Assignment 1

CS340: Theory of Computation

Abhishek Pardhi, Aayush Kumar, Sarthak Kohli
200026, 200008, 200886
UG Y20 - CSE
`apardhi20@iitk.ac.in`, `aayushk20@iitk.ac.in`,
`sarthakk20@iitk.ac.in`

Indian Institute of Technology
Kanpur

Department of Computer Science & Engineering

January 10, 2023

Contents

1 Question 1

We can divide all possible inputs into three cases: ϵ, first digit is 0, first digit is 1

1. **Case 1:**
   Here, our input is ϵ, thus we end at our start state q_0, which is not an accepting state. Thus, this input is rejected.
2. **Case 2:**
   Here, the first digit of our input is 0. So, we will move from start state q_0 to q_1, an accepting state. Any input (0 or 1) following this first digit will lead us to state q_2, which does not have any outgoing transitions, and is a non-accepting state.
   Thus, the only possible input that can be accepted in this case is **0**.
3. **Case 3:**
   Here, the first digit of our input is 1. So, we will move from start state q_0 to q_3. We can observe that upon reaching q_3, we can transition only to states q_4 and q_5 and possibly back to q_3.
   - To transition to q_5 and back, we will first need to transition from q_3 to q_5, then perform any number of self transitions in q_5 and finally transition back to q_3.
   - Based on the description of the DFA given to us, this means that we must first take '0' input to transition to q_5, take any number of '1' inputs, then take another '0' input to transition back to q_3.
   - Thus, our input for this transition to q_5 and back is defined as 01^*0.
   - Simiarly, transitions to q_4 and back will be defined by inputs 10^*1.

   We can perform any number of these returning transitions in any order and still remain at q_3. Thus, the regular expression that describes the input for which we start and end at q_3 is $(10^*1 + 01^*0)^*$
   Thus, we can say that to transition from q_0 to q_3, our input is described by the regular expression $1(10^*1 + 01^*0)^*$.
   From q_3, our input will only be accepted if we end up at accepting state q_4, as there are no transitions possible to reach q_1. To end up at q_4, the rest of the input must start with a 1 (so we reach q_4 from q_3) and must be followed by any number of '0's (which will let q_4 transition only to itself). So, this part of the input is described by 10^*.
   Hence, the accepted input in this case will be $\mathbf{1(10^*1 + 01^*0)^*10^*}$.

Combining all these cases, the final regular expression that describes the language accepted by the given DFA is $\mathbf{0 + 1(01^*0 + 10^*1)^*10^*}$.

2 Question 2

Strategy: If L is regular , there exists a DFA which accepts the strings present in the language L. Using that DFA we will construct a NFA/DFA which accepts the string present in the language *oddL* which will prove that the language *oddL* is regular.

Construction of NFA for *oddL*:

Let us denote the DFA corresponding to L as $D = \{Q, q_0, \sum, \delta, A\}$. The NFA accepting the language $oddL$ is $N = \{Q_N, q_0, \sum, \delta_N, A_N\}$ where ,

- Q_N $=Q \cup Q'$ where $Q' = \{q_i^* | q_i^*$is a new state corresponding to every state q_i in Q$\}$. Thus, now there are totally $2|Q|$ states
- $\sum = \sum$ (Alphabet remains the same)
- δ_N is defined as follows:
  $\delta_N(p, a) = q^*$ and $\delta_N(p^*, \epsilon) = q$ where $\delta(p, a) = q$ and p^*,q^* are the new states made by us corresponding to original states p and q.
- A_N=$A \cup A^*$ where $A^* = \{q_i^* | q_i \in A\}$

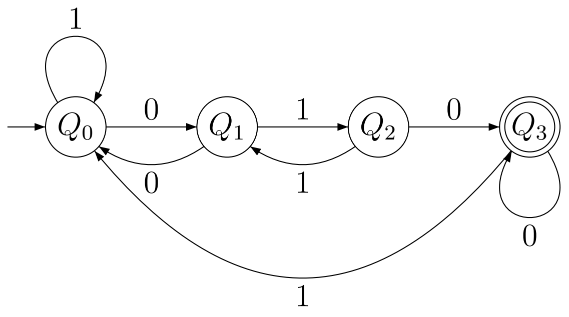
Proof that above NFA accepts *oddL*: By construction of our NFA, as we traverse through the input, we alternatively move between the newly defined states and the original states, following a read of one letter of input (taking us to an original state) by an epsilon transition (taking us to a newly defined state, and skipping over a letter of input that would have been needed for acceptance in the original DFA). This lets us skip past the even position alphabets and still let the input be accepted.

We will consider the following two mutually exclusive and exhaustive cases:

- **When length of input is even:** In this case, we eventually reach an original final state.
- **When length of input is odd:** In this case, we eventually reach a newly defined final state corresponding to an original final state.

3 Question 3

The minimum-state finite automaton corresponding to the given DFA is:



Where, $Q_0 = \{q_0\}, Q_1 = \{q_1\}, Q_2 = \{q_2\}, Q_3 = \{q_3\}$

State	0	1
q_0	q_1	q_0
q_1	q_0	q_2
q_2	q_3	q_1
q_3	q_3	q_0

Table 1: Transition table of the DFA.

0			
-	1		
-	-	2	
-	-	-	3

Table 2: Initial table

0			
-	1		
-	-	2	
✓	✓	✓	3

Table 3: 1st iteration of algorithm

0			
-	1		
✓	✓	2	
✓	✓	✓	3

Table 4: 2nd iteration of algorithm

0			
✓	1		
✓	✓	2	
✓	✓	✓	3

Table 5: 3rd iteration of algorithm

We first removed all the non-reachable states(q_4, q_5, q_6, q_7) from the DFA. q_7 is non-reachable as there are no incoming transitions to it and it is a non-start state. While q_4, q_5 and q_6 can all reach each other, we observe that we cannot reach any of these states from another state (no outgoing transitions from any other state to either q_4, q_5 or q_6). Since none of these states are the starting state, it is impossible to reach any of them.
We have used the *Minimization Algorithm.* [1]
The algorithm will stop at the 3rd iteration since the table after the 3rd iteration is completely filled with checkmarks. The final table tells us that there is no equivalence relation. Therefore, the minimal DFA will have states defined by these equivalence classes: $Q_0 = \{q_0\}, Q_1 = \{q_1\}, Q_2 = \{q_2\}, Q_3 = \{q_3\}$, where Q_0 is the initial state and Q_3 is the final state.

4 Question 4

Strategy:

Since L_1 and L_2 are regular , it implies that there exist DFAs D_{L_1} and D_{L_2} corresponding to L_1 and L_2 respectively. Using these DFAs , we will construct a NFA that will accept only the strings present in Mix(L_1, L_2). By equivalence of DFA and NFA, we will argue that there exists a DFA that accepts the language Mix(L_1, L_2). Hence Mix(L_1, L_2) will be proved regular.

Construction of NFA:

Let us denote the DFA corresponding to the language L_1 as $D_{L_1} = \{Q_{L_1}, q_{L_1}, \sum, \delta_{L_1}, F_{L_1}\}$ and the DFA corresponding to the language L_2 as $D_{L_2} = \{Q_{L_2}, q_{L_2}, \sum, \delta_{L_2}, F_{L_2}\}$. We construct a NFA $N_{mix} = \{Q_{mix}, q_{mix}, \sum, \delta_{mix}, F_{mix}\}$, where ,

- $Q_{mix} = \{\{s, p\} | s \in Q_{L_1}$ and $p \in Q_{L_2}\}$
  Thus, $|Q_{mix}| = |Q_{L_1}| \cdot |Q_{L_2}|$
- $q_{mix} = \{q_{L_1}, q_{L_2}\}$
- δ_{mix} is defined as follows :
  $\delta_{mix}(\{s, p\}, a) = \{\{s', p\}, \{s, p'\}\}$ where $\delta_{L_1}(s, a) = s'$ and $\delta_{L_2}(p, a) = p'$
- $F_{mix} = \{j, k\}$ where $j \in F_{L_1}, k \in F_{L_2}$

Claim:

The above NFA accepts Mix(L_1, L_2).

Proof:

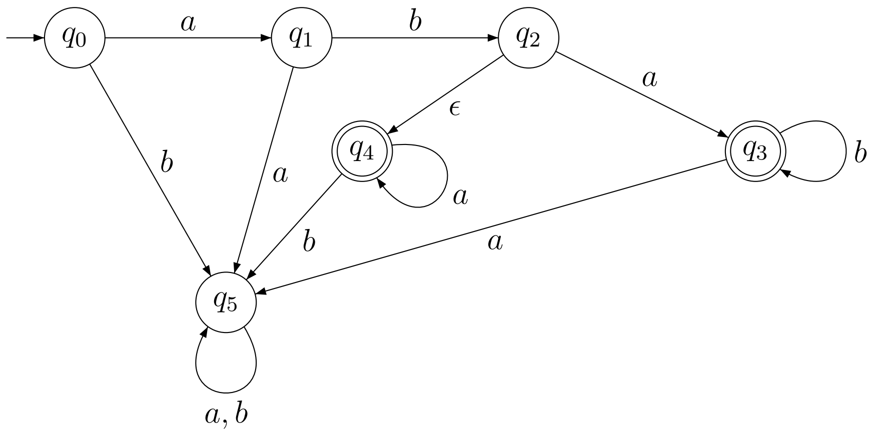
Consider any string belonging to Mix(L_1, L_2). Let this string be $x_1y_1x_2y_2....x_ky_k$, where $x_1x_2x_3.....x_k \in L_1$ and $y_1y_2y_3....y_k \in L_2$, each $x_i, y_i \in \sum^*$. We take this as input to N_{mix}.
Define states s_i (not necessarily distinct) as the state DFA D_{L_1} reaches after reading input $x_1....x_i$. Similarly, define states p_i as the state DFA D_{L_2} reaches after reading input $y_1....y_i$.
We start at state $\{q_{L_1}, q_{L_2}\}$. Clearly, by definition of δ_{mix}, we can reach state $\{s_1, q_{L_2}\}$ on taking input x_1, by simply choosing the $\{s, p\} \to \{s', p\}$ transitions as defined in δ_{mix} while taking input x_1. Now, on choosing the $\{s, p\} \to \{s, p'\}$ transitions as defined in δ_{mix} while taking input y_1, N_{mix} can reach state $\{s_1, p_1\}$ after input x_1y_1. Similarly, on choosing the $\{s, p\} \to \{s', p\}$ transitions while taking input x_2 N_{mix} can reach $\{s_2, p_1\}$ after input $x_1y_1x_2$.
We can proceed like this till x_ky_k and we will eventually reach $\{s_k, p_k\}$. Since $x_1...x_k$ and $y_1...y_k$ are accepted by D_{L_1} and D_{L_2} respectively, we know that $s_k \in F_{L_1}$ and $p_k \in F_{L_2}$. Thus, $\{s_k, p_k\} \in F_{mix}$.

Now, let us consider any string accepted by N_{mix}. The accepted path would also follow the sequence of some $\{s, p\} \to \{s', p\}$ transitions followed by some $\{s, p\} \to \{s, p'\}$ transitions. Each of these sequences can be interpreted as some x_iy_i, corresponding to the states of transition in N_{mix}, D_{L_1} and D_{L_2}. Thus, any string accepted by N_{mix} belongs to Mix(L_1, L_2).

Hence, $\boldsymbol{L(N_{mix})}$ =**Mix**$\boldsymbol{(L_1, L_2)}$, and so we can say that $\mathrm{Mix}(L_1, L_2)$ is regular.

5 Question 5

NFA for the given language:



- q_0 is the starting state of the NFA. States q_1 and q_2 ensure that the first two letters of the input are ab.
- There are two accepting states q_3 and q_4. q_3 accepts all strings of the form $abab^n$, while q_4 accepts all strings of the form aba^n.
- State q_5 is reached when we read the letter of the string that tells us that the string cannot be accepted (in that path); it is possible for a string of the form $abab^n$ to end up here but it will also end up in state q_3. Once a string reaches here, it doesn't leave this state.

6 Question 6

Claim: This language is not a regular language.

Proof: We will prove this by using contra-positive of Pumping Lemma.

Statement: Consider the set of strings accepted by this language is denoted by A.Then we have:

$\forall n > 0$, $\exists x$ such that $|x| > n$ and $\forall u, v, w$ we have $x = uvw, |uv| \leq n, |v| \geq 1$, the following statement is true:

$$uvw \in A \text{ and } \exists i \geq 0 \text{ such that } uv^iw \notin A \Rightarrow \text{set A is not Regular}$$

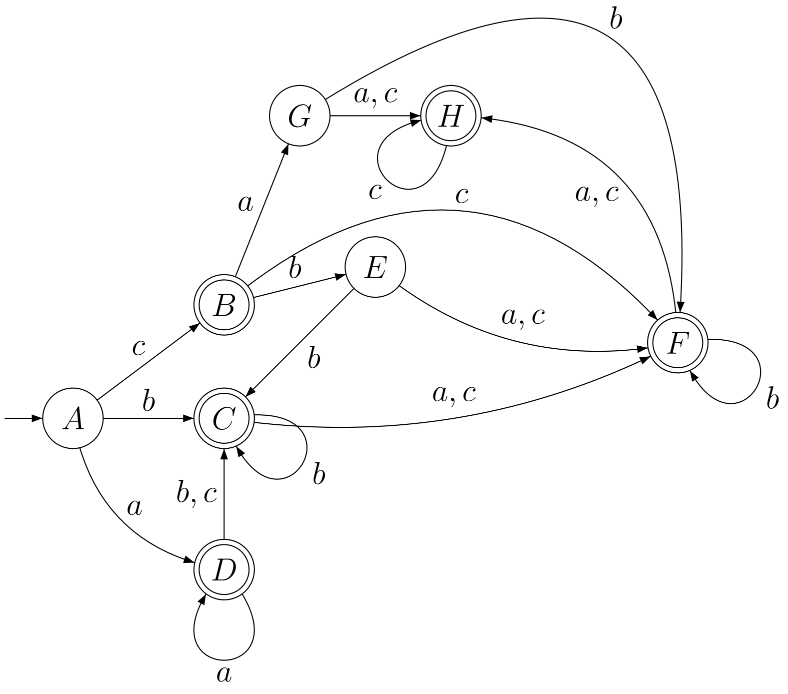
As mentioned in lecture , we will play the game against adversary:

- Adversary chooses $n > 0$
- We choose x such that $|x| > n$ .Let us say x is (000.....n times)(111......n times).Clearly length of x is $2n > n$. Also observe x belongs to set A since the difference between the number of 1's and number of 0's is 0.
- Adversary chooses u, v, w such that $x = uvw$, $|uv| \leq n$, $|v| \geq 1$. Let us say adversary chose $u = 000.....$ (n-1) times, $v = 0$, w=111.....n times
- Observe that $uvw \in A$. We have to prove that $\exists i \geq 0$ such that $uv^iw \notin A$. Consider $i >= 3$ This means we have added more than two zeros to the string keeping the number of ones constant. Hence the difference between number of 1's and number of 0's is greater than equal to 2.
  For example : i=4:
  uv^iw=(000.... (n-1)times)(0000)(111.....(n times))
  =(000.....((n+3) times))(111......(n times))
  Difference between number of ones and number of zeros for this string is 3. Hence this string does not belong to A.

$\Rightarrow$ **Set A is not regular.**

7 Question 7

To convert this NFA with set of states Q to its equivalent DFA, we take the set of states of the DFA to be the powerset of Q, that is, $\mathcal{P}(Q)$. Based on the given NFA, some of the states can be eliminated in the final DFA.
Since the starting state of the NFA, q_0, has an ϵ transition to state q_2, the starting state of our DFA becomes $\{q_0, q_2\}$. Using the definition of δ_D as defined in the lecture, we can construct a DFA as shown.
The final states are all the states which have the final state of the NFA (that is, q_3), as a member of them.



Labels of States: A: $\{q_0, q_2\}$, B: $\{q_1, q_3\}$, C: $\{q_1, q_2, q_3\}$, D: $\{q_0, q_1, q_2, q_3\}$, E: $\{q_1, q_2\}$, F: $\{q_2, q_3\}$, G: $\{q_2\}$, H: $\{q_3\}$

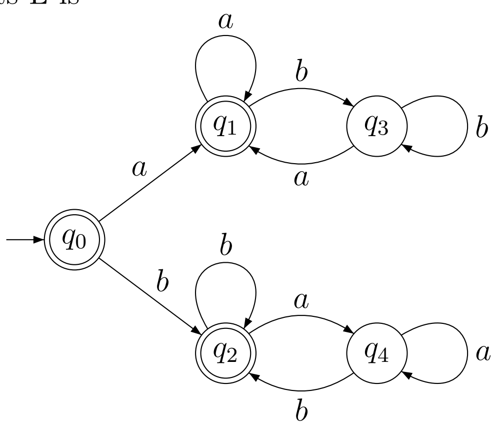
8 Question 8

8.1 8(a)

The Regular Expression whose language is L = $\{w \in \{a, b\}^* | w$ contains an equal number of occurrences of ab and $ba\}$ is $\Rightarrow$ $\boxed{\varepsilon + a(a+b^+a)^* + b(b + a^+b)^*}$

8.2 8(b)

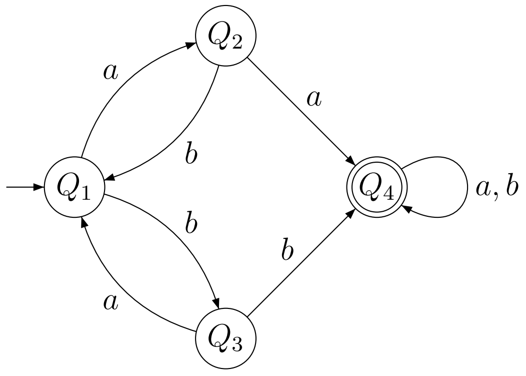
The DFA which accepts L is



- State q_0 is the initial state and the states q_0, q_1, q_2 are the final states.
- State q_0 accepts an empty string (ϵ).
- State q_1 accepts strings of type $a(a + b^+a)^*$ which means strings that start with a and have equal number ab and ba.
- State q_2 accepts strings of type $b(b + a^+b)^*$ which means strings that start with b and have equal number ba and ab.

9 Question 9

The minimal DFA for the given DFA is:



Where, $Q_1 = \{q_1\}, Q_2 = \{q_2\}, Q_3 = \{q_3\}, Q_4 = \{q_4, q_5, q_6\}$

State	a	b
q_1	q_2	q_3
q_2	q_4	q_1
q_3	q_1	q_5
q_4	q_6	q_4
q_5	q_5	q_6
q_6	q_6	q_6

Table 6: Transition table of the original DFA.

1					
-	2				
-	-	3			
-	-	-	4		
-	-	-	-	5	
-	-	-	-	-	6

Table 7: Initial table

1					
-	2				
-	-	3			
✓	✓	✓	4		
✓	✓	✓	-	5	
✓	✓	✓	-	-	6

Table 8: 1st iteration of algorithm

1					
✓	2				
✓	✓	3			
✓	✓	✓	4		
✓	✓	✓	-	5	
✓	✓	✓	-	-	6

Table 9: 2nd iteration of algorithm

1					
✓	2				
✓	✓	3			
✓	✓	✓	4		
✓	✓	✓	-	5	
✓	✓	✓	-	-	6

Table 10: 3rd iteration of algorithm

We have used the *Minimization Algorithm.* [1]. The algorithm will stop at the 3rd iteration since the table after the 2nd and 3rd iterations is the same. The final table tells us that $4 \approx 5$, $5 \approx 6$ and $6 \approx 4$. Therefore, the minimal DFA will have states defined by these equivalence classes:
$Q_1 = \{q_1\}, Q_2 = \{q_2\}, Q_3 = \{q_3\}, Q_4 = \{q_4, q_5, q_6\}$, where Q_1 is the initial state and Q_4 is the final state.

10 Question 10

Claim: The regular expression whose language is L is of the form $(a+b)^+(aa+bb)(a+b)^+$.

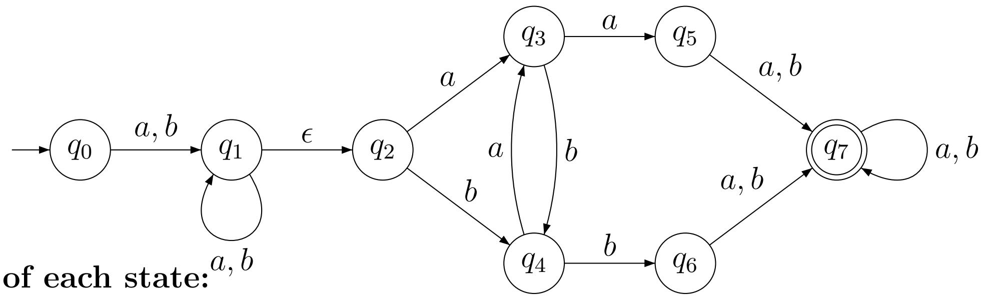
Proof:

Since, we are concatenating v and v^r, It is easy to observe that the string should have at least one occurence of aa or bb if the last character is of v is a or b respectively. Also since , we have to chose non empty strings u, v, w, we make the following choice:

- **Case 1:** $v = a, v^r = a$, u and w can be any non empty strings and hence of the form $(a + b)^+$
- **Case 2:** $v = b, v^r = b$, u and w can be any non empty strings and hence of the form $(a + b)^+$

Note that any string of the language L falls in one of the two cases, and hence we just have to construct a NFA corresponding to the union of the two cases.

Construction of equivalent NFA:



Function of each state:

State	Function
q_0	Start state
q_1	Ensuring u is a non empty string
q_2	Going to v through ϵ transition
q_3	If $v = a$
q_4	If $v = b$
q_5	If $v = a$, $v^r = a$
q_6	If $v = b$, $v^r = b$
q_7	Ensuring w is a non empty string and accepting the final string

The transitions between q_3 and q_4 ensures aa or bb must be present for the string to reach q_7 (accepting state).

11 Question 11

Claim: Only 1. and 2. are equivalent regular expressions.

Proof: There are two parts of this proof:

- Proving first and second regular expressions are equivalent
- Proving first/second are not equivalent to third

Let us first prove that first expression is not equivalent to third expression. To show this, we can just show a string that is accepted by the language made by the first regular expression and is rejected by the language made by the third regular expression.

Consider the string $y = b$:

- b is accepted by the first regular expression. Consider $(a + ba)$ 0 times, you have $b + \epsilon$ which accepts b
- The third expression does not accept y as it needs either a or ba as the prefix.

Proving Equivalence between first and second regular expressions:

Let's analyze $(a + ba)^*$:

- This expression means that we can have either a or ba any number of times and in any sequence
- Thus, we can have any (may be 0) number of $a's$ followed by any (may be 0) number of ba strings, further followed by any number of $a's$ and so on.

- Mathematically we can divide the expression into blocks of $a's$ and ba strings. So it looks like:
  $(aa...\ n_1$ times$)((ba)(ba)(ba)....\ m_1$ times$)(aa...\ n_2$ times$)((ba)(ba)(ba).....\ m_2$ times$)$ and so on, till some n_k and m_k, where all m_i and n_i are whole numbers.
- Creating blocks of all m_i and n_i we have
  $[(aa...\ n_1$ times$)((ba)(ba)(ba)....\ m_1$ times$)][(aa...\ n_2$ times$)((ba)(ba)(ba).....\ m_2$ times$)]$ and so on.
- Each of these blocks represents a member of the regular expression $a^*(ba)^*$
- Since the number of blocks is a whole number, we can say that all the blocks combined are a member of the regular expression $(a^*(ba)^*)^*$
- Since every member of the regular expression $(a^*(ba)^*)^*$ can be represented like this, we can say that these regular expressions are equivalent, that is,
  $(a^*(ba)^*)^* \equiv (a+ba)^*$
- Thus, clearly, $(a^*(ba)^*)^*(b+\epsilon) \equiv (a+ba)^*(b+\epsilon)$
- Let us take any member of a^*. This will contain only some number of a's. Clearly, this will also belong to $a^*(ba)^*$, and thus also to $(a^*(ba)^*)^*$. Similarly, any member of $(ba)^*$ will also belong to $(a^*(ba)^*)^*$.
- So, we can say that $(a^*(ba)^*)^* \equiv (a^*(ba)^*)^* + a^* + (ba)^*$
- Hence , $(a+ba)^*(b+\epsilon)$ is equivalent to $(a^*(ba)^*)^*(b+\epsilon) + a^*(b+\epsilon) + (ba)^*(b+\epsilon)$.

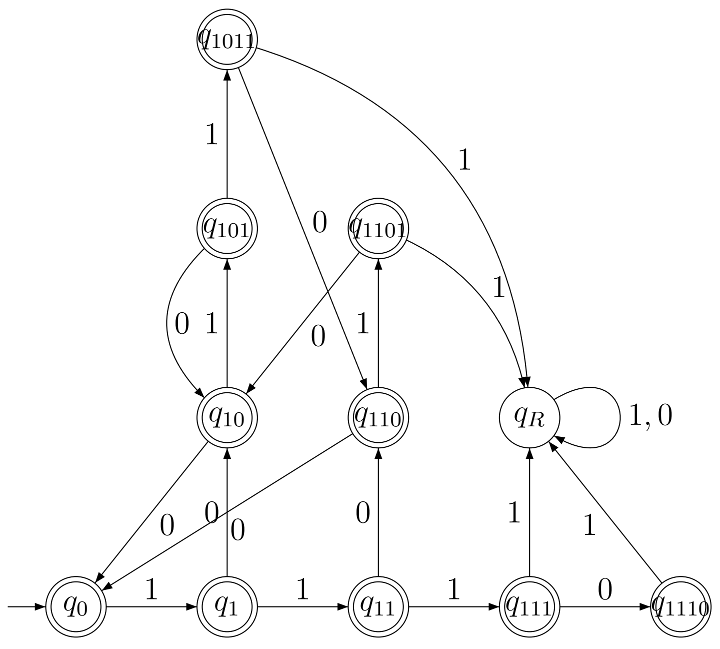
Since first is equivalent to second, and first is not equivalent to third, so second is also not equivalent to third.

$\Rightarrow$ **Only first and second expressions are equivalent**

12 Question 12

12.1 (a)

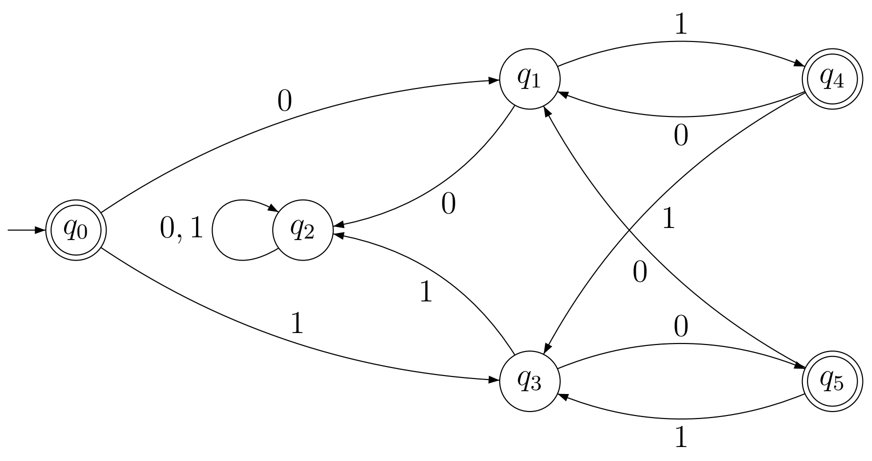
The DFA which accepts such L is:



- This DFA assumes that we check all possible blocks of length 5 present in the input.
- We assume that trivial inputs of length less than 5 are accepted.
- The DFA searches for a block of 5 that has less than two 0's, and on finding such a block sends the input to a reject state (q_R) which has no outgoing transitions, forcing the input to be rejected.
- The subscript of the label of the state (x in q_x) indicates the letters of the block we are currently checking. If we can infer that there will be a block of 5 with less than two 0's, we send the input to the reject state. If we do not find a block like this after having taken all the input, we can say that the input will be accepted, thus all other states are accepting.

12.2 (b)

The DFA which accepts such L is:



- The regular expression which defines this language is $(01+10)^*$.
- State q_0 is the initial state and the states q_0, q_4, q_5 are the final states.
- State q_0 accepts an empty string (ϵ).
- State q_4 accepts strings that end with 01, and satisfy the required property.
- State q_5 accepts strings that end with 10, and satisfy the required property.
- State q_2 rejects all those strings which makes one of the prefix to have more than one 0 than 1 or more than one 1 than 0.

13 Question 13

Strategy:

Since L is regular , there exists a DFA whose set of accepting strings is the language L. Using that DFA , we will construct a DFA/NFA that will accept the strings in the language $L_{\frac{1}{2}}$ and hence $L_{\frac{1}{2}}$ will be proved regular.

Construction of DFA:

Let us denote the DFA corresponding to the language L as $D = \{Q, q_0, \sum, \delta, F\}$.

The DFA corresponding to $L_{\frac{1}{2}}$ is denoted by $N = \{Q_N, s_N, \sum, \delta_N, F_N\}$ where :

$$\text{The states of } Q_N \text{ are of the form } (q, M) \text{ where } q \in Q \text{ and } M \subseteq Q$$
$$s_N \text{ is the starting state } (q_0, F)$$
$$\textstyle\sum \text{ remains the same}$$
$$\delta_N \text{ is defined as follows:}$$
$$\delta_N((q, M), a) = (\delta(q, a), M') \text{ where } M' = \{t | \exists b \in \textstyle\sum \text{ s.t } \exists p \in M \text{ s.t } \delta(t, b) = p : p, t \in Q\}$$
$$F_N = \{(q, S) | q \in S \ \text{ and } S \subseteq Q\}$$

Proof:

Claim:

Let us consider we reach a state (q_i, M_i) after reading i alphabets in the input. Claim is:

- q_i is the state that the DFA D reaches on reading i alphabets of the input.
- The states in the set M_i have a path to the accepting states of the DFA D such that the length of the path is exactly i.

Proof of Claim:

- The first point is evident by the transition table itself.
- Initially M_0 is F that is the set of final states of D. Now, whenever we read an alphabet we change M such that we require one more input to reach the final state from the elements of M. That is we are going in the backward direction in the DFA starting from the set of final states.
  Formal Proof: Induction on Length of input x

Base case : $|x| = 0$
$M_0 = F$ implies that the distance of the states in M_0 to the final states is 0.
Induction Hypothesis:The states in the set M_i have a path to the accepting states of the DFA D such that the length of the path is exactly i.
Induction Step: $\delta_N((q_i, M_i), a_{i+1}) = (\delta(q_i, a), M_{i+1}$ where $M_{i+1} = \{t|\exists b \in \sum$ s.t $\exists p \in M_i$ s.t $\delta(t, b) = p : p, t \in Q\}$
Path length from the states in M_{i+1} to the final states of D = Path length(t,p)+Path length from the states of M_i to the final states of D Path length(M_{i+1})=1+Path length(M_i) since there is a transition from t to p.
Hence proved by induction

Now , using the claim it is easy to observe that the set of final states is $F_N = \{(q, M)|q \in M\}$

As after reading the input, our NFA reaches the state (q, S) where q is the same state that the DFA D will reach after reading the same input. As q belongs to S , there exists a path from q to the final accepting state of D of the same length as that of the input string . Call the string corresponding to this path as y. Clearly xy belongs to F and $|x| = |y|$ hence $x \in L_{\frac{1}{2}}$.
Thus, there is an NFA that accepts $L_{\frac{1}{2}}$.
Thus, $L_{\frac{1}{2}}$ is a regular language.

14 Question 14

General Strategy: Since L is regular , there exists a DFA F whose set of accepting strings forms the language L. With the help of this DFA We will construct a NFA that will accept the reverse of the strings in a language L. Then , by arguing the equivalence of NFA and DFA , we can say that there exists a DFA that accepts L'. Hence L' is regular.

Construction of NFA:

Consider the DFA corresponding to L is $F = \{Q, q_0, \sum, \delta, A\}$. We construct the following NFA $F' = \{Q', t_0, \sum, \delta', A'\}$ defined as follows:

$$Q' = Q \cup \{t_0\} \text{ , where } t_0 \text{ is the new start state we constructed}$$
$$t_0 = t_0 \text{ , the new start state is } t_0$$
$$\sum = \sum \text{, alphabet remains the same}$$
$$A' = \{q_0\} \text{ , the start state of } F \text{ becomes the final state}$$
$$\delta'(t_0, \epsilon) = A \text{ , making epsilon transition to the set of final states of the DFA of L}$$

$$\forall p \in Q, \delta'(p, a) = \{q \in Q|\delta(q, a) = p\} \text{ , reversing the transitions}$$

Description:

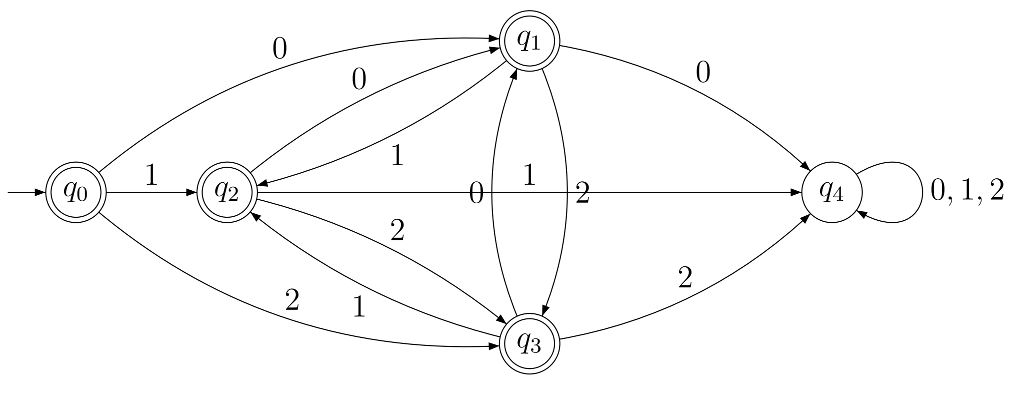
We added a new state to the DFA which is our new start state. We linked this new state to the former final states through epsilon transition . We made the former start state as our new final state.

Once start and final states are decided, we just reverse the direction of the edges so that the new DFA will accept reverse of strings present in L.

Consider the string $s = a_1a_2a_3a_4.....a_n \in L$. Reversing the string gives us $s' = a_na_{n-1}......a_1$. We shall prove that if F accepts s, $F^{'}$ must accept s'. Let F accept s, and end up in some final state q_n. So, in our NFA $F^{'}$, we can reach q_n without any input upon taking an ϵ transition from t_0 to q_n. This transition exists as $\delta'(t_0, \epsilon) = A$. Now, let any transition in F while running input s be as follows:
$\delta(q_{i-1}, a_i) = q_i$ for some states q_i all of which are not necessarily distinct.
By definition of δ', we can say that $\delta'(q_i, a_i) = q_{i-1}$.
Thus, on input a_n, we go to state q_{n-1}. Then, on taking input a_{n-1}, we reach q_{n-2}. Similarly, we can keep traversing backwards on taking s' as input and eventually upon taking a_1 as input we will reach q_0 , which is a final state for F'. So, s' is accepted by NFA F', for all s in L.
$\Rightarrow$ All strings in L' are accepted by an NFA.
$\Rightarrow$ L' is regular.

15 Question 15

The DFA which accepts such L is:



- State q_0 is the initial state and the states q_0, q_1, q_2, q_3 are the final states.
- State q_0 accepts an empty string (ϵ).
- State q_1 accepts strings that end with 0 and doesn't contain any consecutive $0, 1, 2$. Any extension to this string with a 0 will lead to a rejecting string since there can't be consecutive 0s in the string.
- State q_2 accepts strings that end with 1 and doesn't contain any consecutive $0, 1, 2$. Any extension to this string with a 1 will lead to a rejecting string since there can't be consecutive 1s in the string.

- State q_3 accepts strings that end with 2 and doesn't contain any consecutive $0, 1, 2$. Any extension to this string with a 2 will lead to a rejecting string since there can't be consecutive 2s in the string.
- State q_4 rejects all those strings which has consecutive $0, 1, 2$.

16 Question 16

16.1 Part 1

We seek to prove that if $L \subseteq \sum^*$ is regular, then $h(L)$ is also regular.
Let r be the regular expression corresponding to the regular language L. We now modify the regular expression r by replacing each letter of r by its image in Δ. That is, we replace all letters $a \in \sum$ present in r, with their image $h(a) \in \Delta$. Let the expression formed this way be r'. We can similarly define $x^{'}$ for any regular expression x defined over $\sum$.
We now claim that the language of this expression, that is, $L(r')$, is $h(L)$.
Lemma: $h(x)h(y) = h(xy) \forall x, y \in \sum^*$
We show this by induction on length of y.
Base Cases:

- $h(x)h(\epsilon) = h(x\epsilon) = h(x) \forall x \in \sum^*$
- $h(x)h(y) = h(xy) \forall y \in \sum, x \in \sum^*$

Inductive Case: Let $h(x)h(y) = h(xy) \forall x, y \in \sum^*, |y| \le n$
$h(x)h(ya) = h(x)h(y)h(a) \forall x, y \in \sum^*, a \in \sum$
$\Rightarrow h(x)h(ya) = h(xy)h(a) \forall x, y \in \sum^*, a \in \sum$
$\Rightarrow h(x)h(ya) = h(xya) \forall x, y \in \sum^*, a \in \sum$ (as $xy \in \sum^*$)
$\Rightarrow h(x)h(y) = h(xy) \forall x, y \in \sum^*, |y| \le n + 1$

Thus, we can say that $h(x)h(y) = h(xy) \forall x, y \in \sum^*$.

Now, let us look at $L(r^{'})$. We prove by induction that for any regular expression x defined over $\sum$, $L(x^{'}) = h(L(x))$.
Base Cases:

- Let $h(a) = a^{'}$ for some $a' \in \Delta$
  Then, $L(a^{'}) = L(h(a)) = \{h(a)\}$
  We also know that $L(a) = \{a\}$ and $h(\{a\}) = \{h(a)\}$ (by our definition)
  Thus, $L(h(a)) = h(L(a))$
- $L(\epsilon^{'}) = h(L(\epsilon)) = \epsilon$

Induction Cases: Consider two regular expressions x and y where x, y are defined over $\sum$.

- $L((x + y)^{'}) = L(x' + y') = h(L(x)) \cup h(L(y))$ (By definition)
  $\Rightarrow L((x + y)^{'}) = h(L(x)) \cup h(L(y))$ (Induction Hypothesis)
  $\Rightarrow L((x + y)^{'}) = h(L(x) \cup L(y)) = h(L(x + y))$ (By Lemma)

- Similarly, we can say that $L((x \cdot y)') = h(L(x \cdot y))$
- $L(x^{*'}) = L(x^{'*})$ (By definition)
  $\Rightarrow L(x^{*'}) = L((x')^*) = h(L(x))^*$ (Induction Hypothesis)
  Since we know that $h(x)$ is conserved under set unions by the lemma, we can say that $h(L(x))^* = h(L(x)^*)$
  $h(L(x)^*) = h(L(x^*)) \Rightarrow L(x^{*'}) = h(L(x^*))$

Thus, by induction, we can say that $L(x') = h(L(x))$ for any regular expression x defined over $\sum$. Thus, $\boldsymbol{L(r') = h(L)}$.

16.2 Part 2

Let $F = \{Q, q_0, \Delta, \delta, A\}$ be a DFA that accepts L.
We modify our definition of δ so that it can now accommodate inputs in Δ^*, not only those in Δ. We do this by adding two new definitions in δ:

- $\delta(q, \epsilon) = q$
- $\delta(q, xa) = \delta(\delta(q, x), a)$ for $x \in \Delta^*$ and $a \in \Delta$

We create a new DFA $F' = \{Q, q_0, \sum, \delta', A\}$ for $h^{-1}(L)$.
Define δ' as follows:
$\delta'(q, x) = \delta(q, h(x))$, where x $\in \sum^*$.

$$\delta'(q, xa) = \delta'(\delta'(q, x), a)$$

$$\Rightarrow \delta'(q, xa) = \delta(\delta(q, h(x)), h(a))$$

$$\Rightarrow \delta'(q, xa) = \delta(q, h(x)h(a))$$

$$\Rightarrow \delta'(q, xa) = \delta(q, h(xa))$$

Thus, by induction, we can say that $\delta'(q, s) = \delta(q, h(s)) \forall s \in \sum$

Let F' accept some language L'. Then, we know that for all $x \in L', \delta'(q, x) = q_A$ for some $q_A \in A$.
$\iff \delta(q, h(x)) = q_A$
$\iff h(x)$ is accepted by F (as $q_A \in A$) .
$\iff h(x) \in L$.
$\iff x \in h^{-1}(L)$. Thus, F', a DFA, accepts $h^{-1}(L)$.
Thus, $h^{-1}(L)$ is regular.

References

[1] D. C. Kozen. *Automata and Computability.* springer, 1997, pg. 84.