Ejercicios de Introducción a la Teoría de la Computación

Edgar Santiago Ochoa
eochoaq@unal.edu.co

Mateo Andrés Manosalva
mmanosalva@unal.edu.co

Índice general

Capítulo 1

Alfabetos, cadenas y lenguajes

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: 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: Para generalizar la propiedad consideramos la cadena:

$$(u_1u_2...u_n)$$

luego:

$$\begin{aligned}(u_1u_2...u_n)^R &= (u_{n-1}u_n)^R(u_1u_2...u_{n-2})^R \\ &= u_n^R u_{n-1}^R(u_1u_2...u_{n-2})^R\end{aligned}$$

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

$$(u_1u_2...u_n)^R = u_n^R u_{n-1}^R ... u_2^R u_1^R$$

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

Punto 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:

Punto 2: $u = acbd$ aplicando el análisis anterior encontramos $v = acca$

Punto 3: $u = dabcdd$, nuevamente encontramos $v = dabdaa$

Punto 4: $dcaddd$, finalmente $v = dcbaaa$

⧫

1.3. Concatenación de lenguajes

Punto 1: Considere $\Sigma = \{a, b\}$ los lenguajes $A = \emptyset$ y $B = a$, por definición de concatenación de lenguajes:

$$AB = \emptyset = BA$$

Punto 2: 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:

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

Punto 1: 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: 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:

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

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

□

3) *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:

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

□

4) *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:

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

□

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

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

□

Punto 2: 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.

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

Y continuando así llegamos a:

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

De manera análoga se ve la intersección.

♦

Capítulo 2

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:

- 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.

- 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).

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

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

  y acabamos.

- 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.

✎ 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:

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

✎ 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 bes, pero esta expresión no contempla las cadenas del estilo $bb \ldots b$, para esto basta concatenar estas cadenas al final, obteniendo finalmente la expresión:

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

Punto 2:

✎ 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 $2(0 \cup 1 \cup 2)^* 1$.

✎ 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:

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

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

✎ 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:

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

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

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

✎ 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:

✎ 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 $(0 \cup 1)^*(01 \cup 10)(0 \cup 1)^*$.

✎ 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:

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

✎ 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 $(0 \cup 1)^4(0 \cup 1)^*$.

✎ 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 $(0 \cup 1)^4 1(0 \cup 1)^*$.

✎ 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.

✎ 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.

✎ 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:

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

✎ 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.

✎ 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 $((0 \cup 1)^3)^*$. Recuerde que 0 es múltiplo de 3 por eso usamos el $*$ para asegurar la cadena λ.

✎ 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:

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

✎ 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:

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

✎ 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.

⧫

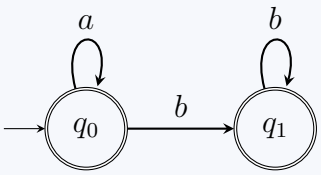
2.2. Diseño de autómatas

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:

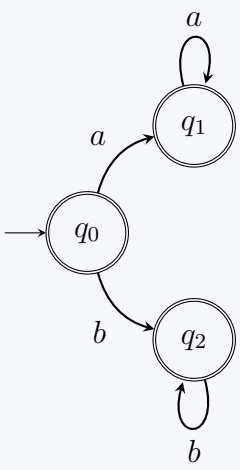
✎ Como la cadena mínima es λ entonces el estado inicial tiene que ser de aceptación, luego como también se aceptan *aes* arbitrarias estas pueden ser aceptadas por medio de un bucle. Apenas aparezca una b el autómata cambiara de estado pero ese seria también de aceptación, incluyendo un bucle para las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

Tikz:.



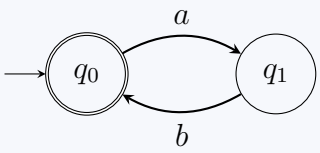
✎ Nuevamente el estado inicial es de aceptación ya que λ pertenece a el lenguaje, luego basta con tomar dos caminos para el caso donde sean cadenas de aes y el de cadenas de bes:

Tikz:.



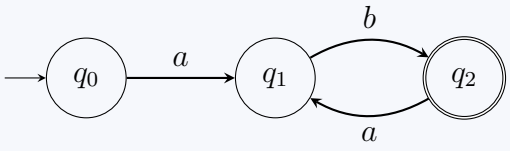
✎ Note que todas las cadenas de este lenguaje son de la forma $ab \dots ab$, es decir siempre son bloques ab y todas las cadenas empiezan en a y terminan en b:

Tikz:.



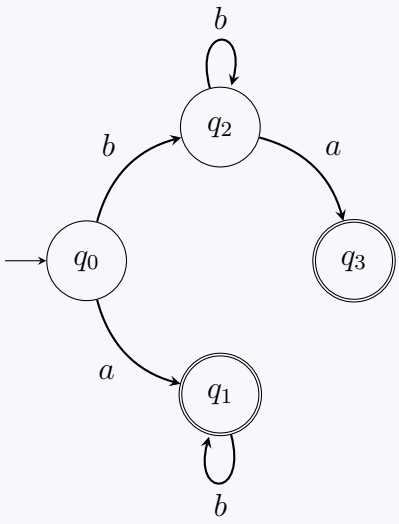
✎ Bastante similar a la anterior excepto que la cadena mínima aceptada es ab debido al $+$, así que forzamos esa cadena:

Tikz:.



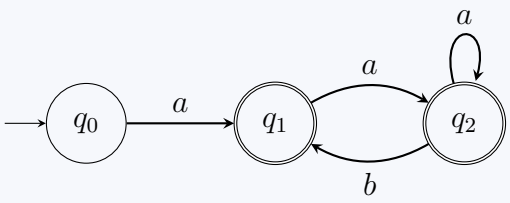
✎ Note que debido a la expresión se forman dos caminos, uno son las cadenas que empiezan por a y luego tienen una cantidad de bes arbitrarias. El otro son aquellas que comienzan por un numero de bes arbitrarias pero están forzadas a terminar en a para ser aceptadas:

Tikz:.



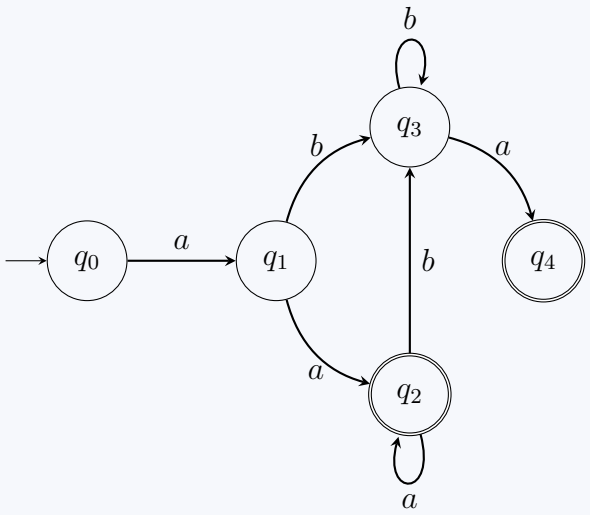
✎ Observe que las cadena mínima de este lenguaje tiene una sola a así que el estado inicial no es de aceptación, luego note que en ambos casos de la $*$ el elemento es una a, por lo que forzamos dos aes y posteriormente realizamos respectivamente el bucle de aes y de $abes$:

Tikz:.



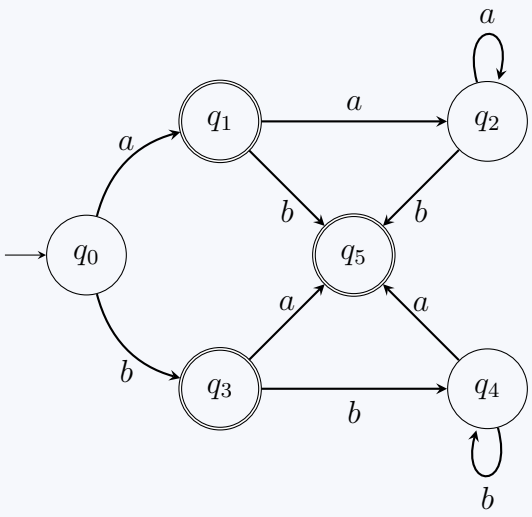
✎ Observe que la cadena mínima es aa entonces debemos forzar ese camino. Ahora note que en ese camino podemos generar el bucle para a^+. Note que hay dos posibilidades para las bes, o inicia la cadena con solo una a y luego tiene una cantidad arbitraria de bes o empieza con aes arbitrarias y luego tiene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

Tikz:.



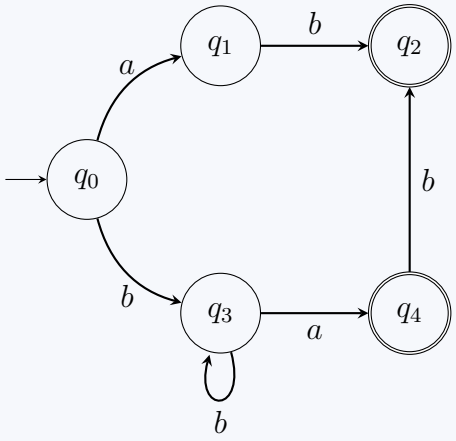
✎ Primero notemos que las cadenas mínimas son a y b, así que debemos forzar dos caminos que las acepten. Ahora observe que si sale una a existen dos posibilidades, es seguida inmediatamente de una b o salen aes arbitrarias y luego si una b. Note que este análisis también es valido respectivamente con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

Tikz:.



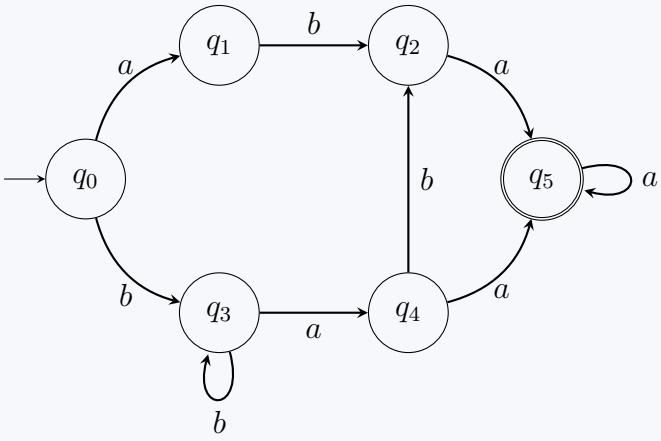
✎ Observe que nuestras dos cadenas mínimas de aceptación son ab y ba así que forzamos esos dos caminos. Posteriormente para hacer que aparezcan las bes arbitrarias podemos agregar un bucle en el camino donde generamos ba. Si lo dejáramos hasta ahí nos faltarían las cadenas de la forma b^*ab pero esto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

Tikz:.



✎ 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^+:

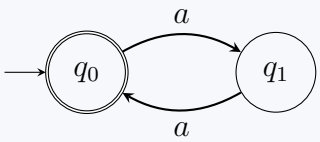
Tikz:.



Punto 2:

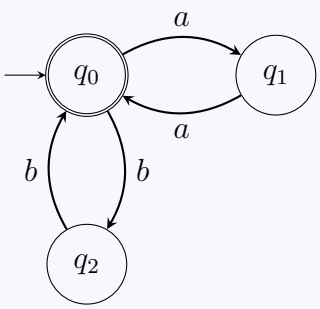
✎ Recordemos que si por ejemplo queremos construir un autómata que acepte el lenguaje de las cadenas que tienen una cantidad par de aes hacemos la siguiente construcción:

Tikz:.



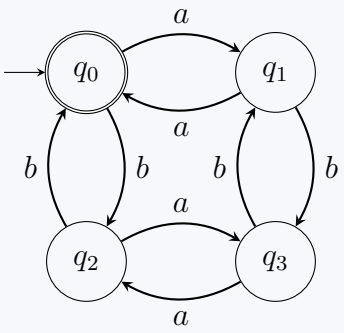
Note como la construcción para cantidad par de bes es la misma entonces basta simplemente con agregar un camino para estas en el autómata ya diseñado hacia un nuevo estado:

Tikz:.



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:

Tikz:.

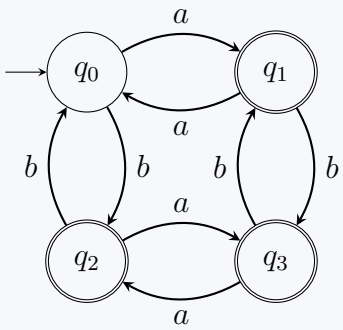


Algo importante a destacar aquí es que el único estado de aceptación es q_0 ya que en los demás siempre hay cantidad impar de *aes* o *bes*. Esto es relevante ya que es la esencia del siguiente ejercicio.

✎ 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.

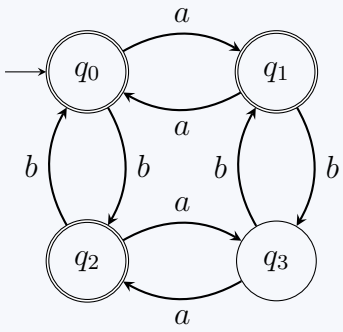
Caso 1: Impar de *aes* o impar de *bes*. Note que en el autómata original q_1, q_2 y q_3 no son de aceptación ya que en todos *aes* o *bes* son impares, luego en este caso estos tres se vuelven de aceptación mientras que q_0 ya no lo es.

Tikz:.



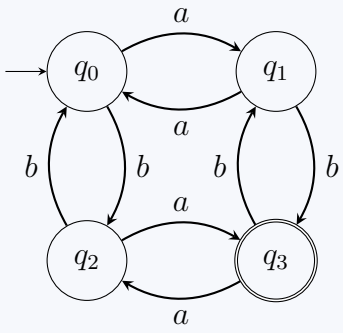
Caso 2: Par de aes o par de bes. Observe que el estado q_0 lo mantenemos como aceptación, luego note que la cadena abb es aceptada por la condición y si seguimos este camino en el autómata q_1 debe ser de aceptación. De la misma manera q_2 tiene que ser de aceptación ya que la cadena baa también es aceptada.

Tikz:.



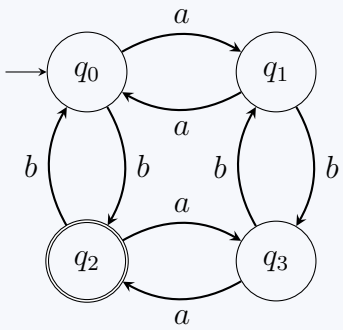
Caso 3: Impar de aes e impar de bes. Note que en la construcción anterior el estado q_3 no es de aceptación ya que es el único lugar donde aes y bes son impares, entonces para este caso ese va a ser nuestro único estado de aceptación.

Tikz:.



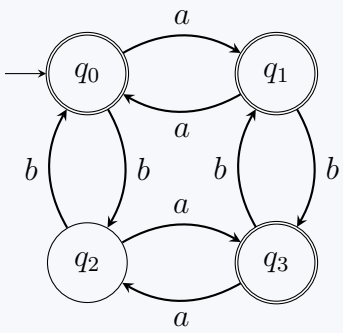
Caso 4: Par de aes e impar de bes. Observe que una cadena aceptada por estas condiciones es la cadena baa y esta es aceptada en q_2 luego este tiene que ser de aceptación. Ahora note que ninguno de los demás es de aceptación ya que q_0, q_1 aceptan cadenas con un numero par de bes y q_3 acepta cadenas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

Tikz:.



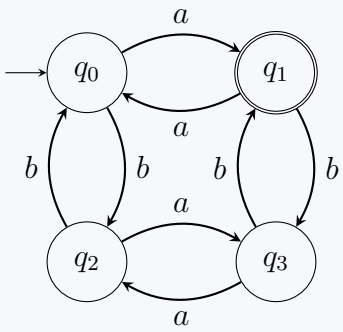
Caso 5: Impar de aes o par de bes. Por las bes pares q_0 es de aceptación. Por el impar de aes, q_1 y q_3 son de aceptación.

Tikz:.



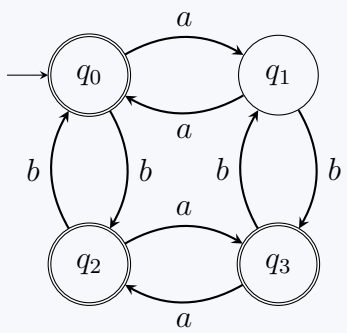
Caso 6: Impar de aes y par de bes. Una cadena que debe ser aceptada es por ejemplo abb, luego el estado que la acepta es q_1, ademas note que los demás estados no son de aceptación ya que aceptan las cadenas λ, b y ab que no cumplen las condiciones.

Tikz:.



Caso 7: Par de aes o Impar de bes.

Tikz:.

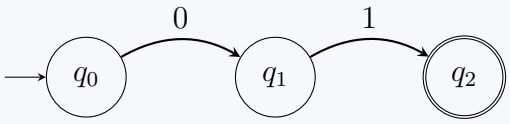


Observe que por ejemplo la condición del caso 7 es la negación de la condición del caso 6. Ademas los estados de aceptación del caso 6 son de no aceptación en el caso 7 y viceversa. Esto quedaría mas que nada como una mera curiosidad sino fuera por que posteriormente se mostrara que esta es una forma 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:

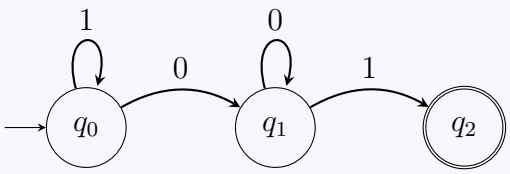
✎ Primero debemos forzar el camino para aceptar 01 que es la cadena mínima.

Tikz:.



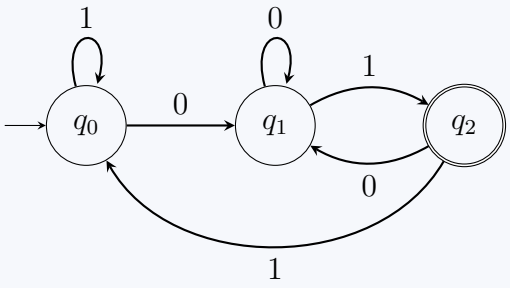
Ahora observe la cadena puede empezar por unos arbitrarios, estos lo generamos por medio de un bucle en el estado inicial. Luego lo mismo pasa en q_1 pero con ceros. Por ultimo al estar ya en el estado de aceptación note que si sale un 0 o un 1 la cadena dejaría de ser de aceptada y uno estaría tentado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

Tikz:.



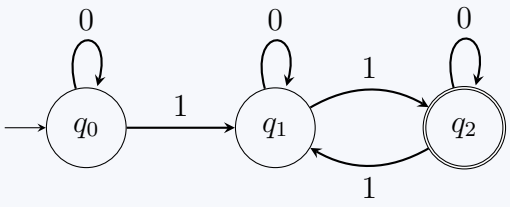
Pero note que las cadenas 0101 y 101101 son cadenas validas pero no son aceptadas por el autómata. Esto se arregla fácilmente con añadir los bucles correspondientes desde el ultimo estado.

Tikz:.



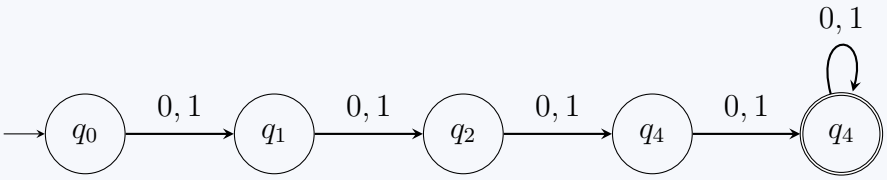
✎ Primero forzamos la mínima cadena par de unos que es 11, luego realizamos la construcción para obtener cantidad par de unos y añadimos bucles de 0 en todos los estados ya que estos no tienen restricción alguna.

Tikz:.



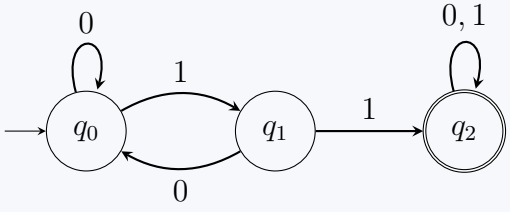
✎ Al igual que hacíamos con las expresiones regulares, hacemos primero que se acepten todas las cadenas de longitud 4 que son las mínimas y posteriormente por medio de un bucle se generaran todas las demás.

Tikz:.



✎ 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

Tikz:.

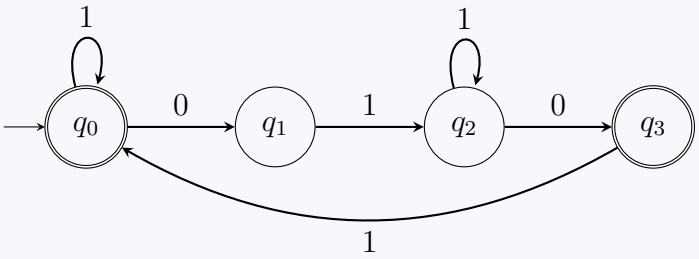


Note que agregamos una flecha de q_1 a q_0 con etiqueta 0 ya que esto permite aceptar

cadenas como 101011001 que sin este no serian aceptadas.

✎ 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).

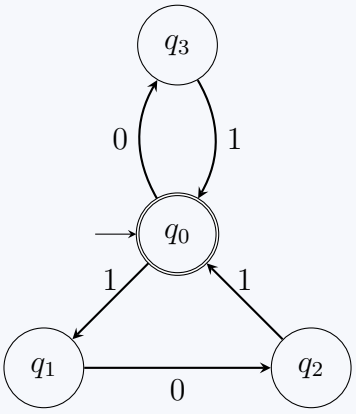
Tikz:.



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.

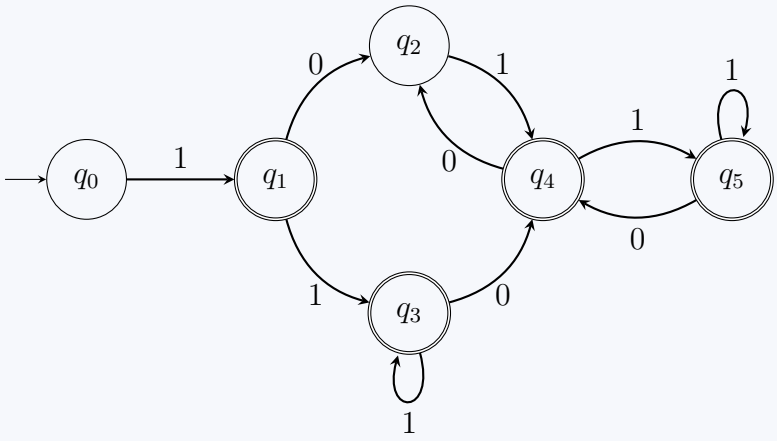
✎ Note que podemos formar los respectivos bucles por separado ya que la cadena empieza por 01 o por 101 y luego estas se concatenan en cualquier orden pero siempre son esos dos bloques.

Tikz:.



✎ 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.

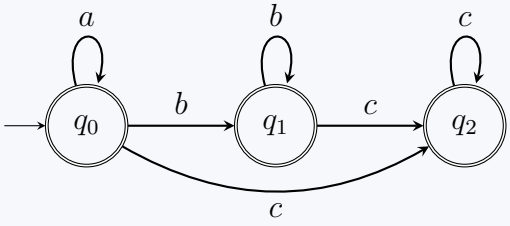
Tikz:.



Punto 4:

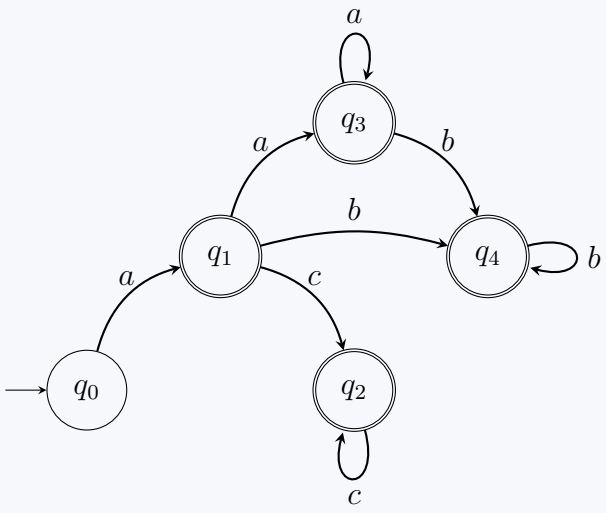
✎ Basta con forzar los bucles para cada elemento y el hecho de que todas las cadenas tienen la misma secuencia de elementos, *aes* arbitrarias luego *bes* arbitrarias y por ultimo *ces* arbitrarias.

Tikz:.



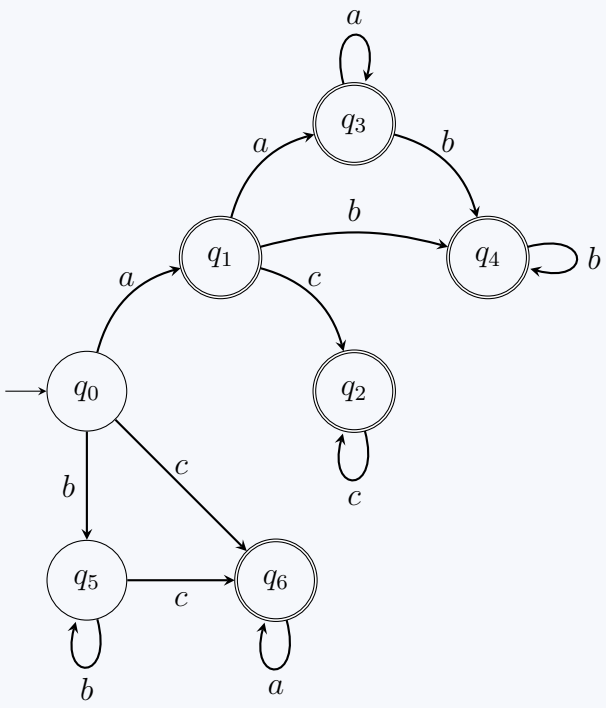
✎ Haremos una construcción por dos caminos distintos a partir de las cadenas mínimas. La primera es el camino que se forma con la cadena a.

Tikz:.



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 ac^*, segundo las cadenas de la forma ab^* y por ultimo las aa^+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 b^*ca^*.

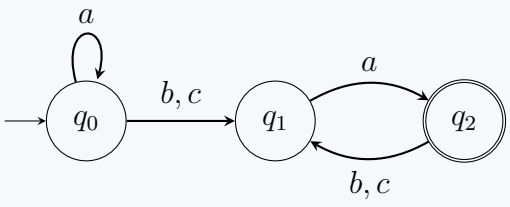
Tikz:.



Note como forzamos la cadena mínima c y usamos un estado extra para generar el bucle de bes 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.

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

Tikz:.



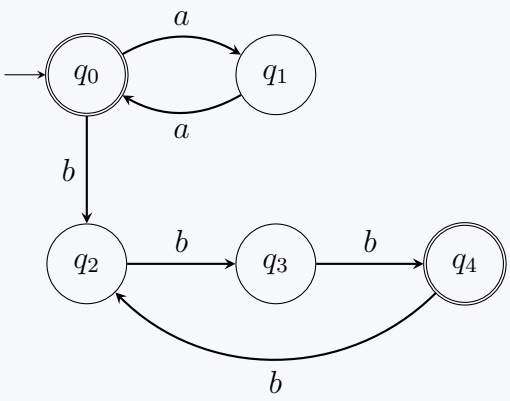
Punto 5: Primero es bastante evidente por la definición que se nos da que la expresión regular que determina a L es:

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

Ahora para la construcción del AFD M tal que $L(M) = L$, basta con primero realizar la construcción para obtener una cantidad par de aes y posteriormente agregamos un camino

para aceptar cantidad de bes 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).

Tikz:.



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.

♦

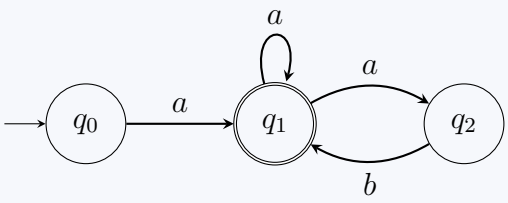
2.3. Autómatas finitos no-deterministas (AFN)

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:

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

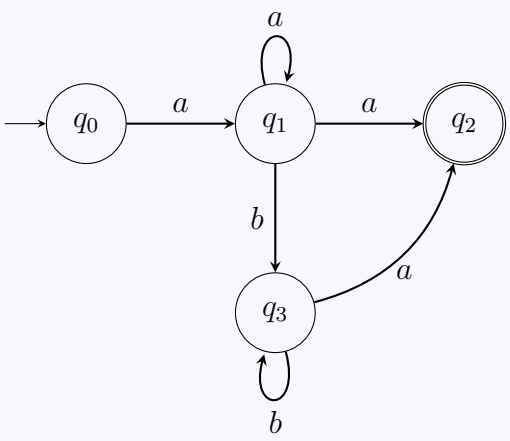
Tikz:.



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

✎ Para la construcción forzamos la cadena mínima aa junto a el bucle de aes inicial.

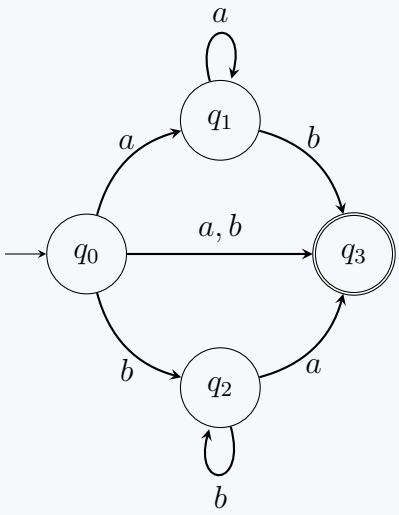
Tikz:.



Note que las *bes* intermedias las agregamos por medio de un estado extra generando un camino nuevo.

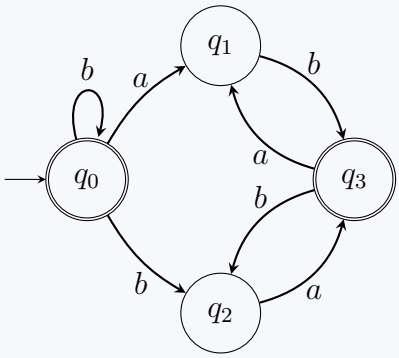
✎ Basta con construir un camino para las cadenas mínimas a y b, luego construimos dos caminos extra para aceptar a^*b y b^*a respectivamente.

Tikz:.



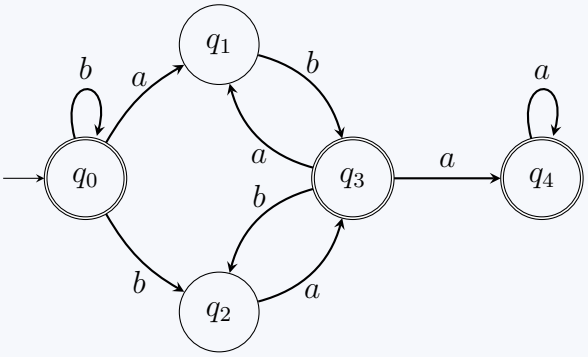
✎ Forzaremos las *bes* arbitrarias en el primer estado y luego generamos los caminos correspondientes para la concatenación de ab y ba.

Tikz:.



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

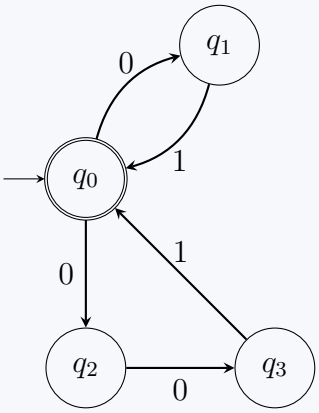
Tikz:.



Punto 2:

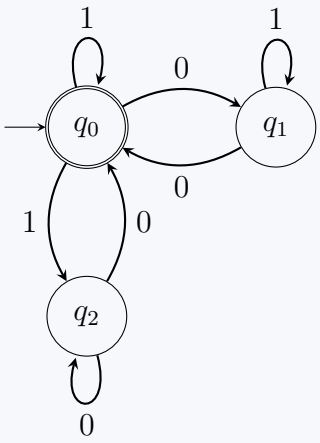
✎ Simplemente construimos los caminos para aceptar 01 y 001 por separado.

Tikz:.



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

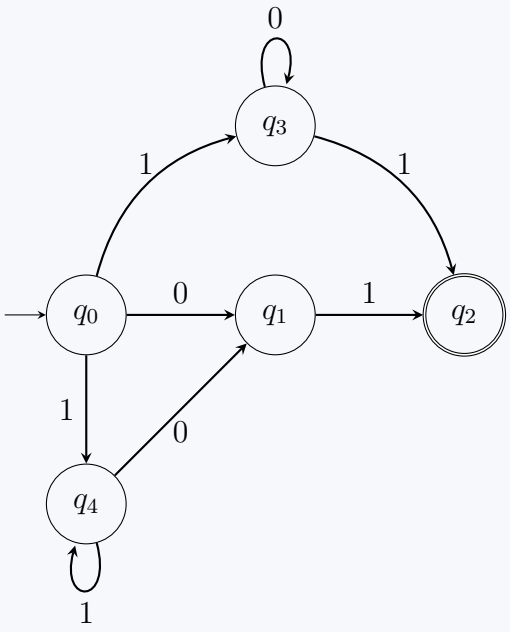
Tikz:.



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).

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

Tikz:.



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.

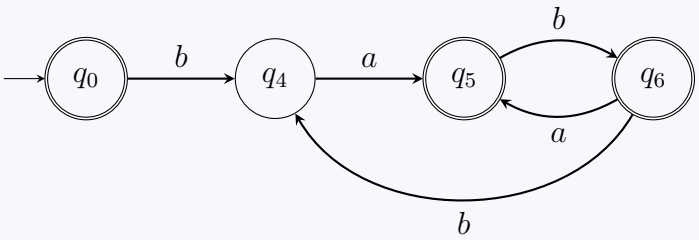
Punto 1:
A continuación en cada AFD que construyamos omitiremos los estados inútiles.

✎ 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 $q_6 := \{q_0, q_1, q_2\}$.

Tikz:.

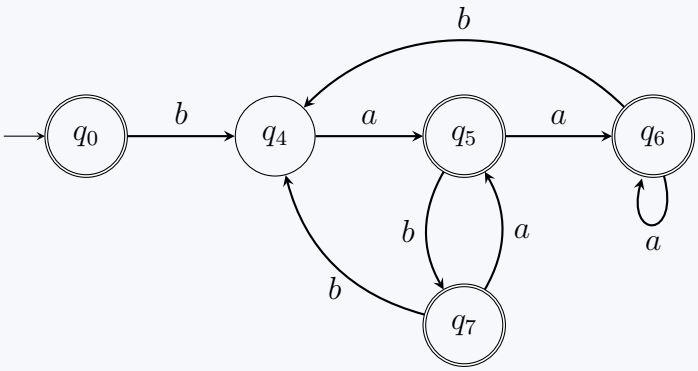


✎ Realicemos la extensión de la tabla.

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 $q_7 := \{q_0, q_1, q_2\}$.

Tikz:.

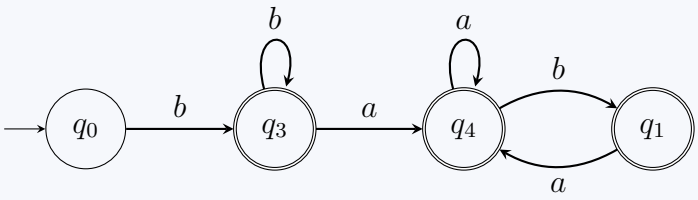


✎ Realicemos la extensión de la tabla.

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 $q_4 := \{q_1, q_2\}$.

Tikz:.

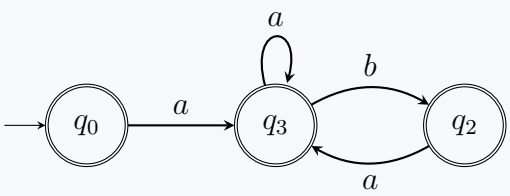


✎ Creo que ya es evidente que es lo primero que haremos.

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 $q_3 := \{q_0, q_1\}$.

Tikz:.



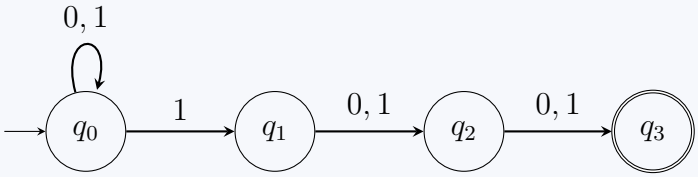
Punto 2:

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.

Tikz:.

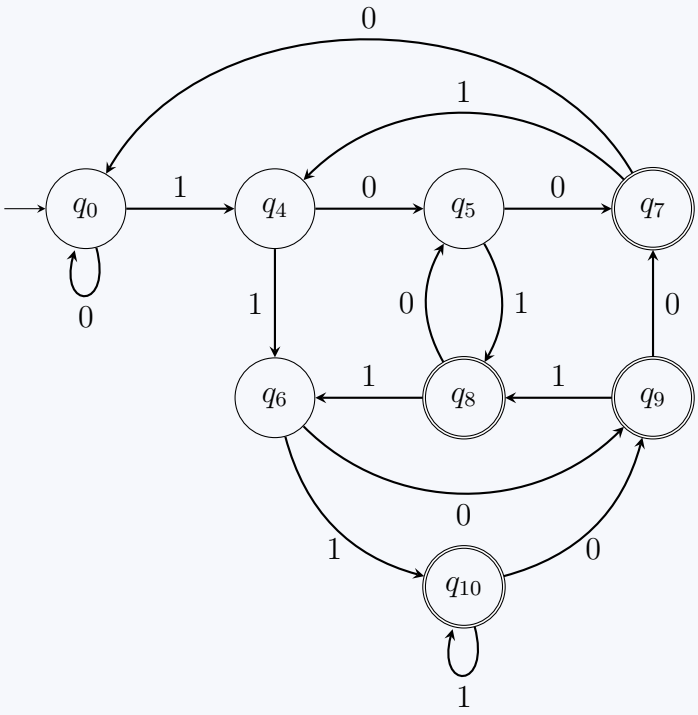


Ahora realicemos la extensión de la tabla.

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 $q_{10} := \{q_0, q_1, q_2, q_3\}$.

Tikz:.



Punto 3:

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:

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.

□

♦

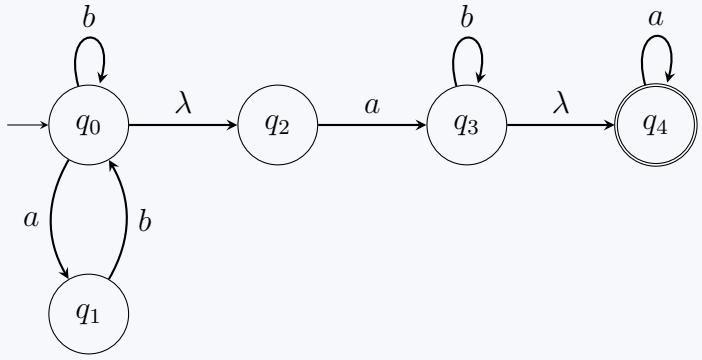
2.5. Autómatas con transiciones λ (AFN-λ)

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:

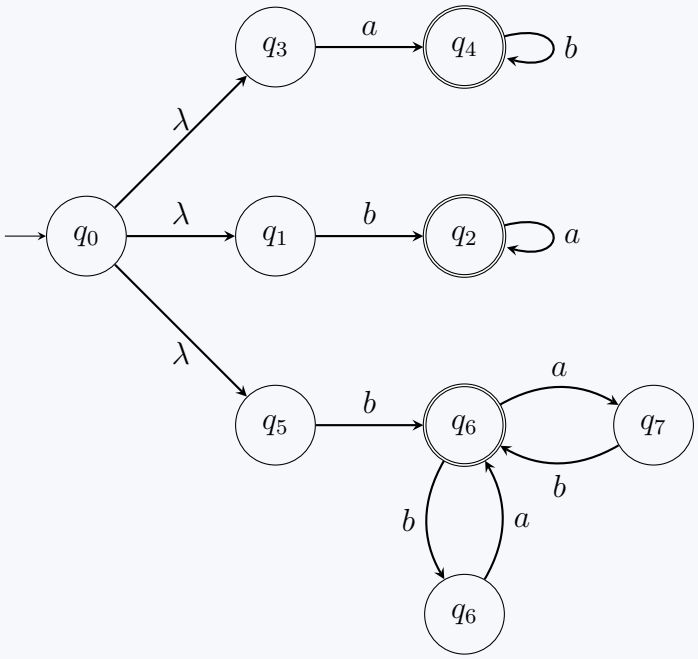
✎ Para este autómata realizaremos una construcción en serie, concatenando los autómatas respectivos de cada parte del lenguaje, note que a es la mínima cadena de aceptación.

Tikz:.



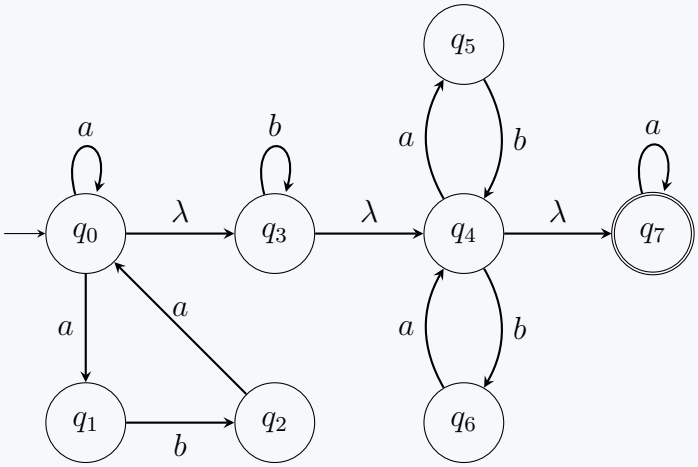
✎ Basta con realizar una construcción en paralelo para cada expresión de la unión. Es decir construimos un AFD para ab^*, ba^* y $b(ab \cup ba)^*$ respectivamente y luego realizamos la construcción en paralelo.

Tikz:.



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

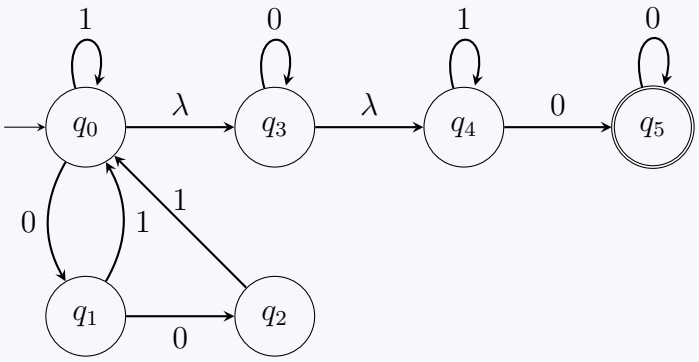
Tikz:.



Punto 2:

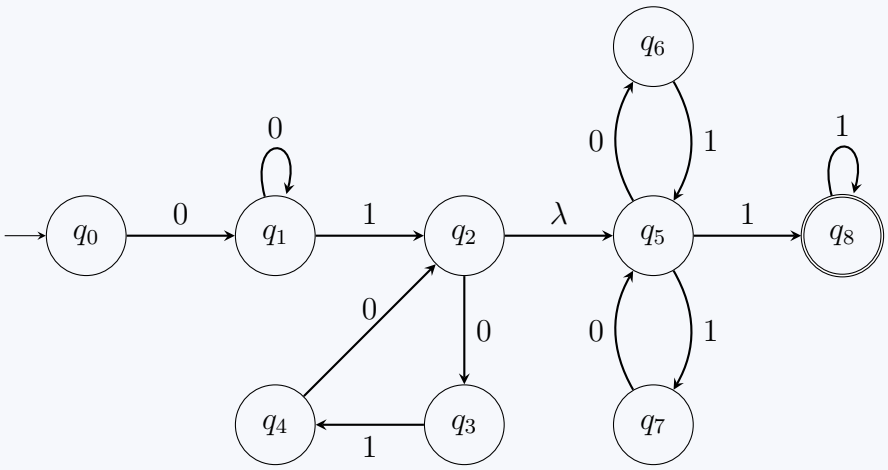
✎ El único autómata que puede resultar un poco complicado de construir es el de la expresión inicial $(1 \cup 01 \cup 001)^*$ pero basta con 3 estados, luego simplemente es realizar una construcción en serie.

Tikz:.



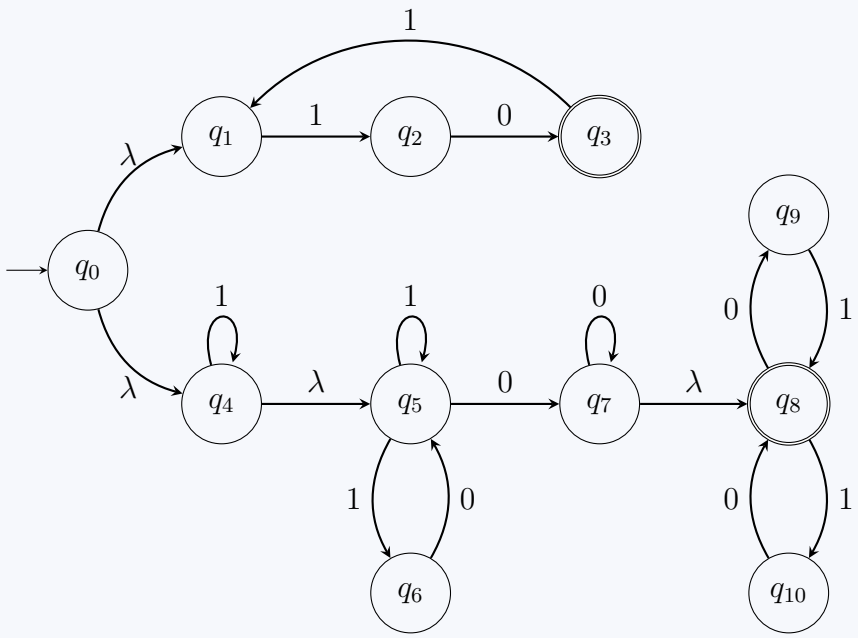
✎ Nuevamente una construcción en serie basta.

Tikz:.



✎ 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.

Tikz:.



♦

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

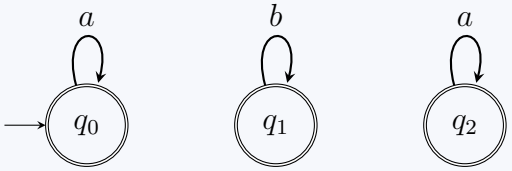
Previamente habíamos observado como si construíamos un AFN para aceptar un lenguaje regular, podíamos por medio de un algoritmo convertirlo en un AFD. En esta sección observaremos como podemos convertir un AFN-λ en un AFN.

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.

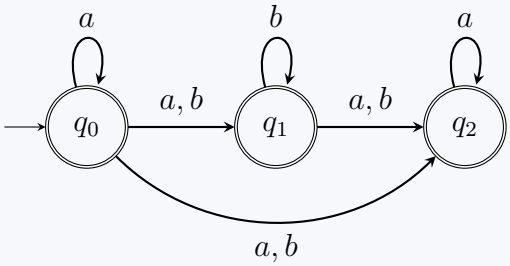
Tikz:.



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 $a\lambda, a\lambda\lambda, \lambda b, \lambda b\lambda$ respectivamente. El AFN obtenido es el siguiente:

Tikz:.

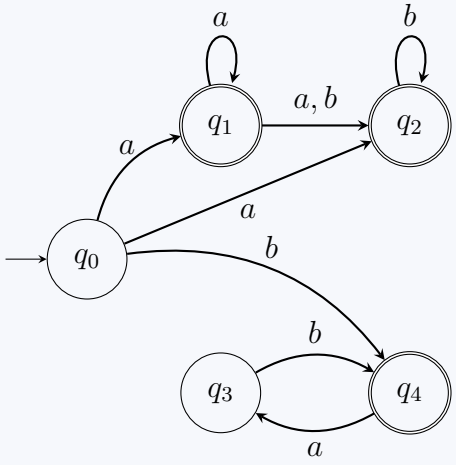


Punto 2: Primero hallemos la λ-clausura de cada estado:

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

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:

Tikz:.



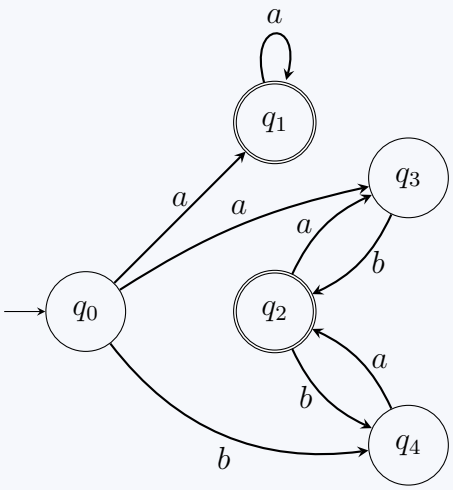
Si bien la presentación del autómata podría ser mas pulida recomendamos dejar los estados en las posiciones originales para hacer mas evidente y sencillo el proceso de inspección.

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:

Tikz:.

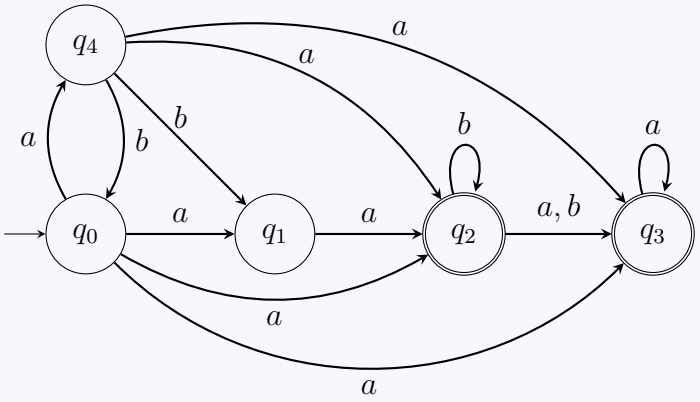


Punto 4: Primero hallemos la λ-clausura de cada estado:

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

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:

Tikz:.



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.

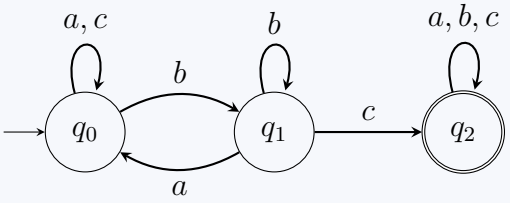
♦

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.

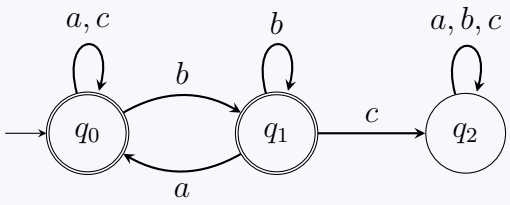
Punto 1: Construyamos primero un AFD tal que acepte todas las cadenas que contienen la subcadena bc. Para esto forzamos bc como aceptación y consideramos los bucles necesarios.

Tikz:.



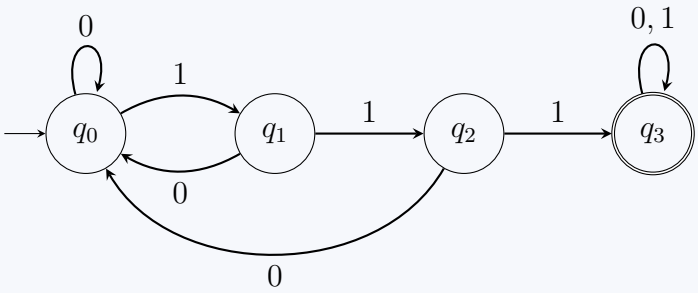
Ahora simplemente cambiamos los estados de aceptación, es decir q_0 y q_1 se vuelven de aceptación mientras que q_2 deja de serlo. De esta manera obtenemos el AFD que acepta el lenguaje deseado.

Tikz:.



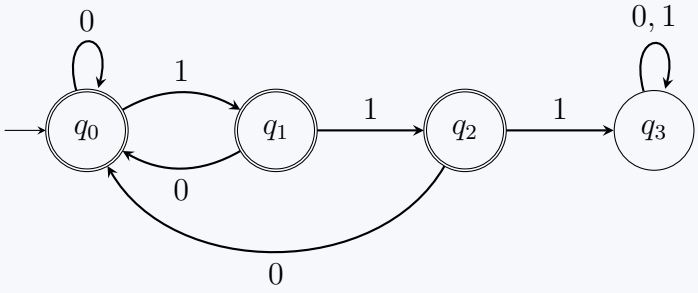
Punto 2: Primero construyamos el AFD que acepte todas las cadenas con tres unos consecutivos.

Tikz:.



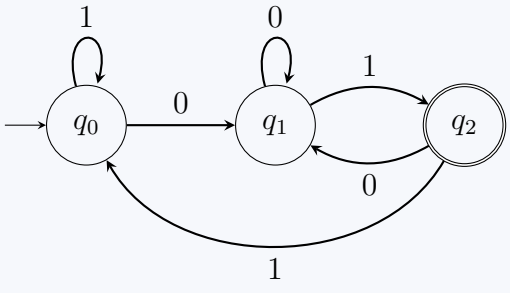
Luego el AFD que acepta el lenguaje deseado es:

Tikz:.



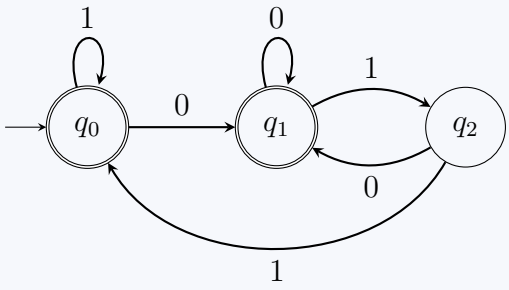
Punto 3: Primero construyamos el AFD que acepta todas las cadenas que terminan en 01.

Tikz:.



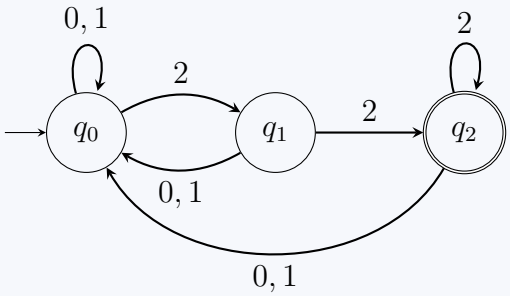
Luego el AFD que acepta el lenguaje deseado es:

Tikz:.



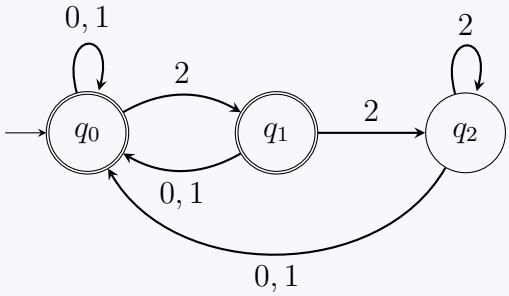
Punto 4: Primero construyamos el AFD que acepta todas las cadenas que terminan en 22.

Tikz:.



Luego el AFD que acepta el lenguaje deseado es:

Tikz:.



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.

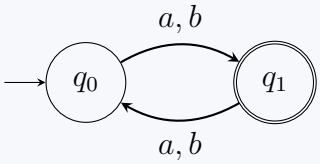
♦

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: Primero considere el AFD que acepta las cadenas de longitud impar:

Tikz:.



Ahora construyamos el que acepta las que contienen dos *bes* consecutivas:

Tikz:.