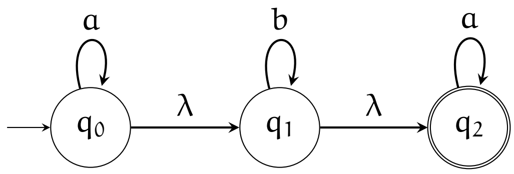


♦

2.6 Equivalencia computacional entre los AFN-λ y los AFN

Previamente habíamos observado como si construíamos un AFN para aceptar un lenguaje regular, podíamos por medio de un algoritmo convertirlo en un AFD. En esta sección observaremos como podemos convertir un AFN-λ en un AFN.

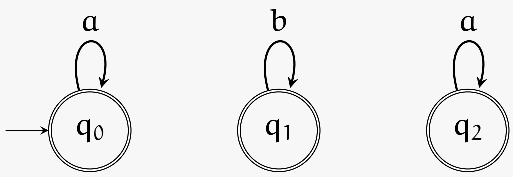
Punto 1:



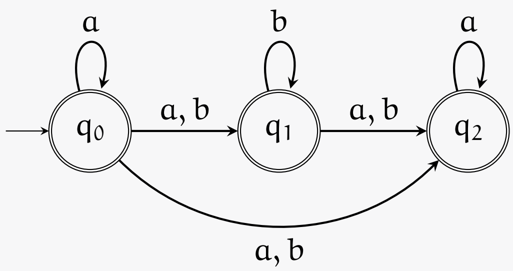
Primero hallemos la λ-clausura de cada estado:

$$\lambda[q_0] = \{q_0, q_1, q_2\}$$
$$\lambda[q_1] = \{q_1, q_2\}$$
$$\lambda[q_2] = \{q_2\}$$

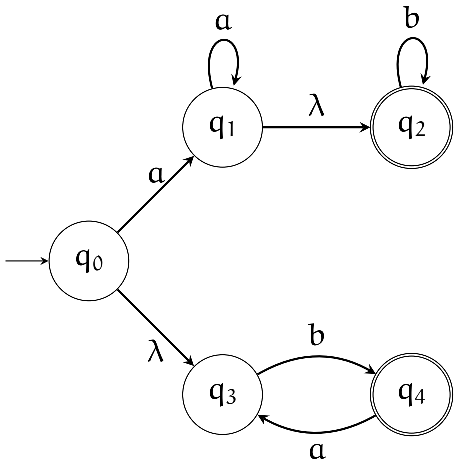
Observe que en el autómata original el único estado de aceptación es q_2, luego como esta pertenece a $\lambda[q_0], \lambda[q_1]$ y $\lambda[q_2]$, q_0, q_1 y q_2 son de aceptación en el AFN.



Antes de hacer la inspección para hallar el AFN, es importante tener en cuenta que las flechas ya existentes con etiquetas distintas a λ se mantienen. Como ejemplo de inspección observe que desde q_0, por medio de las etiquetas a y b se puede llegar tanto a q_1 como a q_2 haciendo las concatenaciones $a\lambda$, $a\lambda\lambda$, λb, $\lambda b\lambda$ respectivamente. El AFN obtenido es el siguiente:



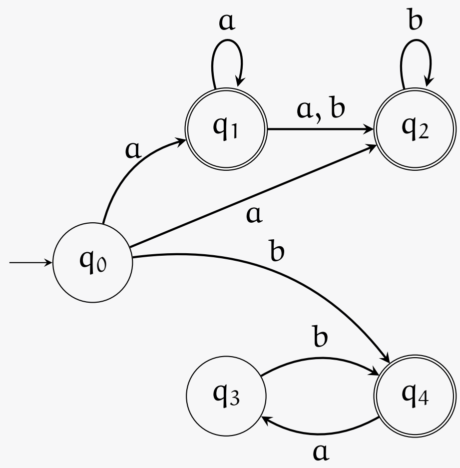
Punto 2:



Primero hallemos la λ-clausura de cada estado:

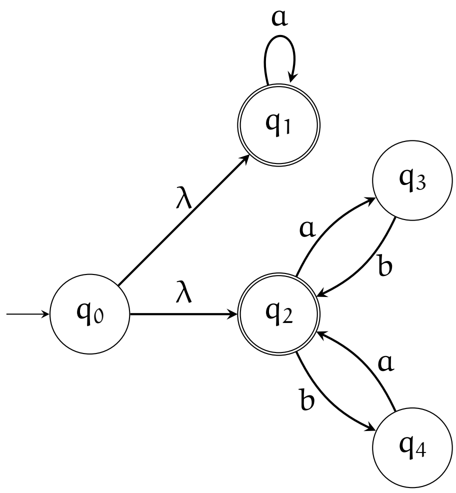
$$\lambda[q_0] = \{q_0, q_3\}$$
$$\lambda[q_1] = \{q_1, q_2\}$$
$$\lambda[q_2] = \{q_2\}$$
$$\lambda[q_3] = \{q_3\}$$
$$\lambda[q_4] = \{q_4\}$$

q_2 y q_4 son de aceptación en el original, por tanto observando las λ-clausuras, q_1, q_2 y q_4 son de aceptación en el nuevo autómata. Realizando el proceso de inspección el AFN obtenido es el siguiente:



Si bien la presentación del autómata podría ser mas pulida recomendamos dejar los estados en las posiciones originales para hacer mas evidente y sencillo el proceso de inspección.

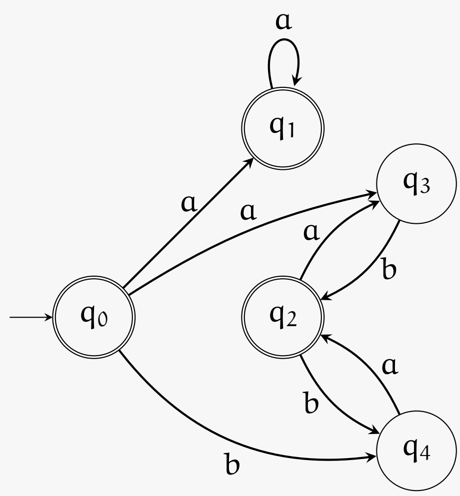
Punto 3:



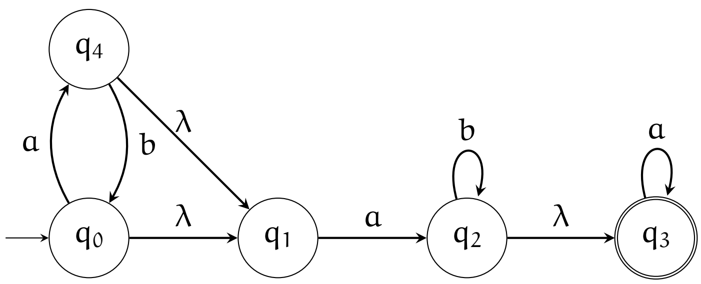
Primero hallemos la λ-clausura de cada estado:

$$\lambda[q_0] = \{q_0, q_1, q_2\}$$
$$\lambda[q_1] = \{q_1\}$$
$$\lambda[q_2] = \{q_2\}$$
$$\lambda[q_3] = \{q_3\}$$
$$\lambda[q_4] = \{q_4\}$$

Los estados q_1 y q_2 son de aceptación en el original, en el nuevo autómata también son de aceptación y ademas lo sera el estado q_0. Realizando la inspección obtenemos el siguiente AFN:



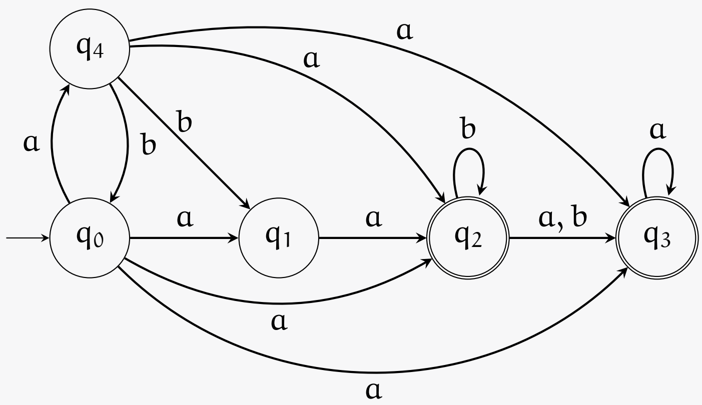
Punto 4:



Primero hallemos la λ-clausura de cada estado:

$$\begin{aligned}\lambda[q_0] &= \{q_0, q_1\}\\ \lambda[q_1] &= \{q_1\}\\ \lambda[q_2] &= \{q_2, q_3\}\\ \lambda[q_3] &= \{q_3\}\\ \lambda[q_4] &= \{q_1, q_4\}\end{aligned}$$

El único estado de aceptación original es q_3 y la única λ-clausura a la que pertenece es $\lambda[q_2]$ entonces en el nuevo autómata los estados de aceptación son q_2 y q_3. Realizando el proceso de inspección obtenemos el siguiente AFN:



Observe que la importancia radica en el hecho de que de ahora en adelante si nos atascamos en la construcción de un AFD e incluso de un AFN podemos siempre construir un AFN-λ, luego convertirlo en un AFN y por ultimo convertirlo en un AFD. Si bien es un proceso largo nos garantiza un AFD que funcione.
Por ultimo en esta sección todas las conversiones fueron hechas por inspección, pero se pueden hacer por medio de la formula brindada en las notas. Sugerimos que revisen que las construcciones hechas son correctas por medio de la formula y como ejercicio interesante conviertan los AFN de esta sección en AFD.

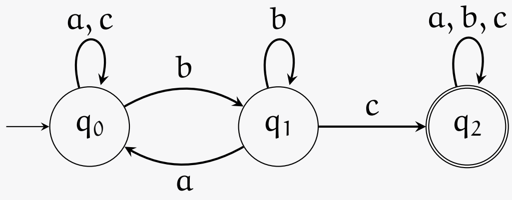
♦

2.7 Complemento de un autómata determinista

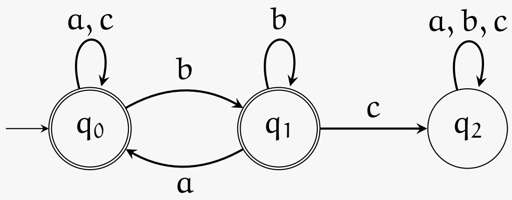
Previamente habíamos mencionado que a partir de un AFD con una condición podíamos construir un AFD para la negación de esa condición cambiando los estados de aceptación. En esta sección eso sera lo que haremos.

Punto 1: El lenguaje de todas las cadenas que no contienen la subcadena bc. Alfabeto: $\{a, b, c\}$.

Construyamos primero un AFD tal que acepte todas las cadenas que contienen la subcadena bc. Para esto forzamos bc como aceptación y consideramos los bucles necesarios.

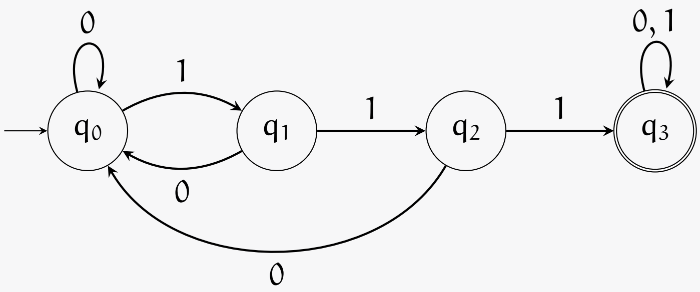


Ahora simplemente cambiamos los estados de aceptación, es decir q_0 y q_1 se vuelven de aceptación mientras que q_2 deja de serlo. De esta manera obtenemos el AFD que acepta el lenguaje deseado.

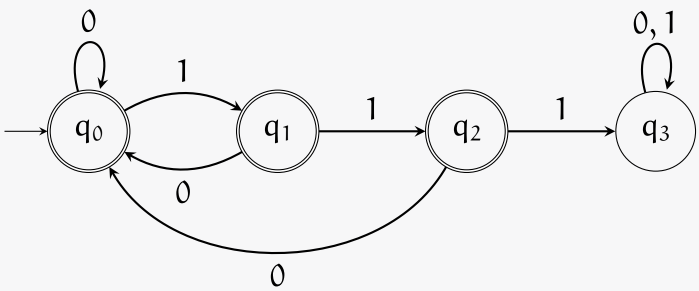


Punto 2: El lenguaje de todas las cadenas que no tienen tres unos consecutivos. Alfabeto: $\{0, 1\}$.

Primero construyamos el AFD que acepte todas las cadenas con tres unos consecutivos.

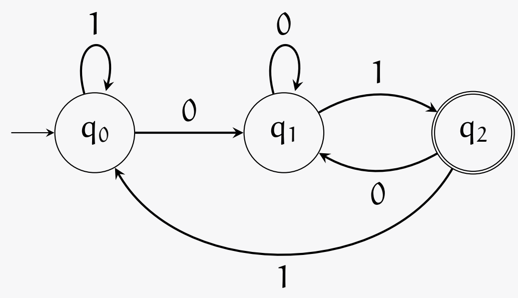


Luego el AFD que acepta el lenguaje deseado es:

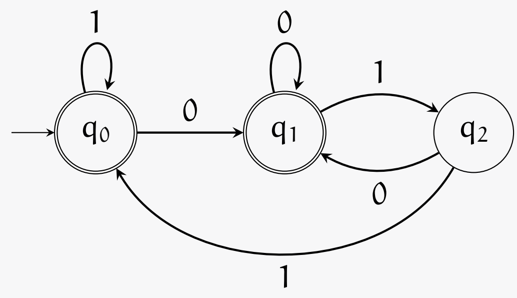


Punto 3: El lenguaje de todas las cadenas que no terminan en 01. Alfabeto: $\{0, 1\}$.

Primero construyamos el AFD que acepta todas las cadenas que terminan en 01.

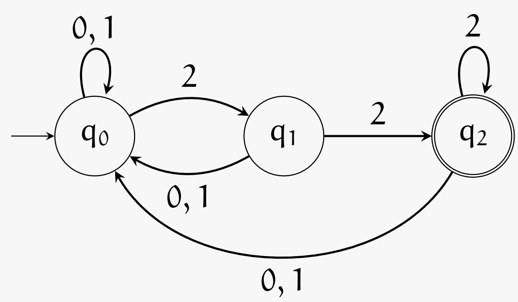


Luego el AFD que acepta el lenguaje deseado es:

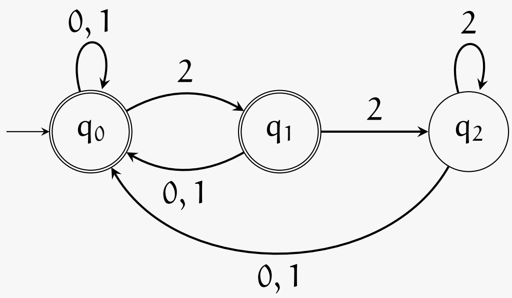


Punto 4: El lenguaje de todas las cadenas que no terminan en 22. Alfabeto: $\{0, 1, 2\}$.

Primero construyamos el AFD que acepta todas las cadenas que terminan en 22.



Luego el AFD que acepta el lenguaje deseado es:



Note que en los puntos 1 y 2, los estados q_2 y q_3 se volvieron limbo respectivamente y es posible quitarlos para una presentación simplificada del AFD, pero es preferible no retirarlos con el propósito de observar correctamente el proceso del complemento.

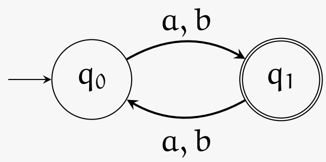
♦

2.8 Producto cartesiano de autómatas deterministas

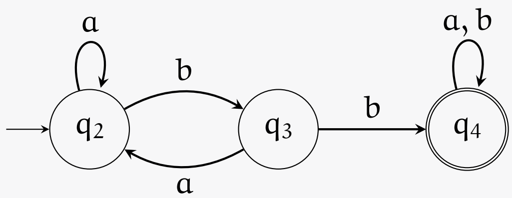
En esta sección observaremos otra manera de construir AFD para lenguajes que tengan dos condiciones los cuales puede que sean complicados por simple inspección, pero que con el algoritmo, se facilita muchísimo.

Punto 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 bes consecutivas, expresando L como diferencia de dos lenguajes.

Primero considere el AFD M_1 que acepta L_1 (las cadenas de longitud impar):

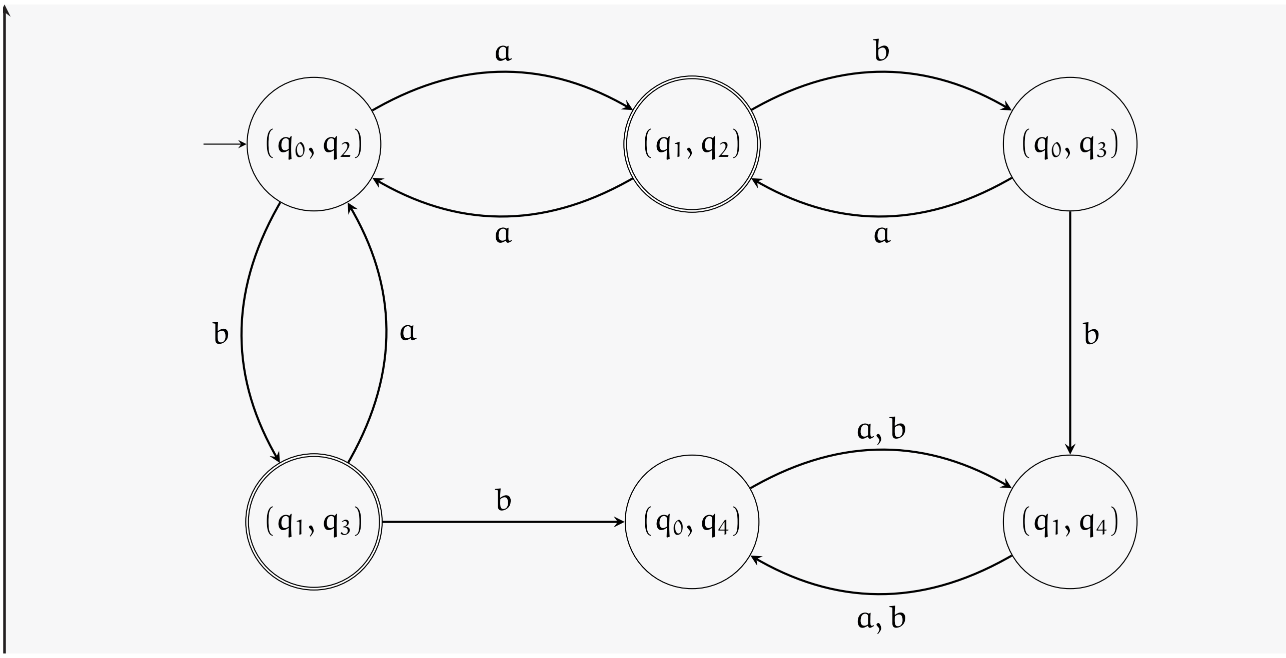


Ahora construyamos el AFD M_2 que acepta L_2 (las cadenas que contienen dos bes consecutivas):

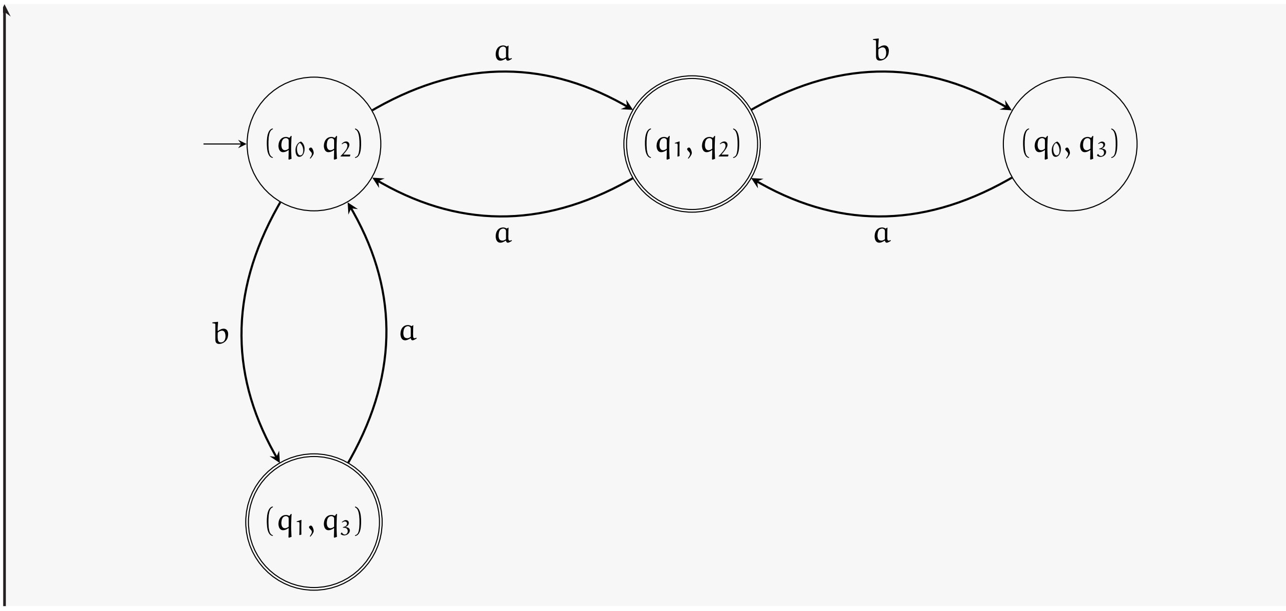


Entonces $L = L(M_1) - L(M_2) = L_1 - L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados:

$(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4); donde los estados de aceptación son (q_1, q_2) y (q_1, q_3) Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:



Note que los estados (q_0, q_4) y (q_1, q_4) son estados limbo, así el autómata simplificado es el siguiente:

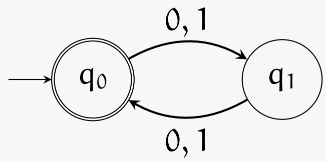


Observe que la solución presentada es igual a la del ejemplo correspondiente a este lenguaje, salvo la posición de los estados.

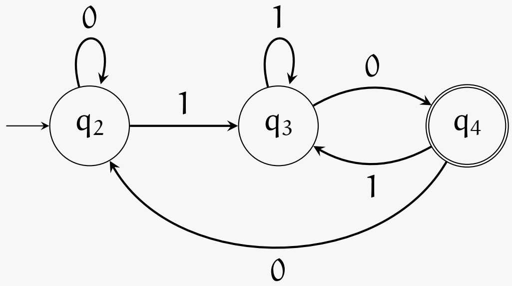
Punto 2: Utilizar el Teorema 2.11.1 para construir AFD que acepten los siguientes lenguajes sobre el alfabeto $\{0, 1\}$:

- El lenguaje L de todas las cadenas que tienen longitud par o que terminan en 10.

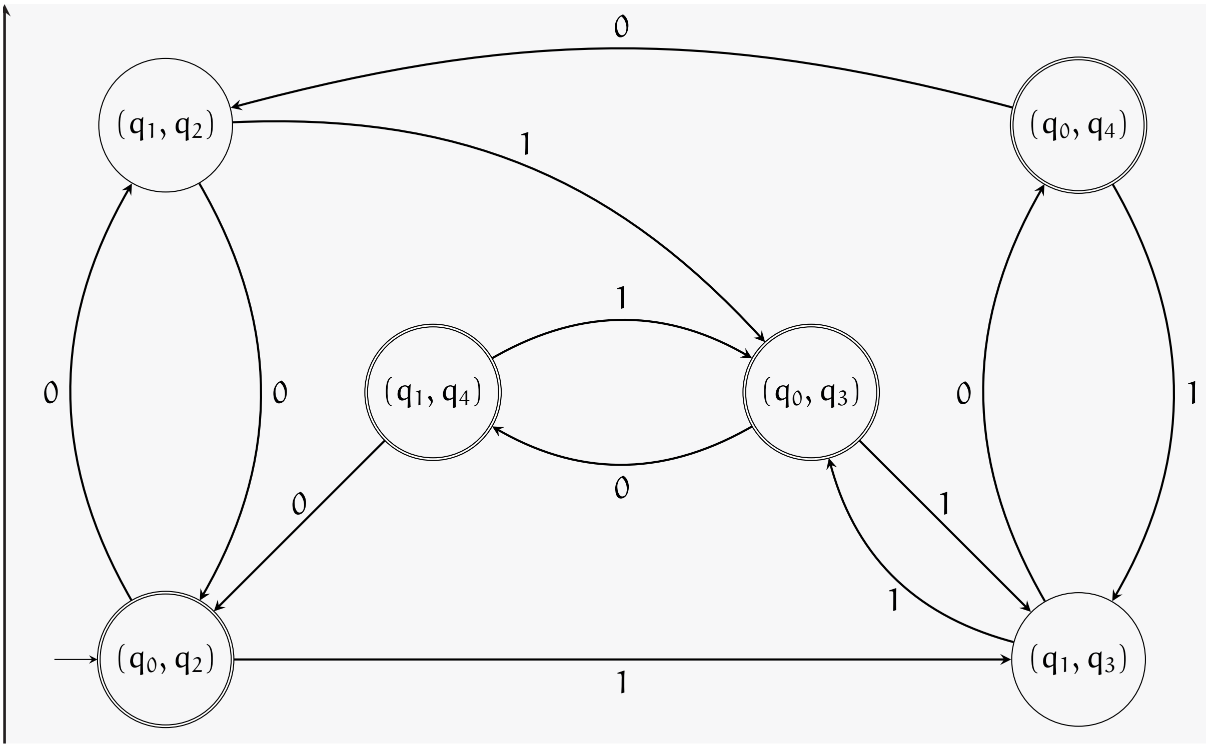
  Considere el AFD M_1 que acepta L_1 (las cadenas de que tienen longitud par):



  Ahora considere el AFD M_2 que acepta L_2 (las cadenas que terminan en 10):

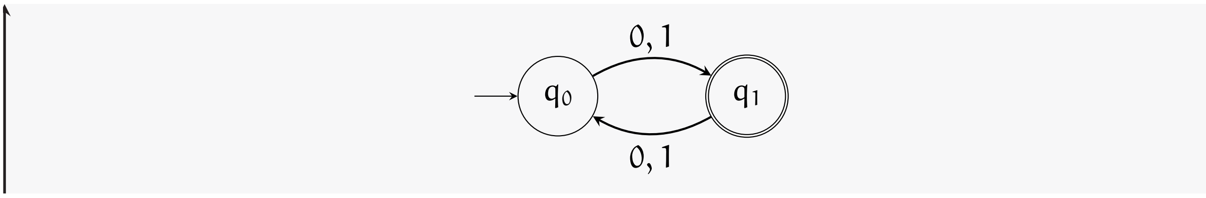


  Entonces $L = L(M_1) \cup L(M_2) = L_1 \cup L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados: $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4); donde los estados de aceptación son $(q_0, q_2), (q_0, q_3), (q_0, q_4)$ y (q_1, q_4). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

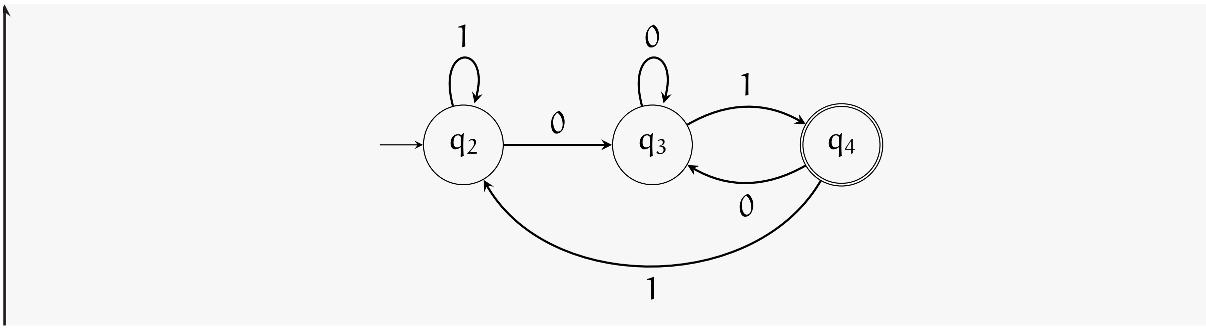


- El lenguaje L de todas las cadenas que tienen longitud impar y que terminan en 01.

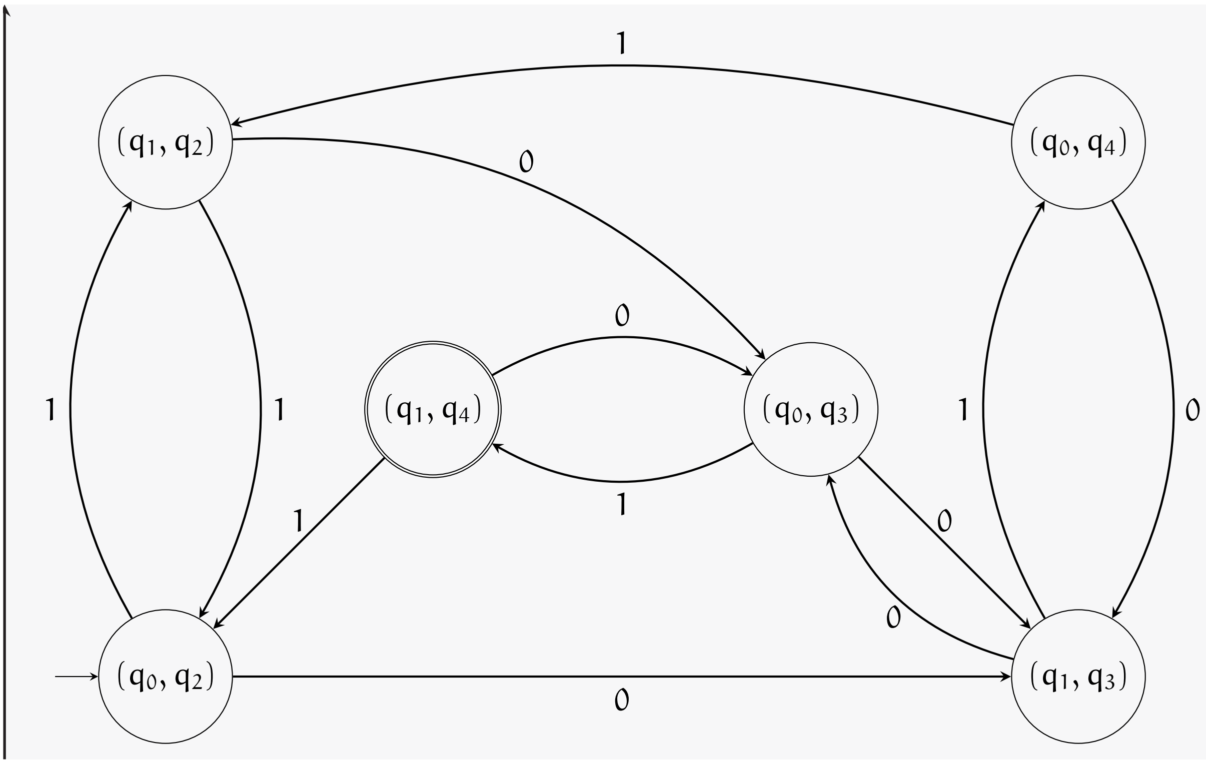
  Considere el AFD M_1 que acepta L_1 (las cadenas de que tienen longitud impar):



  Ahora considere el AFD M_2 que acepta L_2 (las cadenas que terminan en 01):

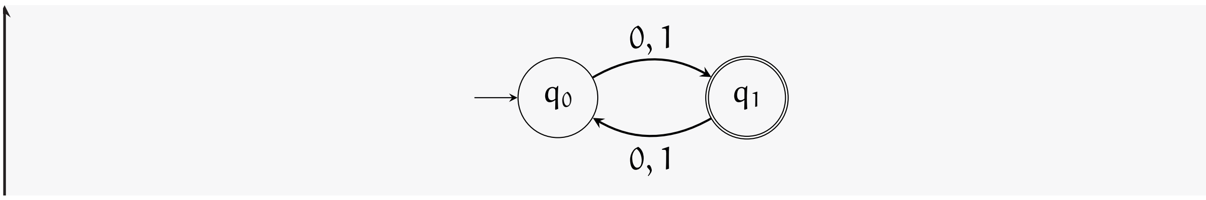


  Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados: $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4); donde el único estado de aceptación es (q_1, q_4). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

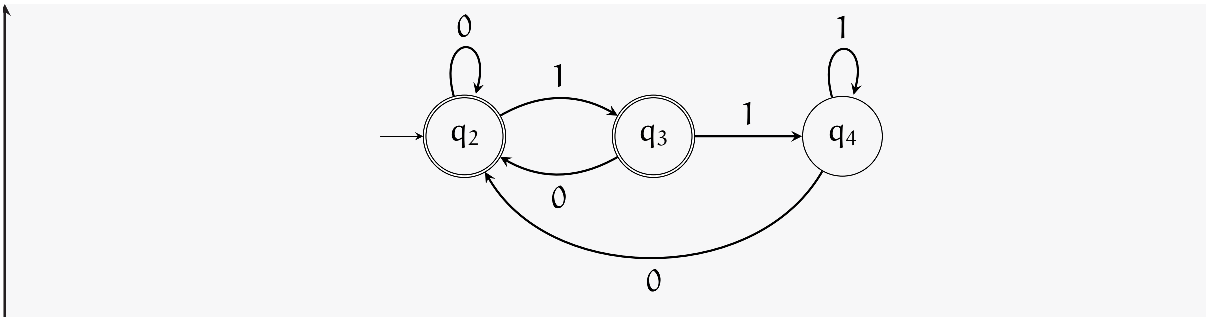


- El lenguaje L de todas las cadenas que tienen longitud impar y que no terminan en 11.

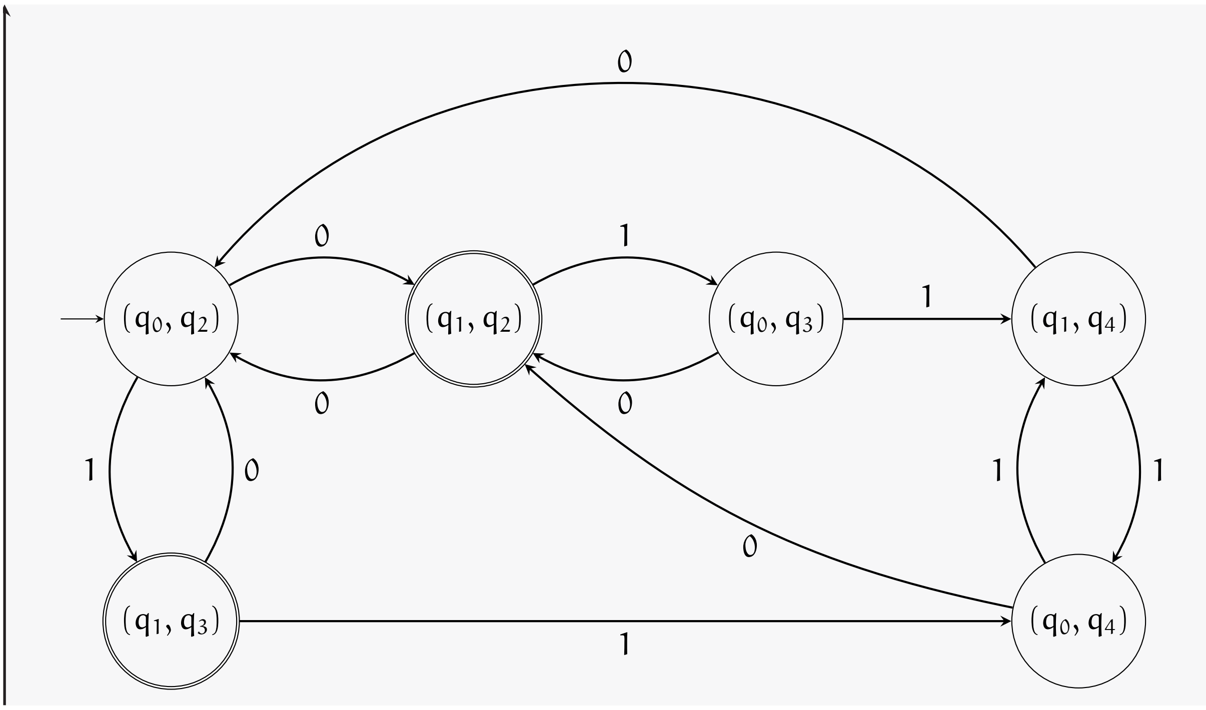
  Considere el AFD M_1 que acepta L_1 (las cadenas de que tienen longitud impar):



  Ahora considere el AFD M_2 que acepta L_2 (las cadenas que no terminan en 11):

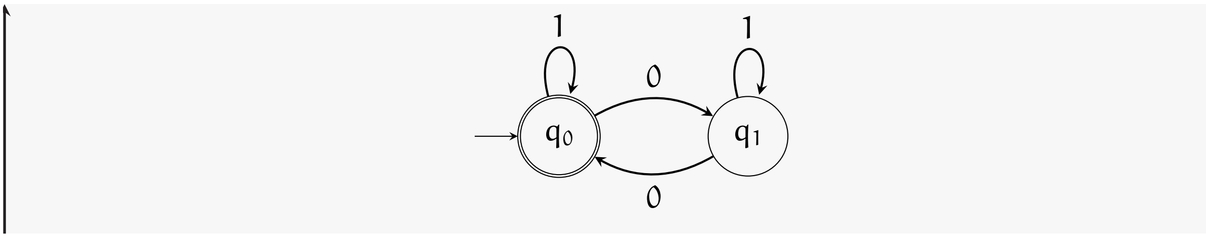


Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados: $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4); donde los estados de aceptación son (q_1, q_2) y (q_1, q_3). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

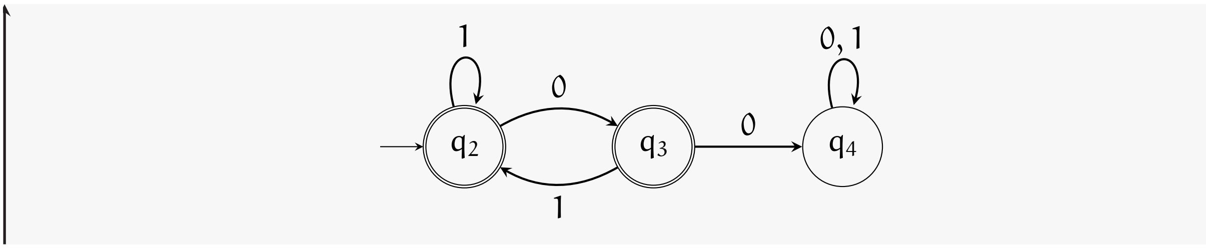


- El lenguaje L de todas las cadenas que tienen un numero par de ceros o que no tienen dos ceros consecutivos.

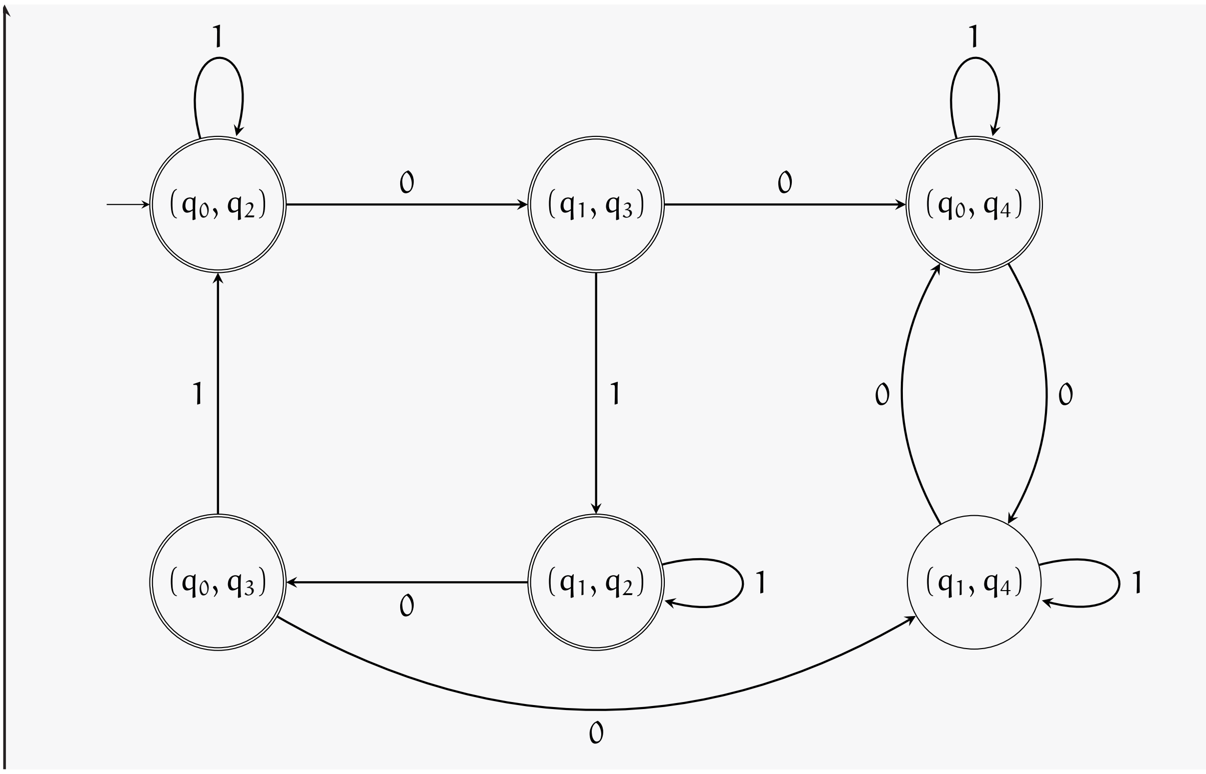
  Considere el AFD M_1 que acepta L_1 (las cadenas de que tienen un numero par de ceros):



Ahora considere el AFD M_2 que acepta L_2 (las cadenas que no tienen dos ceros consecutivos):



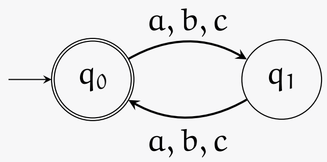
Entonces $L = L(M_1) \cup L(M_2) = L_1 \cup L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados: $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4); donde los estados de aceptación son $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2)$ y (q_1, q_3). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:



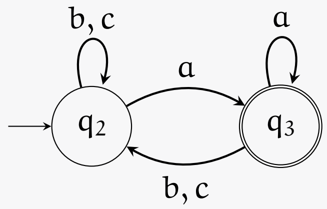
Punto 3: Utilizar el Teorema 2.11.1 para construir AFD que acepten los siguientes lenguajes sobre el alfabeto $\{a, b, c\}$

- El lenguaje L de todas las cadenas que tienen longitud par y terminan en a.

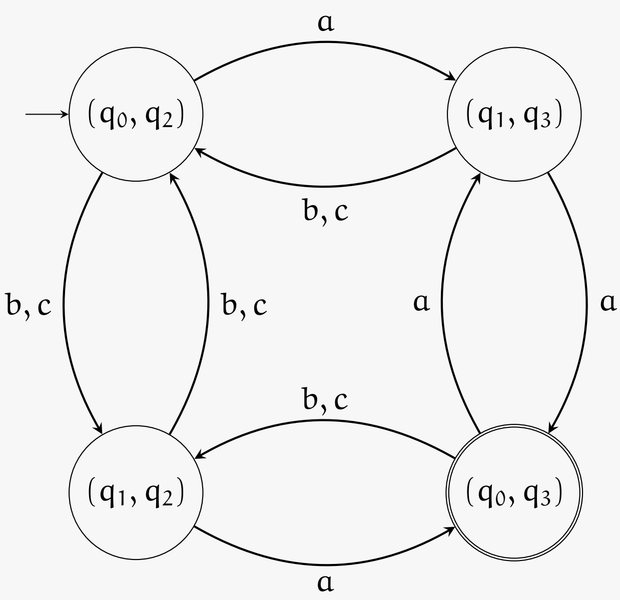
  Considere el AFD M_1 que acepta L_1 (las cadenas de longitud par):



  Ahora considere el AFD M_2 que acepta L_2 (las cadenas que terminan en a):

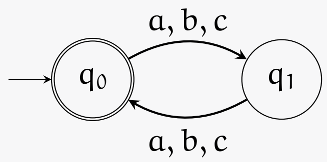


  Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 4 estados: $(q_0, q_2), (q_0, q_3), (q_1, q_2)$ y (q_1, q_3); donde el único estado de aceptación es (q_0, q_3). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

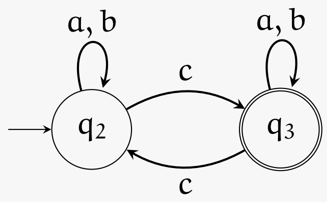


- El lenguaje L de todas las cadenas que tienen longitud par o que tienen un numero impar de c’s.

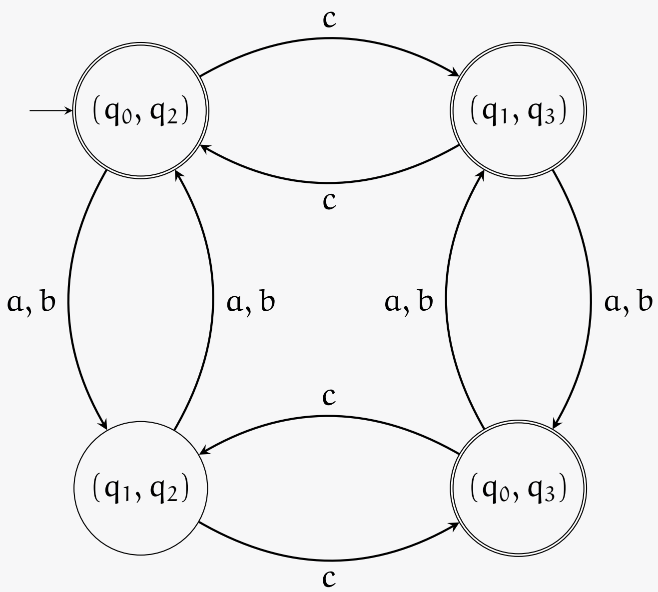
Considere el AFD M_1 que acepta L_1 (las cadenas de longitud par):



Ahora considere el AFD M_2 que acepta L_2 (las cadenas que tienen un numero impar de ces):

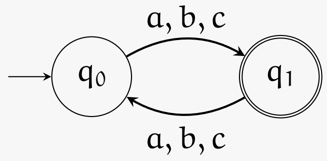


Entonces $L = L(M_1) \cup L(M_2) = L_1 \cup L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 4 estados: $(q_0, q_2), (q_0, q_3), (q_1, q_2)$ y (q_1, q_3); donde los estados de aceptación son $(q_0, q_2), (q_0, q_3)$ y (q_1, q_3). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

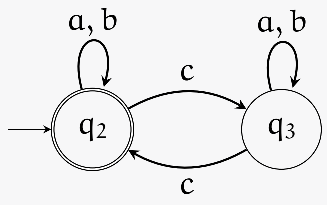


- El lenguaje L de todas las cadenas que tienen longitud impar y que tienen un numero par de ces.

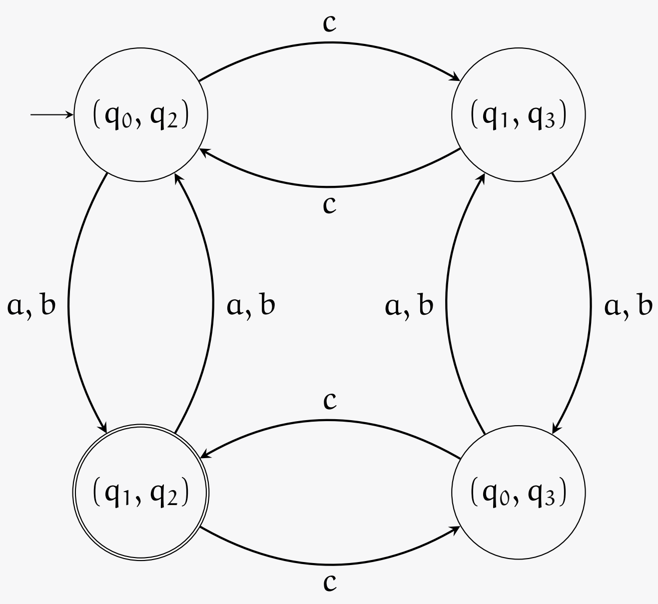
Considere el AFD M_1 que acepta L_1 (las cadenas de longitud impar):



Ahora considere el AFD M_2 que acepta L_2 (las cadenas que tienen un numero par de ces):

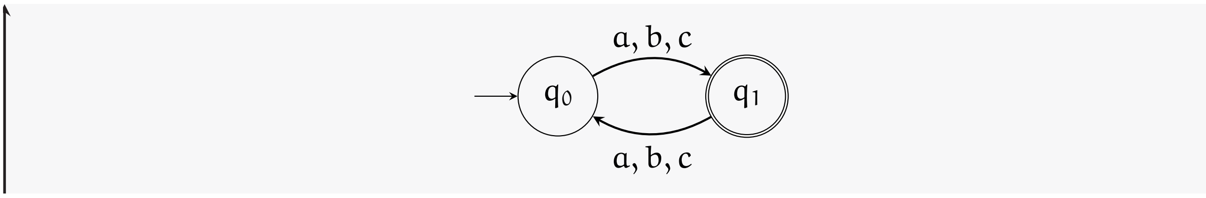


Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 4 estados: $(q_0, q_2), (q_0, q_3), (q_1, q_2)$ y (q_1, q_3); donde el único estado de aceptación es (q_1, q_2). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

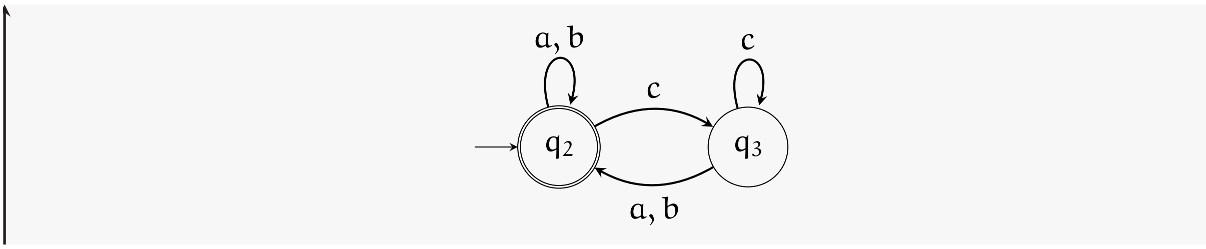


- El lenguaje L de todas las cadenas que tienen longitud impar y que no terminan en c.

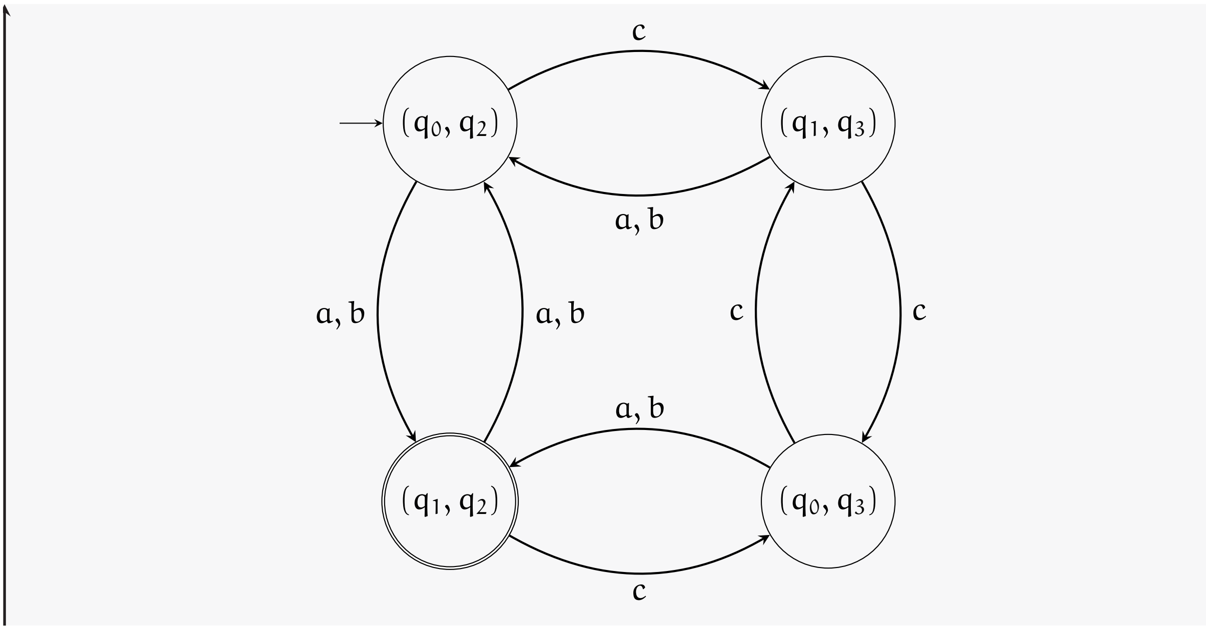
Considere el AFD M_1 que acepta L_1 (las cadenas de longitud impar):



Ahora considere el AFD M_2 que acepta L_2 (las cadenas que no terminan en c):

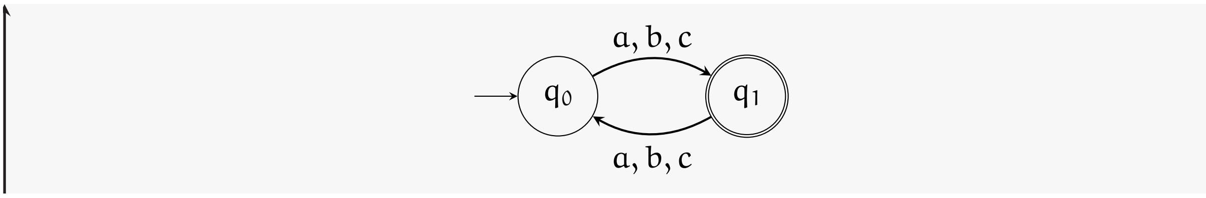


Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 4 estados: $(q_0, q_2), (q_0, q_3), (q_1, q_2)$ y (q_1, q_3); donde el único estado de aceptación es (q_1, q_2). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:

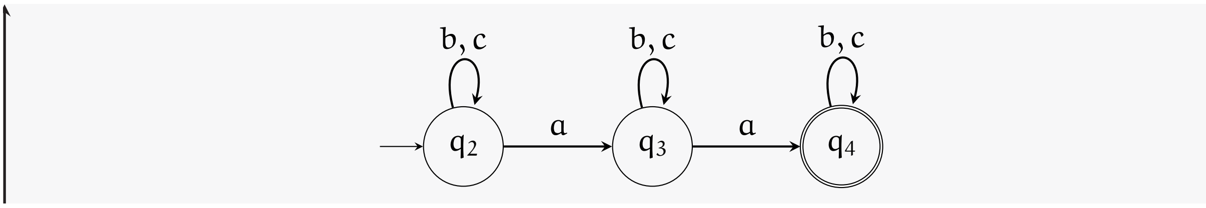


- El lenguaje L de todas las cadenas de longitud impar que tengan exactamente dos $a$es.

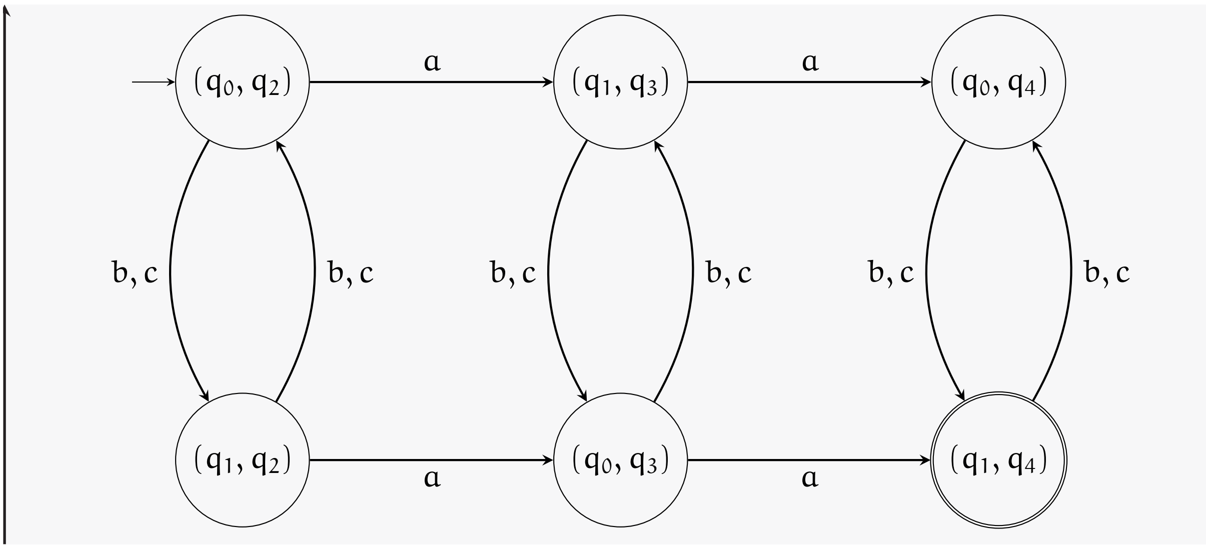
  Considere el AFD M_1 que acepta L_1 (las cadenas de longitud impar):



Ahora considere el AFD M_2 que acepta L_2 (las cadenas que tienen exactamente dos $a$es):



Entonces $L = L(M_1) \cap L(M_2) = L_1 \cap L_2$. El producto cartesiano $M_1 \times M_2$ tiene en total 6 estados: $(q_0, q_2), (q_0, q_3), (q_0, q_4), (q_1, q_2), (q_1, q_3)$ y (q_1, q_4) (Note que como usamos la construcción de intersección podemos ignorar el estado limbo no representado en M_2); donde el único estado de aceptación es (q_1, q_4). Luego siguiendo la función de transición definida se tiene que el AFD M tal que $L(M) = L$ es:



Punto 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 $L_1 \triangleleft L_2$. Recuérdese que la diferencia simétrica se define como

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

Por el teorema presentado en la sección sabemos que el conjunto que acepta $L_1 - L_2$ es:

$$F_1 \times (Q_2 - F_2)$$

Mientras que el que acepta $L_2 - L_1$ es:

$$F_2 \times (Q_1 - F_1)$$

Ahora como $L_1 \triangleleft L_2 = (L_1 - L_2) \cup (L_2 - L_1)$ Basta con unir las dos expresiones anteriores para obtener el conjunto adecuado de estados de aceptación; es decir:

$$F = (F_1 \times (Q_2 - F_2)) \cup (F_2 \times (Q_1 - F_1)).$$

♦

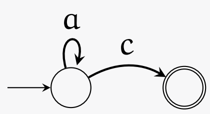
2.9 Teorema de Kleene, parte I

Siempre se puede construir un AFN-λ si conocemos la expresión regular. Es la lección de esta sección; siguiendo unos simples pasos y siendo bastante organizados podemos construir un AFN-λ dada una expresión regular y por tanto siempre podremos llegar a un AFD por los métodos de conversión ya presentados previamente.

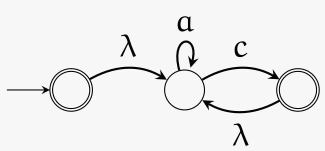
Utilizar el procedimiento del Teorema 2.12.1 para construir AFN-λ que acepten los siguientes lenguajes sobre el alfabeto $\{a, b, c\}$.

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

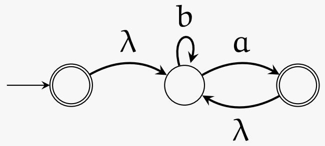
i) Autómata que acepta a^*c:



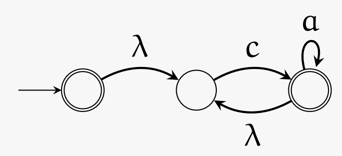
ii) Autómata que acepta $(a^*c)^*$



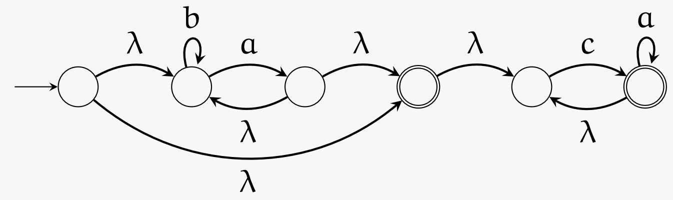
iii) Autómata que acepta $(b^*a)^*$



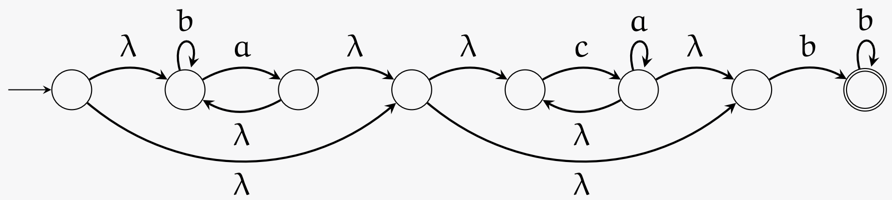
iv) Autómata que acepta $(ca^*)^*$



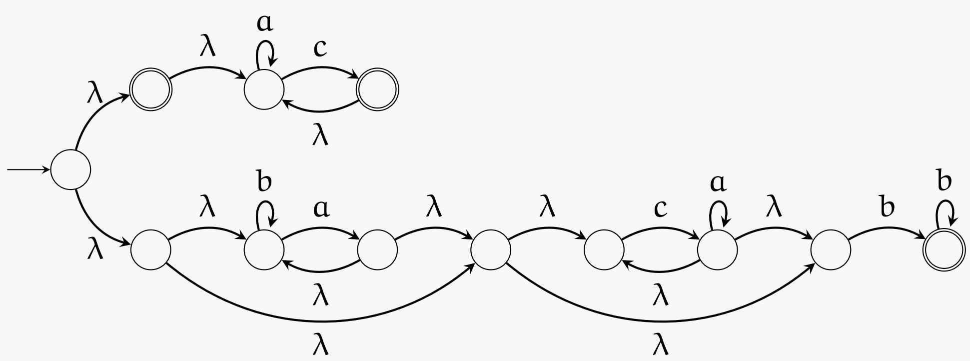
v) Autómata que acepta $(b^*a)^*(ca^*)^*$



vi) Autómata que acepta $(b^*a)^*(ca^*)^*b^+$

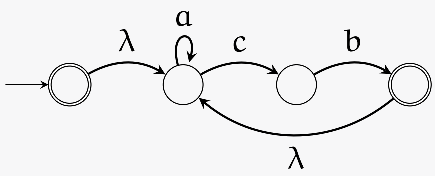


vii) Autómata que acepta $(a^*c)^* \cup (b^*a)^*(ca^*)^*b^+$

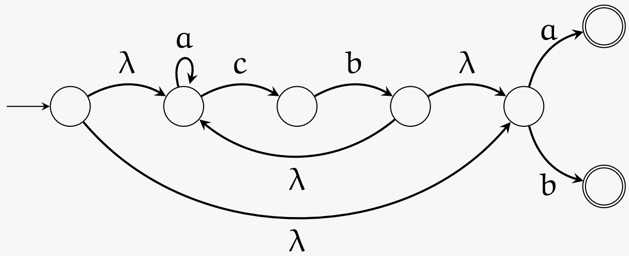


Punto 2: $(a^*cb)^*(a \cup b)(a \cup bc)^*$.

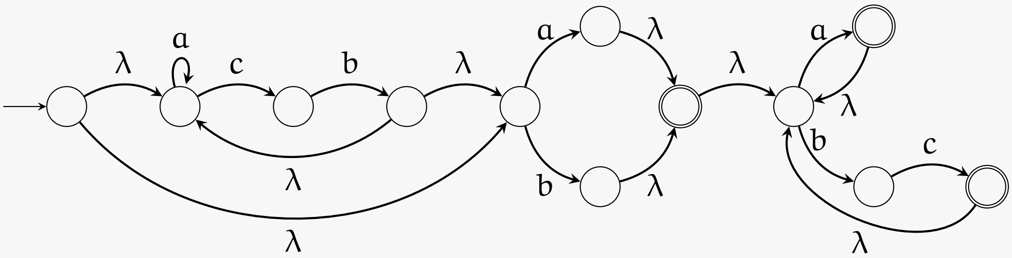
i) Autómata que acepta $(a^*cb)^*$



ii) Autómata que acepta $(a^*cb)^*(a \cup b)$

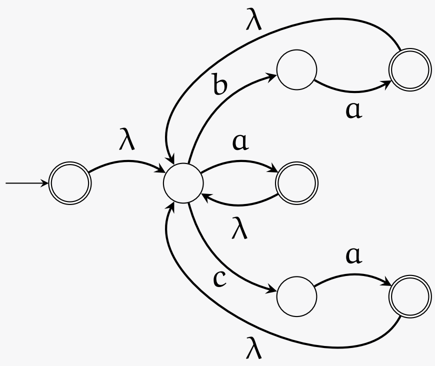


iii) Autómata que acepta $(a^*cb)^*(a \cup b)(a \cup bc)^*$

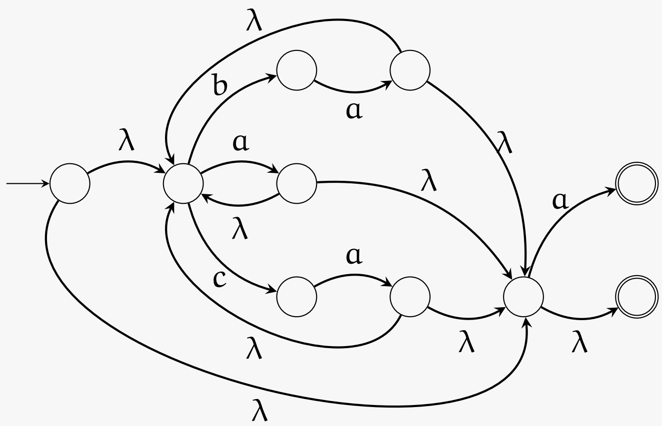


Punto 3: $(a \cup ba \cup ca)^*(\lambda \cup a)b^+c^+$.

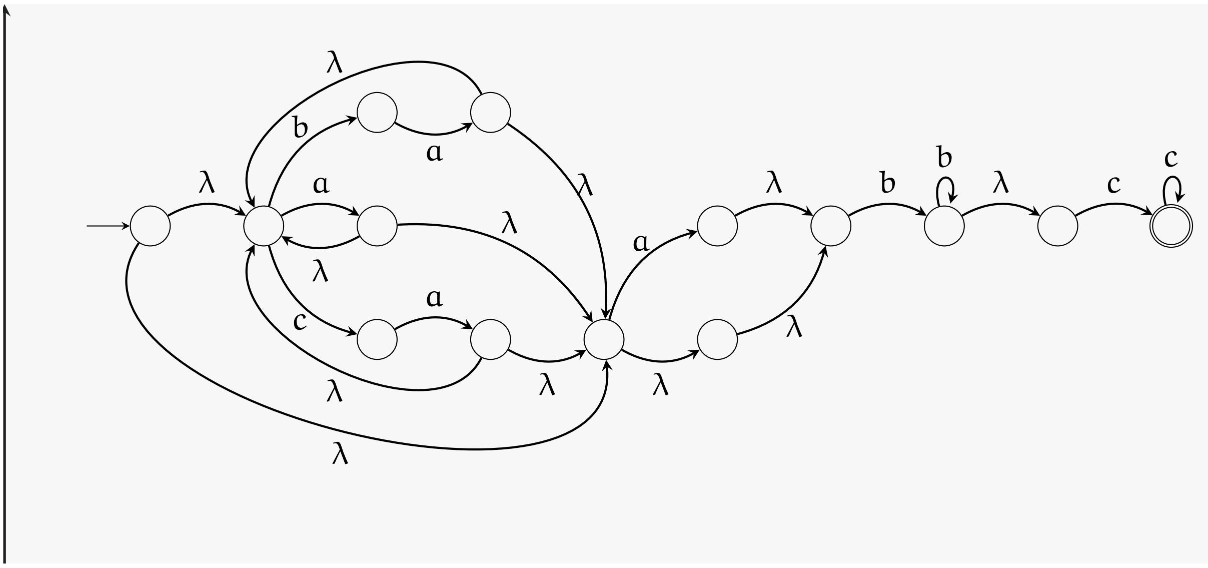
i) Autómata que acepta $(a \cup ba \cup ca)^*$



ii) Autómata que acepta $(a \cup ba \cup ca)^*(\lambda \cup a)$

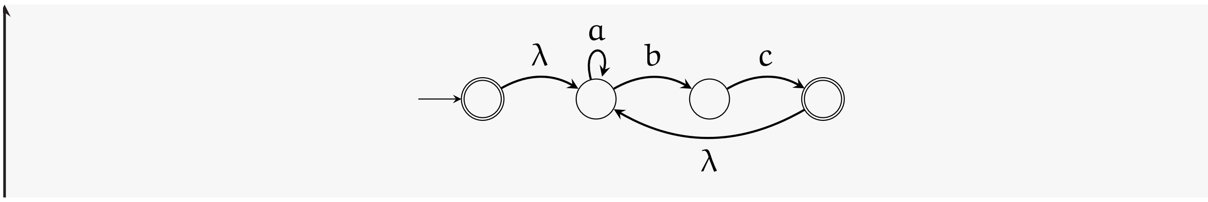


iii) Autómata que acepta $(a \cup ba \cup ca)^*(\lambda \cup a)b^+c^+$

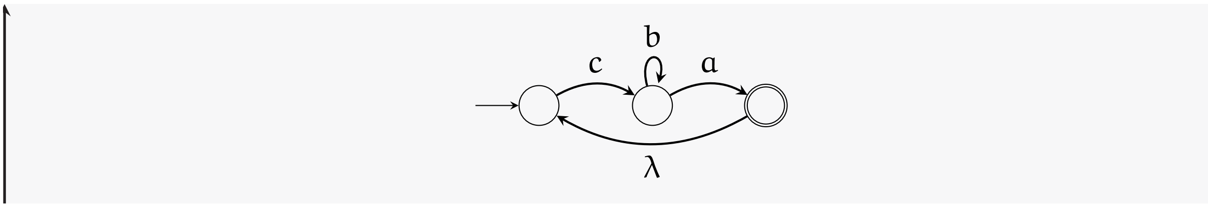


Punto 4: $(a^*bc)^* \cup (cb^*a)^+ \cup (ca \cup cb \cup c^2)^*a^*b^+$.

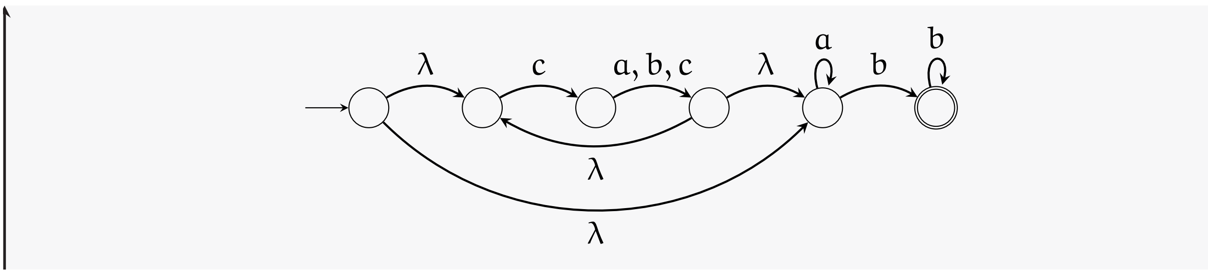
i) Autómata que acepta $(a^*bc)^*$



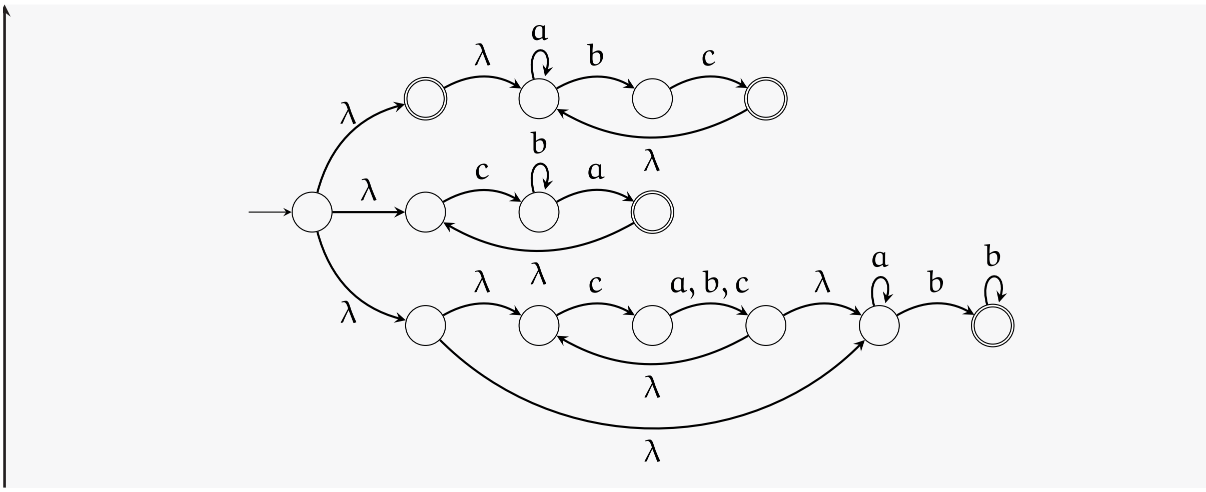
ii) Autómata que acepta $(cb^*a)^+$



iii) Autómata que acepta $(ca \cup cb \cup c^2)^*a^*b^+$

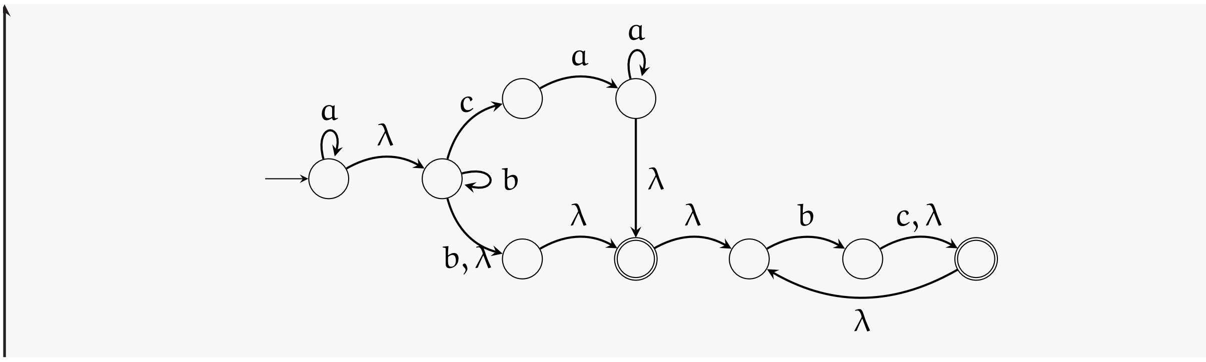


iv) Autómata que acepta $(a^*bc)^* \cup (cb^*a)^+ \cup (ca \cup cb \cup c^2)^* a^* b^+$

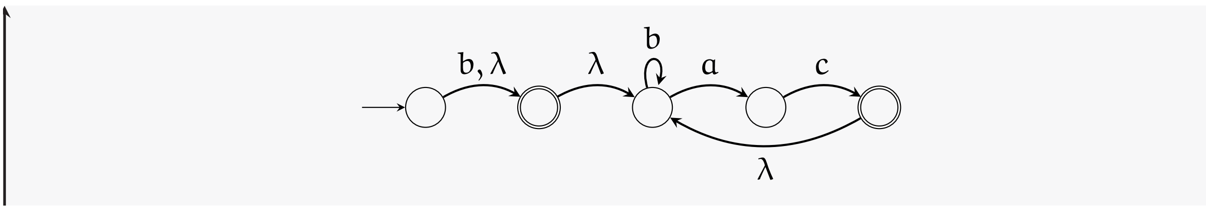


Punto 5: $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^* \cup (b \cup \lambda)(b^*ac)^*$.

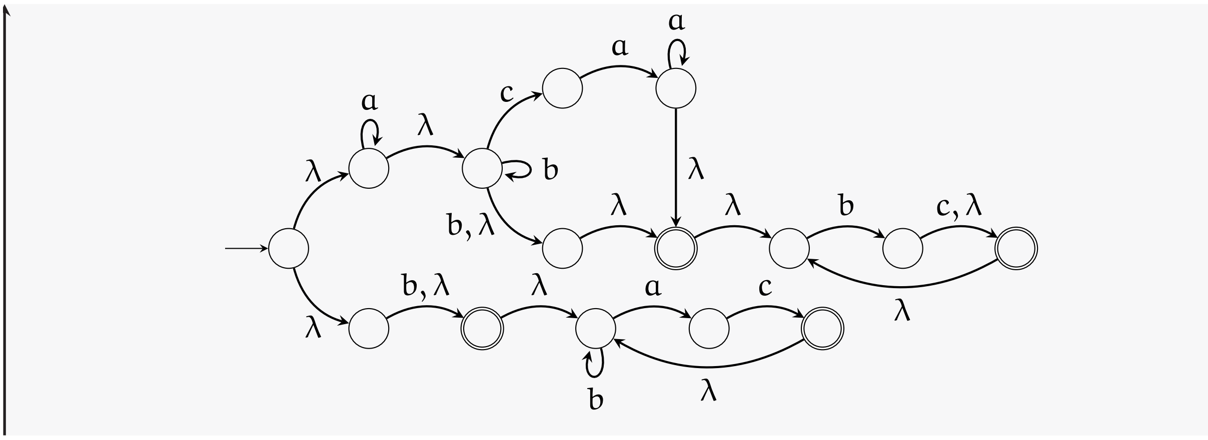
i) Autómata que acepta $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^*$



ii) Autómata que acepta $(b \cup \lambda)(b^*ac)^*$



iii) Autómata que acepta $a^*b^*(ca^+ \cup b \cup \lambda)(b \cup bc)^* \cup (b \cup \lambda)(b^*ac)^*$



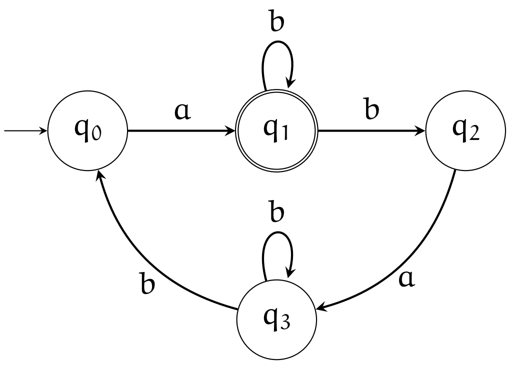
Cabe aclarar que las construcciones realizadas no son únicas ya que este procedimiento es bastante flexible, después de todo las construcciones presentadas aquí tienen bastantes simplificaciones.

♦

2.10 Teorema de Kleene, parte II

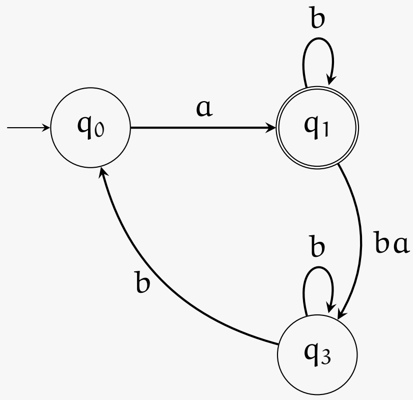
Dado un Autómata Finito, siempre podemos encontrar una expresión regular que sea aceptada por el mismo. Anteriormente nos dimos cuenta que habían problemas al momento de encontrar expresiones regulares para algunos lenguajes mientras que construir un autómata para el mismo era bastante sencillo, en esta sección se explora un algoritmo que nos permite hallar siempre la expresión regular de un lenguaje dado el autómata que acepta ese lenguaje.

Punto 1: Utilizar el procedimiento presentado en la presente sección para encontrar expresiones regulares para los lenguajes aceptados por los siguientes autómatas.

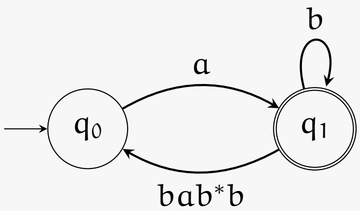
- 

  Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo:

  i) Eliminar estado q_2:



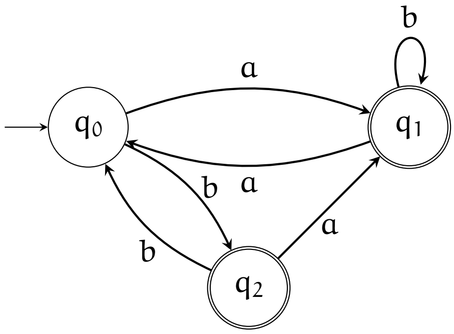
ii) Eliminar estado q_3:



Teniendo en cuenta que $b^*b = b^+$, por simple inspección obtenemos que:

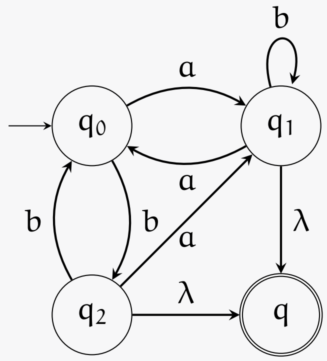
$$L(M) = a(b^* \cup bab^+a)^*$$

•

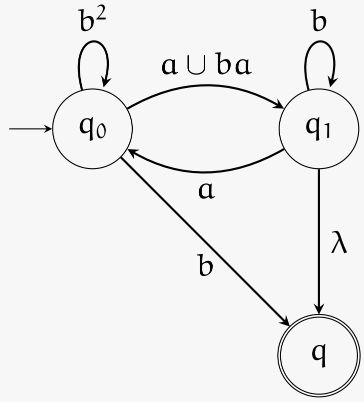


Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo,de paso lo acomodaremos de forma que sea mas sencillo trabajar:

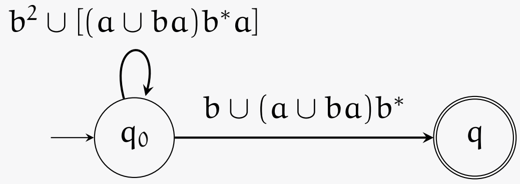
i) Agregamos un nuevo estado de aceptación q:



ii) Eliminar estado q_2:



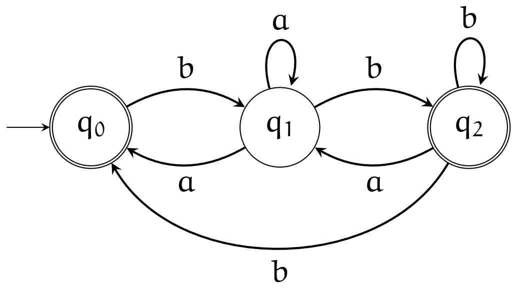
iii) Eliminar estado q_1:



Ahora por simple inspección llegamos a que:

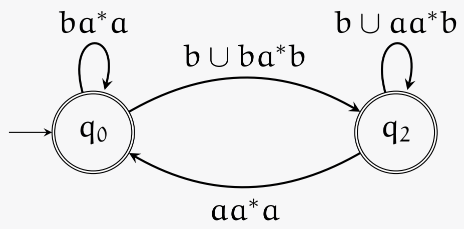
$$L(M) = \left(b^2 \cup [(a \cup ba)b^*a]\right)^* (b \cup (a \cup ba)b^*)$$

•

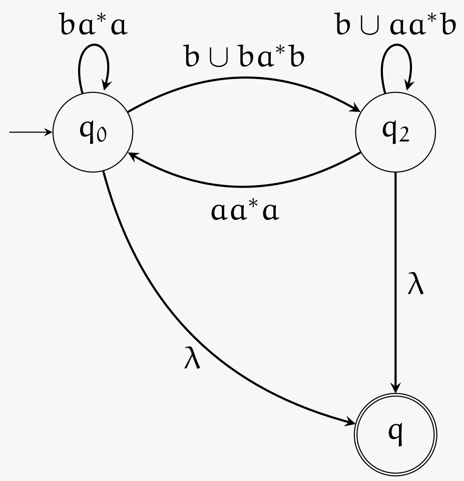


Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo:

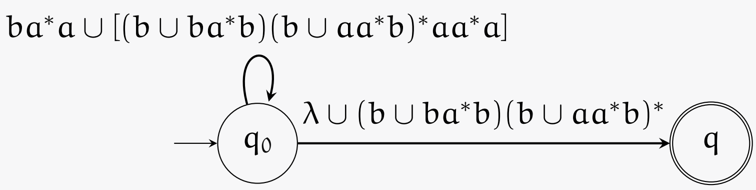
i) Eliminar estado q_1



ii) Agregamos un nuevo estado de aceptación q:



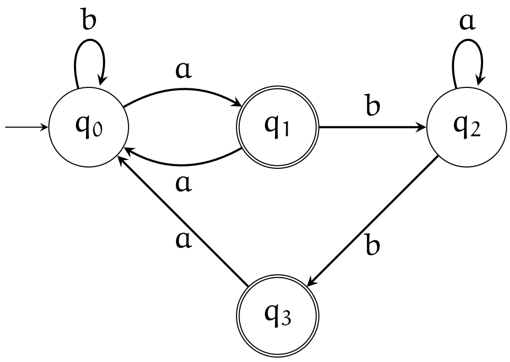
iii) Eliminar estado q_2:



Teniendo en cuenta que $a^*a = aa^* = a^+$, por simple inspección obtenemos que:

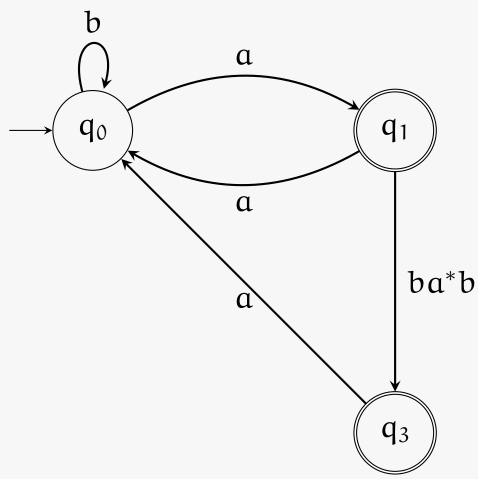
$$L(M) = (ba^*a \cup [(b \cup ba^*b)(b \cup aa^*b)^*aa^*a])^* \,(\lambda \cup (b \cup ba^*b)(b \cup aa^*b)^*)$$

•

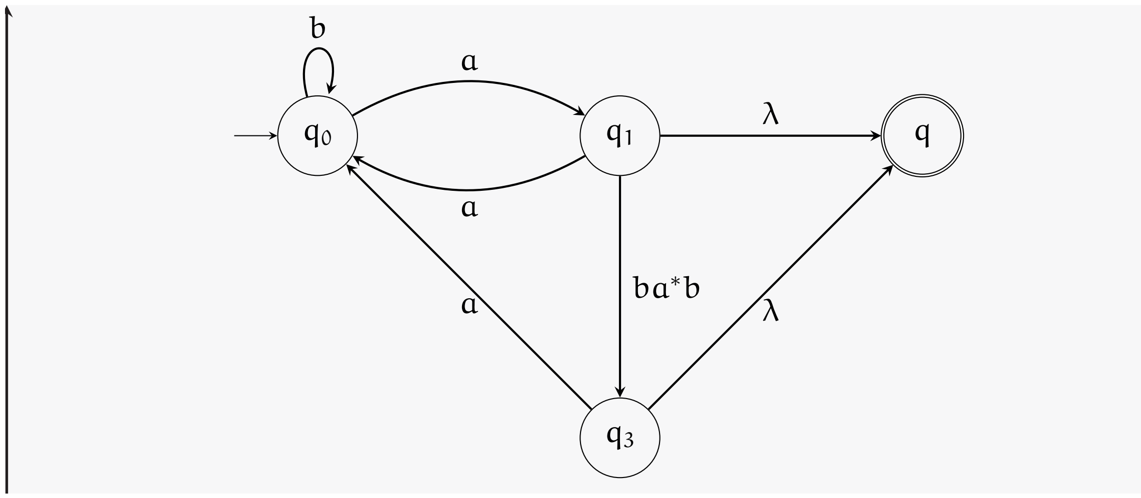


Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo:

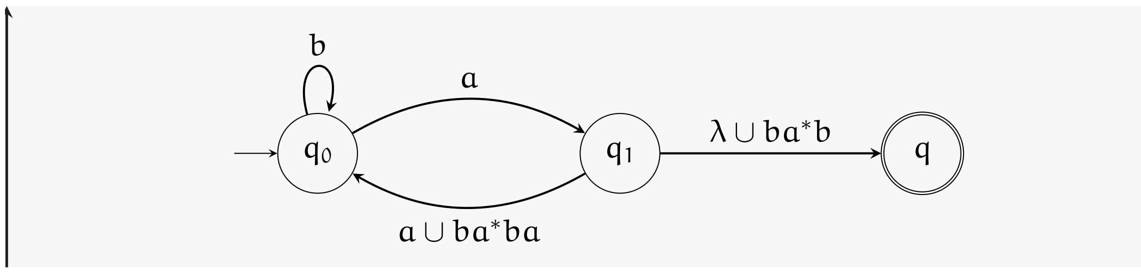
i) Eliminar estado q_2:



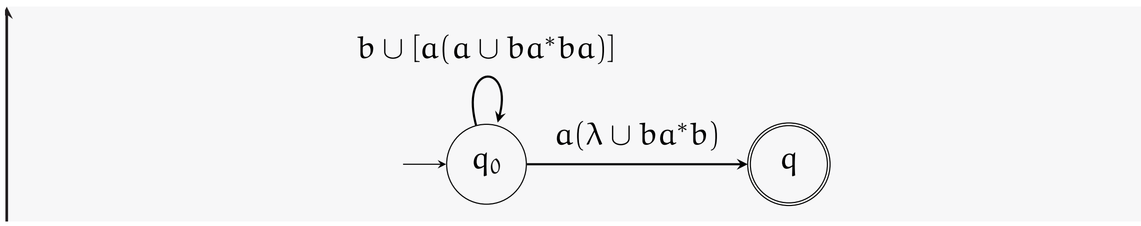
ii) Agregamos un nuevo estado de aceptación q:



iii) Eliminar estado q_3:



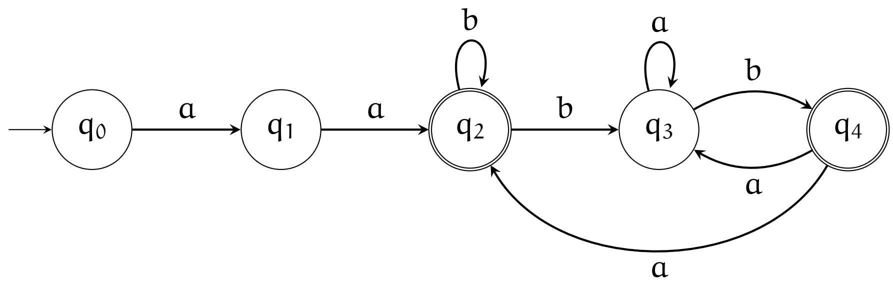
iv) Eliminar estado q_1:



Por simple inspección obtenemos que:

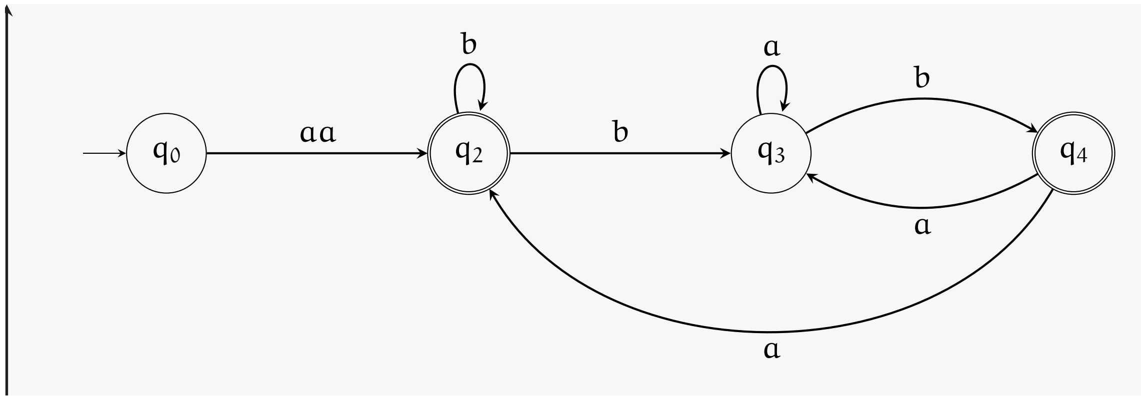
$$L(M) = (b \cup [a(a \cup ba^*ba)])^* \, (a(\lambda \cup ba^*b))$$

•

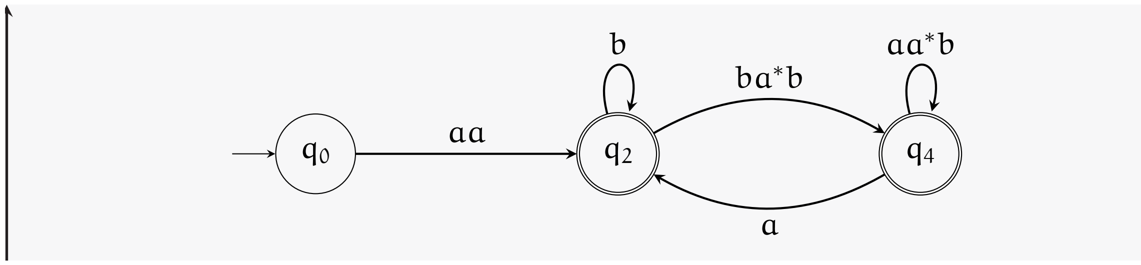


Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo:

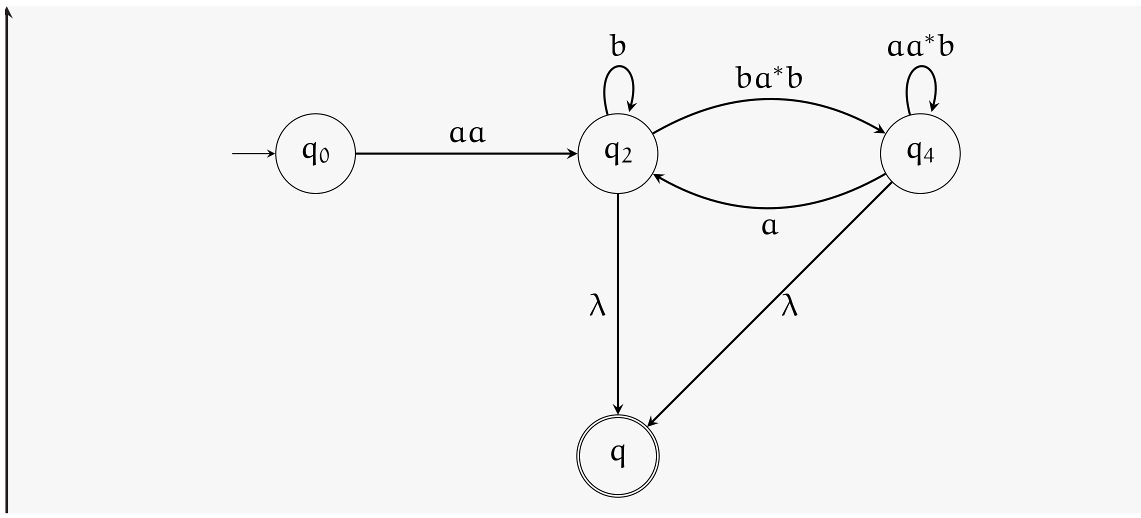
i) Eliminar estado q_1:



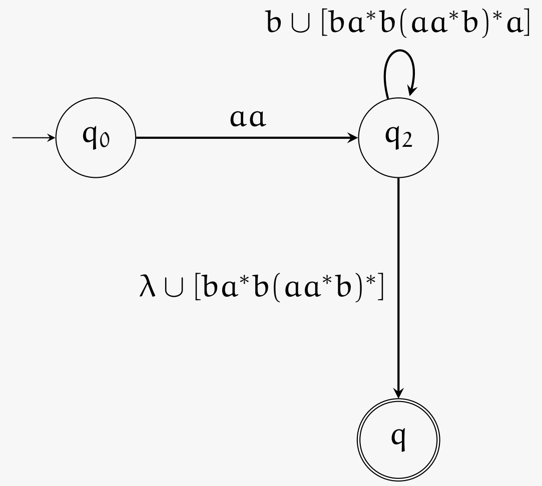
ii) Eliminar estado q_3:



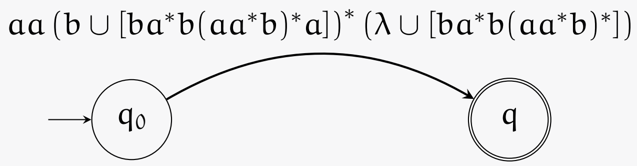
iii) Agregamos un nuevo estado de aceptación q:



iv) Eliminar estado q_4:



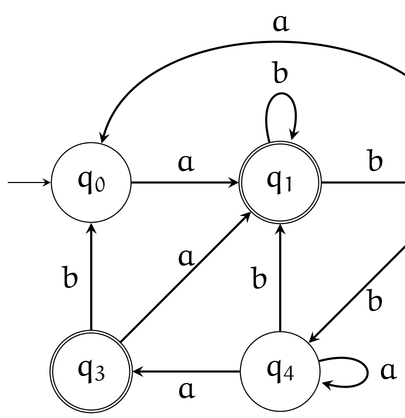
v) Eliminar estado q_2:



Realizando las respectivas simplificaciones y por simple inspección obtenemos que:

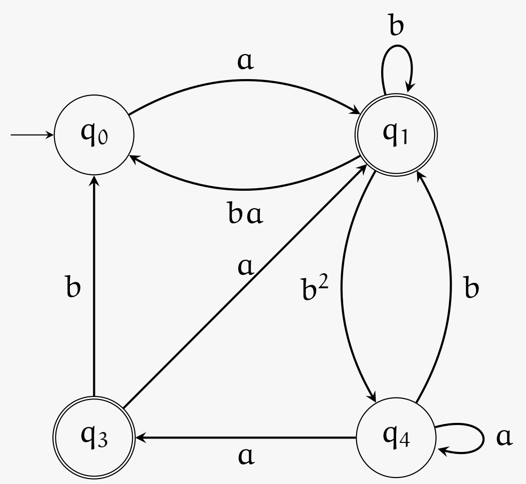
$$L(M) = a^2 \left(b \cup [ba^*b(a^+b)^*a]\right)^* \left(\lambda \cup [ba^*b(a^+b)^*]\right)$$

•

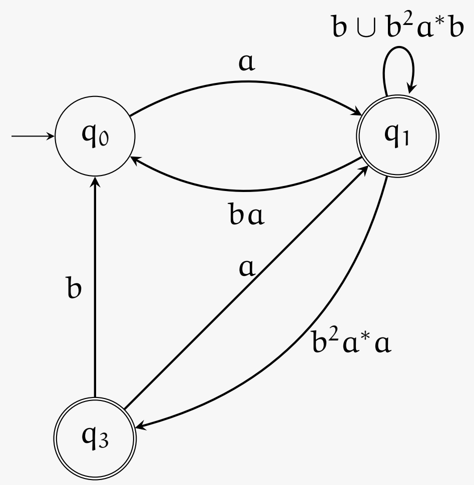


Como el grafo de M ya esta en su forma GEC procedemos con el algoritmo:

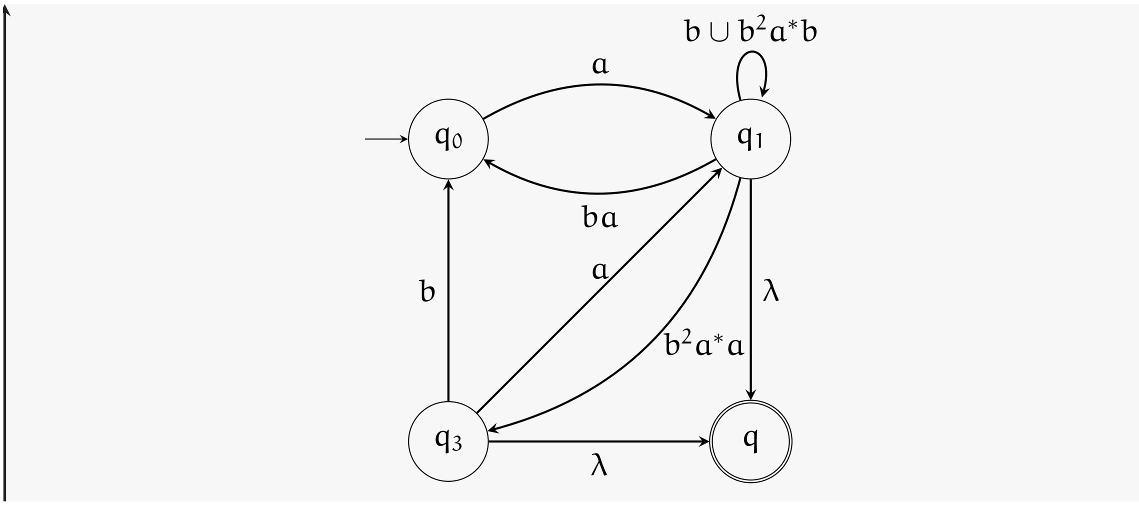
i) Eliminar estado q_2:



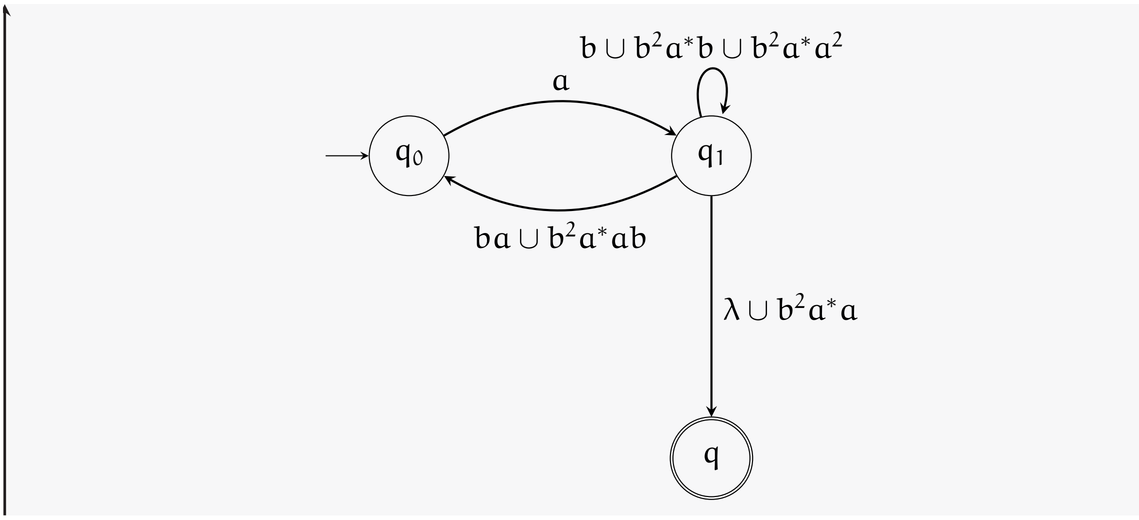
ii) Eliminar estado q_4:



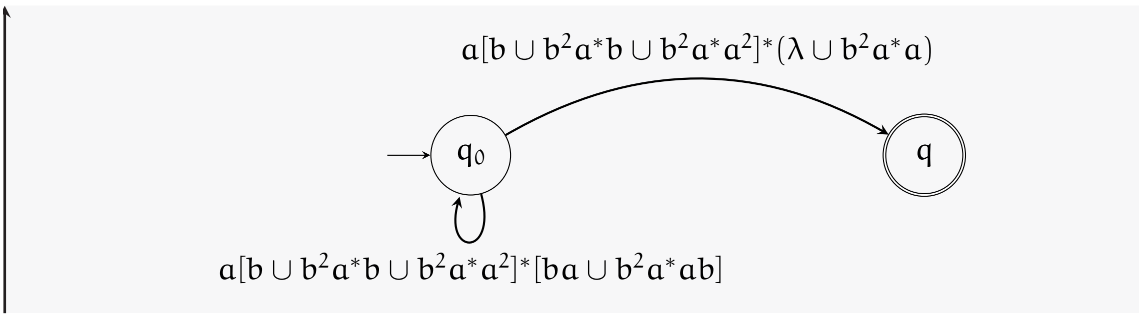
iii) Agregamos un nuevo estado de aceptación q:



iv) Eliminar estado q_3:



v) Eliminar estado q_1:

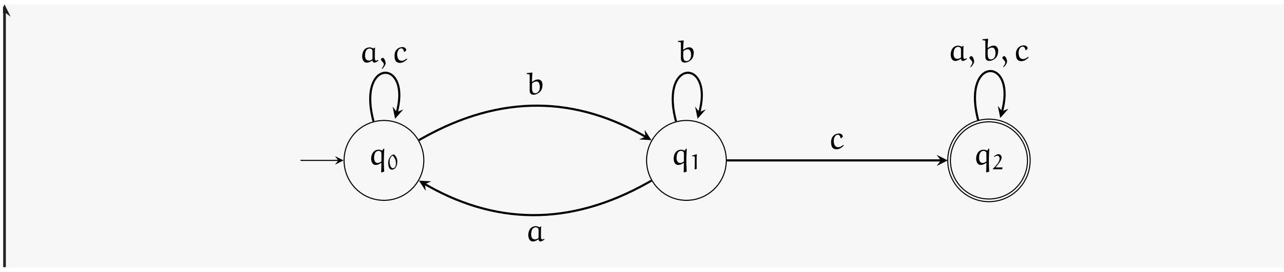


Luego por simple inspección obtenemos que:

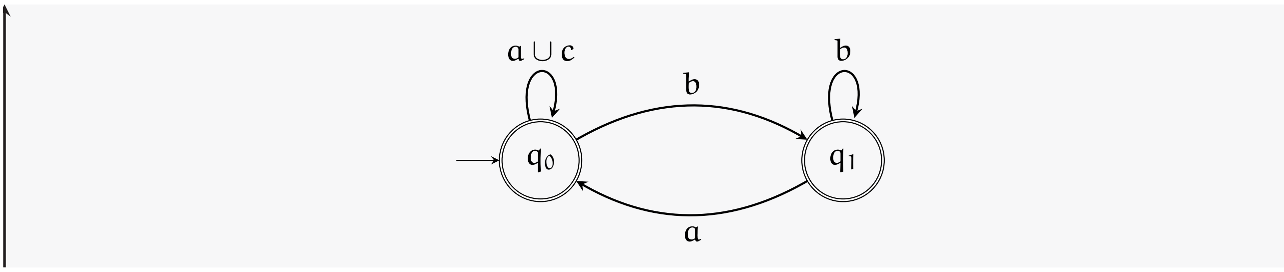
$$L(M) = \left(a[b \cup b^2a^*b \cup b^2a^*a^2]^*[ba \cup b^2a^*ab]\right)^* a[b \cup b^2a^*b \cup b^2a^*a^2]^*(\lambda \cup b^2a^*a)$$

Punto 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.

Primero hagamos el grafo de todas las cadenas que contienen la subcadena bc:

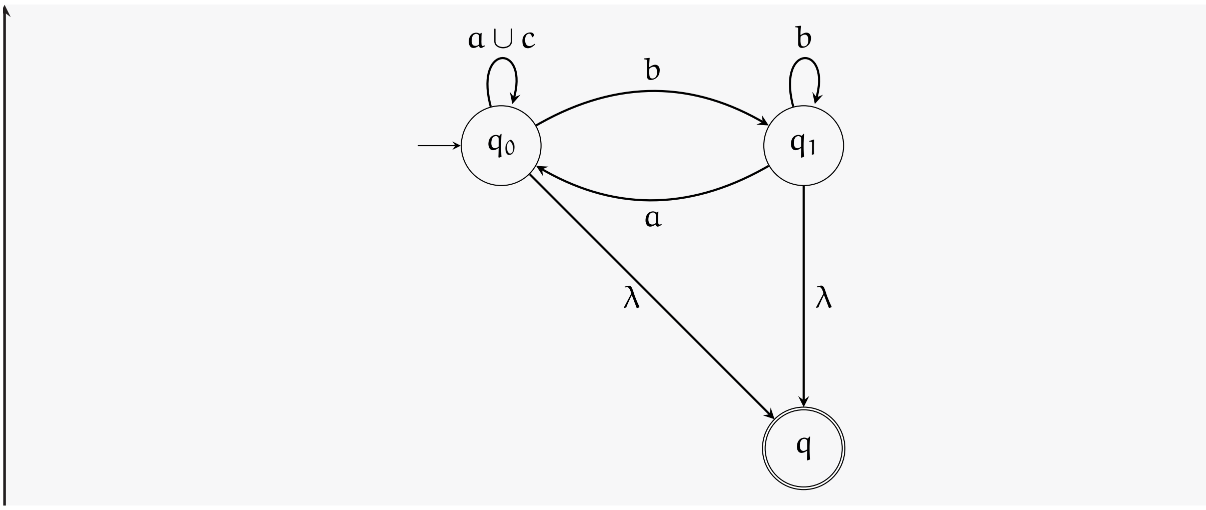


Note que este es un AFD por lo tanto basta considerar el complemento del mismo para obtener el grafo de todas las cadenas que no contienen la subcadena bc :

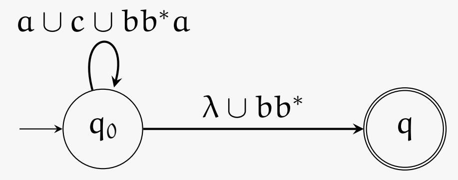


Note que nos saltamos algunos pasos de simplificación ya que al tomar el complemento, el estado q_2 se vuelve un estado limbo y puede ser eliminado. Ademas ya expresamos el grafo M en su forma GEC por lo que podemos proceder con el algoritmo:

i) Agregar nuevo estado de aceptación q:



ii) Eliminar estado q_1:



Ahora por simple inspección podemos observar que:

$$L = L(M) = (a \cup c \cup bb^*a)^*(\lambda \cup bb^*)$$

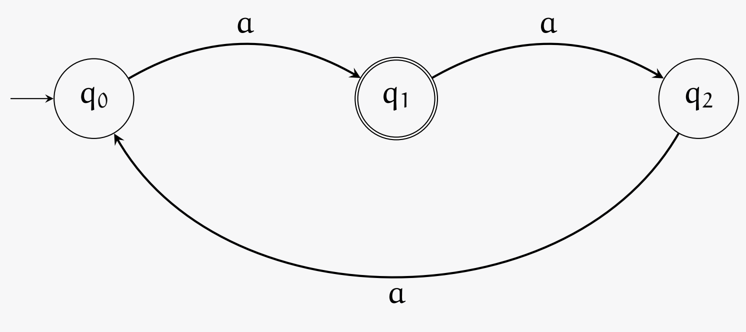
Este seria el lenguaje de todas las cadenas que no contienen la subcadena bc. Note que si realizamos las simplificaciones $bb^* = b^+$ y $\lambda \cup bb^* = \lambda \cup b^+ = b^*$ obtenemos que:

$$L = (a \cup c \cup b^+a)^*b^*$$

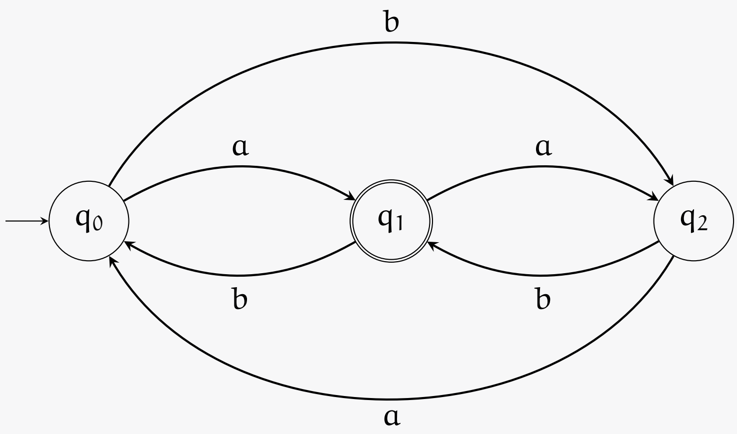
Note que la expresión encontrada es idéntica a una de las expresiones brindadas por las notas en el ultimo ejemplo de la sección 2.2.

Punto 3: Sea $\Sigma = \{a, b\}$ y $L = \{u \in \Sigma^* : [\#_a(u) - \#_b(u)] \equiv 1(\mod 3)\}$. La notación $\#_a(u)$ representa el numero de aes en la cadena u mientras que $\#_b(u)$ es el numero de bes. 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.

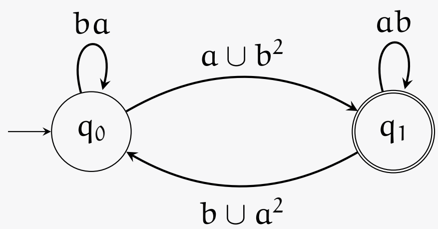
Como estamos restringidos a tres estados basta por medio de cadenas básicas intuir como son las transiciones para M:



Note que si la cadena tiene solo aes basta con ver que el numero sea congruente con 1 modulo 3, y por eso tenemos ese bucle donde cada estado representa la congruencia y es por eso que q_1 es nuestro estado de aceptación. Ahora solo falta ver que pasa con las transiciones para las bes pero esto es sencillo de ver si consideramos cadenas que solo tienen bes a que numero serian congruentes:



Este seria nuestro grafo de M que acepta L; como ya esta en su forma GEC podemos eliminar el estado q_2 según el algoritmo presentado:



De aquí ya por simple inspección podemos observar que la expresión regular para L es:

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

♦

2.11 Teorema de Myhill-Nerode

Esta sección es completamente opcional ya que Korgi no la trata a profundidad, pero puede que algún profesor si lo haga así que haremos las demostraciones correspondientes.

Punto 1: Sea Σ un alfabeto dado, $L \subseteq \Sigma^*$ y I_L la relación de indistinguibilidad definida sobre Σ^*. Para cada $u \in \Sigma^*$, $[u]$ denota la clase de equivalencia de u, determinada por I_L. Demostrar que para todo $u, v \in \Sigma^*$ se cumple

- Si $u \in L$ y $v \in [u]$ entonces $v \in L$.

Demostración. Como $v \in [u]$, por la definición de I_L tenemos que para todo $x \in \Sigma^*$, $ux \in L$ si y solo si $vx \in L$. Particularmente si $x = \lambda$ y como $u \in L$, podemos concluir que $v \in L$. □

- Si uI_Lv entonces uaI_Lva, para todo $a \in Sigma$.

Demostración. Sea $a \in \Sigma$, recordemos que $\Sigma \subset \Sigma^*$, así $a \in \Sigma^*$. Luego por hipótesis como uI_Lv tenemos que para todo $x \in \Sigma^*$, $ux \in L$ si y solo si $vx \in L$. Note que en particular $x = ay$ donde $y \in \Sigma^*$, pero este y es arbitrario, así que podemos decir que para todo $y \in \Sigma^*$, $uay \in L$ si y solo si $vay \in L$. que es lo mismo que decir uaI_Luv. □

Punto 2: Demostrar la afirmación (2.15.4) de la demostración del Teorema de Myhill-Nerode, esto es, demostrar por recursión sobre u, que

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

Demostración. Para el caso $u = \lambda$ tenemos por definición

$$\widehat{\delta_L}([\lambda], \lambda) = [\lambda]$$

Note de manera similar que para el caso $u = a$ con $a \in \Sigma$ tenemos que

$$\widehat{\delta_L}([\lambda], a) = [a]$$

Ahora supongamos que $\widehat{\delta_L}([\lambda], u) = [u]$ y veamos que pasa con ua

$$\begin{aligned}
\widehat{\delta_L}([\lambda], ua) &= \widehat{\delta_L}(\widehat{\delta_L}([\lambda], u), a) && \text{(Propiedad función de transición)}\\
&= \widehat{\delta_L}([u], a) && \text{(Hipótesis recursiva)}\\
&= [ua] && \text{(Definición de la función de transición).}
\end{aligned}$$

Así podemos concluir que para toda cadena $u \in \Sigma^*$ tenemos que $\widehat{\delta_L}([\lambda], u) = [u]$. □

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

- $f(q_0) = q_i$.

Demostración. Por definición de f tenemos que:

$$f(q_0) = [u]$$

donde $\widehat{\delta}(q_0, u) = q_0$ para alguna cadena $u \in \Sigma^*$, pero note que $\widehat{\delta}(q_0, \lambda) = q_0$ para cualquier autómata arbitrario M, así $f(q_0) = [\lambda]$ y como $[\lambda] = q_i$, hemos mostrado que $f(q_0) = q_i$. □

- $q \in F$ si y solo si $f(q) \in F_L$.

Demostración. (⇒) Si $q \in F$, luego $f(q) = [u]$ donde $\widehat{\delta}(q_0, u) = q$, pero como $q \in F$ quiere decir que el $u \in L$ y por definición de F_L concluimos que $f(q) = [u] \in F_L$.
(⇐) Si $f(q) \in F_L$ quiere decir que $f(q) = [u]$ donde $u \in L$, pero como por definición de f, $\widehat{\delta}(q_0, u) = q$ quiere decir que q es un estado de aceptación, y por tanto $q \in F$. □

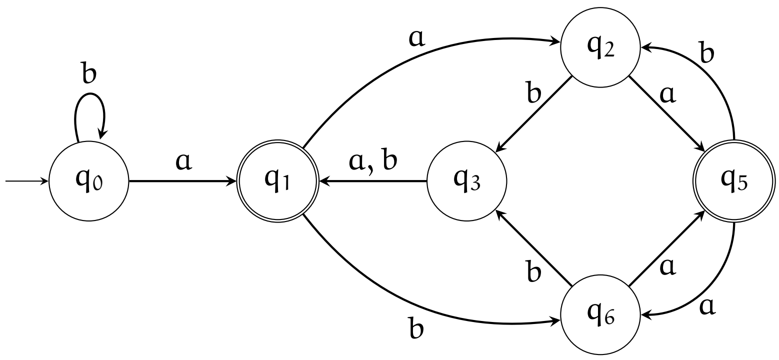
♦

2.12 Algoritmo de minimización de AFDs

En esta sección el propósito sera hallar el autómata con el mínimo numero de estados que acepte un lenguaje en particular. Note que este procedimiento siempre lo haremos partiendo de un AFD que acepte el lenguaje para ver si podemos minimizar el numero de estados o si precisamente ese es el mínimo.

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

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



Para la iteración $i = 1$, marcamos con $\times$ las casillas de estados $\{p, q\}$ donde p es de aceptación y q no, o viceversa:

	q_0	q_1	q_2	q_3	q_5
q_1	$\times$				
q_2		$\times$			
q_3		$\times$			
q_5	$\times$		$\times$	$\times$	
q_6		$\times$			$\times$

Ahora para $i = 2$, examinamos las parejas de casillas no marcadas y sus transiciones:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_0, q_2\}$	$\{q_1, q_5\}$	$\{q_0, q_3\}$
$\{q_0, q_3\}$	$\{q_1, q_1\}$	$\{q_0, q_1\}\times$
$\{q_0, q_6\}$	$\{q_1, q_5\}$	$\{q_0, q_3\}$
$\{q_1, q_5\}$	$\{q_2, q_6\}$	$\{q_6, q_2\}$
$\{q_2, q_3\}$	$\{q_5, q_1\}\times$	$\{q_3, q_1\}\times$
$\{q_2, q_6\}$	$\{q_5, q_5\}$	$\{q_3, q_3\}$
$\{q_3, q_6\}$	$\{q_1, q_5\}$	$\{q_1, q_3\}\times$

Note que marcamos con $\times$ aquellos pares que tenían este símbolo en $i = 1$. Siguiendo con el algoritmo, los pares que tengan alguna $\times$ en su fila deberán ser marcados también:

	q_0	q_1	q_2	q_3	q_5
q_1	$\times$				
q_2		$\times$			
q_3	$\times$	$\times$	$\times$		
q_5	$\times$		$\times$	$\times$	
q_6		$\times$		$\times$	$\times$

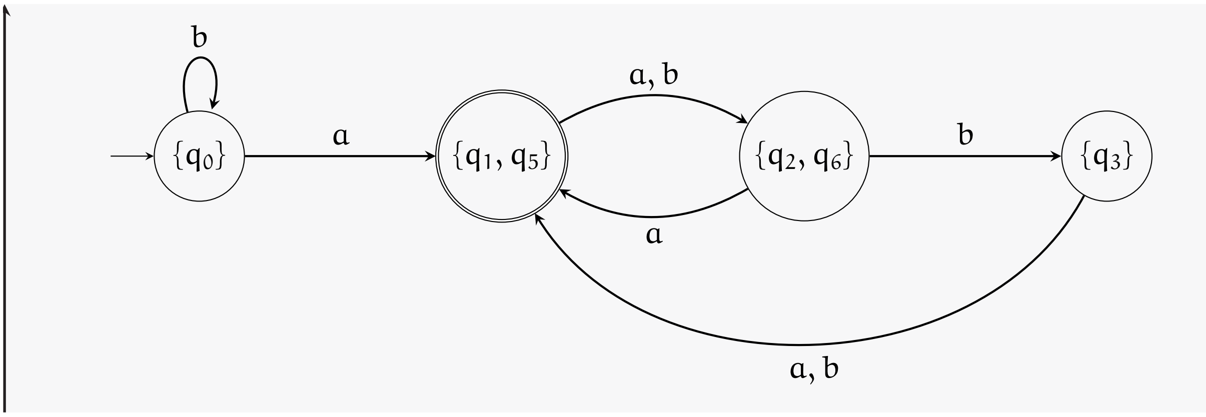
Para $i = 3$ volvemos a repetir el procedimiento con las casillas que aun no han sido marcadas:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_0, q_2\}$	$\{q_1, q_5\}$	$\{q_0, q_3\}\times$
$\{q_0, q_6\}$	$\{q_1, q_5\}$	$\{q_0, q_3\}\times$
$\{q_1, q_5\}$	$\{q_2, q_6\}$	$\{q_6, q_2\}$
$\{q_2, q_6\}$	$\{q_5, q_5\}$	$\{q_3, q_3\}$

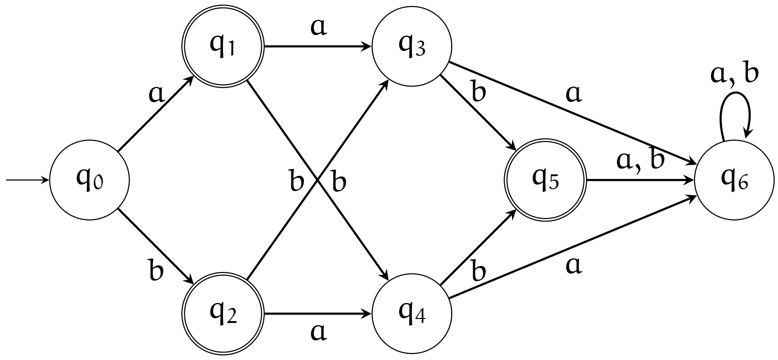
y por el algoritmo nos queda la siguiente tabla:

	q_0	q_1	q_2	q_3	q_5
q_1	$\times$				
q_2	$\times$	$\times$			
q_3	$\times$	$\times$	$\times$		
q_5	$\times$		$\times$	$\times$	
q_6	$\times$	$\times$		$\times$	$\times$

Ahora para $i = 4$ observe que no se pueden marcar mas casillas, de esta manera obtenemos que $q_1 \approx q_5$ y $q_2 \approx q_6$. Luego el autómata cociente tiene los estados $\{q_0\}, \{q_3\}, \{q_1, q_5\}$ y $\{q_2, q_6\}$. De esta manera el grafo del autómata minimizado es el siguiente:



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



Para la iteración $i = 1$ tenemos que:

	q_0	q_1	q_2	q_3	q_4	q_5
q_1	$\times$					
q_2	$\times$					
q_3		$\times$	$\times$			
q_4		$\times$	$\times$			
q_5	$\times$			$\times$	$\times$	
q_6		$\times$	$\times$			$\times$

Ahora en $i = 2$ examinamos las parejas de estados que no fueron tachadas:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a\}$	$\{\delta(p, b), \delta(q, b\}$
$\{q_0, q_3\}$	$\{q_1, q_6\}\times$	$\{q_2, q_5\}$
$\{q_0, q_4\}$	$\{q_1, q_6\}\times$	$\{q_2, q_5\}$
$\{q_0, q_6\}$	$\{q_1, q_6\}\times$	$\{q_2, q_6\}\times$
$\{q_1, q_2\}$	$\{q_3, q_4\}$	$\{q_4, q_3\}$
$\{q_1, q_5\}$	$\{q_3, q_6\}$	$\{q_4, q_6\}$
$\{q_2, q_5\}$	$\{q_4, q_6\}$	$\{q_3, q_6\}$
$\{q_3, q_4\}$	$\{q_6, q_6\}$	$\{q_5, q_5\}$
$\{q_3, q_6\}$	$\{q_6, q_6\}$	$\{q_5, q_6\}\times$
$\{q_4, q_6\}$	$\{q_6, q_6\}$	$\{q_5, q_6\}\times$

Siguiendo el algoritmo tachamos las casillas marcadas en la tabla:

	q_0	q_1	q_2	q_3	q_4	q_5
q_1	$\times$					
q_2	$\times$					
q_3	$\times$	$\times$	$\times$			
q_4	$\times$	$\times$	$\times$			
q_5	$\times$			$\times$	$\times$	
q_6	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

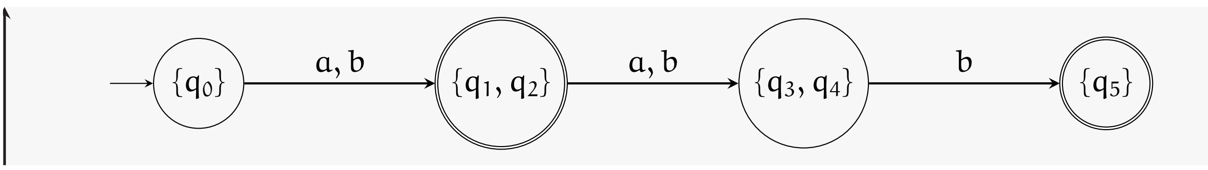
Ahora para $i = 3$ estudiamos las casillas no tachadas en el anterior paso:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a\}$	$\{\delta(p, b), \delta(q, b\}$
$\{q_1, q_2\}$	$\{q_3, q_4\}$	$\{q_4, q_3\}$
$\{q_1, q_5\}$	$\{q_3, q_6\}\times$	$\{q_4, q_6\}\times$
$\{q_2, q_5\}$	$\{q_4, q_6\}\times$	$\{q_3, q_6\}\times$
$\{q_3, q_4\}$	$\{q_6, q_6\}$	$\{q_5, q_5\}$

De esta forma tenemos que tachar las siguientes casillas:

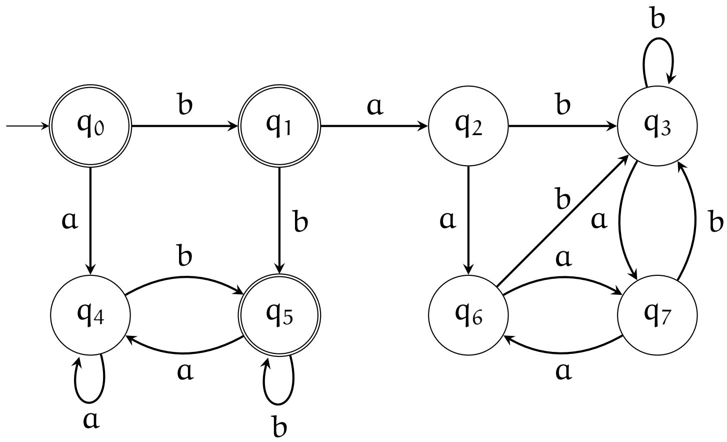
	q_0	q_1	q_2	q_3	q_4	q_5
q_1	$\times$					
q_2	$\times$					
q_3	$\times$	$\times$	$\times$			
q_4	$\times$	$\times$	$\times$			
q_5	$\times$	$\times$	$\times$	$\times$	$\times$	
q_6	$\times$	$\times$	$\times$	$\times$	$\times$	$\times$

Note que para $i = 4$ ya no podemos tachar casillas nuevas, así $q_1 \approx q_2$ y $q_3 \approx q_4$. Luego el autómata cociente tiene los estados $\{q_0\}, \{q_1, q_2\}, \{q_3, q_4\}, \{q_5\}$ y $\{q_6\}$. De esta manera obtenemos que el grafo autómata minimizado es:



Note que simplificamos el grafo ya que $\{q_6\}$ actuá como limbo al igual que en el autómata original, pero recordemos que para el proceso de minimización es importante tener **TODOS** los estados en consideración y por eso al principio es ilustrado en el grafo para mayor claridad.

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



Para la iteración $i = 1$ tenemos

	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_1							
q_2	$\times$	$\times$					
q_3	$\times$	$\times$					
q_4	$\times$	$\times$					
q_5			$\times$	$\times$	$\times$		
q_6	$\times$	$\times$				$\times$	
q_7	$\times$	$\times$				$\times$	

Ahora en $i = 2$ examinamos las parejas de estados no tachadas:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_0, q_1\}$	$\{q_4, q_2\}$	$\{q_1, q_5\}$
$\{q_0, q_5\}$	$\{q_4, q_4\}$	$\{q_1, q_5\}$
$\{q_1, q_5\}$	$\{q_2, q_4\}$	$\{q_5, q_5\}$
$\{q_2, q_3\}$	$\{q_6, q_7\}$	$\{q_3, q_3\}$
$\{q_2, q_4\}$	$\{q_6, q_4\}$	$\{q_3, q_5\}\times$
$\{q_2, q_6\}$	$\{q_6, q_7\}$	$\{q_3, q_3\}$
$\{q_2, q_7\}$	$\{q_6, q_6\}$	$\{q_3, q_3\}$
$\{q_3, q_4\}$	$\{q_7, q_4\}$	$\{q_3, q_5\}\times$
$\{q_3, q_6\}$	$\{q_7, q_3\}$	$\{q_3, q_3\}$
$\{q_3, q_7\}$	$\{q_7, q_6\}$	$\{q_3, q_3\}$
$\{q_4, q_6\}$	$\{q_4, q_7\}$	$\{q_5, q_3\}\times$
$\{q_4, q_7\}$	$\{q_4, q_6\}$	$\{q_5, q_3\}\times$
$\{q_6, q_7\}$	$\{q_7, q_6\}$	$\{q_3, q_3\}$

Siguiendo el algoritmo

	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_1							
q_2	$\times$	$\times$					
q_3	$\times$	$\times$					
q_4	$\times$	$\times$	$\times$	$\times$			
q_5			$\times$	$\times$	$\times$		
q_6	$\times$	$\times$			$\times$	$\times$	
q_7	$\times$	$\times$			$\times$	$\times$	

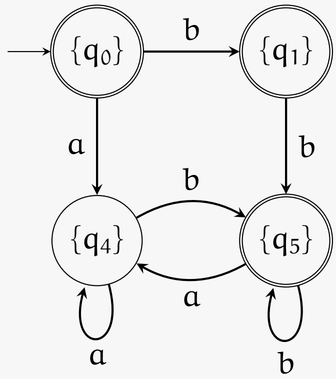
Ahora para $i = 3$, tenemos:

$\{p, q\}$	$\{\delta(p, a), \delta(q, a)\}$	$\{\delta(p, b), \delta(q, b)\}$
$\{q_0, q_1\}$	$\{q_4, q_2\}\times$	$\{q_1, q_5\}$
$\{q_0, q_5\}$	$\{q_4, q_4\}\times$	$\{q_1, q_5\}$
$\{q_1, q_5\}$	$\{q_2, q_4\}\times$	$\{q_5, q_5\}$
$\{q_2, q_3\}$	$\{q_6, q_7\}$	$\{q_3, q_3\}$
$\{q_2, q_6\}$	$\{q_6, q_7\}$	$\{q_3, q_3\}$
$\{q_2, q_7\}$	$\{q_6, q_6\}$	$\{q_3, q_3\}$
$\{q_3, q_6\}$	$\{q_7, q_3\}$	$\{q_3, q_3\}$
$\{q_3, q_7\}$	$\{q_7, q_6\}$	$\{q_3, q_3\}$
$\{q_6, q_7\}$	$\{q_7, q_6\}$	$\{q_3, q_3\}$

y siguiendo el algoritmo

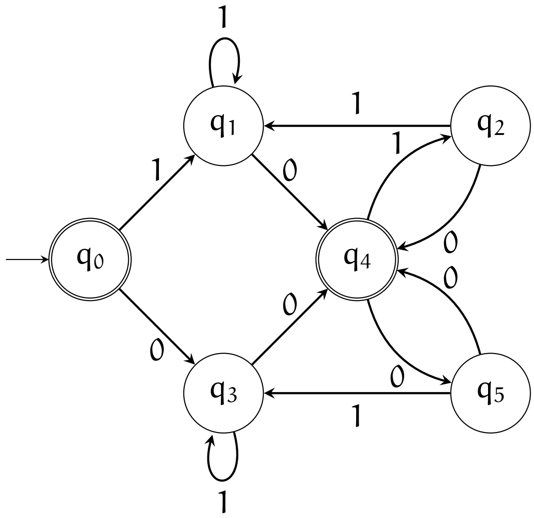
	q_0	q_1	q_2	q_3	q_4	q_5	q_6
q_1	$\times$						
q_2	$\times$	$\times$					
q_3	$\times$	$\times$					
q_4	$\times$	$\times$	$\times$	$\times$			
q_5	$\times$	$\times$	$\times$	$\times$	$\times$		
q_6	$\times$	$\times$			$\times$	$\times$	
q_7	$\times$	$\times$			$\times$	$\times$	

Note que para $i = 4$ ya no podemos tachar mas casillas, así $q_2 \approx q_3 \approx q_6 \approx q_7$. Luego el autómata cociente tiene los estados $\{q_0\}, \{q_1\}, \{q_4\}, \{q_5\}$ y $\{q_2, q_3, q_6, q_7\}$. De esta manera el grafo del autómata minimizado es



Note que esta minimización tiene bastante sentido, ya que por simple inspección podemos notar que toda la parte derecha conectada a q_2 actuá como un limbo, así que puede ser borrado, por esto mismo, en la presentación del grafo minimizado omitimos el estado $\{q_2, q_3, q_6, q_7\}$.

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



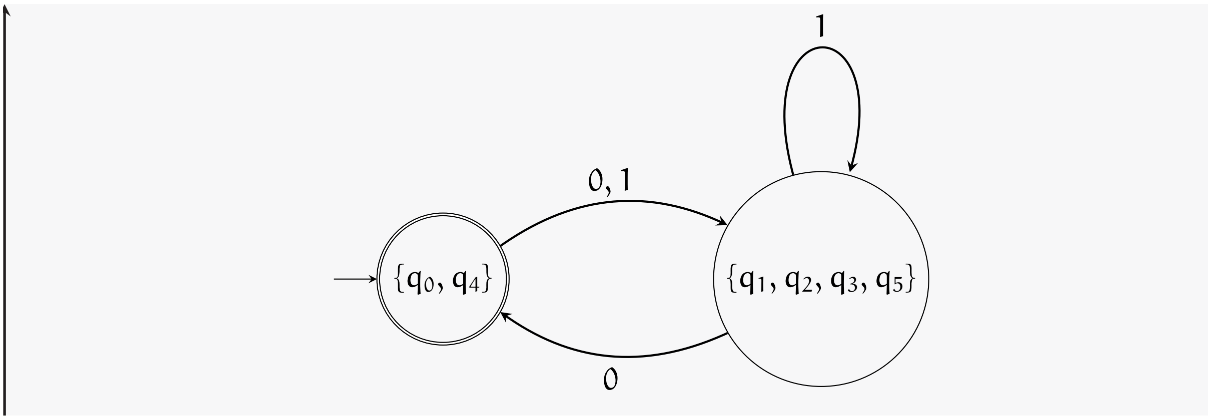
Para la iteración $i = 1$, tenemos

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

Para $i = 2$ examinamos las parejas no tachadas

$\{p, q\}$	$\{\delta(p, 0), \delta(q, 0)\}$	$\{\delta(p, 1), \delta(q, 1)\}$
$\{q_0, q_4\}$	$\{q_3, q_5\}$	$\{q_1, q_2\}$
$\{q_1, q_2\}$	$\{q_4, q_4\}$	$\{q_1, q_1\}$
$\{q_1, q_3\}$	$\{q_4, q_4\}$	$\{q_1, q_3\}$
$\{q_1, q_5\}$	$\{q_4, q_4\}$	$\{q_1, q_3\}$
$\{q_2, q_3\}$	$\{q_4, q_4\}$	$\{q_1, q_3\}$
$\{q_2, q_5\}$	$\{q_4, q_4\}$	$\{q_1, q_3\}$
$\{q_3, q_5\}$	$\{q_4, q_4\}$	$\{q_3, q_3\}$

Note que curiosamente no podemos tachar ninguna casilla nueva, por lo que $q_0 \approx q_4$ y $q_1 \approx q_2 \approx q_3 \approx q_5$, luego el grafo del autómata minimizado esta dado por



- Alfabeto $\Sigma = \{a, b, c\}$.
- 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 bes consecutivas.

Punto 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).

Punto 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).

Ya casi termino este capitulo 2 de mierda, vaina tan eterna :p