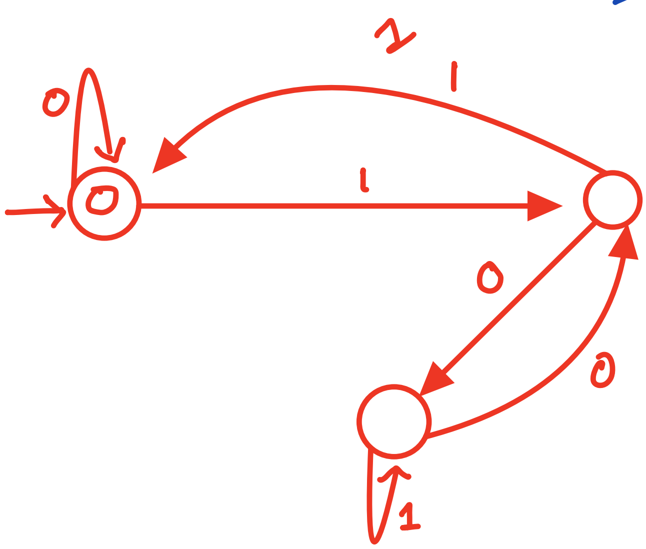
Some more examples of DFAs

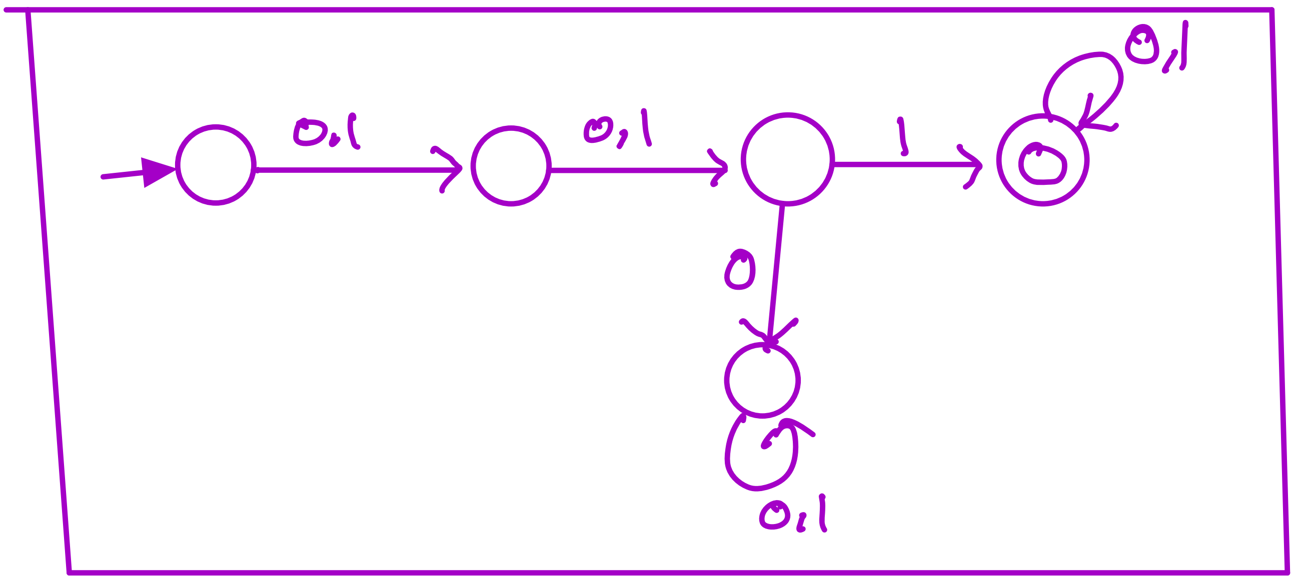
Example:

Let $L = \{ w \in (0+1)^* \mid w$ is a binary representation of an integer divisible by 3 $\}$

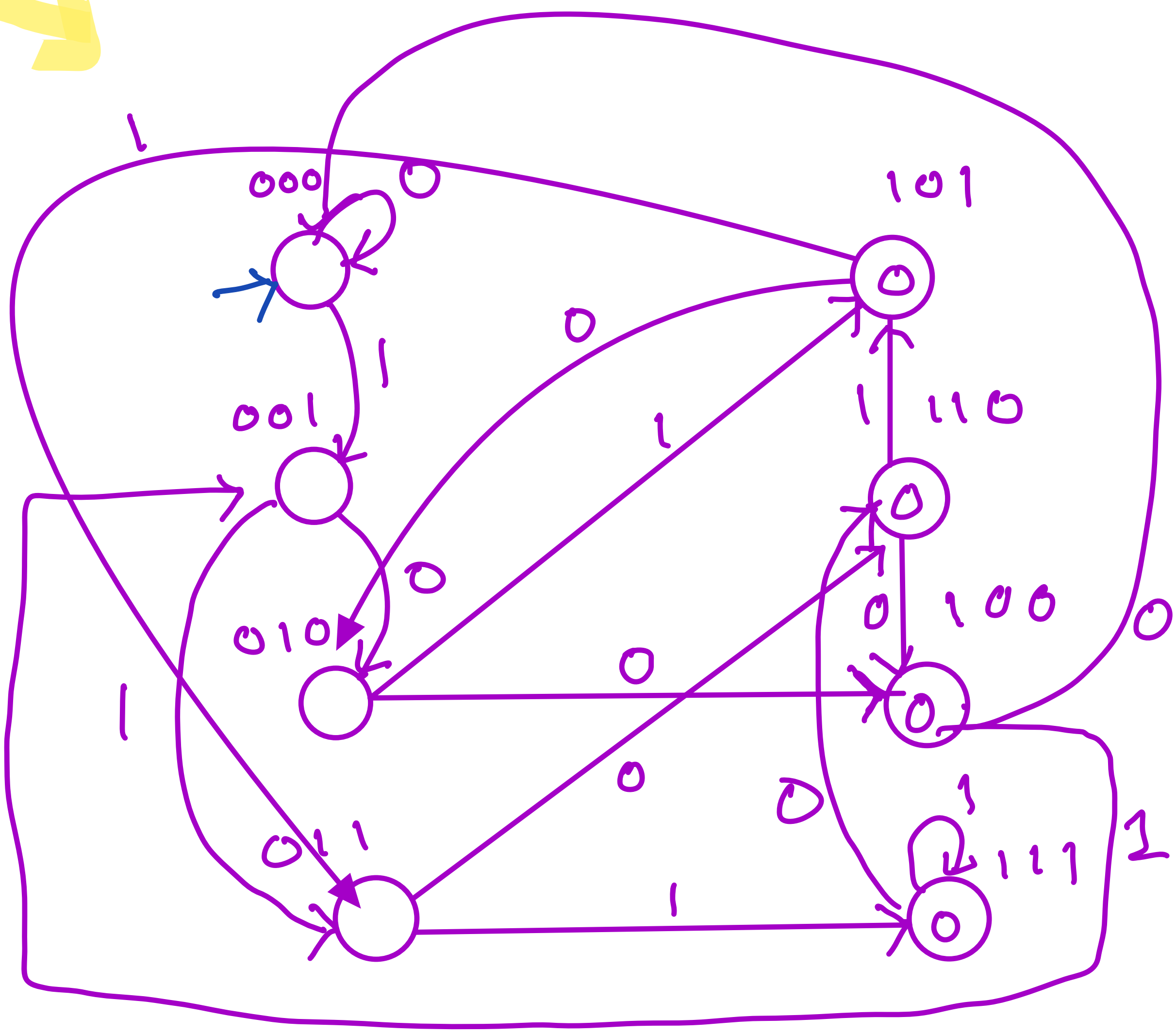


Example 1 Let $L = \{ w \in \{0+1\}^* \mid$ the third symbol of w from the left is 1 $\}$

Example 2 Design a DFA for L^R.



DFA for L



Theorem Prove that it is not possible to design a DFA with ≤ 7 states that accept L^R.

Proof

Suppose that there exists a DFA with < 8 states that accepts L^R.

Consider the following strings

Green strings	red strings
000	100
010	110
001	101
011	111

Since there are atmost 7 States, at least two of the above string goes to the same state after the automata finish scanning the input.

<u>Q1</u> is it possible that a green string and a red string goes to the same state.

<u>Ans:</u> NO. green strings must go to a non-accepting state

and a red string must go to an accepting state

<u>Q2</u> is it possible that two green string goes to the same state ?!

<u>Ans</u> NO,

take any two string x, y from green set.

x, y differs in at least one bit

→ if the second bit is different, then consider the strings $x0$, $y0$

they cannot go to the same state as one of them is in 2^k and the other is not

→ if the third bit is different

Consider the strings

$x00$ and $y00$

one of them $\in L^R$ and the other $\notin L^R$.

Q3 Can two strings from Red Set go to the same State?

Ans NO, Take two strings x, y

$\rightarrow$ second bit is different, take $x0$, $y0$

$\rightarrow$ Last bit is different

$x00$, $y00$

→ Let M be a DFA. The set of strings accepted by M is denoted by L(M).

→ A language L, is called regular if there exists a DFA M such that $\boxed{L(M) = L}$

In the previous lecture, we have said that regular languages are those which can be represented as regular expressions.

Later we will show that, for every regular expression, there is an equivalent DFA.

Closure properties of regular languages

(1) Every language with finite number of strings is always regular.

(2) If L is a regular language, then so is L^c.

(3) If L_1 & L_2 are regular languages, then so is $L_1 \cup L_2$ and $L_1 \cap L_2$

(4) If L_1 & L_2 are regular languages, then so is $L_1 \cdot L_2$.

(5) If L_1 is a regular language then so is L_1^*

Proof

(2) If L is regular, so is L^c.

Proof: Let L is regular.

Therefore there exists a DFA $M = (Q, \Sigma, q_0, \delta, F)$ that accepts L.

We construct a DFA M' from M as follows

$M' = (Q, \Sigma, q_0, \delta, Q \setminus F)$

claim $L(M') = L^c$

(3) L_1, L_2 regular $\Rightarrow$ $L_1 \cup L_2$

Proof Let $M_1 = (Q, \Sigma, \delta_1, q_1, F_1)$

Let $M_2 = (R, \Sigma, \delta_2, r_1, F_2)$

We define M as follows

$$M = (Q_1 \times Q_2, \Sigma, \delta, (q_1, r_1), F)$$

Define

$$\delta(\langle q, r\rangle, x) = \langle q', r'\rangle \text{ if}$$

$$\delta_1(q, x) = q' \text{ and}$$

$$\delta_2(r, x) = r' \text{ for all } x \in \Sigma$$

$$F = \{\langle q, r\rangle \mid q \in F_1 \text{ or } r \in F_2\}$$

Claim $L(M) = L_1 \cup L_2$

Proof Homework.

(4) For L_1, L_2 regular, then so is $L_1 \cap L_2$

~~Proof~~

The construction of the DFA is similar to the DFA for $L_1 \cup L_2$ other than the final state. The final state defined in this case is

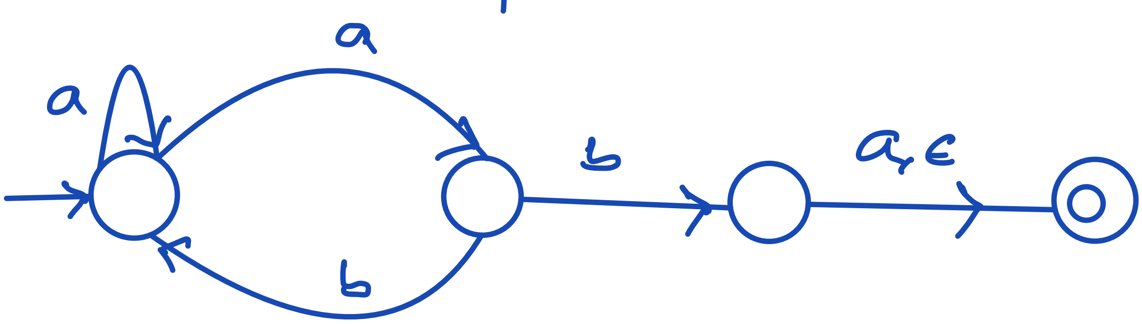
$$F = \{\langle q, r\rangle \mid q \in F_1 \text{ and } r \in F_2\}$$

(5) L_1, L_2 regular, so is $L_1 \cdot L_2$

Before we proceed to proof the above fact, we introduce a 'more relaxed' variation of finite automaton.

Non deterministic finite automaton.

An warm-up example



Although at a first glance, the above automaton looks like a DFA, there are some

Significant differences.

→ multiple transitions from one state (0, 1 or many)

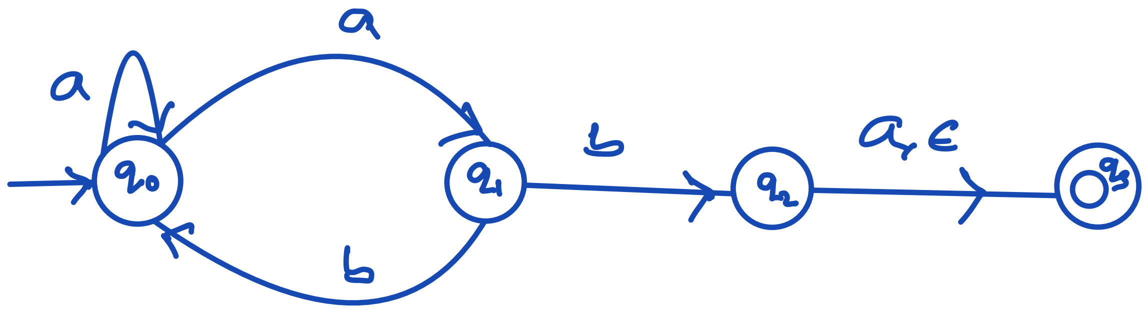
→ allowed to move from one State to another without reading an input symbol

→ Accept an input string if some path leads to a final state.

→ The machine may 'hang' for an input that must be rejected.

Example

What happens if the string ab is given as input in the Automaton given in the warm-up examples !!



Possible end states for ab

(1) q_0 $(q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_0)$

(2) q_2 $(q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2)$

(3) q_3 $(q_0 \xrightarrow{a} q_1 \xrightarrow{b} q_2 \xrightarrow{\epsilon} q_3)$

(4) hangs at q_1

one of the above path reaches to a final state.

therefore, the string ab is accepted.

$\Rightarrow$ $ab \in L(M)$.

what about abb ??

possible end states for abb

(1) it hangs at q_0

(2) it hangs at q_0

(3) it hangs at q_2

(4) it hangs at q_3

None of the above possibilities ends reading the entire input & ends in a final state.

Hence $abb \notin L(M)$

Definition (NFA)

A non-deterministic finite automaton is a 5-tuple $\langle Q, \Sigma, \delta, q_0, F)$ where

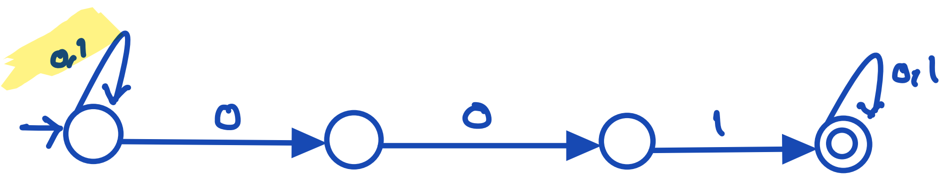
$Q \leftarrow$ A set of states

$\Sigma \leftarrow$ a finite alphabet

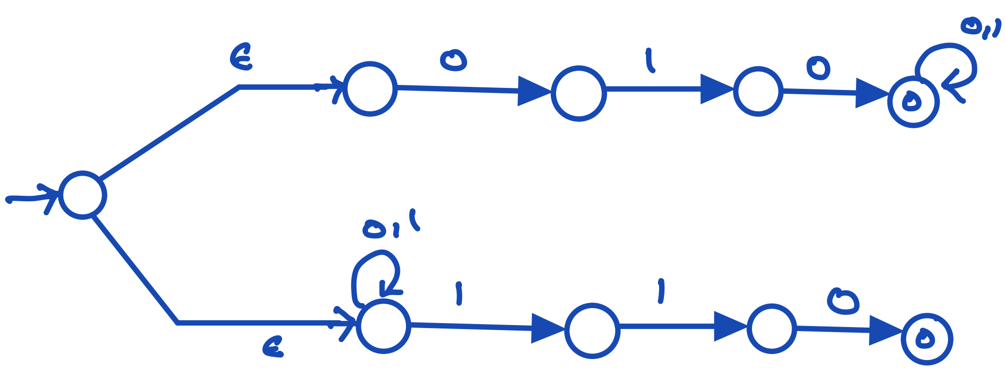
$$\delta : Q \times (\Sigma \times \{\varepsilon\}) \rightarrow 2^Q$$

where 2^Q denotes power set of Q.

Example Find an NFA that accepts the set of binary strings having a substring 001



Example Find an NFA that accepts the set of binary strings beginning 010 & ending with 110



Facts: Given NFA's for two languages L_1 & L_2

(1) Construct an NFA for $L_1 \cup L_2$

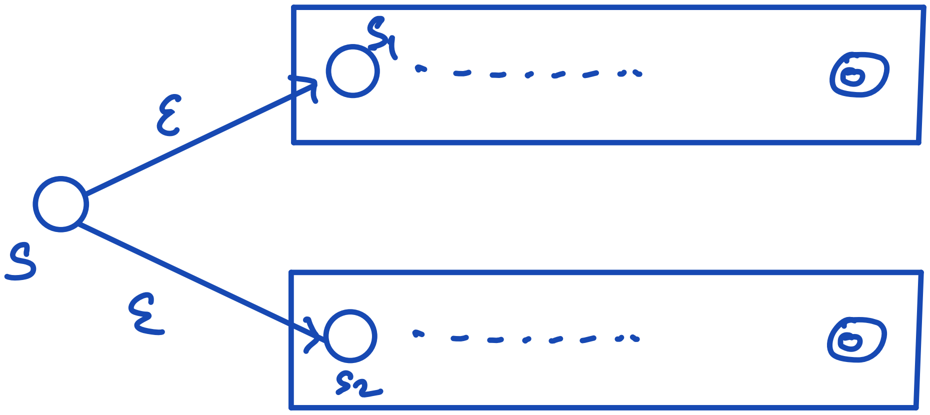
(2) Construct a NFA for $L_1 \cdot L_2$

(3) Construct an NFA for L_1^*

Proof ideas are given below. For the detailed proof, see book "Lewis & Papadimitriou", Page 75

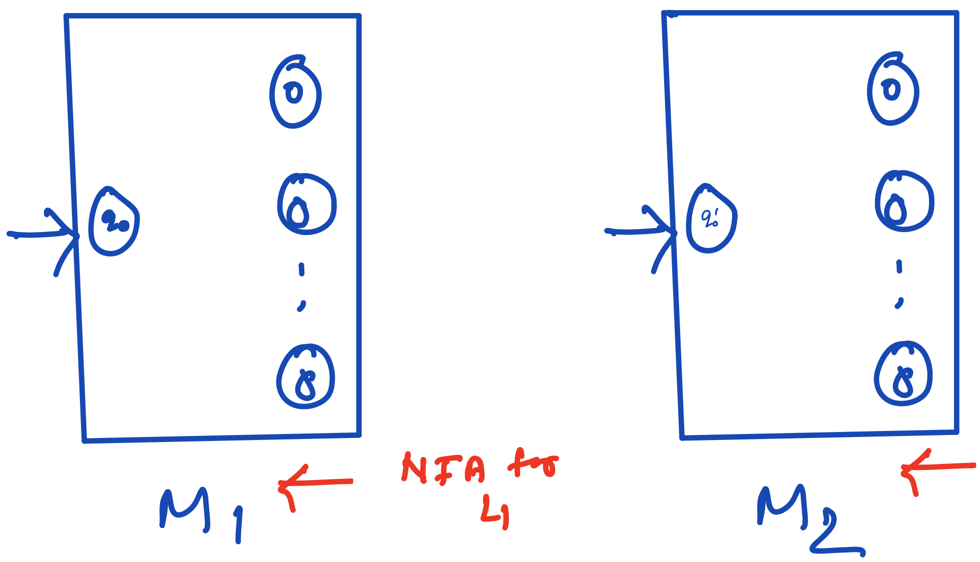
Sol^n 1 NFA for $L_1 \cup L_2$

(1) Add a new start state S and add transitions $\delta(S, \epsilon) = \{S_1, S_2\}$



NFA for $L_1 \cup L_2$

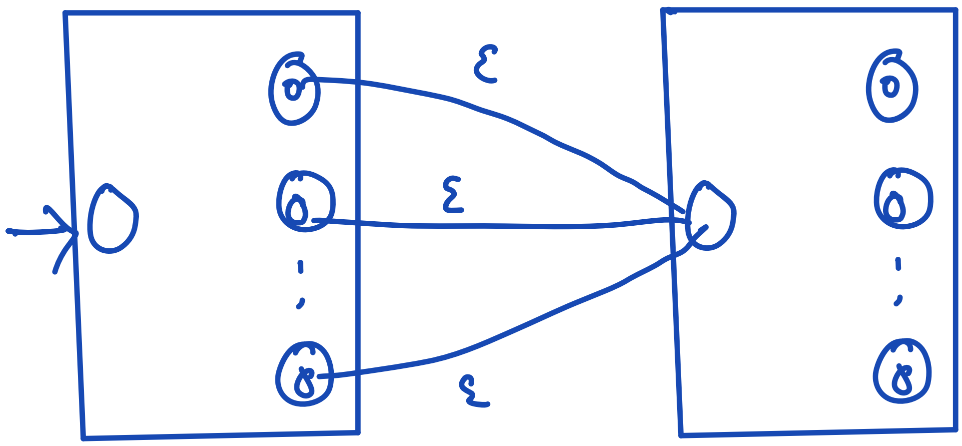
$\underline{\text{Sol}^n}$ 2. NFA for $L_1 . L_2$



NFA for L_1

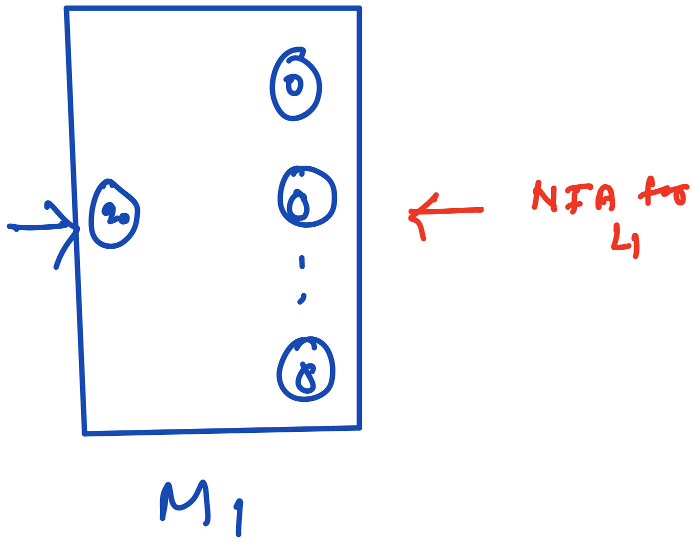
NFA for L_2

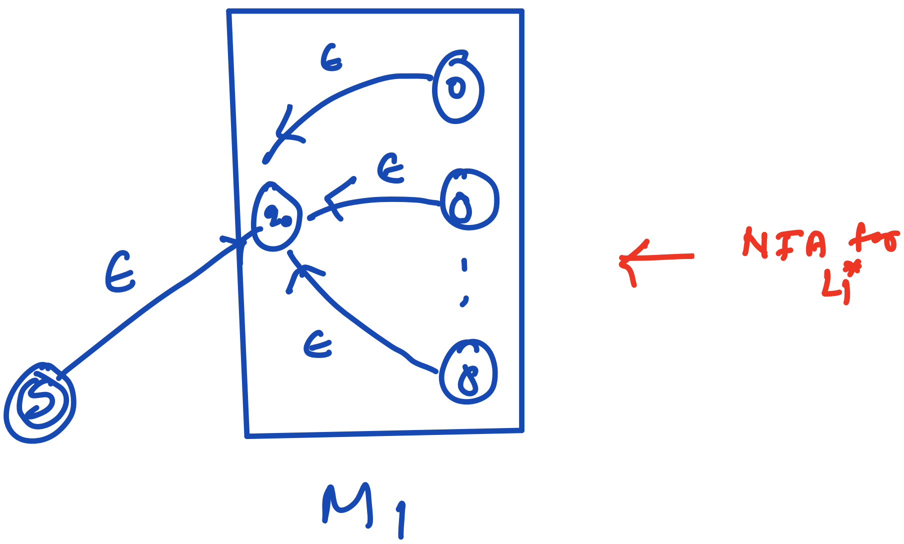
(1) for every final state $f \in M_1$, add transitions $\delta(f, \epsilon) = q_0'$



NFA for $L_1 \cup L_2$

Sol^{n} 3





Summary of the topics till lecture 3

* Discussions on alphabet, string, language
* String operations $\cup, \cdot, *$
* Regular expression.
* Deterministic finite automaton as language acceptor.
* Non-deterministic finite automaton.