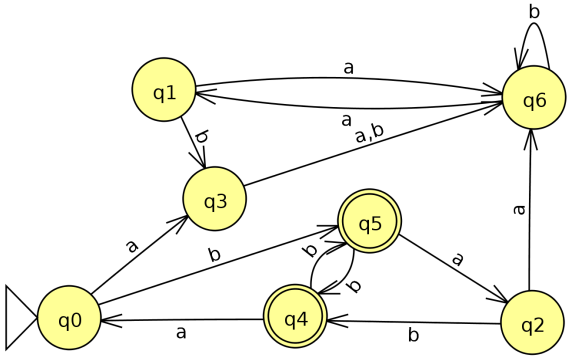
Universidad Técnica Federico Santa María
Campus Santiago San Joaquín
Segundo Semestre 2016

Tarea 2 - TALF

Jorge Contreras 201573547-6
Juan Pablo Jorquera 201573533-6

1. Pregunta 1

Primero realizamos el AFD de la tabla para poder trabajar sobre él.



1.1. Parte (a)

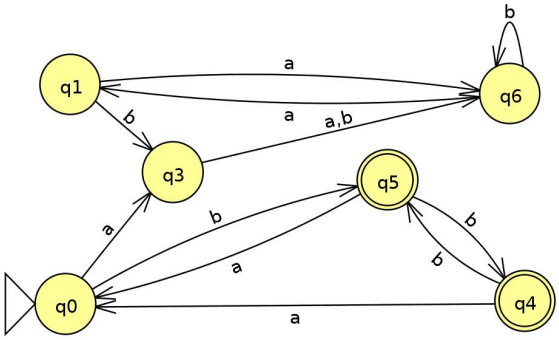
Primero hacemos la tabla correspondiente al autómata.

q1	X					
q2	O	X				
q3	X	O	X			
q4	X	X	X	X		
q5	X	X	X	X	O	
q6	X	O	X	O	X	X
	q0	q1	q2	q3	q4	q5

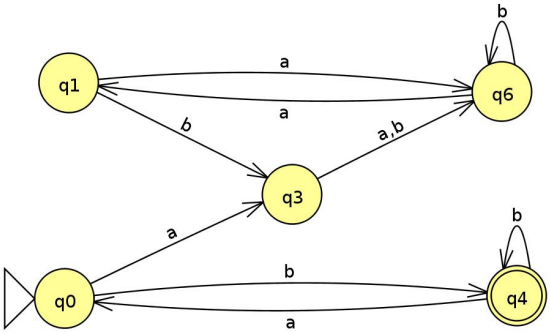
Tabla 1: Tabla de equivalencias

Luego continuamos eliminando los que son equivalentes:

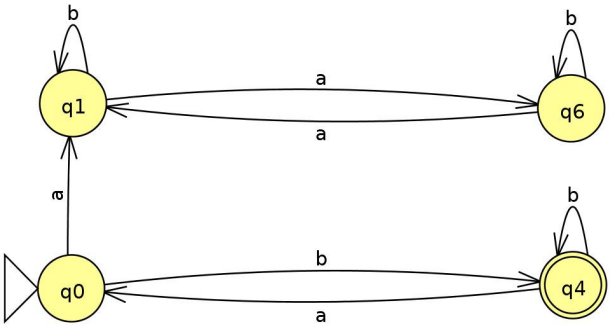
- Eliminar q2:



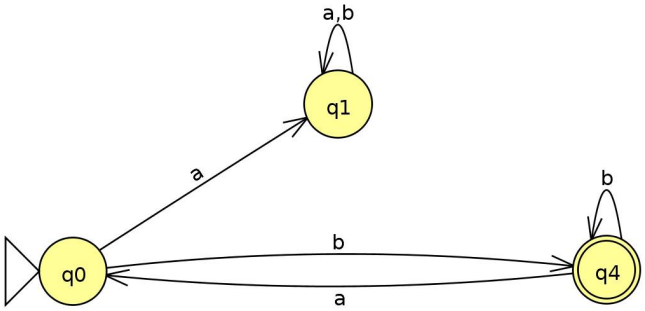
- Eliminar q5:



- Eliminar q3:



- Eliminar q6:

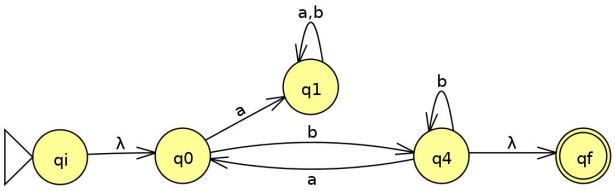


1.2. Parte (b)

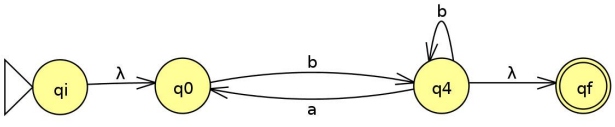
- Tomando q0 y q1: la palabra “b” distingue a ambos, ya que quedan en distinto estado de aceptación.
- Tomando q0 y q4: simplemente “ab” los distingue.
- Tomando q1 y q4: “b” los distingue.

1.3. Parte (c)

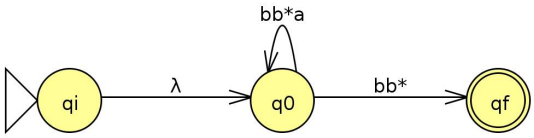
- Primero normalizamos para tener un nodo de entrada y de salida claros.



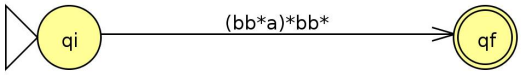
- Luego eliminamos el nodo “basura” que se encuentra.



- Eliminamos q4.



- Finalmente eliminamos q0.



1.4. Parte (d)

Corresponde al lenguaje que comienza con una o más "b" y, si lo sigue una "a", se repite el proceso con "b" nuevamente.

2. Pregunta 2

Si tomamos $L_B = LL_bL$ donde $L_b = \{b\}$, claramente L_B y L_b son regulares al ser parte L. Por otro lado tenemos que:

$$L' \Leftrightarrow L_B - L_b \tag{1}$$

Usando propiedades de conjuntos tenemos:

$$L' \Leftrightarrow L_B - L_b \Leftrightarrow L_B \cap L_b^c \tag{2}$$

Donde por propiedades de clausura se sabe que como L_b es regular, entonces L_b^c también es regular. De igual forma, se deduce que la intersección de ambos lenguajes regulares L_B y L_b^c es regular, por lo que L' también es regular.

3. Pregunta 3

3.1. Parte (a)

Sea $w = a^n b^{n-1}$, $w \in L_1$
$w = xyz$, con $|xy| \leqslant n$; $|y| \geqslant 1$.

Necesariamente x e y están dentro de a^n, Así:
$x = a^p$, $p \geqslant 0$
$y = a^q$, $q \geqslant 1$
$z = a^{n-p-q} b^{n-1}$

Aplicamos Teorema del Bombeo, k veces y.

$xy^k z = a^{n+q(k-1)} b^{n-1}$

Para que la palabra pertenezca a L_1 se debe cumplir que:

$|n + q(k-1) - (n-1)| \leqslant 42$
$|q(k-1) - 1| \leqslant 42$

Elegimos k=50, quedando
$|49q - 1| \leqslant 42$
Como $q \geqslant 1$, lo anterior es una contradicción y por lo tanto el lenguaje L_1 No es regular.

3.2. Parte (b)

Sea $w = a^n c a^n, w \in L_2$
$w = xyz$, con $|xy| \leqslant n; |y| \geqslant 1$

Necesariamente x e y están dentro de a^n, Así:
$x = a^p$, $p \geqslant 0$
$y = a^q$, $q \geqslant 1$
$z = a^{n-p-q} c a^n$

Aplicamos Teorema del Bombeo, k veces y.

$xy^k z = a^{n+q(k-1)} c a^n$

Para que la palabra pertenezca a L_2 se debe cumplir que:

$n + q(k-1) = n$
$q(k-1) = 0$

Elegimos k=2, quedando

q = 0
Pero $q \geqslant 1$, por lo cual la palabra no pertenece al lenguaje y así demostramos que L_2 No es regular.

3.3. Parte (c)

Como pudimos generar un autómata para el lenguaje, éste es regular.

3.4. Parte (d)

Sea $w = a^{2n}b^{2n-1}$, $w \in L_4$
$w = xyz$, con $|xy| \leqslant 2n$; $|y| \geqslant 1$.

Necesariamente x e y están dentro de a^{2n}, Así:
$x = a^p$, $p \geqslant 0$
$y = a^q$, $q \geqslant 1$
$z = a^{2n-p-q}b^{2n-1}$

Aplicamos Teorema del Bombeo, k veces y.

$xy^kz = a^{2n+q(k-1)}b^{2n-1}$

Para que la palabra pertenezca a L_4 se debe cumplir:

(1)
$2n + q(k-1) > 2n - 1$
$q(k-1) > -1$

Elegimos k=0, quedando
$-q > -1$
$q < 1$
Pero $q \geqslant 1$, existiendo así una contradicción.

(2)

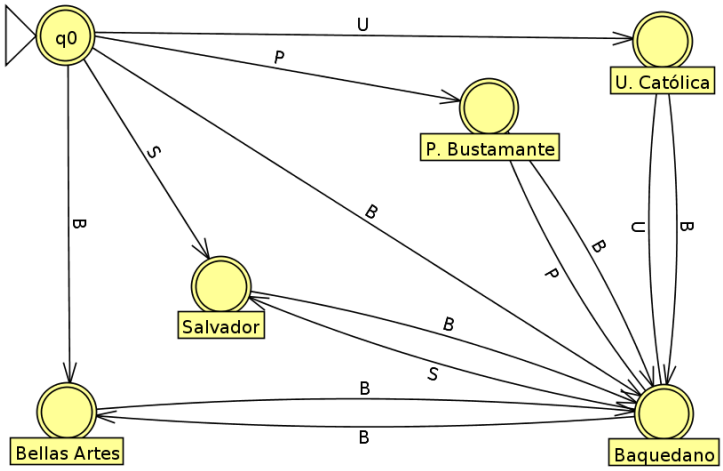
$2n + q(k-1)$ es par

Elegimos $k = \frac{1}{q} + 1$
$2n + q(\frac{1}{q} + 1 - 1)$
$2n + 1$
Lo cual siempre es impar.

Como no se cumple ni (1) ni (2), podemos asegurar que el lenguaje L_4 No es regular

3.5. Parte (e)

El metro de Santiago posee 5 líneas y 100 estaciones en total, pudiendo llegar desde cualquier estación de cualquier línea a otra estación de cualquier línea igualmente, es decir, es un grafo conexo. Como la cantidad de estaciones es finita, se puede crear un automáta finito en el cual las estaciones hacen de estados, y las transiciones de los estados corresponden a las estaciones que se encuentran inmediatante conectadas a otra estación, por ejemplo Camino Agrícola posee dos transiciones, a San Joaquín y a Carlos Valdovinos, mientras que Los Heroes posee 4 transiciones, a Toesca, Santa Ana, Moneda y República. Cada transición entre estados está ligada a la primera letra del nombre del estado al cual se llega, habiendo un estado inicial qo(que no corresponde a ninguna estación de metro) que llega a todos los demás estados(estaciones) usando las transiciones explicadas anteriormente, y todos los estados del autómata finito son estados de aceptación. A continuación se presenta un modelo reducido del automáta que abarca sólo a la estación Baquedano y sus respectivas transiciones(considerando que el automáta total es demasiado denso).

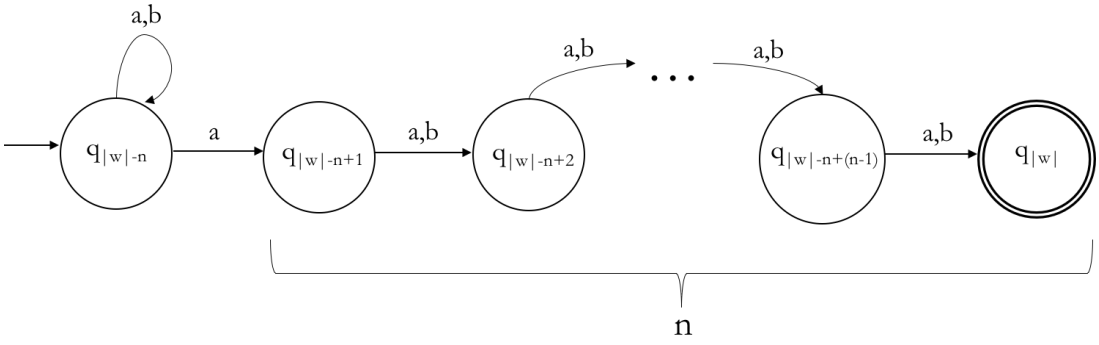


Como es posible la creación de un autómata finito, el lenguaje L_5 es regular.

4. Pregunta 4

4.1. Parte (a)

Como se puede ver el AFND a continuación, antes del $n-$ésimo símbolo (contado desde atrás) no importa lo que haya, por lo que se produce un loop, luego continua con dicho símbolo seguido por $n-$estados, por lo que en éste autómata hay $n+1$ estados.



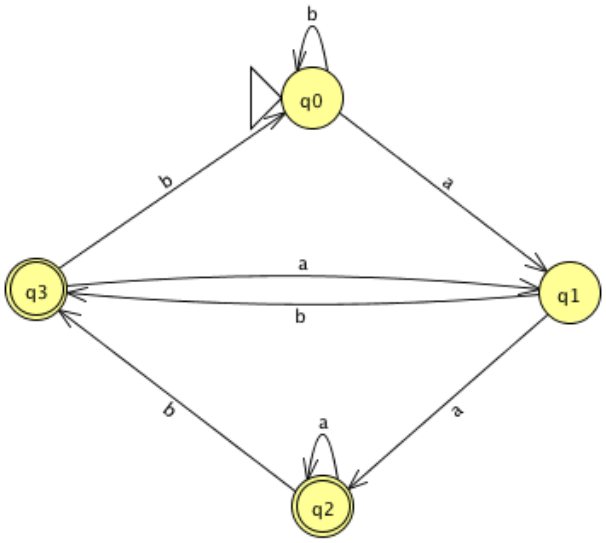
4.2. Parte (b)

En primer lugar damos más importancia para cada estado después de la "a" requerida (inicialmente siempre hay un loop de "b", ya que no importa lo que esté concatenado antes), encontes verificamos lo que podemos concatenar después según Myhill-Nerode, ya que dependiendo del n puede variar el resultado, por ejemplo si tenemos (a partir del n-ésimo caracter):

- "aa": Al concatenar al final de la palabra "a" se distingue convirtiéndose el segundo "a" en el nuevo n-ésimo y quedando finalmente en aceptación.
- "ab": Por otro lado si concatenamos "a" en este caso el "b" pasaría a ser el n-ésimo quedando fuera de la aceptación.

Entonces nos damos cuenta que para cada caracter después del n-ésimo caracter se crean clases de equivalencia distintas diferenciándose en cual es la última posición de "a" y pudiéndo volver a éstas según corresponda, por lo que es necesario crear dos estados nuevos por cada caracter de n, por lo que finalmente la cantidad de estados será por lo menos 2^n. Cabe resaltar que la mitad de 2^n serán de aceptación al estar en el último nivel y que en uno de ellos se encontrará un loop de "a", ya que si se sigue un camino de sólo "a" da lo mismo las que se agreguen, el n-ésimo seguirá siendo "a".

A continuación se muestra un ejemplo para $n = 2$, para ayudar la visualización.



5. Pregunta 5

5.1. Parte (a)

Es decidible, ya que dado un lenguaje regular, con su respectivo AFD (o AFND), siempre se puede identificar si alguna de las palabras que aceptan es. Una forma de realizar esto sería:

1. Ver caminos posibles desde el inicio hasta cualquier estado de aceptación.
2. Calcular distancia de dichos caminos ignorando los loops que se encuentren, si ésta es de tamaño par, inmediatamente se cumple. Por otro lado si es de tamaño impar, hay que verificar loops que se encuentren:
   - Si hay un loop de tamaño par, simplemente ignorar, ya que no afecta la paridad de la palabra.
   - Si, por otro lado, hay loop de tamaño impar (incluyen los que se hacen sobre el mismo estado), se cumple la condición, ya que se puede recorrer cualquier número de veces el loop hasta formar la paridad requerida.
   - Si simplemente no hay loops o si todos ellos eran de tamaño par, significa que el lenguaje no cumple con incluir palabras de largo par.

5.2. Parte (b)

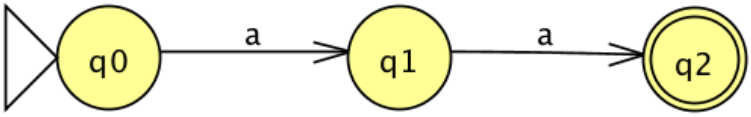
En este caso, el problema también es decidible, una forma de hacer eso es tomando el algoritmo usado en la Parte (a) y modificarlo en el punto 1: La única diferencia es que en vez de realizar el proceso para cualquier camino entre el inicio y estado de aceptación, se debe hacer para todo camino posible hacia todo estado de aceptación, para así verificar la paridad para la totalidad del lenguaje.

5.3. Parte (c)

También resulta un problema decidible, para ello tomamos el AFD mínimo del lenguaje que se requiera identificar y sobre él verificamos si ya es par, en caso de no ser:

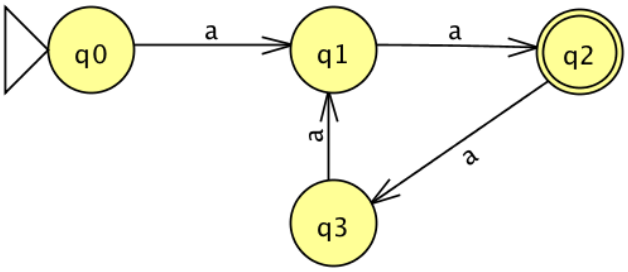
- Tomar un estado que tenga más de un camino entrante. Si hay, se puede agregar un estado adicional:
  1. Dirigir alguno de los caminos entrantes hacia un nuevo estado, que cumpla la misma aceptación que el objetivo anterior.
  2. Copiar caminos que partan del estado anterior, esta ver partiendo desde el estado nuevo.
  3. Finalmente con dicho estado, el AFD tiene una cantidad par de estados.
- En caso de no haber, no se pueden agregar más estados y no hay ADF con cantidad par de estados.

Un ejemplo de un caso en que no haya AFD con cantidad par de estados se muestra a continuación com un alfabeto = {a}:

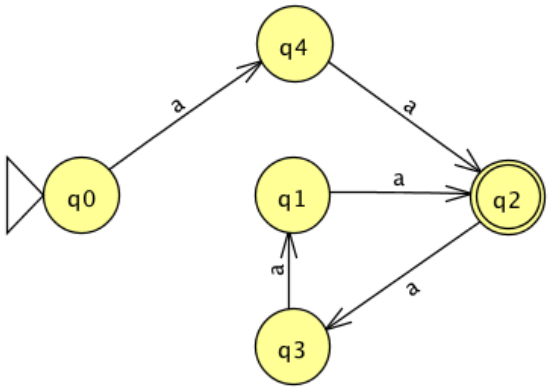


Se ve que no se pueden agregar estados adicionales, ya que para agregar alguno sobre los caminos que llegan a q1 o q2, habría simplemente que reemplazar el estado elegido, dejándolo inalcanzable y sacándolo así del AFD.

El caso opuesto lo vemos con el siguiente AFD que cumple la condición:



Donde obligatoriamente de elige q1 y se puede agregar un estado sobre cualquiera de los dos caminos, un ejemplo de ello sería:



Donde se utilizan los pasos mencionados y se deja el AFD inalterado para el lenguaje que lo representa, esta vez con una cantidad par de estados