1. The central Concepts of Automata Theory.

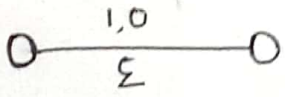
- These concepts include the alphabet, strings and language.

Alphabet = a finite, nonempty set of symbols. Σ

1. $\Sigma = \{0, 1\}$; binary alphabet
2. $\Sigma = \{a, b, .., z\}$; the set of all lower-case letters.
3. the set of all ASCII characters, or
   the set of all printable ASCII characters.

Strings = A list of symbols from an alphabet.

example: 01101 is a string from the binary alphabet, $\Sigma = \{0, 1\}$

Empty string = 

The empty string is the string with zero occurrences of symbols. ϵ is empty symbol.

power of alphabet = Σ^k ; k = strings length

example: if $\Sigma = \{0, 1\}$.

then, $\Sigma^1 = \{0, 1\}$:

$\Sigma^2 = \{00, 01, 10, 11\}$.

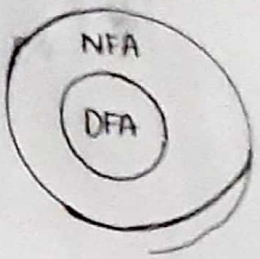
Language = is a set of strings, all of which choose their symbols from some one alphabet.

- Empty string = $|\epsilon| = 0$; $\delta(off, \epsilon) = \{off\}$.
- output call-string.

finite Automata

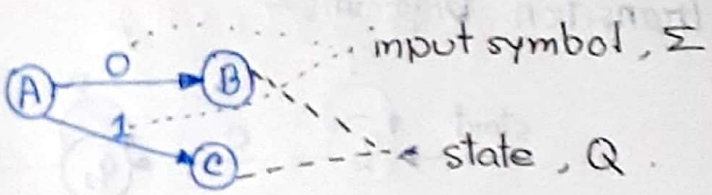
07.23.2022

DFA = Deterministic finite Automata
NFA = Non-Deterministic finite Automata

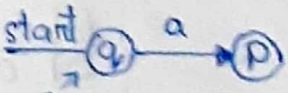


Definition of a DFA —

1. a finite set of states, often denoted, Q.
2. input symbol, Σ



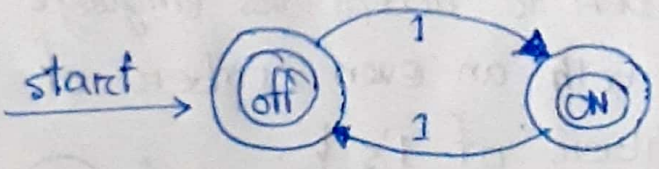
3. transition function δ that ~~taken~~ as arguments a state and an input symbol and a returns a state.
   - δ (state, input symbol) = {return a state} ; = δ(q, a) = {p}
   - fun^c factorial (5) ... argument



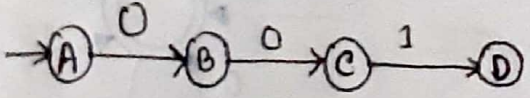

```
{
return }
```

= 5 × 4 × 3 × 2 × 1 = 120.

4. start state, q
5. final state or accepting state ২টের অধিক হতে পারে।



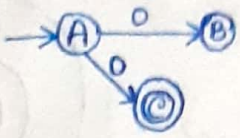
→ its accept infinite strings = ϵ
= 1
= 11
= 111
= 1111
= .....

ex: 

string 001 ✓accepted

string 000001 · doesn't accept.

07.29.2022



$= \delta(A, 0) = \{B, C\}$

** যদি কোনো transition func যদি more than one state return করে তবে তা DFA থাকবে না, NFA হয়ে যাবে।

Transition Diagrams -

The transition diagrams for the DFA accepting all strings with a substring 01.

Transition Table - is a conventional, tabular representation of a function like δ that taken two arguments and returns a value.

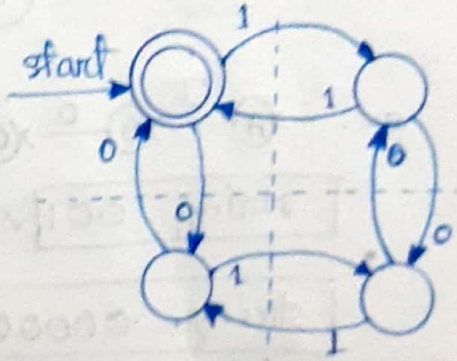
	0	1
$\rightarrow q_0$	q_2	q_0
$*q_1$	q_1	q_1
q_2	q_2	q_1

transition table for the DFA of example.

Example 2.4:

Let us design a DFA to accept the language

$L = \{w \mid w$ has both an even number of 0's and an even number of 1's$\}$.



q_0 = 0,1 - even

q_1 = 0-even -1-odd

q_2 = 0-odd - 1-even

q_3 = 0,1 - odd

= 00011001101111

07.31.2022

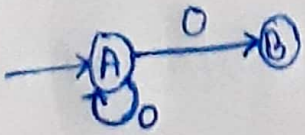
NFA = Non-deterministic finite Automata.

it is easy to construct an NFA than DFA for a given regular language. The finite Automata are called NFA when there exist many paths for specific input from the current state to the next state. Every

- Every NFA is not DFA, but each NFA can be translated into DFA.

NFA represented by $A = (Q, \Sigma, \delta, q_0, F)$

1. Q is a finite set of states.
2. Σ is a finite set of - input symbols. $\Sigma = \{a, b, \ldots z\}$
3. q_0 = member of Q, is the start state
4. F = a subset of Q, is the set of final states.
5. δ = transition function is a function that taken a state in Q and an input symbol in Σ as arguments and returns a subset of Q.

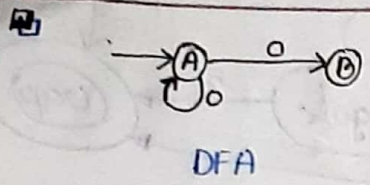


: it's NFA, ~~DFA~~ DFA

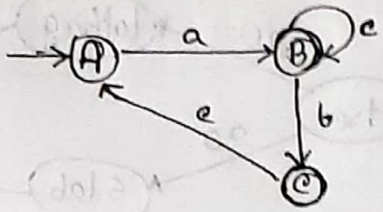
$\delta(A, 0) = \{A, B\}$ - 0 input থাকায় পরে A, B দুটোই return করবে.

= DFA তে একটা state return করে.

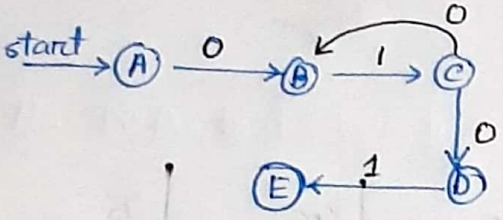
= But NFA তে set return করে একাধিক.



DFA

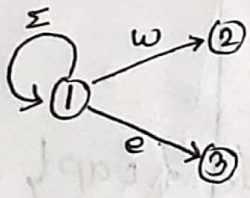


NOT DFA



$\delta(C, 0) = \{B, D\}$; C থেকে B তেও যায়, D তেও যায়।

তাই এটা DFA হবে না, একাধিক return করে তাই।



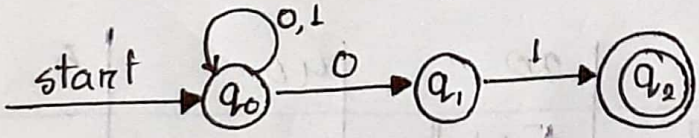
; NFA, NOT DFA

২টা Automata কে equivalent বলে যখন

২টা Automata same strings accept করে

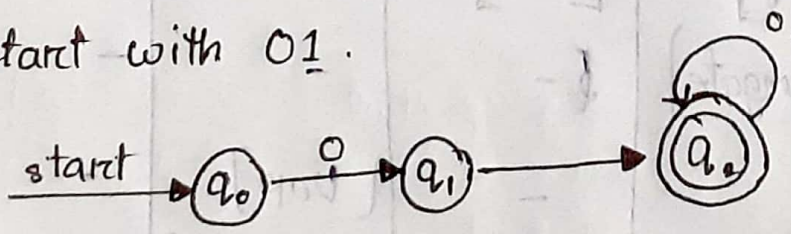
এবং reject করে।

2.9 : An NFA accepting all strings that end in 01

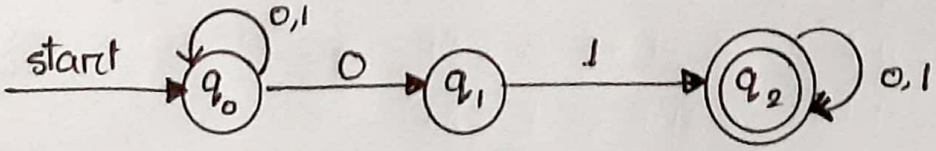


• string = ϵ
01
10
101
110
111
1000
1111

① start with 01.



② has a 'substring' 01

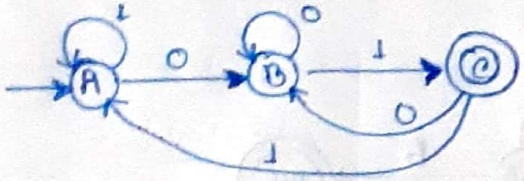


Transition table

07.04.2022

	0	1
$\{q_0\}$ A	$\{q_0, q_1\}$ B	$\{q_0\}$ A
$\{q_0, q_1\}$ B	$\{q_0, q_1\}$ B	$\{q_0, q_2\}$ C
* $\{q_0, q_2\}$ * C	$\{q_0, q_1\}$ B	$\{q_0\}$ A

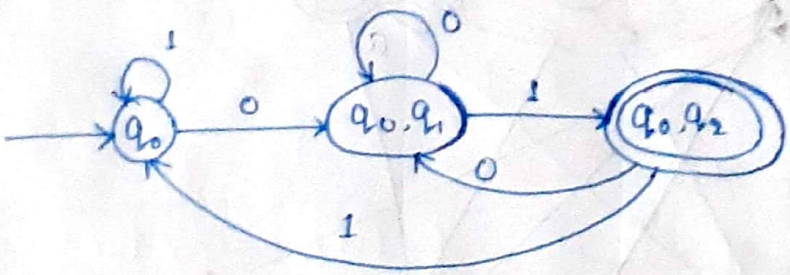
$\delta(\{q_0, q_1\}, 0)$
$= \delta(q_0, 0) \cup \delta(q_1, 0)$
$= \{q_0, q_1\} \cup \{\}$
$= \{q_0, q_1\}$



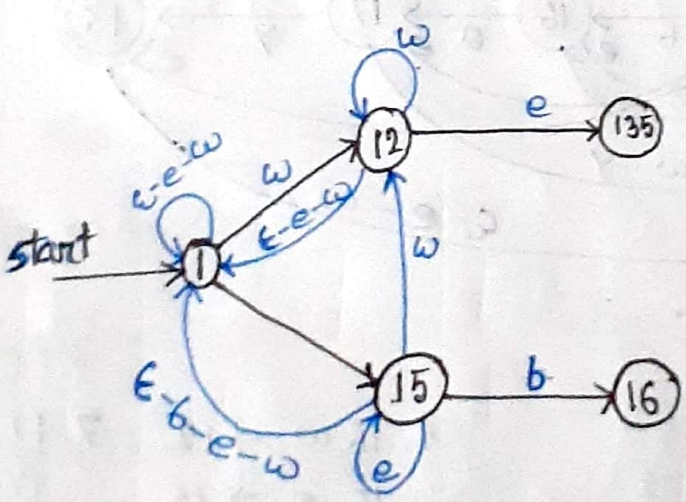
DFA

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

Let, $\{q_0\} = A$
$\{q_0, q_1\} = B$
$\{q_0, q_2\} = C$



#

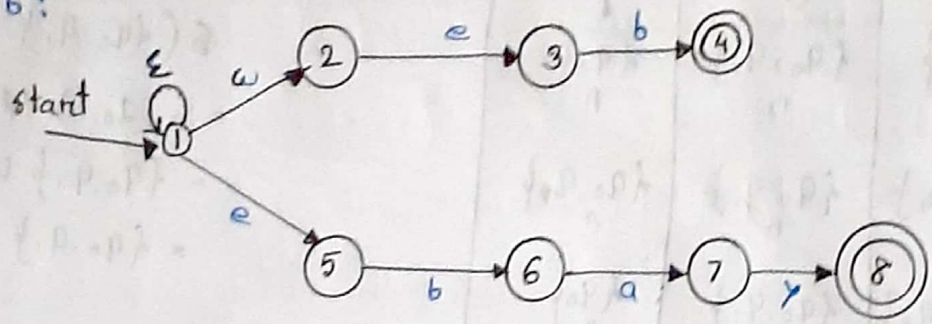


12, e → 135
12, ω → 12
12, ω-e-ω → 1

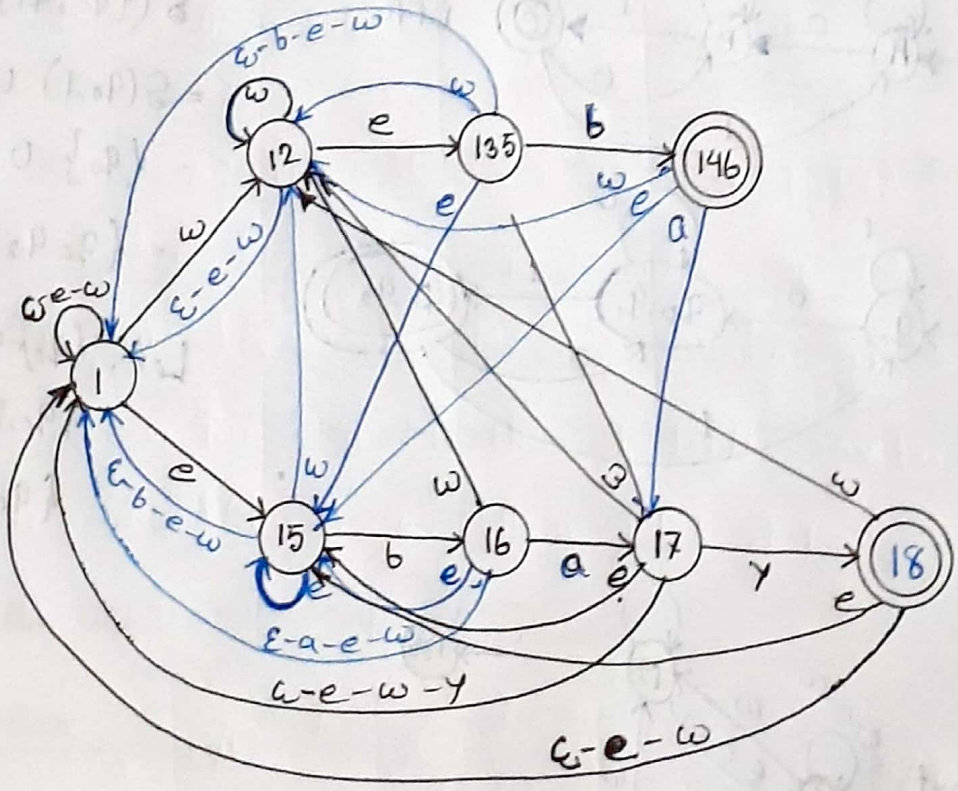
15, ω → 12
15, e → 15
15, b → 16
15, ω-b-e-ω → 1

08.07.2022

2.16:



An NFA that searches for the words web and ebay.



1. 1, ω → 12
   1, e → 15
   1, ε-e-ω → 1

2. 12, ω → 12
   12, e → 135
   12, ε-e-ω → 1

3. 15, ω → 12
   15, e → 15
   15, b → 16
   15, ε-b-e-ω → 1

4. 135, ω → 12
   135, e → 15
   135, b → 146
   135, ε-b-e-ω → 1

5. 16, ω → 12
   16, e → 15
   16, a → 17
   16, ε-a-e-ω → 1

6. 146, a → 17
   146, e → 15
   146, ω → 12
   146, ε-a-e-ω → 1

7. 17, ω → 12
   17, e → 15
   17, y → 18
   17, ε-e-ω-y → 1

8. 18, e → 15
   18, ω → 12
   18, ε-e-ω → 1

ϵ- NFA 08 11 2022

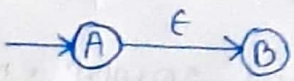
NFA allows ϵ-transition which call ϵ-NFA.

NFA এর একটি বিশেষ প্রকার যেখানে শুধু ϵ-NFA যা DFA এর মধ্যে।

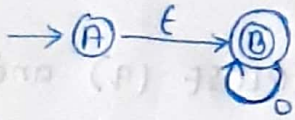
ϵ-NFA থেকে DFA transition বানানো সহজ।

□ কোনো নির্দিষ্ট input signal ছাড়াই A state থেকে B state -এ যেতে পারবে। Like –

** কিন্তু উল্টো বানানো যাবে না.



$S(A, \epsilon) = A$

2. 

$S(A, \epsilon) = \{A, B\}$.

** এই একাধিক return করা যাবে, NFA এর বিশেষত্ব) তাই ϵ-NFA ও altimately NFA.

Use of ϵ- transition: we should begin with an informal treatment of ϵ-NFA's, using transition diagrams with ϵ allowed as a label. In the examples, to follow, think of the automation, as accepting those sequences of labels along paths from the start state to an accepting state. However, each ϵ along a path is 'invisible', it contributes nothing to the string along the path.

ϵ-NFA that accepts decimal numbers consisting of:

1. An optional + or − sign. ((+5, −55))
2. A string of digits. (5)

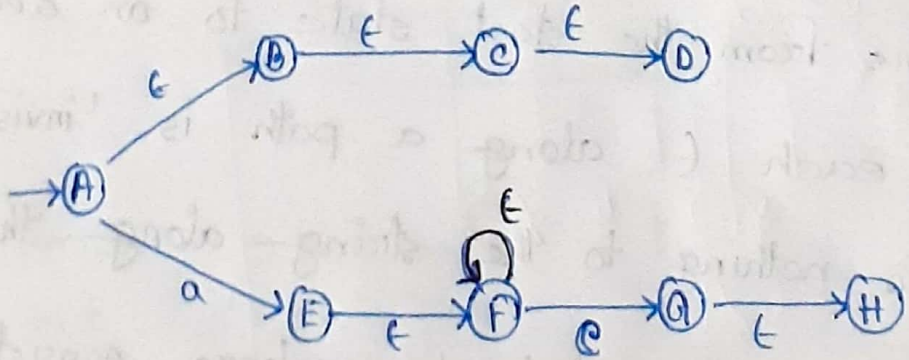
3. A decimal point, and (01011 ε = 01011)

4. Another string of digits, either this string of digits or the string (2) can be empty (মান ε), but at least one of the two strings of digits must be NONempty.

Ⓐ —— Ⓑ —— Ⓒ

** দুটো একসাথে empty হতে না।
যেকোনো একটা empty হতে।

ε-closures: if state p is in ECLOSE (q) and there is a transition from state p to state r labeled ε, the r is in ECLOSE (q), More precisely, if S is the transition function of the ε-NFA involved, and p is in ECLOSE (q), then ECLOSE (q) also contains all the state in S(P, ε).



concatenated
↓
aεε = aε
a·ε = εa = a.

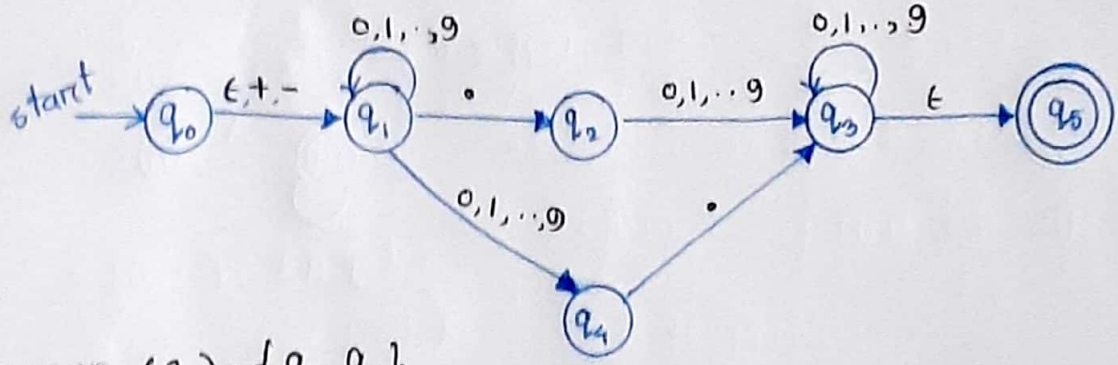
ε - প্রতিটিতে একটা লুপানো ε থাকে।
তাই প্রত্যেক node- নিজের মধ্যে একটা যাবে।

Ⓐ → Ⓑ ECLOSE (A) = {A, B} হবে তাই যাবে।

ECLOSE (A) = {A, B, C, D}
ECLOSE (B) = {B, C, D}
ECLOSE (C) = {C, D}
ECLOSE (D) = {D}

ECLOSE (E) = {E, F} → E থেকে E, F এই যায়, অন্য কোথাও যাওয়া যাচ্ছে না তাই F থেকে G এ যাওয়ার path ε নয়, তাই।
ECLOSE (F) = {F}
ECLOSE (G) = {G, H}
ECLOSE (H) = {H}.

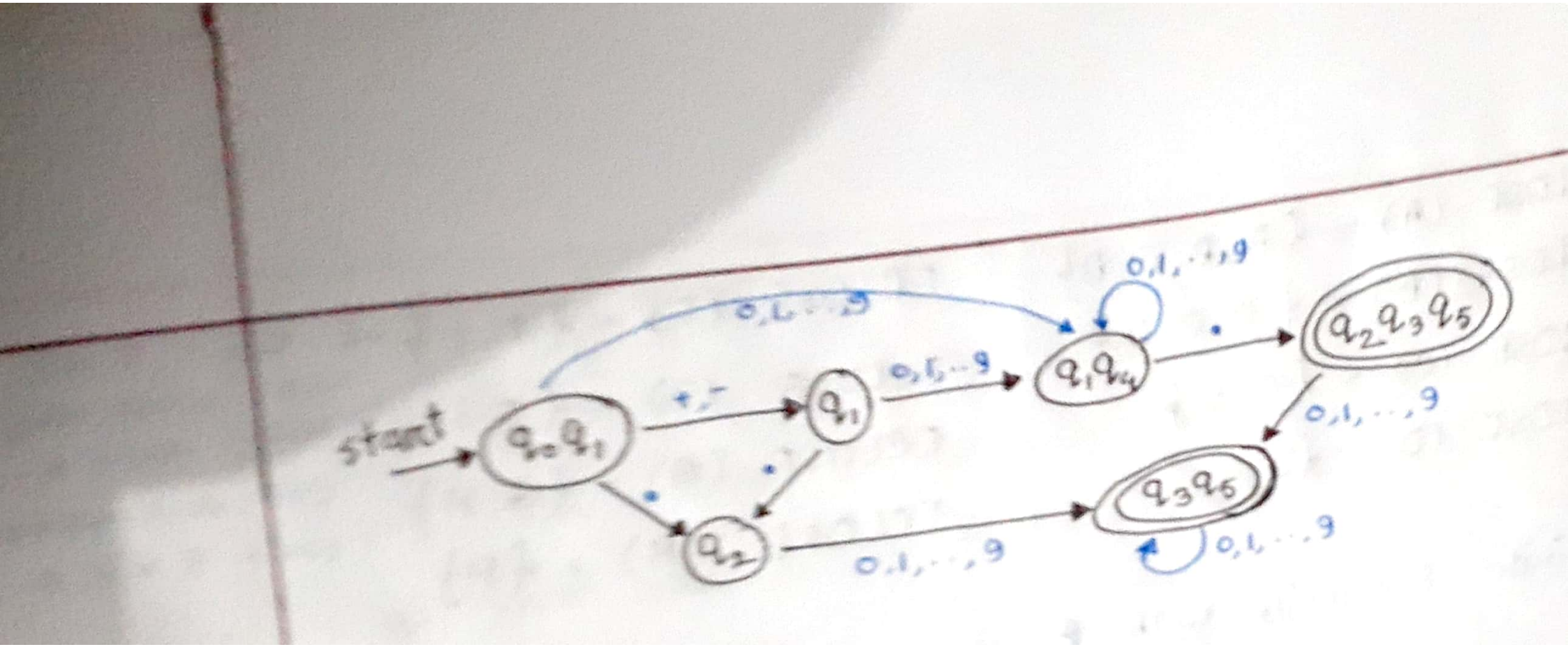
- Finite Automata with Epsilon - transition -



ECLOSE (q_0) = {q_0, q_1}.

	0, 1, .., 9	+, -	.
→ {q_0, q_1}	ECLOSE ({q_1, q_4}) = {q_1, q_4}.	ECLOSE ({q_1}) = {q_1}	ECLOSE ({q_2}) = {q_2}
{q_1}	ECLOSE ({q_1, q_4}) = {q_1, q_4}	-	ECLOSE ({q_2}) = {q_2}
{q_2}	ECLOSE ({q_3}) = {q_3, q_5}	-	-
{q_1, q_4}	ECLOSE (q_1, q_4) = {q_1, q_4}	-	ECLOSE (q_2, q_3) = {q_2, q_3, q_5}.
* {q_3, q_5}	ECLOSE (q_3, q_5) = {q_3, q_5}	-	-
* {q_2, q_3, q_5}	ECLOSE (q_3, q_3, q_5) = {q_3, q_5}	-	-

q_3 এ যে অবস্থায় আছে সে q_5 এ যায়, যার অন্য কোনো (+, -, 0, 1, .., 9, .) নিচ্ছে আছে।

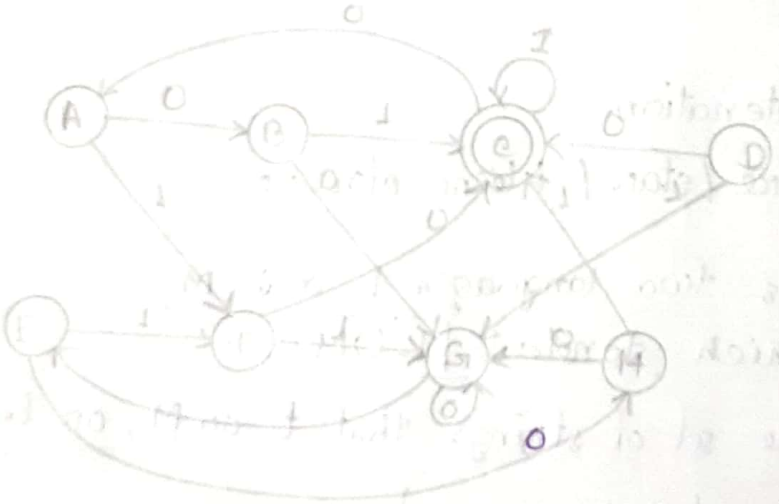


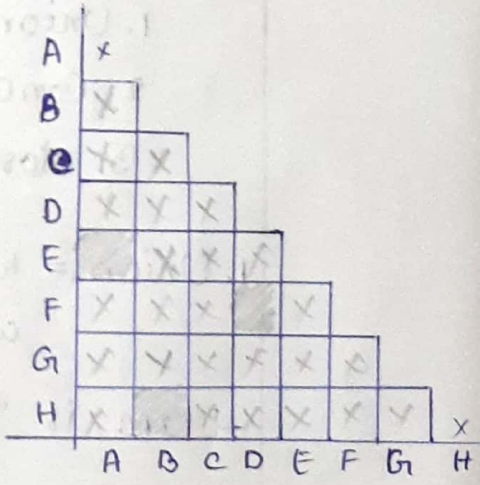
Chapter 04 = Properties of Regular languages

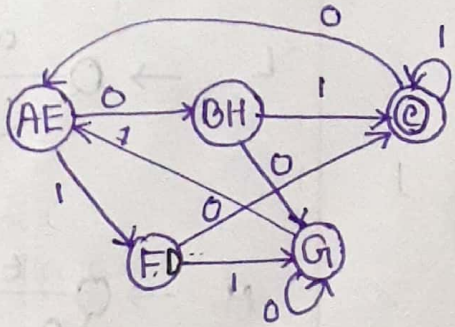
topic = Equivalence and Minimization of Automata :

here, we can take any DFA and find an equivalent DFA that has the minimum number of states. In fact, this DFA is essentially unique. given any two minimum state DFAs that are equivalent, we can always find a way to rename the states so that the two DFA's become the same.

ex:







* এখানে D থেকে কিছুই 100 হচ্ছে না, কিন্তু FD equivant F = D + F এর জন্য D নিচ্ছে।

Chapter 3: Regular Expressions and Languages –

= we switch our attention from machine-like descriptions of languages DFA and NFA - to an algebraic description called "regular expression".

- it's denote languages
- finite automata equivalent / (≡) Regular Expression.

= has three (3) operations on languages –

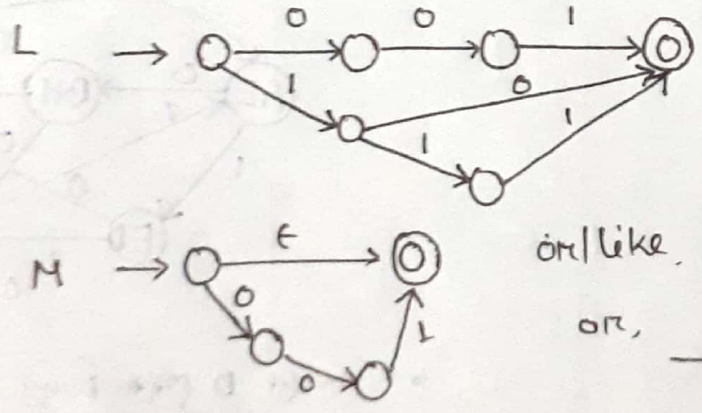
1. Union
2. Concatenation
3. closure / star / kleene closure.

1. Union = has two languages L and M, which denoted **L ∪ M**

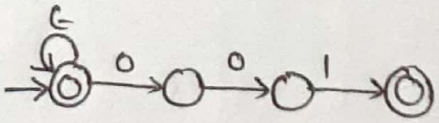
-L ∪ M, is the set of strings that L or M, or both.

example = $L = \{001, 10, 111\}$

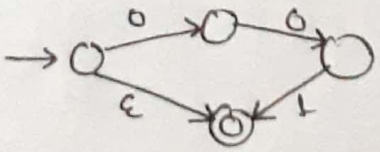
$M = \{\epsilon, 001\}$



or/like



or,



∴ $L \cup M = \{001, 10, 111, \epsilon, 001\}$.

∴ $L \cup M = M \cup L$.

02. Concatenation – declare [.]

- denote concatenation of languages either with a dot or with no operator at all.

like $L.M = LM$ but not $LM \neq ML$

$L = \{001, 10, 111\}$ $M = \{\epsilon, 001\}$

$LM = \{001, 10, 111, 001001, 10001, 111001\}$

03. Closure – denote L^* and represents the set of those strings that can be formed by taking any number of strings from L. $L = \{0, 11\}$ $L^0 = \{\epsilon\}$

$L^1 = \{0, 11\}$

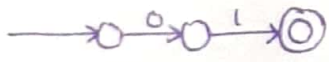
$L^2 = \{00, 011, 110, 1111\}$

not = $\{0\underline{10}\underline{11}$ or $,\underline{10}\underline{1}\}$.

□ More formally, L^* is the infinite union, $\bigcup_{i \geq 0} L^i$

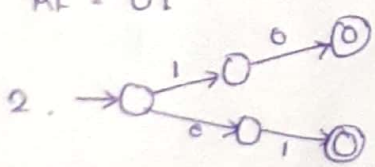
where, $L^0, L^1, L^2 \ldots L^i = LLL \ldots L$.

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

1. 

string 01

RE = 01

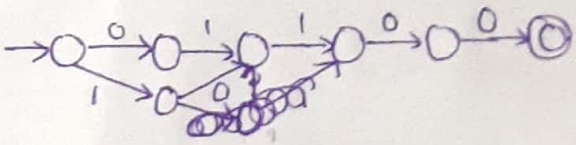
2. 

string = 01, 10

RE = (01+10)

বিকল্প

3. $L = \{01100, 10100\}$

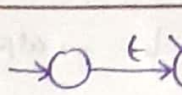


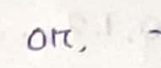
RE = (01+10)100

RE = a^*

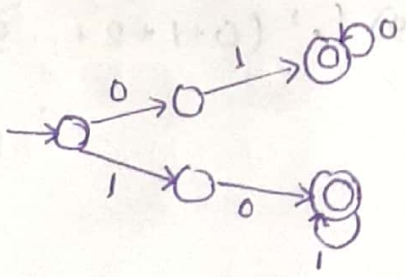
= L = {a}

strings = ϵ, a, aa, aaa, ...



or, 

2.

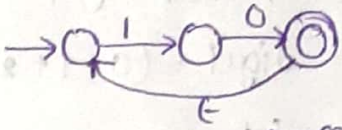


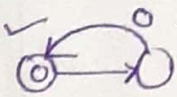
RE = $01 0^* + 10^*$

01 0 00 ... 10 11 ...

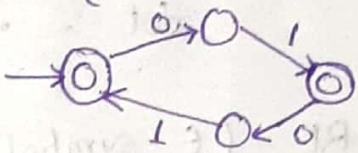
3. RE = $(10)^*$

strings = ϵ, 10, 1010, 101010, ...



or 

or,



4. RF = $01(001)^*$

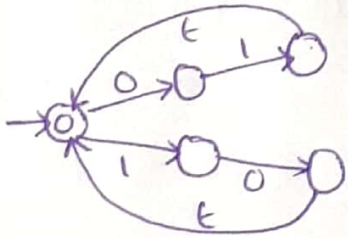
strings : 01

01001

01001001..

5. RE = $(01 + 10)^*$ strings =

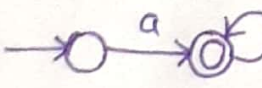
01ϵ01ϵ + 10ϵ10ϵ



↑

$a^+ = a.(a^*)$

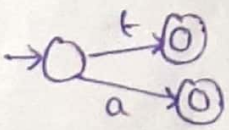
$= a\,(\epsilon, a, aa, aaa, \ldots)$



$a? = \epsilon + a$

→ হয় ϵ নয়ে final হয়ে যাও

নয় a নয়ে final হয়ে যাও



2.18 → ϵ-NFA of fig – just collected not in chapter

$\Sigma = \{+, -, 0, 1, 2, \ldots 9, \cdot\}$ → alpabet

RE = $(\epsilon + '+' + '-')\ (0+1+2+\ldots 9)^*\ ('.' + (0+1+2+\ldots 9)(0+1+2+\ldots 9)^*$

union, symbol

$\epsilon + (0+1+2+9)\ '.'\ (0+1+2+\ldots 9)^*\ \epsilon)$

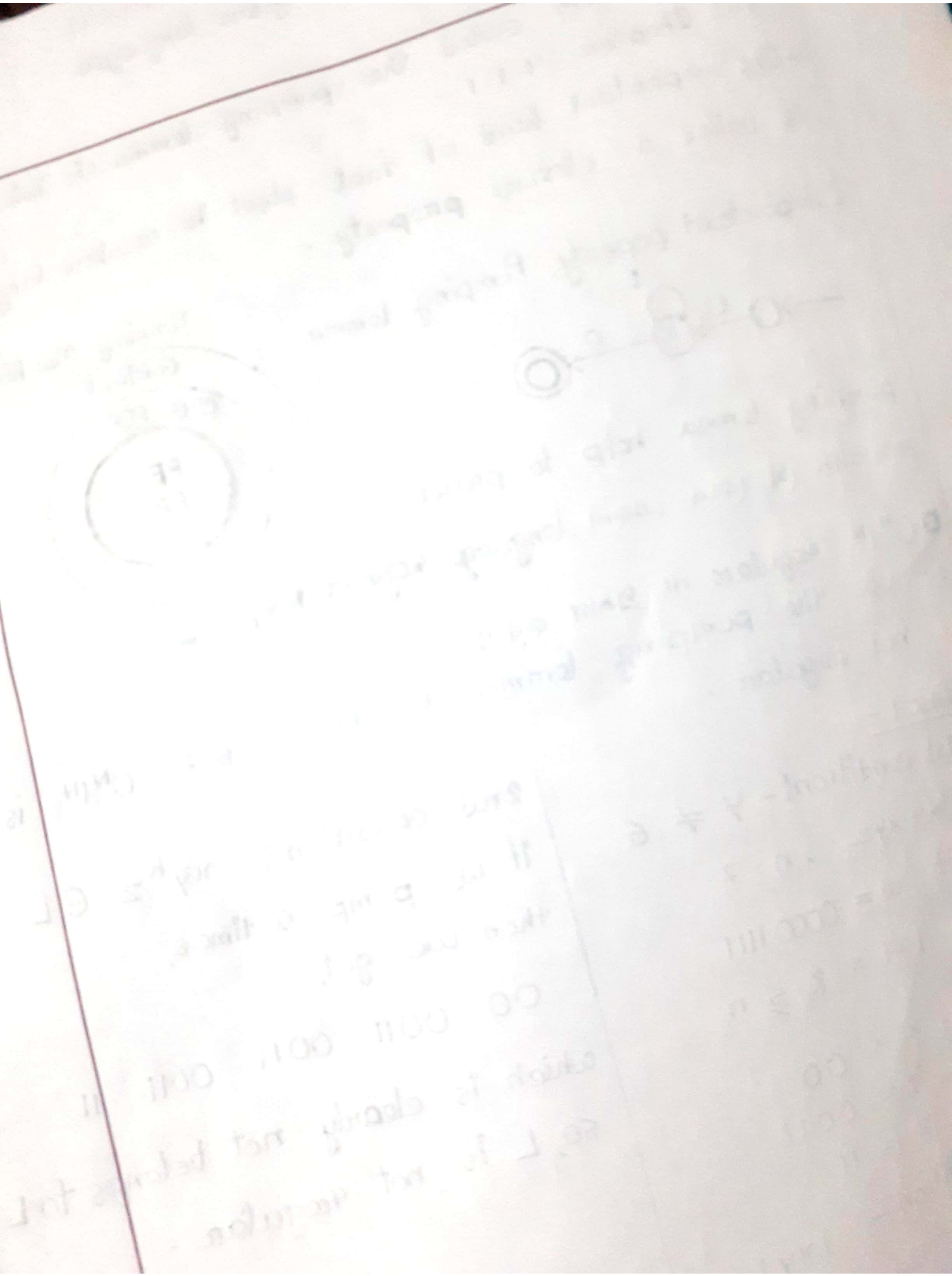
here, symbol = $('+' + '-')$

digit = $(0+1+2+\ldots 9)$

dot = $\cdot$

RE = $(\epsilon + \text{symbol})\ (\text{digit})^*\ (\text{dot} + \text{digit}.\ \text{digit}^*\ \epsilon + \text{digit dot digit}^*\epsilon)$

$= (\epsilon + \text{symbol})\ \{\text{digit}^*\ (\text{dot} + \text{digit}^+\epsilon + \text{digit}^+\ \text{dot}\epsilon)$

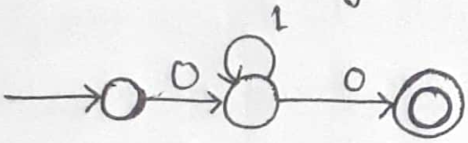


The pumping lemma for Regular languages

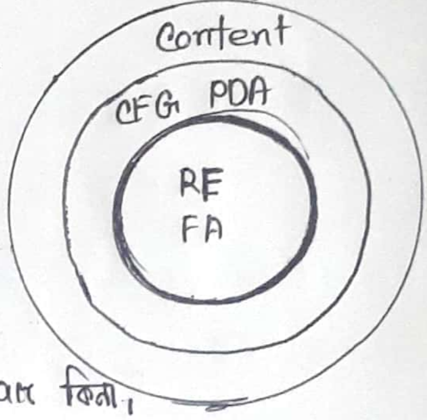
= This theorem, called the pumping lemma is introduced in Chapter 4.1.1

- One important kind of fact about the regular languages is called a closure **property**.

= important poperty Pumping lemma



Turning Machine.



pumping lemna help to prove
করা যায় যে কোন একটা language regular কিনা।

- $0^N 1^N$ regular না প্রমাণ কর?

or. Use the pumping lemma to show that $0^N 1^N$ is not regular.

⇒ Case 1 -

1st condition:- $y \neq \epsilon$

$W = xyz$, $n = 7$.

let, $w = 0000\,1111$

$|w| = 8 \geq n$

let ; $x = 00$

$y = 0011$

$z = 11$

2nd condition:- $|xy| = 6$

$|xy| \leq n$

$6 \leq n$

3rd condition: $xy^k z \in L$

if we pump 2 times,

then we get,

00 0011 0011 0011 11

which is clearly not belongs to L

so, L is not regular.

maximum ২টা case দেখালেই enough

n, x, y, z এর মান নিজে করে দিতে পারো.

Case 2 - 1st condition : $y \neq \epsilon$

let, $x = \epsilon$

$y = 00$

$z = 001111$

2nd condition : $|xy| \leq n$

$2 \leq 7$

3rd condition : $xy^kz \in L$

if we pump 3 times

then we get,

ϵ 00 00 00 00 001111,

which does not belong to L

so, L does not satisfy pumping lemma. L is not regular.

Scanned with CamScanner

5.1.2 ⇒ CFG - Context free Grammar

- has two type.
  1. terminal Variable (no production)
  2. Non-terminal variable (production)

□ Variable সব সময় বড় হাতের অক্ষরের থাকবে,

□ $0^n 1^n$ এর combination থাকলে হবে।

□ string যা finite A/RL দিয়ে solve করতে পারতে ছিলাম না same problem CFG দিয়ে solve হবে।

□ right side - এ সব variable, digit অথবা any combination থাকতেও পারে but left side এ শুধু variable থাকবে।

like = left → right

$P \rightarrow \epsilon$

$P \rightarrow 0$

$P \rightarrow 1$

$P \rightarrow 0P0$

$P \rightarrow 1P1$

এটাকে একসাথে ও লেখা যায় —

$P \rightarrow \epsilon / 0 / 1 / 0P0 / 1P1$

variable → terminal.

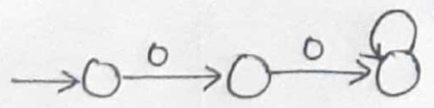
□ $\Sigma = \{\_, \_, \_\}$ symbol গুলো হচ্ছে terminal

□ right side terminal, non-terminal combination থাকতে পারে.

□ left side always non-terminal তথা মানে variable n n.

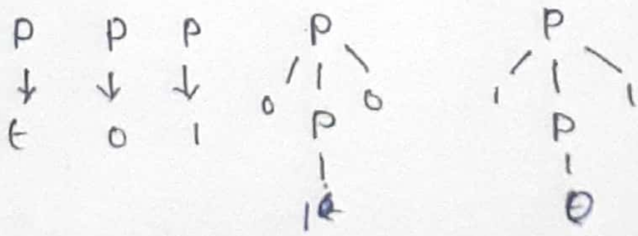
□ Production বা grammar এর output গুলো ও string হবে।

CFG, $G = \{V, T, S, P\}$.

□ 

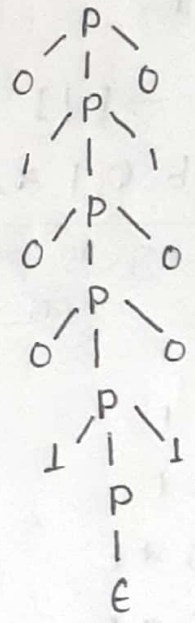
string = 001
= 001
= 0011
= 00111

□ $P \rightarrow \epsilon / 0 / 1 / 0P0 / 1P1 /$



Palindromes

⇒ 0100110010



ex = 01101000010110

Grammer

= ϵ , 0, 1, 0P0, 1P1 → terminal 3 variable এর combination থাকবে। symbol গুলোকে grammer অনুযায়ী terminal গুলোর production থাকবে না।

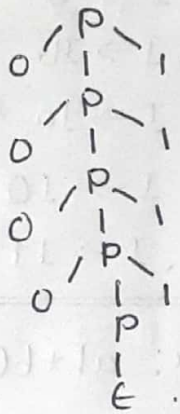
0^n1^n এর grammar design -

= n = 4 ; 00001111

P → ϵ

P → 0P1

P → 00P11



ex=

derivation এর যে 3type ও —

1. Left most derivation - lm
2. Right most derivation - rm
3. Parse tree derivation

= a context free grammar for simple expressions

1. $E \to I$
2. $E \to E+E$
3. $E \to E*E$
4. $E \to (E)$
5. $I \to a$
6. $I \to b$
7. $I \to Ib$
8. $I \to Ia$
9. $I \to I0$
10. $I \to I1$

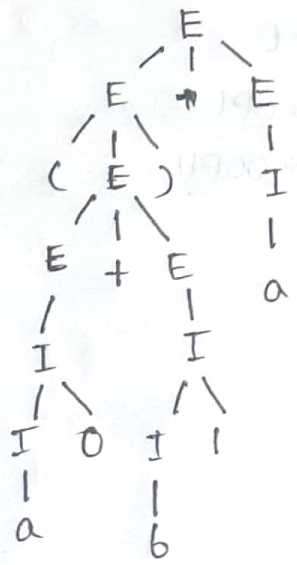
or, $E = I \mid E+E \mid E*E \mid (E)$

$I = a \mid b \mid Ia \mid Ib \mid I0 \mid I1$

- non-terminal / variable $\to E, I$
- terminal / symbol $\to a, b, 0, 1, *, +, (,)$

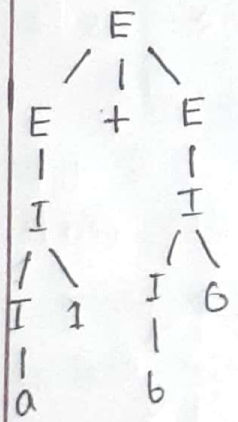
→ syntax protection নেই।

ex: $(a0+b1)*a$



$E \xrightarrow{lm} E*E$
$\to (E)*E$
$\to (E+E)*E$
$\to (I+E)*E$
$\to (I0+E)*E$
$\to (a0+E)*E$
$\to (a0+I)*E$
$\to (a0+I1)*E$
$\to (a0+b1)*E$
$\to (a0+b1)*I$
$\to (a0+b1)*a$

ex: $a1+b0$



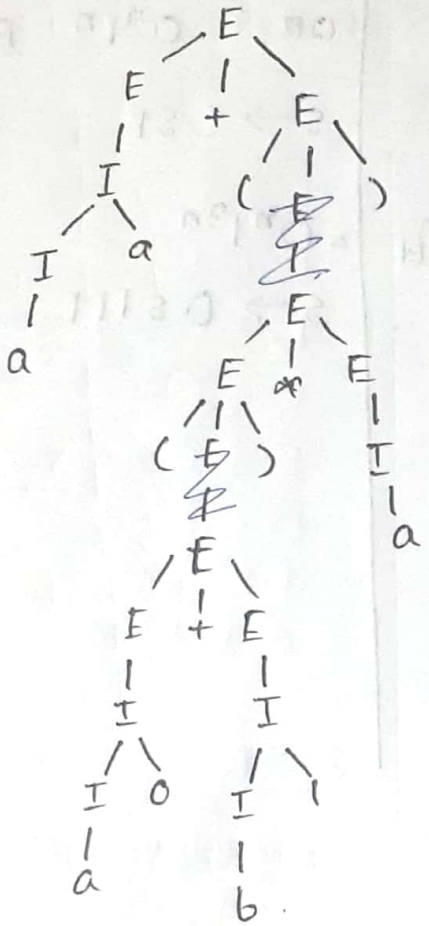
$E \xrightarrow{lm} E+E$
$\to I+E$
$\to I1+E$
$\to a1+E$
$\to a1+I$
$\to a1+I0$
$\to a1+b0$

$E \xrightarrow{rm} E+E$
$\to E+I$
$\to E+I0$
$\to E+b0$
$\to I+b0$
$\to I1+b0$
$\to a1+b0$

do lm, or rm, both are equal.

□ lm, rm এ parse tree same হলে, তা Ambiguous Grammar.

ex: aa + ((a0+b1) * a)



$E \xrightarrow{Rm} E + E$

$\rightarrow E + (E)$

$\rightarrow E + (E * E)$

$\rightarrow E + (E * I)$

$\rightarrow E + (E * a)$

$\rightarrow E + ((E) * a)$

$\rightarrow E + ((E + E) * a)$

$\rightarrow E + ((E + I) * a)$

$\rightarrow E + ((E + I1) * a)$

$\rightarrow E + ((E + b1) * a)$

$\rightarrow E + ((I + b1) * a)$

$\rightarrow E + ((I0 + b1) * a)$

$\rightarrow E + ((a0 + b1) * a)$

$\rightarrow I + ((a0 + b1) * a)$

$\rightarrow Ia + ((a0 + b1) * a)$

$\rightarrow aa + ((a0 + b1) * a)$

* Use the pumping lemma to show that $\{n^a a^{2a}\}$ is not regular.
a. give lm b. give rm c) give parse tree for the string
$S \rightarrow +SS \mid *SS \mid a$ with string + *aaa.

my name is Asma Sultana

$I = \{a, s\}$ $I = \{0, 1\}$ like

give $n^a\ a^{2a}$.

means, nn 2ta 2(mr aaaa 4ta

$a = 2$, $n = 2$, $a = 2^2 = 4$.

so, $n^a a^{2a} \rightarrow a^a s^{2a}$.

$S \rightarrow aaSSSS$

$S \rightarrow \epsilon$.

aaSssss

aaaSssssssss

or, $= 0^n 1^n$ pattern

$S \rightarrow 0S1$

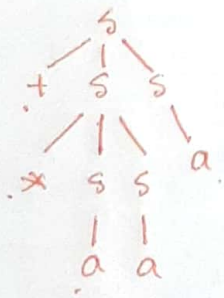
$= 0^n 1^{3n}$

$S \rightarrow 0S111$

Slove = $S \rightarrow +SS \mid *SS \mid a$

string = +*aaa.

3. do string Parse tree



1. $S \xrightarrow{lm} +SS$
$\rightarrow +*SSS$
$\rightarrow +*aSS$
$\rightarrow +*aaS$
$\rightarrow +*aaa$

2. $S \xrightarrow{rm} +SS$
$\rightarrow +Sa$
$\rightarrow +*SSa$
$\rightarrow +*Saa$
$\rightarrow +*aaa$.

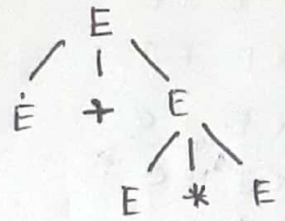
5.4 = Ambiguity in Grammars and Languages.

Ex 5.25 = $E \to E+E \mid E*E$

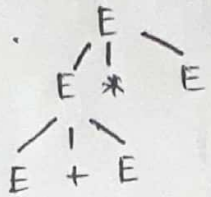
Solve = 1. $E \Rightarrow E+E \Rightarrow E+E*E$

2. $E \Rightarrow E*E \Rightarrow E+E*E$

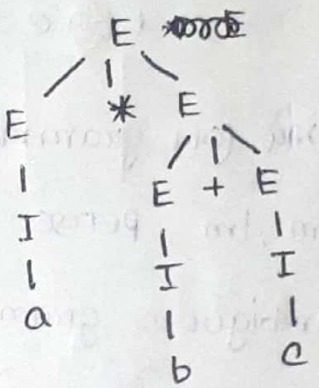
1.



OR . 2.



3. perse tree



Ambiguous Grammar →

$E \to E+E \mid E*E \mid I$

$I \to a \mid b \mid c \mid \ldots \mid z$

OR $I \to 0 \mid 1 \mid 2 \mid 3 \mid \ldots \mid 8$

string = a * b + c

=> 1. left most derivation

$E \xrightarrow{lm} E*E$

$\to I*E$

$\to a*E$

$\to a*E+E$

$\to a*I+E$

$\to a*b+E$

$\to a*b+I$

$\to a*b+c$

2. Right most derivation

$E \xrightarrow{rm} E*E$

$\to E*E+E$

$\to E*E+I$

$\to E*E+c$

$\to E*I+c$

$\to E*b+c$

$\to I*b+c$

$\to a*b+c$

2nd solution - a*b+c → doing + fast.

Left most derivation

$E \xrightarrow{lm} E+E$

$\rightarrow E*E+E$

$\rightarrow I*E+E$

$\rightarrow a*E+E$

$\rightarrow a*I+E$

$\rightarrow a*b+E$

$\rightarrow a*b+I$

$\rightarrow a*b+c$.

Right most derivation

$E \xrightarrow{rm} E+E$

$\rightarrow E*E+E$

$\rightarrow E*E+I$

$\rightarrow E*E+c$

$\rightarrow E*I+c$

$\rightarrow E*b+c$

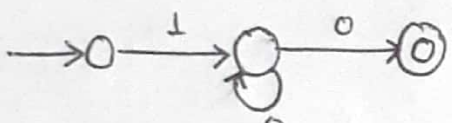
$\rightarrow I*b+c$

$\rightarrow a*b+c$

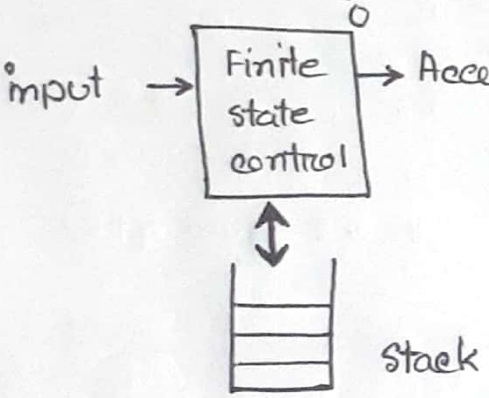
যদি কোন grammar থেকে derive করার সময় ২ বা তার বেশি rm, lm, parse tree পাই তাহলে ঐ grammar কে ambiguous grammar বলে। ag ভালো না। চেষ্টা করা উচিত এটা (ag) দূর করা।

Chapter 6 = Pushdown Automata

it an extension of the nondeterministic ~~future~~ finite automaton with ϵ-transitions, which is one of the ways to define the regular languages.



; it's works Stack. (LIFO).



** এই ক্ষেত্রে POP আগে হয় then PUSH.

আগের নিয়মে @ $\xrightarrow{1}$ (b)

এখন @ $\xrightarrow{1, a \to b}$ (b)

1, a → b
↓ pop ↓ push
string symbol

0 $\xrightarrow{1, \epsilon \to \omega}$ 0

যেহেতু ϵ মানে nothing.
তাই ω push হবে।

empty stack নিয়ে কাজ start করব। এবং final state ও যেখানে stack টা empty হয়ে যাবে।

* 0^n1^n → finite Automata দিয়ে design করতে পারছিলাম না. push down Automata দিয়ে solve করব.

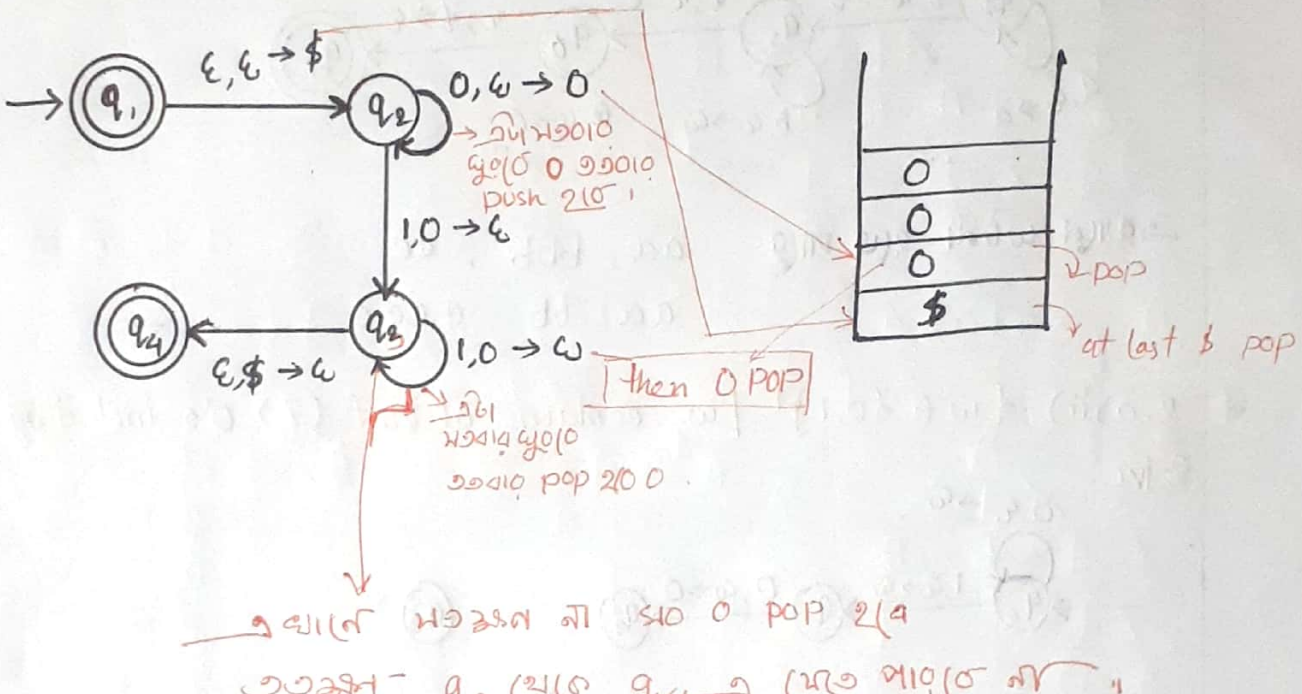
FA ≡ RE

CFG ≡ PDA

state diagram for the PDA M_1, that recognizes $\{0^n 1^n \mid n \geq 0\}$

stack এ first এ $, marker push করে।

then, push-pop করতে করতে final state এ যখন আসবে তখন আবার marker এ আসবে, then $ pop করে empty করে.

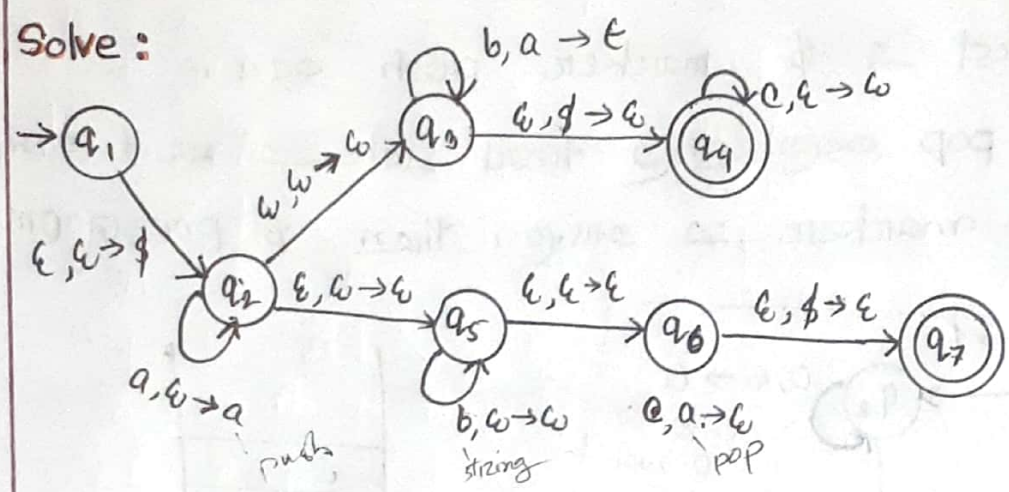


- ε → ε → No pop No push
- ε → a → No pop a push
- a → ε → a pop No push
- a → b → a pop b push.
- Marker $ push 1st
- last Marker pop.

Ex : 2.16 - This example illustrates a pushdown Automata that recognizes languages.

$\{ a^i b^j c^k \mid i, j, k \geq 0 \text{ and } i = j \text{ or } i = k \}$

Solve :

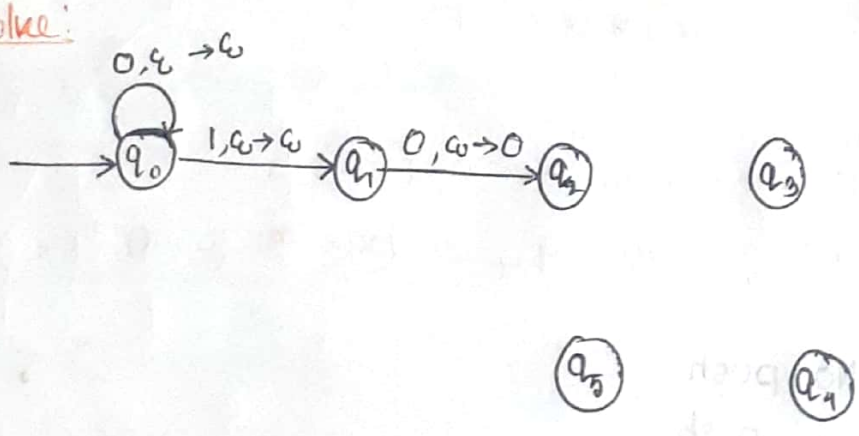


a = b
a = c

→ ଏହାର ଭାଷା ହେଉ ପାରେ aa, bbb, cc
aa bb cccc

* 2.a)ii) $\{ w \in \{0,1\}^* \mid w$ contains at least (7) 0's including emp
str

Solve:



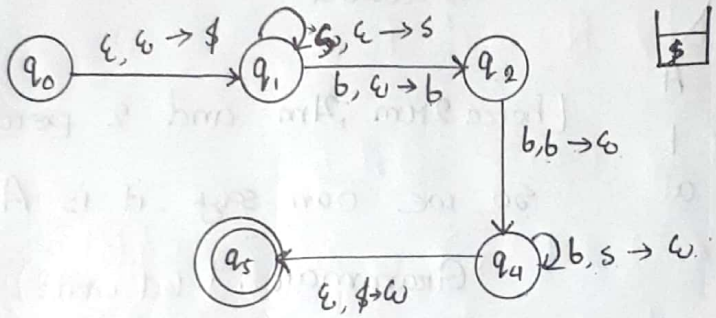
(i) $L = \{ s^n b^{n+2} \mid n >= 0 \}$

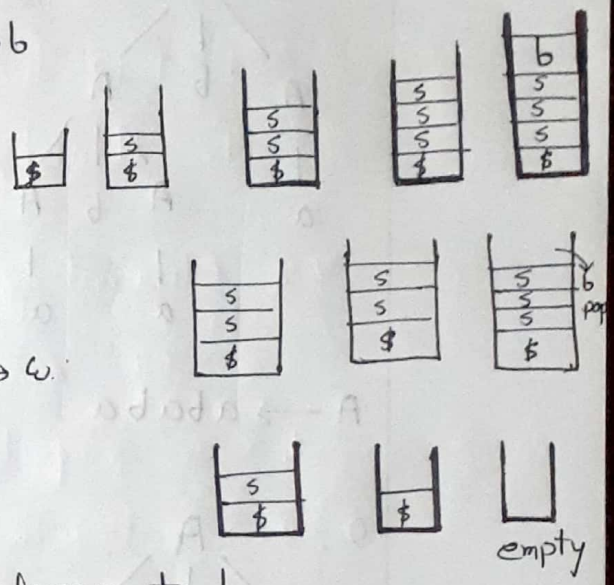
if $n = 2$, $L = ssbbbb$

if $n = 3$, $L = sssbbbbb$

$L = sbbb$, $ssbbbb$, $sssbbbbb$, $ssssbbbbbb$, ...

now, we have to design for $sssbbbbb$



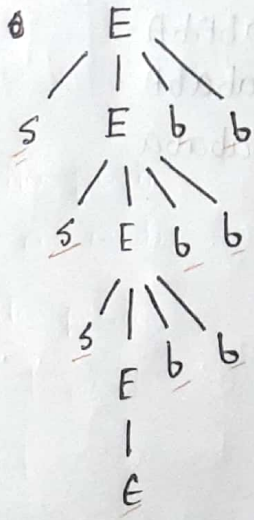


3.16 * $L = s^n b^{2n}$

(44) = let take string $sssbbbbbb$ that belongs to L

$E \rightarrow sEbb$

$E \rightarrow \epsilon$



(45) $A \rightarrow AbA$

$A \rightarrow a$

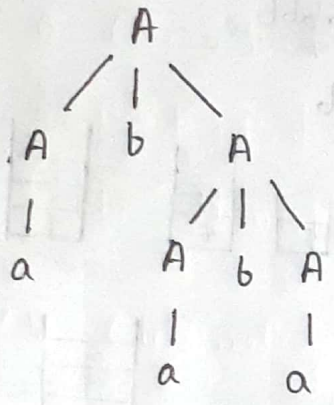
it generate = $sss \epsilon bbbbbb$ = $sssbbbbbb$

Prove that below grammar is an ambiguous –

A → AbA

A → a

1.



A ⟶ ababa

$A \xrightarrow{lm} A$

→ AbA

→ abA

→ abAbA

→ ababA

→ ababa

$A \xrightarrow{rm} A$

→ AbA

→ AbAbA

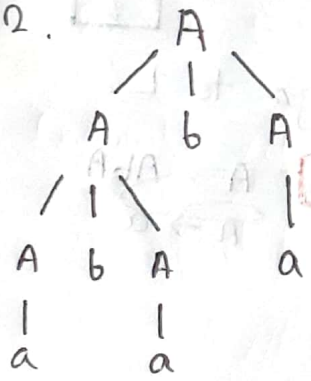
→ AbAba

→ Ababa

→ ababa

[here 2rm, 2lm and 2 perse tree so, we can say, it is Ambiguous Grammar.] last write

2.



A → ababa

$A \xrightarrow{lm} A$

→ AbA

→ AbAbA

→ abAbA

→ ababA

→ ababa

$A \xrightarrow{rm} A$

→ AbA

→ Aba

→ AbAba

→ Ababa

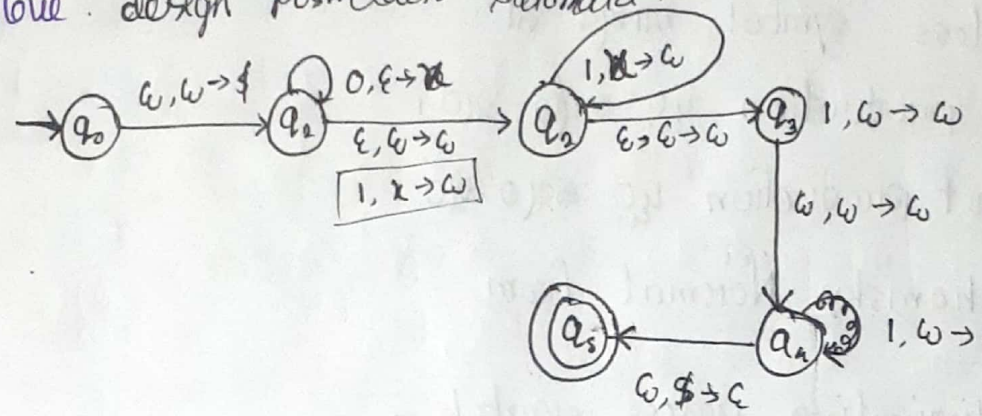
→ ababa

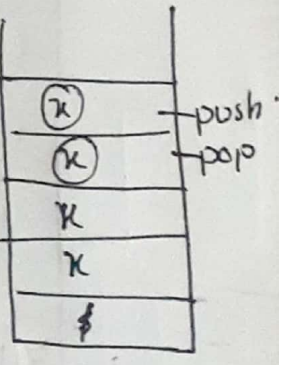
CT#02 - 20/10/2022

01. $\{0^n 1^{n+2} \mid n>0\}$.

possible combination 0111, 001111, 00011111, 0000 111111 ....

Slove: design push down automata.

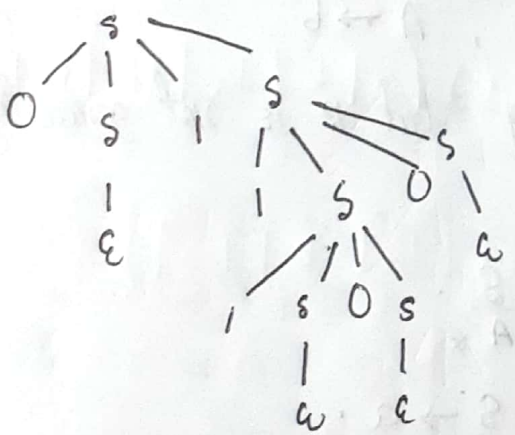




02. $S \rightarrow 0S1S \mid 1S0S \mid \varepsilon$

string → 011100

Slove: Parse tree -



now, $S \xrightarrow{lm}$ 0S1S

→ 0ε1S [ω represent null set]

→ 01S

→ 011S0S

→ 0111S0S0S

→ 0111ω0S0S

→ 0111 0S0S

→ 0111 0ε0S

→ 0111 00S

→ 0111 00ε

→ 011100

Chapter 7 = Properties of Context-free languages

Property of concept.

1. Useless symbol থাকবে না।
2. ϵ production থাকা যাবে।
3. Unit production থাকা যাবে।
4. Chomsky Normal from

7.1 = Eliminating useless symbols –

S = Start symbol ($\xrightarrow{*}$) ; w = word (string)

1. X is generating, $x \overset{*}{\Rightarrow} w$
2. X is reachable, $S \overset{*}{\Rightarrow} \alpha X \beta$

Example 7.1: $S \to AB \mid a$, $A \to b$

= All symbols are generating but B is not generating

So, $S \to AB \mid a$ বাদ

means, $\frac{S}{a}$ but not going $\frac{S}{A}$ X.

So, it's only generating $S \to a$.

আবার, a = small letter terminal.

আর, A = capital letter যা production

যে যে ফর্ম, production থাকা যাবে। তার অতিরিক্ত production

কে ϵ দিয়ে replace করা যাবে।

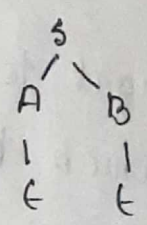
Ex: $S \to OS1 \mid \epsilon$

string : $S \to \boxed{\epsilon}$ X

7.13 : Eliminating ϵ production -

xm 7.8 : ১. $S \to AB$, ।

২. $A \to aAA \mid \epsilon$ ৩. $B \to bBB \mid \epsilon$



Solve: need replace ϵ.

1. $S \to A\epsilon \mid \epsilon B \mid AB$

   $S \to A \mid B \mid AB$

2. $A \to aAA \mid a\epsilon\epsilon \mid a\epsilon A \mid aA\epsilon$

   $A \to aAA \mid a \mid aA \mid aA$ same so, 1 pick

3. $B \to bBB \mid b\epsilon\epsilon \mid b\epsilon B \mid bB\epsilon$

   $B \to bBB \mid b \mid bB$

Step 3 : Unit production - $A \to B$

- এখানে একটা variable এর string একটা variable হয়ে যাওয়া না

1. $I \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1$
2. $F \to I \mid (E)$
3. $T \to F \mid T * F$
4. $E \to T \mid E + T$

যে production এ যে line টায় কোন Unit production নাই সেই line গুলা replace করতে হবে।

after replace ⇒ $F \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E)$

$T \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F$

$E \to a \mid b \mid Ia \mid Ib \mid I0 \mid I1 \mid (E) \mid T * F \mid E + T$

Let us, convert the grammar to CNF.

For part xm 7.12, here notice that there are eight terminals a, b, 0, 1, +, * (, and), each of which ~~the new variable is replaced by its body~~ appears in a body that is not a single terminal.

Using the obvious initials as the new variables,

A → a B → b ~~O → 0~~ ~~P → +~~

Zero, Z → 0

One, O → 1

~~Position~~

Plus, P → +

Multiplecation, M → *

Left, L → (

Right, R →)

Ans: E → E+T | T*E | (E) | a | b | Ia | Ib | IO | I1 |

replace result,

E → EPT | TME | LER | a | b | IA | IB | IZ | IO

question pattern

1. turing Machine
2. PDA
3. grammar
4. a. CFG b. tm, rtm
5. a. Mid topic
   b. theory + Pumping ***
      must do : defination

   OR, a. b.

6. a. Mid minimize DFA
   b. theory FA, NFA, DFA.