DISCRETE MATHEMATICS

LECTURE 11

Plan:

Non-Deterministic Finite Automata

(NFA)

Reference:

Section 1.2 of book:

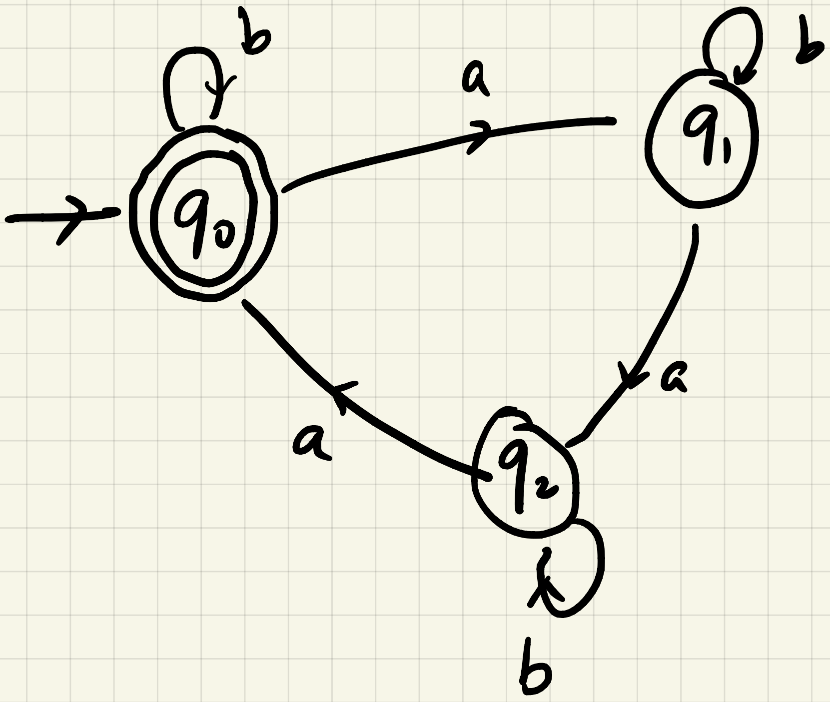
Introduction to the Theory of Computation

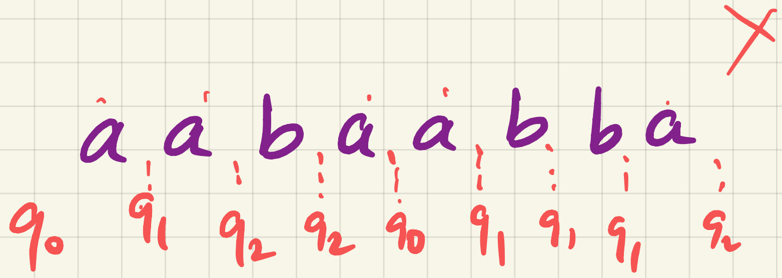
(third edition)

by Michael Sipser

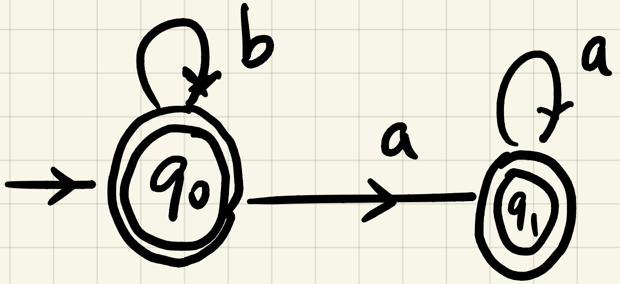
Recap of DFA:

No. of a's is a multiple of 3:

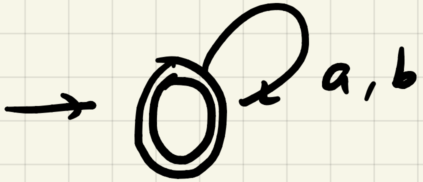




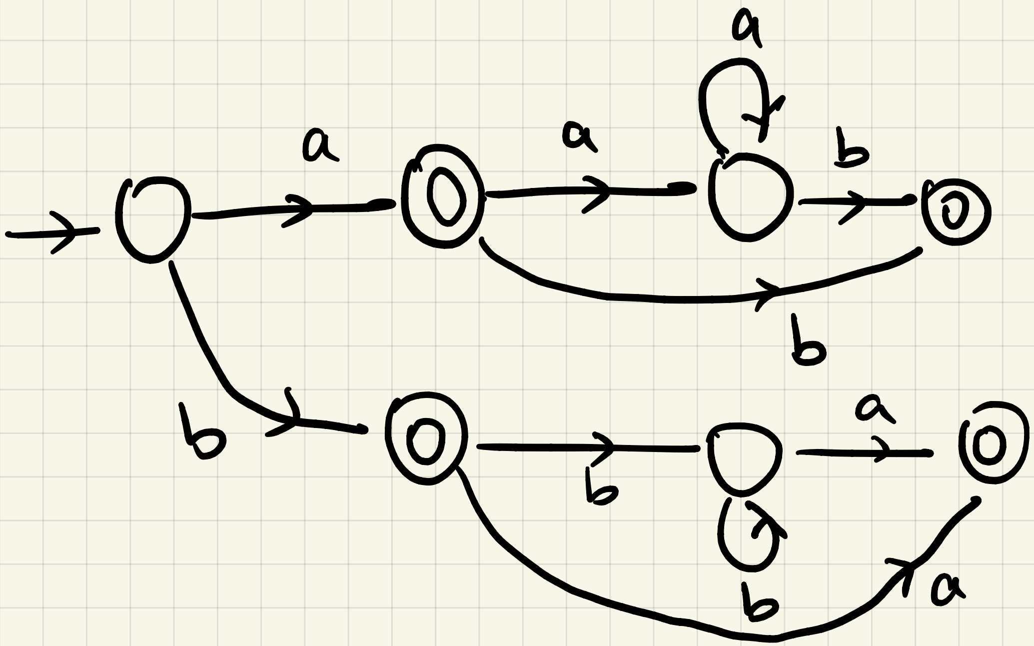
$b^* \quad a^*$

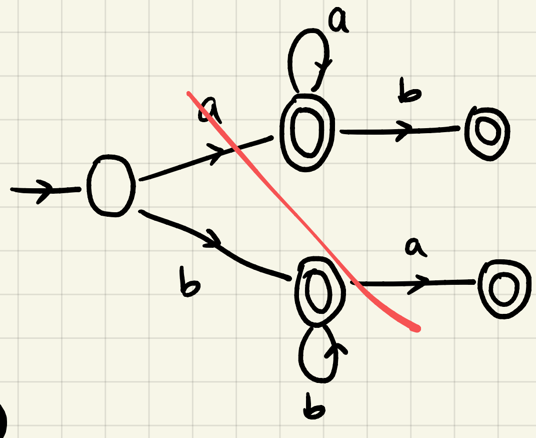


$(a+b)^*$



$b^*a + a^*b$





since it accepts a^*.

Next goal:

$$\text{Regular expressions} \xrightarrow{\text{convert}} \text{DFA}$$

Why is this useful?

Regular expressions

$\downarrow$ convert

DFA

Suppose we have a procedure that takes an expression R and outputs a DFA $\mathcal{D}(R)$

Algorithm to check if a string w contains a pattern given by expression R:

- 1. Consider expression $\Sigma^* R \Sigma^*$
- 2. Build DFA D for $\Sigma^* R \Sigma^*$
- 3. Run the string w on DFA D.
- 4. If D accepts w then w contains pattern R. Else no.

Regular expressions

$\downarrow$ convert

DFA

How to convert reg. expr. to DFA?

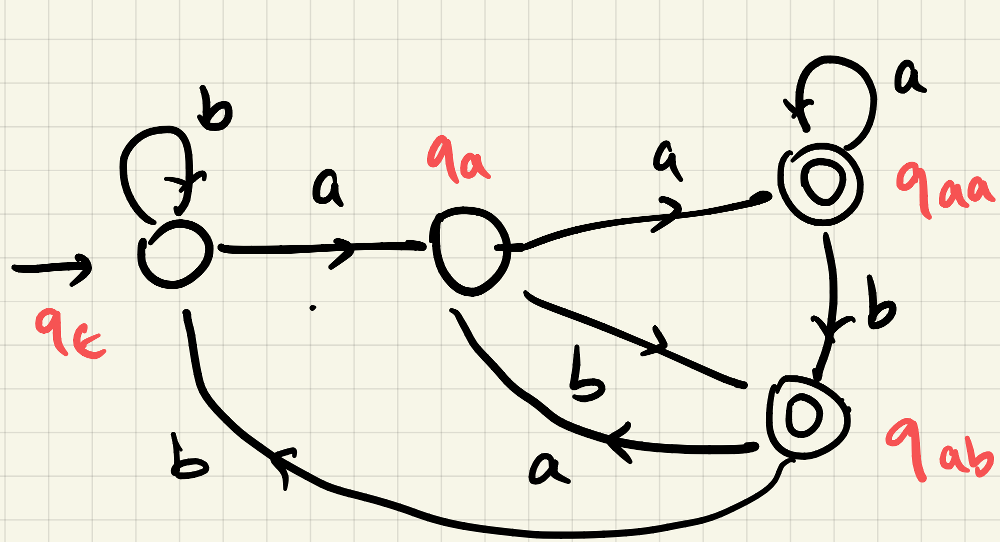
- We have already seen several examples.

Here are some difficult cases.

$a^* b + b^* a$ → already done

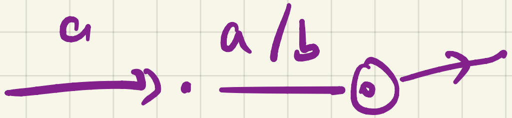
$\Sigma^* a \Sigma \Sigma$

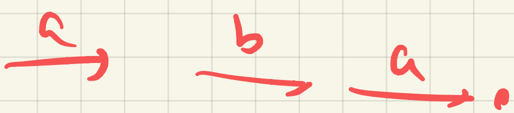
$$\Sigma^* a \Sigma$$

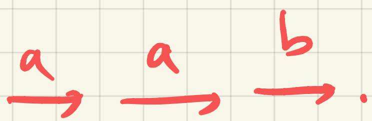


"Remember the last two letters that are seen."

Every transition $p \to \textcircled{q}$ leading to an accepting state is from a state 'p' that is reached on seeing an 'a'.







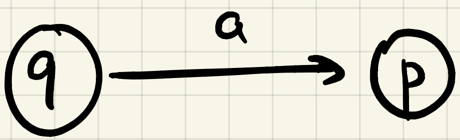
To make the conversion smoother, we will go via another intermediate formalism called

Non-deterministic Finite Automata (NFA)

$$
\text{Regular expressions} \xrightarrow{\text{convert}} \text{NFA} \xrightarrow{\text{convert}} \text{DFA}
$$

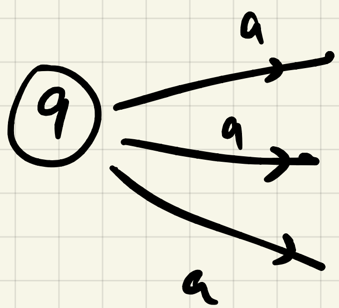
Non-deterministic Finite Automata (NFA): relaxations from DFA:

DFA: single initial state



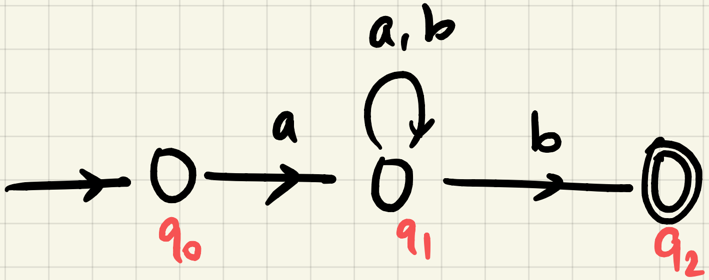
atmost one state.

NFA: multiple initial states.



multiple states on 'a'
from 'q' possible.

Example 1: NFA for $a \Sigma^* b$



ab : $q_0 \; a \; q_1 \; b \; q_1$.

$\qquad\qquad\qquad\qquad - b - q_2$

abb : $q_0 \; a \; q_1 - b - q_1 - b - q_1$

$\qquad\qquad\qquad\qquad\qquad\qquad \hookrightarrow b - q_2$

$\qquad\qquad\qquad\quad \hookrightarrow b - q_2 - b \; \times$

There exists a run going to an accept state.

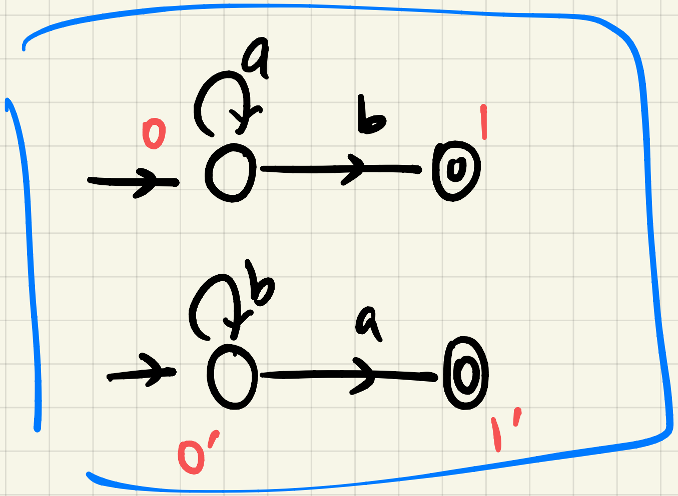
aba

$q_0 - a - q_1 - b - q_1 - a - q_1$

$\qquad\qquad \hookrightarrow b - q_2 \; \times$

Not accepted since there is no run leading to an accept state.

Example 2: $a^* b + b^* a$



→ a single NFA

bba

$0 — 1 \cdots \times$

$0' — 0' \cdots 0' \cdots \boxed{1'}$

accepting.

aba

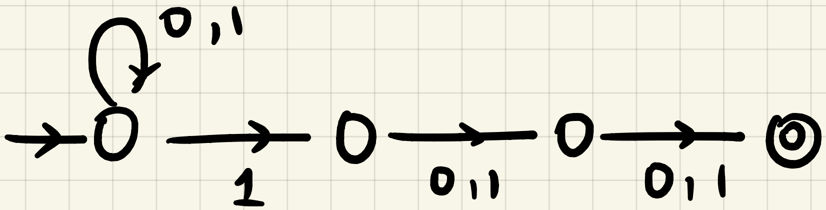
$0 \quad 0 \quad 1 \quad \times$

$0' \quad 1' \to \times$

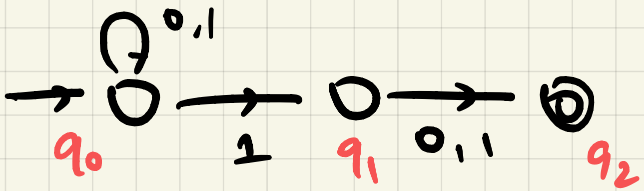
rejected

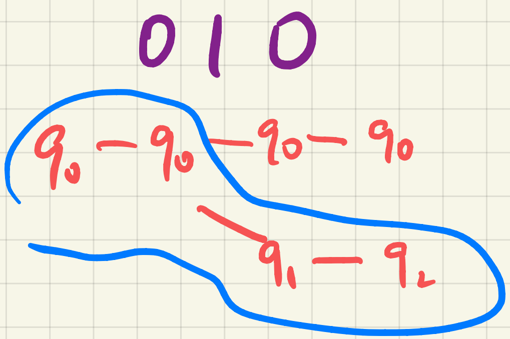
Example 3:

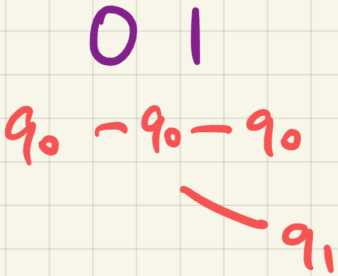
$\Sigma = \{0,1\}$ $\quad\quad$ $\Sigma^* 1 \Sigma\Sigma$



$\Sigma^* 1 \Sigma$:







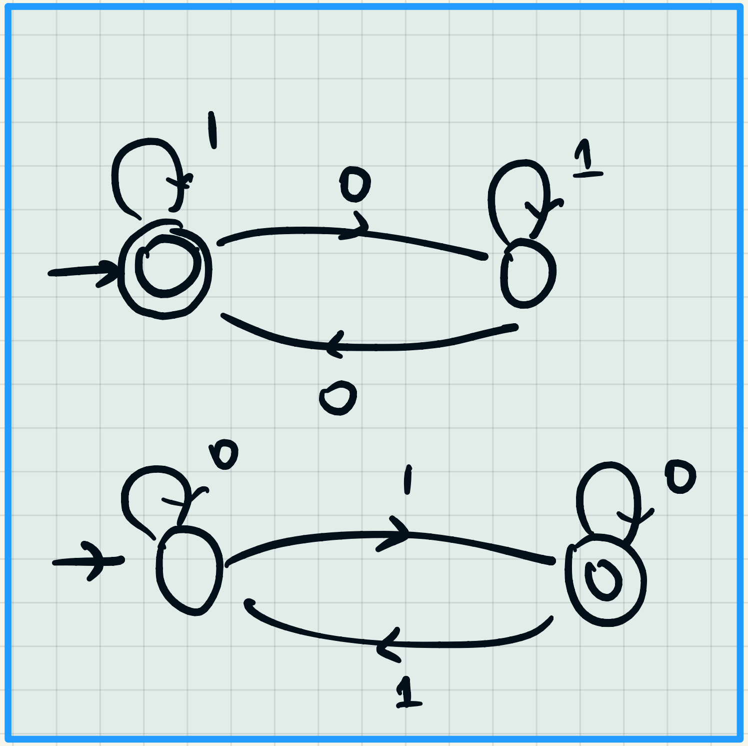
Example 4: Give NFA / DFA for the following languages over $\{0, 1\}$

4.1. Even no. of 0's or odd no. of 1's

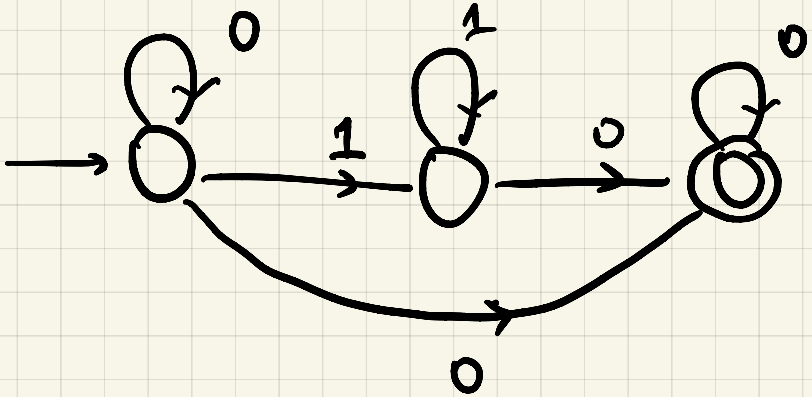
4.2. $0^* 1^* 0 0^*$

4.3. $\{ w \mid w$ ends with $00 \}$

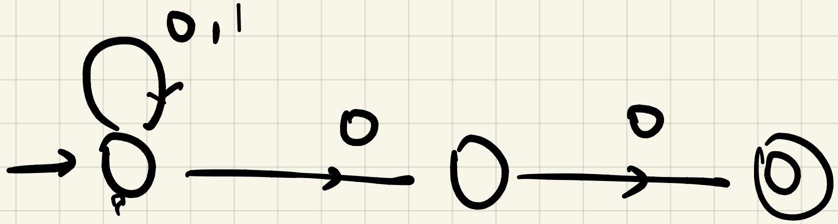
4.1. Even no. of 0's or odd no. of 1's



$0^* \; 1^* \; 0 \; 0^*$



$\{ w \mid w \text{ ends with } 00 \}$



Non-deterministic Finite Automata NFA : Syntax :

$$(Q, \Sigma, Q_0, \Delta, F)$$

Q : Finite set of states

Σ : alphabet

Q_0 : initial states

Δ : Transition relation (next slide)

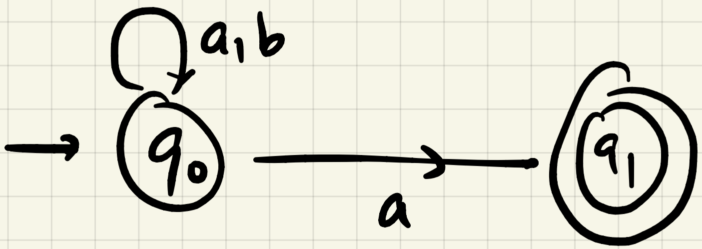
F : Final states

Transition relation Δ:

$$\Delta \subseteq Q \times \Sigma \times Q$$

is a subset of $Q \times \Sigma \times Q$

Eg:



Δ: (q_0, a, q_0)

(q_0, b, q_0)

(q_0, a, q_1)

Non-deterministic Finite Automata NFA : Semantics

- Run of an NFA on a word $w_1 w_2 \cdots w_n$ is a labeling of states:

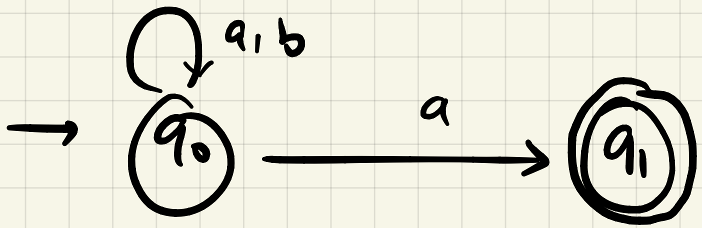
$$q_0 \xrightarrow{w_1} q_1 \xrightarrow{w_2} q_2 \cdots \xrightarrow{w_n} q_n$$

such that $(q_i, w_i, q_{i+1}) \in \Delta$.

- Run is accepting if q_n is accepting.

Language of NFA: is the set of all words w s.t. there exists at least one accepting run.

eg:



- Word bbb has no accepting run. All runs ends at q_0.
- Word bba has one accepting run.
- Word baa has one accepting run.

Next:

Regular expressions

$\downarrow$ Convert

NFA

$\downarrow$ convert

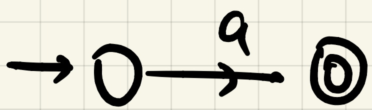
DFA

Converting reg. expressions to NFA.

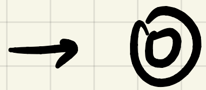
Regular expressions to NFA: algorithm:

Basic regular expressions: $a \in \Sigma$, ϵ , Φ

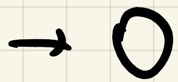
a :



ϵ



Φ



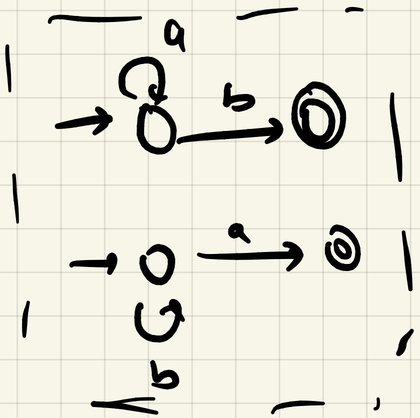
$R_1 + R_2$?

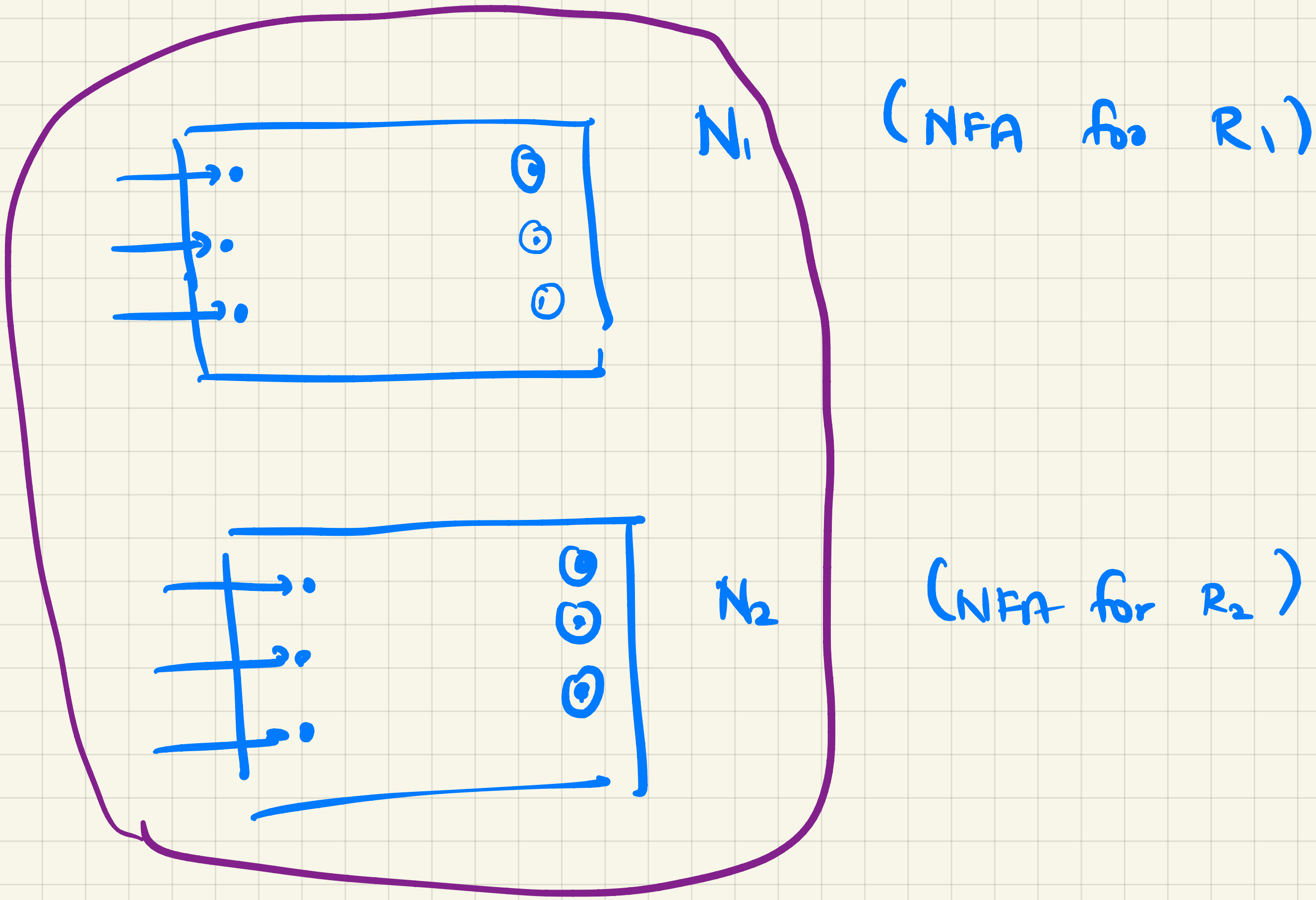
Suppose we have constructed NFAs for R_1 and R_2.

What is the NFA for $R_1 + R_2$?

$a^* b$: R_1

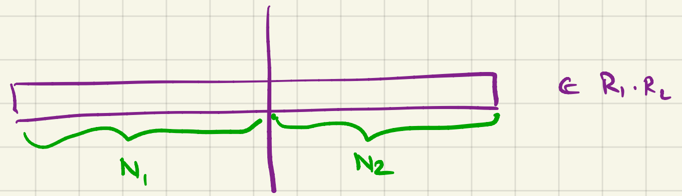
$b^* a$: R_2

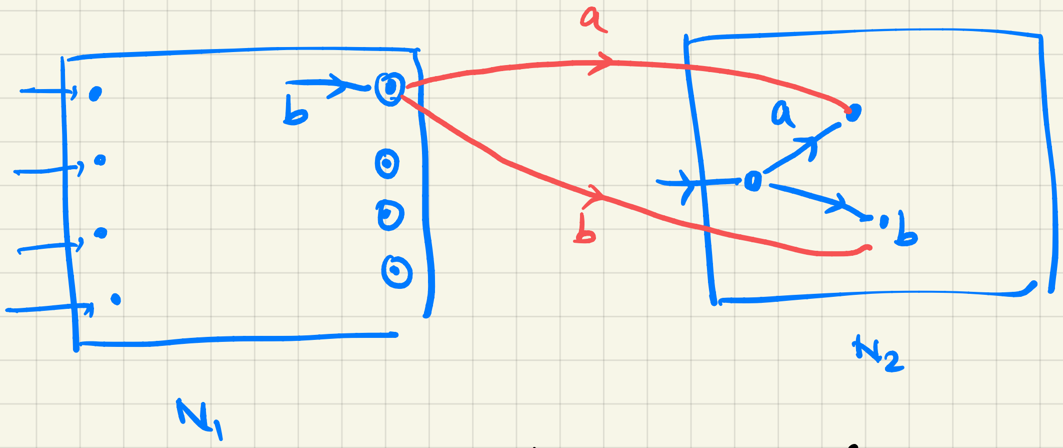




as one NFA N (for $R_1 + R_2$)

$R_1 \cdot R_2$





1. From every final state 'q_1' of N_1, add a transition

   $q_1 \xrightarrow{a} p$ for all transitions

   $p_0 \xrightarrow{a} p$

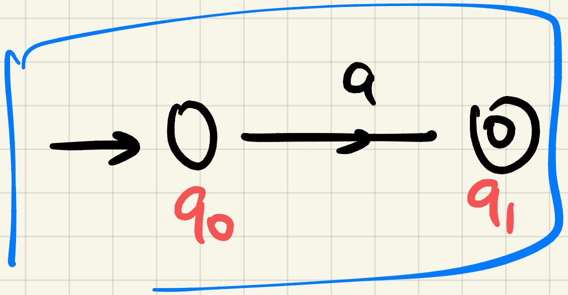
   where p_0 is an initial state of N_2.

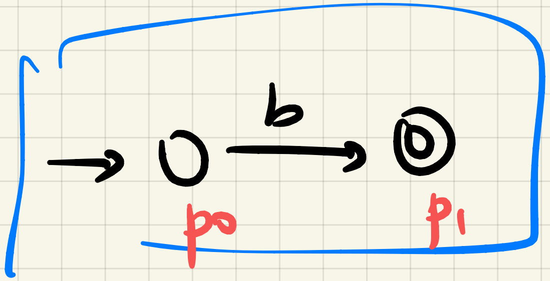
2. Make all final states of N_1 as non-final if ϵ is not in N_2.

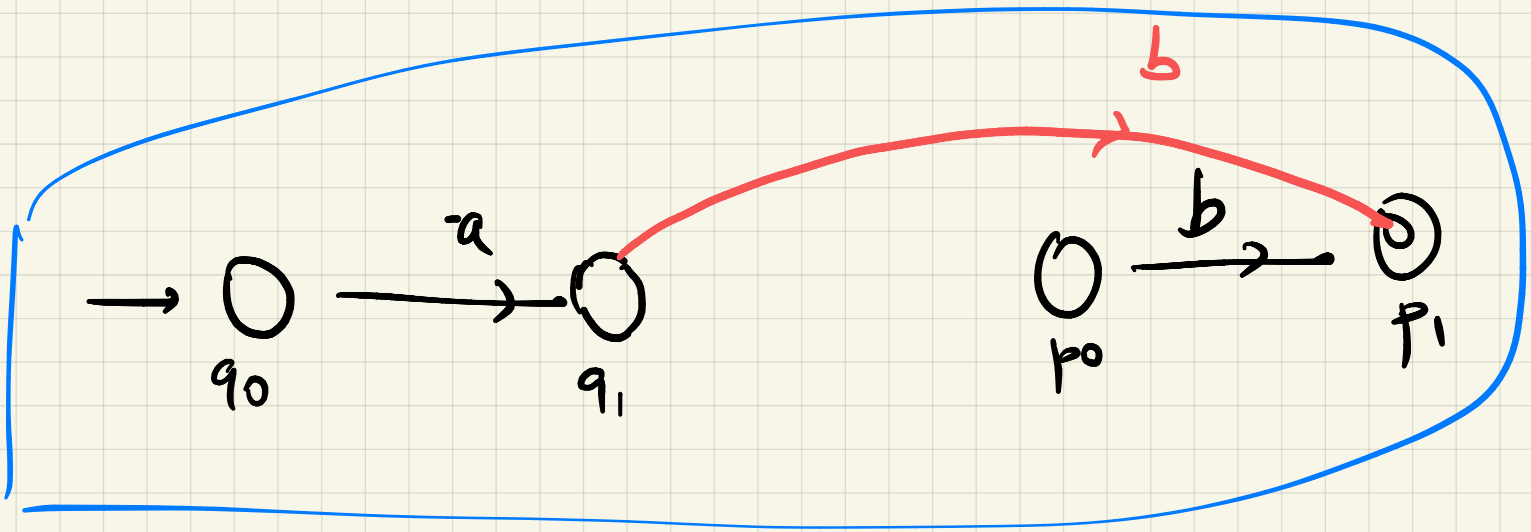
3. Make all initial states of N_2 as non-initial if ϵ is not present in N_1

R_1: a

R_2: b







Summary:

→ NFA

→ Part of translation of Reg Expr. to NFAs.

Reg Exp

↓

NFA

↓

DFA

R_1: a $\qquad$ R_2: $b^* + ab$