Problem type 1:

L_1, L_2, ... are all regular languages representable by the DFAs $M_1 = (Q_1, \Sigma, \delta_1, s_1, A_1)$, $M_2 = (Q_2, \Sigma, \delta_2, s_2, A_2)$, etc.

Give the *formal* description of the NFA (N') that describes the language below that is the composite of one or more of the above regular languages.

> *(See variants below)*

We want to see the definition in terms of: $Q' = ...$, $\delta' = ...$, $s' = ...$, $A' = ...$ Assume $\Sigma = \{\mathtt{0}, \mathtt{1}\}$.

a. **BYC**

$L' = L_1 \cup L_2$

Solution: Two ways to do this. We can either do this the proper DFA way where:

- $Q' = Q_1 \times Q_2 = \{(q_1, q_2) \mid q_1 \in Q_1, q_2 \in Q_2\}$
- $s' = (s_1, s_2)$
- $\delta : Q' \times \Sigma \to Q$ where $\delta'((q_1, q_2), a) = (\delta_1(q_1, a), \delta_2(q_2, a))$
- $A' = \{(q_1, q_2) \mid q_1 \in A_1 \text{ or } q_2 \in A_2\}$

Or since the problem specifically states that we define the NFA we can use epsilon transitions:

- $Q' = Q_1 \cup Q_2 \cup \{s'\}$
- $s' = s'$
- $\delta' = \delta_1 \cup \delta_2 \cup \{\delta'(s', \varepsilon) = \{s_1, s_2\}\}$
- $A' = A_1 \cup A_2$

■

b. **BYE**

L' = $\overline{L_1}$

Solution: Since we are given the *DFA* for the original languages, flipping the accept/reject states will suffice

- $Q' = Q_1$
- $s' = s_1$
- $\delta' = \delta_1$
- $A' = Q_1 \backslash A_1$

Potential future question: Why can't we reverse the states on a NFA to get the complement?

■

c. **BYA**

$L' = L_1 \cdot L_2$

Solution: Just arrange one DFA after another and use epsilon transitons:

- $Q' = Q_1 \cup Q_2$
- $s' = s_1$
- $\delta' = \delta_1 \cup \delta_2 \cup \{\delta'(q_a, \varepsilon) = \{s_2\}\} \qquad \forall q_a \in A_1$
- $A' = A_2$

■

d. **BYH**

$L' = L_1 \cup \{\varepsilon\}$

(adding empty string to strings L_1 regardless of if it is there (or not))

Solution: A number of solutions but the safest solution is to simply have a new start state that is a accepting state and the epsilon transition to the rest of the string:

- $Q' = Q_1 \cup \{s'\}$
- $s' = s'$
- $\delta' = \delta_1 \cup \{\delta'(s', \varepsilon) = s_1\}$
- $A' = A_1 \cup s'$

Potential future question why can't we simple make s_1 a accepting state? ■

e. **BYD**

$L' = \mathtt{0} \cdot L_1$

Add a `0` before every string in L_1

Solution:

- $Q' = Q_1 \cup \{s'\}$
- $s' = s'$
- $\delta' = \delta_1 \cup \{\delta'(s', \mathtt{0}) = s_1\}$
- $A' = A_1$

■

f. **BYF**

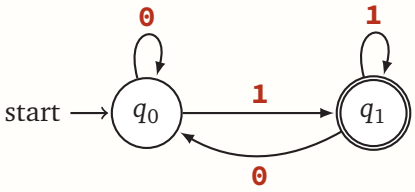
$L' = L_1 \cup \{\mathtt{0}\}$

(add the string "**0**" to strings in L_1 regardless of if it is there (or not))

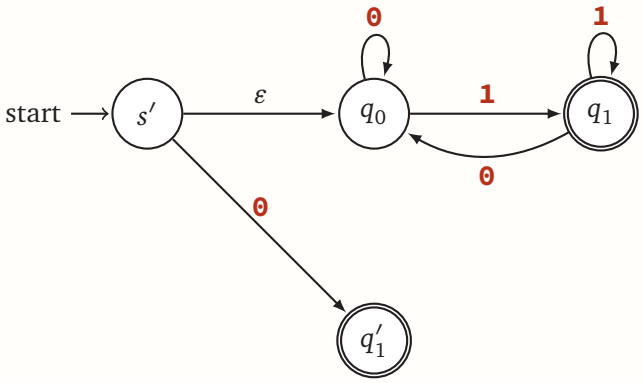
Solution:

- $Q' = Q_1 \cup \{s', q_1'\}$
- $s' = s'$
- $\delta' = \delta_1 \cup \{\delta'(s', \mathtt{0}) = q_1', \delta'(s', \mathtt{0}) = s_1\}$
- $A' = A_1 \cup q_1'$

Why do you need the extra start state? Suppose L_1 is represented by the following DFA:



This DFA represents all strings ending in **1**. Now if we simply add a single state (q_1') and transition to it from q_0 on character **0**, we would accept the string **0**, but we would also accept **00**, and **00000**, and so on. That's not what we want. The potential for the start state to be part of a loop means that we should add a new start state to make sure this new string accepts the string **0** and doesn't help accepting any other strings. Hence, the transformation above prevents this by adding the new start state:



Huge thanks to CA Shreyansh Agrawal for identifying the bug. I am very blessed to have an amazing support staff and this is just one of many examples why they're awesome.

■

g. **BYB**

$L' = L_1 \cdot \mathtt{0}$

Add a $\mathtt{0}$ after every string in L_1

Solution:

- $Q' = Q_1 \cup \{q'_F\}$
- $s' = s_1$
- $\delta' = \delta_1 \cup \{\delta'(q_f, \mathtt{0}) = q'_F\} \quad \forall q_f \in A_1$
- $A' = q'_F$

■

h. **BYG**

$L' = \{\mathtt{0}^* a_0 \mathtt{0}^* a_1 \dots \mathtt{0}^* a_{n-1} \mathtt{0}^* | w = a_0 a_1 \dots a_{n-1}, w \in L_1, a_0, a_1, \dots, a_{n-1} \in \Sigma\}$

(basically what this is saying is that L' will accept any string in L_1 and that string in L_1 can have any number of $\mathtt{0}$'s shoved in between each character).

Solution: The hardest part of this question is understanding the language. But the description starts that this is simply L_1 with runs of $\mathtt{0}$'s shoved between each character. Once we know that, hopefully we see that the solution is to simply shove a bunch of self loops in the DFA to allow the acceptance of additional $\mathtt{0}$'s. So the NFA for L' becomes:

- $Q' = Q_1$
- $s' = s_1$
- $\delta' = \delta_1 \cup \{\delta'(q_a, \mathtt{0}) = q_a\} \quad \forall q_a \in Q_1$
- $A' = A_1$

Note: A earlier version of this problem had a slightly different formal definition that didn't match the textual description completely. We edited the formal definition in the question/solutions documents but when grrading we assumed either definition. ■