Introduction.

- Problem: A definition of what is to be done.
- Problem ≡ Function
- Computable: A function or problem is computable iff there exists an algorithm that describes how to solve the problem in finite amount of time.

- Out of all problems only a tiny fraction is computable.
- Most of the time in ToC, we study how to build a machine, given a language, which recognizes the language.
  Recognizing language is same as solving computational problems.

Ex. $L = \{a^2, a^3, a^5, a^7, \ldots\} = \{a^p \mid p \text{ is prime}\}$.
So, say there's a machine which recognizes this language, then it basically recognizes prime number.

- Automata: Given a string, outputs, Yes or No (or Accepted or Rejected).

Basics

* Symbols: The most atomic (indivisible) entity, of length one.

- Alphabet: Non-empty finite set of symbols. (Σ)

Ex. English Alphabet: {a, b, c, ... z}
Ex. C language: ASCII Chars {A...Z, a...z, 0...9, +, -, ...}.
Ex: Binary Alphabet: {0, 1}, {a, b}, etc.

- Strings (over an Alphabet): A string over alphabet Σ is a finite sequence of symbols, where each symbol belongs to Σ.
  - Order matters. Ex: $ab \neq ba$.
  - Repetition matters. Ex: $aa \neq aaa$.
  - Over an alphabet Σ, there can be ∞ no. of strings (of finite length).

- Null String (λ or ϵ): Over any alphabet, a null string is a string without any alphabet symbol, i.e. $|\epsilon| = 0$.

Ex. Σ = {a, b}. Which of the following are strings over Σ?

✓(a) a	✓(d) ϵ	(g) ϕ
✓(b) b	✓(e) aaaaa	(h) {}
✓(c) ab	(f) aaaaa...	(i) abacba

(f) is not finite.
(g) & (h) are empty set
(i) 'c' isn't a symbol in alphabet.

Ex. ab vs {ab}

ab → String

{ab} → set containing string.

Ex. ∈ vs {∈} vs ∅

∈ → Null String

{∈} → Set containing null string

∅ → Empty set

- **Reversal of String:** Reverse of a string ω is denoted by ω^R, and for
  $\omega = s_1 s_2 \dots s_n$ is given by
  $\omega^R = s_n s_{n-1} \dots s_2 s_1$.

- **Palindrome:** $\omega = \omega^R \leftrightarrow \omega$ is palindrome.

- **Concatenation of strings:** Concatenation of two strings, ω_1 & ω_2 is given by $\omega_1 . \omega_2$.
  $\omega_1 = a_1 a_2 \dots a_n$ $\omega_2 = b_1 b_2 \dots b_n$.
  $\omega_1 . \omega_2 = a_1 a_2 \dots a_n b_1 b_2 \dots b_n$
  & $|\omega_1 . \omega_2| = |\omega_1| + |\omega_2|$
  over alphabet Σ.

Ex. $\Sigma = \{a, b, c\}$ $u = aaa$ $v = abccb$ $\omega = a$.

(1) uvw aaaabccba

(2) u^2v aaaaaabccb

(3) uvw^2 aaaabccbaa.

- For any string ω:
  $\omega.\epsilon = \epsilon.\omega = \omega$
  & $|\omega| = |\omega\epsilon| = |\epsilon.\omega|$

- For any string ω
  $\omega^n = \omega\omega\omega\ldots\omega$ {n times}
- $\omega^0 = \lambda$
  So, $\epsilon^0 = \epsilon$
- Any string power 0 is ϵ.

* - Prefix (of a string): Any substring of ω of length
  - 0 or more from the start.

- Suffix (of a string): Any substring of ω of length 0 or more till the end.

- Substring: Any contiguous section of ω.

- Proper Prefix: Any prefix of ω other than ω.
- Proper Suffix: Any suffix of ω other than ω.

Ex. $\Sigma = \{a, b\}$ $\omega = abaaba$.

Prefixes: ϵ, a, ab, aba, abaa, abaab, abaaba.

Suffixes: ϵ, a, ba, aba, aaba, baaba, abaaba.

substrings = ${}^7C_2 + 1$ (max. possible substrs. of $|\omega| = 6$)

0 len substr: ϵ

1 len substr: a, b

2 len: ab, ba, aa

3 len: aba, baa, aab,

4 len: abaa, baab, aaba

5 len: abaab, baaba

6 len: abaaba.

* Given a string w:

- u is called prefix of w iff $\exists v,\ uv = w$
- u is called suffix of w iff $\exists v,\ vu = w$
- u is called substring of w iff $\exists x, y,\ xuv = w$

Ex. Given a string w, $|w| = n$:

- No. of prefixes $n+1$
- No. of suffixes $n+1$
- No. of substrings

  Min. (all n symbols are same) $= n+1$

  Max. (all n symbols are distinct) $= {}^{n+1}C_2 + 1$

  $\Rightarrow \frac{n(n+1)}{2} + 1$

  $\therefore\ n+1 \leq \#\text{substrings} \leq {}^{n+1}C_2 + 1$

- Subsequence (of a string): A subsequence of a string can be formed by deleting 0 or more symbols without changing the order of other symbols.

* Set of all strings (over an alphabet): All strings of length 0, 1, 2, ... that can be created using symbols from Σ.
(Denoted using Σ^*)

Ex. $\Sigma = \{a, b\}$
$\Sigma^* = \{\epsilon, a, b, aa, ab, ba, bb, aaa, aab, aba, abb, baa, bab, bba, bbb, ...\}$

* Language (over an alphabet): Any set of strings over Σ is called language.
So, language over Σ is a subset of Σ^*.

∴ Any subset L of Σ^* is called language over Σ.

$$L \subseteq \Sigma^*$$

Ex. $\Sigma = \{a, b\}$. Which of the following are languages over Σ?

✓(a) $\{aa\}$
(b) $\{a, b, c\}$
✓(c) Σ
✓(d) $\{\}$
✓(e) $\{\epsilon\}$
✓(f) $\{\omega \mid \omega = a^{2n}, n \geq 0\}$
✓(g) $\{a\}$
✓(h) $\{a^i b^{2^i} \mid 0 \leq i \leq 10\}$

Ex. $\Sigma = \{a, b\}$ $L_1 = \{a^n b^m \mid n \geq 0, m \geq 0\}$
$L_2 = \{a^n b^n \mid n \geq 0\}$

(A) $L_1 \subseteq L_2$
(B) $L_1 \subset L_2$
(C) $L_2 \subseteq L_1$ ✓
(D) $L_2 \subset L_1$ ✓

- Since languages themselves are sets, all basic set operations, such as union, intersection, difference, sym. difference, etc. are valid on languages

- Reversal (of Language): Reversal of a language

$$L = \{ w_1, w_2, w_3, \ldots \}$$

is given as:

$$L^R = \{ w_1^R, w_2^R, w_3^R, \ldots \}$$

- Palindrome (Language): $L = L^R \leftrightarrow L$ is palindrome

Ex. If L is Palindrome then $\forall w \in L$ w is palindrome?

False. Ex. $L = \{ab, ba\}$

$L^R = \{ba, ab\}$

$\therefore L = L^R$

but $\exists w \in L$ w is not pal.

- Concatenation (of Language): The language concatenation of L_1 & L_2, is the set:

$$L_1 . L_2 = \{ wx \mid w \in L_1, x \in L_2 \}$$

Ex. $L_1 = \{a, ab\}$ $L_2 = \{ba, ab, \epsilon\}$

$L_1 . L_2 = \{aba, aab, abba, abab, a, ab\}$

Ex. $L = \{a^n b^n \mid n \geq 1\}$

L^2 ?

$L^2 = L \cdot L = \{a^n b^n \mid n \geq 1\} \cdot \{a^m b^m \mid m \geq 1\}$

$\Rightarrow \{a^n b^n a^m b^m \mid n \geq 1, m \geq 1\}$

- For any language, concatenation with empty language is empty language, i.e.

$$L \cdot \phi = \phi \quad \& \quad \phi \cdot L = \phi$$

- $$L \cdot \{\epsilon\} = L \quad \& \quad \{\epsilon\} \cdot L = L$$

i.e. $\{\epsilon\}$ is identity element for language concatenation.

Ex. If $L \cdot M = L$ then $M = \{\epsilon\}$?

No. Say, $L = \phi$, then for all M, $L \cdot M = L$.

Say $L = \Sigma^*$ then for all M, $L \cdot M = L$.

$L = \{a^{2n} \mid n \geq 0\}$ $M = \{a^2, \epsilon\}$, then $L \cdot M = L$

- Power of Language

$\Rightarrow$ $L^0 = \{\epsilon\}$

$L^n = LL \ldots L$ {n times} $n \geq 1$

- Kleene Star (of Language): for any language L,

$L^* = L^0 \cup L^1 \cup L^2 \ldots$

$\Rightarrow \bigcup_{i \geq 0} L^i$

- Kleene Plus (of language): for any language L,

$L^+ = L^1 \cup L^2 \cup L^3 \ldots$

$\Rightarrow \bigcup_{i > 0} L^i$

Ex. $\Sigma = \{a\}$, $L = \{aa\}$

$L^0 = \{\epsilon\}$ $L^1 = \{aa\}$ $L^2 = \{a^4\}$ $L^3 = \{a^6\}$ ...

$\therefore L^* = \{a^{2n} \mid n \geq 0\}$

$L^+ = \{a^{2n} \mid n \geq 1\}$

Ex. $L = \phi$

$L^0 = \{\epsilon\}$ $L^1 = \phi = \{\}$ $L^2 = \phi$ $L^3 = \phi$

$\therefore L^* = \{\epsilon\}$

$L^+ = \phi$

Ex. $L = \{\epsilon\}$

$L^0 = \{\epsilon\}$ $L^1 = \{\epsilon\}$ $L^2 = \{\epsilon\}$ ...

$\therefore L^* = \{\epsilon\}$

$L^+ = \{\epsilon\}$.

Ex. $L = \{ab, a\}$

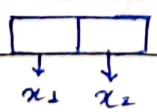
$L^0 = \{\epsilon\}$ $L^1 = \{ab, a\}$ $L^2 = \{abab, aba, aab, aa\}$

$L^3 = \{ababab, ababa, abaab, \ldots\}$

So, L^n consists of strings each of which consists of n parts. Each part is a string $x_i \in L$.

i.e.

$$L^n = \{x_1 x_2 \ldots x_n \mid \forall i \; x_i \in L\}$$

So, $L^2 = \{$  $\mid x_1, x_2 \in L\}$

$\Rightarrow \{aba, abab, aaab, aa\}$

Ex. $L = \{ab, a\}$. Which belongs to L^*?

(a) a ✓
(b) ϵ ✓
(c) ababaaaba.
(d) abaabbabba.

(c) can be broken as: ab|ab|a|a|ab|a.
(d) can't be broken down in n such parts.

* Power of Alphabet

Since alphabet is also a language by defn, we can define the power of alphabet as well.

$\Sigma^1 = \Sigma$

$\Sigma^2 = \Sigma\Sigma$

⋮

& $\Sigma^0 = \{\epsilon\}$

So, Σ^n gives the set of all n length strings over Σ.

$\therefore \Sigma^* = \Sigma^0 \cup \Sigma^1 \cup \Sigma^2 \ldots$ {set of all strings over Σ}
& Σ^+ is set of all non-emply strings over Σ.

Ex. If $L^* = \Sigma^*$, then $L = \Sigma$?
Yes.

* Complement of Language.

Complement of a language L is given as:

$$\bar{L} = \Sigma^* - L$$

where $(L \subseteq \Sigma^*)$

Ex. FInd complement of:

$L = \{a^n b^n \mid n \geq 0\}$.

$$\bar{L} = \{a^n b^m \mid n \neq m\} \cup \{\omega ba x \mid \omega, x \in \Sigma^*\}$$

Ex. Complement of $L = \{ww \mid \omega \in \Sigma^*\}$.

$$L^c = \{\omega \mid |\omega| \text{ is odd}\} \cup \{\omega x \mid \omega \neq x \wedge |\omega| = |x|\}$$

Ex. Complement of $L = \{\omega x \mid \omega, x \in \Sigma^*, \omega \neq x\}$.

$L = \{a, b, aa, \ldots\}$

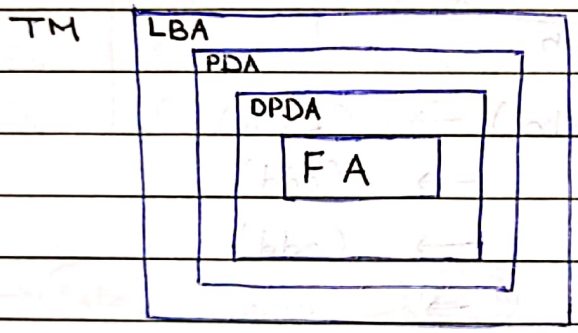
$a = a\epsilon \quad b = b\epsilon \quad aa = aa\epsilon \ldots$

$\therefore L = \Sigma^+ \qquad \bar{L} = \{\epsilon\}$.

Finite Automata

- Alan Turing came up with an Automaton called, "Turing Machine". It has been proven that "a problem is computable iff it is computable by Turing Machine". Hence TM is the most powerful model.

Later in 1950-60, people started to study more restricted models of computation by putting restrictions on TM.



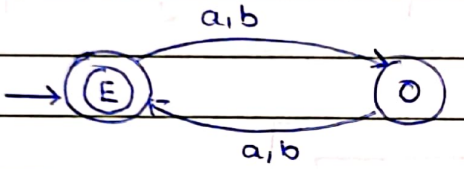
* Finite Automata

FA can be thought of as space-restricted TM.
A FA can be described using the following information:

(1) States
(2) Input / Alphabet
(3) Initial State.
(4) Set of final states.
(5) Transitions.

Ex. Recognize set of all even length strings, over $\Sigma = \{a, b\}$.

We have two states : EvenLen & OddLen.
Initially with 0 symbols, the length is even.
The string should be accepted if in the end the length is even.



(State, inp symbol)	→	(state)
(even, a)	→	(odd)
(even, b)	→	(odd)
(odd, a)	→	(even)
(odd, b)	→	(even)

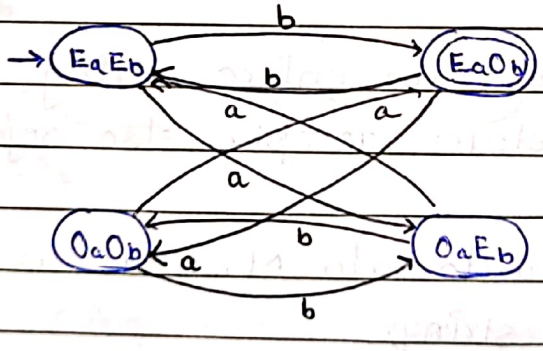
∴ States : {even, odd}.
Initial State : even
Inputs : {a, b}
Set of Final states : {even}
Transition : Given above

- For a machine M, set of all the strings accepted by M is known as language of machine M.
  L(M)

Ex. Recognize set of all strings with even no. of a's & odd no. of b's, over $\Sigma = \{a, b\}$.

- Four States: ~~$\{E_aE_b, O_aO_b, E_bO_b\}$~~ $\{E_aE_b, E_aO_b, O_aE_b, O_aO_b\}$
- Inputs: $\{a, b\}$
- Initial State: E_aE_b
- Final States: $\{E_aO_b\}$

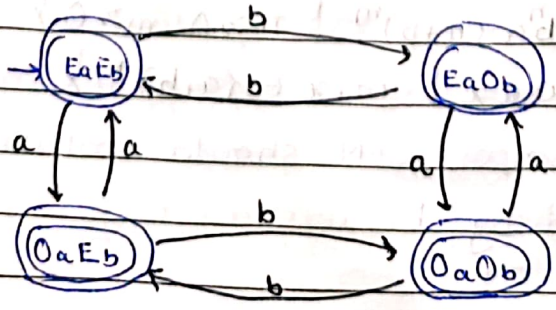
	Curr State	Next state a	Next state b
→	E_aE_b	O_aE_b	E_aO_b
*	E_aO_b	O_aO_b	E_aE_b
	O_aE_b	E_aE_b	O_aO_b
	O_aO_b	E_aO_b	O_aE_b



Ex. Recognize set of all strings with even no. of a's or odd no. of b's over $\Sigma = \{a, b\}$.

Same as above, but with following final states:

$\{E_aE_b, E_aO_b, O_aO_b\}$



- So, a Finite Automata is a tuple of 5 members:

$(Q, \Sigma, q_0, F, \delta)$

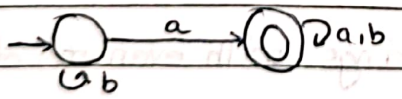
Q: A non-empty finite set of states
Σ: Input alphabet
q_0: Initial state.
F: Set of final states. (could be empty)
δ: Transition function

- Since strings by def^n are of finite length, the input to FA is also of finite length.

- FA, after consuming the entire string, if it is in one of the final states, accepts, else rejects the string.

- Language of finite automata M, L(M), is the set of all and only accepted strings.

Ex.



(1) $L(M) = \{abb\}$
(2) $abb \in L(M)$
(3) $L(M) = \{ b^n a (a+b)^m \mid n \geq 0, m \geq 0 \}$
(4) $L(M) = \{ w a x \mid w, x \in \{a,b\}^* \}$

1 is wrong since L(M) should contain all & only strings accepted by M.

* Deterministic Finite Automata:

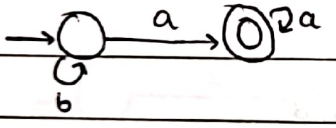
DFA is a FA, from every state, for every symbol there should be exactly one transition.

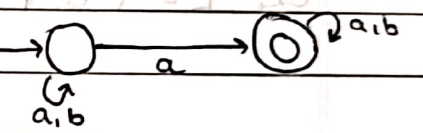
i.e. DFA = $(Q, \Sigma, q_0, F, \delta)$

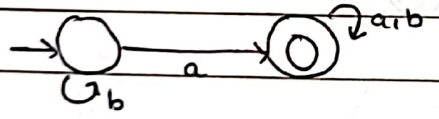
where δ is the transition function defined as

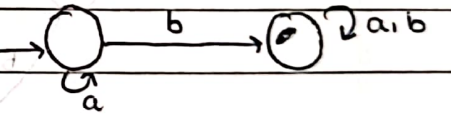
$$\delta: Q \times \Sigma \longrightarrow Q$$

Ex. $\Sigma = \{a, b\}$. Which of the following is DFA.

(a) 

(b) 

✓(c) 

✓(d) 

(a) has missing $\xrightarrow{b}$ transition for second state.

(b) has 2 transitions for $\xrightarrow{a}$ for first state.

(d) has no final state. $\therefore L(M) = \emptyset$

- Guidelines for building DFA for a given language:

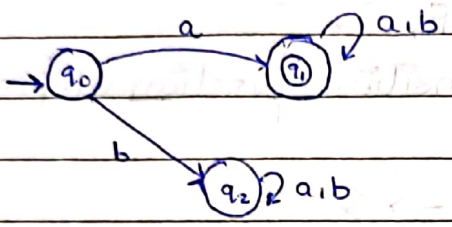
(1) Understand language.

(2) One state at a time & one symbol for it at a time.

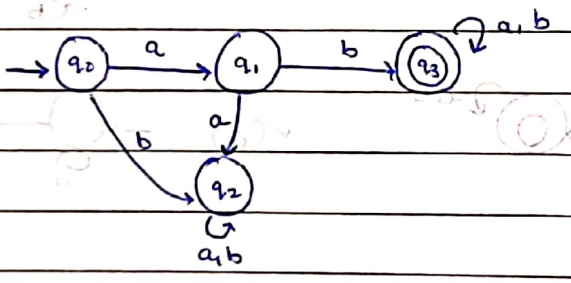
(3) Keep track of what you've already seen & what remains to be seen.

Ex. $L = \{w \mid w \text{ has prefix 'a'}\}$ $\Sigma = \{a, b\}$

w should start with a. Rest can be whatever.
w can't be ϵ.

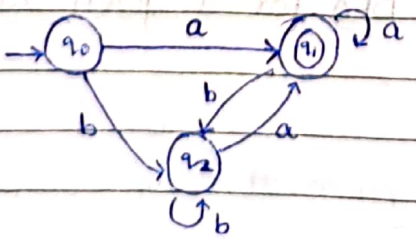


Ex. $L = \{w \mid w \text{ starts with 'ab'}\}$ $\Sigma = \{a, b\}$

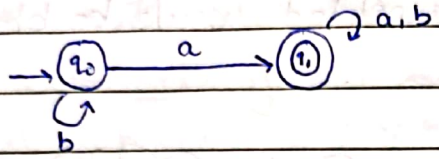


Ex. $L = \{w \mid w \text{ ends with 'a'}\}$ $\Sigma = \{a, b\}$

We want that when string ends then it should have the last seen symbol as 'a'.



Ex. $L = \{ wax \mid w, x \in \{a,b\}^* \}$ $\Sigma = \{a,b\}$.

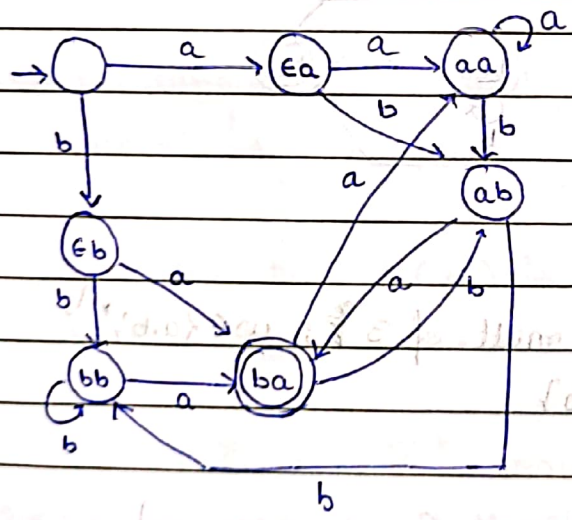


Ex. $L = \{ w \mid w$ ends with 'ba' $\}$ $\Sigma = \{a,b\}$

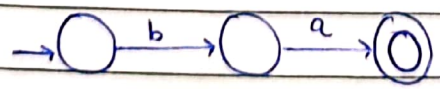
We have to keep track of last two seen symbols.

Idea 1: Each state represents last two symbols.

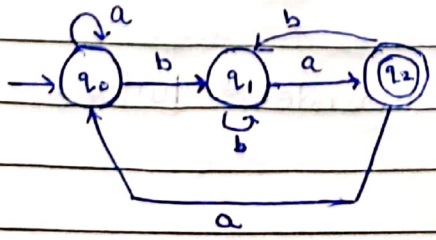
$Q = \{ \epsilon a, \epsilon b, aa, ab, ba, bb \}$



Idea 2:



⇐ Minimal accepted string 'ba'



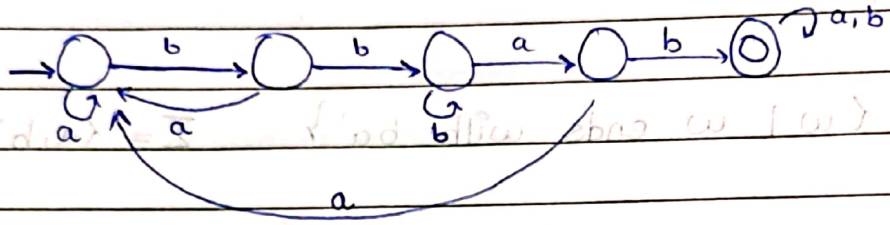
q_0: waiting for ba.
Seen ϵ

q_1: waiting for a.
seen b.

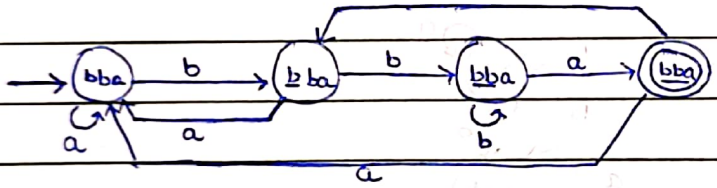
q_2: seen ba.

Ex. $L = \{ w \mid w$ has substring 'bbab' $\}$

The minimal string accepted would be bbab.



Ex. $L = \{ w \mid w$ ends with "bba" $\}$ $\Sigma = \{a,b\}$

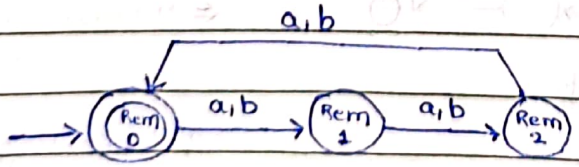


Ex. $L = \{ w \mid |w|$ is mult. of 3 ; $w \in \{a,b\}^* \}$.

$\Sigma = \{a, b\}$

Min. accepted string : ϵ since $|\epsilon| \% 3 = 0$

So, the initial state without any input should also be accepted.



States represent remainder $\Rightarrow |w| \% 3$.

* Regular Languages

W.K.t. over alphabet Σ,

- Σ^* is set of all strings over Σ.
- All subsets $L \subseteq \Sigma^*$ are languages over Σ^*.
- And power set of Σ^*

$$P(\Sigma^*) = \{ L \mid L \subseteq \Sigma^* \}$$

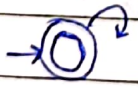
is set of all possible languages over Σ.

Out of all languages over Σ, some languages can be recognized by some DFA (or DFA can be constructed to recognize strings of the language).

Any such language L, for which there exist a DFA which recognizes L, is known as **Regular Language**.

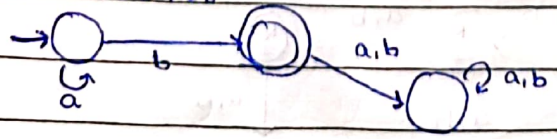
∴ For a DFA M, L(M) is always regular since M can recognize L(M).

Ex. For some Σ, is Σ^* regular language ?
(Hint: Create DFA. If possible, then regular)



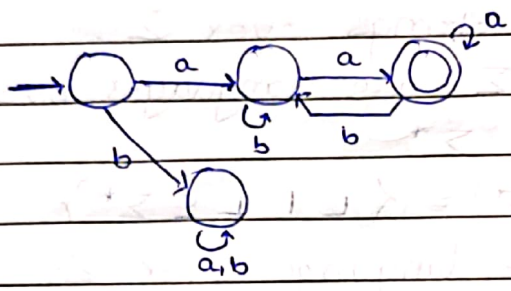
Ex. Is $L = \{ a^n b \mid n \geq 0 \}$ Regular ?

$L = \{ b, ab, aab, \ldots \}$



Ex. $L = \{ w \mid |w| \geq 2 \text{ and } w \text{ starts \& ends with a} \}$

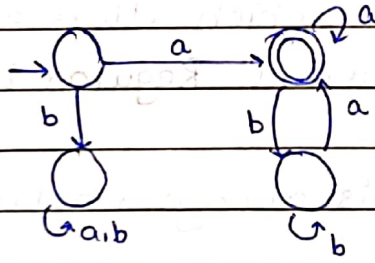
$\Sigma = \{a, b\}$.



Ex. $L = \{ w \mid w \text{ starts \& ends with a} \}$ $\Sigma = \{a, b\}$

$L = \{a, aa, a\alpha a\}$.

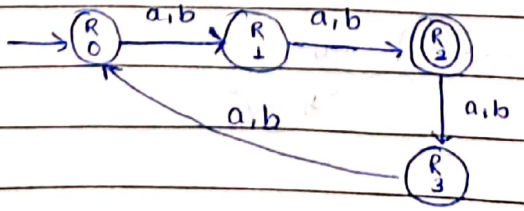
$\alpha \in \{a, b\}^*$.



Ex. $L = \{ w \mid |w| \% 4 = 2 \text{ ; } w \in \{a, b\}^* \}$

$L = \{ (a+b)^2, (a+b)^6, (a+b)^{10}, \ldots \}$

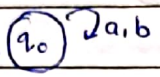
States represent mod value or remainder



Ex. $L = \{ w \mid w \in \{a,b\}^*;\ |w| \% 5 = 5 \}$

No string w, such that $|w| \% 5 = 5$.

$\therefore L = \emptyset$

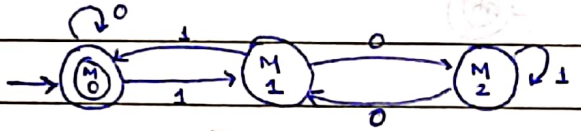
$\therefore$ 

Ex. $L = \{ w \mid w \in \{0,1\}^*;\ \text{Decimal}(w) \% 3 = 0 \} \cup \{\epsilon\}$

↑ since Decimal value of ϵ can't be found

$L = \{ \epsilon, 0, 11, 110, 1001, 1100, \ldots \}$

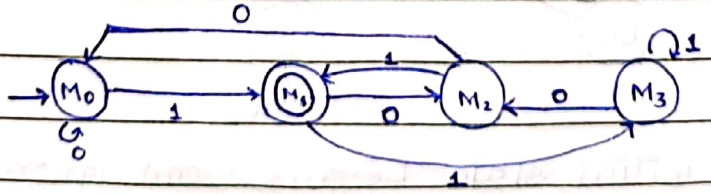
States represent mod value. String is parsed left to right.



Ex. $L = \{ w \mid w \in \{0,1\}^*,\ \text{Dec.}(w) \% 4 = 1 \}$

$L = \{ 1, 101, 1001, 1101, \ldots \}$

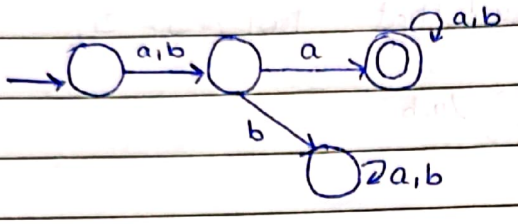
States represent mod values & strings are parsed L to R.



(This is not min. DFA. Min DFA would have 3 states)

(For Dec(w)%K type of lang., there always exist a DFA with K states which accepts the lang, but it may/maynot be minimal)

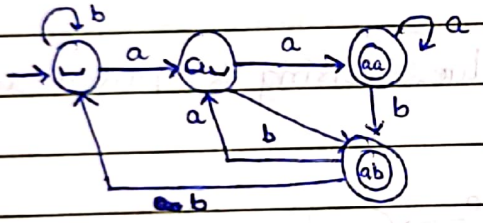
Ex. $L = \{ w \mid$ second symbol from left is 'a'; $w \in \{a,b\}^* \}$



Ex. $L = \{ w \mid 2^{nd}$ symbol from right is 'a'; $w \in \{a,b\}^* \}$

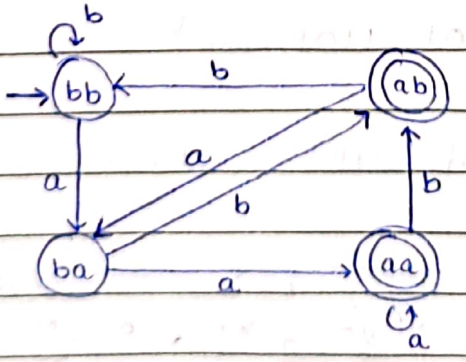
String parsed left to right.

Min. string = aa or ab



[K^{th} symbol from right is important]

basically in the above we can keep track of last two symbols.

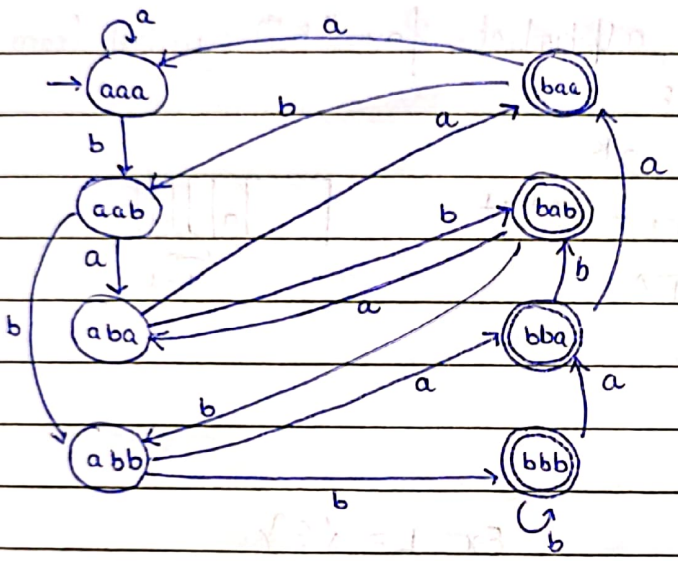


bb is initial state because can ignore any number of b's as they don't contribute to final string.

Ex. $L = \{w \mid w \in \{a,b\}^*$; 3rd symbol from right is 'b' $\}$

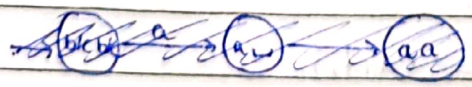
Final states: bxy , $x, y \in \{a,b\}$.

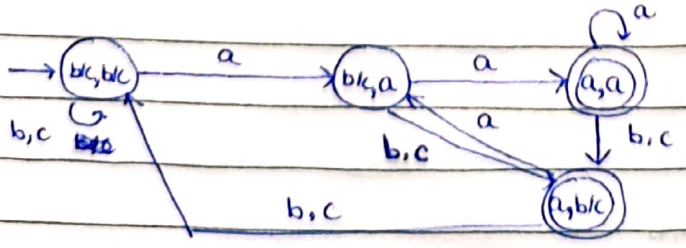
Init State: aaa. Doesn't contribute any b's.



aaa is init state since it is equivalent to ϵ since all a's can just be ignored.

Ex. $L = \{w \mid w \in \{a,b,c\}^*$; 2nd symbol from right is 'a' $\}$



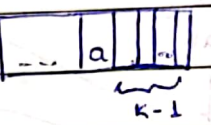


- For unary alphabet, for "k^{th} symbol from right should be a":

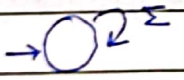
  # states = $k+1$
  # final states = 1
  } minimal DFA.

- For non-unary alphabet, for "k^{th} symbol from right should be a[99]":

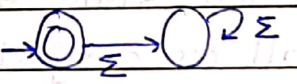
  # states = 2^k
  # final states = 2^{k-1} in minimal DFA.



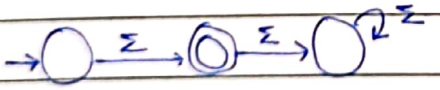
Ex. $L = \phi$



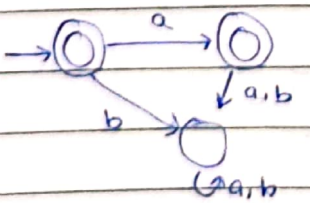
Ex. $L = \{\epsilon\}$



Ex. $L = \Sigma$

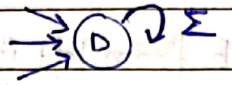


Ex. $L = \{\epsilon, a\}$ $\Sigma = \{a, b\}$



Ex. For a finite language can we guarentee a dead state?

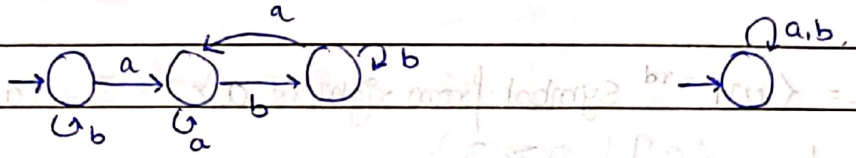
Dead State: A non-final state from which all transitions are to itself only.



$(\forall x \in \Sigma,\ (q, x) \to q) \to q$ is D.S.

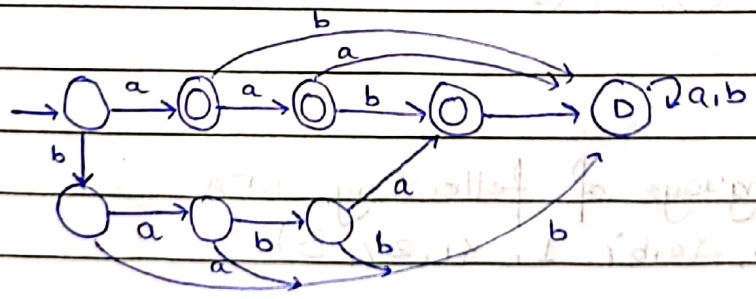
No. Only in Minimal DFA of a finite language can we guarentee a dead state.

Ex. $L = \phi$

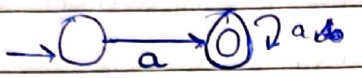


- a finite language $\longleftrightarrow$ $\exists$ some longest string.

Ex. $L = \{a, aa, aab, baba\}$

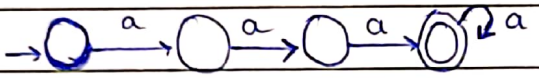


Ex. $L = \{w \mid w \text{ contains a}\}$ $\Sigma = \{a\}$



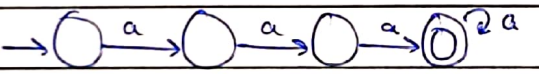
Ex. $L = \{w \mid 3^{rd} \text{ symb. from left is a}\}$ $\Sigma = \{a\}$

$L = \{a^n \mid n \geq 3\}$



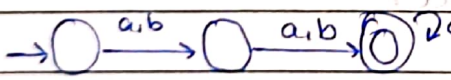
Ex. $L = \{w \mid 3^{rd} \text{ symbol from right is a}\}$ $\Sigma = \{a\}$

$L = \{a^n \mid n \geq 3\}$.



Ex. $L = \Sigma^+ - \Sigma$ $\Sigma = \{a, b\}$

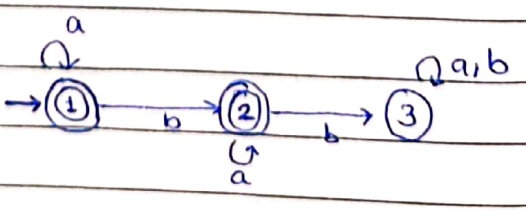
L = strings of len at least 2.



Ex. Find language of following DFA:

$(\{1,2,3\}, \{a,b\}, 1, \{1,2\}, \delta)$

δ	a	b
$\rightarrow 1$	1	2
* 2	2	3
3	3	3



$L(M) = \{\epsilon, a^n, a^n b, a^n b a^m\}$

$\Rightarrow \{a^n b a^m \mid n, m \geq 0\} \cup \{\epsilon\} \cup \{a^n \mid n \geq 0\}$

$\therefore$ All strings with atmost one b.

* Extended Transition Function

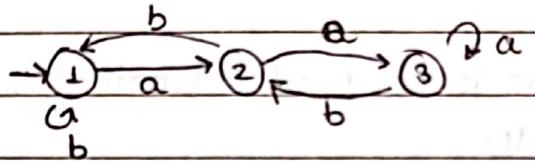
W.K.t for DFA transition function is defined as :

$$\delta : Q \times \Sigma \to Q.$$

We can defined an Extended Transition func. with state & string as input :

$$\delta^* : Q \times \Sigma^* \to Q.$$

Ex.



(1) $\delta(2, abab)$

Not defined. $\delta : Q \times \Sigma \to Q$.

(2) $\delta^*(2, abba)$

$\Rightarrow \delta^*(3, bba)$

$\Rightarrow \delta^*(2, ba)$

$\Rightarrow \delta^*(1, a)$

$\Rightarrow 2.$

(3) $\delta^*(3, \epsilon)$

$\Rightarrow 3.$

• Formal Defn: Extended Transition Func.

$$\delta^* : Q \times \Sigma^* \to Q$$

is defined as.

• $\delta^*(q, \epsilon) = q$

• $\delta^*(q, aw) = \delta^*(\delta(q, a), w)$

$a \in \Sigma$, $w \in \Sigma^*$

- So, for a DFA $M(Q, \Sigma, q_0, F, \delta)$, and a given string w, the initial config can be defined as:

$$\delta^*(q_0, w)$$

And the language of M is:

$$L(M) = \{w \mid \delta^*(q_0, w) \in F\}$$

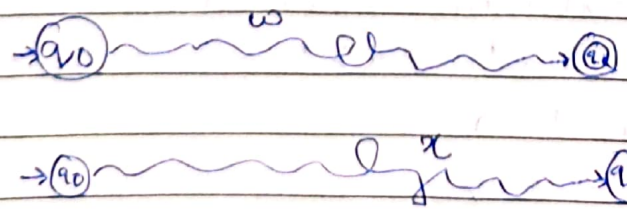
* Notes:

(1) Since every string is of finite length n, a DFA always halts.

(2) For a n-length string, a DFA goes through (n+1)-len sequence of states, and for every string this sequence is unique.

* DFA for complement of Language.

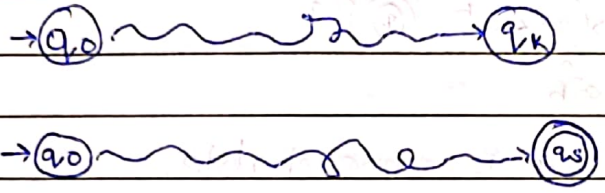
Lets take two strings: w & x, such that $w \in L(M)$ for some DFA M & $x \notin L(M)$.
Then w would have a unique sequence of states which ends on one of final states & x would end on a non-final state.



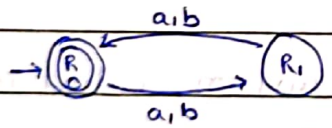
So, to find the DFA for $\overline{L(M)}$, we just change all non-final states to final & vice-versa.

So, given DFA M, with $\in L(M)$, to find DFA for $\overline{L(M)}$, exchange final & non-final states.

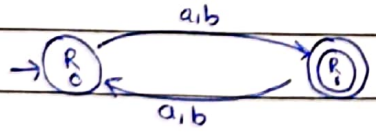
Now, for $w \in L(M)$ & $x \notin L(M)$, in DFA of $\overline{L(M)}$



Ex. $L = \{w \mid |w| \text{ is even}\}$ $\Sigma = \{a, b\}$

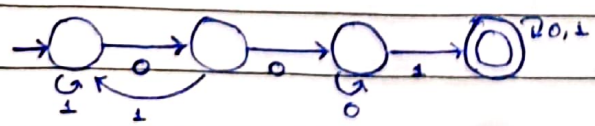


⇐ Accepts all $w \in L(M)$

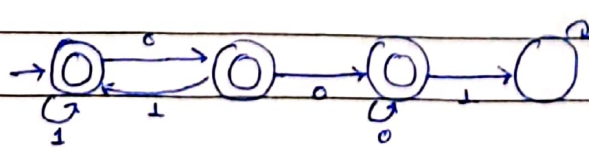


⇐ Accepts all $w \in \overline{L(M)}$

Ex. $L = \{w \mid w \text{ doesn't contain } 001 \text{ substring}\}$ $\Sigma = \{0, 1\}$



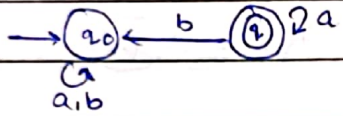
⇐ Recognizes $\overline{L}$.



- So, for a DFA accepting lang. L, $(Q, \Sigma, q_0, F, \delta)$ the DFA accepting $\bar{L}$ would be, $(Q, \Sigma, q_0, Q-F, \delta)$

* Unreachable State: A state q in a DFA is unreachable if no path exists from the initial state q_0 to q.

Ex.



q is unreachable.

Unreachable states don't affect the language. An unreachable state can be added or removed from DFA without affecting the accepted language.

Ex. Given a DFA M, string w, $|w|=n$, what is the running time complexity of M on w?

✓(1) $O(n)$

✓(2) $\Theta(n)$

✓(3) $\Omega(n)$

For a n-len string DFA takes $(n+1)$ steps irrespective of DFA.

Ex. $L_1 = \{w \mid |w|_1 \% 2 = 0\}$ $L_2 = \{w \mid |w|_1 \% 3 = 0\}$

DFA for $L_1 \cap L_2$

$L_1 \cap L_2 = \{w \mid |w|_1 \% 6 = 0\}$

states = 6 } minimal DFA.

final states = 1

* There can be certain languages for which operations are hard to create DFA for.

Ex. $\Sigma = \{0, 1\}$

$L_1 = \{w \mid w \text{ contains } 01\}$ $L_2 = \{w \mid |w|_1 \% 2 = 0\}$

Then its difficult to construct DFA for:

① $L_1 \cup L_2$ ③ $L_1 - L_2$

② $L_1 \cap L_2$ etc.

although DFA of L_1 & L_2 individually are easy.

- If for two languages L_1 & L_2, we can create DFAs M_1 & M_2, then we can run them in parallel by giving the same string as input.

① $L_1 \cap L_2 = \{w \mid w \in L_1 \wedge w \in L_2\}$

This means w is accepted by M_1 & M_2 both.

② $L_1 \cup L_2 = \{w \mid w \in L_1 \vee w \in L_2\}$

w is accepted by M_1 or M_2 or both.

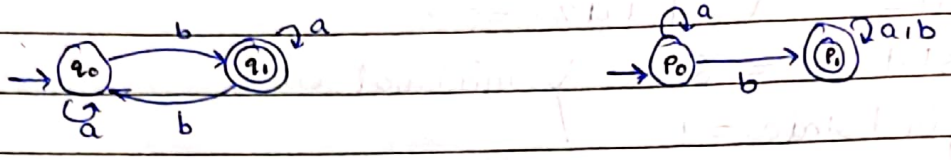
③ $L_1 - L_2 = \{w \mid w \in L_1 \wedge w \notin L_2\}$

w is accepted by M_1, but not M_2.

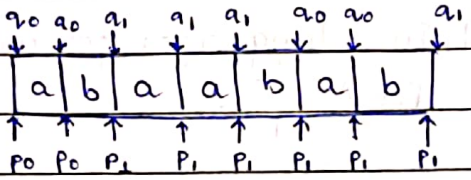
Ex. $\Sigma = \{a, b\}$

$L_1 = \{ w \mid w$ has odd no. of b's $\}$

$L_2 = \{ w \mid w$ contains $b \}$



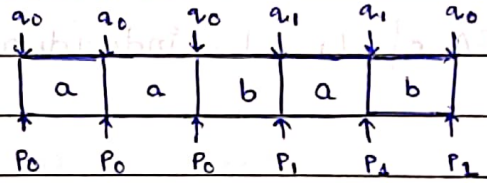
- Say $w = abaabab$



So, $w \in L_1 \cup L_2$, $w \in L_1 \cap L_2$, $w \notin L_1 - L_2$

$w \notin L_2 - L_1$

- Say $w = aabab$

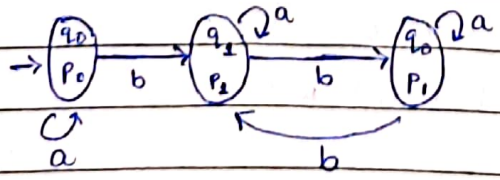


$w \in L_1 \cup L_2$, $w \notin L_1 \cap L_2$, $w \notin L_1 - L_2$

$w \in L_2 - L_1$

- For two DFAs, M_1 & M_2, $M_1 \times M_2$ means M_1 & M_2 are running in parallel.

Ex. In above example find $M_1 \times M_2$.



So just start from initial states of both machines & include all reachable states for all symbols.

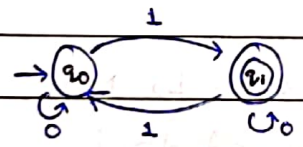
Now, for $L_1 \cap L_2$: $F = \{q_1p_1\}$ {both in final states}
for $L_1 \cup L_2$: $F = \{q_1p_1, q_0p_1\}$ {atleast one in F.S.}
for $L_1 - L_2$: $F = \{\}$
for $L_2 - L_1$: $F = \{q_0p_1\}$

{$L_1 - L_2$ means odd no. of b's, but w shouldn't contain b, which is impossible}

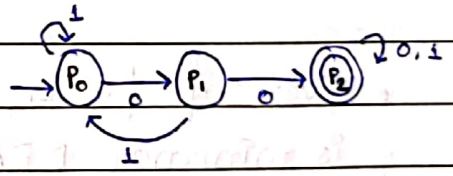
This process is called Product Construction.

Ex. $L_1 = \{w \mid w$ has odd no. of 1s$\}$ $\Sigma = \{0,1\}$
$L_2 = \{w \mid w$ contains 00$\}$

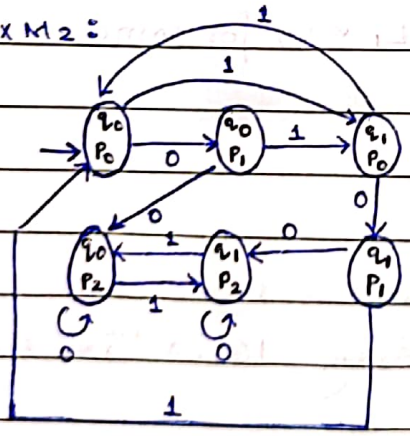
M_1:



M_2:



$M_1 \times M_2$:



$L_1 \cap L_2 \Rightarrow F = \{q_1p_2\}$
$L_1 \cup L_2 \Rightarrow F = \{q_1p_0, q_1p_1, q_1p_2, q_0p_2\}$
$L_1 - L_2 \Rightarrow F = \{q_1p_0, q_1p_1\}$
$L_2 - L_1 \Rightarrow F = \{q_0p_2\}$
$L_1 \Delta L_2 \Rightarrow F = \{q_1p_0, q_1p_1, q_0p_2\}$

- Formal Defⁿ: Given language L_1 & L_2 with DFAs M_1 & M_2 respectively, such that

$$M_1 = (Q_1, \Sigma, q_0, F_1, S_1)$$

$$M_2 = (Q_2, \Sigma, p_0, F_2, S_2)$$

there Product $M_1 \times M_2$ would be:

$$M_1 \times M_2 = (Q_1 \times Q_2, \Sigma, q_0 p_0, F, S)$$

where,

$$S : (Q_1 \times Q_2) \times \Sigma \rightarrow (Q_1 \times Q_2)$$

& F depends on the purpose of $M_1 \times M_2$.

	F
$L_1 \cup L_2$	$(x,y),\ x \in F_1 \vee y \in F_2$
$L_1 \cap L_2$	$(x,y),\ x \in F_1 \wedge y \in F_2$
$L_1 - L_2$	$(x,y),\ x \in F_1 \wedge y \notin F_2$

- Even if for L_1 M_1 is minimal DFA & for L_2, M_2 is minimal DFA, it is not guarenteed that $L_1 \times L_2$ is minimal DFA for $L_1 * L_2$ for some operation *.

Note: Always go from start states & trace states for each input, rather than creating $|Q_1| \times |Q_2|$ number of states.

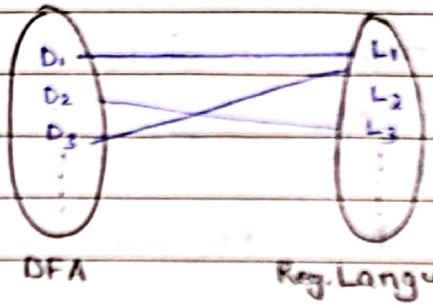
- We can similarly have n DFAs running in parallel.

classmate
Date
Page

* DFA Minimization

- For a given regular language we can create infinite number of DFAs. (say by adding unreachable states).

- For a given dfa M it recognizes a unique language L(M).



So this mapping is a func. and since for every regular regular language ∃ some DFA, this is onto function.

- For a given regular language, the DFA with minimum number of states is known as **Minimal DFA** of that language. (This would also be Minimum DFA. Proved by Myhill-Nerode).

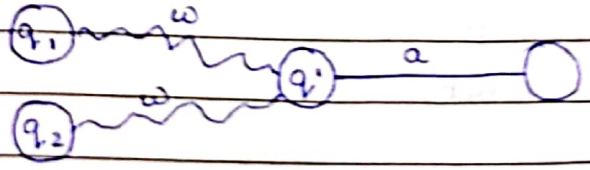
- **Steps for DFA Minimization:**

(1) Remove Unreachable States

(2) Find Equivalent States & merge them.

Ex. If $\delta^*(q_1, w) = \delta^*(q_2, w) = q'$ then
$\delta^*(q_1, wa) = \delta^*(q_2, wa)$? for DFA?

Yes. Since DFA $\delta^*(q_1, w) = \delta^*(q_2, w) = q'$
then $\delta(q', a)$ would be a unique state.



- **Equivalent States** : Two states A, B of a DFA are equivalent iff :

$\forall w \in \Sigma^*$

$[\ \delta^*(A, w) \in F \wedge \delta^*(B, w) \in F\ ]$

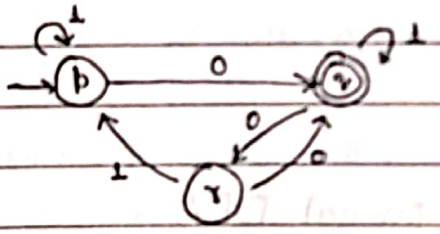
or

$[\ \delta^*(A, w) \notin F \wedge \delta^*(B, w) \notin F\ ]$

i.e. for all strings w, either both go to final or both don't go to non-final states.

(Equivalent ≡ Indistinguishable ≡ Mergable)

Ex.



(p, q) : $\left(\delta^*(p, \epsilon) \in NF\right) \neq \left(\delta^*(q, \epsilon) \in Final\right)$

$\therefore$ p & q are non-eq.

(q, r) : $\delta^*(q, \epsilon) \in$ Final $\wedge$

$\delta^*(r, \epsilon) \in$ Non-Final.

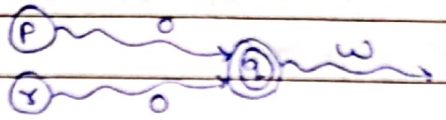
$\therefore$ q, r are non-eq.

(p, r) :

δ^*	p	r
ϵ	NF	NF
0	F	F
1	NF	NF

We can notice, $\delta^*(p,0) = \delta^*(r,0) = q$, i.e. they go to same state, then for all strings $0w$,

$$\delta^*(p,0w) = \delta^*(r,0w).$$

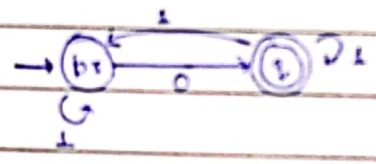


Similarly, $\delta^*(p,1) = \delta^*(r,1) = p$, so for all strings $1w$

$$\delta^*(p,1w) = \delta^*(r,1w).$$

$\therefore \forall w \in \Sigma^*, \delta^*(p,w) = \delta^*(r,w)$.

So, p & r are mergable.

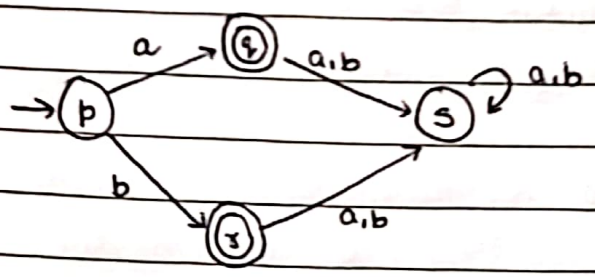


This is the minimal DFA, since these two states are distinguishable, and can't be merged.

Ex. So in a DFA, can final & non-final state be equivalent?

No. $\delta^*(f, \epsilon) \in F \wedge \delta^*(\bar{f}, \epsilon) \in$ Non-F.

Ex.

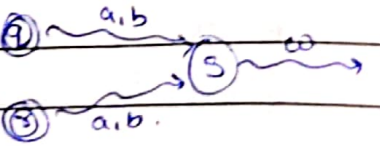


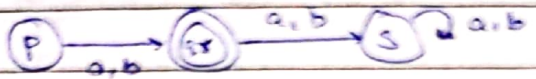
(p,s): $\delta^*(p,a) \in F \wedge \delta^*(s,a) \notin F$

$\therefore$ Non-eq.

(q,r): $\forall w \in \Sigma^* \; \delta^*(q,w) = \delta^*(r,w) = s \notin F$

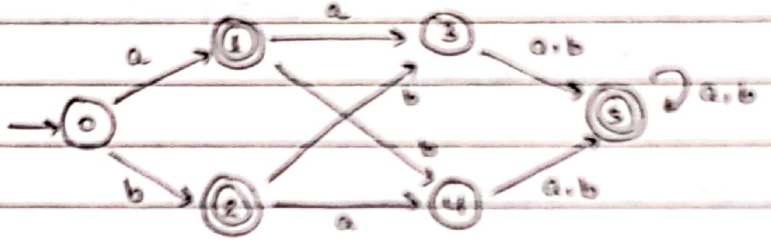
$\therefore$ Eq.





Minimal DFA, since all states distinguishable.

Ex.

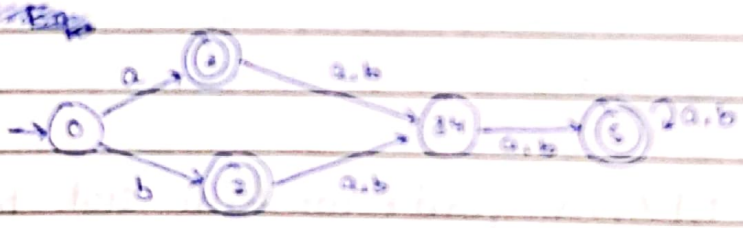


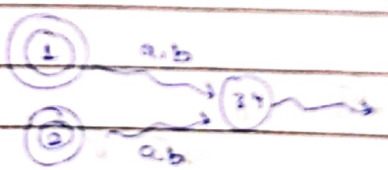
(0,3): Disting. by ab

(0,4): Disting. by ab.

(3,4): Eq. 

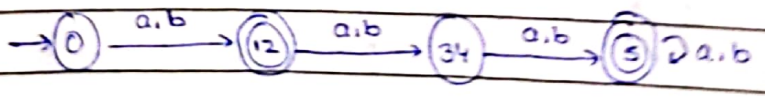
~~(1,2): Eq.~~



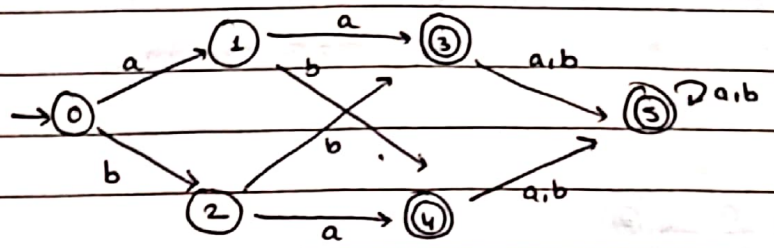
(1,2): Eq. 

(1,5): Dist. by a

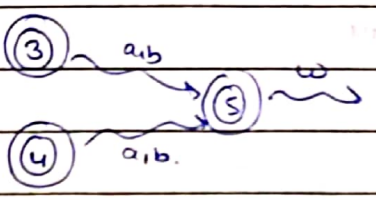
(2,5): —"—



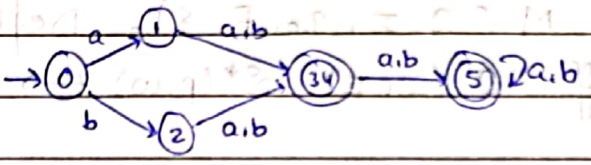
Ex.



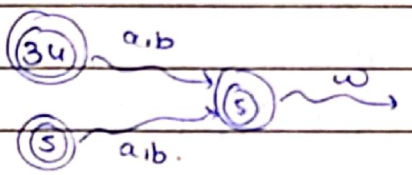
(3,4): Eq.



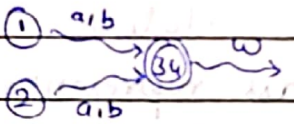
Ex.

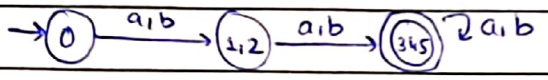


(34,5): Eq.



(1,2): Eq.



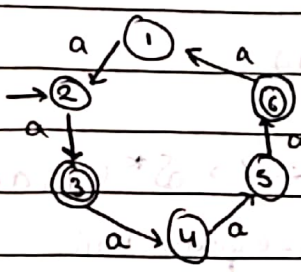


$L = \{w \mid |w| \geq 2\}$

No eq. states. ∴ Min. DFA.

- We don't actually need to check equivalence condⁿ for all strings, just strings w, where $|w| \leq n-2$ where n is the no. of states in DFA are enough

Ex.



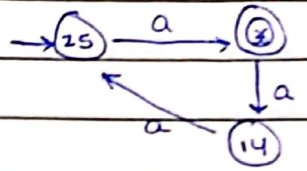
here we can't apply  idea for 3 & 6, as they never meet.

So, we check all strings of len. 6-2=4.

i.e., a, aa, aaa, aaaa.

∴ 3, 6 are eq.

Similarly based on ω. $|\omega| \leq 4$. 1 & 4 are eq.
& 2 & 5 are eq.



Ex. Suppose a DFA $M(Q, \Sigma, q_0, F, \delta)$. Define a rel. R on Q. pRq iff $\forall w \in \Sigma^*$ $\delta^*(p,w) \longleftrightarrow \delta^*(q,w)$. Comment on R.

R is reflexive, since $\forall p \in Q$ pRp.
R is symmetric, since $\forall p,q \in Q$ $pRq \rightarrow qRp$
R is transitive.
∴ R is equivalance relation.
Each equivalance class represents a unique state in Minimal DFA of $L(M)$.

Ex. Let $\delta(p,a) = p'$ & $\delta(q,a) = q'$. If p' & q' are distinguishable then p & q are distinguishable too.

Given: p' & q' are non-equiv.
i.e. $\exists w$ $\delta^*(p', w) \in F \longleftrightarrow \delta^*(pq', w) \in F$

Also, $\delta^*(p, aw) = \delta^*(p', w)$
& $\delta^*(q, aw) = \delta^*(q', w)$

∴ $\exists w$ $\delta^*(p, aw) \in F \longleftrightarrow \delta^*(q, aw) \in F$
∴ p & q are non-equivalent.

Partition Algorithm

This algo does same thing, but one symbol at a time.
Suppose we only check no symbol (ϵ). We get two partition Final & Non-Final.

Π_0 : Non-Final Final.

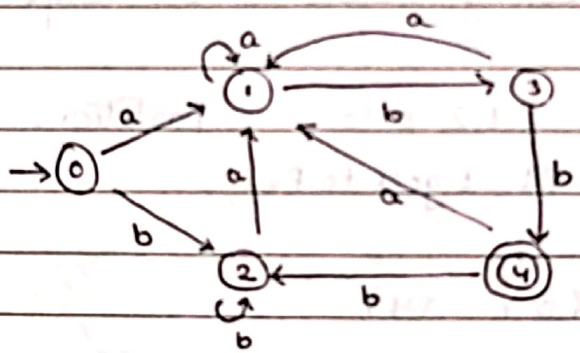
Now, if say on symbol a p & q go to non-equivalent states then p & q are also non-equivalent.

$p \rightsquigarrow p'$

$q \rightsquigarrow q'$

$\not\equiv \rightarrow p \not\equiv q.$

Ex.



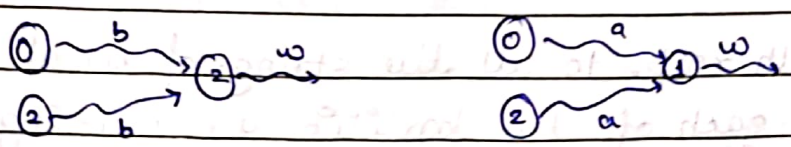
M1: Basic Idea.

3 isn't equivalent to 0, 1, 2, for $w = b$.

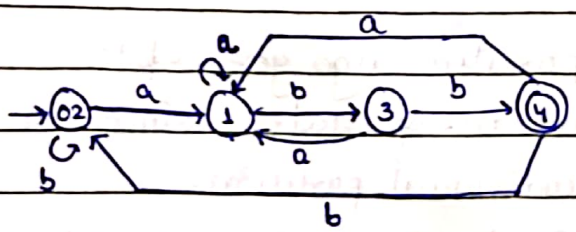
$1 \not\equiv 2$ for $w = bb$

$1 \not\equiv 0$ for $w = bb$.

(0, 2) :



$\therefore \quad 0 \equiv 2.$



$L = \{ w \mid w \text{ has suffix } abb \}$

Me: Partition Algorithm

Π_0 : {0, 1, 2, 3} {4}
↑
with resp. to 0 len string they are eq.

on a all of 0, 1, 2, 3 go to same partition.
on b, (0, 1, 2) go to part 1, 3 goes to part 2.

Π_1 : {0, 1, 2} {3} {4}
with resp to strings of len atmost 1, each partition is indistinguishable within itself.

on a all of 0, 1, 2 go to same partition.
on b 0, 2 go to P1, 1 goes to P2.

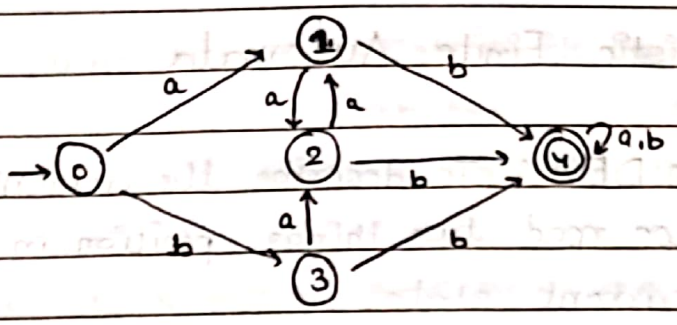
Π_2 : {0, 2} {1} {3} {4}
on a all of 0, 2 go to same partition: P2.
on b ——— " ——— : P1.

Π_3 : {0, 2} {1} {3} {4}.
∴ No more partitions can be created.

So,
Π_n : With resp. to all the strings of len at most n, each of the partition are indistinguishable within the partition.

Note: In worst case partition algo goes upto Π_{n-2} where n is the no. of states, since in Π_{n-2} all states have individual partition.
Π_0 has 2 partition, Π_1 has 3, ... Π_{n-2} has n partitions.

Ex.

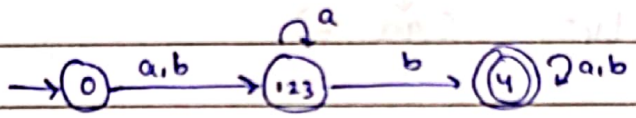


Π_0: {0,1,2,3} {4}.

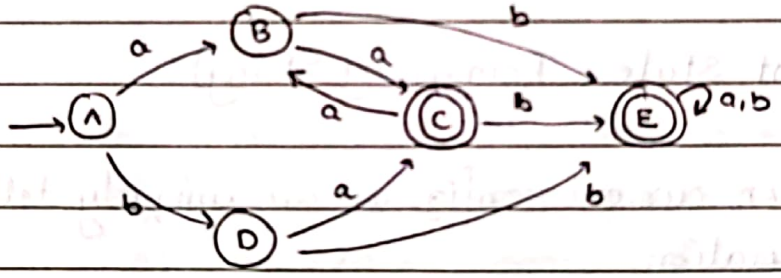
Π_1: {0} {1,2,3} {4} { because of b }

Π_2: {0} {1,2,3} {4}.

∴



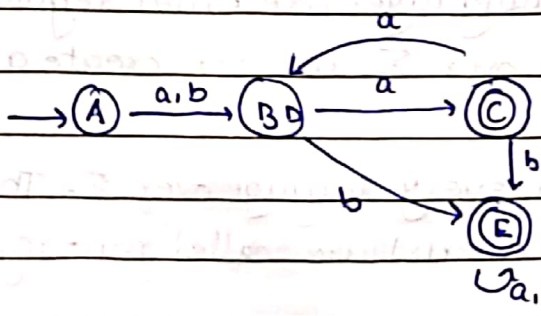
Ex.



Π_0: {A,B,D} {C,E}

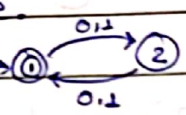
Π_1: {A} {B,D} {C} {E} { cause of a }

Π_2: {A} {B,D} {C} {E}.



* Non-Deterministic Finite Automata

- Configuration in DFA: To describe the current snapshot in DFA we need two things, position in the string & the current state.

Ex. $w = 1001$ M: 

Initial config: $(q_1, 1001)$ } Transition/ Move

t_1: $(q_2, 001)$

t_2: $(q_1, 01)$

t_3: $(q_2, 1)$

t_4: (q_1, ϵ).

So, we can describe current config. of a DFA as:

(Current State, Remaining String)

In DFA given current config. we can uniquely tell the next configuration.

- For given Σ, there exists infinite languages, $L \subseteq \Sigma^*$. A DFA can be created for a lot of the languages over Σ, and all such languages are called regular. For any finite language over Σ, we can create a DFA.

We can't create DFA for every languge over Σ. The languages for which no DFA exist are called non-regular.

Ex. $L = \{a^n b^n \mid n \geq 0\}$

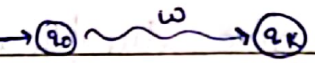
Ex. $L = \{a^p \mid p \text{ is prime}\}$

Ex. Claim: No DFA exists which recognizes L.

$L = \{a^n b^n \mid n \geq 1\}$.

$= \{ab, a^2b^2, a^3b^3, \ldots\}$

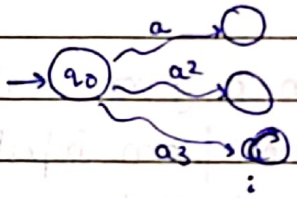
Proof: Let's say M is a DFA, such that M recognizes L.
In DFA, for every state, given a string w, it goes to some state.



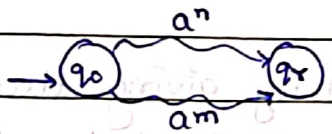
Let's suppose each of the strings in the inf. set

$A = \{a, a^2, a^3, \ldots\}$

go to unique state.

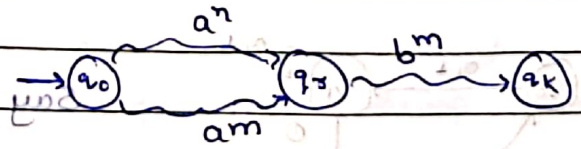


But this isn't possible, since DFA by defn has finite no. of states. So we can guarentee at least two of the strings end on same state.



$n \neq m$

Now suppose:



if q_k is final, $a^n b^m$ is accepted.
if q_k is non-final $a^m b^m$ is not accepted.
Both ways we get contradiction.
∴ No DFA exists for L.

- One interesting question arises: will giving the ability to be Non-Deterministic give more power to DFA (i.e. make it recognizes Non-Regular languages)?

The answer is No. {will be proved later}

- To a DFA if we give the following powers:

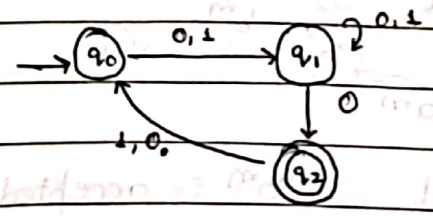
(1) Non-determinism: For a given state, given a symbol as input we can go to 1, 2, ... states (not exactly one).

(2) Null Move: From a given state without reading any symbol from input tape we can move to a different state.

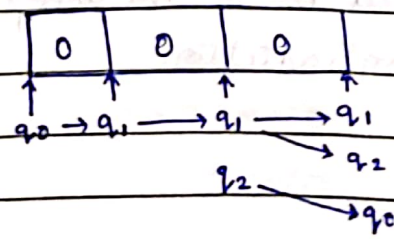
(3) No Move: On a state, we can have no transition for a given symbol.

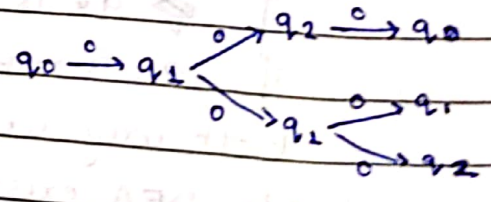
The machine we get by giving these powers to a DFA, is Non-Deterministic Finite Automata (NFA).

Ex.



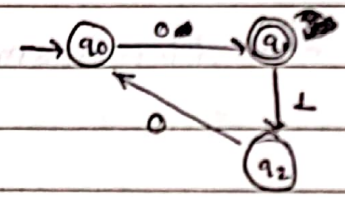
Say w = 000.





In one of the paths $q_0 \to q_1 \to q_1 \to q_2$ we end up on a final state, so we can accept the string.

- Acceptance of string by NFA: After scanning the entire string by the NFA, iff at least one of the paths ends up on a final state, the NFA accepts the string, else rejects the string.

Ex. 

$w = 0110$.

$q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_2$

↑ This is dead end.

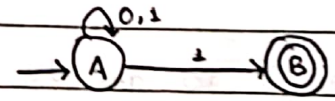
Since the string wasn't scanned in full this string is rejected.

$w = 00$

$q_0 \xrightarrow{0} q_1$

↑ Dead End.

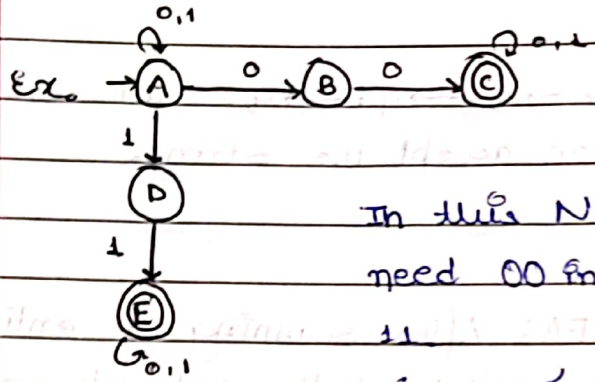
Although q_1 is final state, but string wasn't scanned in full. ∴ Rejected.

Ex. 

What is this NFA recognizing?

The machine can go to accept state for a string only if the string ends with a 1.

So the language recognized: All strings over {0,1} ending with 1.

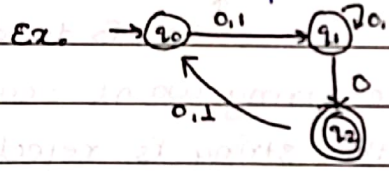


In this NFA to get to final state C, we need 00 in string & to get to E we need 11

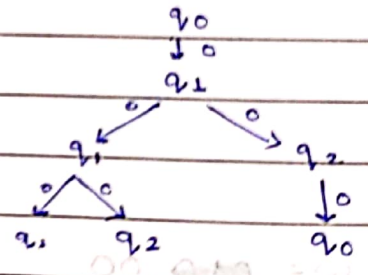
$\therefore L = \{ w \mid 00$ is substr. of w or 11 ——— // ——— $\}$.

- Three ways of thinking about non-determinism:

(1) Tree computation:



$w = 000$



(2) Guess Method:

We can directly guess what is need to reach one of the final states.

Ex. At least 00 or 10 is needed to get to q_2.

So we guess the path:

$q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_1 \xrightarrow{0} q_2$

(3) Massive Parallelism: Run parallel paths in head.

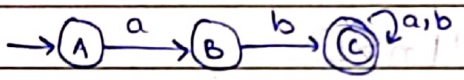
classmate
Date
Page

- Guidelines to design NFA:

(1) Focus on what needs to be accepted

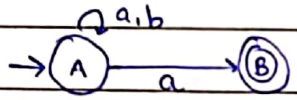
(2) NFA can reject by no transition, i.e. dead path. or path ending on non-final

Ex. $L = \{w \mid ab \text{ is prefix of } w\}$ $\Sigma = \{a, b\}$.

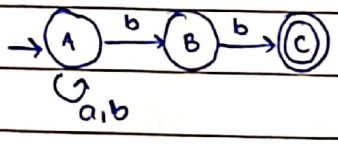


So, at A, if b comes the path is dead.

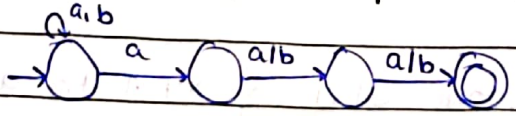
Ex. $L = \{w \mid a \text{ is suffix}\}$ $\Sigma = \{a, b\}$



Ex. $L = \{w \mid bb \text{ is suffix}\}$ $\Sigma = \{a, b\}$



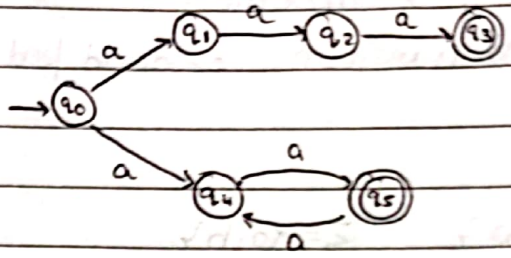
Ex. $L = \{w \mid 3^{rd} \text{ symbol from right is a}\}$ $\Sigma = \{a, b\}$



(Note: The minimal DFA had $2^3 = 8$ states) $\{2^k\}$

(In minimal NFA we have 3+1 states) $\{k+1\}$

Ex. What is the language of :

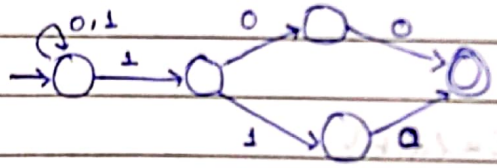


$\Sigma = \{a\}$

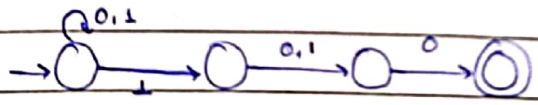
$L(M) = L(q_3) \cup L(q_5)$

$\Rightarrow \{aaa\} \cup \{a^{2n} \mid n \geq 1\}$

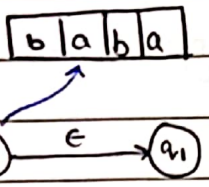
Ex. $L = \{w \mid w$ ends with 100 or w ends with 110$\}$ $\quad \Sigma = \{0,1\}$



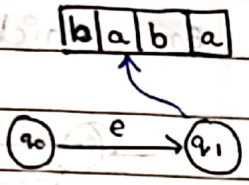
OR



- **Null Move:** Move from one state of NFA to another without consuming any symbol on input tape.
  Denoted by: $\xrightarrow{\epsilon}$
  Although same symbol, Null String & Null move are different.

Ex. 

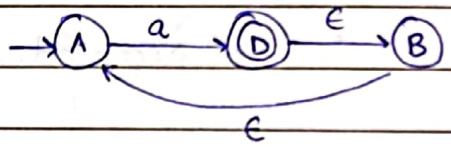
If we take $q_0 \to q_1$ transition.



Ex. Why is the symbol for Null Move same as Null String?

Because Null move behaves like Null String.

Say the NFA:

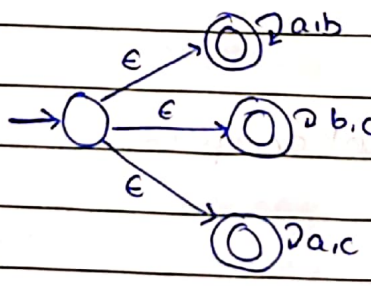


on $w=a$, we can go $A \to D$ or $A \to D \to B$ or $A \to D \to B \to A$. So $w=a$ is behaving as $w=a\epsilon$

on $w=aa$ we can go, $A \xrightarrow{a} D \xrightarrow{\epsilon} B \xrightarrow{\epsilon} A \xrightarrow{a} D$ & be accepted.

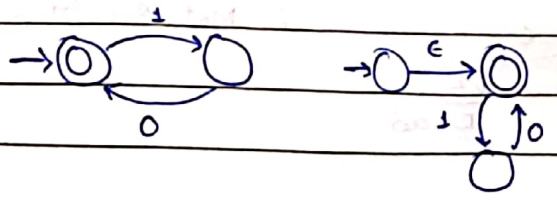
So, $w=a$ is behaving as $w=a\epsilon a$.

Ex. $L = \{w \mid$ either w has no a, or no b, or no c$\}$ $\Sigma = \{a, b, c\}$

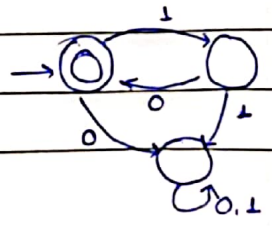


Ex. $L = \{(10)^n \mid n \geq 0\}$

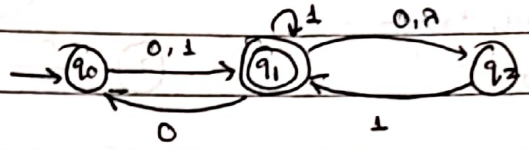
NFA:



mDFA:



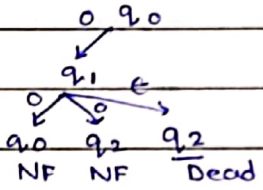
Ex. Which strings are accepted by:



(A) 00 (C) 10010 (E) 0000

(B) 01001 (D) 000

00:

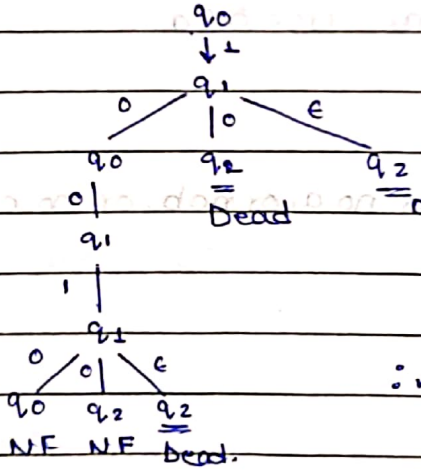


∴ Not accepted.

01001 : ① $q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1 \xrightarrow{0} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1$

② $q_0 \xrightarrow{0} q_1 \xrightarrow{\epsilon} q_2 \xrightarrow{1} q_1 \xrightarrow{0} q_0 \xrightarrow{0} q_1 \xrightarrow{1} q_1$.

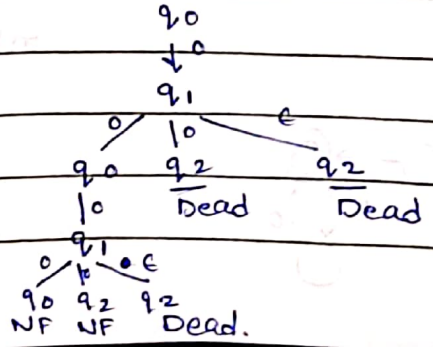
10010:



∴ Not accepted.

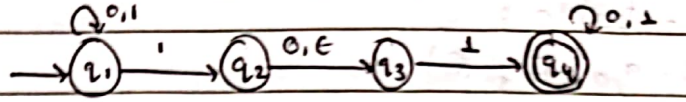
000: $q_0 \xrightarrow{0} q_1 \xrightarrow{0} q_0 \xrightarrow{0} q_1$

0000:



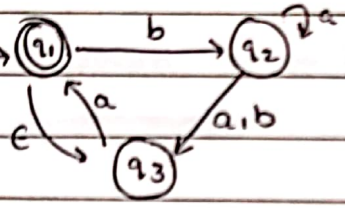
∴ Not Accept.

Ex. Find language of:



$(0+1)^* 1 (0+\epsilon) 1 (0+1)^*$

$L(M) = \{ w \mid w \text{ contains } 11 \text{ or } 101 \}$.

Ex. 

① Without reading any symbol which states can the m/c be in?
$\{q_1, q_3\}$

ϵ is accepted.
$a^n, n \geq 0$ is accepted

$Q = \{q_1, q_2, q_3\}$
$\Sigma = \{a, b\}$ { ϵ is not part of alphabet, it's just a notation}
$q_0 = q_1$
$F = \{q_1\}$
δ: $\delta(q_1, a) = \emptyset$; $\delta(q_2, a) = \{q_2, q_3\}$. $\delta(q_3, a) = \{q_1\}$
$\delta(q_1, b) = \{q_2\}$; $\delta(q_2, b) = \{q_3\}$. $\delta(q_3, b) = \emptyset$
$\delta(q_1, \epsilon) = \{q_3\}$; $\delta(q_2, \epsilon) = \{\}$. $\delta(q_3, \epsilon) = \{\} = \emptyset$
↑ null transition $= \emptyset$

$\delta^*(q_1, \epsilon) = \{q_3, q_1\}$
↑ nullstring
$\delta^*(q_2, \epsilon) = \{q_2\}$
$\delta^*(q_3, \epsilon) = \{q_3\}$.

- Formal Defⁿ: Like DFA, NFA is also defined as 5-tuple:

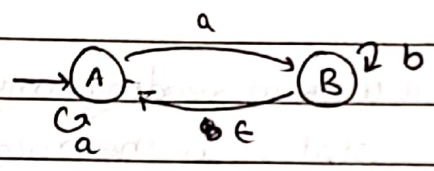
$(Q, \Sigma, q_0, F, \delta)$

where

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

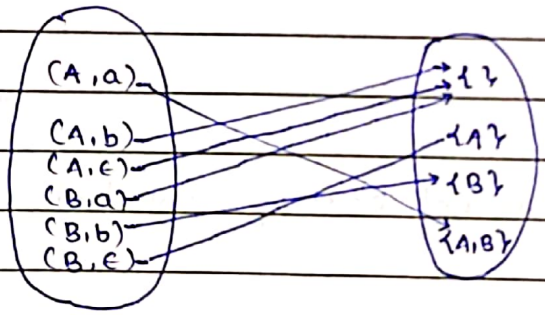
(where 2^Q is powerset of Q).

Ex.



$Q = \{A, B\}$ $\Sigma = \{a, b\}$ $q_0 = A$ $F = \emptyset$.

δ:



String Acceptance: A string w is accepted by NFA M $(Q, \Sigma, q_0, F, \delta)$, iff

$\exists q \in Q, q \in \delta^*(q_0, w) \wedge q \in F$

i.e.

$\delta^*(q_0, w) \cap F \neq \emptyset$

- Language Recognition: A language L is recognized by NFA M iff

$L = \{w \mid w \text{ is accepted by } M\}$.

* Equivalence of DFA & NFA

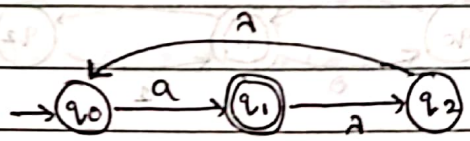
- NFA was given more power in hope to create machine which could accept more languages.
- But a NFA can ~~accept~~ recognize a language over some Σ iff some DFA can recognize the language.

i.e. $\forall L \subseteq \Sigma^*$

NFA exists for L $\longleftrightarrow$ DFA exists for L

This is because, DFA can simulate all three features of NFA.

Ex. Convert to DFA:

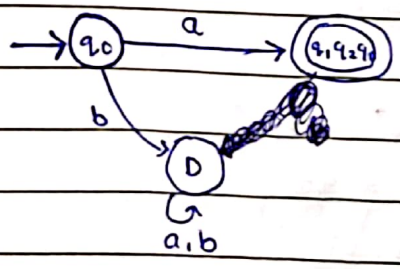


$\Sigma = \{a, b\}$

Initial State: q_0 since without reading anything we can only be on q_0.

On symbol a, DFA can go from q_0 to q_1, q_2, q_0.

On b, q_0 goes to nowhere, ∴ Dead State.



For $q_0q_1q_2$:

On a, q_0 goes to $q_0q_1q_2$

q_1 goes to $q_0q_1q_2$

q_2 goes to $q_0q_1q_2$

$\therefore \delta(q_0q_1q_2, a) = q_0q_1q_2$

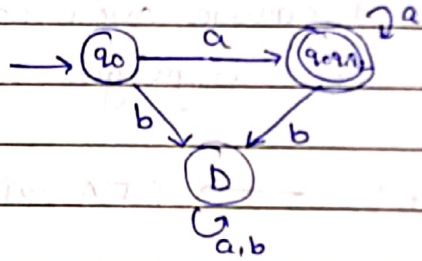
On b, q_0 goes to Nowhere.

q_1 goes to Nowhere.

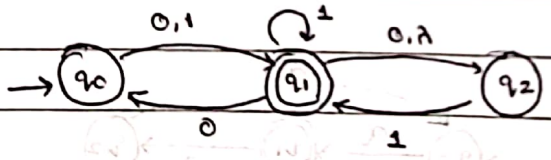
q_2 goes Nowhere.

$\therefore \delta(q_0q_1q_2, b) = \phi$.

$\therefore$



Ex.



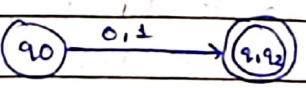
$\Sigma = \{0,1\}$

Initial state: q_0.

$\delta(q_0, 0) = \{q_1, q_2\}$

$\delta(q_0, 1) = \{q_1, q_2\}$

$\therefore$



For q_1q_2 :

$\delta(q_1q_2, 0) = \delta^*(q_1, 0) \cup \delta^*(q_2, 0)$

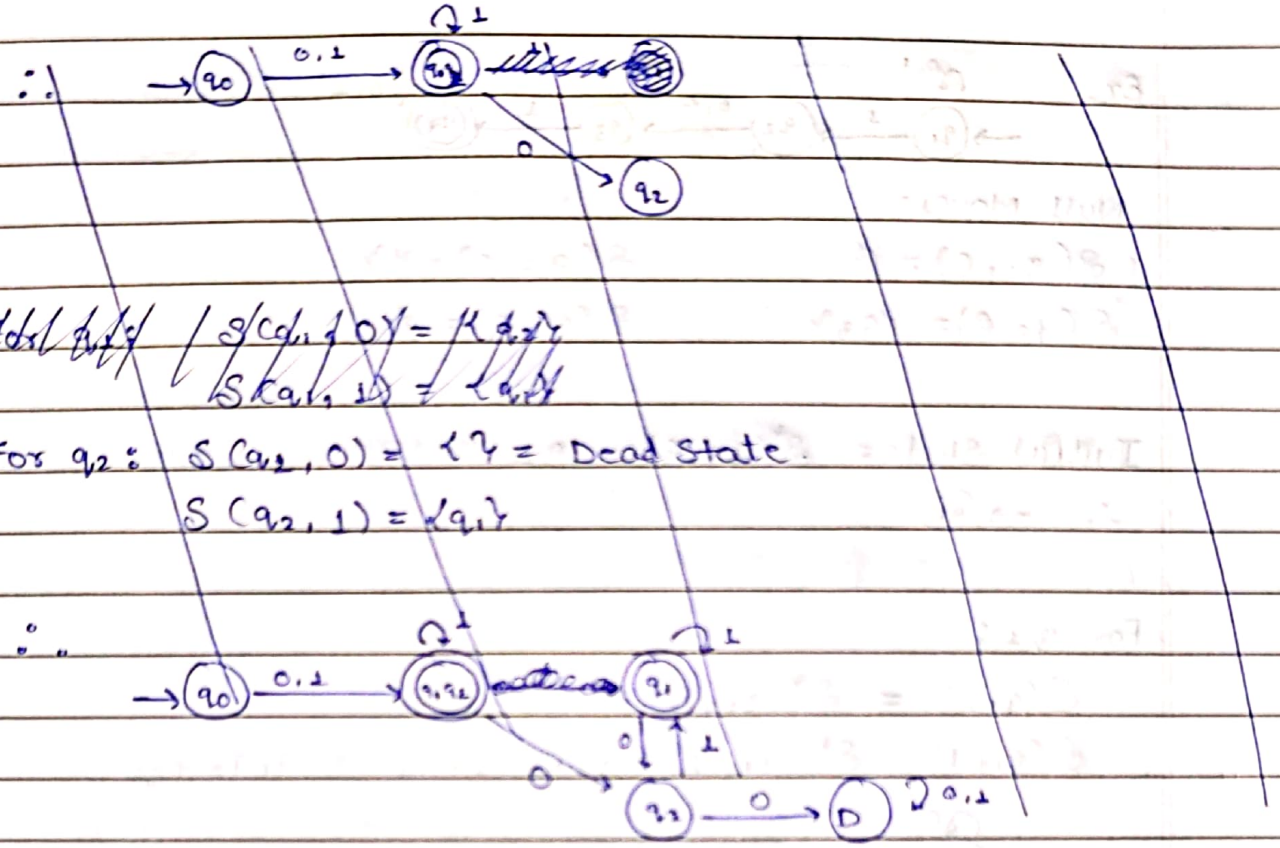
$= \{q_2\} \cup \{\} = q_2$

$\delta(q_1q_2, 1) = \delta^*(q_1, 1) \cup \delta^*(q_2, 1)$

$\Rightarrow \{q_1, q_2\} \cup \{q_1\} = q_1q_2$

To prevent this: $q \xrightarrow{\epsilon^*} q_1 \xrightarrow{a} q_1 \xrightarrow{\epsilon^*} q_2$

Check null moves.
Check symbol.
Check null move again.



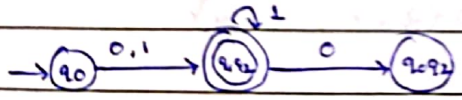
~~For q_1: $\delta(q_1, 0) = \{q_2\}$~~
~~$\delta(q_1, 1) = \{q_1\}$~~

For q_2: $\delta(q_2, 0) = \{\} =$ Dead State.
$\delta(q_2, 1) = \{q_1\}$

- For $q_1 q_2$

$\delta(q_1 q_2, 0) = \delta^*(q_1, 0) \cup \delta^*(q_2, 0)$
$\Rightarrow \{q_0, q_2\} \cup \{\} = q_0 q_2$

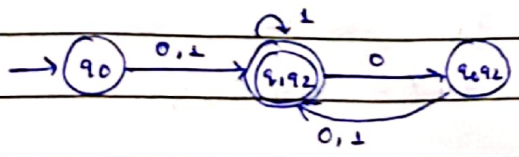
$\delta(q_1 q_2, 1) = \delta^*(q_1, 1) \cup \delta^*(q_2, 1)$
$\Rightarrow \{q_1, q_2\} \cup \{q_1\} = q_1 q_2$

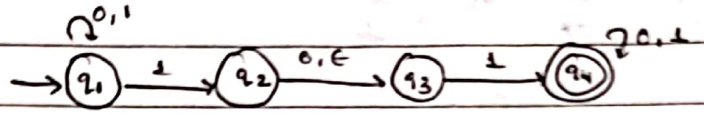


- For $q_0 q_2$

$\delta(q_0 q_2, 0) = \delta^*(q_0, 0) \cup \delta^*(q_2, 0)$
$\Rightarrow \{q_1, q_2\} \cup \{\} = q_1 q_2$

$\delta(q_0 q_2, 1) = \delta^*(q_0, 1) \cup \delta^*(q_2, 1)$
$\Rightarrow \{q_1, q_2\} \cup \{q_1\} = q_1 q_2$



Ex. 

Null Moves:

$\delta(q_1, \epsilon) = \phi$ $\quad$ $\delta(q_3, \epsilon) = \{\}$

$\delta(q_2, \epsilon) = \{q_3\}$ $\quad$ $\delta(q_4, \epsilon) = \phi$

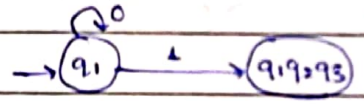
Initial State = $\delta^*(q_1, \epsilon) = q_1$

$\therefore \rightarrow (q_1)$

For q_1:

$\delta(q_1, 0) = \delta^*(q_1, 0) = q_1$

$\delta(q_1, 1) = \delta^*(q_1, 1) = \{q_1, q_2, q_3\} = q_1 q_2 q_3$.

$\therefore$ 

For $q_1 q_2 q_3$:

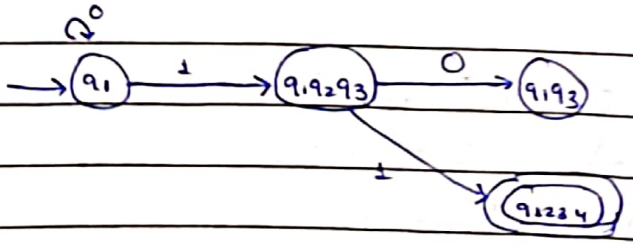
$\delta(q_1 q_2 q_3, 0) = \delta^*(q_1, 0) \cup \delta^*(q_2, 0) \cup \delta^*(q_3, 0)$

$\Rightarrow \{q_1\} \cup \{q_3\} \cup \{\}$

$\Rightarrow q_1 q_3$

$\delta(q_1 q_2 q_3, 1) = \{q_1, q_2, q_3\} \cup \{q_4\} \cup \{q_4\}$

$\Rightarrow \{q_1 q_2 q_3 q_4$.

$\therefore$ 

For $q_1 q_3$:

$\delta(q_3 q_1, 0) = \{q_1\} \cup \{\phi\} = q_1$

$\delta(q_1 q_3, 1) = \{q_1, q_2, q_3\} \cup \{q_4\} = q_1 q_2 q_3 q_4$

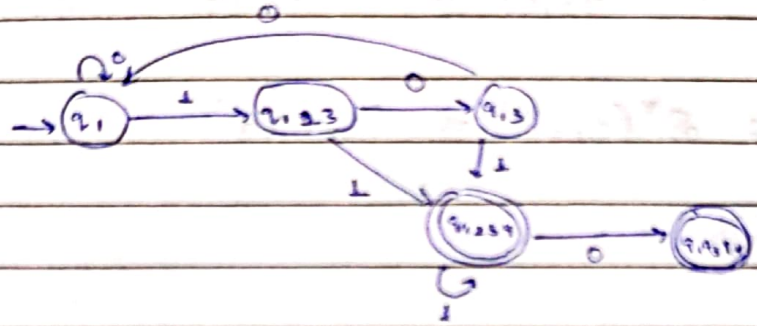
For $q_1q_2q_3q_4$:

$\delta(q_1q_2q_3q_4, 0) = \{q_1\} \cup \{q_3\} \cup \{\} \cup \{q_4\}$

$\Rightarrow q_1q_3q_4$.

$\delta(q_1q_2q_3q_4, 1) = \{q_1q_2q_3q_4\}$.

$\therefore$



For $q_1q_3q_4$

$\delta(q_1q_3q_4, 0) = q_1q_4$

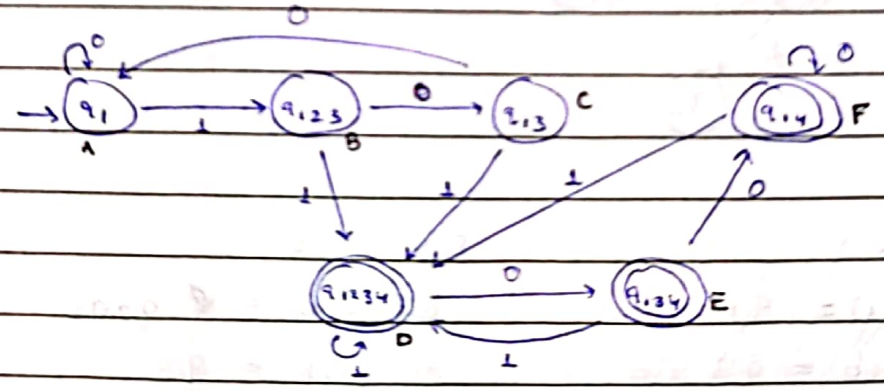
$\delta(q_1q_3q_4, 1) = q_1q_2q_3q_4$

For q_1q_4

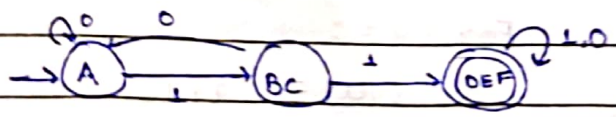
$\delta(q_1q_4, 0) = q_1q_4$

$\delta(q_1q_4, 1) = q_1q_2q_3q_4$

$\therefore$

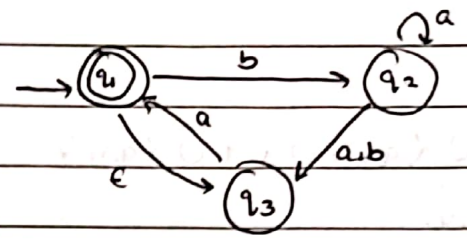


mDFA:



(B & C shouldn't be merged. Wrong!!)

Ex.



Null Transitions:

$\delta(q_1, \epsilon) = \{q_3\}$ $\delta(q_2, \epsilon) = \emptyset$ $\delta(q_3, \epsilon) = \emptyset$.

Initial state: $\delta^*(q_1, \epsilon) = \{q_1, q_3\}$

∴ → $(q_1 q_3)$

For $q_1 q_3$:

$\delta(q_1 q_3, a) = \delta^*(q_1, a) \cup \delta^*(q_3, a)$

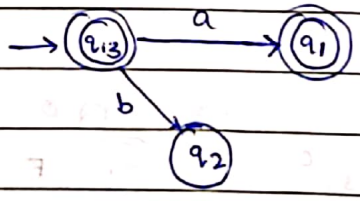
$= \{q_3, q_1\} \cup \{q_1, \emptyset, q_3\}$

$\Rightarrow q_1 q_3$

$\delta(q_1 q_3, b) = \delta^*(q_1, b) \cup \delta^*(q_3, b)$

$\Rightarrow \{q_2\} \cup \{\} = q_2$

∴



For q_1:

$\delta(q_1, a) = q_1$

$\delta(q_1, b) = q_2$

For q_2:

$\delta(q_2, a) = q_2 q_3$

$\delta(q_2, b) = q_3$

For $q_2 q_3$:

$\delta(q_2 q_3, a) = q_1 q_2 q_3$

$\delta(q_2 q_3, b) = q_3$

For q_3:

$\delta(q_3, a) = q_1 q_3$

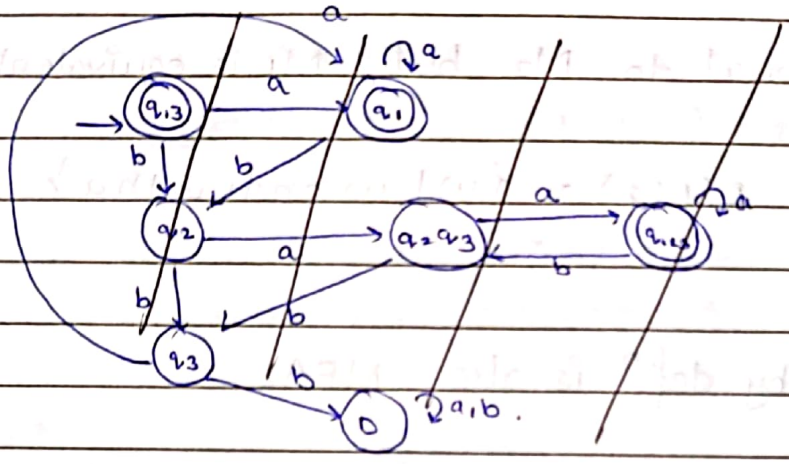
$\delta(q_3, b) = \emptyset = D$.

For $q_1q_2q_3$:

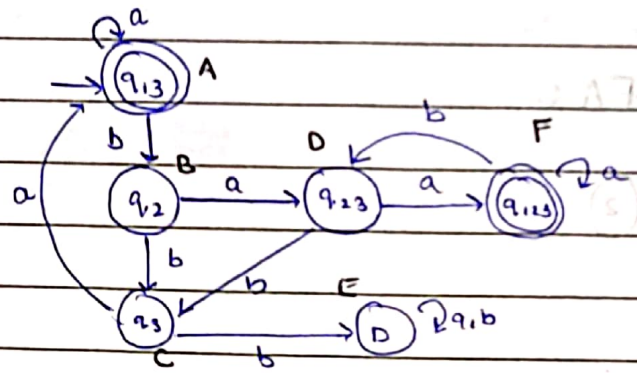
$\delta(q_1q_2q_3, a) = q_1q_2q_3$

$\delta(q_1q_2q_3, b) = q_2q_3$.

∴



∴



① ~~C ≠ D~~ C ≠ D on $w = ba$.

② B ≠ C on $w = ba$.

③ B ≠ D on $w = a$.

④ A ≠ F on $w = ba$.

∴ This is an DFA.

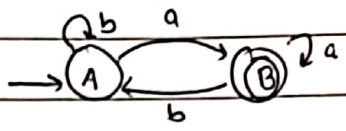
- Power of NFA ≡ Power of DFA.

Therefore a language over some Σ is called regular iff there exist some NFA that recognizes that language.

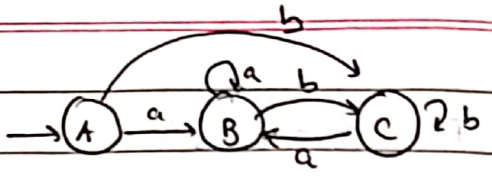
- Two machines M_1 & M_2 are equivalent iff

$$L(M_1) = L(M_2)$$

Ex.



M_1



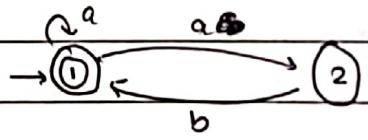
M_2

M_1 is not equal to M_2 but M_1 is equivalent to M_2.

$L(M_1) = L(M_2) = \{w \mid w \text{ ends with a}\}$.

- Every DFA by defⁿ is also NFA.

Ex. Convert to DFA:



Init State: 1

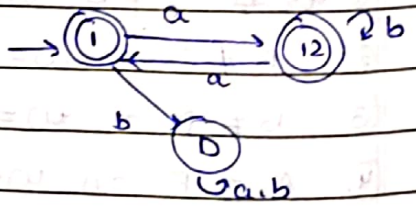
For 1: $\delta(1, a) = \{1, 2\} = 12$

$\delta(1, b) = \phi = D$

For 12: $\delta(12, a) = 12$

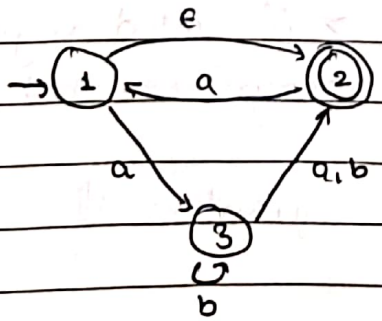
$\delta(12, b) = 1$

$\therefore$



This is also nDFA.

Ex. Convert to DFA:



Init State: 1 2.

For 12: $\delta(12, a) = \{1,2,3\} \cup \{1,2\} = 123$

$\delta(12, b) = \{\} = D$

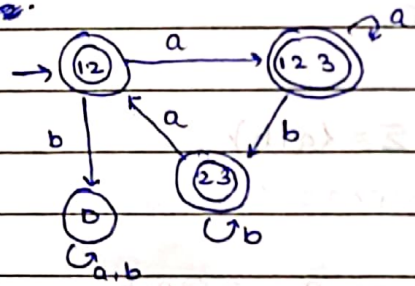
For 123: $\delta(123, a) = 123$

$\delta(123, b) = \{\} \cup \{\} \cup \{3,2\} = 23$

For 23: $\delta(23, a) = \{1,2\} \cup \{2\} = 12.$

$\delta(23, b) = \{\} \cup \{3,2\} = 23.$

For ~~12~~.



This is also mDFA, since all states non-equivalent.

- The algorithm we are using for NFA to DFA conversion is called "Subset Construction".

$NFA(Q, \Sigma, q_0, F, \delta) \longrightarrow DFA(Q', \Sigma, \delta'(q_0, \epsilon), F', \delta')$

where $Q' \subseteq 2^Q$.

$\forall f \in F', \; f \cap F \neq \emptyset$

Ex. Suppose you are converting NFA (with n states) to DFA. What is the max. no. of states in DFA?

2^n.

Ex. For a lang. L, we have NFA with n states. A DFA for L has:

(a) $\leq 2^n$ (b) $= 2^n$ (c) $> 2^n$ (d) $< 2^n$

(c): we can have any arbitrary DFA.

(d): we can't be sure of this.

Ex. Give a language class, for which the mNFA has $\Theta(n)$ states & mDFA has $\Theta(2^n)$ states?

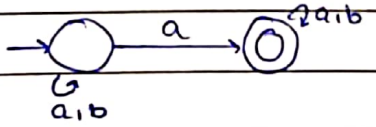
$L = \{w \mid k^{th}$ symbol from right is $x\}$
$\Sigma = \{x, y, \ldots\}$.

mNFA has $(k+1)$ states.
mDFA has (2^k) states.

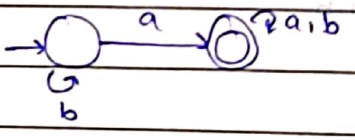
- Minimal NFA is not unique.

Ex. $L = \{w \mid w$ contains $a\}$ $\Sigma = \{a, b\}$

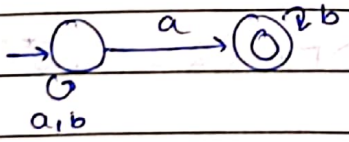
mNFA 1: (any a)



mNFA 2: (first a)



mNFA 3: (last a)



* Regular Expression

- Like DFA & NFA, RE is another model of computation which describes Regular Languages.

- A Regular Expression is an expression (of symbols & operators) which depicts a set of strings. i.e. a language.

Ex. $a^+ \Rightarrow \{a^n \mid n \geq 1\}$

$\epsilon \Rightarrow \{\epsilon\}$

$a+b \Rightarrow \{a, b\}$

- RE is widely used for pattern matching applications, such as lexing, text search, etc.

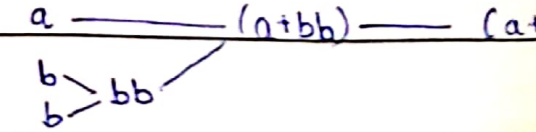
Ex. $(a+b)^* a$

$L = \{w \mid w \in \{a,b\}^* \wedge w \text{ ends with } a\}$

- Let r & s be two regular expressions:

RE	Language	
r , (r)	$L(r)$	
s , (s)	$L(s)$	
r^*	$L(r)^*$	$\{L(r^*)\}$
r^+	$L(r)^+$	$\{L(r^+)\}$
$r \cup s$, $r \mid s$, $r+s$	$L(r) \cup L(s)$	$\{L(r+s)\}$
$r.s$, rs	$L(r) . L(s)$	$\{L(rs)\}$
ϕ	$\{\}$	
ϵ	$\{\epsilon\}$	

Ex. $R = (a+bb)^*$



RE	Lang
a	{a}
b	{b}
bb	{b}•{b}={bb}
a + bb	{a} ∪ {bb} = {a, bb}
(a+bb)*	L(a+bb)* = {a, bb}*

Ex. ϕ^* , ϕ^+

RE	Lang
ϕ	{ }
ϕ^*	{ε}
ϕ^+	{ }

- Priority (Higher to Lower):

( ) > *, + > . > +

* : Kleene Plus

+ : OR

Ex. $\Sigma = \{a, b\}$

RE	Lang
Σ	{a, b}
$\Sigma . \Sigma$	{aa, ab, ba, bb}
$\Sigma\Sigma\Sigma\Sigma$	all strings of len 4
Σ^*	Σ^*

Ex. GO^* , $(GO)^*$

GO^*	$\{G, GO, GOO, GOOO, \ldots\}$
$(GO)^*$	$\{\epsilon, GO, GOGO, \ldots\}$

Ex. $L = \{w \mid w \text{ contains } aa\}$ $\quad \Sigma = \{a, b\}$.

~~aa~~

$\Rightarrow \Sigma^* aa \Sigma^*$

$\Rightarrow (a+b)^* aa (a+b)^*$

Ex. $L = \{w \mid w \text{ contains } a\}$ $\quad \Sigma = \{a, b\}$.

Idea 1: ~~a~~

$\Rightarrow \Sigma^* a \Sigma^*$

$\Rightarrow (a+b)^* a (a+b)^*$

Idea 2: bbb... a ~~ $\quad$ {First a}

$\Rightarrow b^* a (a+b)^*$

Idea 3: ~~ abb... $\quad$ {First b}

$\Rightarrow (a+b)^* a b^*$

- There are three atomic regular expressions:
  (1) ϕ
  (2) ϵ
  (3) for some Σ $\forall a \in \Sigma$, a

Using these and operations: $(), \cdot, +, ^*, ^+$, more sophisticated REs can be created.

Ex. $\Sigma = \{a, b\}$ $L = \{w \mid w \in \Sigma^*, |w| = 4\}$

$\Rightarrow \Sigma\Sigma\Sigma\Sigma$

$\Rightarrow (a+b)(a+b)(a+b)(a+b)$

$\Rightarrow (a+b)^4$ { just for convinence }

Ex. $L = \{ab, abb\}$

$\Rightarrow a(b+\epsilon)b$ or $ab(b+\epsilon)$ or $(ab+abb)$

Ex. $a^* + b^*$

$\Rightarrow L(a)^* \cup L(b)^*$

$\Rightarrow \{a^n \mid n \geq 0\} \cup \{b^n \mid n \geq 0\}$

Ex. $\Sigma = \{0, 1\}$ $L = \{w \mid w$ contains exactly one $0\}$

$1^* 0 1^*$

$\Rightarrow 1^* 0 1^*$

Ex. Atmost one 0.

$1^* + 1^* 0 1^*$ This isn't correct : $1^* (1+0) 1^*$

$\Rightarrow 1^* \epsilon 1^* + 1^* 0 1^*$ No ϵ in this.

$\Rightarrow 1^* (0+\epsilon) 1^*$

- Any expression is a RE iff it can be built using primitive REs and concatenation, union, star & plus operations.

Ex. $a^* b^*$

$a \longrightarrow a^*$
$b \longrightarrow b^*$ $\Big\rangle$ $a^* . b^*$

- The following are not regular expression:

(1) Variable power of RE r.

r^n , where n is variable.

$r^0, r^1, r^2, r^{5+10^{50}}$ all are RE but r^x isn't.

(2) Power dependence b/w two REs: r & s.

$r^n s^n$, where n is variable.

- Concatenation is distributive over union:

$r.(s+t) = r.s + r.t$

Proof:

c1: $r.(s+t) \subseteq r.s + r.t$ {writing $L(r)$ as just r}

let $w \in r.(s+t)$

$xy \in r.(s+t)$ such that $x \in r$, $y \in s$ or $y \in t$

So, $xy \in r.s$ or $xy \in r.t$

$\Rightarrow xy \in rs + r.t$

$\Rightarrow w \in (r.s + r.t)$

c2: $r.s + r.t \subseteq r.(s+t)$

:

Hence Proved

- Union isn't distributive over concatenation, i.e.

$r + st \neq (r+s).(r+t)$

Counterexample: $r = a \quad s = b \quad t = c$.

$L(a+bc)$

$\Rightarrow \{a, bc\}$

$L((a+b)(a+c))$

$\Rightarrow \{aa, ac, ba, bc\}$

Ex. $0^* 1^*$. Analyze

$L(0^* 1^*) \neq \{0^n 1^n \mid n \geq 0\}$

$= \{0^n 1^m \mid n, m \geq 0\}$

Ex. $(01^+)^2$

$\Rightarrow (01^+) . (01^+)$

$L = \{01^n \mid n \geq 1\} . \{01^n \mid n \geq 1\}$

$\Rightarrow \{01^n 01^m \mid n, m \geq 1\}$

Ex. $(01^+)^*$

$L = \{01^n \mid n \geq 1\}^*$

$\Rightarrow \{(01^n)^m \mid n \geq 1, m \geq 0\}$. X This is wrong since () > *,+

So, this wouldn't include:

$(01^2) . (01^3)$

unequal.

$\therefore L = \{01^n \mid n \geq 1\}^*$

$\Rightarrow \{01^n \mid n \geq 1\} \cdot \{01^m \mid m \geq 1\} \ldots$

$L = \{\epsilon\} \cup \{w \mid w$ starts with 0 & every 0 is followed by (atleast) one 1$\}$

Ex. $1^*(01^+)^*$

Some generated strings: $\epsilon, 1, 11, 1^*, 101, 1011 \ldots$

Some not generated: $0, 00, 10, \ldots$

So, $L = \{w \mid$ every 0 is followed by at least one 1$\}$.

or

$\{w \mid w$ has no consecutive 0s & w doesn't end with 0$\}$.

Ex. $(0^*1^*)^*$

I1: Generated: $\epsilon, (0^*1^*)(0^*1^*) \ldots$ n times $\ldots (0^*1^*)$

Each can be 0 or 1.

$\therefore L = \{0,1\}^*$

I2: $(0^*1^*)^* = r^*$

$r = \{\epsilon, 0, 1, 00, 01, 10, 11 \ldots\}$

$r^* = \{\epsilon, 0, 1 \ldots\}^*$

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

Ex. $a^*(\underbrace{ba^*}_{r})^*$

$r = \{b, ba, baa, \ldots\}$

Gen: $a, aa \ldots a^n, b, bb, b^n, ba^n, a^n ba^m, \epsilon$

~~Not Gen: abba~~

$\Rightarrow a^*(ba^*)^*$

$\Rightarrow a^*(\epsilon + ba^* + baba^* + ba^*ba^*ba^* \ldots)$

$\Rightarrow \underbrace{a^*}_{\text{no } b} + \underbrace{a^*ba^*}_{\text{exactly 1 } b} + \underbrace{a^*ba^*ba^*}_{\text{exactly 2 } b} + \ldots$

$\therefore (a+b)^*$

Ex. $a(aa+b)^*$

Gen: $a, aaa, ab, ab^*, a(aa)^*, a^{2n+1}$ $n \geq 0$

Not Gen: ϵ, $b(a+b)^*$, a^{2n} $n \geq 0$, ababa.

L = {w | w starts with a & all other a appear in pair}

Ex. L = {w | w is a string of even len} $\Sigma = \{a, b\}$

$((a+b)(a+b))^*$

Ex. Starts and ends with same symbol.

$(a(a+b)^*a + b(a+b)^*b + a + b)$

- For any RE r:

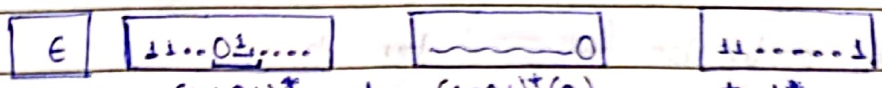
① $r + \emptyset = r$

② $r \cdot \epsilon = r$

③ $r \cdot \emptyset = \emptyset$

④ $r + \epsilon \neq r$

Ex. $\Sigma = \{0,1\}$. No consecutive 0s.



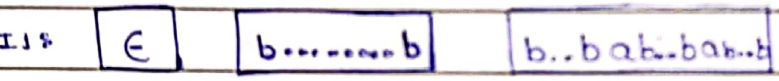
$$(1+01)^*(0+\epsilon)$$

i.e. every 0 is followed by a 1, except if 0 is at last pos.

This can also be written as:

$$1^*(01^+)^*(0+\epsilon)$$

Ex. Containing even no. of as. $\Sigma = \{a,b\}$.



$$b^*(ab^*ab^*)^*$$

I2: Strings can have: b's or a^*b^*a.

$$(b + ab^*a)^*$$

Ex. $(0+10)^*$. This expression is:

(1) No consecutive 1s.

(2) No consecutive 1s, ending with 0?

(3) No consecutive 1s, not ending with 1

(1) Counterex: $1 \notin (0+10)^*$

(2) Counterex: $\lambda \notin$ ending with 0.

(3) ✓

Ex. $\Sigma = \{a, b\}$. All strings not containing ba.

I_1 : b can only be followed by b or ε

$a^* b^*$

Ex. $\Sigma = \{a, b\}^*$. Find RE for Σ^*.

- $(a+b)^*$
- $(a^* b^*)^*$ or $(a^* b^*)^+$
- $(a + ba^*)^*$
- $(a^* + b^*)^*$

Ex. Which are equal to Σ^* for $\Sigma = \{a, b\}$?

(A) $(a+bb)^*$ (C) $(a+bb)^* b^*$

(B) $(a+b)^* b^*$ (D) $a^* (ba^*)^*$

(A) No. Can't generate 'b'.

(B) Yes $L((a+b)^* b^0) = \Sigma^*$

(C) No. Can't generate ba, aba, etc.

(D) Yes.

I_1 : $a^* \underbrace{(ba^*)^*}_{\text{all string starting with b}}$

$\Rightarrow (a^0 + a^1 + a^2 + \dots)(ba^*)^*$

I_2 : $a^* [\underset{\text{No b}}{\varepsilon} + \underset{\text{1 b exactly}}{ba^*} + \underset{\text{2 bs exactly}}{ba^* ba^*} + \dots]$

Ex. Given two REs r & s what is the largest language that can be created?

$$(r+s)^*$$

- For given expr: $a^*(ba^*)^*$ remember:

$$a^*(\epsilon + ba^* + ba^*ba^* + \dots)$$

$$\Rightarrow a^* + a^*ba^* + a^*ba^*ba^* + \dots$$

a^*: no b; a^*ba^*: exactly 1 b; $a^*ba^*ba^*$: exactly 2 b.

Similarly, say $r = a$ $s = bb$, $a^*(bba^*)^*$

$$\Rightarrow a^*(\epsilon + bba^* + bba^*bba^* + \dots)$$

- Some properties of RE: (r is the RE)

(1) $rr^* = r^+$

(2) $(r^*)^* = r^*$

(3) $(r+s)+t = r+(s+t)$

(4) $(r.s).t = r.(s.t)$

(5) $r+s = s+r$

(6) $r.(s+t) = r.s + r.t$

* Equivalence of FA & RE

- For every language over Σ, for which a FA can be created to recognize the language, a RE can also be written, and vice-versa.

i.e.

$\forall_{L \subseteq \Sigma^*}$, FA exists to accept L

$\updownarrow$

RE exists to generate L

Therefore a language is regular iff there exists a RE which can generate the language.

Ex. (A) Every NFA can be converted to equivalent NFA with single accept state.

(B) Every DFA can be converted to equivalent DFA with single accept state.

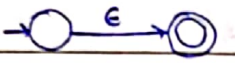
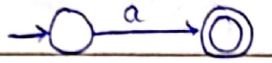
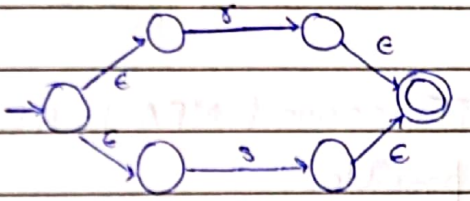
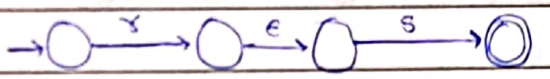
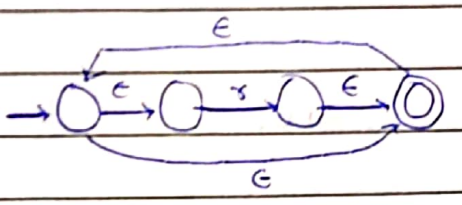
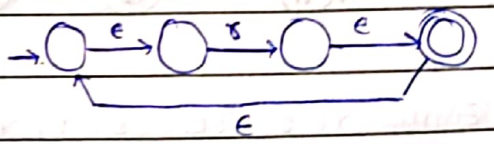
(A) True. Using null moves.

(B) False. Counterex. $\Sigma = \{a, b\}$ $L = \{\epsilon, a, b\}$. Min DFA has 2 final states.

- To prove equivalence of RE & FA, we need to show:

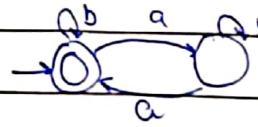
(1) For every RE we can create some NFA.

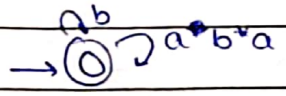
(2) For every FA we can write RE.

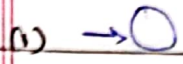
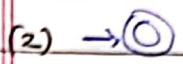
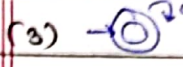
	RE	NFA
(1)	ϕ	
(2)	ϵ	
(3)	Var a	
(4)	$r + s$	
(5)	$r.s$	
(6)	r^*	
(7)	r^+	

- One idea to write complicated language's RE is to create NFA and convert to RE.

Ex. Even no. of a's. $\Sigma = \{a, b\}$

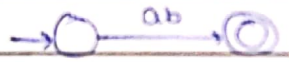


$\equiv$  $\equiv (b + ab^*a)^*$

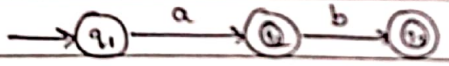
NFA	Lang	RE
(1) 	ϕ	ϕ
(2) 	$\{\epsilon\}$	ϵ
(3) 	Σ^*	$(a+b)^*$

- To convert NFA to RE, remove as many states as possible.

Ex. 



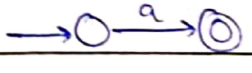
$\therefore$ ab

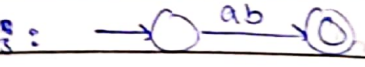
Ex. 

Final re: re(q_2) $\cup$ re(q_3)

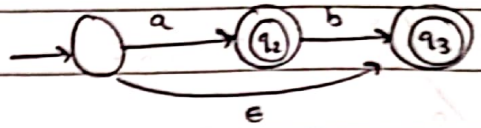
i.e. $\bigcup_{q \in F}$ re(q_f)

For q_2:

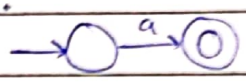
 $r_1 = a$

For q_3:  $r_2 = ab$

$\therefore$ a + ab.

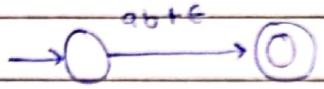
Ex. 

For q_2:



$r_1 = a$

For q_3:



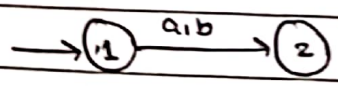
$r_2 = ab + \epsilon$.

$\therefore \; a + ab + \epsilon$

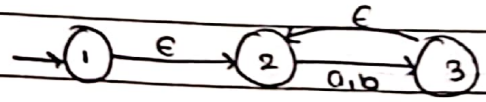
• State Removal

For every reachable state q_i in FA can accept all the strings which from starting state end on that state.

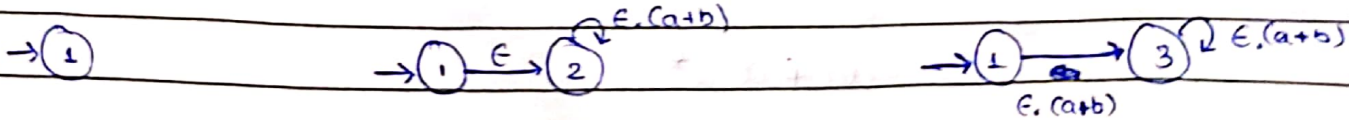
$L(q_i) = \{w \mid \delta^*(q_0, w) = q_i\}$

Ex. 

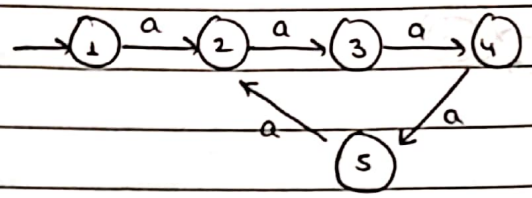
$L(1) = \epsilon$ $L(2) = a+b$

Ex. 

$L(1) = \epsilon$ $L(2) = ((a+b).\epsilon)^* = (a+b)^*$

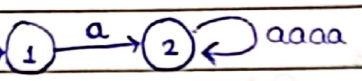


$L(3) = (a+b).(a+b)^* = (a+b)^+$

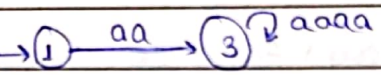
Ex. 

$\Sigma = \{a\}$

$L(1) = \epsilon$ 

$L(2) = a + a(aaaa)^*$ 

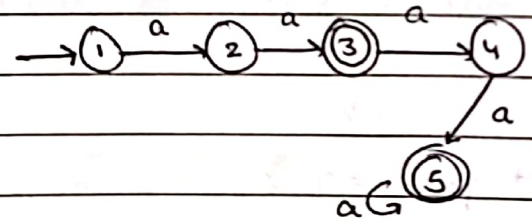
$\Rightarrow a(aaaa)^*$

$L(3) = aa + aa(aaaa)^*$ 

$\Rightarrow aa(aaaa)^*$

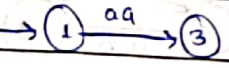
$L(4) = aaa(aaaa)^*$

$L(5) = aaaa(aaaa)^*$

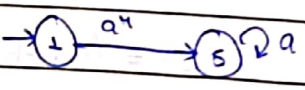
Ex. 

$L(FA) = \bigcup_{q \in F} L(q)$

$\Rightarrow L(3) \cup L(5)$

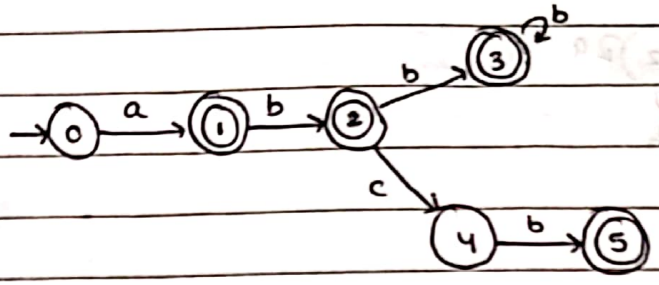
$L(3) = aa$ 

$L(5) = a^4 + a^4a^*$

$= a^4a^*$ 

$\therefore \quad aa + a^4a^*$

Ex.



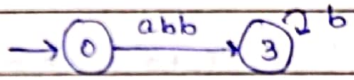
$L(FA) = L(1) \cup L(2) \cup L(3) \cup L(5)$.

$L(1) = a$

$L(2) = ab$

$L(3) = abb + abbb^*$

$= abbb^*$



$L(5) = abcb$

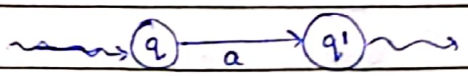
$\therefore L = a + ab + abbb^* + abcb$

- Guidelines for State Removal

(1) $L(FA) = \bigcup_{q \in F} L(q)$

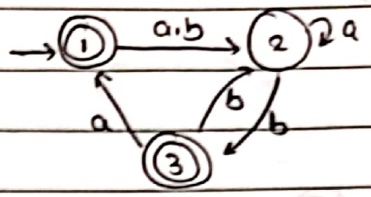
(2) Only keep q_0 & q to find $L(q)$.

(3) Remove all such states q' which don't have path to q, while finding $L(q)$.



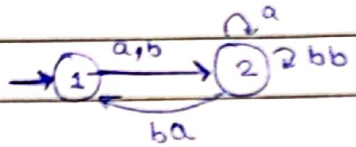
(4) While finding $L(q)$, remove all other q's and take care of all paths going through q'.

Ex.

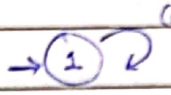


$L(FA) = L(1) + L(3)$

$L(1)$:

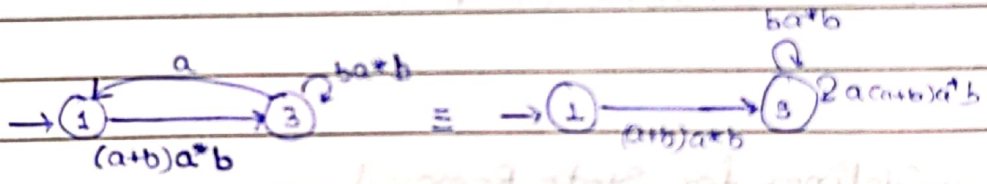


$\Rightarrow$



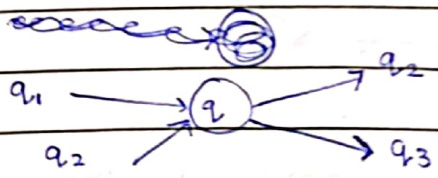
$\therefore L(1) = ((a+b)(a+bb)^* ba)^*$

$L(3)$:



$\therefore L(3) = (a+b)a^*b \cdot (a(a+b)a^*b + ba^*b)^*$

- While removing a state q :



take care of all direct paths :

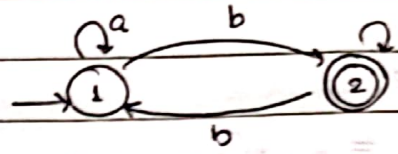
$q_1 \rightarrow q_2$

$q_1 \rightarrow q_3$

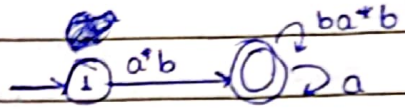
$q_2 \rightarrow q_2$

$q_2 \rightarrow q_3$

classmate
Date
Page

Ex. 

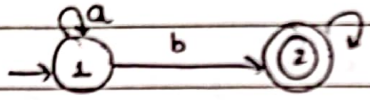
Odd no. of b's.

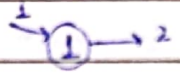


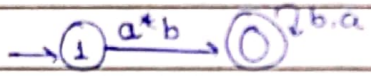
Removing 1

$\Rightarrow$ $a^*b + a^*b(ba^*b + ba)^*$

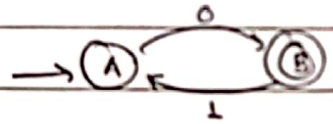
$\Rightarrow$ $a^*b(ba^*b + ba)^*$

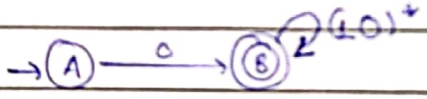
Ex. 

Removing 1 : 



$a^*b(b+a)^*$

Ex. 

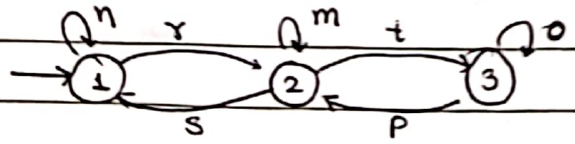


$\Rightarrow$ $0 + 0(10)^*$

$\Rightarrow$ $0.(10)^*$

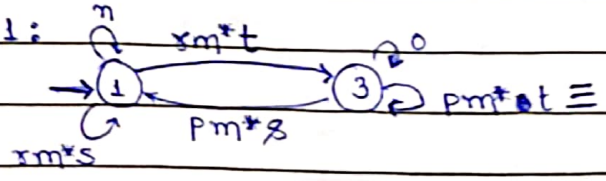
L(A) : 

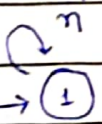
$\Rightarrow \epsilon + (01)^*$

Ex. 

$rm^*t(o + pm^*t)^*pm^*s$

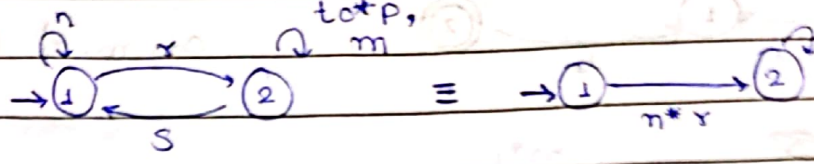
rm^*s.

for 1 : 

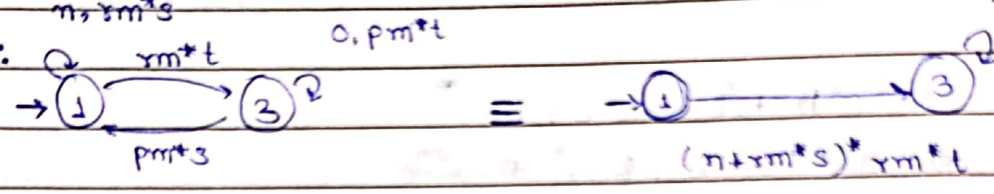


classmate
Date
Page

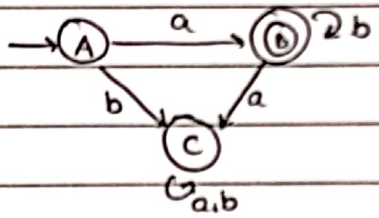
for 2:



for 3:

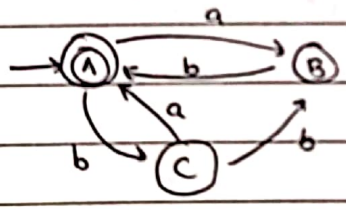


Ex.

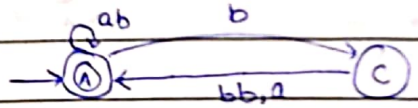


⇒ ab^*

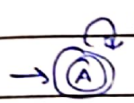
Ex.



Del. B:



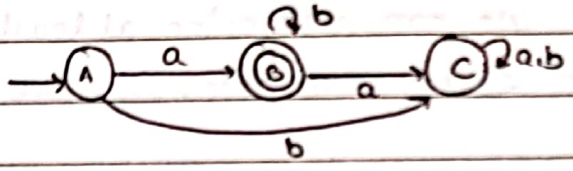
Del C:



$\therefore (ab + bbb + ba)^*$

* Pumping Lemma.

- Given a DFA, any string $w \in \Sigma^*$ goes through a sequence of $(n+1)$ states where $n = |w|$

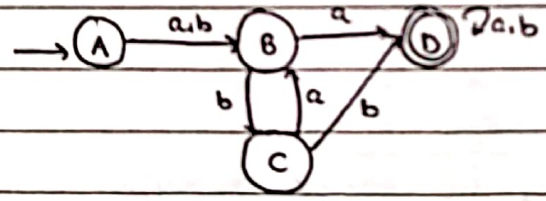
Ex. 

$w = abbbbb$.

a	b	b	b	b	b
A	B	B	B	B	B	B

$\therefore 5+1 = 7$ state sequence.

Also b/w the initial & final state we see state B repeating.

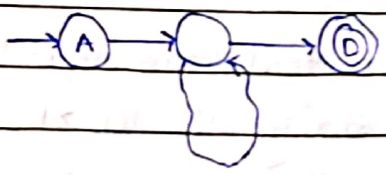
Ex. 

$w = ababaaa$

a	b	a	b	a	a	a
A	B	C	B	C	B	D	D

$w = abababaaa$ ABCBCBCBDD

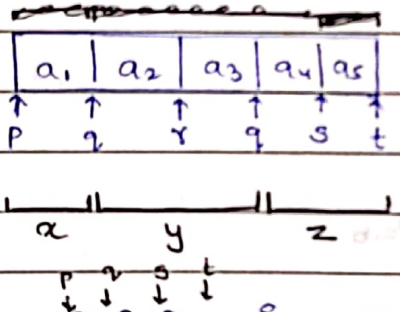
So we can see "ba" as a repeting pattern & hence BC state repeating.



- Suppose we have a w, such that $|w| = K$ where K is the #states in the DFA. We know w will go through a seq. of $(K+1)$ states. Therefore at least one state repeats (by pigeonhole principle).

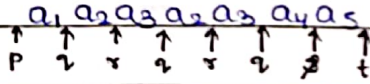
So any w, $|w| \geq K$, we can guarentee at least one state repeats in w's state sequence.

- Suppose we have a string for which state is repeating:



then, $a_1 a_4 a_5$ is xz.

$a_1 a_2 a_3 a_2 a_3 a_4 a_5$ is $xyyz$ or xy^2z.



So we can notice on xyz, DFA went to t.
on xz also DFA went to t,
on xy^2z —"— .

For all xy^nz, $n \geq 0$, DFA goes to the same state as xyz, since state q repeats.

- So if we can guarentee that some repetition happens by taking a string of len $\geq K$, then we can divide the string in three parts:

x y z

such that $\forall n \geq 0$, $xy^n z$ lands on same state as xyz.

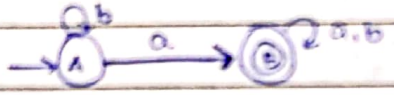
i.e.

$\forall n \geq 0$

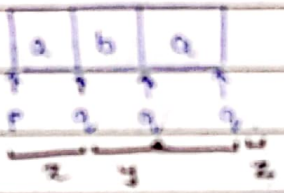
$\delta^*(q_0, xy^n z) = \delta^*(q_0, xyz)$

where q_0 is initial state.

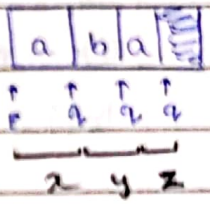
Ex. Say we have following DFA:



for $w = aba$

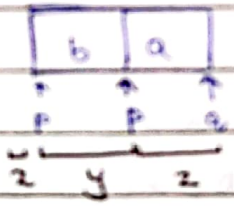


or



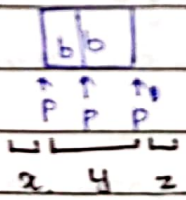
So z can be empty.

for $w = ba$



So, x can be empty.

for $w = bb$



∴ Both x & z can be empty. y can't be empty, for $|w| \geq k$.

- Iff a language L is regular, then there exists a DFA which recognizes the language L, say D. Let's assume D has K states, then if we take a string $w \in L$ such that $|w| \geq K$, then we can guarentee that some state of D repeats and $\delta^*(q_0, w) \in F$, i.e we can divide w.

$$w = xyz$$

So, if $w \in L$, then $xy^n z$ also belongs to L, since $\forall n \ \delta(q_0, xyz) = \delta(q_0, xy^n z) \in F$.

We know that for K state DFA repetition would happen for $|w| \geq K$, and $w = xyz$, so,

$$|xz| \leq K$$

- **Pumping Lemma:** For a language $L \subseteq \Sigma^*$, if L is regular then $\exists$ DFA D, with K states, $K \in \mathbb{N}$, such that, if $w \in L$, $|w| \geq K$, then w can be split into three parts $w = xyz$, such that $|y| \geq 1 \geq |xyz| \leq K$, and $\forall n \ xy^n z \in L$, $n \geq 0$.

The converse is used to prove that a language isn't regular.

The value K would always be greater or eq. to the no. of states in min DFA for L, i.e. $K \geq m$.

- So, if m DFA for L has K states $\forall p \geq k$ is valid pumping length. Pumping length may/maynot be less than K as well.

classmate
Date
Page

- **Theorem:** If L is regular then $\exists p \geq 1$ such that $\forall w \in L$, $|w| \geq p$, we can split w as xyz, such that $|y| \geq 1$, $|xy| \leq p$ and $\forall i \geq 0$ $xy^iz \in L$.

Ex. L = starting with a. $\Sigma = \{a, b\}$.

$L = a(a+b)^*$

p=1:

we only have $w = 'a'$. we can divide w only one way.

$w = x\,a\,y\,z$

So. $|y| \geq 1$ ✓

$|xz| \leq p$ ✓

Now, $xy^iz = a$

$xy^0z = xz = \theta\partial \notin L$

$\therefore$ p=1 doesn't work.

p=2: we have $w = aa$, $w = ab$.

$w = x\,a\,y\,a\,z$ | $x\,a\,y\,a\,z$ | $x\,a\,y\,a\,z$

Let's say $w = x\,a\,y\,a\,z$.

$xy^1z = aa \in L$

$xy^0z = a \in L$

$xy^2z = aaa \in L$

$\vdots$

p=2 works for $\forall w \in L$, $|w| \geq 2$.

- Pumping Length (p) is property of the language not on the DFA.
- For Pumping Lemma for regular lang. L, with some string $w = xyz$, we know that repetition happens within first p symbols & y is the repeated part. so,

$$|xy| \leq p$$

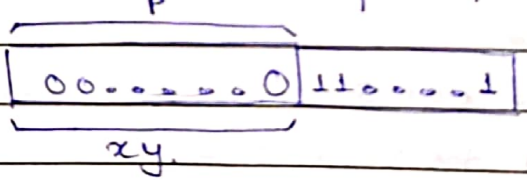
Ex. $L = \{0^n 1^n \mid n \geq 0\}$

Let's assume L is Regular & some $p \geq 1$ exists which satisfies PL, i.e. $\forall w \in L$, $|w| \geq p$, w must be able to pump.

So, $w = 0^p 1^p$ must be able to pump, i.e. $\forall i \geq 0$ $xy^i z \in L$.

Also, $|xy| \leq p$.

So y must be within first p 0s of string $0^p 1^p$.



Let's take $y = 0^k$, $k \leq p$ and $k \geq 1$

$$w = 0^{p-k} 0^k 1^p$$

$$xy^0 z = 0^{p-k} 1^p \notin L \text{ since } p-k \neq p.$$

∴ L doesn't satisfy PL.

∴ L isn't regular

Ex. $L = \{ w \mid \#a = \#b \}$ $\Sigma = \{a, b\}$.

Let's assume L is Reg., i.e. L satisfies PL.

Lets take some arbitrary $p \geq 1$.

Then $\forall w \in L$, $|w| \geq p$, w must be such that $w = xyz$, $|xz| \leq p$, $|xy| \leq p$.

Lets take $w = a^p b^p \in L$.

y again must be somewhere in first p symbols.

$w = a^{p-k} a^k b^p$, $1 \leq k \leq p$.

$xy^iz = a^{p-k} a^k b^p$

$xy^0z = a^{p-k} b^p \notin L$ $\therefore \#a \neq \#b$.

$\therefore$ L doesn't satisfy PL.

Hence L is non-regular.

Ex. $L = \{ a^n b^m \mid n, m \geq 1 \}$

w.K.t if L is regular, then for all strings w, w should be able to pump.

But it doesn't help to prove L is reg. We'll have to check every string w with $|w| \geq p$. We can find the pumping length in finite number of steps.

p = 1 : No string

p = 2 : ab. $w = ab$ $x = a$ $y = b$ $z = \epsilon$.

$w = xyz \in L$. $|xy| \leq p$. $|xz| \leq p$.

$\forall i$ $xy^iz \notin L$

$xy^0z = a \notin L$

p = 3 : abb $x = a$ $y = b$ $z = b$ $\forall i$ $xy^iz \in L$ $\therefore$ Min pumping len = 3.

$|xz| \leq p$.

- Minimum pumping length must be greater than length of min. string of L, since if
  $p \leq$ min string length
  then xy^0z won't be possible since
  $|xy^0z| < |xyz|$
  ↑ xyz is min string

- y in $w = xyz$ represents a substring which can be pumped 0 or more times & the string xy^iz would still be in language.
  We know, $|xy| \leq p$.
  Saying that pumping length is p means than you have the pumping substring (y) within first p symbols.

- If for a language L mDFA has K states then min. Pumping Length is $\leq K$ & for all values $x \geq K$, x is pumping length.

Proof: Any DFA of L with K or more states would repeat state on $|w| \geq K$. So, any such value x can be pumping length. For $x \leq K$, the state sequence may repeat states.
∴ min. PL $\leq K$

- To prove that a language L is not regular using PLemma you prove that all values of pumping length $p \geq 1$, there exist a counterexample w. every partition of which either doesn't satisfy : $|xy| \leq p$: or for some $i \geq 0$ $xy^iz \notin L$.

If $\forall p \geq 1 \; \exists_{\substack{w \in L \\ |w| = xyz}} \left[(|xy| \nleq p) \vee \exists i \geq 0 \; (xy^iz \notin L) \right]$

then L is not regular.

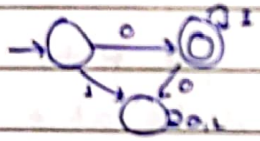
Ex. Find min Pumping Length for $L = 01^*$.

w.k.t

PL > min string len

i.e. $p > 1$, i.e. $p \geq 2$

also



∴ mDFA = 3. ∴ $\forall x \geq 3$ x is PL.

and minPL ≤ 3.

lets check for p = 2

only, w = 01

for w = 01

x = 0 y = 1 z = ϵ

$|xy| \leq 2$ ✓

$\forall i \; xy^iz \in L$

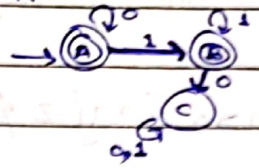
∴ Since for all strings of len 2 or more PLemma holds

mPLen = 2.

Ex. $L = 0^* 1^*$. Find minPLen.

min string = ϵ ∴ mPL > 0.

mDFA:



∴ mPL ≤ 3.

for p = 1:

w = 0 w = 1

for w = 0 x = ϵ y = 0 z = ϵ

- $|xy| \leq 1$ ✓
- $\forall i \geq 0\ xy^i z \in L$ ✓

for w = 1.

x = ϵ y = 1 z = ϵ.

- $|xy| \leq 1$ ✓
- $\forall i \geq 0\ xy^i z \in L$ ✓

So can we say p = 1 is mPL? No. Since for p to be mPL all strings w, $|w| \geq p$ should be able to pump.

Right now we know all strings, $|w| \geq 3$ would be able to pump. We have to check for p = 2 as well.

for p = 2:

w = 00	w = 01	w = 10	w = 11
x = 0 y = 0 z = ϵ	x = 0 y = 1 z = ϵ	x = 1 y = 0 z = ϵ	x = 1 y = 1 z = ϵ
✓	✓	✓	✓

∴ p = 2 is valid PL.

∴ $\forall w \in L$ $|w| \geq 1$ satisfy PLemma.

∴ mPL = 1.

Ex. $L = \phi$. mPL ?

By def of PLemma.
If L is regular
then

$$\exists_{p>0}\ \forall_{w \in L \wedge |w| \geq p}\ \exists_{x,y,z} (w = xyz \wedge |xy| \leq p \wedge \forall_{i \geq 0} xy^i z \in L)$$

So, $p = 1$ satisfies for all strings, since forall on empty set is true. (No counterexample).

- For finite ~~regular~~ lang. L, the min. Pumping Length is K+1 where K is ~~the~~ maximum length string.
  (Every finite lang is regular).

Ex. $L = \{ww \mid w \in \{a,b\}^*\}$.

Let's assume L is regular.
Let p be arbitrarily pumping length. $p \geq 1$.
There exists a $w = a^p a^p$, then $|xy| \leq p$

let $w = xyz$ where $x = a^{p-n}$ $y = a^n$ $z = a^p$

$w = a^{p-n} a^n a^p$ $\forall_{n \geq 1}$

then

$xy^0z = a^{p-n} a^p \notin L$

∴ Violates P.L.

∴ L is not regular.

* Myhill Nerode Theorem

- In a DFA M. for every $w \in \Sigma^*$, w goes to exactly one state. i.e.

$$\forall w \in \Sigma^* \quad \delta^*(q_0, w) \text{ is a single state.}$$

- In a DFA M. for every state q_i, for every $w \in \Sigma^*$, w goes to exactly one state. i.e.

$$\forall q \in Q \; \forall w \in \Sigma^* \quad \delta^*(q, w) \text{ is a single state}$$

- On a DFA with n states, any set of n+1 strings, at least two strings end in same state. (Pegionhole Principle)

So, since DFA is by defn finite states, then any ~~∞ste~~ set of strings, at least two strings end in same state.

Ex. $L = \{a^n \mid n \geq 1\}$. Can there exist a DFA such that for all strings the DFA goes to a unique state?

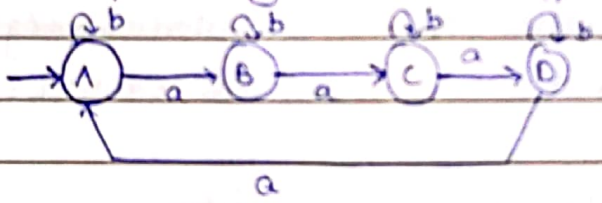
No. DFA by defn is finite, while L is infinite. If for all $w \in L$ DFA were to end on unique state the no. of states would have to be infinite.

Ex. $L = \{a^n \mid 1 \le n \le 10\}$. Can there be some DFA which for each string ends on a different state?

Yes, we can create such a DFA. The DFA would have to have atleast 10 states for ($|L| = 10$) all strings to go to different state.

Ex. $S = \{a^n b^n \mid 1 \le n \le 4\}$. Can there be DFA which for each string ends on a diff. state?

(S here is not the lang. of the DFA. We just want to know if some DFA exists for which this random set of strings end in diff states for diff strings).



$\delta^*(A, a^1b^1) = B$ $\quad\quad$ $\delta^*(A, a^3b^3) = D$

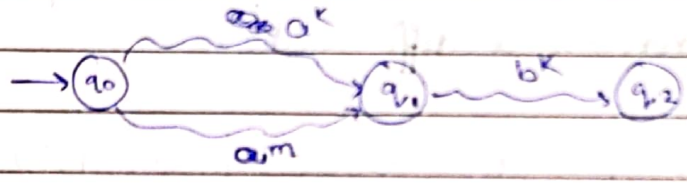
$\delta^*(A, a^2b^2) = C$ $\quad\quad$ $\delta^*(A, a^4b^4) = A$

- **Non-Regular Language:** Any language L for which no DFA exists such that $L(D) == L$, is known as Non-Regular language.

Ex. $L = \{a^n b^n \mid n \geq 0\}$

Lets assume we have a DFA D with n states
Since the lang. L is inf. at least two strings go to same state.

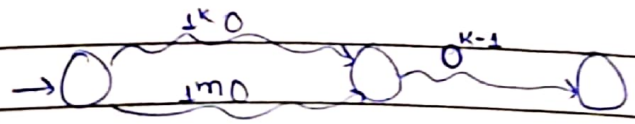
Also from the start state a^k, a^m for some k, m go to same state



Now there's a problem. From q_2, on b^k goes to some state q_{i2}.
Now if $q_{i2} \in F$, the $a^m b^k$ is accepted. Wrong DFA.
if $q_{i2} \notin F$, then $a^k b^k$ isn't accepted. wrong DFA.

∴ No DFA exists for L.
∴ L is non-regular.

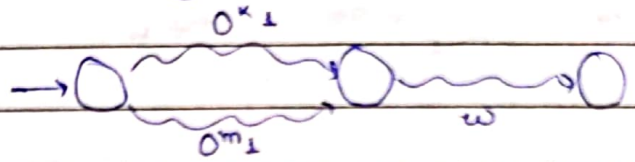
Ex. $L = \{w \in \{0,1\}^* \mid \#0\text{'s} = \#1\text{'s in } w\}$.
This is superset of $L_1 = \{0^n 1^n \mid n \geq 0\}$.
~~Since DFA~~
Let M be a DFA, $L(M) = L$.
Lets assume for $s = \{1^n 0 \mid n \geq 0\}$,
$1^k 0$, $1^m 0$ go to same state ($\because \infty$ lang, Finite state)



This causes problem. ∴ L is non-regular.

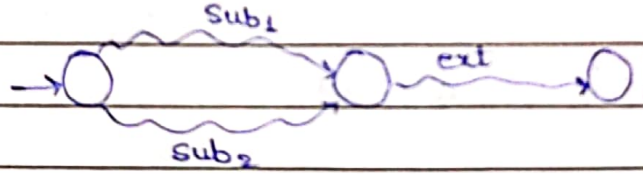
Ex. $L = \Sigma^*$ $\Sigma = \{a, b\}$.

Lets assume DFA exists. Now if we take any substring set S, say $S = \{0^n 1 \mid n \geq 0\}$ and $0^x 1 \neq 0^m 1$, $x \neq m$.



then there exists no extension which causes problem.
∴ L must be regular.

Basically we want a infinite substring set which cause the assumed DFA to go to same state on at least two string (by Pegionhole principle), and then show that there exists some extension which causes problem from that state.



Then L is non-regular.

Ex. $L = \{a^n b^m c^m \mid n, m \geq 0\}$

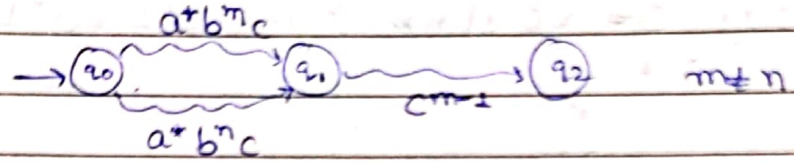
Informal: We have ∞ no. of possibilities to remember, exact count of b: $a^* b^1$, $a^* b^2$, $a^* b^3$, ...

Formal:

If we take $S = \{a^n \mid n \geq 0\}$ then there's no extension which creates problem. Ext: $a^* b^n c^n$, $b^n c^n$, $b^n c^m$, none cause any problem.

Problem is when on same ext (Sub1 ext) & (Sub2 ext), one goes to final & other to non final.

- If we take $S = \{a^n b^m c \mid n, m \geq 0\}$, then for $sub1 = a^* b^m c$ & $sub2 = a^* b^n c$ which go to same state q_1 from input q_0.



$ext = c^{m-1}$ causes problem as either ~~both~~ q_0 can be final or non-final. Not both.

∴ No DFA exists.

∴ L is non-regular.

Ex. $L = \{\omega \in \{a,b\}^* \mid \omega = a^m b^n, m > n\}$

$S = \{a^m \mid m \geq 0\}$ a^l, a^n

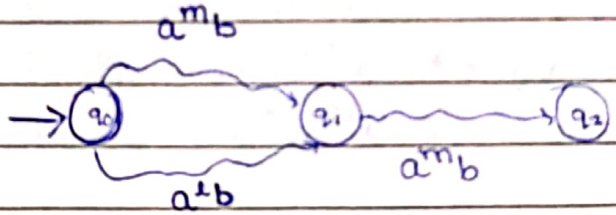
Then there exists extension b^p, $l < p < n$, then $a^l b^p$ doesn't go to final. $a^n b^p$ goes to final.

∴ L isn't regular.

- If for a language L we can show that we have a inf. subset $S = \{ s \mid s \text{ is prefix}(w), w \in L \}$, such that for every two $s_1, s_2 \in S$ for which $\delta^*(q_0, s_1) = \delta^*(q_0, s_2)$, there exist $e \in \Sigma^*$, such that $\delta^*(\delta^*(q_0, s_1), e) \in F \not\leftrightarrow \delta^*(\delta^*(q_0, s_2), e) \in F$ i.e only one belongs to final & other to non-final, then we have contradiction & L is non-regular.

Ex. $L = \{ ww \mid w \in \{a,b\}^* \}$.

- $S = \{ a^n b \mid n \geq 0 \}$, $a^m b$, $a^l b$, $m \neq l$.



$sub_1 e = a^m b a^m b$ $\quad$ $sub_2 e = a^l b a^m b$

∴ Either q_2 is final or non-final.

∴ L is non-regular.

- Distinguishable: For a lang. L, if $w, u \in \Sigma^*$, then w, u are distinguishable iff

$$\exists y \in \Sigma^* \; wy \in L \not\leftrightarrow uy \in L$$

i.e one belongs to language and other doesn't.

- Equivalent Strings: w, u are equivalent iff

$$\forall y \in \Sigma^* \; wy \in L \leftrightarrow uy \in L$$

Ex. $L = \{a^n b^n \mid n \geq 0\}$

(A) a^2, a^{10} (C) $a^{10}b^{20}, a^{10}b^5$

(B) b^2, b^{10} (D) $a^{10}b^{20}, a^{20}b^{40}$

(A) distinguishable by b^2. $a^2b^2 \in L$

$a^2 b^{10} \notin L$.

(B) Equivalent. No y exists which distinguishes them.

$b^2, b^{10} \notin L$.

(C) Dist. by b^5. $a^{10}b^{20}b^5 \notin L$

$a^{10} b^5 b^5 \in L$

(D) Equivalent $\forall y \in S$ $a^{10}b^{20}y \notin L$

$\wedge\ a^{20}b^{40}y \notin L$.

Now can we find an inf. set of strings S such that every two strings in S are distinguishable?

Yes. $S = \{a^n b \mid n \geq 1\}$

So, L is non-regular. We can prove this using S.

- So for a language L, if we can find a inf. set S, such that every two strings in S is distinguishable on L, then L is non-regular.

Ex. $L = \{ w \mid w \text{ is palindrome} \}$. $w \in \{a,b\}^*$

$S = \{ a^n b \mid n \geq 1 \}$

$\forall_{\substack{n,m \\ n \neq m}}$ $a^n b$ is dist. to $a^m b$ by a^n, or by a^m.

Ex. $L = \{ 0^m 1^n \mid m \neq n \}$

$S = \{ 0^m \mid m \geq 1 \}$

$\forall_{\substack{n,m \\ n \neq m}}$ 0^m & 0^n are dist. by 1^m.

$0^m 1^m \notin L$

$0^m 1^n \in L$.

Ex. $L = \{ 0^p \mid p \text{ is prime} \}$

$S = L$ itself.

Distinguishable by $y = 0^y$.

Ex. $L = \{ wcw \mid w \in \{a,b\}^*, c \text{ is symbol} \}$.

$S = \{ a^n c \mid n \geq 1 \}$

$\forall_{\substack{n,m \\ n \neq m}}$ $a^n c$ & $a^m c$ are distinguishable by a^n

$a^n c a^n \in L$

$a^n c a^m \notin L$

(This language solves the problem of string matching. $w = "x \stackrel{?}{=} y"$. If $w \in L$ then $x = y$ else $x \neq y$.)

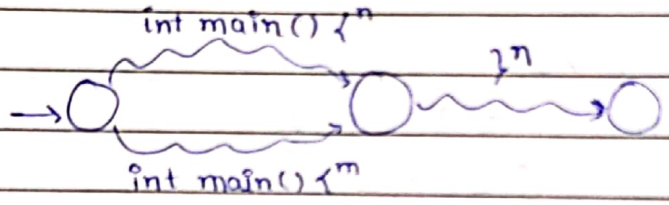
Ex. Prove C lang. isn't regular.

Σ = ASCII

We knows that in C, {} must be balanced.

$S = \{ \text{int main()}\{^n \mid n \geq 1 \}$

Then



∴ For all $m \neq n$ all strings are distinguishable from other. So, C isn't regular.

Ex. $L = \{ 01001000^i1 \mid i \geq 1 \}$

$S = \{ 01001\ldots0^i1 \mid i \geq 1 \}$.

$\forall i \neq j \quad 01\ldots0^i1 \neq 01\ldots0^j1$ are dist. by $0^{i+1}1$.

- Formal Statement of Myhill-Nerode Theorem

Given a lang. L, if there exists an inf. set S of strings such that every two strings of S are distinguishable by L, then L is not regular.

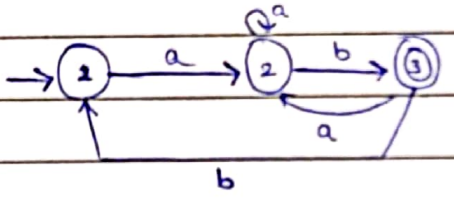
classmate
Date
Page

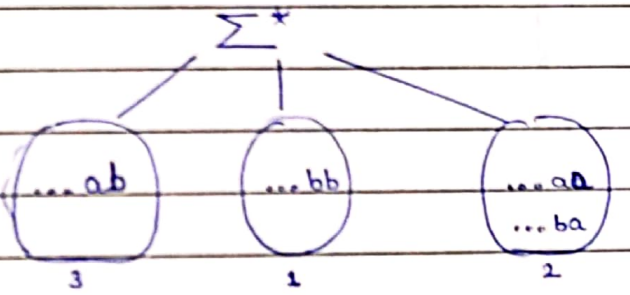
Myhill Nerode Equivalence Classes

A lang. L, if regular, divides Σ^* into equivalence classes for rel. R: Two strings w, u are related iff they are equivalent

$R: \Sigma^* \to$ to Σ^*

Ex. $L = \{ w \mid w \text{ ends with } ab,\ w \in \Sigma^* \}$ $\Sigma = \{a, b\}$.





Ex. Prove that R defⁿ on Σ^* as xRy iff string x & y are equivalent, i.e. $\forall_{z \in \Sigma^*} \delta^*(q_0, xz) \in F \iff \delta^*(q_0, yz) \in F$ is Equivalence relⁿ.

Ref: $\forall x \in \Sigma^*$ x is equivalent to itself. ✓

Symm: If $x \equiv y$ then $y \equiv x$ ✓

Trans.: xRy, $yRz \to xRz$ ✓

∴ R is equivalence relⁿ.

The no. of equivalence classes L divides Σ^* in is equal to the no. of states in mDFA of L.

For non-regular L, the no. of equivalence classes would be ∞.

Context Free Languages, Grammar, PDA

- We have seen till now that Automata's, such as DFA & NFA are Acceptor or Recognizer models, and Regular Expression is a generator model.

* Grammars

- Grammar is another model of computation which describes languages.

Ex. Say English language has a rule like this:

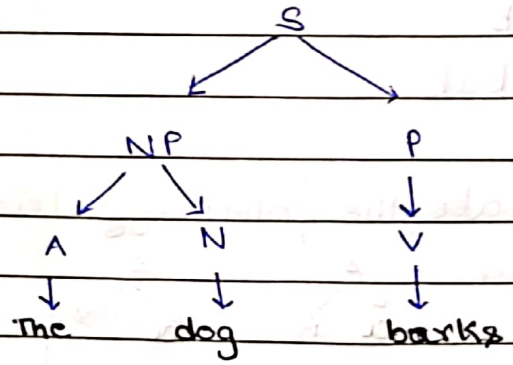
<sentence> → <noun-p><predicate>

<noun-p> → <article> <noun>

<predicate> → <verb>

Using these rules we can generate sentences for English language.

like



- So Grammar is a set of rules which can be used to derive a language, and that would be the language of that grammar.

Ex. <Sentence> → <Noun> eats

Sucah a rule in grammar is called Production Rules.

<Sentence>, <Noun> are variables or Non-Terminals.

"eats", is Terminals

- So a grammar can be described using:
  (1) Set of Variables
  (2) Set of Terminals
  (3) Set of Production Rules
  (4) Start Variable.

Ex. $V = \{A, B\}$ $T = \{a, b\}$

$P = \{A \to aB,\ A \to b,\ B \to a,\ B \to bab\}$

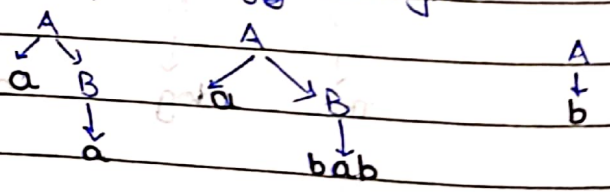
$S = A$

$A \to aB \mid b$

$B \to a \mid bab$

We can generate the following strings:

(1) aa
(2) abab
(3) b.



We can also represent the string generation as derivation:

$A \to aB \to abab$

And, $L(G) = \{b, aa, abab\}$.

- Language of a Grammar is the set of all ~~the str~~ and only strings generated by the grammar.

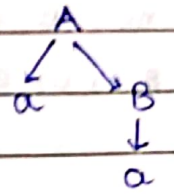
- Comparision of Automata & Grammar

Automata	Grammar
• Starts at initial state	• Starts at initial symbol.
• Transitions to States.	• Transitions to : $A \to aB$
• Has transition rules	• Has production rules.

Ex. $A \to aB \mid b$ $\quad V = \{A, B\}$ $\quad T = \{a, b\}$

$B \to a \mid \epsilon$

- $A \to aB \to a\epsilon$ $\quad a$
- $A \to aB \to aa$ $\quad aa$
- $A \to b$ $\quad b$



- Note: "$X \to \epsilon$" is a special rule which is allowed. Doesn't mean "Null String" (ϵ) is part of terminal set / alphabet set.

- Derivation Length: The no. of productions used in deriving a string.

Ex. For $w = aa$ in above grammar

$A \longrightarrow aB \longrightarrow aa$.

$\therefore$ 2 derivation rules used

$\therefore$ Derivation length = 2.

- To get the yeild or string from parse tree do a left to right depth first search.

Ex. Find lang. of:

$A \to aAb \mid a$.

$A \to aAb \to aab$

$A \to aAb \to aaAbb \to aaabb$

$\vdots$

This behaves like recurrence rel^n with termination cond.

$L = \{ a^n a b^n \mid n \geq 0 \}$

$= \{ a^{n+1} b^n \mid n \geq 0 \}$

Notation: $A \xrightarrow{*} w$

means w can be generated from A in 0 or more steps. ($w \in (V+T)^*$)

Ex. $A \xrightarrow[1]{} 0A1 \mid \underset{2}{B}$

$B \xrightarrow[3]{} \#$

We have three rules.

Rule 3 can only yeild : #

Rule 2 can yeild : B so, #.

Rule 1 can yeild : $0^n A 1^n$ and A can only be replaced with # (with no variable).

$\therefore L(G) = \{ 0^n \# 1^n \mid n \geq 0 \}$

$A \xrightarrow{*} 0^n A 1^n$

$A \xrightarrow{*} 0^n \# 1^n$

The grammar, let's say G_1 could have been simplified as G_2:

$$A \rightarrow 0A1$$
$$A \rightarrow \#$$

Simplifying G_1 to G_2 doesn't change language of the grammar.

$$L(G_1) = L(G_2) \longleftrightarrow G_1 \equiv G_2$$

but the grammars are not same, i.e.

$$G_1 \neq G_2.$$

Ex. $S \rightarrow \overset{①}{SS} \mid \overset{②}{a}$

R_2 gives : a

R_1 gives : $S^n S^m \equiv S^k$, $k \geq 2$.

$$L = \{a\} \cup \{a^k \mid k \geq 2\}$$
$$\therefore L = \{a^k \mid k \geq 1\}.$$

Formal Defⁿ of

- ~~Context Free~~ Grammar : ~~CFG~~ is represented using four tuple :

$$(V, T, P, S)$$

V : Non-empty set of variables.
T : Alphabet / Terminal set
P : Production Rules
S : Start symbol.

- **Context Free Grammar:** The grammar where each production rule is of the form:

$$V \longrightarrow (V+T)^*$$

Note: $(V+T)^*$ doesn't mean we can have ∞ len sequence on RHS

$$L^* = \bigcup_{n \in W} L^n \qquad (W \text{ is whole number})$$

So, RHS is also finite length sequence.

- All grammars are fourtuples (V, T, P, S) only differentiated by the form of production rules.

- We can denote the language of a Grammar G (V, T, P, S) as:

$$L(G) = \{ w \in T^* \mid S \xrightarrow{*} w \}$$

Ex. $S \xrightarrow{①} Sa \mid d\overset{②}{T}$

$T \xrightarrow{③} bTb \mid \overset{④}{c}$.

$L(T) = \{ b^n c b^n \mid n \geq 0 \}$.

$\Rightarrow L(R_2) = \{ d b^n c b^n \mid n \geq 0 \}$ ← $= 0$

$\Rightarrow L(R_1) = \{ S^{\bullet} a^n \mid n \geq 1 \}$ ← > 0

$\Rightarrow L(S) = \{ d b^n c b^n a^m \mid n \geq 0, m \geq 0 \}$

Ex. $L = \{a^n b^{n+1} \mid n \geq 0\}$. Write grammar.

$a^n b^n b \Longrightarrow \quad S \rightarrow Ab$

$\Downarrow$

$A \rightarrow aAb \mid \epsilon$

∴ Grammar :

$S \rightarrow Ab$

$A \rightarrow aAb \mid \epsilon$

Ex. Find language of :

$S \rightarrow \underset{①}{a}\underset{②}{Ab} \mid \underset{②}{\epsilon}$

$A \rightarrow \underset{③}{a}\underset{④}{Ab} \mid \underset{④}{\epsilon}$

$L(R_1) = \{\epsilon\}$

$L(R_2) = \{a^n b^n \mid n \geq 1\}$

$\Rightarrow L(A) = \{a^n b^n \mid n \geq 0\}$ i.e. $L(R_1) \cup L(R_2)$

$L(R_2) = \{\epsilon\}$

$L(R_1) = \{a^n a^m b^m b^n \mid n \geq 1, m \geq 0\}$

$= \{a^l b^l \mid l \geq 1\}$

$L(S) = L(R_1) \cup L(R_2)$

$= \{a^l b^l \mid l \geq 0\}$

Ex. Write CFG for RE a^*

$S \rightarrow aS \mid \epsilon$

$S \rightarrow a^+$ why is this wrong?

RHS should be made up of terminals & non-terminals.
a^* isn't either.

Ex. Write CFG for: $a^* b$. $\Sigma = \{a, b\}$

$a^* b \Rightarrow S \rightarrow Ab$.

$\Downarrow$

$A \rightarrow aA \mid \epsilon$

$\therefore$ $S \rightarrow Ab$

$A \rightarrow aA \mid \epsilon$

Ex. CFG for: $a(b+c^*)$

$S \rightarrow aB$

$B \rightarrow b \mid C$

$C \rightarrow cC \mid \epsilon$

Ex. CFG for: $a^* b^*$

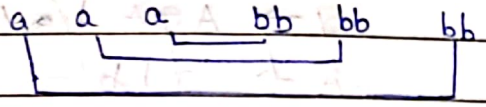
$S \rightarrow AB$

$A \rightarrow aA \mid \epsilon$

$B \rightarrow bB \mid \epsilon$

Ex. CFG for $L = \{a^n b^{2n} \mid n > 0\}$

$a^n (bb)^n$



So,

$S \rightarrow aSbb \mid \epsilon$

Ex. CFG for $L = \Sigma^*$ $\Sigma = \{a, b\}$

$S \rightarrow aS \mid bS \mid \epsilon$

Ex. CFG for $L = \Sigma^+$ $\Sigma = \{a, b\}$

$S \rightarrow aS \mid bS \mid a \mid b$

Ex. CFG for $L = \{a, ab, abab\}$

$S \rightarrow a \mid ab \mid abab$

Ex. Simplify: $S \rightarrow aA \mid B$; $A \rightarrow a \mid B$; $B \rightarrow \#$

- We can replace B with #

  $S \rightarrow aA \mid \#$; $A \rightarrow a \mid \#$; $B \rightarrow \#$

- B is unreachable now. We can remove B's rules.

  $S \rightarrow aA \mid \#$; $A \rightarrow a \mid \#$

Ex. L = All even len palindrome. $\Sigma = \{a, b\}$

$S \rightarrow ASA \mid \epsilon$ $S \rightarrow aSa \mid bSb \mid \epsilon$

$A \rightarrow aAa$

Ex. L = All palindrome strings. $\Sigma = \{a, b\}$

$S \rightarrow ASA \mid aA \mid \epsilon$

$A \rightarrow a \mid b$

$S \rightarrow aSa \mid bSb \mid a \mid b \mid \epsilon$

Ex. $L = \{a^n b^m \mid n > m\}$

we can write this as: $a^l a^m b^m$

where $a^{l+m} = a^n$.

a^l $a^m b^m$

$\Downarrow$

$A \rightarrow aAb \mid \epsilon$

$B \rightarrow aB \mid a$

$\therefore S \rightarrow BA$

$B \rightarrow aB \mid a$

$A \rightarrow aAb \mid \epsilon$

Ex. $L = \{a^n b^m \mid n \neq m\}$

$\Rightarrow \{a^n b^m \mid n > m\} \cup \{a^n b^m \mid n < m\}$

$S \rightarrow S_1 \mid S_2$

$S_1 \rightarrow B_1 A_1$

$B_1 \rightarrow aB_1 \mid a$

$A_1 \rightarrow aA_1b \mid \epsilon$

$S_2 \rightarrow aB_2 \mid A_2$

$B_2 \rightarrow aB_2 \mid a$

$A_2 \rightarrow aA_2b \mid \epsilon$

Ex. $L = \{w \mid \#a's = \#b's\}$ & $\Sigma = \{a, b\}$

$S \rightarrow aSb \mid bSa \mid \epsilon \mid SS$

Ex. L = Balanced Paranthesis $\Sigma = \{(,)\}$

$S \rightarrow (S) \mid SS \mid \epsilon$

Ex. $L = \{a^n b^m a^{2m} \mid n, m \geq 0\}$

$S \rightarrow aSaa \mid B$
$B \rightarrow Bb \mid \lambda$

- Sometimes writing grammar isn't easy. Verify grammar using some strings which are in grammar's language and some which can't be generated, and also from small strings like ϵ, a, b, ab, etc.

Ex. L = Strings starting and ending with same symbol. $\Sigma = \{a, b\}$.

$S \rightarrow aAa \mid bAb \mid a \mid b$
$A \rightarrow Aa \mid Ab \mid \epsilon$

Ex. L = #a's > #b's. $\Sigma = \{a, b\}$.

$S \rightarrow Aa \mid ES \mid SEA$
$A \rightarrow aA \mid \epsilon$ } a^*
$E \rightarrow aEb \mid bEa \mid EE \mid \lambda$ } #a's = #b's

Ex. Grammar for basis arithmetic expressions.
+, -, x, /, ( )

$E \rightarrow E\ O\ E \mid (E) \mid \text{digit}$
$O \rightarrow + \mid - \mid x \mid \div$

- Linear Grammar: A CFG in which on RHS of any production rule, atmost one variable can occur.

Ex. $S \rightarrow A$
$A \rightarrow aB \mid \lambda$
$B \rightarrow Ab$

$\Rightarrow$

$S \rightarrow aB \mid \lambda$
$B \rightarrow Sb$

$\rightarrow$

$S \rightarrow aSb \mid \lambda$

$\therefore L = \{a^n b^n \mid n \geq 0\}$.

- Context-Free Language: A lang. L is called CFL iff we can write some Context Free Grammar for it.

- Regular Grammar:

Right Linear
$V \rightarrow T^*V \mid T^*$

Left Linear
$V \rightarrow V T^* \mid T^*$

A grammar G is Regular Grammar iff all productions are right linear or all productions are left linear.

Ex. $S \to abS \mid \epsilon$

CFG ✓
LLG ✗
RLG ✓
RegG. ✓

Ex. $S \to Sa \mid A$
$A \to aA \mid \epsilon$

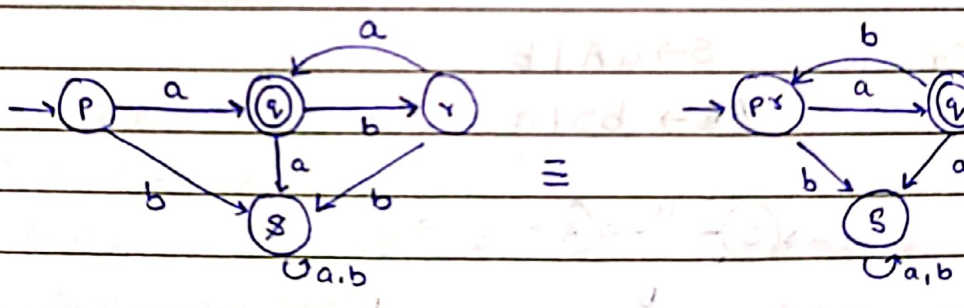
CFG ✓
LLG ✗
RLG ✗
Reg.G ✗

Ex. $S \to aSa \mid \epsilon$

CFG ✓
LLG ✓
RLG ✗
Reg.G ✓

Ex. $S \to abS \mid a$

$L = \{(ab)^n a \mid n \geq 0\}$ $\quad (ab)^* a$.



- Every RLG is Reg. Grammar.
  Every LLG is Reg. Grammar.

Ex. $S \to \overset{①}{a}S \mid \overset{②}{S}\overset{③}{b} \mid \epsilon$ Not regular.

- $L(\text{Rule}_1) = \{a^n \mid n \geq 1\}$
  $L(R_2) = \{b^m \mid n \geq 1\}$
  $L(R_3) = \{\epsilon\}$
  $\therefore L(S) = \{a^n b^m \mid n, m \geq 0\}$

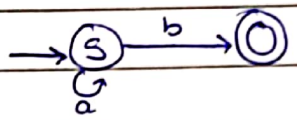
(We can see that Gram. isn't reg. but lang. is regular)

If grammar G is regular then it generates regular lang.
For regular lang. L there exists some regular grammar G.

Ex. [RLG to NFA]

$S \rightarrow aS \mid b$

We said earlier variables are analogous to states.

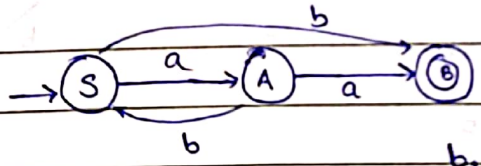


a^*b.

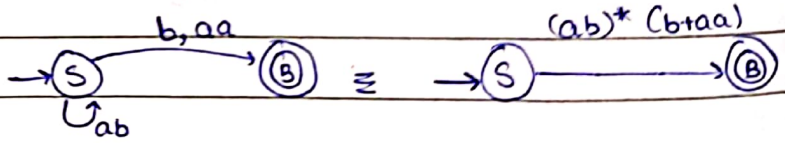
terminals behave like final state.

Ex. $S \rightarrow aA \mid b$

$A \rightarrow bS \mid a$.



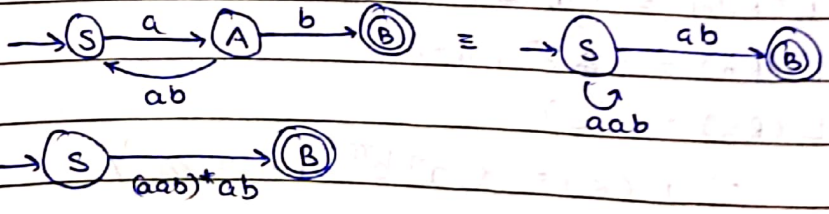
removing A:



$\therefore (ab)^* (b+aa)$

Ex. $S \rightarrow aA$

$A \rightarrow abS \mid b$

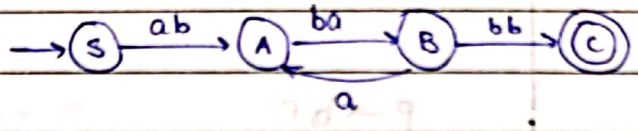


$\therefore (aab)^* ab$

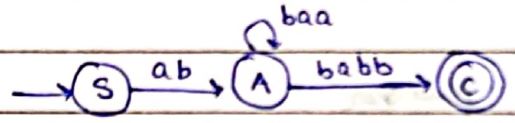
Ex. $S \rightarrow abA$

$A \rightarrow baB$

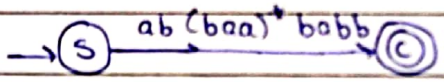
$B \rightarrow aA \mid bb$



$\equiv$



$\equiv$



$\therefore \quad ab(baa)^* babb.$

- RLG to NFA

(1) Variables behave like states.

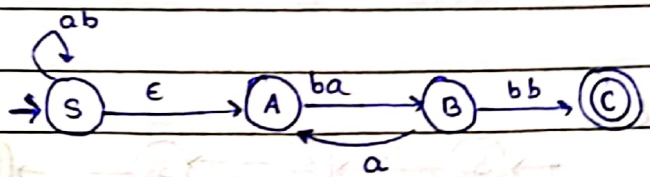
(2) Terminals or T^* take you to some final state.

(3) Production rules behave like transitions.

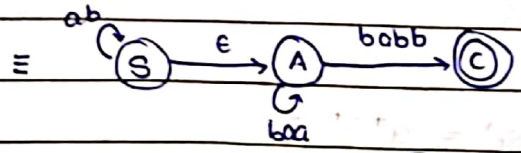
Ex. $S \rightarrow abS \mid A$

$A \rightarrow baB$

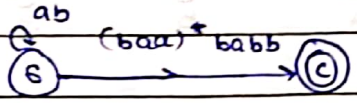
$B \rightarrow aA \mid bb$



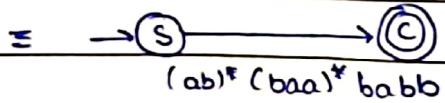
$\equiv$



$\equiv$

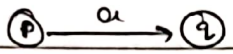
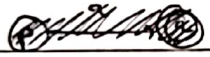


$\equiv$

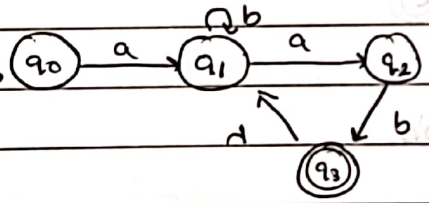


$\therefore (ab)^*(baa)^* babb$

- RNFA to RLG

	$P \rightarrow aq$
	$P \rightarrow \epsilon$
	$P \rightarrow bP$
	

Ex.



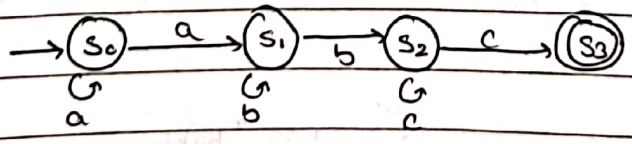
$q_0 \rightarrow a q_1$

$q_1 \rightarrow b q_1 \mid a q_2$

$q_2 \rightarrow b q_3$

$q_3 \rightarrow \lambda q_1 \mid \lambda$ {$\because q_3$ is final, $\therefore q_3 \rightarrow \lambda$}

Ex.

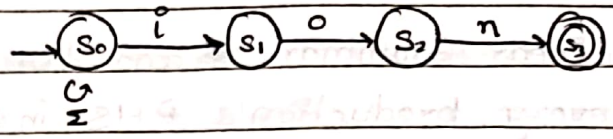


$S_0 \rightarrow a S_0 \mid a S_1$ $L(S_0) = a^+ b^+ c^+$

$S_1 \rightarrow b S_1 \mid b S_2$

$S_2 \rightarrow c S_2 \mid c S_3$

$S_3 \rightarrow \lambda$

Ex. 

$\Sigma = \{i, o, n, a\}$.

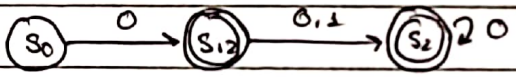
$S_0 \rightarrow iS_0 \mid oS_0 \mid nS_0 \mid aS_0 \mid iS_1$

$S_1 \rightarrow oS_2$

$S_2 \rightarrow nS_3$

$S_3 \rightarrow \lambda$.

$L = \Sigma^* ion$

Ex. 

$S_0 \rightarrow 0S_{12}$

$S_{12} \rightarrow 0S_2 \mid 1S_2 \mid \lambda$ $\quad L = 0(0+1)0^*$

$S_2 \rightarrow 0S_2 \mid \lambda$

Ex. G: $S \rightarrow a \mid abl \; Abbb$ $\quad$ L(G).

$A \rightarrow Aa \mid \epsilon$

$L(A) = a^*$ $\quad L(G) = L(S) = a + ab + a^* bbb$

Lets reverse RHS of each production.

G_2: $S \rightarrow a \mid ba \mid bbbA$

$A \rightarrow aA \mid \epsilon$

$L(A) = a^*$ $\quad L(G_2) = L(S) = a + ba + bbba^*$

We can see that $L(G_2)$ is $L(G)^R$, i.e. reverse of $L(G)$.

- If G is a Left Linear Grammar, and let $L(G) = L$. If we reverse every production's RHS in G & obtain grammar G', then $L(G') = L^R$.

- **Derivation:** The derivation of a string w with respect to some CFG is the sequence of substitutions from start ~~symbol~~ variable that yields w.

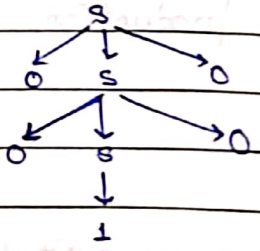
Ex. $S \rightarrow 01 | 1 | 0S0 | 1S1 | \epsilon$ $\quad w = 00100$

$S \rightarrow 0S0 \rightarrow 00S00 \rightarrow 00100$

∴ length of derivation = 3.

- **Parse Tree:** Tree repr. of derivation of a string w from start symbol. (aka. Derivation Tree)

Ex. Previous example.



- **Sentential Form:** From start symbol whatever you can derive is Sentential Form, in 0 or more steps.
  if $S \xrightarrow{*} \alpha$, then α is sentential form.

- **Sentence:** A sentential form with only terminals.

Ex. $S \rightarrow SS \mid a$. $\quad w = aa$.

$S \rightarrow SS \rightarrow Sa \rightarrow aa$.

$S \rightarrow SS \rightarrow aS \rightarrow aa$.

We can have multiple derivations for some strings for a given CFG.

- **Leftmost Derivation:** Whenever in a Sentential form you have choice b/w variables to replace, replace the leftmost variable.

- **Rightmost Derivation:** Whenever in a Sentential form you have choice b/w variables to replace, replace the rightmost variable.

Ex. $S \rightarrow AB$
$A \rightarrow aaA \mid \lambda$ $\quad w = aab$.
$B \rightarrow Bb \mid \lambda$

Leftmost Derivation: $S \rightarrow AB \rightarrow aaAB \rightarrow aaB \rightarrow aaBb \rightarrow aab$

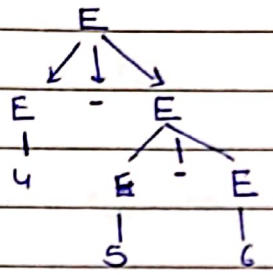
Rightmost Derivation: $S \rightarrow AB \rightarrow ABb \rightarrow Ab \rightarrow aaAb \rightarrow aab$

Ex. Consider the following grammar:

G: $E \rightarrow E+E \mid E-E \mid 4 \mid 5 \mid 6$.

$w = 4-5-6$

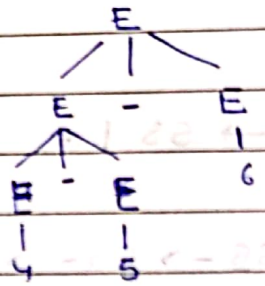
We can have two parse trees.



$\Rightarrow 4-(5-6)$

$\Rightarrow 4-(-1)$

$\Rightarrow 5$



$\Rightarrow (4-5)-6$

$\Rightarrow -1-6$

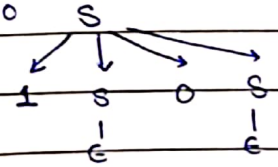
$\Rightarrow -7.$

Results vary drastically.

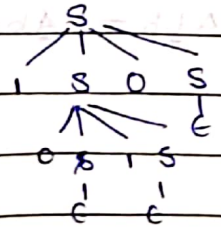
- Ambiguity of CFG: A CFG G is called ambiguous, iff there exists some string w for which there are more than one parse tree. ($w \in L(G)$)

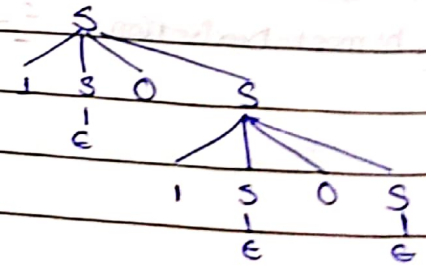
Ex. $S \rightarrow 1SOS \mid 0S1S \mid \epsilon$

$w = 10$



$w = 1010$





- Ambiguity can also be defⁿ in terms of LMD & RMD.

G is Ambig. $\longleftrightarrow$ ∃w. there are more than one LMDs.

G is Ambig. $\longleftrightarrow$ ∃w. there are more than one RMDs.

Ex. $S \to 0S1S \mid 1S0S \mid \epsilon$

$w = 1010$

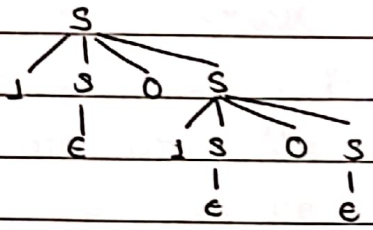
① $S \to 1S0S \to 1\epsilon0S \to 1\epsilon0S1S0S \to 1010S \to 1010$

② $S \to 1S0S \to 10S1S0S \to 101S0S \to 1010S \to 1010$.

∴ Ambiguous grammar.

- For a parse tree we have unique LMD & unique RMD & vice-versa.

Ex.



$S \to 1S0S \to 10S \to 101S0S \to 1010S \to 1010$

- W.K.t for every Regular Language, there exists either some Left Linear Grammar or some Right Linear Grammar. Linear Grammar by defn is also CFL.

∴ Every Regular language is also Context Free Language by defn.

Ex. $(a+b)^*a(a+b)^*$

$S \rightarrow AaB$
$A \rightarrow a \mid b$
$B \rightarrow aB \mid bB \mid \epsilon$

This grammar is neither LLG nor RLG, not even Linear Grammar but it is CFG.

So, for every Regular language there exists some CFG. A CFG G is language L(G) may or maynot be Regular.

* Push Down Automata

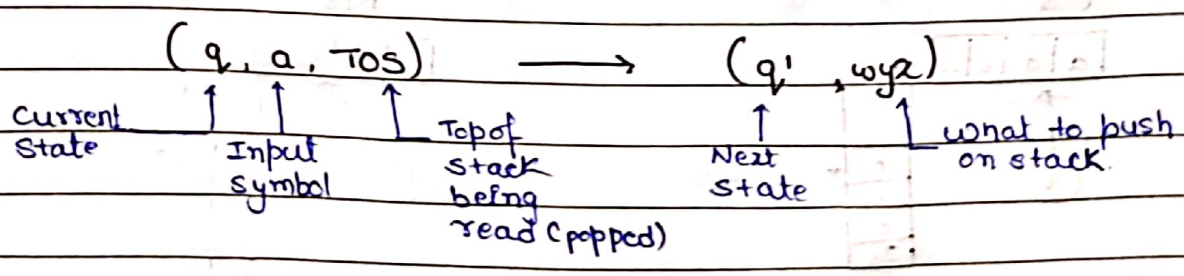
- We started with DFA and were able to recognize all Regular Languages. We then gave three additional powers to DFA and formed NFA in hope of recognizing some more languages than DFA. NFA was proved to be equally powerful as DFA, i.e all the lang. over some Σ recognizable by some NFA is also recognizable by some DFA.

- Both DFA & NFA had finite states. Each state was acting as memory. So FA had finite memory. So, let's give FA the power of infinite m/m. We can give the additional ∞ m/m in terms of array, queue, tree, etc.

- PDA is a Finite Automata with additional infinite memory in the form of stack.

- In infinite stack we keep track of a special symbol Z_0, which indicate bottom of stack.
At any time only the top of the stack is visible.
New symbols can be pushed & popped off from the stack.

- Push Down Automata: Finite Automata + 1 Stack (∞).
(NFA)
- Move / Transition in PDA:

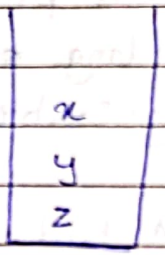


- Reading Input Tape: Reading current symbol on input tape.

- **Null Move:** Without reading current symbol make transition.

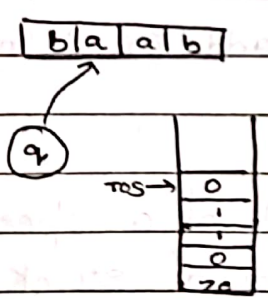
- **Reading Stack:** Consuming / Popping the ToS.
- **Null Move on Stack:** Make transition without reading ToS.
- **Convention:**

Push $\overleftarrow{xyz}$



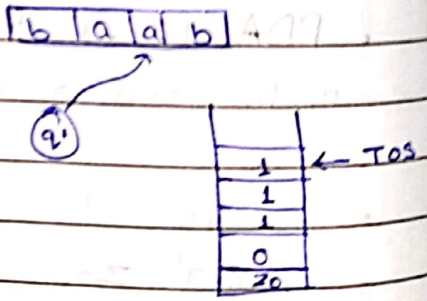
Ex. $(q, a, 0) \longrightarrow (q', 1)$

Current State.



$\Rightarrow$

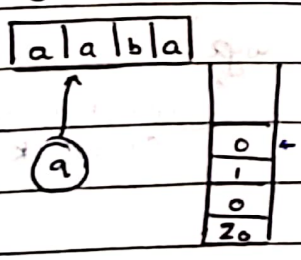
Next State



0 is read from stack & 1 is pushed.

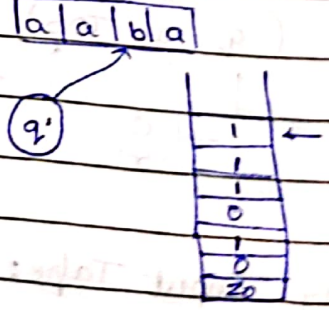
Ex. $(q, a, 0) \longrightarrow (q', 1110)$

CS



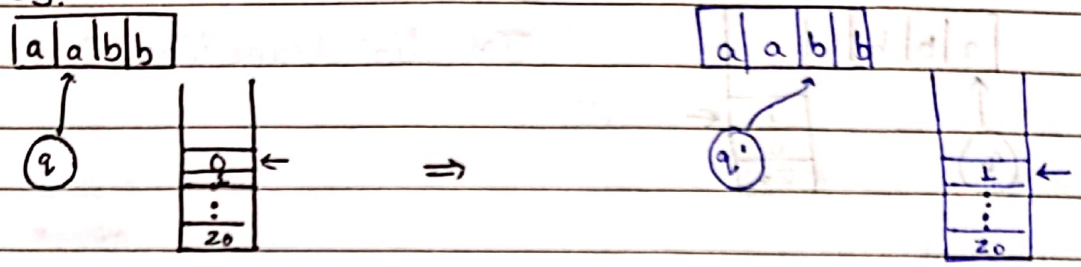
$\Rightarrow$

NS



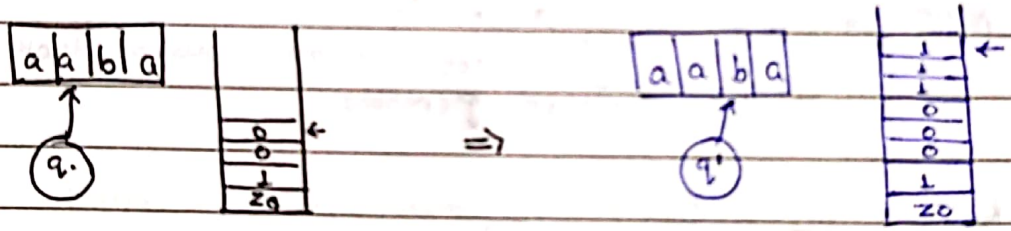
Ex. $(q, a, 0) \longrightarrow (q', \epsilon)$

CS.



Pop 0. Push Nothing.

Ex. $(q, a, \epsilon) \longrightarrow (q', 1110)$



Pop Nothing. Push 1110 ←

Ex. $(q, a, 01) \longrightarrow (q', 1110)$

Invalid transition. We can only read Top of stack.

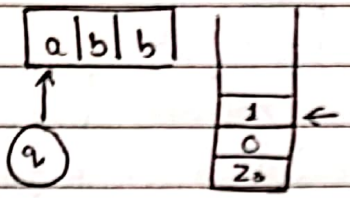
Ex. $(q, ab, 0) \longrightarrow (q', 01)$

Invalid Transition. Only one symbol can be read from input tape at a time.

Ex. $(q, \epsilon, 0) \longrightarrow (q', \epsilon)$

Valid trans. Don't read any symbol from input tape. Pop 0. Don't push anything.

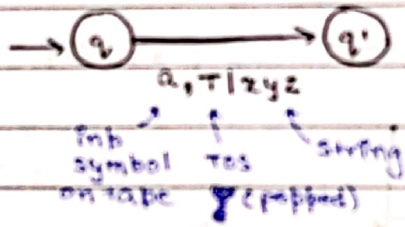
Ex. $(q, a, 0) \longrightarrow (q', \epsilon)$



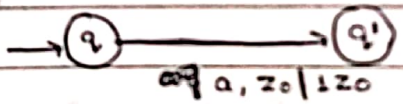
Invalid transition. 0 isn't TOS.

Ex. Design PDA for $L = \{a^n b^n \mid n \geq 1\}$

- Convention:



Ex.

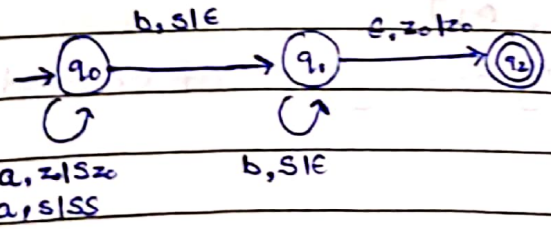


This means:

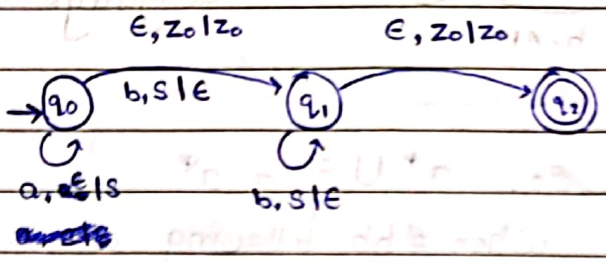
$(q, a, Z_0) \longrightarrow (q', 1Z_0)$

or

$(q, a, \epsilon) \longrightarrow (q', 1)$



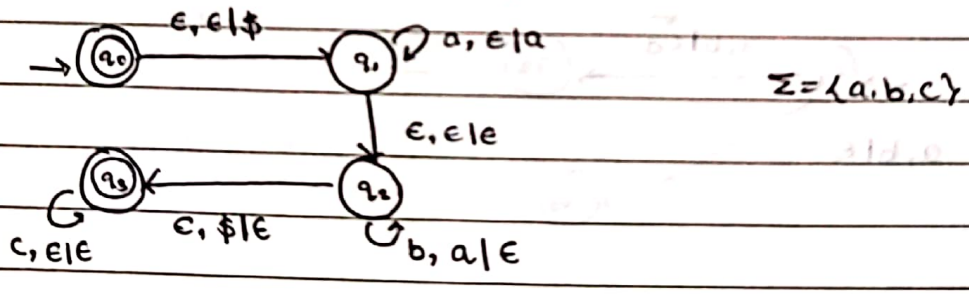
Ex. $L = \{a^n b^n \mid n \geq 0\}$



- Acceptance of String: After consuming the entire string the PDA can go to some final state.

- By default PDA is non-deterministic, i.e. PDA ≡ NPDA.

Ex. Lang. of the following PDA:



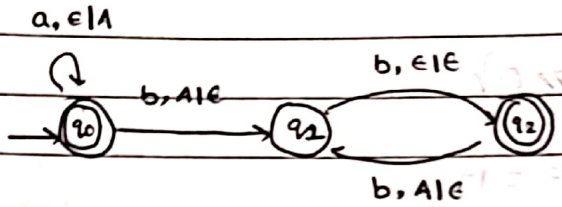
$\Sigma = \{a, b, c\}$

Strings accepted: ϵ, ab, c, aabb, $\{a^n b^n c^m, n \geq 0 \& m \geq 0\}$
Strings not accepted: abb, aab, abbc, etc.

$\therefore L = \{a^n b^n c^m \mid n, m \geq 0\}$

- w = aabbcccab Accepted or rejected?
Rejected can't read ab after aabbccc, so string isn't entirely consumed.

Ex.



Language ?

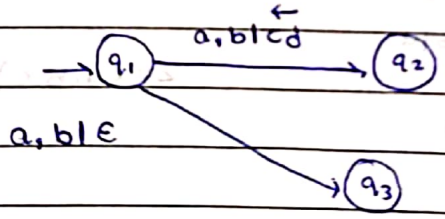
q_0 is final state. So, $a^* \cup \epsilon = a^*$

q_2 is f.s. accepts when #bb following a^n is less than equal to n.

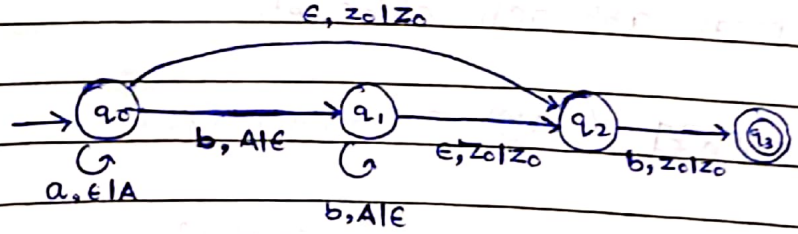
$$L = \{a^n (bb)^m \mid n \geq m, n.m \geq 0\}$$

Ex. what does the following transition rule for PDA (by default NPDA) means ?

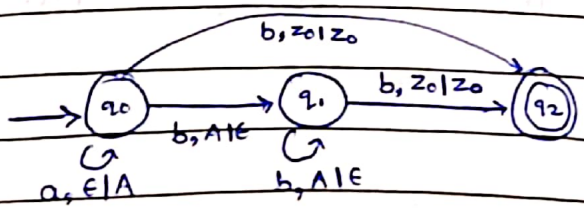
$$\delta(q_1, a, b) = \{(q_2, cd), (q_3, \epsilon)\}$$



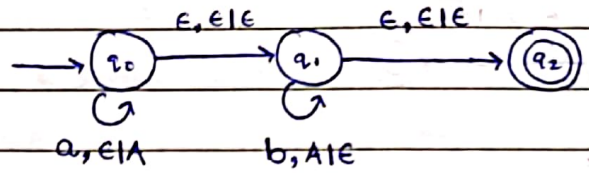
Ex. $L = \{a^n b^{n+1} \mid n \geq 0\}$



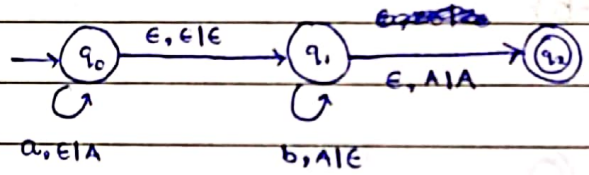
OR



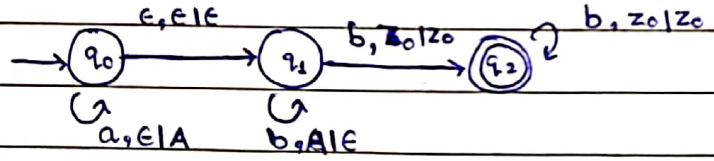
Ex. $L = \{a^n b^m \mid n \geq m \geq 0\}$



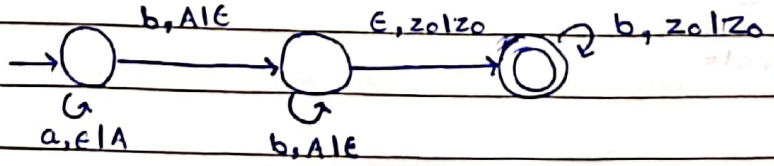
Ex. $L = \{a^n b^m \mid n > m \geq 0\}$



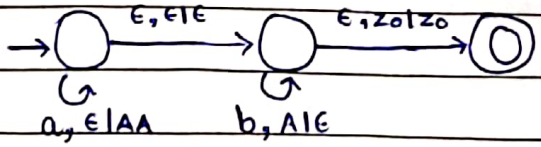
Ex. $L = \{a^n b^m \mid m > n \geq 0\}$



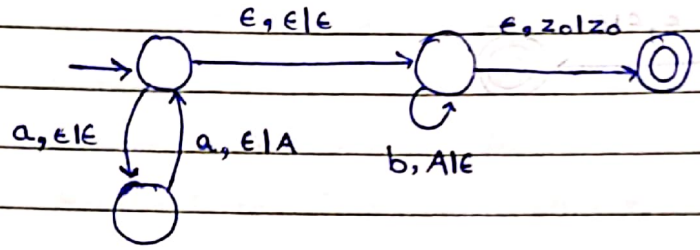
Ex. $L = \{a^n b^m \mid m \geq n \geq 1\}$



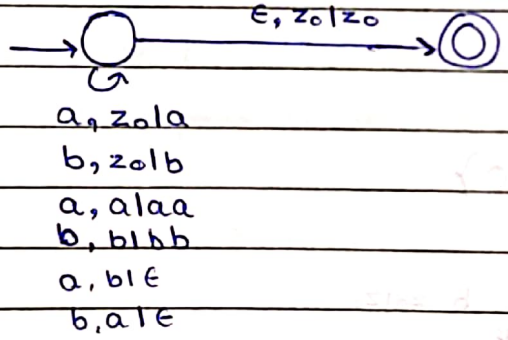
Ex. $L = \{a^n b^{2n} \mid n \geq 0\}$



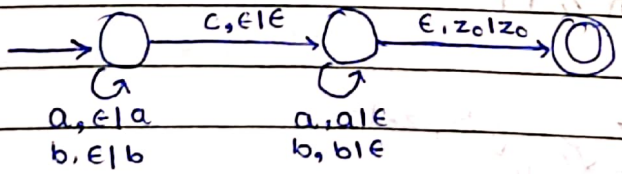
Ex. $L = \{a^{2n} b^n \mid n > 0\}$



Ex. $L = \{w \mid \#a == \#b\}$



Ex. $L = \{wcw^R \mid w \in \{a,b\}^*\}$ $\Sigma = \{a,b,c\}$

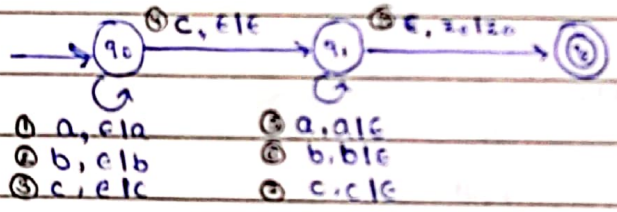


Ex. $L = \{wcw^R \mid w \in z^*\}$ $\Sigma = \{a,b,c\}$

We can guarentee that middle is c. before & after may/maynot be c.

Let's say $w = abac...$

PDA scans ~~from left to right~~ one symbol at a time. So when c comes, should it push it on stack & assume it to be part of w & later check w^R, or should this c be considered middle? PDA will make Assumption 1 & try to accept, if it fails, it will backtrack, then make Assumption 2 & try to accept. If after all the assumption no path leads to acceptance then it rejects. If some assumption leads to acceptance then PDA just accepts.



Let's say $w = abcccba$

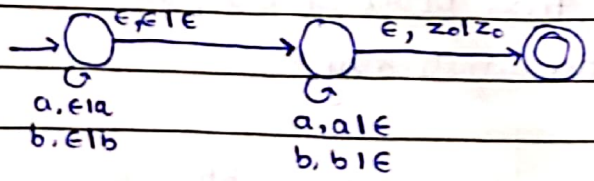
Path1: $q_0 \xrightarrow{1} q_0 \xrightarrow{2} q_0 \xrightarrow{4} q_1 \longrightarrow x$

P2: $q_0 \xrightarrow{1} q_0 \xrightarrow{2} q_0 \xrightarrow{3} q_0 \xrightarrow{3} q_0 \xrightarrow{4} q_1 \longrightarrow x$

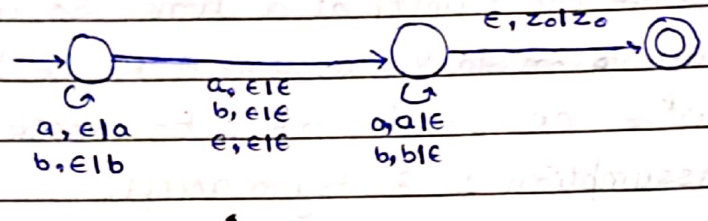
P3: $q_0 \xrightarrow{1} q_0 \xrightarrow{2} q_0 \xrightarrow{3} q_0 \xrightarrow{4} q_1 \xrightarrow{7} q_1 \xrightarrow{6} q_1 \xrightarrow{5} q_1 \xrightarrow{8} q_2$ ✓

∴ Some path accepts w. ∴ $w \in L(PDA)$.

Ex. L = all even len palindrome. $\Sigma = \{a, b\}$



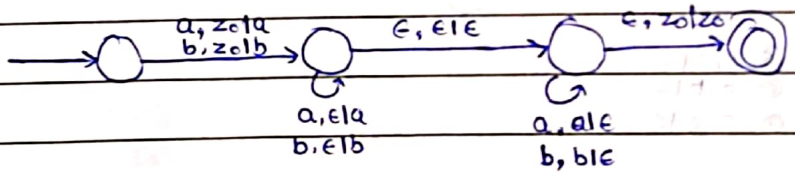
Ex. L = All palindromes. $\Sigma = \{a, b\}$



In the middle there can be a, b or nothing (even ϵ)

Ex. $L = \{ww^R \mid w \in \{a,b\}^+\}$

$\lambda \notin L$.



- Note: when for some variable, in TOC, say, n, no range is mentioned, assume it to be ≥ 0.

Ex. $L = \{a^n b^m \mid n \leq m \leq 2n\}$

$L = \{\epsilon, ab, abb, aabb, aabbb, aabbbb, \ldots\}$

~~The idea is to have two PDAs in one, and jump to both non-deterministically.~~

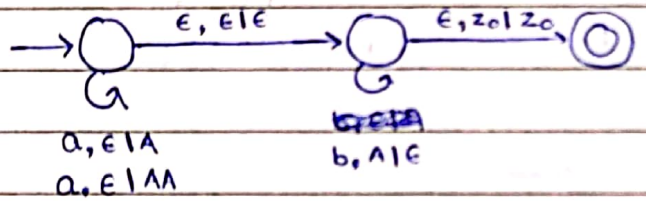
The idea is to push either one A for each a on input tape or two. For some seq. of pushes b's in w should be equal to #As on stack.

Say w = aa bbb

① a a
↓ ↓
1+1 = 2 ≠ #bs

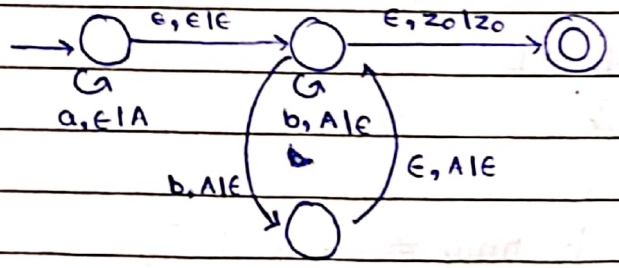
② a a
↓ ↓
2+2 = 4 ≠ #b's

③ a a
↓ ↓
1+2 = 3 == #b's ∴ w is accepted.

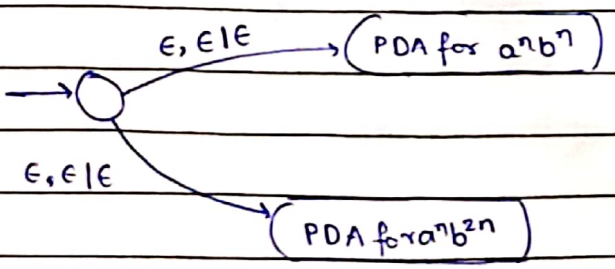


Ex. $L = \{a^n b^m \mid m \leq n \leq 2m\}$

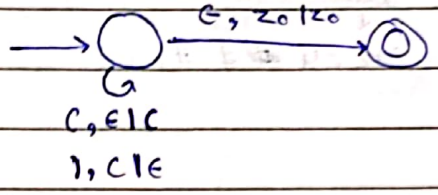
Idea is to add one A per a. Pop either 1 or 2 A's for each B.



Ex. $L = \{a^n b^n\} \cup \{a^n b^{2n}\}$ {∵ range of n not given, $n \geq 0$}

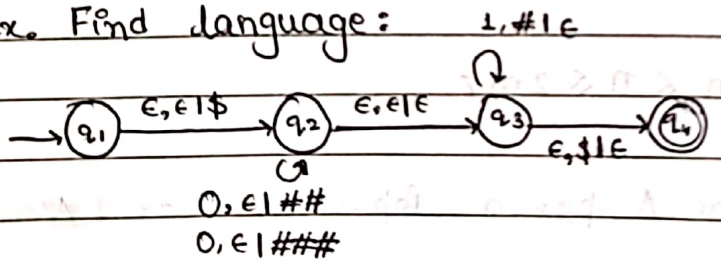


Ex. L = All strings of balanced parenthesis. $\Sigma = \{(,)\}$



- Note: There isn't one universal def^n of PDA, but authors use different def^n with slight variations. We have used the def^n with least restriction, so it covers all types of transitions.

Ex. Find language:

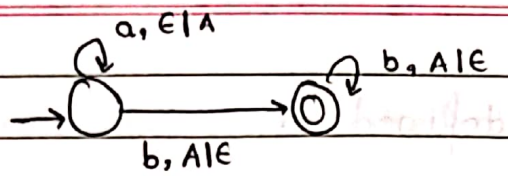


We have designed such PDA.
$\{0^n 1^m \mid 2n \leq m \leq 3n\}$

Non-determinism while pushing # for each 0, either 2# or 3#. Some combination sum should match the no. of 1s in string.

classmate
Date
Page

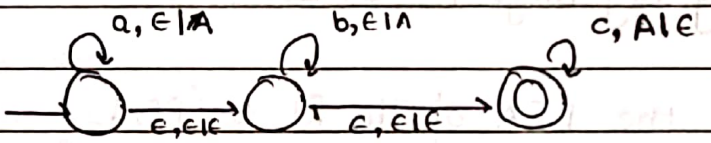
Ex.



The lang. doesn't contain ε.
There must be at least one a. so that when first b comes A is popped.

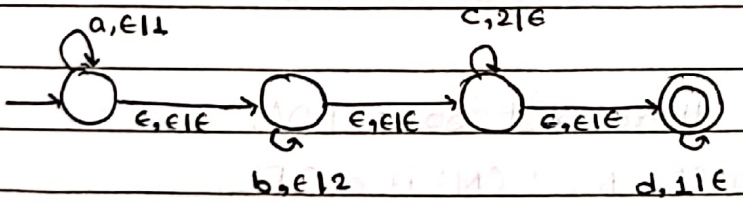
$\{ a^n b^m \mid n \geq m \geq 1 \}$.

Ex.



$L = \{ a^n b^m c^l \mid n+m \geq l \geq 0 \}$

Ex.



$L = \{ a^w b^x c^y d^z \mid w \geq z, x \geq y \}$ ✗

(Doesn't work for $w = abbcd$) So wrong.

$\therefore L = \{ a^w b^x c^y d^z \mid w \geq z, x \geq y$, AND if $z \geq 1$, $x = y \}$

- Formal Def^n: PDA is a 7-tuple. (a.k.a. NPDA)

$(Q, \Sigma, q_0, F, \delta, z_0, \Gamma)$

Q: Non-empty finite set of states
Σ: Input Alphabet
q_0: Initial state
F: Set of finite states
z_0: Stack bottom marker
Γ: Stack alphabet

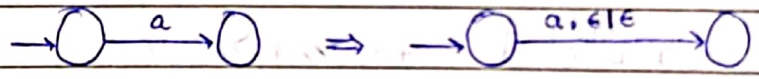
δ: Transition function defined as

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times (\Gamma \cup \{\epsilon\}) \longrightarrow 2^{Q \times (\Gamma \cup \{\epsilon\})^*}$$

where $2^{Q \times (\Gamma \cup \{\epsilon\})^*}$ is power set
& $(\Gamma \cup \{\epsilon\})^*$ is set of all strings over stack alphabet.

- Prove that every Reg. Lang. is CFL.

Proof: Convert the DFA of the RL to PDA.



This represents that we are just not using the stacks.

- For every CFL there exist some PDA.
The language of a PDA M, L(M) is a CFL.

- Acceptance of string

We have already seen the main def^n of string by PDA:

- "At the end of string, the PDA should be able to go to some final state." (stack content doesn't matter).
- There's one other def^n (there's no final state):
"At the end of string, the stack must be empty"
- There can be other def^{n}s. and all accept equivalently for any language.