

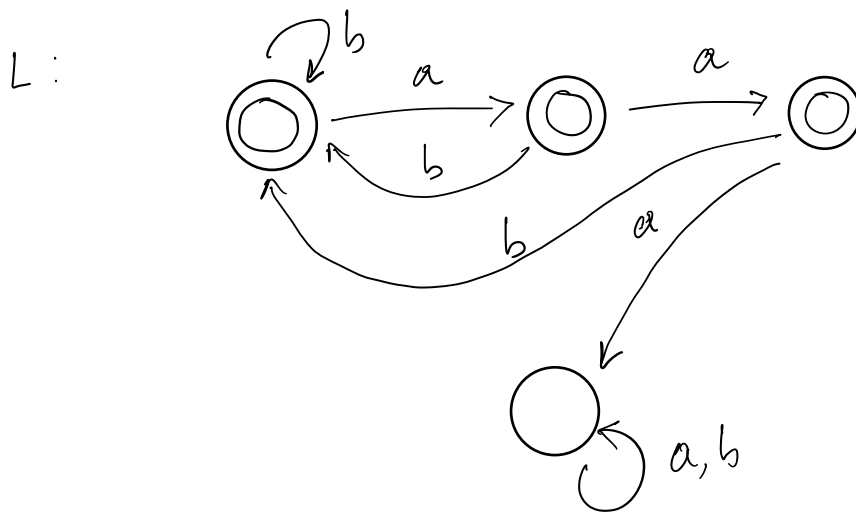
# Theory of Computation '23

## Problem Set 1

1. Give a DFA over the alphabet  $a, b$  for each of the following:

- a. Accepts strings which do not contain  $aaa$  as a substring.

**Solution,** The best way to solve this one is by eyeballing. What you need to realize is that there should be states to 'memorize' how many a's have been encountered so far.



An alternative way to solve this one would be to use the closure property under complements. This means, if a language  $L$  which is a subset of  $\Sigma^*$  is described by a DFA  $M$ , then the language  $\tilde{L} = \Sigma^* \setminus L$  is accepted by a DFA  $M'$  which has the same states and transitions as  $M$ , but the accept and non-accept states are reversed. You can use this to solve the above by first constructing a DFA for the language

$$\tilde{L} = \{w \in \Sigma^* | w \text{ contains the substring } aaa\}$$

and then construct the complement DFA.

**Additional Remark:** The above fact does not hold for NFAs. Construct an example (that needs at least two states) to show that it does not. The solution here is trivial. Just think of an NFA where the alphabet is  $\Sigma = \{a, b\}$  and it accepts only the single-length string  $a$ . An NFA for this will have two states :  $q_0, q_1$ .  $q_1$  is an accepting state. The only transition we allow is  $\delta(q_0, a) = q_1$ . Clearly reversing the accepting and non-accepting state does not make the NFA accept the language  $\Sigma^* \setminus a$

- b. Accepts all strings which contain  $aab$  as a substring but do not contain  $aaa$  as a substring. (Hint: First construct two simple DFAs that accept strings satisfying the given conditions and then try to ‘compose’ them. You should not require more than 9 states for the final DFA)

**Solution.** Usually, to solve these kind of exercises, you would be expected to break down the language in to simpler components. Here, the language  $L$  is an *intersection* of two different languages as follows

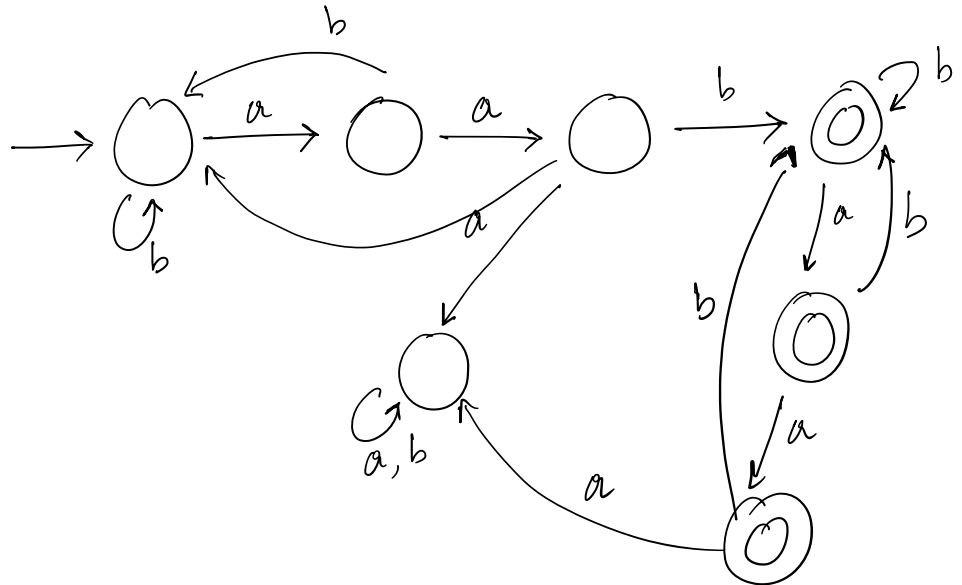
$$L_1 = \{w \in \Sigma^* | w \text{ does not contain } aaa\}$$

$$L_2 = \{w \in \Sigma^* | w \text{ contains } aab\}$$

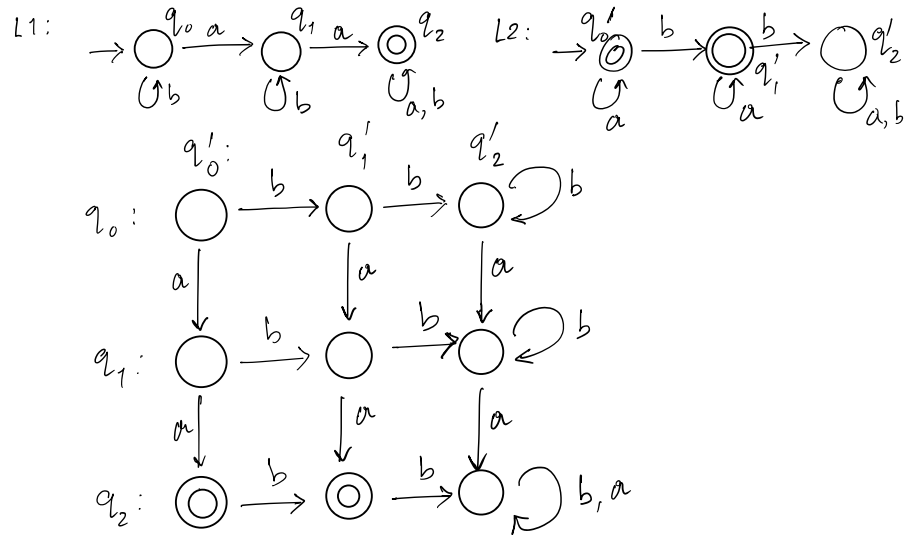
So first we will construct the individual DFAs for  $L_1$ , say  $M_1$  and  $L_2$ , say  $M_2$  as follows. Now one option is to apply the same construction as the proof of union closure done in class. The only difference is, here, the accepting states would be those that contain accept states from both  $M_1$  and  $M_2$ . The final construction will have 16 states. However, as you can imagine, this will be quite complicated, especially the transitions.

So for this case, its is better to not take the intersection blindly but try to imagine what is going to happen. One potential thought process is as follows. Let us first take the DFA for the  $L_2$  since it is somewhat of an ‘must include’ condition. Now once this reaches the accepted state for  $L_2$ , all you need to make sure that you satisfy  $L_1$  - in other words, no matter what symbols you see, you will remain in accepted state, unless - you see  $aaa$ . The overall construction is shown below.

$$L = \{w \in \{a,b\}^* | w \text{ contains } aab \text{ but not } aaa\}$$

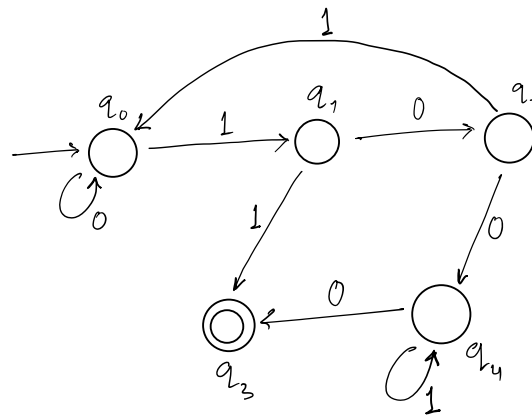


- c. Accepts  $L = \{w \in \{a,b\}^* | w \text{ contains at least two } a's \text{ and at most one } b's\}$  **Solution.** Apply the ‘intersection’ as above. For the second language, it might be easier to use the complementing technique.



- d. Accepts all strings that are binary representations of numbers that are  $3 \pmod{5}$ , that is leaves a remainder 3 when divided by 5. (Hint : This one is not easy. Any integer  $\pmod{5}$  can take 5 possible values. Try creating a state for each of these values that is supposed to 'memorize' the decimal value of the string read so far  $\pmod{5}$  )

**Solution.** This one is slightly hard unless you know the trick. The idea is the following. For any non-negative integer  $x$ ,  $x \pmod{5}$  can take 5 possible values  $\{0, 1, 2, 3, 4\}$ . We will now have 5 states in our DFA,  $q_0, \dots, q_4$ , each representing one of these 5 possibilities. The state  $q_i, i = 0, 1, 2, 3, 4$  is supposed to 'memorize' the fact that the substring seen by the DFA so far is a binary string whose decimal is  $i \pmod{5}$ . Now the transitions should not be hard to figure out. Suppose you are at state  $q_i$ , Then  $\delta_{q_i, 0}$  'means' that you multiply the number by 2. Hence, this should take you to the state that represents  $2i \pmod{5}$ . Similarly,  $\delta_{q_i, 1}$  takes you to the state that represents  $(2i + 1) \pmod{5}$ . Finally, it is obvious that  $q_3$  is the accept state here.



2. Give a NFA for the following

$$L = \{wc | w \in \{0, 1, 2\}^*, c \in \{0, 1, 2\} \text{ and } c \text{ occurs in } w\}$$

(Hint : Use closure)

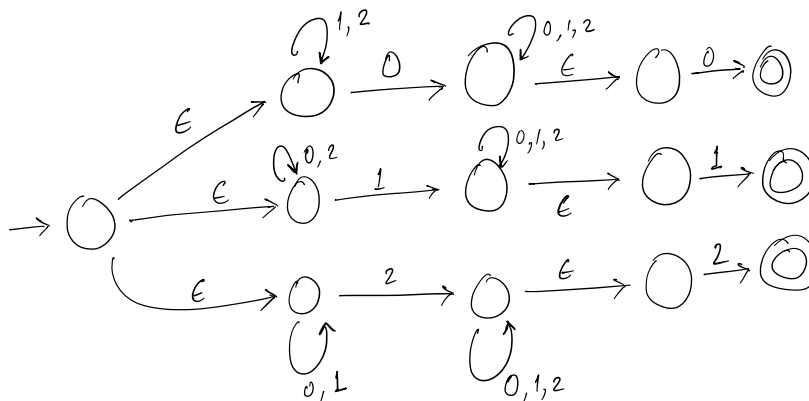
**Solution.** This one is very easy ones you know how to use the closure constructions properly. Again, it is easy to observe that  $L$  is a union of languages  $L_c$ ,  $c=0,1,2$  such that

$$L_c = \{wc|w \in \{0,1,2\}^* \text{ and contains } c\}$$

Ok, now how to construct  $L_c$  ? Again, think of breaking it down in to concat of two languages -  $\{wc|w \in \{0,1,2\}^* \text{ and contains } c\}$  and  $\{c\}$ . Now construct an NFA for the first one. This is almost trivial to construct. You just need two states :  $q_0$  and  $q_1$ , where  $q_1$  is accepting. Now  $q_0$  will have transitions labelled with everything other than  $c$  back to itself.  $\delta(q_0, c) = q_1$ .  $q_1$  has all transitions pointing back to itself.

Now the NFA for  $\{c\}$  is even easier. Again two states  $q_0, q_1$  and just one transition out of  $q_0$  to  $q_1$  which is labelled with  $c$ .  $q_1$  is the only accepting state, Note that you do not need all transitions for an NFA.

One you have the above pieces, concatenation is easy. Finally, take the union for  $c = 0, 1, 2$ .



3. Suppose  $\Sigma = \{a_1, a_2, \dots, a_k\}$ . Given an NFA with  $k + 1$  states that accept the following language

$$L = \{w | \exists i, 1 \leq i \leq k, w \text{ does not contain } a_i\}$$

(Hint: Break down in to smaller NFAs)

**Solution.** The trick is to again break this down in to smaller components. First design NFAs for the individual  $i = 1, 2, \dots, k$  such that it does not accept string which contains  $a_i$ . This can be easily done with only one state. That state itself is the starting and final one. The only trick is that there would be *no transitions* labelled with  $a_i$ . This means, whenever it sees an  $a_i$ , the computation will be dead. Then just take the union of these - you just need one additional state.

4. Construct a DFA for

$$L = \{w \in \{a, b\}^* | w \text{ does not contain exactly two } a\}$$

**Solution** Again, one can first construct a DFA for the complement language and then just swap the states.

