



Práctica #05: Autómatas Finitos en JFLAP

CyA: 2024-2025

16/10/2024 - 9:30

Pablo García De Los Reyes

C/ Padre Herrera s/n
38207 La Laguna
Santa Cruz de Tenerife. España

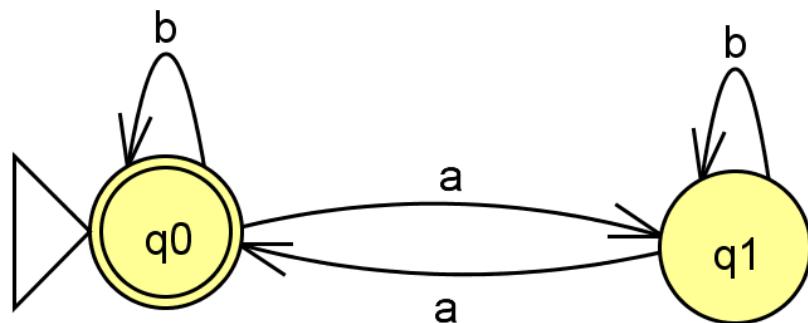
T: 900 43 25 26

ull.es



Ejercicios prácticos DFA:

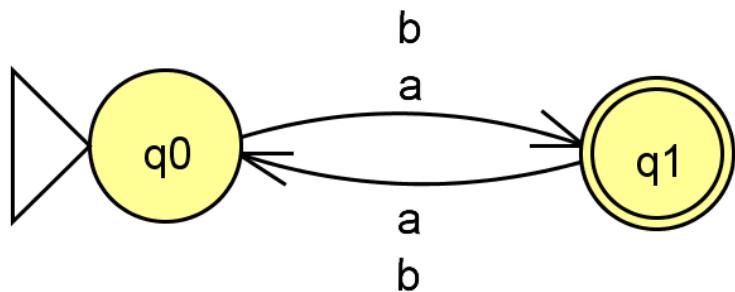
1. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ con número de "a's" par.



Input	Result
abba	Accept
aaaa	Accept
bb	Accept
abababa	Accept
aa	Accept
aaa	Reject
bab	Reject
ababa	Reject
aaabbb	Reject
ab	Reject

Para este DFA, usamos el estado inicial como el de aceptación pues la cadena vacía es aceptada y nos apoyamos en otro estado (q_1) para la comprobación de la entrada de “a” pares.

2. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ con longitud impar.

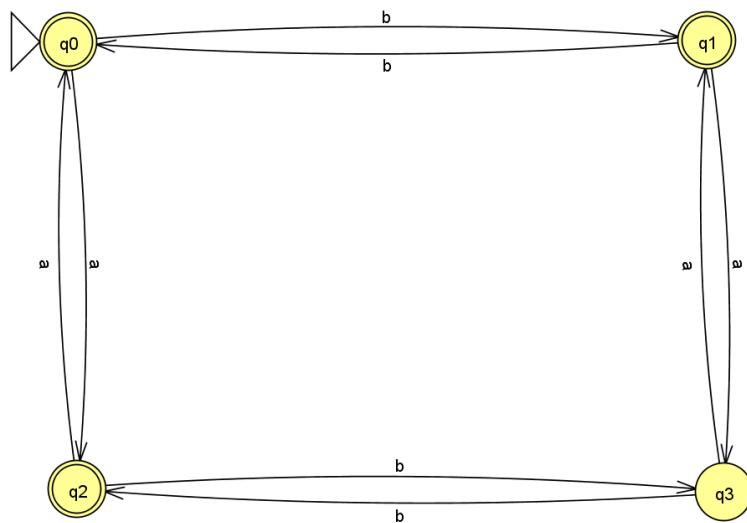


Input	Result
a	Accept
aba	Accept
abab	Accept
aaa	Accept
bbbbaaaa	Accept
ab	Reject
abba	Reject
bbaabb	Reject
abab	Reject
aaaa	Reject

Al contrario que el DFA anterior, en este el estado inicial no va a ser de aceptación pues la cadena vacía no es aceptada al ser de longitud 0, sin embargo empleamos el mismo método que en el anterior ejercicio, apoyándonos en otro

estado (q_1) para comprobar la imparidad de la longitud de la cadena introducida.

3. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ con número de “a’s” par o longitud impar.

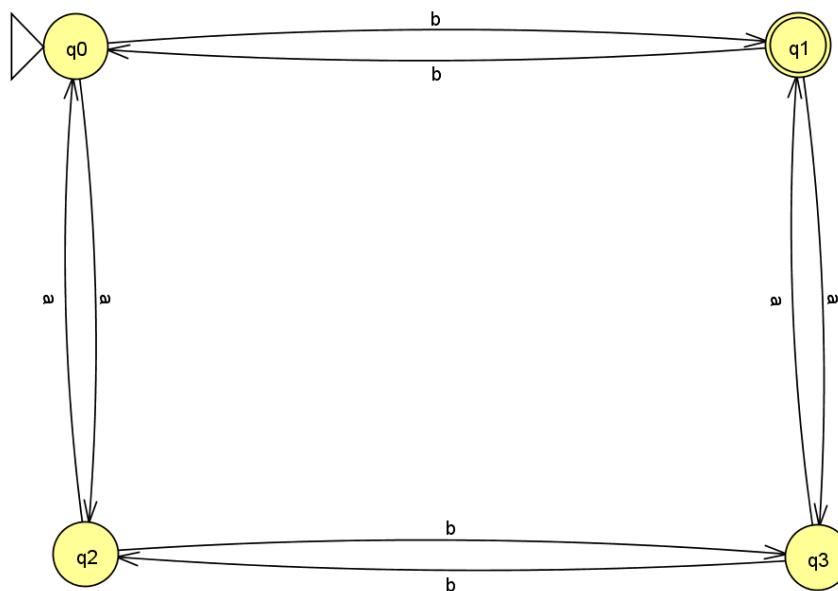




Input	Result
abab	Accept
bab	Accept
aaa	Accept
abbababa	Accept
a	Accept
abbb	Reject
bbbbab	Reject
aaab	Reject
baabab	Reject
aaaaab	Reject

En este caso para sacar el DFA he realizado la construcción de Thompson de disyunción con los dos DFA anteriores contemplándose como NFA's y tras hacer el algoritmo de subconjuntos y el de minimización de DFA me ha quedado de esta manera.

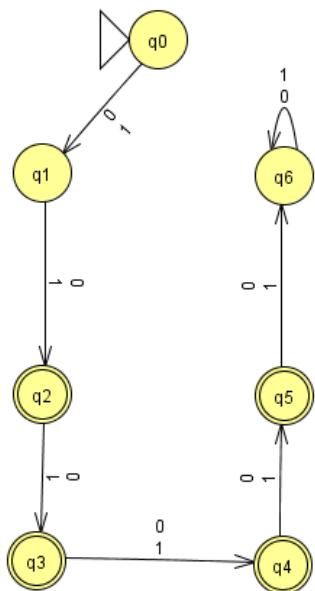
4. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ con número de “a’s” par y longitud impar.



Input	Result
aab	Accept
aba	Accept
aaaab	Accept
abababa	Accept
b	Accept
ababab	Reject
aaa	Reject
abbbbbbab	Reject
aaababaaaab	Reject
aabb	Reject

En este caso he aplicado lo visto en el anterior DFA, fijándome muy bien en el anterior, pues la similitud se ve a simple vista. El único cambio respecto al 3 ha sido quitar los estados de aceptación q_0 y q_2 , pues estos servían para aceptar aquellas cadenas impares que no tuviesen número de a par, o cadenas con a par que no fueran impares.

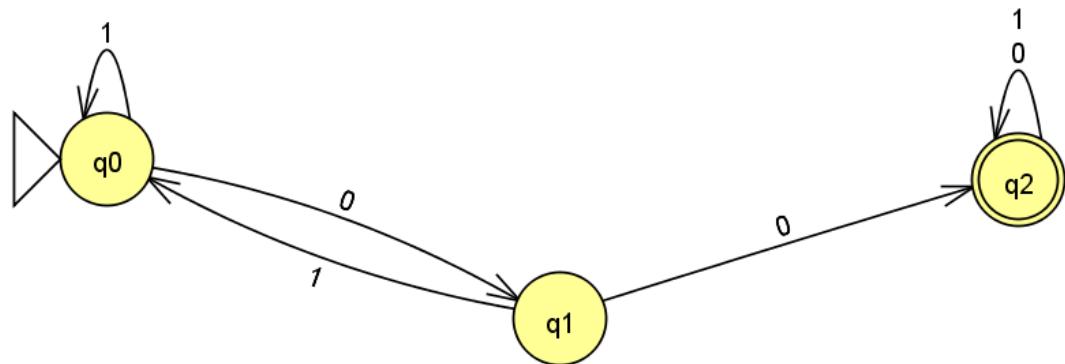
5. Diseñar un autómata finito determinista que reconozca cadenas w sobre el alfabeto $\{0, 1\}$ tales que $2 \leq |w| \leq 5$.



Input	Result
01	Accept
011	Accept
0011	Accept
00001	Accept
11001	Accept
1	Reject
00000011	Reject
1110101011010	Reject
01101010101011010	Reject
00011100011	Reject

Dado que solo reconoce 4 distintas longitudes de cadena entre 2 y 5, se han utilizado 6 estados, 4 de ellos de aceptación y los otros dos para las cadenas con longitud menor a 2 y mayor a 5.

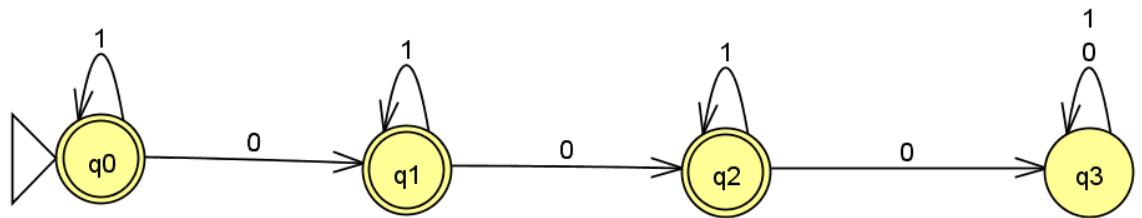
6. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\{0, 1\}$ que tengan como mínimo dos ceros consecutivos.



Input	Result
1001	Accept
101001	Accept
00	Accept
100000001	Accept
111001	Accept
0101	Reject
11	Reject
0	Reject
1010101011	Reject
1	Reject

Se usan tres estados para verificar al menos dos transiciones consecutivas de 0s, en este caso desde que se introducen dos ceros seguidos, se queda en un estado de aceptación en el cual se acepta el resto de la cadena sea cual sea.

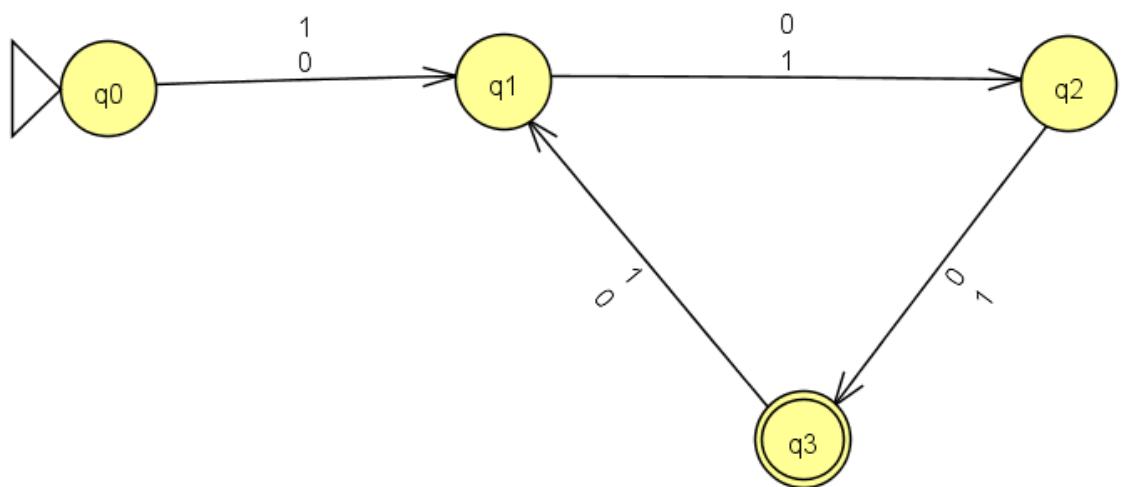
7. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\{0, 1\}$ que tengan como máximo dos ceros.



Input	Result
00	Accept
0101	Accept
1101	Accept
11111111	Accept
0111110	Accept
000	Reject
100001	Reject
10101010	Reject
0111011110	Reject
0110110	Reject

Al contrario que el anterior, ahora usaremos un estado de muerte por lo que ya no podremos contar con solo 3 estados. Al transicionar más de dos ceros, sin importar cual sea el resto de la cadena, transiciona a un estado de muerte que nunca será aceptado.

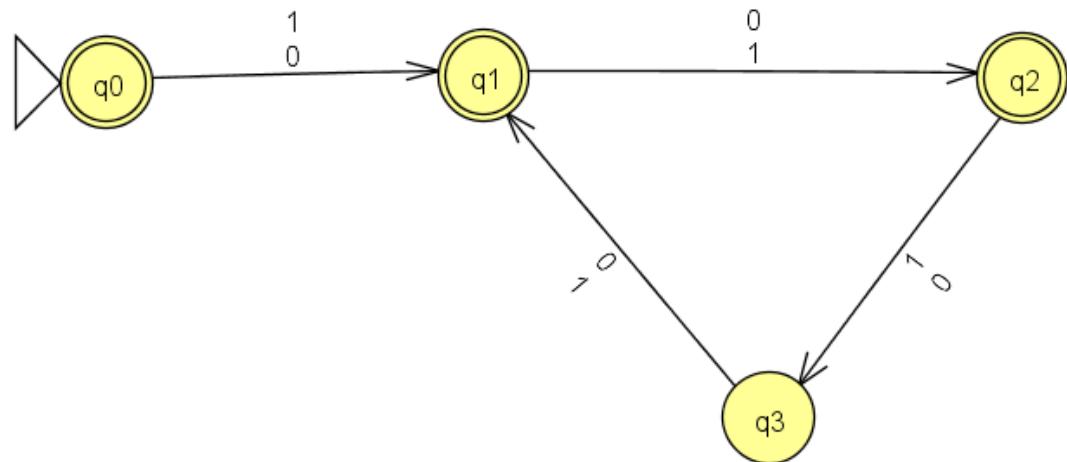
8. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\{0, 1\}$ con longitud múltiplo de 3.



Input	Result
010	Accept
000111	Accept
010101001	Accept
111	Accept
000111000111000	Accept
10	Reject
0	Reject
1	Reject
1111	Reject
0101100	Reject

Solo consigue aceptar una cadena si ha transicionado un mínimo de 3 veces, después de eso vuelve a repetir el ciclo hasta aceptar la cadena si la longitud de esta es múltiplo de 3.

9. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\{0, 1\}$ con una longitud que no sea múltiplo de 3.

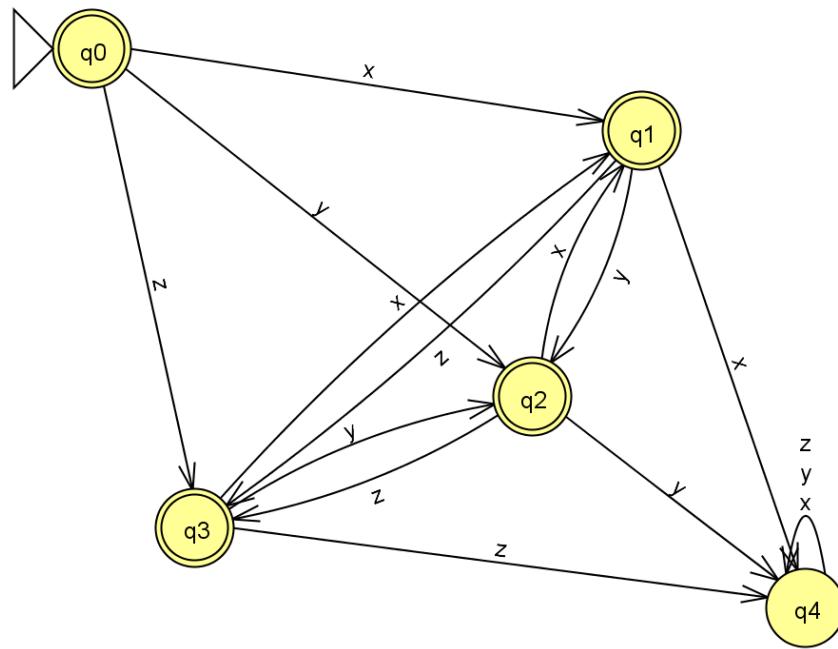


Input	Result
010	Reject
000111	Reject
010101001	Reject
111	Reject
000111000111000	Reject
10	Accept
0	Accept
1	Accept
1111	Accept
0101100	Accept

Este ejercicio consiste en el complemento del anterior, pues contiene los mismos estados y transiciones, sin

embargo los estados que no eran de aceptación ahora pasan a serlo y viceversa.

10. Diseñar un autómata finito determinista que reconozca cadenas sobre el alfabeto $\{x, y, z\}$ que no contengan dos símbolos iguales consecutivos.





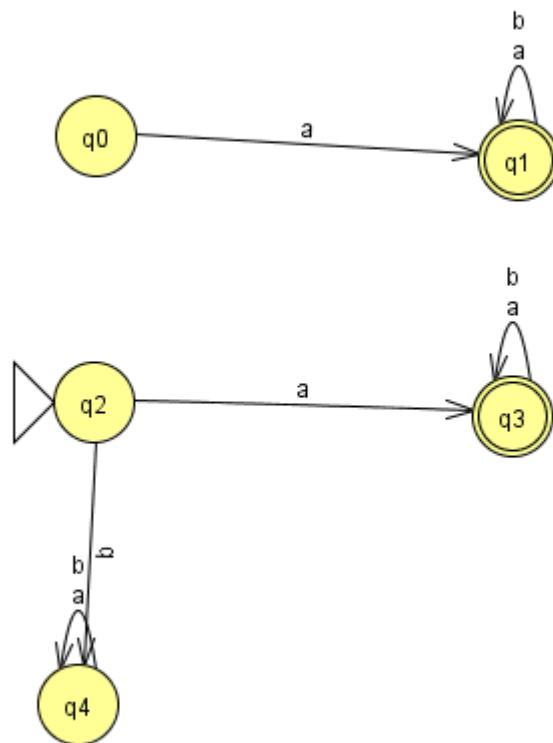
Input	Result
Xyz	Accept
XyzYzx	Accept
X	Accept
yz	Accept
zyx	Accept
xxyz	Reject
zzyxz	Reject
xxzzyy	Reject
yzxxy	Reject
zz	Reject

Este DFA tiene un diseño estructural en el cual consideramos a los estados (q_1, q_2, q_3) como los estados base para los símbolos (x,y,z) respectivamente, pues si en uno de estos se quiere transicionar con el símbolo que anteriormente se ha consumido para transicionar a este mismo, entonces pasará a un estado de muerte.



Ejercicios prácticos NFA:

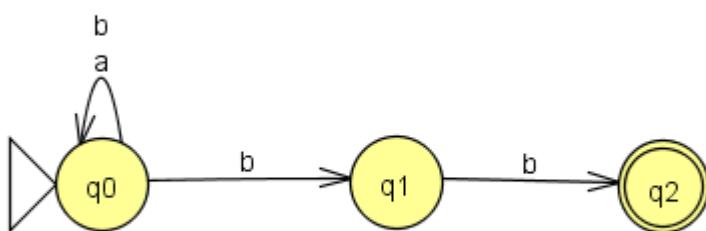
1. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que empiecen por “a”. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.

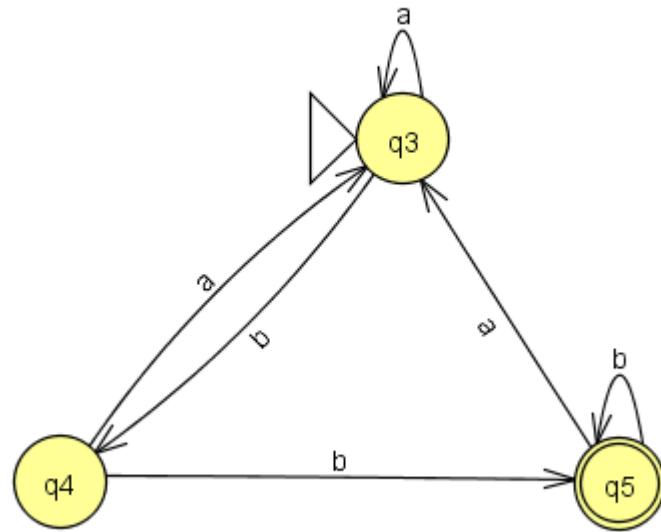


Input	Result
abaa	Accept
a	Accept
abbababaaba	Accept
aaaaaa	Accept
abba	Accept
baaab	Reject
bbbab	Reject
b	Reject
bababab	Reject
bbbbbb	Reject

Para este NFA he optado por no hacer el algoritmo de subconjuntos y minimización, pues la poca complejidad del ejercicio no lo requería para sacar el DFA equivalente. Para la aceptación de las cadenas empezadas por a, simplemente tenemos que tener en cuenta que el estado inicial no puede ser de aceptación y que debemos llegar al estado de aceptación mediante una transición con el símbolo “a”.

2. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que terminen en “bb”. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.

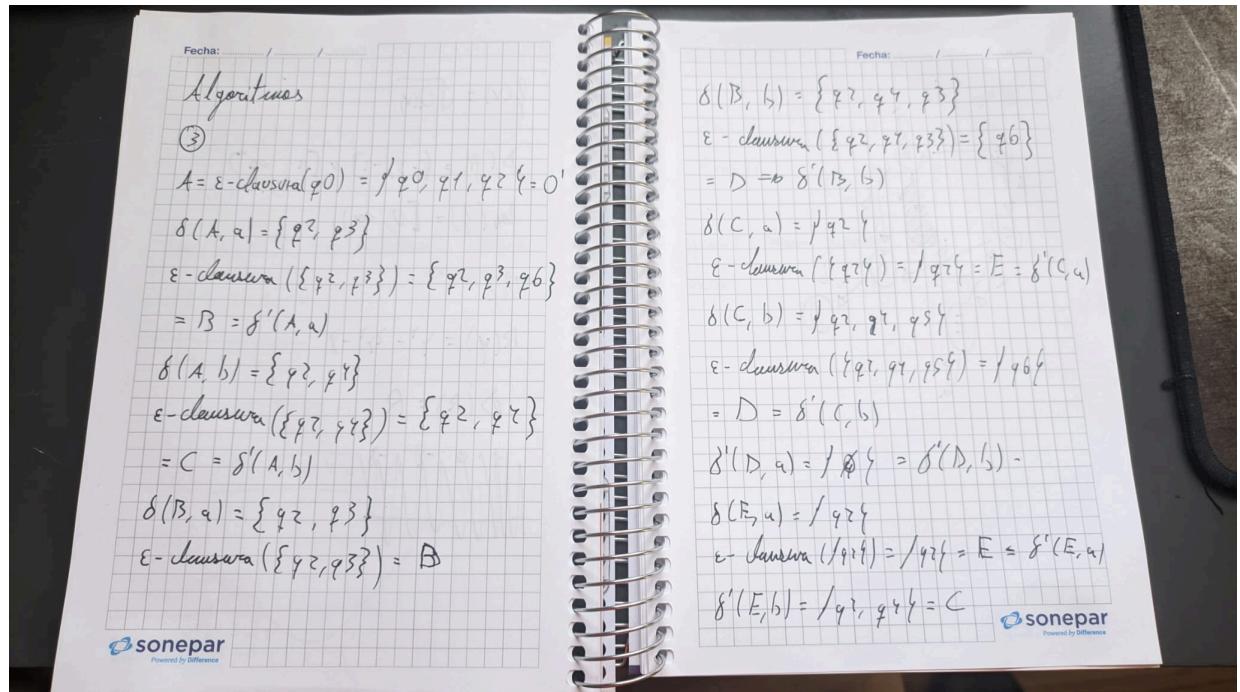
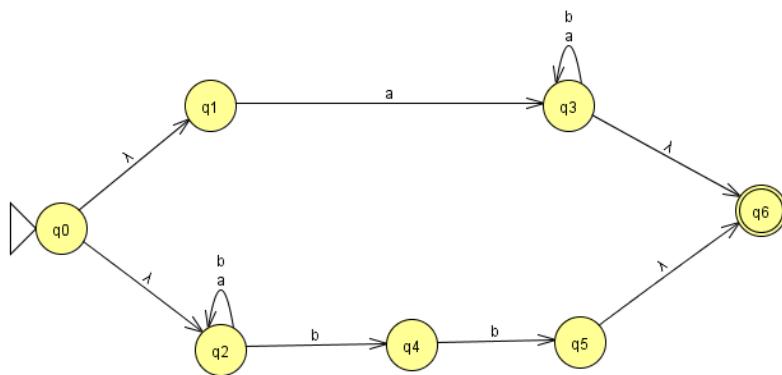


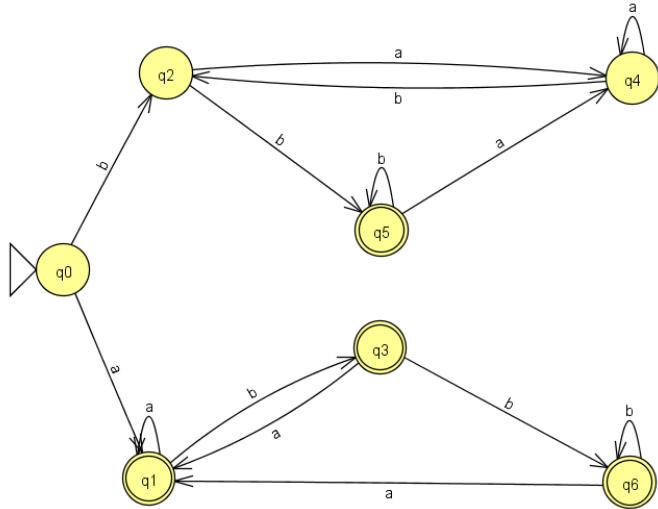


Input	Result
abb	Accept
babababb	Accept
bb	Accept
aabb	Accept
aaabbb	Accept
baba	Reject
abab	Reject
bbaab	Reject
b	Reject
aa	Reject

Para este caso he hecho lo mismo, pues al no tener muchos estados el NFA he podido sacar un DFA equivalente y mínimo sin problema. Únicamente teniendo en cuenta que el estado de aceptación se dará si los dos últimos símbolos son b

3. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que empiecen por "a" o terminen en "bb". A partir del NFA diseñado, obtenga un DFA mínimo equivalente.

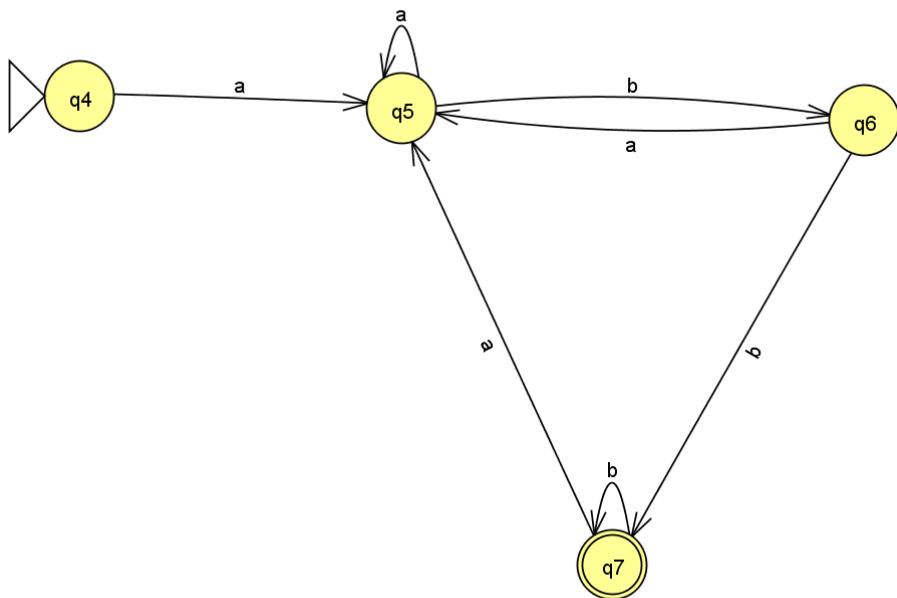
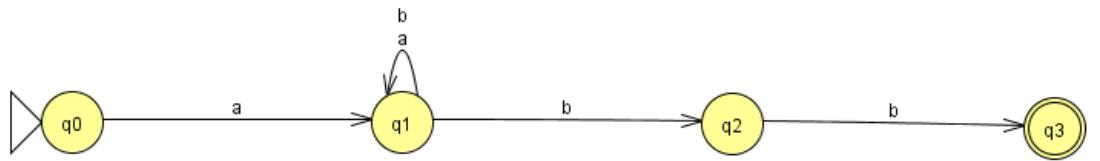




Input	Result
abb	Accept
babb	Accept
abbbbaba	Accept
baabb	Accept
bb	Accept
baa	Reject
bab	Reject
bbaa	Reject
babbbba	Reject
ba	Reject

En este NFA he aplicado la construcción de Thompson para la disyunción, así podía utilizar los dos NFA de los primeros ejercicios para dar lugar al resultado de este. Después realicé el algoritmo de subconjuntos para obtener el DFA equivalente (comprobado además por JFLAP) y después comprobar para ambos (NFA y DFA) que las salidas son las esperadas para las cadenas ejemplo.

4. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ que empiecen por “a” y terminen en “bb”. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.

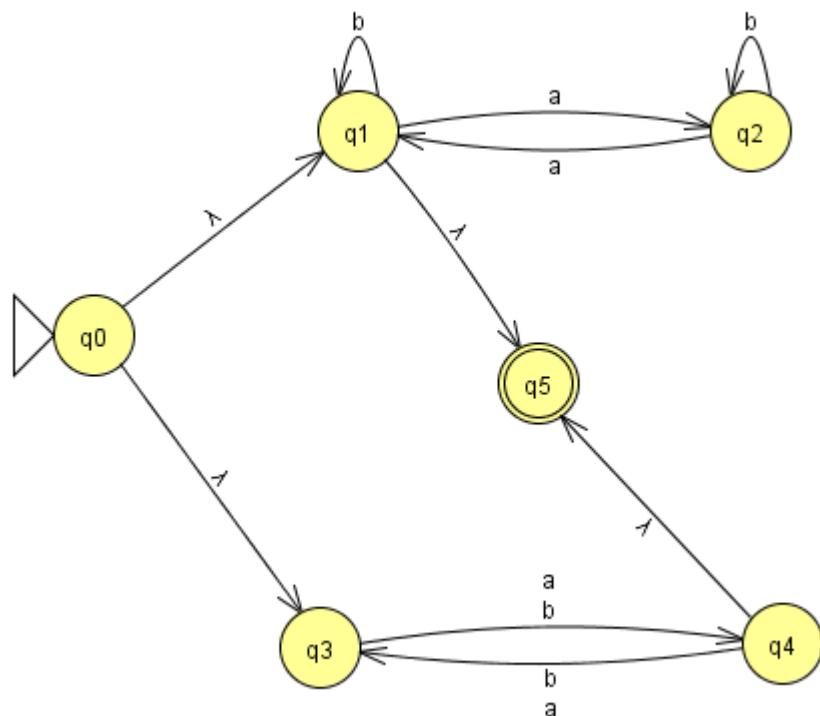


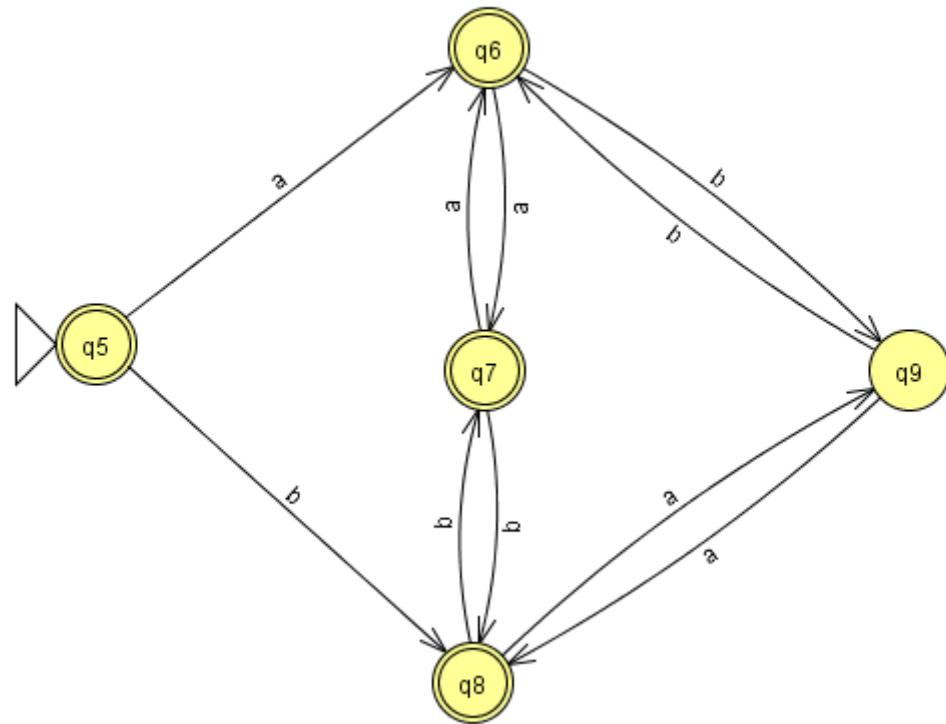


Input	Result
abb	Accept
abbabababb	Accept
abbabb	Accept
abbbb	Accept
aaaaaaabbbbbbb	Accept
ab	Reject
bab	Reject
aaaaa	Reject
b	Reject
bb	Reject

En este caso no me ha sido necesario utilizar el algoritmo de subconjuntos pues he podido realizar un DFA equivalente con el mismo número de estados sin mucha complejidad. El NFA se asegura de que empiece por a y acabe en bb pudiendo introducir cualquier subsecuencia en medio, y el DFA hace lo mismo pero de forma circular, dado que si después de una bb se introduce otro símbolo que no sea una b, no lo acepta hasta que se produzca de nuevo este fenómeno.

5. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\Sigma = \{a, b\}$ con número de “a’s” par o longitud impar. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.



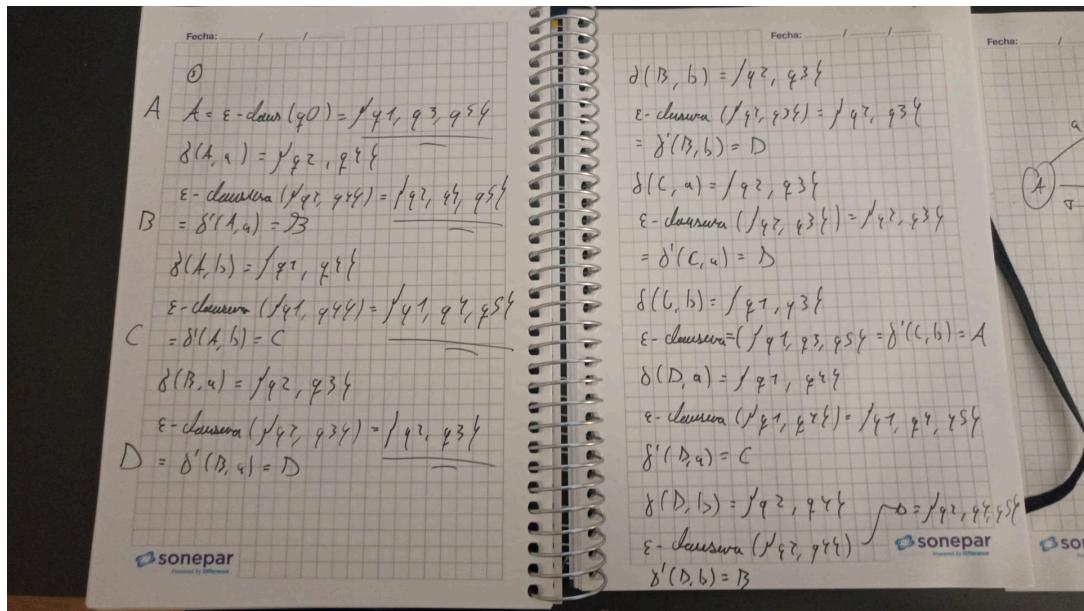


Input	Result
abb	Accept
abbabababb	Accept
abbbab	Accept
abbbb	Accept
aaaaaabbbbbbb	Accept
ab	Reject
abbb	Reject
babaab	Reject
ba	Reject
abbaab	Reject

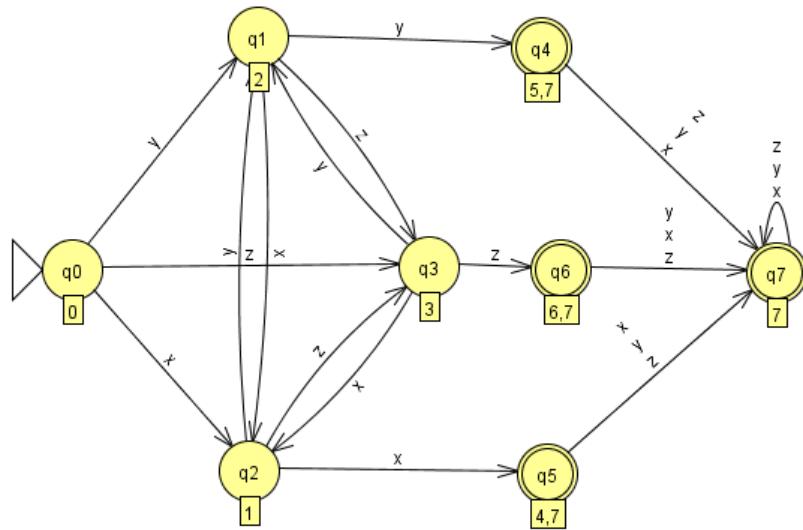
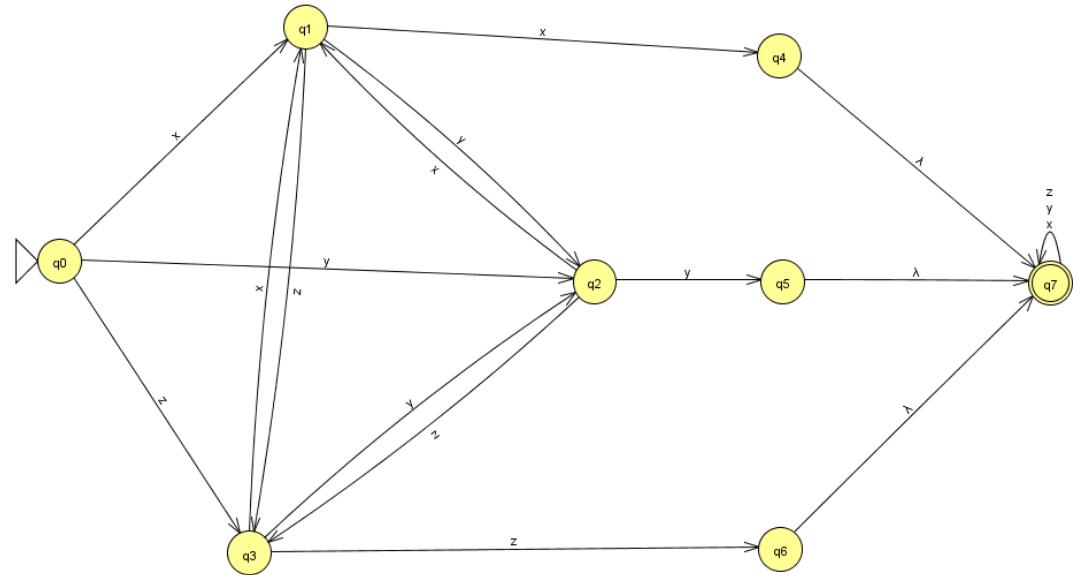
Para realizar el NFA me he fijado en la construcción de Thompson para la disyunción, y aunque no la he aplicado tal y como es, la variación que he propuesto es correcta para este caso, simplemente conectar con epsilon transiciones a los estados iniciales y de aceptación de los NFA ya propuestos anteriormente. Para el DFA he realizado



el algoritmo de subconjuntos y comprobado la equivalencia, adjunto foto.



6. Diseñar un autómata finito no determinista que reconozca cadenas sobre el alfabeto $\{x, y, z\}$ que contengan al menos dos símbolos iguales consecutivos. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.



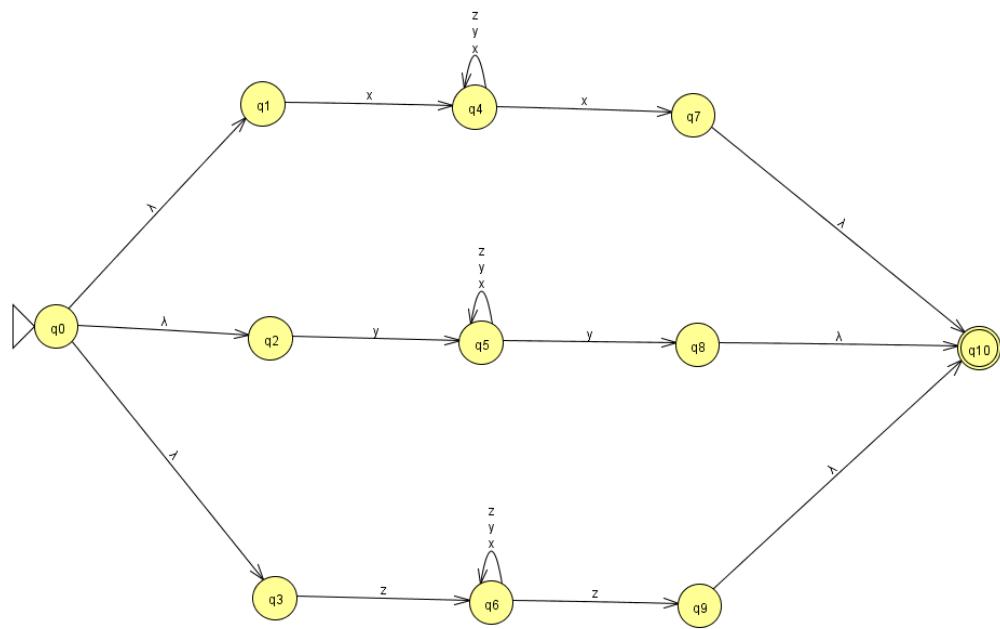


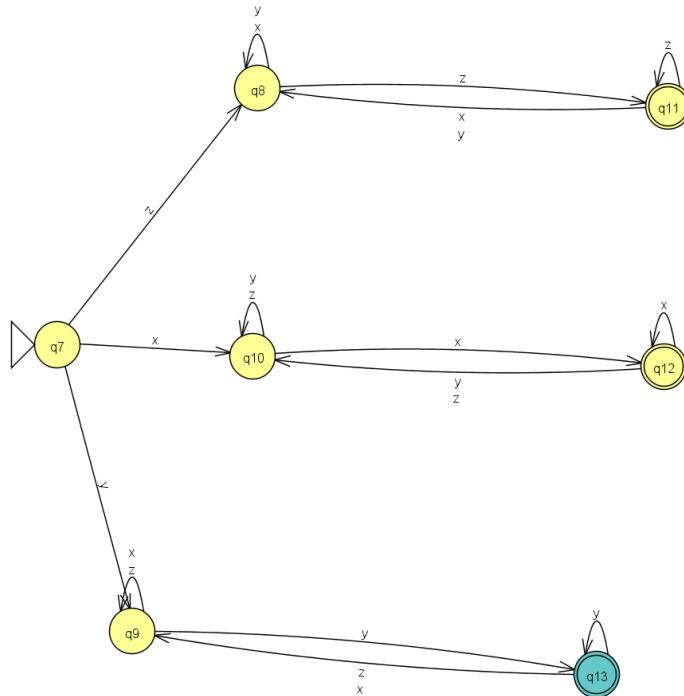
Input	Result
xyxz	Accept
xx	Accept
yy	Accept
zz	Accept
xyzxyz	Accept
xyz	Reject
yzyz	Reject
yzx	Reject
x	Reject
zyz	Reject

Para este ejercicio me surgió una cuestión algo inusual, pues había realizado el NFA pero no contaba como uno, y es que la aplicación solo detecta que es NFA si hay epsilon transiciones o si no hay transiciones de estado para cada símbolo, y dado esto tuve que añadir epsilon transiciones al estado de aceptación. Sin embargo esto me llevó a la conclusión de que como todo DFA también es un NFA, el “NFA” que había realizado al principio era equivalente al que refleja en la foto y cuenta con el mismo número de estados, por lo que no vi necesario realizar el algoritmo a mano dado que tenía el DFA resultante ante mí.

El funcionamiento es muy general, pues es el contrario del ejercicio 10 de los DFA, según el símbolo que reciba irá a un estado que tiene “asignado” ese símbolo, es decir, en cualquier estado estés, si consumes el símbolo “x”, irás al estado q1, y así con los demás.

7. Diseñar un autómata finito no determinista que reconozca cadenas w sobre el alfabeto $\{x, y, z\}$, con $|w| \geq 2$, tales que w empieza y termina por el mismo símbolo. A partir del NFA diseñado, obtenga un DFA mínimo equivalente.





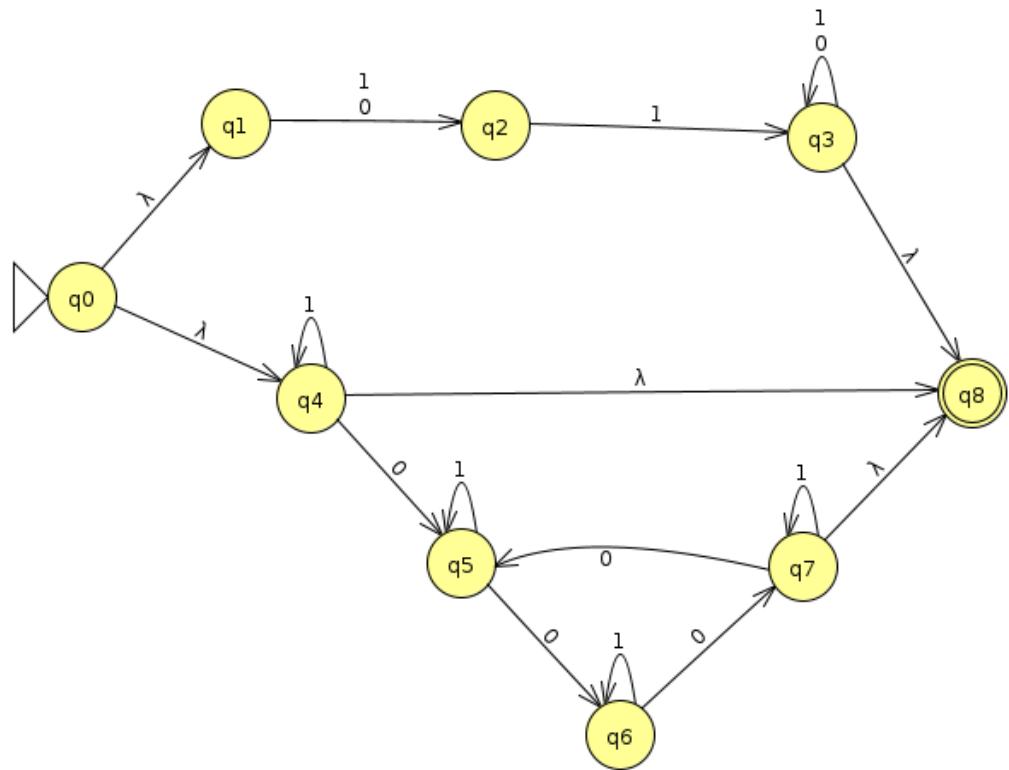
Input	Result
<u>xyz</u>	Accept
<u>zz</u>	Accept
<u>yy</u>	Accept
<u>xyyyyyzzzzzzx</u>	Accept
<u>zyzxzxz</u>	Accept
<u>xyy</u>	Reject
<u>x</u>	Reject
<u>yzyzxz</u>	Reject
<u>xyz</u>	Reject
<u>zyx</u>	Reject

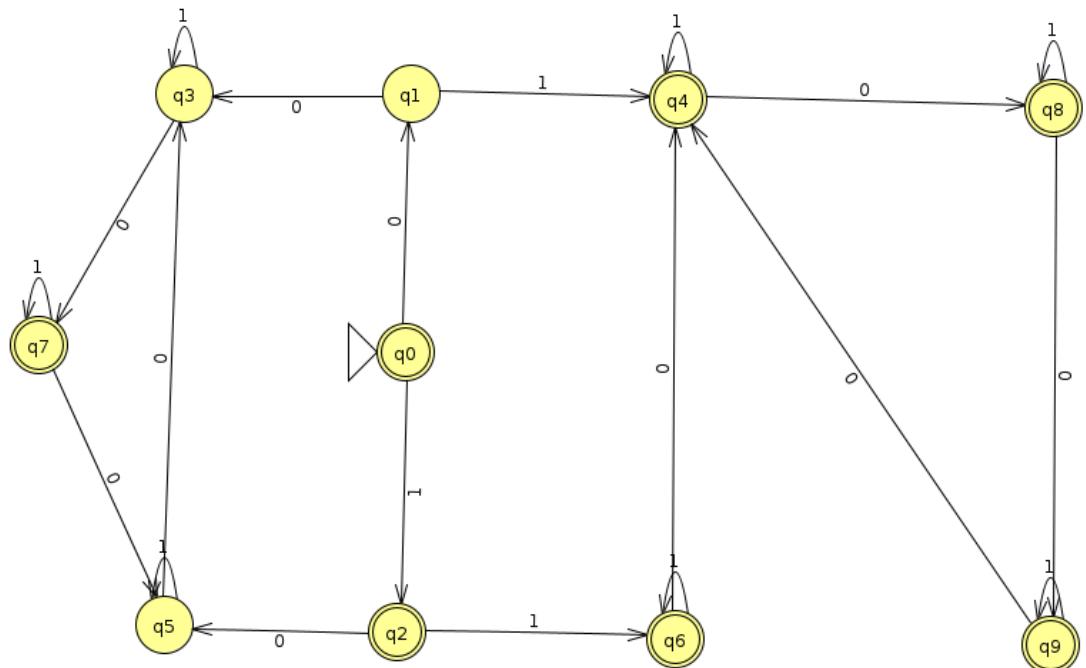
En este último caso me ha sucedido igual que en algunos otros, el DFA era bastante sencillo de implementar teniendo el NFA y he comprobado la equivalencia de estos.

Para el NFA aplico la construcción de Thompson para la disyunción, dividiendo el enunciado en 3 casos posibles, que empiece por x, y, o z. De esta manera puedo saber en qué símbolo deberá terminar la cadena para ser aceptada.

Ejercicio de clase:

1. Cadenas que tienen un 1 en la segunda posición o nº de ceros múltiplo de 3.





Input	Result
0110	Accept
101010	Accept
000111	Accept
0100	Accept
1100	Accept
10	Reject
0011	Reject
10000	Reject
1	Reject
0000	Reject



ULL

Fecha: / /

$A = \{90, 97, 94, 98\}$

$\delta(A, 0) = \{92, 95\} = \delta'(A, 0) = B$

$\delta(A, 1) = \{92, 94\} = \text{E-class}$

$E\text{-class} = \{92, 94, 98\} = \delta'(A, 1) = C$

$\delta(B, 0) = \{96\} = \delta'(B, 0) = D$

$\delta(B, 1) = \{93, 95\}$

$E\text{-class} = \{93, 95, 98\} = \delta'(B, 1) = E$

$\delta(C, 0) = \{95\} = \delta'(C, 0) = F$

$\delta(C, 1) = \{93, 94\}$

$E\text{-class} = \{93, 94, 98\} = \delta'(C, 1) = G$

Fecha: / /

$\delta(D, 0) = \{97\}$

$E\text{-class} = \{92, 95\} = \delta'(D, 0) = B$

$\delta'(D, 1) = \{96\} = D$

$\delta(E, 0) = \{93, 96\}$

$E\text{-class} = \{92, 95, 98\} = \delta'(E, 0) = H$

$\delta(E, 1) = \{93, 95\}$

$E\text{-class} = \{92, 95, 98\} = \delta'(E, 1) = D$

$\delta'(F, 0) = \{96\} = D$

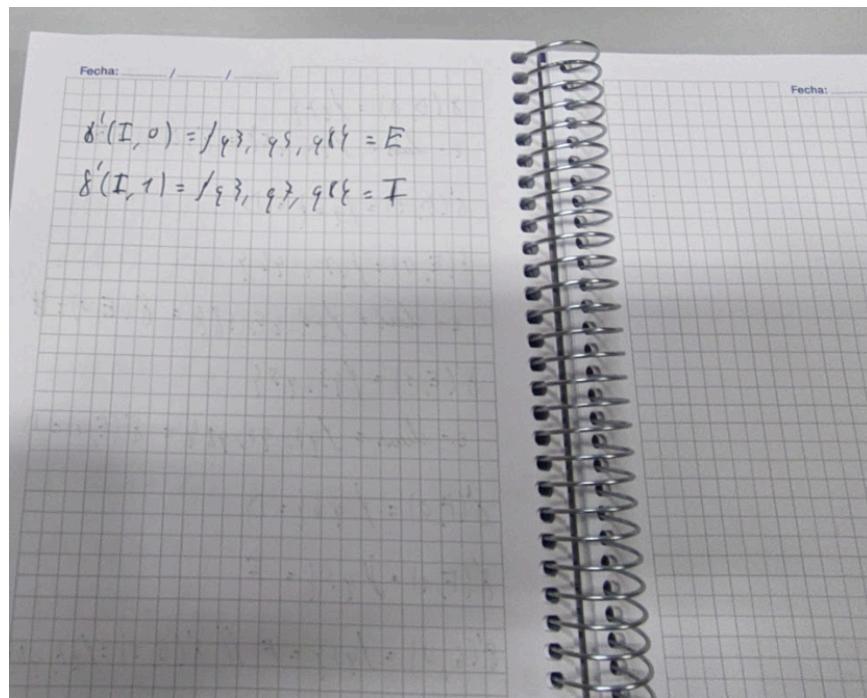
$\delta'(F, 1) = \{95\} = F$

$\delta'(G, 0) = \{95\} = F // \delta'(G, 1) = \{92, 98\} = G$

$\delta(H, 0) = \{93, 97, 98\} = I$

$\delta'(H, 1) = \{93, 96, 98\} = H$

sonepar
Powered by Difference



Para este NFA he utilizado la construcción de Thompson para la Disyunción pues así puedo separar las condiciones que enumera el enunciado. Desde el estado q_1 permito la entrada de cualquier símbolo mientras que en q_2 es obligatorio que consuma un 1 para transicionar y así llegar con epsilon transición al estado de aceptación. Por otro lado en q_4 aceptamos la cadena vacía dado que 0 es múltiplo de 3 y obligamos a que haya 0 en la cadena para transicionar, de lo contrario no podrá cambiar de estado, en caso de que ese nº de 0 sea múltiplo de 3 podrá ser aceptada la cadena mediante epsilon transición a q_8 .