1. For any DFA, prove that δ^* is associative; that is for any input strings x and y, $\delta^*(q, xy) = \delta^*(\delta^*(q, x), y)$.

   Solution:
   Do induction by the length of y.

   Base case $y = \epsilon$ and $|y| = 0$:
   $\delta^*(q, xy) = \delta^*(q, x) = \delta^*(q, x\epsilon) = \delta^*(\delta^*(q, x), \epsilon) = \delta^*(q, x)$

   When $y = a$, where $a \in \Sigma$ and $|y| = 1$:
   $\delta^*(q, xy) = \delta^*(q, xa) = \delta(\delta^*(q, x), a) = \delta^*(\delta^*(q, x), a) = \delta^*(\delta^*(q, x), y)$

   Inductive case, assume the theorem holds for $|y| \leq n, n \geq 0$:
   When $|y| = n + 1$, $y = za$, where $a \in \Sigma$ and $|z| = n$

$$\begin{aligned}
\delta^*(q, xy) =& \delta^*(q, xza) \\
=& \delta^*(\delta^*(q, xz), a) \\
=& \delta^*(\delta^*(\delta^*(q, x), z), a) && \text{(by I.H.)} \\
=& \delta(\delta^*(\delta^*(q, x), z), a) \\
=& \delta^*(\delta^*(q, x), za) \\
=& \delta^*(\delta^*(q, x), y)
\end{aligned}$$

2. In the cross-product construction, we also extended δ^* (see slides, the last page). There are two very different definitions for δ^*, now prove they are equivalent.
   Hint: we will take the first definition of δ^* on the new DFA, and then prove the following theorem by induction:

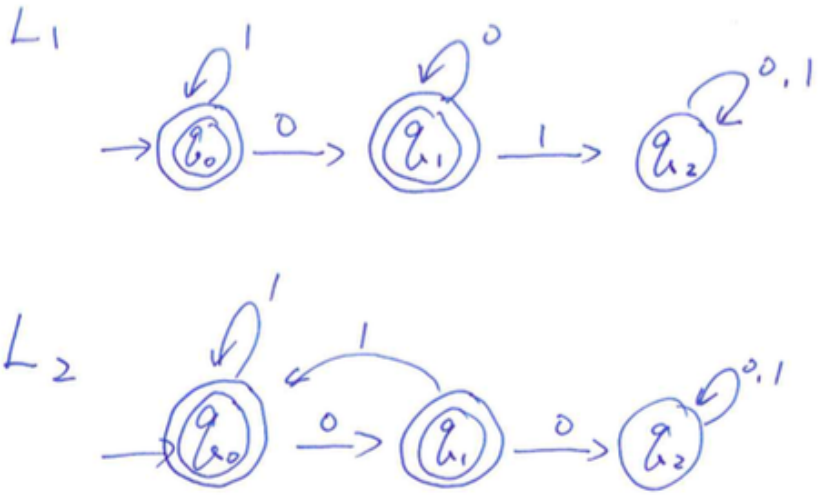
   $$\delta * ((p, q), w) = (\delta_1^*(p, w), \delta_2^*(q, w))$$

   Solution:
   Proof:
   Do induction proof by the length of w:
   Base case:
   when $w = \epsilon$
   $\delta^*((p, q), \epsilon) = (p, q)$
   $\delta^*((p, q), \epsilon) = (\delta_1^*(p, \epsilon), \delta_2^*(q, \epsilon)) = (p, q)$
   Inductive case: assume theorem holds for $|w| \leq n, n \geq 0$
   when $|w| = n + 1$, denote $w = w'a$ where $a \in \Sigma$ and $|w'| = n$

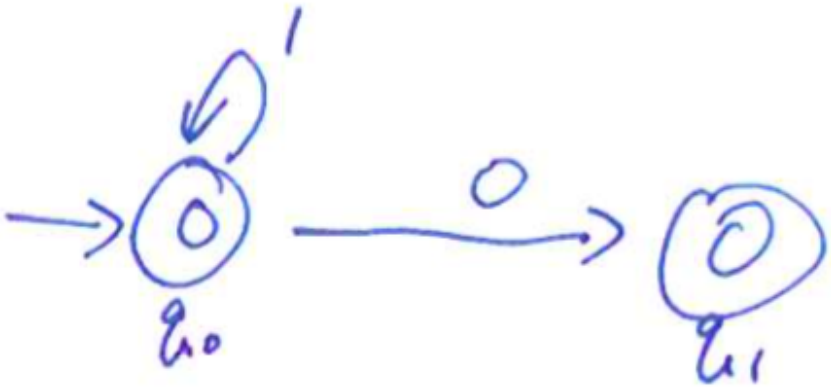
$$\begin{aligned}
\delta^*((p, q), w) =& \delta(\delta^*((p, q), w'), a) && \text{(by definition of } \delta^*) \\
=& \delta(\delta^*((p, q), w'), a) && \text{(by I.H.)} \\
=& \delta((\delta_1^*(p, w'), \delta_2^*(q, w')), a) && \text{(by definition of } \delta^*) \\
=& (\delta(\delta_1^*(p, w'), a), \delta(\delta_2^*(p, w'), a)) && \text{(by definition of } \delta) \\
=& (\delta_1^*(p, w'a), \delta_2^*(q, w'a)) && \text{(by definition of } \delta) \\
=& (\delta_1^*(p, w), \delta_2^*(q, w))
\end{aligned}$$

3. L_1 = bitstrings that do NOT contain 01.
   L_2 = bitstrings that do NOT contain 00.

Construct machines for L1 and L2, but both in 2 states plus the trap state. Now construct new machines that recognizes:
(a) $L_1 \cap L_2$
(b) $L_1 \cup L_2$
(c) $L_1 - L_2$
(d) $L_2 - L_1$
Solution:
First construct L_1 and L_2



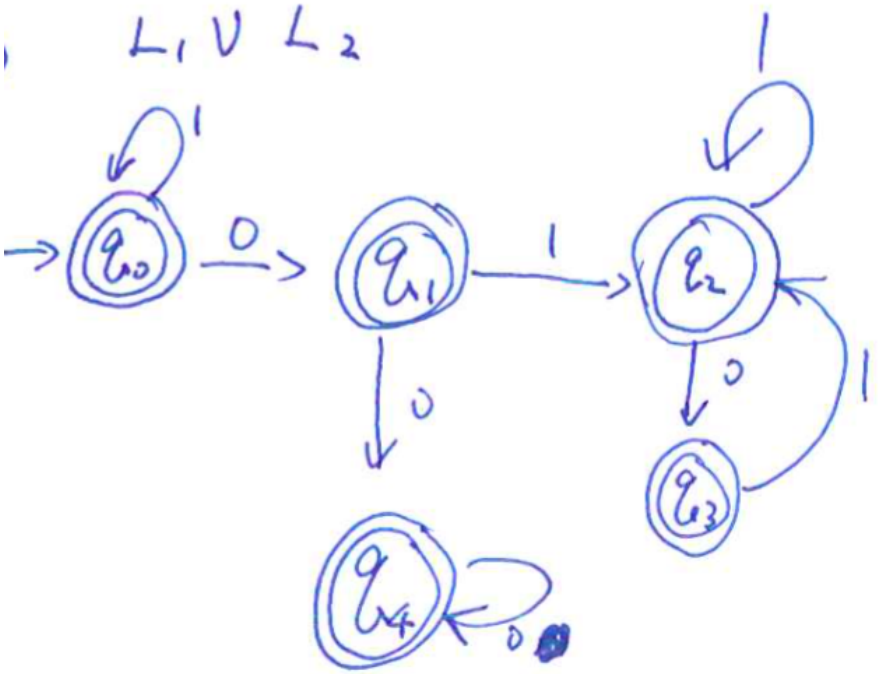
(a) $L_1 \cap L_2$



The intersection of L_1 and L_2 is a language in which strings contain at most one 0 (but any number of 1s), and the 0 must be the last character. This makes sense since L_1 rules out 01, and L_2 rules out 00, which implies that $L_1 \cap L_2$ does not allow anything to follow a 0. This DFA need to accept the string immediatly when it recieves 0. From the graph we can tell the combined DFA will accept the language when it receives 0.

(b) $L_1 \cup L_2$

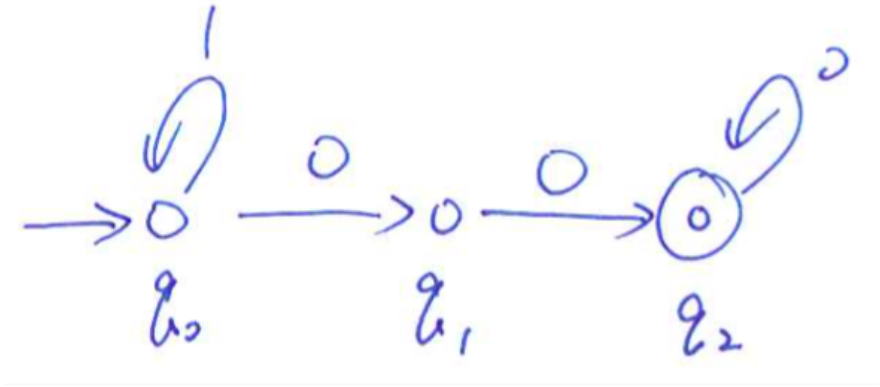
Cont.



The union of L_1 and L_2 is a langauge in which strings either contain no 0 (but any number of 1s, q_0), or one 0 as the end (q_1), or receives one 0 then split into two different branches. The first branch allows 0 only (q_1, q_4), and the second does not allow continuous 0s (q_1, q_2, q_3).

This makes sense since if a string reach any branch (a string not in (a)), it must contain one 00 or 01. If it contains one 00, there must be all 0s followed by (it cannot contain 01); if it contains one 01, the rest 0s must not be continuous (it cannot contain 00). This DFA can accept strings at any state. Since our discussion include all the cases of strings does not contain 00 or 01, this DFA would recognize the language.

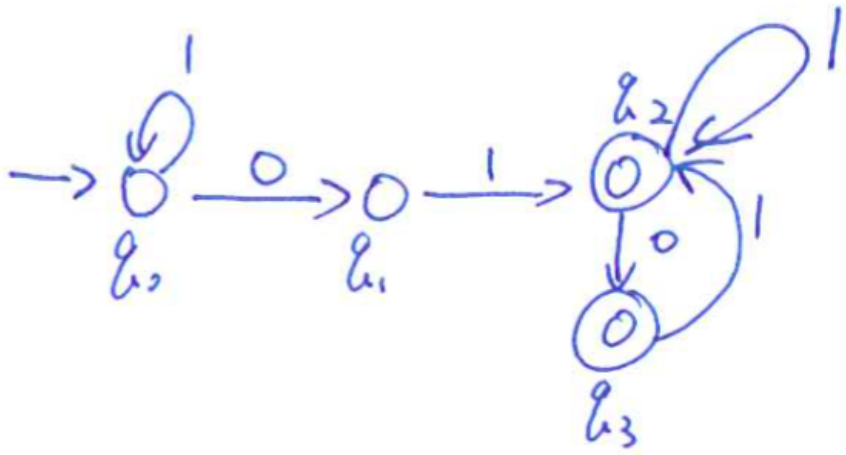
(c) $L_1 - L_2$



$L_1 - L_2$ is a language in which strings contain 00 but not 01, i.e., the first branch of (b). The DFA accepts strings that already contain 00.

(d) $L_2 - L_1$

Cont.



$L_2 - L_1$ is a language in which strings contain 01 but not 00, i.e., the second branch of (b). The DFA accepts strings that already contain 01.

4. If δ is a partial function in each of the two machines, how do you construct the combined DFA? note that the state (trap, trap) is always rejecting so you don't need to include it. Also, for different operations (intersection, union, ...), you can define different partial δ to produce a smaller DFA (e.g. for intersection you need fewer states than union).
Solution:
$M_1 = \{Q_1, \Sigma, \delta_1, q_1, F_1\}$
$M_2 = \{Q_2, \Sigma, \delta_2, q_2, F_2\}$
Please note that both δ_1 and δ_2 here are partial funtion. q_1 and q_2 are the initial state for M_1 and M_2 respectively. The set of states Q for new DFA is $Q_1 \times Q_2$. The initial state for new DFA is $q_0 = (q_1, q_2)$. The only thing different here is the transaction function. T is trap state.

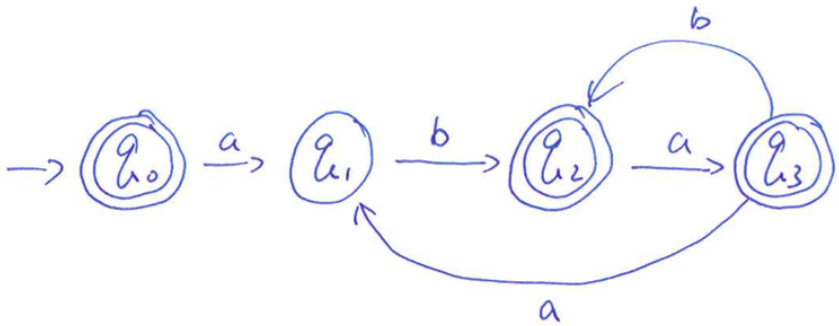
In the case of union, δ_U is the transaction function for our combined DFA.
The final state $F_U = \{(q_f, q'_f) | q_f \in F_1 \vee q'_f \in F_2\}$.

$$\delta_U((p,q),a)) \begin{cases} (\delta_1(p,a), \delta_2(q,a)) & \text{if } \delta_1(p,a) \text{ and } \delta_2(q,a) \text{ are defined} \\ (\delta_1(p,a), T) & \text{if } \delta_2(q,a) \text{ is not defined} \\ (T, \delta_2(q,a)) & \text{if } \delta_1(p,a) \text{ is not defined} \\ \text{undef} & \text{otherwise} \end{cases}$$

In the case of intersection, δ_I is the transaction function for our combined DFA. The final state $F_I = \{(q_f, q'_f) | q_f \in F_1, q'_f \in F_2\}$.

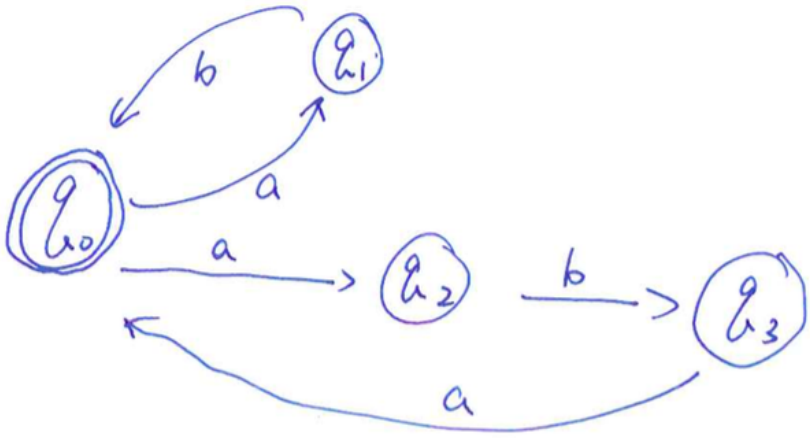
$$\delta_I((p,q),a)) \begin{cases} (\delta_1(p,a), \delta_2(q,a)) & \text{if } \delta_1(p,a) \text{ and } \delta_2(q,a) \text{ are defined} \\ \text{undef} & \text{otherwise} \end{cases}$$

5. Construct a DFA for the language $\{ab, aba\}^*$, i.e., $\{\epsilon, ab, aba, abab, abaaba, ababa, abaab, ababab, ...\}$
Solution:

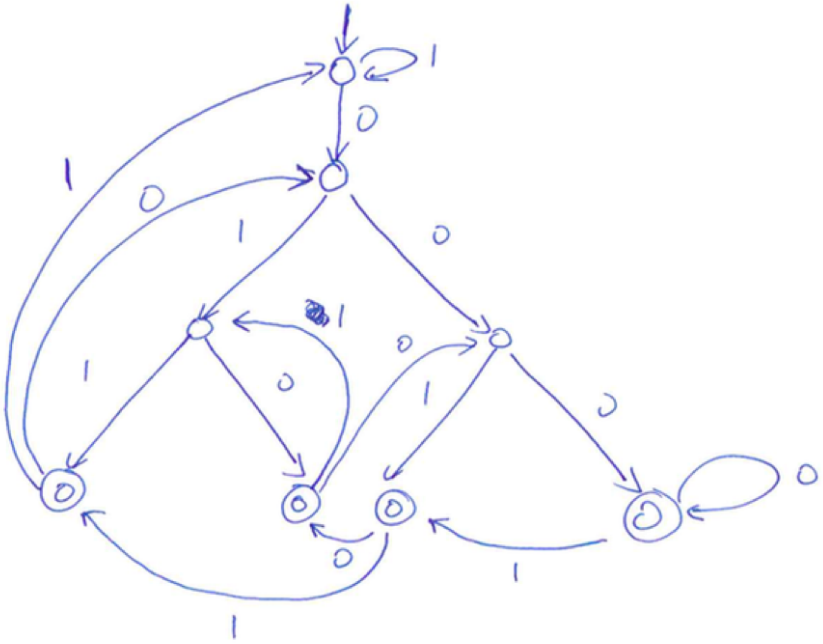


Cont.

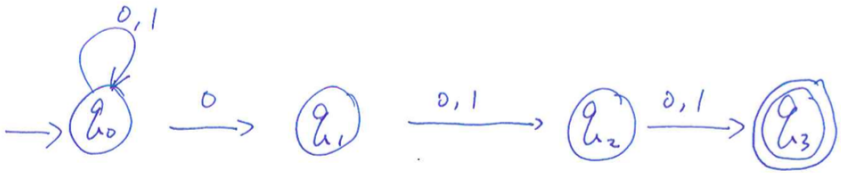
6. Redo the above using NFA.
Solution:



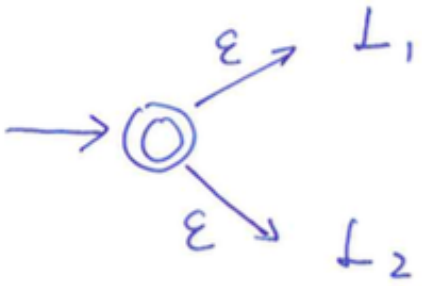
7. Construct a DFA for bitstrings with 0 as the third last symbol from the end.
Solution:



8. Redo the above using NFA.
Solution:

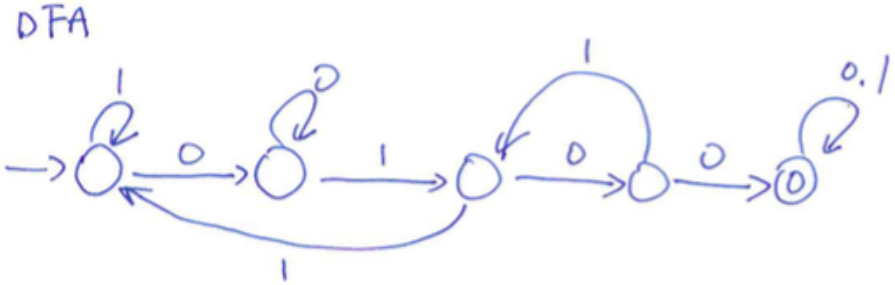


9. Redo Problem 3 (b) using NFA.
Solution:



NFA will not help the DFA for the cases of NOT contain some string.

10. Construct both DFA and NFA for: bitstrings that do contain 0100.
Solution:



NFA will not help the DFA for the cases of NOT contain some string.

The End.