CS340 Assignment - 1

Church of Turing

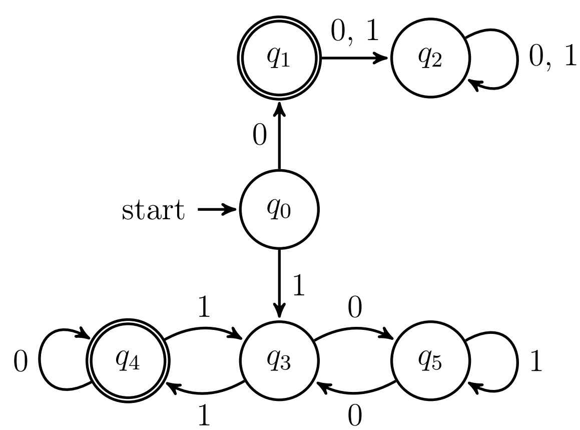
Samarth Arora 200849 | Soham Samaddar 200990 | Aditya Tanwar 200057

September 2022

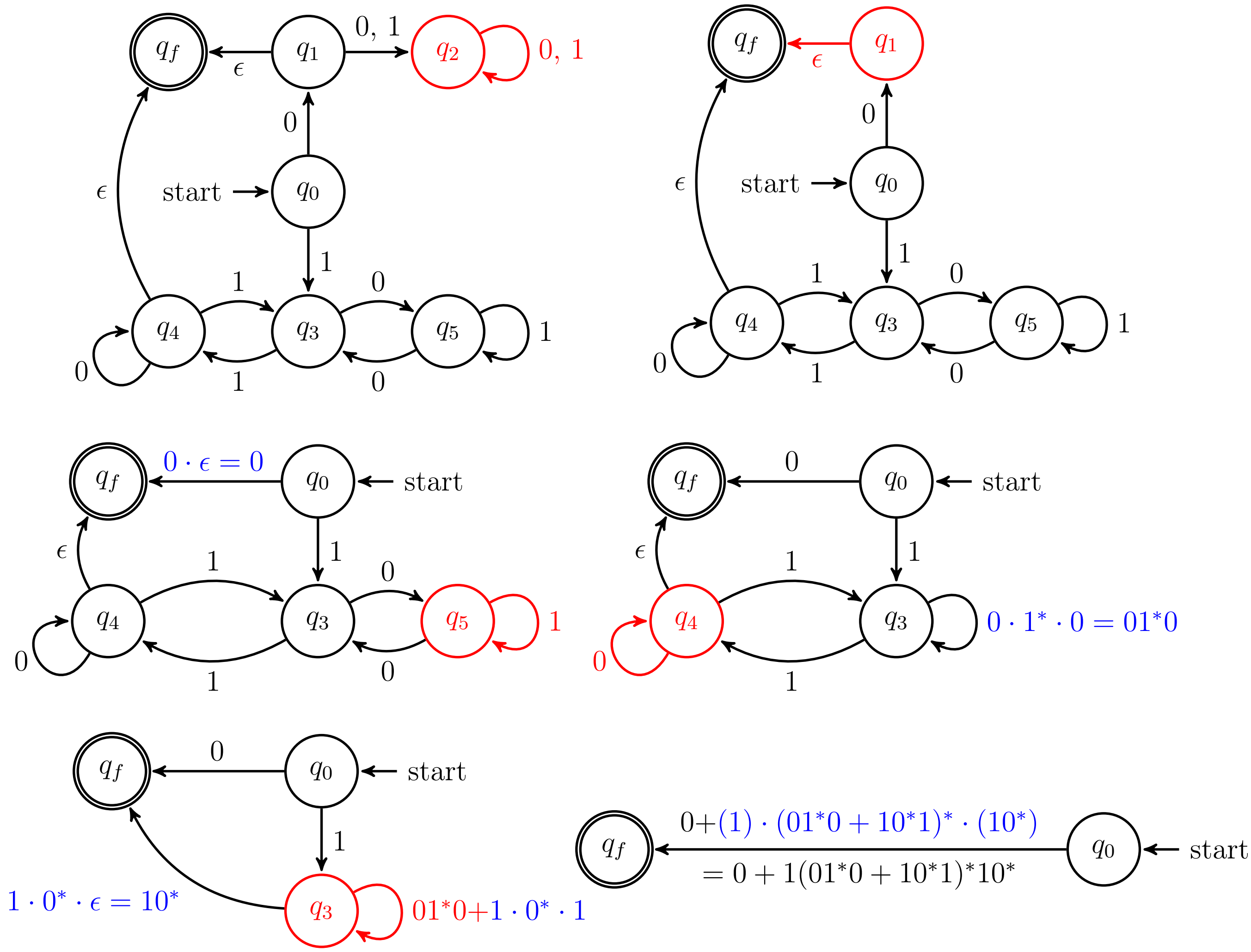
Unless mentioned otherwise, we will use standard notations related to finite automata.

$$
\begin{aligned}
\Sigma &: \text{Alphabet}\\
Q &: \text{Set of states}\\
s &: \text{Starting state}\\
\delta &: \text{Transition function with input from } \Sigma\\
\hat{\delta} &: \text{Transition function with input from } \Sigma^*\\
F &: \text{Set of final states}\\
L(r) &: \text{Regular set corresponding to a regular expression } r
\end{aligned}
$$

Q1. Describe the language accepted by the following DFA:



Solution: We start by converting the given NFA to an equivalent GNFA with just two states, and inferring the regular expression. At each step, the state about to be removed is colored red and the new transitions added as a consequence of the previous removal are colored blue. We make transitions leading to rejection implicit. Begin by introducing ϵ-transitions to a final state q_f:



Thus, we conclude that the regular expression for the set of strings accepted by the automaton is

$$r = 0 + 1(01^*0 + 10^*1)^*10^*$$

Now let us define the language that accepts this sort of expression. Define a function $f : \Sigma^* \to \mathbb{N}$. Call $f(s)$ as the *score*(t) of a string s.

f is defined as follows:

Traverse the string bit wise

$t \longleftarrow$ `initialised to 0.`

`if` $(t == 0)$:

  `if (current bit ==` 1 `)` $t \longleftarrow 2$
  `else` $t \longleftarrow -1$

`else if` $(t < 0) : t \longleftarrow t - 1$

`else if` $(t == 1)$:

```
    if (current bit == 0 ) t ⟵ t + 1
else if (t == 2) :
    if (current bit == 1 ) t ⟵ t + 1
    else t ⟵ t − 1
else :
    if (current bit == 1 ) t ⟵ t − 1
```

When the string ends, we accept strings of scores -1 and 3 only. Therefore the language accepted by the DFA are those strings that have score -1 or 3. Finally, the language may be defined as:

$$L(r) = \{s \mid s \in \Sigma^*,\ f(s) \in \{-1, 3\}\}$$

■

Q2. Let w be a string over some alphabet Σ. $Odd(w)$ refers to the string obtained by deleting symbols at all even positions of w. Example, if $w = a_1a_2a_3 \dots a_n$, then odd(w) $= a_1a_3a_5 \dots a_m$, where m is n or $n-1$ when m is odd or even respectively. For a language $L \subseteq \Sigma^*$, define $oddL = \{odd(w) \mid w \in L\}$. Prove that if L is regular, then $oddL$ is regular too.

Solution: Let $\mathcal{F} = (Q, \Sigma, s, \delta, F)$ be the DFA which accepts a regular language L. We prove the regularity of $oddL$ by constructing a DFA that accepts it. Let $\mathcal{F}' = (Q', \Sigma, s', \delta', F')$ be an automata that accepts $oddL$. Essentially, for an input given to $\mathcal{F}'$, it tries to fill the *"blanks"*, and checks if it is possible to construct any string which is in L. By filling *"blanks"*, we refer to inserting letters between two consecutive letters in the input w' to $\mathcal{F}'$ and possibly a letter after the last letter in w' (corresponds to a string of even length if it is inserted, and odd otherwise).
The elements of $\mathcal{F}'$ are elaborated below:

Q' $:= \langle 2^Q,\ 2^Q \rangle$
Q' is a *2-tuple* that holds two possibilities, the first corresponding to states corresponding to strings of odd lengths, and the second corresponding to strings of even lengths. We expand any state q^i as $\langle q^i_o,\ q^i_e \rangle$.

s' $:= \langle \{s\},\ \{s\} \rangle$

δ' : The transition function is defined as follows:

$$\delta'(q = \langle q_o,\ q_e \rangle,\ a) = \langle q'_o,\ q'_e \rangle \qquad \text{where}$$
$$q'_o := \{p' \mid p' = \delta(p,\ a) \text{ for some } p \in q_e\}$$
$$q'_e := \{p' \mid p' = \delta(\delta(p,\ a),\ b) \text{ for some } p \in q_e \text{ and } b \in \Sigma\}$$

In words, q'_o captures all the possible states if the input string upto a corresponded to a string of odd length in L, while q'_e contains all the possible states corresponding to a string of even length.

$F' := \{q = \langle q_o, q_e\rangle \mid (q_o \cup q_e) \cap F \neq \varnothing\}$
In words, F' contains states which have at least one accepting state of F either in q_o or q_e, i.e., if it was possible to construct a string from the given input of odd length accepted by $\mathcal{F}$ (corresponds to q_o) or a string of even length accepted by $\mathcal{F}$ (corresponds to q_e).

We now go on to prove that $\mathcal{F}'$ accepts odd strings of all strings accepted by $\mathcal{F}$ and that any string accepted by $\mathcal{F}$ has its odd string accepted in $\mathcal{F}'$ too.

Lemma. $\mathcal{F}'$ accepts odd strings of all strings accepted by $\mathcal{F}$, i.e., w' accepted by $\mathcal{F}' \Rightarrow\ \exists\, w$ accepted by $\mathcal{F}$ such that $\text{odd}(w) = w'$.
Proof. We give a recursive algorithm which constructs such a w by essentially backtracking the steps of $\mathcal{F}'$. Let the function be called f which takes three inputs, a string $w' = a_1 \ldots a_k$, a state from q' from Q' and a state q from Q. It is mandatory that $q \in q'_e$.
This is how the function operates:

$$f(a_1 \ldots a_k,\, q',\, q) = \begin{cases} f(a_1 \ldots a_{k-2},\, p',\, p) \cdot a_k a_{k+1} & q \in q'_e \\ f(a_1 \ldots a_{k-2},\, p',\, p) \cdot a_k & q \in q'_o \end{cases}$$
$$f(\epsilon,\, s',\, s) = \epsilon$$

We now explain how to find p', p, a_{k+1} and what the initial arguments are in f to compute w:

- p' is such that $\delta'(p', a_k) = q'$
  It can be found deterministically since a_k is known and $\mathcal{F}'$ itself is a DFA.
- p and a_{k+1} are found symbiotically by ensuring that $p \in p'_e$ and $\delta(\delta(p, a_k), a_{k+1}) = q$ if $q \in q'_e$. If $q \in q'_o$ then we simply pick p to be any state in p'_e.
- Initially, we send $a_1 \ldots a_k$ as the string to the function. q' is the state that $\mathcal{F}'$ ends up in on reading the complete input, and $q \in (q'_e \cup q'_o) \cap F'$, i.e.,

$$w = f(w',\, \hat{\delta}(s', w'),\, q)$$

Thus, by this construction and definition of $\mathcal{F}'$, we are ensured that $\exists\, w$ for every w' that $\mathcal{F}'$ accepts, such that $odd(w) = w'$ ☐

Lemma. w accepted by $\mathcal{F} \ \Rightarrow\ \text{odd}(w) = w'$ accepted by $\mathcal{F}'$
Proof. Suppose $w = a_1 a_2 \ldots$, then $w' = a_1 a_3 \ldots$
We show by induction that after reading every letter in w' (say upto and including a_k), the current state of $\mathcal{F}'$ contains at least one state which corresponds to $w[1:k]$ and one that corresponds to $w[1:k+1]$ as well (if $k < |w|$).

Base Case: Before $\mathcal{F}'$ has read any letter, it is at the state $s' = \langle s, s\rangle$ which is in line with the state that $\mathcal{F}$ starts with. After reading a_1, it moves to the state $\langle q_o, q_e\rangle$, where $q_o = \delta(s, a_1)$ by definition and q_e contains the state $\delta(\delta(s, a_1), a_2) \in Q$ by definition of δ'. Thus, q_o contains a state corresponding to the string upto and including a_1, and q_e at least contains the state which corresponds to reading $a_1 a_2$ in $\mathcal{F}$. So, it holds for the base cases.

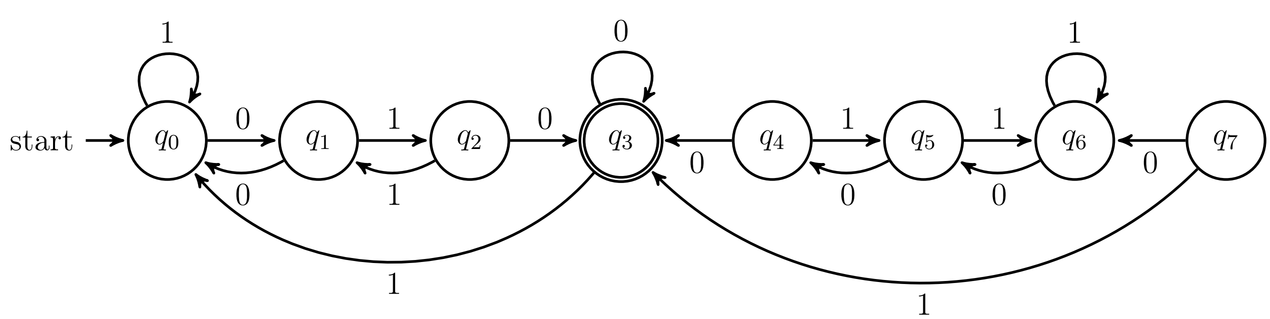
Inductive hypothesis: Suppose $\mathcal{F}'$ has read input upto a_k (k is odd). If the current state q^i in $\mathcal{F}'$ contains the state $q_1 \in Q$ (due to $a_1 \ldots a_k$) in q_o^i and the state $q_2 \in Q$ (due to $a_1 \ldots a_{k+1}$) in q_e^i, then $q^f = \delta'(q^i,\, a_{k+2}$ is such that $q_1' = \delta(q_1,\, ,a_{k+2}) \in q_o^f$ and $q_2' = \delta(q_2,\, a_{k+3} \in q_e^f$.

Inductive Step: Assuming the induction hypothesis to hold when $\mathcal{F}'$ has read upto (and including) a_k, we show that it holds after it has read a_{k+2} as well.
Let the state before reading a_{k+2} be q^i and after reading it be q^f, i.e., $q^f = \delta'(q^i,\, a_{k+2})$. As q_e^i has a state corresponding to $a_1 \ldots a_{k+1}$, say $q_1 \in Q$, then $q_1 \in q_e^i \Rightarrow \delta(q_1,\, a_{k+2}) \in q_o^f$ by the definition of δ' on q_o. But $q_1' = \delta(q_1,\, a_{k+2})$ is the state that $\mathcal{F}$ is on after reading a_{k+2}. Thus, the hypothesis holds for q_o. Now, observe that $q_2 = \delta(\delta(q_1,\, a_{k+2}),\, a_{k+3})$ is the state of $\mathcal{F}$ after reading $a_1 \ldots a_{k+1}$. By induction hypothesis, $q_1 \in q_e^i \Rightarrow \delta(\delta(q_1,\, a_{k+2}),\, a_{k+3})) = \delta(q_2,\, a_{k+3}) \in q_e^f$ since $a_{k+3} \in \Sigma$ and from the definition of δ' on q_e.
Thus, $\delta(q_1,\, a_{k+2} \in q_o^f$ and $\delta(q_2,\, a_{k+3}) \in q_e^f$, and the inductive hypothesis holds.

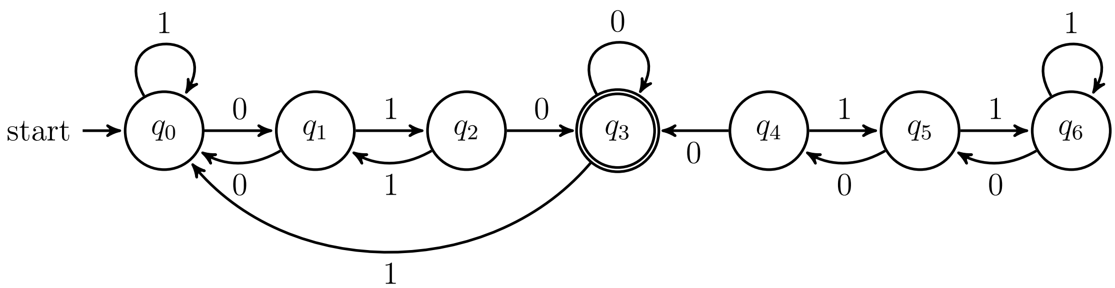
Using the above result, after $\mathcal{F}'$ and $\mathcal{F}$ have completely read their respective inputs, if $\mathcal{F}$ is in an accepting state, and $\mathcal{F}'$ is in the state q, then either q_o or q_e contains this accepting state, and by the definition of F', $\mathcal{F}'$ accepts this string as well. $\square$

Combining the above two results, we can conclude that $\mathcal{F}'$ accepts only *oddL*. Thus, *oddL* is a regular set as well. $\blacksquare$

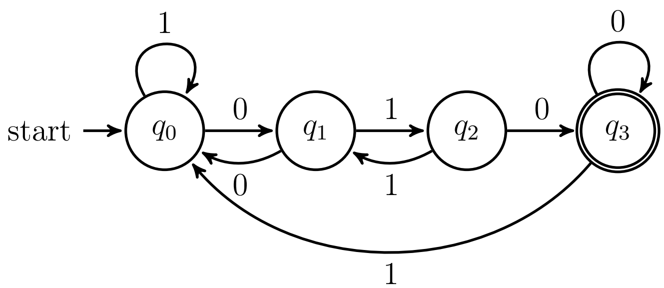
Q3. Find the minimum-state finite automaton corresponding to the following DFA. Show in details all the steps of minimization.



Solution: First let us remove all those states which will never be visited.
q_7 is not the start state and has no incoming transitions, therefore the DFA will remain the same upon removal of q_7 and the transitions that arise from it. The DFA will then look as follows.



Now let us look at node q_6. This node can only be reached by node q_5 which in turn can only be reached from q_4 which can never be reached from any one of q_0, q_1, q_2, q_3. Therefore none of q_4, q_5, and q_6 can be reached therefore we can eliminate those nodes and the transitions they offer. The DFA will look as follows:

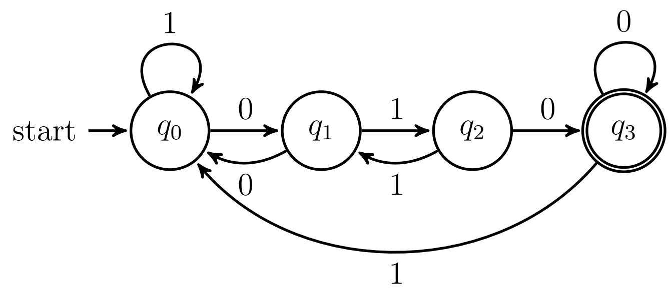


Now since all the nodes that can never be visited are removed, the next step in achieving minimum state DFA is to group together nodes that belong to the same equivalence class.

Let us consider nodes q_1 and q_2. Upon receiving input '0' they transition to q_0 and q_3 respectively. q_0 and q_3 can never belong to the same equivalence class since one is an accepting state and the other is not.

Now if we see nodes q_0 and q_2. Upon receiving input '0' they transition to q_1 and q_3 respectively. Similar to the above reason they do not belong to the same equivalent class.

Now in the case of nodes q_0 and q_1. Upon receiving input '1' they transition to q_0 and q_2 respectively. Since proved above q_0 and q_2 do not belong to the same equivalent class, q_0 and q_1 also do not belong to the same equivalent class. Trivially, q_3 cannot be equivalent to any of the other three states because it is an accepting state, while the other three are all rejecting states.
Therefore, the minimum state DFA is:



■

Q4. For languages L_1 and L_2 over Σ, define

$$\mathrm{Mix}(L_1, L_2) = \{w \in \Sigma^* \mid w = x_1y_1x_2y_2 \ldots x_ky_k, \text{where } x_1x_2 \ldots x_k \in L_1 \text{ and } y_1y_2 \ldots y_k \in L_2, \text{each } x_i, y_i \in \Sigma^*\}$$

Show that if L_1 and L_2 are regular then $\mathrm{Mix}(L_1, L_2)$ is also regular.

Solution: Let L_1 and L_2 be accepted by the **DFA**, $A_1 = (Q_1,\ \Sigma,\ s_1,\ \delta_1,\ F_1)$ and $A_2 = (Q_2,\ \Sigma,\ s_2,\ \delta_2,\ F_2)$ respectively. We construct an **NFA** to accept $\mathrm{Mix}(L_1, L_2)$.

Design an NFA, $A = (Q_1 \times Q_2,\ \Sigma,\ (s_1, s_2),\ \delta,\ F_1 \times F_2)$. The transition function is defined as follows:

$$\delta((s_1, s_2), a) = \{(\delta_1(s_1, a), s_2), (s_1, \delta_2(s_2, a))\}$$

Essentially, the first element of the pair in a state emulates transitions in A_1 while the second element of the pair emulates transitions in A_2. For every input, we either use it to make a transition A_1 or in A_2 (*but not both*).

Now to prove correctness.

Claim: If A accepts an input string $w = w_1w_2 \ldots w_l$, then $w \in \mathrm{Mix}(L_1, L_2)$.

Proof: Let the final state be $f = (f_1, f_2)$. Let us look at the sequence of states $s = q_0, q_1, q_2, \ldots, q_{l-1}, q_l = f$ (for some $f \in F$) which one of the walks of A undergoes on input w. For two consecutive states q_i, q_{i+1}, let g be a function which denotes the automata in which the transition was made for input w_{i+1}. More formally, $g : \{(q_0, q_1), (q_1, q_2), \ldots (q_{l-1}, q_l)\} \to \{1, 2\}$:

$$g((q_i, q_{i+1})) = \begin{cases} 1 & \text{if the transition to input } w_{i+1} \text{ occurs in } A_1 \\ 2 & \text{if the transition to input } w_{i+1} \text{ occurs in } A_2 \end{cases}$$

Now, run the following algorithm to obtain two strings x and y.

```
for i from 1 to l
        if g((q_{i-1}, q_i)) = 1:  append w_i to x
        else:  append w_i to y
```

By the construction of the transition function, $\hat{\delta_1}(s_1, x) = f_1$ and $\hat{\delta_2}(s_2, y) = f_2$. So $x \in F_1$ and $y \in F_2$, and $w \in \mathrm{Mix}(L_1, L_2)$. Hence proved.

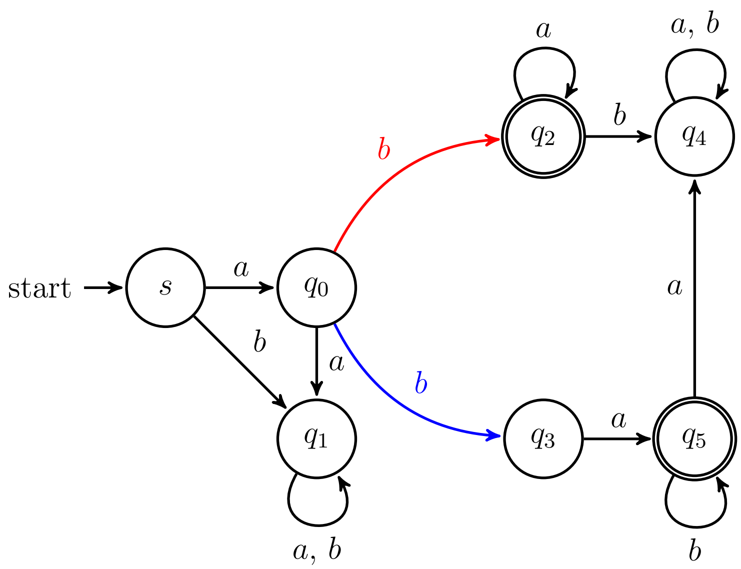
Claim: If $x \in L_1$ and $y \in L_2$ and w is some mix of x, y, then A accepts w.

Proof: While processing w, if the alphabet belongs to x, then simply perform a transition in the first element of the pair, otherwise perform a transition in the second element of the pair. Finally, after processing the entire input, the state will be $(\hat{\delta_1}(s_1, x), \hat{\delta_2}(s_2, y)) = (f_1, f_2)$ for some $f_1 \in F_1$ and $f_2 \in F_2$. This is an accepting state in A. Hence proved.

By the above two lemmas, we have shown that A accepts precisely $\text{Mix}(L_1, L_2)$, proving that the set is regular. ■

Q5. Design an NFA to accept the following language,

$$L = \{w \mid w \text{ is } abab^n \text{ or } aba^n \text{ where } n \geq 0\}.$$



The red transition and state q_2 correspond to accepting strings of the type aba^n, while the blue transition and state q_5 correspond to accepting strings of the type $abab^n$. Further, q_1 and q_4 are sink vertices to consume strings that are not in L, they can be merged into one, but have been made separately here for the sake of clarity. The interpretations of the different states are as follows:

- s : Start state
- q_0 : The string we have read up until now is a
- q_1 : Reject sink. No string starting with b is an element of L, thus it should not be accepted
- q_2 : The string we have read up until now is ab. This branch only accepts strings of the form aba^n. On receiving b, the input string takes the form aba^nb, it goes to the reject sink q_4, because the string is neither of the kind aba^n nor of the kind $abab^n$.
- q_3 : The string we have read up until now is ab. This branch leads to q_4, which only accepts strings of the form $abab^n$

q_5 : Branch of the NFA which accepts the strings of the form $abab^n$. On receiving a, the string becomes of the kind $abab^n a$, thus it goes to the reject sink q_4

q_4 : Reject sink for the strings which are rejected by q_2/q_4

■

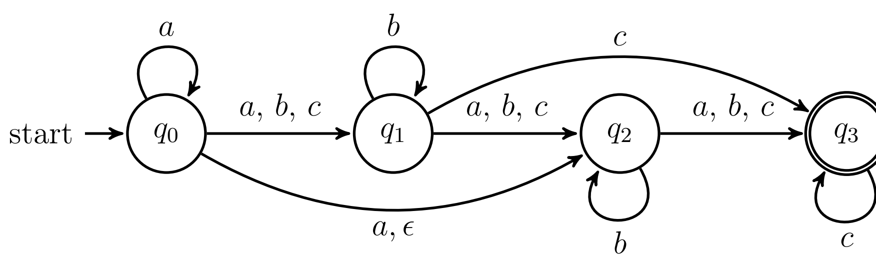
Q6. Let L be the language containing all strings over the alphabet $\{0, 1\}$ that satisfy the property that in every string the difference between the number of $0'$s and $1'$s is less than two (e.g., 00101 will be in the language, while 010001 will not). Is this a regular language? Explain your answer.

Solution: We show that L is not a regular language. The number of equivalences classes induced by L will be shown to be infinite. Consider the set of strings $S = \{0^{2m} \mid m \in \mathbb{N}\}$. We show that the elements of S belong to distinct equivalence classes. Towards contradiction, assume the existence of $m_1, m_2 \in \mathbb{N}$, with $m_1 < m_2$, and $[0^{2m_1}] = [0^{2m_2}]$. Now, $0^{2m_1}1^{2m_1} \in L$ by the definition of L. By equivalence, $0^{2m_2}1^{2m_1} \in L$. But,

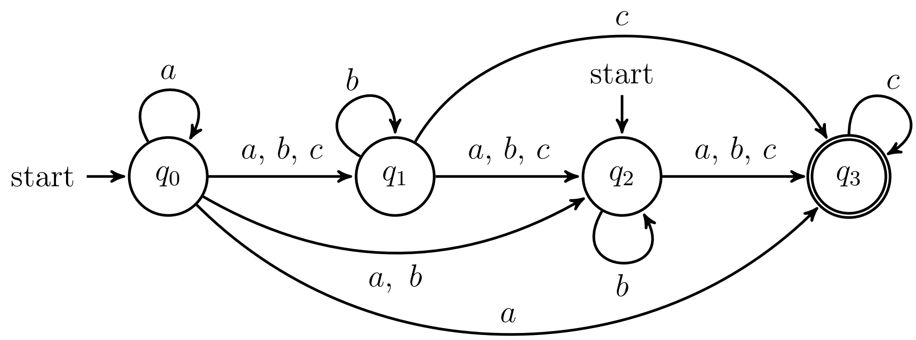
$$2m_2 - 2m_1 \geq 2(m_1 + 1) - 2m_1 = 2 \implies 0^{2m_2}1^{2m_1} \notin L$$

a contradiction. ■

Q7. Convert below NFA to its equivalent DFA.



Solution: *Assumption:* On taking the input a or b while in the state q_3, the DFA transitions to an implicit reject sink, labelled q_r, where all the inputs lead back to q_r itself.
We transform the given ϵ-NFA into an equivalent NFA by replacing the only ϵ transition, which is from q_0 to q_2, by making appropriate changes to the ϵ-NFA. Since q_2 is not a final state and it does not have further ϵ transitions, q_0 does not become a final state either. But, as q_0 is a start state and it contains an ϵ transition to q_2, q_2 becomes a start state as well. Now, as for the transitions, all the transitions originating from q_2 are *"duplicated"* and given to q_0, where the final state remains the same, but the input state becomes q_0.
After incorporating these changes, the new NFA with no ϵ transitions looks like:



Let $A = (Q, \Sigma, s, \delta, F)$ be the given NFA.
It's equivalent DFA will be of form $B = (2^Q, \Sigma, \{s\}, \delta_D, F_D)$, where,

$$F_D = \{H \mid H \subseteq Q \text{ and } H \cap F \neq \phi\}$$

$$\delta_D(H, a) = \bigcup_{q \in H} \text{reach}(q, a)$$

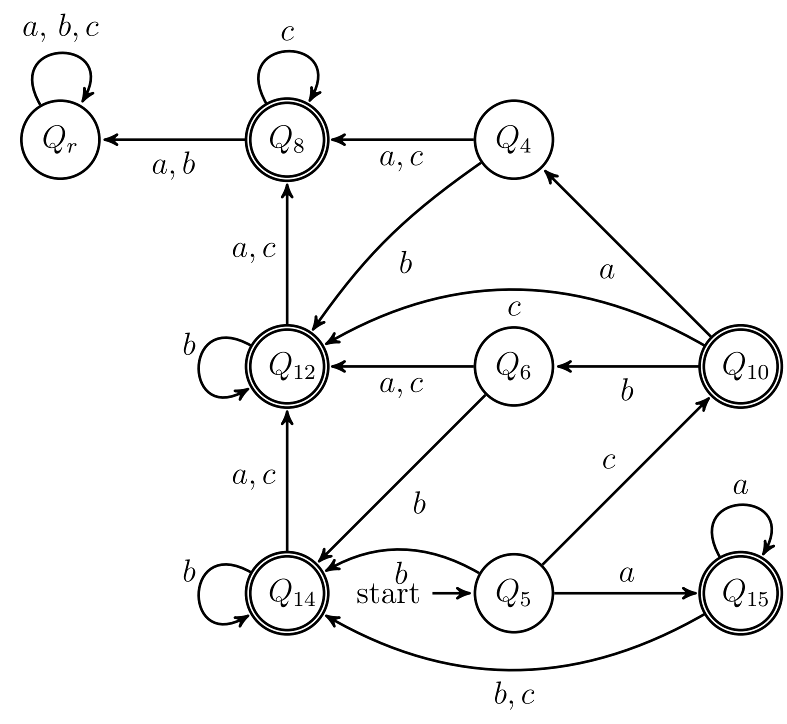
Since the states are subsets of the states of the NFA, we denote their subscript as the binary to decimal conversion of their existence. For example, the state $\{q1, q2, q3\}$ will be denoted as Q_{1110} or Q_{14}.
Although we have assumed the existence of another state, the reject sink, we ignore it in the transitions where there exists at least one state from the set $\{q_0,\ q_1,\ q_2,\ q_3\}$ and use the notation Q_r when the reject sink is the only possible transition in the NFA.

Our start state is $\{q_0,\ q_2\}$, i.e., Q_5.

$$\begin{aligned}
\delta_D(Q_5, a) &= \{q_0, q_1, q_2, q_3\} = Q_{15} \\
\delta_D(Q_5, b) &= \{q_1, q_2, q_3\} = Q_{14} \\
\delta_D(Q_5, c) &= \{q_1, q_3\} = Q_{10} \\
\delta_D(Q_{15}, a) &= \{q_0, q_1, q_2, q_3\} = Q_{15} && (\delta(q_3, a) = q_r \text{ but omitted}) \\
\delta_D(Q_{15}, b) &= \{q_1, q_2, q_3\} = Q_{14} && (\delta(q_3, b) = q_r \text{ but omitted}) \\
\delta_D(Q_{15}, c) &= \{q_1, q_2, q_3\} = Q_{14} \\
\delta_D(Q_{14}, a) &= \{q_2, q_3\} = Q_{12} && (\delta(q_3, a) = q_r \text{ but omitted}) \\
\delta_D(Q_{14}, b) &= \{q_1, q_2, q_3\} = Q_{14} && (\delta(q_3, b) = q_r \text{ but omitted}) \\
\delta_D(Q_{14}, c) &= \{q_2, q_3\} = Q_{12} \\
\delta_D(Q_{10}, a) &= \{q_2\} = Q_4 && (\delta(q_3, a) = q_r \text{ but omitted}) \\
\delta_D(Q_{10}, b) &= \{q_1, q_2\} = Q_6 && (\delta(q_3, b) = q_r \text{ but omitted}) \\
\delta_D(Q_{10}, c) &= \{q_2, q_3\} = Q_{12} \\
\delta_D(Q_{12}, a) &= \{q_3\} = Q_8 && (\delta(q_3, a) = q_r \text{ but omitted}) \\
\delta_D(Q_{12}, b) &= \{q_2, q_3\} = Q_{12} && (\delta(q_3, b) = q_r \text{ but omitted}) \\
\delta_D(Q_{12}, c) &= \{q_3\} = Q_8
\end{aligned}$$

$$\delta_D(Q_4, a) = \{q_3\} = Q_8$$
$$\delta_D(Q_4, b) = \{q_2, q_3\} = Q_{12}$$
$$\delta_D(Q_4, c) = \{q_3\} = Q_8$$
$$\delta_D(Q_6, a) = \{q_2, q_3\} = Q_{12}$$
$$\delta_D(Q_6, b) = \{q_1, q_2, q_3\} = Q_{14}$$
$$\delta_D(Q_6, c) = \{q_2, q_3\} = Q_{12}$$
$$\delta_D(Q_8, a) = \{q_r\} = Q_r \qquad (\because \text{ there is no other state in the set})$$
$$\delta_D(Q_8, b) = \{q_r\} = Q_r \qquad (\because \text{ there is no other state in the set})$$
$$\delta_D(Q_8, c) = \{q_3\} = Q_8$$
$$\delta_D(Q_r, a) = \{q_r\} = Q_r \qquad (\because \text{ there is no other state in the set})$$
$$\delta_D(Q_r, b) = \{q_r\} = Q_r \qquad (\because \text{ there is no other state in the set})$$
$$\delta_D(Q_r, c) = \{q_r\} = Q_r \qquad (\because \text{ there is no other state in the set})$$

The DFA is as follows:



∎

Q8. Let L be the language $L = \{w \in \{a, b\}^* \mid w \text{ contains an equal number of occurrences of } ab \text{ and } ba\}$

(a) Give a regular expression whose language is L.

(b) Design a DFA/NFA/ϵ-NFA to accept L.

Solution: Let $A = (a)^+$ and $B = (b)^+$, and $\#_{ab}(s)$ denote the count of occurrences of ab in string s, and similarly define $\#_{ba}(s)$.

Observation: The empty string $\epsilon \in L$, as $\#_{ab}(s) = 0 = \#_{ba}(s)$.
Throughout the rest of the discussion, we focus only on non-empty strings.

Lemma. 2 consecutive occurrences of a in a string s can be replaced by a single instance of a without affecting $\#_{ab}(s)$ and $\#_{ba}(s)$.
Proof. We mark the occurrences of the a's in (aa) as (a_1a_2). Suppose this occurrence of a_1a_2 in s is at the end, i.e., $s = t(a_1a_2)$ for some string t, define $\sigma \coloneqq t(a_1)$. It is easy to see how removing a_2 in s neither decreases $\#_{ab}(\sigma)$ nor $\#_{ba}(\sigma)$, since no new occurrence of ab has formed by the removal of a_2, and as a_2 was not followed by b, the occurrence of ab has not decreased either. The count of ba is trivially not affected by the removal of a_2 as it is preceded by a_1. Thus, $\#_{ab}(\sigma) = \#_{ab}(s)$ and $\#_{ba}(\sigma) = \#_{ba}(s)$ in this case.
A similar analysis can be done for the case when (a_1a_2) is followed by an a in s.
For the case $s = t(a_1a_2)bp$ for some strings t and p, consider $\sigma = t(a_1)bp$. On the removal of a_2 we lose one occurrence of ab (a_2b from s) but also gain one occurrence (a_1b in σ). The overall count of ab thus remains the same. The count of ba is trivially not affected by the removal of a_2. Thus, $\#_{ab}(s) = \#_{ab}(\sigma)$ and $\#_{ba}(s) = \#_{ba}(\sigma)$ in this case as well. $\square$

Corollary. Any number of consecutive a's can be replaced by a single a in a string s.

Lemma. Any string s starting with an a and ending with an a has an equal number of occurrences of ab and ba.
Proof. We invoke the corollary on s to reduce it to $s_n = a_0b_1a_1 \dots b_na_n$. Let $s_0 = a_0$. Observe how $s_n = A(BA)^n$.
We apply induction on n. In the base case ($n = 0$), it is easy to see that $\#_{ab}(s_0) = 0 = \#_{ba}(s_0)$. Assuming the induction hypothesis to hold for $n = k$, consider $s_{k+1} = a_0 \dots b_ka_kb_{k+1}a_{k+1} = s_kb_{k+1}a_{k+1}$ for some s_k. Clearly, $\#_{ba}(s_{k+1}) = \#_{ba}(s_k) + 1$. For the occurrences of ab, observe that s_{k+1} has a single new occurrence due to the addition of b_{k+1} (the occurrence is a_kb_{k+1} to be precise). Thus, $\#_{ab}(s_{k+1}) = \#_{ab}(s_k) + 1$.
By induction hypothesis, we have $\#_{ab}(s_k) = \#_{ba}(s_k)$, and so $\#_{ab}(s_{k+1}) = \#_{ba}(s_{k+1})$. Induction is complete covering all strings starting with an a and ending with an a. $\square$

Corollary. $s = A(BA)^* \Rightarrow s \in L$, i.e., strings starting with a and ending with a are members of L.

Lemma. Any string s starting with an a and ending with a b has an unequal number of occurrences of ab and ba. Specifically, $\#_{ab}(s) = \#_{ba}(s) + 1$.
Proof. We again invoke corollary on s to reduce it to $s_n = a_1b_1 \dots a_nb_n$. Observe how $s_n = (AB)^n$.
We apply induction on n. In the base case ($n = 1$, since $n = 0$ gives $s = \epsilon$), we have $\#_{ab}(s) = 1$ and $\#_{ba}(s) = 0$, so the hypothesis holds for $n = 0$.
Assuming the hypothesis to hold for $n = k$, we consider $s_{k+1} = a_1b_1 \dots a_kb_ka_{k+1}b_{k+1} =$

$s_k a_{k+1} b_{k+1}$ for some s_k. It is easy to see how the occurrence of ab has increased by 1, i.e., $\#_{ab}(s_{k+1}) = \#_{ab}(s_k)$. For the occurrences of ba, observe that s_{k+1} has a single new occurrence (the occurrence is $b_k a_{k+1}$ to be precise) so that $\#_{ba}(s_{k+1}) = \#_{ba}(s_k) + 1$. Since $\#_{ab}(s_k) = \#_{ba}(s_k)+1$, we get $\#_{ab}(s_{k+1})-1 = \#_{ba}(s_{k+1})-1+1 \Rightarrow \#_{ab}(s_{k+1}) = \#_{ba}(s_{k+1})+1$. Induction is complete for all starting with a and ending with b. □

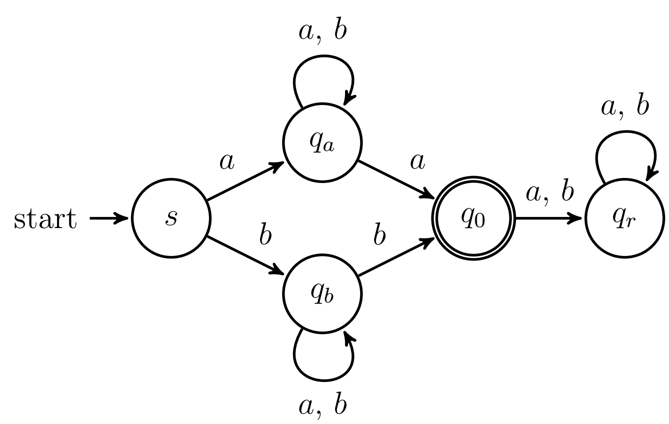
Corollary. $s = (AB)^* \Rightarrow s \notin L$, i.e., strings starting with a and ending with b are not members of L.

An analogous analysis can be done for strings starting b and ending with a b (or a). Trivially, a non-empty string s either starts with and ends with the same letter ($\Rightarrow s \in L$) or starts and ends with different letters ($\Rightarrow s \notin L$). Thus, simply checking if s starts and ends with the same letter is sufficient to check if $s \in L$.

(a) Given that we only need to ensure that a string is either empty or starts and ends with the same letter, the regular expression whose language is L can be written as:

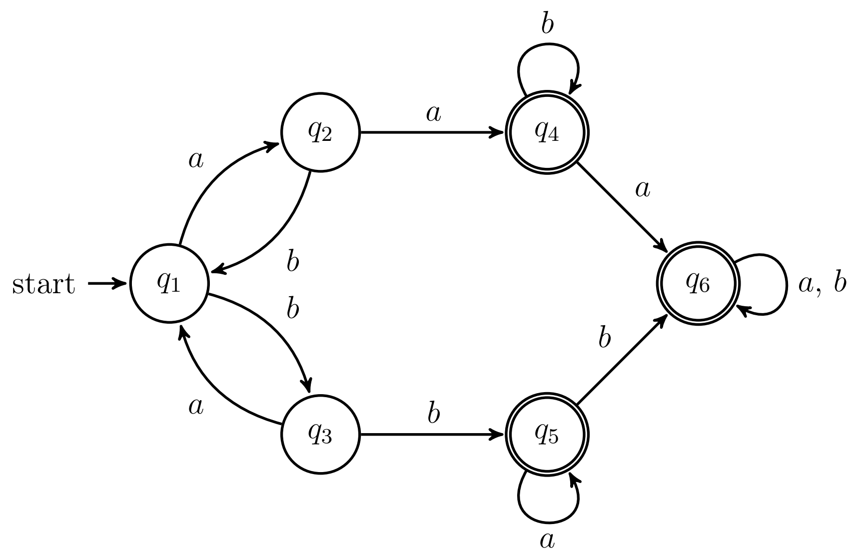
$$\boxed{r = \epsilon + a^+((b)^+(a)^+)^* + b^+((a)^+(b)^+)^*}$$

(b) The NFA for L is given below:



■

Q9. Use the DFA state-minimization procedure to convert this DFA to an equivalent DFA with the minimum possible number of states.

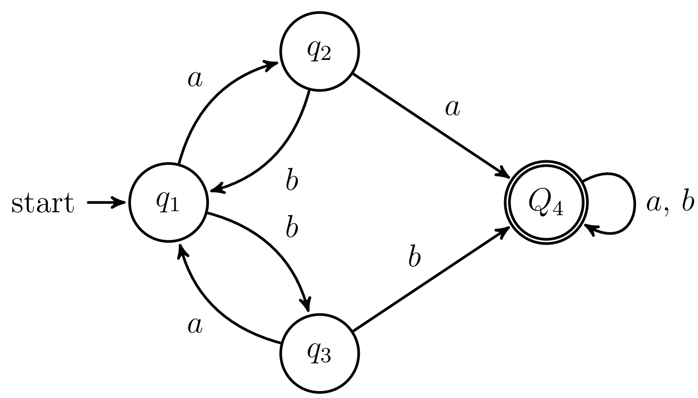


Solution: Here all states are reachable from the start state.

Let us look at all the accepting states and their transitions.

$\delta(q_4, a) = q_6$ $\qquad$ $\delta(q_5, a) = q_5$ $\qquad$ $\delta(q_6, a) = q_6$
$\delta(q_4, b) = q_4$ $\qquad$ $\delta(q_5, b) = q_6$ $\qquad$ $\delta(q_6, b) = q_6$

Any transition from q_4, q_5, and q_6 belongs to the same set of accept states. So q_4, q_5 and q_6 are indistuingishable and can be replaced with one state viz. Q_4.
At this stage the DFA will look as follows:

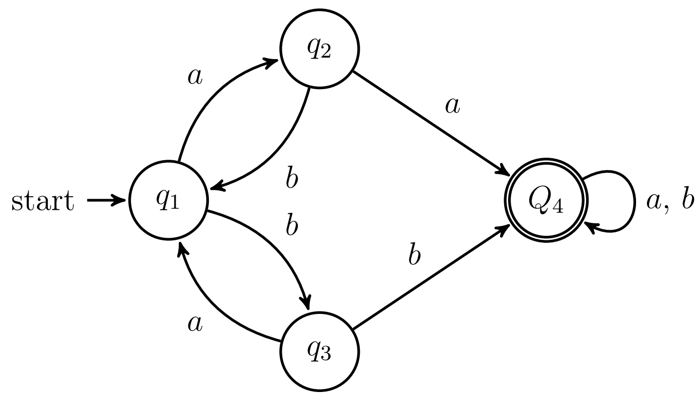


Now let us look at the remaining states and their transitions

$$\delta(q_1, a) = q_2 \qquad \delta(q_2, a) = Q_4 \qquad \delta(q_3, a) = q_1$$
$$\delta(q_1, b) = q_3 \qquad \delta(q_2, b) = q_1 \qquad \delta(q_3, b) = Q_4$$

It is very clear that q_2 and q_3 are distinguishable, i.e., do not belong to the same equivalence class because q_2 reaches an accepting state on reading a while q_3 reaches a rejecting state

on reading a. For the pair q_1 and q_2, upon receiving input a they transition to q_2 and Q_4 respectively which can never belong to the same equivalence class since one is an accepting state and the other is not. Similarly q_1 and q_3 are distinguishable because of their transitions on input b.
Therefore the final minimum state DFA will look like :



■

Q10. The language $L = \{uvv^r w \mid u, v, w \in \{a, b\}^+\}$ is regular. Here v^r is the reverse of v. Design a regular expression whose language is L. Convert the regular expression to an equivalent NFA.

Solution: We start by simplifying L through a series of observations which are stated as lemmas. Once it is simplified enough, we use its interpretation to frame the regular expression.

Lemma. The string vv^r is a palindrome of even length $\forall\, |v| > 0$.
Proof. It is easy to realize the above lemma by induction on the length of $|v|$. Firstly, $|vv^r| = |v| + |v^r| = 2 \cdot |v|$, thus it is even.
As for it being palindromic, we choose the *base case* of $|v| = 1$, then $v = v^r$ and thus it is palindromic. Assuming the hypothesis to hold for all $|v| \leq n$, we write $v = c \cdot s$, where $c \in \Sigma$ and $|v| = n + 1$, then $v^r = s^r \cdot c$ and $vv^r = css^r c$. The first and last character of vv^r are the same (c), and *by induction hypothesis* we have that ss^r is palindromic ($|s| = n$). Thus, vv^r is also palindromic and the induction is complete. □

Lemma. The string vv^r has a substring s of length two which is palindromic ($\Rightarrow s = cc,\ c \in \Sigma$).
Proof. Proof is trivial, assume $|v| = n$ and s is the concatenation of the n^{th} and the $(n+1)^{th}$ character (both of them are equal) of vv^r, i.e., s is of the kind cc where c is the n^{th} character of v, and it is trivially palindromic. □

Now, suppose $s = uvv^r w \in L$, then vv^r is a substring of s for some $v \in \{a, b\}^+$, i.e., there is some substring of $s[2 : |s| - 1]$, which is a palindrome ($s[2 : |s| - 1]$ denotes the substring of s made by dropping the first and the last character of s).

Since vv^r is a string of even length, we have, from this lemma, that cc is a substring of vv^r, and thus cc is a substring of $s[2 : |s| - 1]$. Put this way,

$$s \in L \Rightarrow cc \text{ substring of } s[2 : |s| - 1] \qquad (c \in \{a,\, b\})$$

Also, if $s[2 : |s| - 1]$ has aa (or bb) as a substring, then $s \in L$ by choosing $v = a$ (or $v = b$ respectively), and choosing u and w as an appropriate prefix and suffix respectively, or in other words,

$$cc \text{ substring of } s[2 : |s| - 1]\ (c \in \{a,\, b\}) \Rightarrow s \in L$$

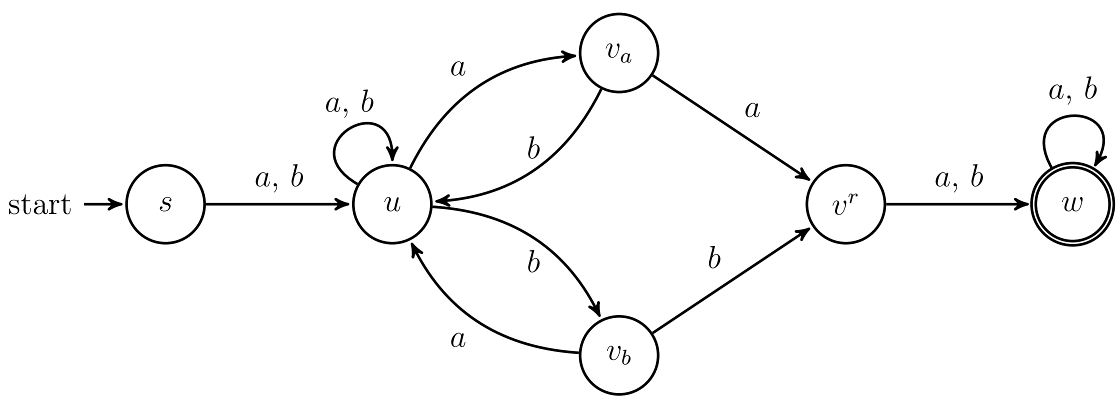
On combining both the results,

$$s \in L \Leftrightarrow cc \text{ substring of } s[2 : |s| - 1] \qquad (c \in \{a,\, b\})$$

We are now in a position to frame a regular expression r for L, which has some (non-zero) occurrences of $\{a,\, b\}$, at least one occurrence of aa or bb, followed by at least one occurrence of $\{a,\, b\}$ in each of its strings:

$$\boxed{r = (a + b)^+(aa + bb)(a + b)^+}$$

It is easy to draw the correspondence for u, v, w from r.
We now design an NFA that accepts r:



■

Q11. Find out the equivalent regular expressions from the list given and show why they are equivalent :
1. $(a + ba)^*(b + \epsilon)$
2. $(a^*(ba)^*)^*(b + \epsilon) + a^*(b + \epsilon) + (ba)^*(b + \epsilon)$
3. $(a + ba)(a + ba)^*(b + \epsilon)$

Solution: It is easy to see that $b \in (a+ba)^*(b+\epsilon)$ but $b \notin (a+ba)(a+ba)^*(b+\epsilon)$. Thus, **1.** and **3.** are clearly not equivalent. Also, $b \in (a^*(ba)^*)^*(b + \epsilon) + a^*(b + \epsilon) + (ba)^*(b + \epsilon)$, so expressions **2.** and **3.** are not equivalent either.

Comparing **1.** and **2.**, both of them are suffixed by $(b+\epsilon)$, so we can remove this from the both of them and compare $r_1 := (a+ba)^*$ and $r_2 := (a^*(ba)^*)^* + a^* + (ab)^*$ instead. Let $ba = c$, for simpler notations, then $r_1 = (a+c)^*$ and $r_2 = (a^*c^*)^* + a^* + c^*$. We now prove a series of lemmas. In each of them r, r_1 and r_2 denote regular expressions.

Lemma. $L(r) \subseteq L(r^*)$
Proof. This is obviously true since by definition, $L(r^*)$ contains all combinations of string concatenations using strings in $L(r)$. If we simply concatenate each string once it immediately follows that $L(r) \subseteq L(r^*)$. □

Lemma. $L(r_1^*) \subseteq L(r_1^* r_2^*) = L(r_1^*) \cdot L(r_2^*)$
Proof. $L(r_1^* r_2^*)$ contains all strings of the form $s_1 s_2$ where $s_1 \in L(r_1^*)$ and $s_2 \in L(r_2^*)$. Consider $s_2 = \epsilon \in L(r_2^*)$. Then $s_1 = s_1\epsilon \in L(r_1^* r_2^*)\ \forall\, s_1 \in L(r_1^*)$. Hence proved. □

Lemma. If $L(r_1) \subseteq L(r)$ and $L(r_2) \subseteq L(r)$, then $L(r_1 + r_2) \subseteq L(r)$
Proof. This is trivially true since $L(r_1 + r_2) = L(r_1) \cup L(r_2)$. □

Lemma. If $L(r_1) \subseteq L(r)$, then $L(r_1 + r) = L(r)$
Proof. Again, this is trivially true as $L(r_1 + r) = L(r_1) \cup L(r) = L(r)$, since $L(r_1) \subseteq L(r)$. □

Lemma. If $L(r_1) \subseteq L(r)$, then $L(r_1^*) \subseteq L(r^*)$
Proof. Let $s \in L(r_1^*)$ with $s = a_1 a_2 \ldots a_k$ with $a_i \in L(r_1) \Rightarrow a_i \in L(r)$. Hence $s \in L(r^*)$. □

Lemma. $L((r^*)^*) = L(r^*)$
Proof. Property of $*$ operator.

First let's simplify expression 2 using the lemmas above. $L(a^*) \subseteq L(a^*c^*)$ and $L(c^*) \subseteq L(a^*c^*)$. So, $L(a^* + c^*) \subseteq L(a^*c^*)$. Also, $L(a^*c^*) \subseteq L((a^*c^*)^*)$. Therefore, $L(a^* + c^*) \subseteq L((a^*c^*)^*)$. Finally, $L(a^* + c^* + (a^*c^*)^*) = L((a^*c^*)^*)$. Now we show $L((a+c)^*) = L((a^*c^*)^*)$.

Claim. $L((a+c)^*) \subseteq L((a^*c^*)^*)$
Proof. $L(a) \subseteq L(a^*) \subseteq L(a^*c^*)$. $L(c) \subseteq L(c^*) \subseteq L(a^*c^*)$. Therefore, $L(a+c) \subseteq L(a^*c^*)$. Finally, $L((a+c)^*) \subseteq L((a^*c^*)^*)$. Hence proved. □

Claim. $L((a^*c^*)^*) \subseteq L((a+c)^*)$
Proof. Consider any $s \in L(a^*c^*)$. We can further decompose s into $a_1 a_2 \ldots a_k c_1 c_2 \ldots c_l$ where $a_i \in L(a)$ and $c_i \in L(c)$. But note that $s \in L((a+c)^*)$ since each $a_i, c_i \in L(a+c)$. Hence $L(a^*c^*) \subseteq L((a+c)^*)$. So, $L((a^*c^*)^*) \subseteq L(((a+c)^*)^*) = L((a+c)^*)$. Hence proved. □

By the two claims above, $L((a+c)^*) = L((a^*c^*)^*)$. Hence, only expressions **1.** and **2.** are equivalent. ■

Q12. Design DFA for the following languages over $\{0, 1\}$:

(a) The set of all strings such that every block of five consecutive symbols have at least two 0s.

(b) The set of strings with an equal number of 0s and 1s such that each prefix has at most one more 0 than 1s and at most one more 1 than 0s.

Solution: ($a1$): Let $A = (Q, \Sigma, s, \delta, F)$ be the required DFA. We now define each component of A.

Q: States in Q will be indexed by bit string of length *at most* 5. Instead of writing the index as a subscript, we denote states by $q[X]$ where X is a binary string. The state $q[X]$ denotes the latest sequence of inputs in chronological order with the leftmost bit being the oldest and the rightmost bit being the newest input. We also include $q[\phi]$ which denotes an empty binary string and a sink, rejecting state r. The total number of states added is this $2^0 + 2^1 + 2^2 + 2^3 + 2^4 + 2^5 + 1$ (rejecting sink) $= 64$ states.

s: The starting state is $q[\phi]$.

δ: The definition of δ is pretty obvious from the definition of the states themselves. It is as follows:

$$\delta(q[X], a) = q[Xa] \qquad (|X| \leq 4)$$
$$\delta(q[x_1x_2x_3x_4x_5], a) = q[x_2x_3x_4x_5a] \qquad (x_i \text{ are the individual bits of } X)$$

F: For input strings with less than 5 bits, they vacuously belong to the language. Hence, $q[\phi]$ and $q[X]$ are all accepting states for $|X| \leq 4$. If at any point of the input, 5 consecutive symbols have less than two 0s, then the string must be rejected. Hence, if any time during the computation, the DFA ends up in any of the states: $q[11111]$, $q[11110]$, $q[11101]$, $q[11011]$, $q[10111]$, or $q[01111]$, then the input string is invalid. So combine these 6 states with the rejecting sink, r. Except for r, all the states should be accepting.

We have thus defined the DFA that accepts the required language. It contains 58 states. However, many of these states are unnecessary and the total number of states can be reduced to 11 if the DFA is implemented efficiently. This efficiency comes at the cost of clarity though. The efficient implementation of the DFA is given below.

($a2$): Q: States in Q are as follows:

q_0 : State which denotes strings ending with at least min(2, length of string) 0s

q_1 : State which denotes strings ending with 1

q_{11} : State which denotes strings ending with 11.

q_{111} : State which denotes strings ending with 111.

q_r : State which denotes strings which are rejected. This is a reject sink.

q_{1110} : State which denotes strings ending with 1110.

q_{110} : State which denotes strings ending with 110.

q_{1101} : State which denotes strings ending with 1101.

q_{10} : State which denotes strings ending with 10.

q_{101} : State which denotes strings ending with 101.

q_{1011} : State which denotes strings ending with 1011.

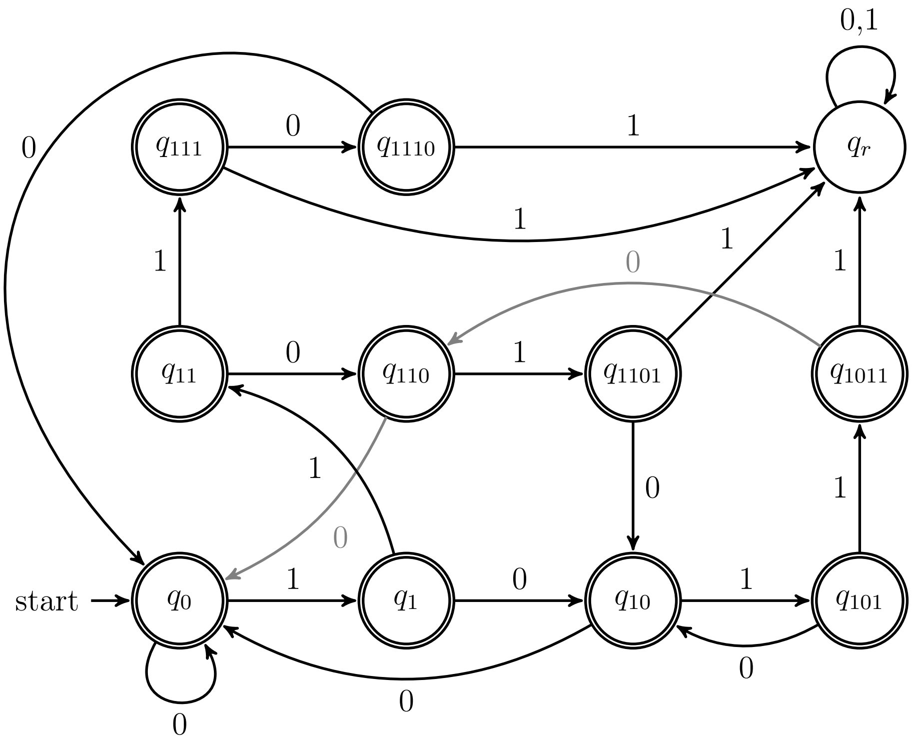
s: The starting state is q_0.

δ: The definition of δ is as follows:

$$
\begin{aligned}
\delta(q_0, 1) &= q_1 && \text{(by definition)} \\
\delta(q_0, 0) &= q_0 && \text{(by definition)} \\
\delta(q_1, 0) &= q_{10} && \text{(by definition)} \\
\delta(q_1, 1) &= q_{11} && \text{(by definition)} \\
\delta(q_{11}, 0) &= q_{110} && \text{(by definition)} \\
\delta(q_{11}, 1) &= q_{111} && \text{(by definition)} \\
\delta(q_{111}, 0) &= q_{1110} && \text{(by definition)} \\
\delta(q_{111}, 1) &= q_r && \text{(block with 1111 has can have max one 0)} \\
\delta(q_{10}, 0) &= q_0 && \text{(by definition)} \\
\delta(q_{10}, 1) &= q_{101} && \text{(by definition)} \\
\delta(q_{110}, 0) &= q_0 && \text{(by definition)} \\
\delta(q_{110}, 1) &= q_{1101} && \text{(by definition)} \\
\delta(q_{1110}, 0) &= q_0 && \text{(by definition)} \\
\delta(q_{1110}, 1) &= q_r && \text{(if the last 5 bits are 11101 this block has only one 0)} \\
\delta(q_r, 0) &= q_r && \\
\delta(q_r, 1) &= q_r && \\
\delta(q_{101}, 0) &= q_{10} && \text{(since all strings upto the 0 have at least two 0s)} \\
\delta(q_{101}, 1) &= q_{1011} && \text{(by definition)} \\
\delta(q_{1101}, 0) &= q_{10} && \text{(since all strings upto the 0 have at least two 0s)} \\
\delta(q_{1101}, 1) &= q_r && \text{(if the last 5 bits are 11011 this block has only one 0)} \\
\delta(q_{1011}, 0) &= q_{110} && \text{(since all strings upto the 0 have at least two 0s)} \\
\delta(q_{1011}, 1) &= q_r && \text{(if the last 5 bits are 10111 this block has only one 0)}
\end{aligned}
$$

F: For input strings with less than 5 bits, they vacuously belong to the language. Hence, all states are accepting states for $|X| \leq 4$. If input string has more than 5 bits then all states apart from q_r are accepting, i.e., q_r will be the only rejected state for $|X| \geq 5$

The DFA is as follows:



(b): Again, let $A = (Q, \Sigma, s, \delta, F)$ be the required DFA. The components of A are:

Q: There will be 4 states. A rejecting sink r, and three states q_0, q_{+1} and q_{-1}. q_i denotes the difference between the number of 0s and 1s up to the latest input. If at any point, the absolute value exceeds 1, we move to the rejecting sink.

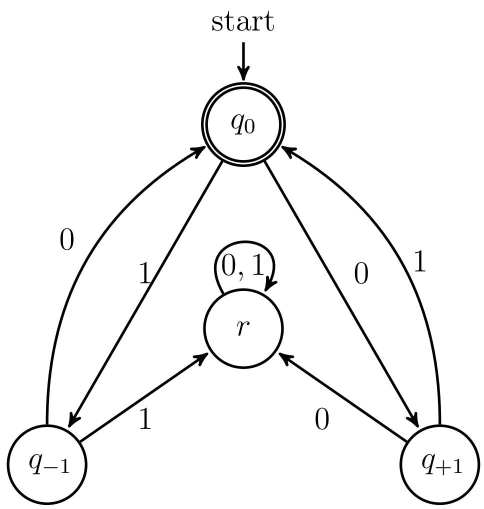
s: The starting state is q_0.

δ: The transition function is defined as follows:

$$
\begin{aligned}
\delta(q_0, 0) &= q_{+1} \\
\delta(q_0, 1) &= q_{-1} \\
\delta(q_{+1}, 0) &= r \\
\delta(q_{+1}, 1) &= q_0 \\
\delta(q_{-1}, 0) &= q_0 \\
\delta(q_{-1}, 1) &= r \\
\delta(r, a) &= r
\end{aligned}
$$

F: Only q_0 is accepting.

The DFA is shown below:



■

Q13. Given that language L is regular, is the language $L_{1/2} = \{x \mid \exists y \text{ such that } |x| = |y|, xy \in L\}$ regular? If yes, give a formal proof.

Solution: We claim that $L_{1/2}$ is regular, by constructing an NFA which accepts the same. Let $A = (Q, \Sigma, s, \delta, F)$ be an NFA with a *single final state* f which accepts L. Essentially, the idea is to run two copies of A in parallel. One runs normally and the other runs in "reverse". Finally, if both the automata end up in the same state, we accept the input. Let us formalize these notions now.

Let Q_k denote the set of states in A which can reach f by processing some string of length k. Also, define $\delta(q, S)$, $S \in \Sigma^*$ as the state that is reached from q by processing the string S. We can recursively define Q_k.

$$Q_0 = \{f\} \cup \{q \mid \delta(q, \epsilon^l) = f, \text{ for some } l \in \mathbb{N}\}$$
$$Q_k = \{q \mid \delta(q, \epsilon^{l_1} a \epsilon^{l_2}) = q', \text{ for some } a \in \Sigma,\ l_1, l_2 \in \mathbb{N}_{\cup\{0\}} \text{ and } q' \in Q_{k-1}\} \qquad (k \geq 1)$$

We are finally ready to define the NFA which accepts $L_{1/2}$. Let $A_{1/2} = (Q \times Q, \Sigma, (s, f), \delta_{1/2}, F_{1/2})$ denote the required NFA. Suppose the input to the NFA is the string $x_1 x_2 \ldots x_n$. The transition functions is defined as:

$$\delta_{1/2}((q_1,\, q_2),\, x_k) = \delta(q_1,\, x_k) \times Q_k$$

The final states are:

$$F = \{(q, q) \mid q \in Q\}$$

We prove the correctness of the above construction now.

Claim: If $A_{1/2}$ accepts an input x of length k, then $\exists$ another string y of length k such that A accepts xy.

Proof: Suppose $A_{1/2}$ accepts an input of x of length k. We are in a state (q, q) when the input is exhausted. Now, the first element of the pair in a state in $A_{1/2}$ simply emulates A.

So, $q \in \delta(s, x)$. By the construction of the transition function, the second element of any state the NFA is currently in after processing k inputs is always in Q_k. So, $q \in Q_k$. Hence, by definition, $\exists$ some string input y of length k such that $f \in \delta(q, y)$. So, it is easy to see that $f \in \delta(s, xy) \Rightarrow A$ accepts xy. So, we have found the existence of a suitable y. Hence proved.

Claim: If A accepts an input xy of length $2k$, where x and y are both of length k, then $A_{1/2}$ accepts x.

Proof: Suppose $\delta(s, x) = q$. Since the first element of the pair in a state of $A_{1/2}$ emulates A and the second element of the pair is Q_k after any input of length k,

$$\{q\} \times Q_k \subseteq \delta_{1/2}((s, f), x)$$

Since A accepts xy, $f \in \delta(s, xy) \Rightarrow f \in \delta(q, y)$. Now, by definition, $q \in Q_k$ since there is a k length string y such that $f \in \delta(q, y)$. So, $(q, q) \in \delta_{1/2}((s, f), x)$, which is an accepting state. So $A_{1/2}$ accepts x. Hence proved.

With both these implications, we can see that $A_{1/2}$ accepts precisely $L_{1/2}$. Hence $L_{1/2}$ must be regular. ■

Q14. Define L' as the language consisting of the reverse of the strings in a language L. Give a formal proof that if L is regular, then L' is regular.

Solution: Let $A = (Q, \Sigma, s, \delta, F)$ be an NFA with a *single final state* f which accepts L. We construct another NFA, A' which accepts L'. The idea is to essentially reverse the direction of all the arrows in A and to start from f while the final state is s. So, let $A' = (Q, \Sigma, f, \delta', \{s\})$. The transition function is defined as:

$$p \in \delta(q, a) \Leftrightarrow q \in \delta'(p, a)$$

Claim. If an input string is accepted by L, then L' accepts the reverse string.
Proof. Analyze the walk across states which accepts some string in L. Let the sequence of states visited be $q_0 = s, q_1, q_2, \ldots, q_{k-1}, q_k = f$ for some $k \in \mathbb{N}$. Note that this k may not be equal to the length of the input string because the accepting walk might have ϵ transitions in it. Let the sequence of alphabets chosen during each transition be $\{a_0, a_1, \ldots a_{k-1}\}$. Equivalently, $q_{i+1} \in \delta(q_i, a_i)$. Now, we show that L' can trace the exact same path in a reversed manner using induction.

Base Case: L' can trace the path q_1, q_0 using a_0. This immediately follows from the definition of δ'; $q_1 \in \delta(q_0, a_0) \Rightarrow q_0 \in \delta'(q_1, a_0)$.

Induction Hypothesis: For some $l < k$, L' can trace the path $q_l, q_{l-1}, \ldots, q_0$ using the alphabet sequence $a_{l-1}, a_{l-2}, \ldots, a_0$.

Inductive Step: We show that the hypothesis holds for $(l + 1)$. To do so, notice that $q_{l+1} \in \delta(q_l, a_l) \Rightarrow q_l \in \delta'(q_{l+1}, a_l)$. So, from q_{l+1}, walk to q_l using a_l. After that, we can invoke the inductive hypothesis and complete the walk till q_0. Hence we have proved the hypothesis for $(l+1)$, and we can conclude the original claim.

Now, it can be easily seen that $(L')' = L$ and $(A')' = A$. So, if we invoke the claim and substitute L with L', we get that for any string accepted by L', $(L')' = L$ accepts the reverse string. Thus we have:

$$S \in L \Leftrightarrow S' \in L'$$

where S' denotes the reverse of string S. Since we have constructed a valid NFA for L', it is regular. Hence proved. ■

Q15. Let L be the language over the alphabet $\{0, 1, 2\}$ such that for any string in $s \in L$, s does not have two consecutive identical symbols, i.e. strings of L are any string in $\{0, 1, 2\}^*$ such that there is no occurrence of 00, no occurrence of 11, and no occurrence of 22. Design a DFA that accepts L.

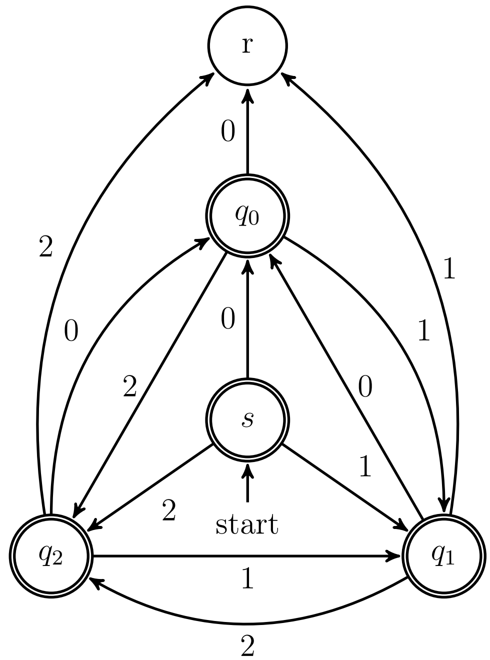
Solution: We design a DFA containing 5 states defined as follows:

s	: The starting state
q_0	: State denoting the last input processed was 0
q_1	: State denoting the last input processed was 1
q_2	: State denoting the last input processed was 2
r	: A *sink state* used for rejecting inputs

Now, the transitions can be set up as follows:

$$\begin{aligned}
\delta(s, a) &= q_a \\
\delta(q_a, b) &= q_b \qquad (a \neq b) \\
\delta(q_a, a) &= R \\
\delta(R, a) &= R
\end{aligned}$$

The DFA for the same is given below:



∎

Q16. Let Σ and Δ be two alphabets and let $h : \Sigma^* \to \Delta^*$. Extend h to be a function from Σ^* to Δ^* as follows:

$$\begin{aligned} h(\epsilon) &= \epsilon, \\ h(wa) &= h(w)h(a) \qquad\qquad \text{where } w \in \Sigma^*, a \in \Sigma \end{aligned}$$

(Such a function h is called a *homomorphism*.)
Now, for $L \subseteq \Sigma^*$,

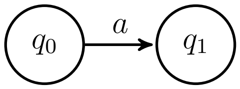
$$h(L) = \{h(w) \in \Delta^* \mid w \in L\}.$$

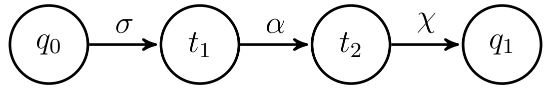
Also, for $L \subseteq \Delta^*$,

$$h^{-1}(L) = \{w \in \Sigma^* \mid h(w) \in L\}.$$

1. Prove that if $L \subseteq \Sigma^*$ is regular, then so is $h(L)$.
2. Prove that if $L \subseteq \Delta^*$ is regular, then so is $h^{-1}(L)$.

Solution: 1. Let L be accepted by the DFA $\mathcal{F} = (Q, \Sigma, s, \delta, F)$. We show that $h(L) \subseteq \Delta^*$ is regular by constructing an NFA $\mathcal{F}' = (Q', \Delta, s, \delta', F)$ from $\mathcal{F}$. $Q' \supseteq Q$ and the accepting states of $\mathcal{F}'$ remain the same as $\mathcal{F}$. To construct $\delta' : Q' \times \Delta \to 2^{Q'}$, we pick a transition (on input letter say $a \in \Sigma$) in $\mathcal{F}$ and replace it by $|h(a)|$ many transitions, each corresponding to a letter in $h(a)$. This gives us the construction for both δ' and Q', which just contains some extra, intermediate states (given rise to by the replacements) in addition to each state in Q and an implicit reject sink.
For example, say $a \in \Sigma$, and $h(a) = \sigma\alpha\chi$, then we replace each transition concerning the input a in $\mathcal{F}$ by 3 transitions, one corresponding to each letter in $h(a)$:





Essentially, $\mathcal{F}'$ tries to check if the input string s' given into it can be made with the help of a string s in L, i.e., it tries to find if $\exists\, s \in L$, such that $h(s) = s'$.
For correctness, we prove two things, that if $s \in L$, then $\mathcal{F}'$ accepts s, and secondly, if $\mathcal{F}'$ accepts s' then there exists an $s \in L$ such that $h(s) = s'$ or in other words, that $s' \in h(L)$.

- Showing that $s \in L \Rightarrow h(s)$ is accepted by $\mathcal{F}'$
  Proof. Let $s = a_1 a_2 \dots a_n$ where $a_i \in \Sigma\, \forall\, 1 \leq i \leq n$. Now, s must be accepted by $\mathcal{F}$ which means that there $\mathcal{F}$ takes transitions corresponding to $a_1, a_2 \dots a_n$ to reach an accepting state. Also, by the definition of h, we have,

  $$h(s) = h(a_1 a_2 \dots a_n) = h(a_1)h(a_2)\dots h(a_n)$$

  By the definition of δ' and Q', $\mathcal{F}'$ must make the transitions due to $h(a_1)$, followed by $h(a_2)$, and so on, eventually reaching the exact same final state that $\mathcal{F}$ reaches on completely reading s. Thus, by the very definition of $\mathcal{F}'$, the above-mentioned property follows. □

- Showing that s' accepted by $\mathcal{F}' \Rightarrow \exists\, s \in L$ such that $h(s) = s'$
  Proof. As s' is accepted by $\mathcal{F}'$, pick the path in which $\mathcal{F}'$ ends up on an accepting state, say q_f. We are guaranteed that $q_f \in Q$, since, $q_f \in F \subseteq Q$ and also $s \in Q$. The path might possibly have more states which come from Q.
  Given this path and the states from Q that this path goes through, it is trivial to reverse engineer the path into a string $s \in \Sigma^*$, simply by looking up how $\mathcal{F}$ was converted to $\mathcal{F}'$. In essence, we take the transitions from one state in Q to another state in Q, and replace it by a letter in Σ, thus, making a string s from s'. This is possible because of the way $\mathcal{F}'$ has been constructed.
  Now, on reading the string s, $\mathcal{F}$ must also land on the same states in Q and in the same order as $\mathcal{F}'$, because $\mathcal{F}$ is a DFA, and by the definition of $\mathcal{F}'$. Since, the states in Q are same for both $\mathcal{F}$ and $\mathcal{F}'$, and $\mathcal{F}'$ ends up on q_f, $\mathcal{F}$ must also end up on $q_f \in F$. Thus, $\mathcal{F}$ accepts s and our proof is complete. □

We now have an NFA $\mathcal{F}'$ that accepts exactly $h(L)$, which was constructed from a DFA accepting L. Showing that if L is regular, then so is $h(L)$.

2. Let $A = (Q, \Delta, s, \delta, F)$ be a DFA which accepts a language $L \subseteq \Delta^*$. We construct another DFA, $A' = (Q, \Sigma, s, \delta', F)$ which accepts $h^{-1}(L) \subseteq \Sigma^*$. Note that A' has the same set of states, starting state and final state as A. It differs on the alphabet and the transition function. Define the transition function as follows:

$$\delta'(q, a) = \hat{\delta}(q, h(a))$$

Lemma. For any string $x \in \Sigma^*$, $\hat{\delta}'(q, x) = \hat{\delta}(q, h(x))\, \forall q \in Q$.

Proof. We prove the same using induction. Let ϵ_Σ denote the empty string in Σ^* and ϵ_Δ denote the empty string in Δ^*. We show the base case for $x = \epsilon_\Sigma$. Since both A and A' are DFAs, $\delta'(q, x) = \delta'(q, \epsilon_\Sigma) = q$ and $\delta(q, h(x)) = \delta(q, \epsilon_\Delta) = q$. The base case is done.

Inductive Hypothesis: For any string x of length k and any $q \in Q$, $\hat{\delta}'(q, x) = \hat{\delta}(q, h(x))$.

Inductive Step: We now need to prove the hypothesis for a string x of length $(k + 1)$. Indeed, let $x = ya$ where y is of length k.

$$\begin{aligned}
\hat{\delta}'(q, x) &= \hat{\delta}'(q, ya) \\
&= \delta'(\hat{\delta}'(q, y), a) \\
&= \delta'(\hat{\delta}(q, h(y)), a) && \text{(Induction hypothesis)} \\
&= \hat{\delta}(\hat{\delta}(q, h(y), h(a)) && \text{(Definition of } \delta'(q, a)) \\
&= \hat{\delta}(q, h(y)h(a)) && \text{(Property of transition function)} \\
&= \hat{\delta}(q, h(ya)) && \text{(Definition of homomorphism)} \\
&= \hat{\delta}(q, h(x))
\end{aligned}$$

Hence we are done by induction. □

If x is any string accepted by A', then $\hat{\delta}'(s, x) \in F \Rightarrow \hat{\delta}(s, h(x)) \in \Rightarrow$ by the lemma. So A accepts $h(x)$. Conversely, if $h(x)$ is a string accepted by A, then $\hat{\delta}(s, h(x)) \in F \Rightarrow \hat{\delta}'(s, x) \in F$, again by the lemma. So A' accepts x. Thus A' precisely accepts $h^{-1}(L)$. ■