REG
↓
REC
↓
TM

Definition: Type-0

A phase structure grammer/ unrestricted grammer/ Recursively enumerable grammer is a portable 4-tuple grammer.

$$G = (V, T, P, S) \quad \text{or} \quad G = (N, T, P, S)$$

Where,

If the production rules is of the form, $\alpha \rightarrow \beta$

where $\alpha \in (V+T)^* V (V+T)^*$

$\beta \in (V+T)^*$

Examples: $aAb \rightarrow bb$ $\quad aAb \rightarrow b$ derived

$aA \rightarrow \epsilon$ (a|b|0|1) strings should be in Terminal symbols

↓ Variables/Terminals ↓ Terminals

Language generated by Grammer G is set of terminal symbols in the grammer from Start Symbol.

eg: (1)

Startsymbol

$S \rightarrow aSc$

$S \rightarrow aAC$

$A \rightarrow b$

Production rules

$G = (V, T, P, S)$

$T = \{a, b, c\}$

$V = \{S, A\}$

$S \Rightarrow aSc$

$\Rightarrow aaAcc$ (or) $S \Rightarrow aSc$

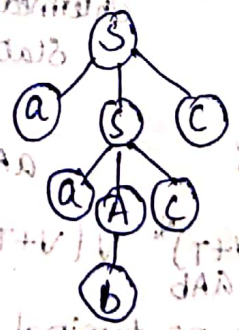
$\Rightarrow aabcc$ $\Rightarrow aascc$

$\Rightarrow aaaScc$

$\Rightarrow aaaaAccc$

$\Rightarrow aaaabcccc$

$\Rightarrow a^4bc^4$



2. Generate a String aabcc using the grammer $G = (V, T, P, S)$, where

V = Variables $\{S, A\}$, $T = \{a, b, c\}$ and

$P = \{aSc, aAC, b\}$

Type-1 Grammer

$A \rightarrow \epsilon$ Not allowed

If the production rules are of the form $\alpha A \beta \rightarrow \alpha \gamma \beta$

where $\alpha, \beta \in (NUT)^*$, → There can be null

$A \in N$.

$\gamma \in (NUT)^+$ → There consists atleast one terminal or variable.

is called Type-1 Grammer or Contex Sensitive grammer or Non-contracting Grammer

CSG → CSL → Linear bounded automaton

(or)

If in a production rule $u \rightarrow v$, $|u| \leq |v|$ grammer is called length-increasing grammer.

($|u|$ = Length of u, $|v|$ = Length of v)

Examples: 1. $aAb \rightarrow bbb$ satisfies CSG

2. $aA \rightarrow b$ not CSG.

Eg: Generate a String ab^2cd^2 using the production rules are

$S \rightarrow aAB \mid aB$

$A \rightarrow aAX \mid ax$

$B \rightarrow bBd \mid bYd$

$Xb \rightarrow bX$

$XY \rightarrow Yc$

$Y \rightarrow c$

$T = \{a, b, c, d\}$

$V = \{S, A, B, X, Y\}$

$S \Rightarrow aB$ ②

$\Rightarrow abYd$ ⑥

$\Rightarrow abcd$ ⑦

or

$S \Rightarrow aAB$ ②

$\Rightarrow abBd$ ⑤

$\Rightarrow abbBdd$ ⑥

$\Rightarrow abbbdd$ ⑨

$\Rightarrow ab^2cd^2$

$S \Rightarrow aAB$

$\Rightarrow aaXB$

$\Rightarrow aaXbBd$

$\Rightarrow aabxBd$

$\Rightarrow aabxbYdd$

$\Rightarrow aabxcdd$.

$\Rightarrow aa$

*Type-2 Grammer

If in a grammer, rules are of the form $A \to \alpha$ where $A \in N$ and $\alpha \in (NUT)^*$, grammer is called Type-2 grammer Contex-free Grammer (CFG).

Contex Free → CFL → Pushdown Automata

Language generated by CFG is called CFL and then machine required is Pushdown-Automator.

$A \to \alpha$

$A \in N$, $\alpha \in (NUT)^*$

$D \to \epsilon$

accepted by machine

Example: 1. $D \to \epsilon$

2. $A \to BCD$

Adv: Multiple variables can be present at righthand side

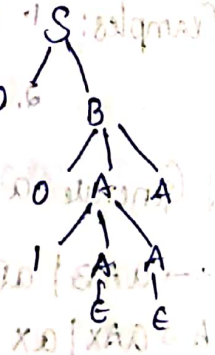
eg: ① $G = (V, T, P, S)$

$V = \{S, A, B\}$

$T = \{0, 1\}$

$P = \{S \to \underset{①}{0B}, A \to \underset{②}{1AA} \mid \underset{③}{\epsilon}, B \to \underset{④}{0AA}\}$, 001

$S \Rightarrow 0B$
$\Rightarrow 00AA$
$\Rightarrow 001AA$
$\Rightarrow 001\epsilon\epsilon$
$\Rightarrow 001$



Derivation Tree

(LDT) Left most Derivation tree

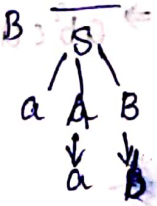
(RDT) Right most Derivation tree

② $S \to a$
$S \to aAS$
$A \to bS$
$W = abaabaa$

$S \to aAS$
$\to aAAAS$
$\to absaAS$
$\to abaabbSS \Rightarrow abaabSS$
$\to ababbSa \Rightarrow abaabaa$
$\to aa$

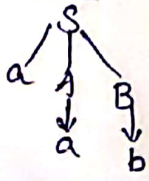
LDT

$S \to aAB$
$A \to a$
$B \to b$



aab

RDT



aab

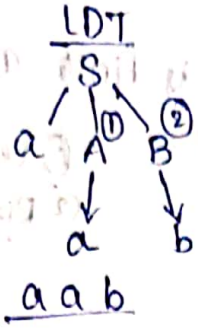
Types of Derivation Trees

① Left most Derivation Tree

② Right most Derivation Tree

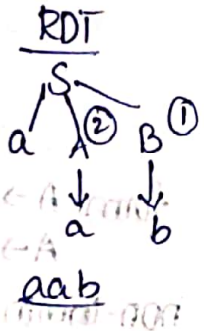
① LDT : Obtained by Applying productions to leftmost variable in each step

eg: $S \to aAB$
$A \to a$
$B \to b$



a a b

② RDT : Obtained by applying productions to rightmost variable in each step.



aab

Derive Left Derivation Tree and Right Derivation Tree for the following grammer. $S \to aAs \mid ass \mid \epsilon\}$, aabaa
$A \to sbA \mid ba$

$S \to$

Type-3 Grammer:

If the production rules are of the form $A \to \alpha B$ and $A \to \beta$

Where $A, B \in N$
$\alpha, \beta \in T^*$ is called Right Linear grammer or Type-3 grammer and language generated is called Type-3 languague or Regular Set the machine accepted is finite automata.

RG → RLG – $A \to \alpha B / \beta$
$A \to xB / x$
LLG
$A \to B\alpha / \beta$
$A \to Bx / x$

RG → RL → finite Automata

Regular Grammer
① Left Linear Grammer
② Right Linear Grammer.

If the production rule of the form $A \to B\alpha / \beta$ or $A \to Bx / x$.

$A \to \alpha B$ & $A \to \beta$ is called left linear Regular Grammer
$A, B \in N$
$\alpha, \beta \in T^*$

Eg:

If all the production rules of the form $A \to aB$ / $A \to a$ that is if the terminal symbol are to be left of non-terminal symbol, then this grammer is called ~~Left~~ Right Regular Grammer.

If the Production rules of the form $A \to Ba$ / $A \to a$ i.e, if the terminal symbol are to be right of the non-terminal symbol. it is called ~~Left~~ Right Regular Grammer

Eg: ① $S \to as$
$S \to bS$
$S \to \epsilon$

② $S \to AB$
$A \to a$
$B \to b$
} ab

$S \to AB$
$\Rightarrow aB$
$\Rightarrow ab$

3. $S \to AB$
$A \to aA/a$
$B \to bB/b$

$a^m b^n$
$m \geq 0,$
$n \geq 0.$

Finite Automata < with o/p / without o/p

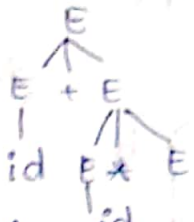
Grammer

Ambiguous
(having more than two forms)

unAmbiguous
only one form

$E \to E+E$
$E \to E*E$
$E \to id$

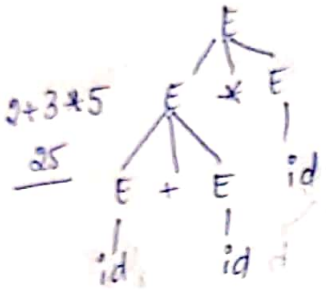
productions { centential or / Sequential form



2+3*5
25

id+id*id

id+id*id



2+3*5
25

Parse tree

id+id*id

E ⇒ Expression
→ Combination of different expressions
→ Can be an identifier

Ambiguity / Ambiguous grammer:

In grammer $g=(N,T,P,S)$, a string in language $L(G)$ is said to Ambiguous grammer or Ambiguously derived if it has 2 or more different derivation trees.

eg: ① Centential forms / Sequential forms { $E \to E+E$, $E \to E*E$, $E \to id$ } > production rules

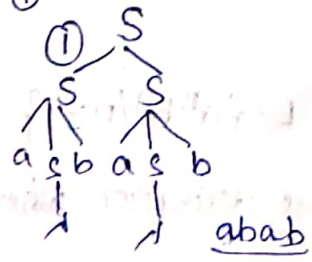
$V = \{E\}$

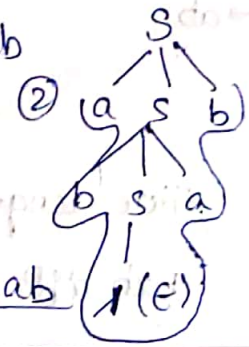
$T = \{id, *, +\}$

In any derived tree the lower level elements has higher associativity.

eg: ② $S \to asb/bsa/ss/\epsilon$ (① ② ③ ④), derive string abab

$S \to asb$
$S \to bsa$
$S \to ss$
$S \to \epsilon$



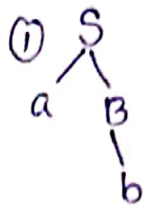


abb

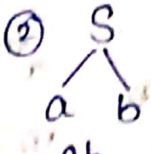
3) $S \to aB \mid ab$

$A \to aAB$

$B \to Abb \mid b$

① 

ab

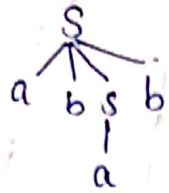
② 

ab

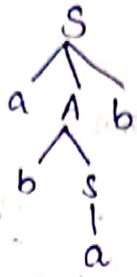
4) $S \to a \mid aAb \mid absb$

$A \to aAAb \mid bs$

abab

① 

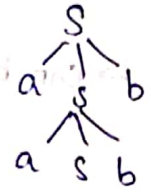
abab

② 

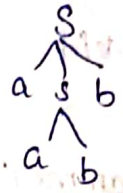
abab

⇒ 5) $S \to asb$

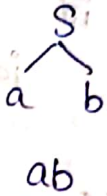
$S \to ab$



a^2b^2 / a^nb^n

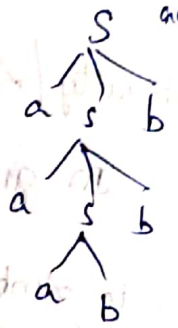


aabb



ab

a^3b^3 aaabbb



$L = \{a^n b^n / n \geq 1\}$ where $n=1$, ab

$n=2$, a^2b^2

Infinite no. of strings $n=3$, a^3b^3, ----------

Each and every string will generates different forms is called inherently ambiguous language. (This is a unambiguous)

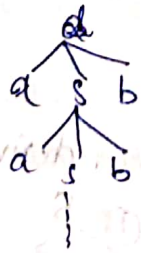
Unambiguous:

Consider grammer G with the production rules

$S \to asb$

$S \to ab$

, derive the string



(a^2b^2)

Here language is $L = \{a^n b^n / n \geq 1\}$

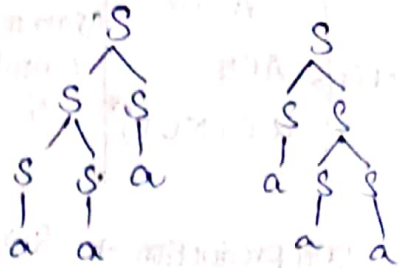
Each a^nb^n as unique derivation where rule① is used $(n-1)$ times

and rules used once at the end depending on the n value.

Hence grammer is unambiguous.

Simple grammer: $S \to SS$ $L = \{a^n / n \geq 1\}$
$S \to a$

a^3 (aaa)



- Abiguous Grammer

Inherently ambiguous:

A language may have grammers $G_1, G_2, \ldots G_n$, if all of them are ambiguous then language L is said to be Inherently -Ambiguous.

[length increases no. of derivation trees also increases.
a^3 = aaa, a^4 aaaa]

consider grammer G having rules $S \to SS$ $S \to a$ then a^3 has 2 different derivation trees and

a^4 has 5 derivation trees and a^5 consists of 14 derivation tree.

finally, as the length increases number of derivation trees also increases.

⇒ Whether the grammer is ambiguous or not

→ ① ST CFG $S \to SS/a/b$ is ambiguous (show that)

② a) $S \to \varepsilon$
$S \to asb$
$S \to bSa$
$S \to SS$

b) $S \to AA$
$A \to AAA$
$A \to bA$
$A \to Ab$
$A \to a$

c) $S \to A$
$S \to asb$
$S \to bs$
$A \to Aa$
$A \to a$

③ $L = \{a^n b^n c^p / n, p \geq 1\}$

④ $L = \{a^n b^m c^m / n, m \geq 1\}$

⑤ $L = \{a^n b^m c^p / n, m, p \geq 1, n = m \text{ or } m = p\}$

ikation of Contextfree Grammer (CFG):

$A \to \alpha$

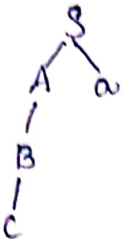
$A \in N$

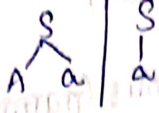
$\alpha \in (NUT)^*$

Grammer consists of
- ⇒ ε rules (epsilon) ($B \to \epsilon, C \to \epsilon$,)
- ⇒ Unit production ($A \to B, C \to D, \dots$) → $S \to Aa$, $A \to B$, $B \to C$
- ⇒ Useless Symbols ($S \to Aa/a$, $B \to b$)

} normal elimination

Unit production → $S \to Aa$, $A \to B$, $B \to C$





⇒ Simplification of Contextfree Grammer:

Simplification is done on productions and symbols of grammer. modify grammer by removing the following rules.

1. presence of ε rules in any production.
2. Unit rule productions and
3. Useless Symbols.
   1. Remove Symbols not deriving terminal strings and
   2. Remove Symbols not reachable from S.

1. Removal of ε-rules or ε productions: (Episilon - ε) Epsilon

Any production of the form $A \to \epsilon$ is called ε-rule.

If $A \overset{*}{\Rightarrow} \epsilon$, A (* - no. of steps), A is a nullable symbol

$S \to A$

$A \to \epsilon$

Steps to remove nullable variables:

$S \to aSb/aAb/ab$

$A \to \epsilon$

① Find all nullable variables

② Create two versions of all productions containing nullable variables on their RHS, one without and one with null variables

③ Remove ε-productions.

④ productions whose RHS does not have any null variable are included directly.

1. $S \to asb \mid aAb \mid ab$
$A \to \epsilon$

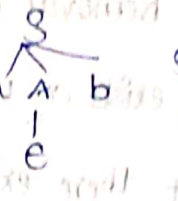
i) Nullable variable → {A}

2) $S \to asb \mid ab$ } two versions
$S \to aAb$

3) Remove ϵ production
$A \to \epsilon$

Before removing ϵ we have to try to replace in RHS.

$S \to aAb$
$A \to \epsilon$



$S \to asb \mid ab$
$S \to ab$

4. $S \to asb \mid ab$.

i. Nullable variable
{A, B, S}

2. $S \to AB$
$A \to aAA \mid \epsilon$
$B \to bBB \mid \epsilon$

$S \to AB$
$A \to aAA$
$A \to \epsilon$
$B \to bBB$
$B \to \epsilon$

2. $S \to AB$
$A \to aAA$
$B \to bBB$

3. $S \to AB \mid B \mid A \mid \epsilon$
$A \to aAA \mid aA \mid a$
$B \to bBB \mid bB \mid b$
~~$S \to AB \mid A$~~

$A \to \epsilon$
$B \to \epsilon$

(2) $S \to Abac$
$A \to Bc$
$B \to b \mid \epsilon$
$C \to D \mid \epsilon$
$D \to d$

→ $S \to Abac \mid Abc \mid bac \mid ac$
$A \to Bc \mid c \mid B$
$B \to b$
$C \to D$
$D \to d$

4.

(1) $S \to aS \mid bS \mid \epsilon$, eliminate ϵ-Production if any

(2) $S \to ABAC$
$A \to aA \mid \epsilon$
$B \to bB \mid \epsilon$
$C \to c$

(3) $S \to bef$
$E \to bEC$
$E \to GGC$
$G \to b$
$G \to kL$
$k \to ckd$
$k \to \epsilon$
$L \to dLe$
$L \to \epsilon$

(4) $S \to ese$
$S \to GH$
$G \to CGb$
$G \to e$
$H \to JHd$
$H \to e$
$J \to bJ$
$J \to f$

Removal of unit Productions: $\alpha \to \beta$, $\alpha, \beta \in N/V$

A → B }
B → C { unit
D → E } Productions

Any rule of the form $\alpha \to \beta$ or $X \to Y$, where $\alpha, \beta \in N/V$

Steps for Removal of unit productions:

*1. There exists a unitproduction A → B, Select unit production A → B such that there exists a production B → x, x is terminal

*2. For every non-unit Production B → x, Add production A → x to grammar and eliminate A → B.

eg: ① S → AB
A → a
B → C/b
C → D
D → E
E → a

unit productions
B → C
C → D
D → E

S → AB
A → a
B → b
E → a

B → C → D → E → a
C → D → E → a
D → E → a

Non-unit productions
S → AB
A → a
B → b/a
C → a
D → a
E → a

NT = {S, A, B, C}
T = {0, 1}

② S → 0A | 1B | C
A → 0S | 00
B → 1/A
C → 01

unit productions
S → C
B → A

S → C → 01
B → A → 0S / 00

NonUnit Productions
S → 0A | 1B | 01
A → 0S | 00
B → 1 | 0S | 00
C → 01

eg: ① S → Aa/B
B → A/bb
A → a/bc/B

S → B
B → A
A → B

S → Aa/a/bc/bb
B → bb/a/bc
A → a/bc/bb
A → Abbs
B → aaa

S → B → A → a/bc
B → A → a/bc (bb), A → B → bb

② S → CBA
S → B
A → CB

③ S → AB
A → a
B → C/b
C → D
D → E/bc
E → d/Ab

S → AB
B → C
C → D
D → E
E → ~~∅~~

S → AB
A → a
B → b/d/Ab
D → bc
E → d/Ab

B → C → D → E → d/Ab

② S → CBA
S → B
A → CB
A → Abbs
B → aaa

S → B → aaa

S → CBA/aaa
A → CB
A → Abbs
B → aaa

Removal of Useless Symbols:

Let G = (N, T, P, S) is a context free grammar, a variable X in N is said to be usefull if & only if there exists a string $\alpha \rightarrow L(G)$ such that

$S \overset{*}{\Rightarrow} \alpha_1 X \alpha_2 \overset{*}{\Rightarrow} \alpha$,

Where α_1, α_2 belongs to $(N \cup T)^*$, X is usefull because it appears in atleast one derivation from S to a word α.

Production involving X is usefull, otherwise it is useless.

Eg: ① S → AB/a
A → Bc/b
B → aB/C
C → aC/B

N = {A, B, C, S}
T = {a, b, ∅}

1. Remove symbols not deriving terminal string.
2. Remove symbols not reachable from S.

① {a, b, S, A}

② S → AB/a
A → Bc/b
⇒ S → AB/a, A → b
⇒ S → a, A → b

Remove production not reachable from S
the S → a

② S → AB/AC
A → aAb/bAa/a
B → bbA/aaB/AB
C → abCA/aDb
D → bD/aC

N = {S, A, B, C}
T = {a, b}

① {a, b, A, B, S} (AB)

② S → AB
A → aAb/bAa/a
B → bbA/aaB/AB

equivalent actual CFG

① $S \to ABC \mid BaB$

$A \to aA \mid Ba \mid aaa$

$B \to bBb \mid a$

$C \to CA \mid AC$

$N = \{A, B, C, S\}$

$T = \{a, b, \epsilon\}$

② $S \to Sa \mid A \mid C$

$A \to a$

$B \to bb$

$C \to aC$

① $\{a, b, A, B, S\}$

② $S \to Sa \mid A$

$A \to a$

~~$B \to bb$~~

① $\{a, b, A, B, S\}$

② $S \to BaB$

$B \to bBb \mid a$

Example: Consider Contextfree Grammer $G = (V, T, P, S)$ where write variable & Terminals where Variables are $\{S, A, B, C\}$

Terminals are $\{a, b\}$

Production are $S \to ABC \mid a$

$A \to a$

$C \to b$

$S \to a$

① $\{a, b, S, C\}$

② $S \to a$

$A \to a$

$C \to b$ ✗ $S \to a$

② $S \to aA \mid bB \mid a$

$A \to aB \mid B \mid \epsilon$

$B \to b \mid \epsilon$

3 rules.

find equivalent Contextfree Grammer

$S \to a \mid b$

① $A \to \epsilon$

$B \to \epsilon$

② $S \to bB \mid a$

$S \to aA$

$A \to aB$

$A \to B$

$B \to b$

Normal forms

$G = (N, T, P, S)$ where $A \in N^+$, $A \in N$, $a \in T$

$A \to \Delta$

$A \to a$ or $A \to \epsilon$

1) Weak chomsky NF (WCNF)

eg: $S \to ASB/AB$ $(A \to \Delta$

$A \to a$ $\}A \to T$

$B \to b$

2) chomsky NF (CNF)

Let $G=(N,T,P,S)$ be a context free grammer, if the productions or production rules are of the form $A \to BC$ or $A \to a$ where A,B,C belongs to non-terminal, $a \in T$ then grammer is said to be CNF.

$A \to BC$ where $A,B,C \in N$

$A \to a$ $a \in T$

eg: $S \to AB/AC/SS$

① $A \to 0$

$B \to 1$

$C \to SA$

eg: ② $S \to AS/SB$

$A \to AB/a$

$B \to b$

$S \to SS_1$ ← $S \to SAB$

$S_1 \to AB$

$S \to SS_2$ ← $S \to SBC$

$B \to BAB$

$S_2 \to BC$

$B \to BS_1$

$S_1 \to AB$

③ $S \to SAB/AB/SBC$

$A \to AB/a$

$B \to BAB/b$

$C \to b$

WCNF

$S \to SS_1/SS_2/AB$

$A \to AB/a$

$B \to BS_1/b$

$C \to b$

CNF

$S_1 \to AB$

$S_2 \to BC$

eg: ④ $S \to 1A/0B$

$A \to 1AA/0S/0$

$B \to 0BB/1$

→ It is not a chomsky NF

$V=\{S,A,B\}$ → The production rules voilating the CNF 1,2,3,4,6

$T=\{0,1\}$

① $S \to 1A$ ← $S \to S_1A$, $S_1 \to 1$

② $S \to 0B$ ← $S \to S_2B$, $S_2 \to 0$

③ $A \to 1AA$ ← $A \to S_1S_3$, $S_3 \to AA$

④ $A \to 0S$ ← $A \to S_2S$, $S_2 \to 0$

⑥ $B \to 0BB$ ← $B \to S_2S_4$, $S_4 \to BB$

→ It is not CNF but in WCNF

→ Hence for $S \to SAB$ production you will write the equivalent rules are $S \to SS_1$ & $S_1 \to AB$.

→ for $S \to SBC$ production equivalent rules are

$S \to SS_2$

$S_2 = BC$.

Ans

④ CNF

$S \to S_1A \mid S_2B$

$A \to S_1S_3 \mid S_2S \mid 0$

$B \to S_2S_4 \mid 1$

$S_1 \to 1$

$S_2 \to 0$

$S_3 \to AA$

$S_4 \to BB$

⑤ $S \to \overset{①}{b}A \mid \overset{②}{a}B$

$A \to \overset{③}{b}AA \mid \overset{④}{a}S \mid \overset{⑤}{a}$

$B \to \overset{⑥}{a}BB \mid \overset{⑦}{b}S \mid \overset{⑧}{b}$

1, 2, 3, 4, 6, 7

① $S \to bA$ < $S \to S_1A$, $S_1 \to b$

② $S \to aB$ < $S \to S_2B$, $S_2 \to a$

③ $A \to bAA$ < $A \to S_1S_3$, $S_3 \to AA$

④ $A \to aS$ < $A \to S_2S$, $S_2 \to a$

⑤ $B \to aBB$ < $B \to S_2S_4$, $S_4 \to BB$

⑥ $B \to bS$ < $B \to S_1S$, $S_1 \to b$

$S \to S_1A \mid S_2B$

$A \to S_1S_3 \mid S_2S \mid a$

$B \to S_2S_4 \mid S_1S \mid b$

$S_1 \to b$

$S_2 \to a$

$S_3 \to AA$

$S_4 \to BB$

⑤ $S \to bA \mid aB$

$A \to bAA \mid as \mid a$

$B \to aBB \mid bS \mid b$

④

$S \to BA \mid AB$

$A \to SA \mid AS \mid a$

$B \to SB \mid BS \mid b$

⑥ $S \to aaaaS$

$S \to aaaa$

3. Strong Chomsky Normal form (SCNF):

Context free grammer $G=(N,T,P,S)$ is said to be in Strong Chomsky Normal form when rules in P are of the form $A \to a$, $A \to BC$ where $A,B,C \in N$, $a \in T$, subject to the conditions if

① $A \to BC \in P$, then $B \neq C$

② $A \to BC \in P$, then foreach rule $X \to DE \in P$; $E \neq B, D \neq C$

eg:

Greibach Normal form (GNF):

If grammer $G=(N,T,PS)$ be a contexfree grammer if each rule in Production rewrites a variable into a word in TN^* ($Var \to TN^*$) that is each rule of the form $A \to a\alpha$ where $A \in T, \alpha \in N^*$, then grammer is said to be GNF.

$A \to aV/a$

Used to Construct a machine PDA (pushDown Automator) for Contexfree Grammes.

eg: ① $S \to abasa/aba$

→ Replace $A \to a$; $B \to b$

$S \to aBASA$

$S \to aBA$

$A \to a$

$B \to b$

② $S \to asb/ab$

Replace $B \to b$

$S \to aSB$

$S \to aB$

$B \to b$

③ $S \to AB$

$A \to aA/aB/b$

$B \to b$

Replace $A \to a$

$S \to aB$

$A \to aA/aB/b$

$B \to b$

$A \to a$

④ $S \to abSb/aa$

Replace $A \to a, B \to b$

$S \to aBSB$

$S \to aA$

$A \to a$

$B \to b$

Find CNF and GNF equivalent to the following grammer.

1) $S \to S \wedge S$

$S \to S \vee S$

$S \to \neg S$

$S \to (S)$

$S \to P$

$S \to q$

2) $E \to E+E$

$E \to E*E$

$E \to (E)$

$E \to a$

05/02/2020

UNIT-02

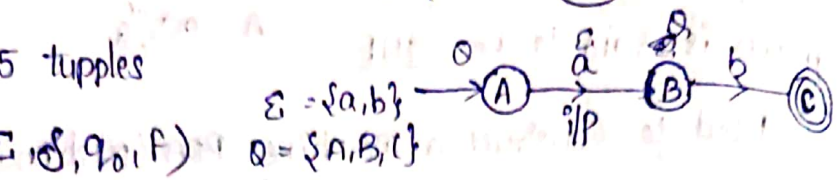
- Finite State Machine / Finite State Automaton / finite state Automata / Finite Automata
  - → Deterministic finite Automata
  - → Non-Deterministic finite Automata

FA consists of 5 tupples

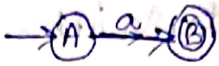
$(Q, \Sigma, \delta, q_0, F)$

$\Sigma = \{a, b\}$
$Q = \{A, B, C\}$
$\delta : Q \times \Sigma \to Q$

eg: ① ab



② a



③ $L = \{\epsilon, a, aa, ---\}$

representation of ϵ

Initial State itself is the final State. It accepts ϵ.

Q = Set of States
Σ = String of alphabets (Terminal Symbols)
δ = Transition function
q_0 = Initial State
F = final state

Initial state allways starts with input arrow →
final state always ends with two circles (double circles). ◎

Finite State Automata (FSA):

It is a simplest machine or recognising device.

figures are called State diagrams.
Nodes represent States with labels inside them.
Initial state is marked with an arrow pointing to it.
Final States are represented as double circles.
Transition from one State to another State is represented by directed edge.

Deterministic Finite State Automata (DFSA)

formal definition for DFSA is

It is a ~~finite~~ five tupple $(Q, \Sigma, \delta, q_0, F)$ or $(K, \Sigma, \delta, q_0 f)$

where Q is finite Set of States
Σ is Finite set of I/p symbols

δ = transition-function mapping from one state to another state as

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

q_0 = Input State / Initial State / Start State

F = is Subset or equals to Q set of final states

$F \subseteq Q$ (There can be morethan 1 number of final states in machines).

Construct DFSA for String aaabb

$Q = \{A, B, C\}$

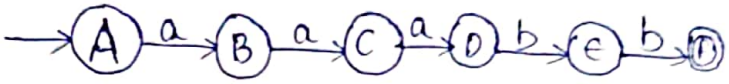
$\Sigma = \{a, b\}$

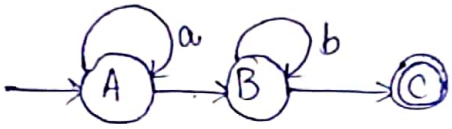
$\delta = Q \times \Sigma \to Q$

$q_0 = A$

$F = C$

$Q = \{A, B, C, D, E, F\}$





26/02/2020

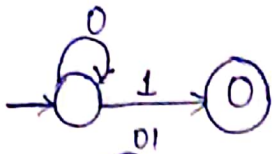
DFSA (Deterministic finite State Automata).

In DFSA, for some alphabet it goes to one State

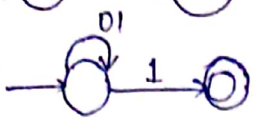
$$\delta : Q \times \Sigma \to Q$$

28/02/2020

Deterministic Finite State Automata



NonDeterministic finite state Automata



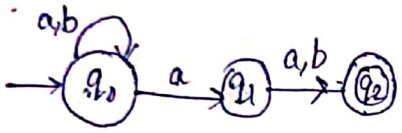
We can Convert NDFSA to DFSA but not from DFSA to NFSA

Steps to Convert NDFSA to DFSA

1. Write all the strings of language
2. NFSA
3. State Transition table for NFSA
4. State Transition table for DFSA
5. DFSA

eg: L={2nd symbol RHS is 'a'}

1. L={aa, ab, aaa, baa, ----}



2. STT for NFSA

	a	b
→q_0	$\{q_0, q_1\}$	q_0
q_1	q_2	q_2
*q_2	ϕ	ϕ

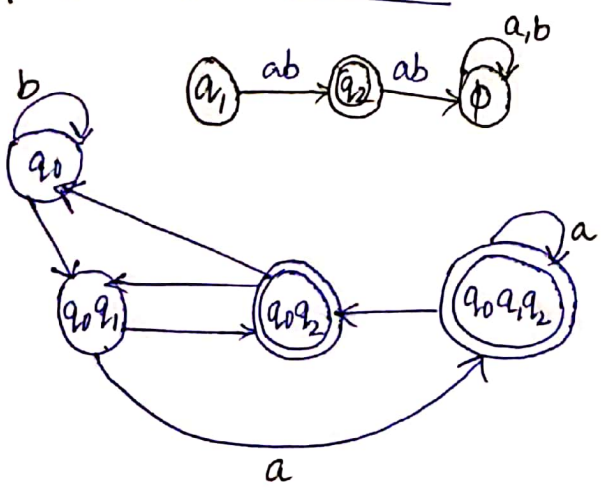
3. STT for DFSA

	a	b
→q_0	$\{q_0 q_1\}$	q_0
q_1	q_2	q_2
*q_2	D	D
$\{q_0, q_2\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_2\}$
D	D	D

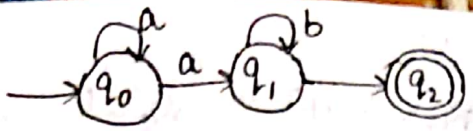
4. STT for DFSA

	a	b
→q_0	$\{q_0 q_1\}$	q_0
q_1	q_2	q_2
*q_2	D	D
$\{q_0, q_1\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_2\}$
*$\{q_0 q_2\}$	$\{q_0 q_1\}$	q_0
*$\{q_0 q_1 q_2\}$	$\{q_0 q_1 q_2\}$	$\{q_0 q_2\}$

5. DFSA



NFSA



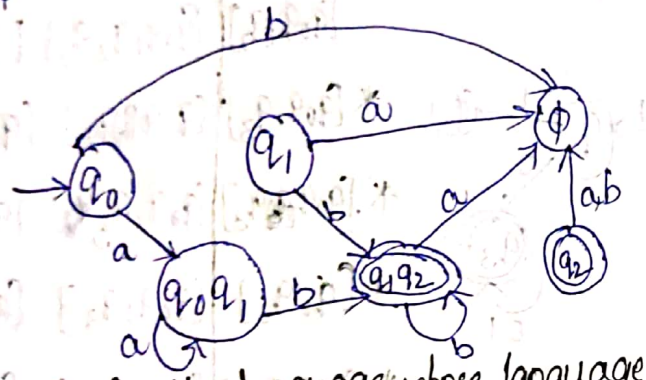
1. STT FOR NFSA

	a	b
→q_0	$\{q_0, q_1\}$	ϕ
q_1	ϕ	$\{q_1, q_2\}$
*q_2	ϕ	ϕ

2. STT For DFSA

	a	b
→q_0	$\{q_0, q_1\}$	D
q_1	ϕ	$\{q_1, q_2\}$
$[q_0 q_1]$	$\{q_0, q_1\}$	$\{q_1, q_2\}$
*$[q_1 q_2]$		

	a	b
→q_0	$[q_0 q_1]$	D
q_1	D	$[q_1 q_2]$
*q_2	D	D
$[q_0 q_1]$	$[q_0 q_1]$	$[q_1 q_2]$
*$[q_1 q_2]$	ϕ	$[q_1 q_2]$

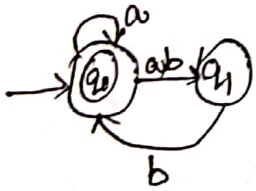


1. Construct DFSA for the language whose language accepts 3rd symbol from RHS is 'b'.

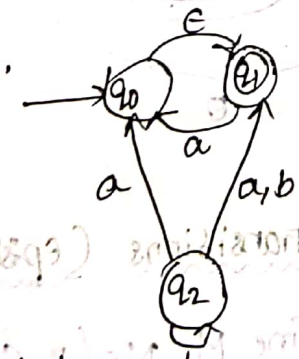
2. Construct DFSA for the language whose accepts 3rd symbol from LHS is 'b'

3. Convert the following NFSA to DFSA using using substitute direction method

1.

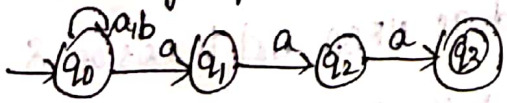


2.

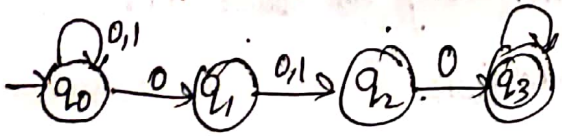


4. What is the language accepted by NFSA M.(Machine)

M:



5. Find equivalent DFSA for the following NFSA.

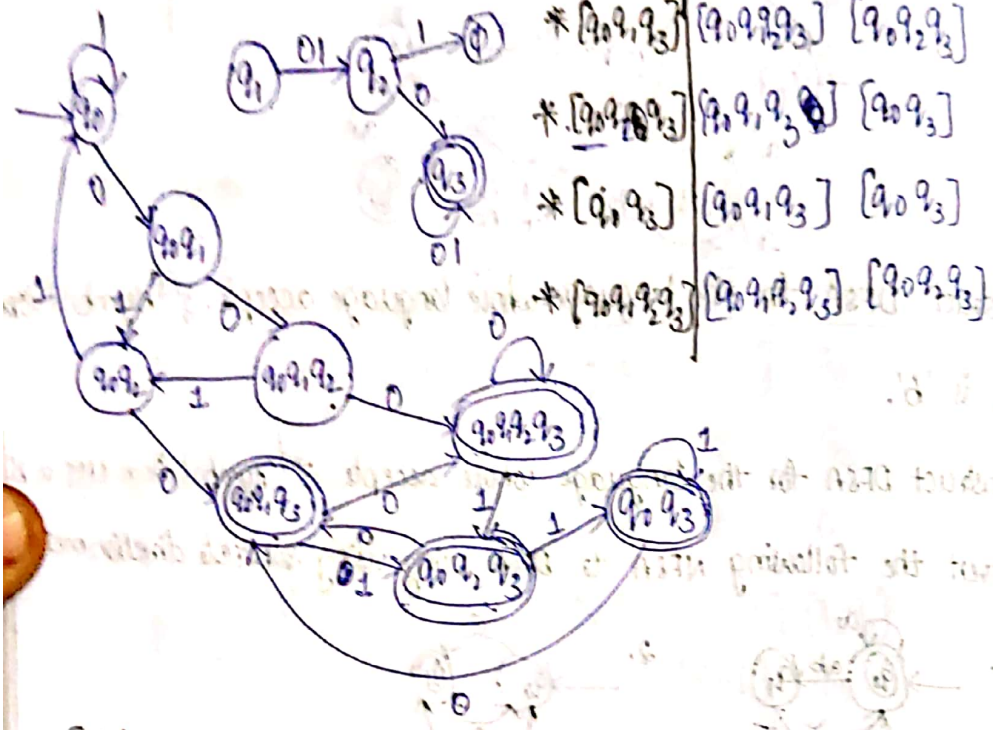


STT for NFSA

	0	1
→q_0	$\{q_0q_1\}$	$\{q_0\}$
q_1	$\{q_2\}$	$\{q_2\}$
q_2	$\{q_3\}$	$\{\phi\}$
*q_3	$\{q_3\}$	$\{q_3\}$

② STT for DFSA

	0	1
→q_0	$\{q_0q_1\}$	$\{q_0\}$
$[q_0q_1]$	$[q_0q_1q_2]$	$\{q_0q_2\}$
*$[q_0q_2]$	$\{q_0q_1q_3\}$	$\{q_0\}$
$[q_0q_1q_2]$	$[q_0q_1q_2q_3]$	$[q_0q_2]$
*$[q_0q_1q_3]$	$[q_0q_1q_2q_3]$	$[q_0q_2q_3]$
*$[q_0q_2q_3]$	$[q_0q_1q_3]$	$[q_0q_3]$
*$[q_0q_3]$	$[q_0q_1q_3]$	$[q_0q_3]$
*$[q_0q_1q_2q_3]$	$[q_0q_1q_2q_3]$	$[q_0q_2q_3]$



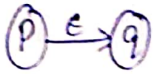
∈ Moves

NFSA with ∈ transitions (epsilon)

It is a 5 tuple machine M, $M=(K,\Sigma,\delta,q_0,F)$

Where K, Σ, q_0, F are defined as NFSA and transition δ

$\delta: K \times \Sigma \cup \{\epsilon\}$ into finite subsets of K. It is read as epsilon closure (ϵ^*) include ∈-transition to NFSA, change the state without reading any symbol represented by ∈-transition.



Construct ε-NFSA for the following

ε-closure(q_0) = {q_0, q_1}

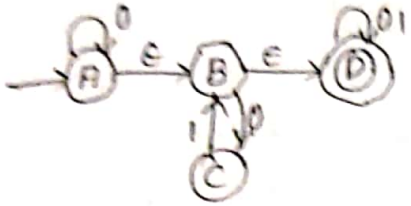
ε-closure(q_1) = {q_1}

ε-closure(q_2) = {q_2, q_3}

ε-closure(q_3) = {q_3}

$L = \{a^m b^n c^p d^q \mid m, n, p, q \geq 0\}$, $n \geq 1$

Construct ε-closure for the following



ε-closure(A) = {A, B}

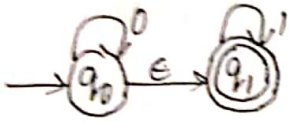
ε-closure(B) = {B, D}

ε-closure(C) = {C}

ε-closure(D) = {D}

First we have to convert ε-NFSA to NFSA and then NFSA to DFSA, but not directly from ε-NFSA to DFSA.

Convert ε-NFSA to NFSA without ε-transitions.



1. Transition Table

	0	1	ε
→q_0	{q_0}	ϕ	{q_1}
*q_1	ϕ	{q_1}	ϕ

2. Find Epislon closure

ε-closure(q_0) = {q_0, q_1}

ε-closure(q_1) = {q_1}

3. Processing States

3.1 For q_0 :-

	ε*	under 0 δ	ε*
→q_0	$q_0 \to q_0$	$\to$ {$q_0 q_1$}	
	$q_1 \to$ ϕ	$\to$ ϕ	

	ε*	under 1 δ	ε*
→q_0	$q_0 \to$ ϕ	$\to$ ϕ	
	$q_1 \to$ {q_1}	$\to$ {q_1}	

3.2 For q_1 :-

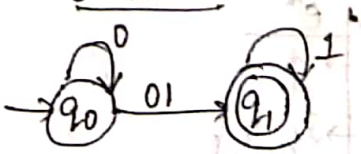
	ε*	under 0 δ	ε*
*q_1	$q_1 \to$ ϕ	$\to$ ϕ	

	ε*	under 1 δ	ε*
*q_1	$q_1 \to$ {q_1}	$\to$ {q_1}	

4. Transition Table

	0	1
→q_0	{$q_0 q_1$}	{q_1}
*q_1	ϕ	{q_1}

5. NFSA

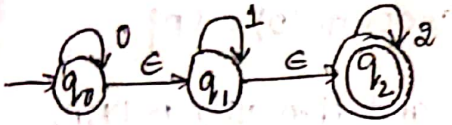


Convert ε-NFSA to NFSA without ε-transitions.

2. Find

1. Transition Table

	0	1	ε
→q_0	q_0	ϕ	q_1
q_1	ϕ	q_1	q_2
*q_2	q_2	q_2	ϕ



1. Transition Table.

	0	1	2	ε
→q_0	q_0	–	–	q_1
q_1	–	q_1	–	q_2
*q_2	–	–	q_2	–

2. Epsilon (q_0) = $\{q_0, q_1\}$

Epsilon(q_1) = $\{q_1, q_2\}$

Epsilon(q_2) = $\{q_2\}$

3. Processing States: $\delta(q_0) = \epsilon\text{-closure}\left[\delta(\epsilon\text{-clos}(q_0), \begin{matrix}0\\1\\2\end{matrix})\right]$

$\delta'(q_0)$ =

	ϵ^*	$\delta(0,1,2)$	ϵ^*
→q_0	q_0	q_0	$\{q_0, q_1\}$
	q_1	ϕ	ϕ

	ϵ^*	$\delta(2)$	ϵ^*
→q_0	q_0	ϕ	ϕ
	q_1	ϕ	ϕ

	ϵ^*	$\delta(1)$	ϵ^*
→q_0	q_0	ϕ	ϕ
	q_1	q_1	$\{q_1, q_2\}$

$\delta(q_1)$

	ϵ^*	δ_0	ϵ^*
q_1	q_1	ϕ	ϕ
	q_2	ϕ	ϕ

	ϵ^*	δ_1	ϵ^*
q_1	q_1	q_1	$\{q_1, q_2\}$
	q_2	ϕ	ϕ

	ϵ^*	δ_2	ϵ^*
q_1	q_1	ϕ	ϕ
	q_1	q_1	q_2

$\delta(q_2)$

	ϵ^*	δ_0	ϵ^*
$*q_2$	q_2	ϕ	ϕ

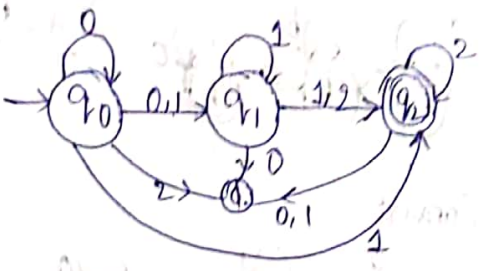
	ϵ^*	δ_1	ϵ^*
$*q_2$	q_2	ϕ	ϕ

	ϵ^*	δ_2	ϵ^*
$*q_2$	q_2	q_2	q_2

4. Transition Table for NFA

	0	1	2
$\to q_0$	$\{q_0, q_1\}$	$\{q_1, q_2\}$	ϕ
q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
q_2	ϕ	ϕ	$\{q_2\}$

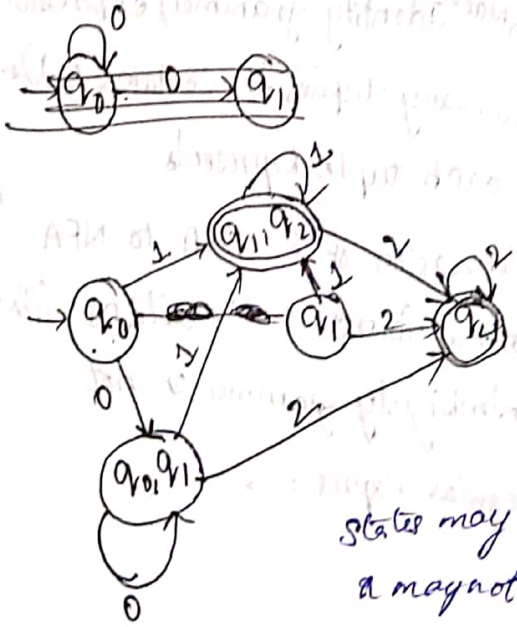
5. Representation of NFA



5. Transition State for DFSA

		0	1	2
②	$\to q_0$	$\{q_0, q_1\}$	$\{q_1, q_2\}$	ϕ
	$*q_1$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
	$*q_2$	ϕ	ϕ	$\{q_2\}$
	$[q_0, q_1]$	$\{q_0, q_1\}$	$\{q_1, q_2\}$	$\{q_2\}$
	$*[q_1, q_2]$	ϕ	$\{q_1, q_2\}$	$\{q_2\}$

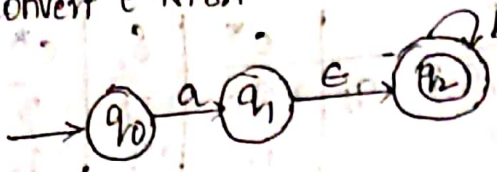
①		0	1	2
	$\to q_0$	$\{q_0, q_1\}$	$\{q_1, q_2\}$	ϕ
	q_1	ϕ	$\{q_1, q_2\}$	$\{q_2\}$
	$*q_2$	ϕ	ϕ	$\{q_2\}$



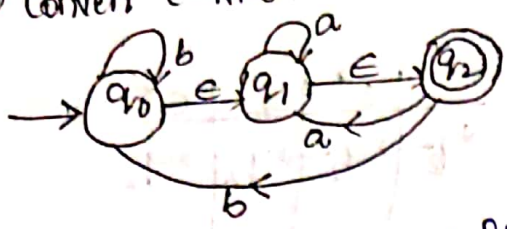
states may increase a may not

NFA → DFA

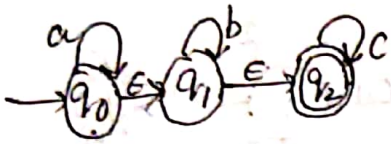
① Convert ϵ-NFSA to NFSA without ϵ-transitions.



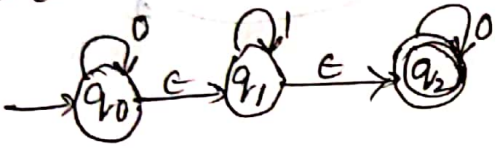
② Convert ϵ-NFSA to DFSA

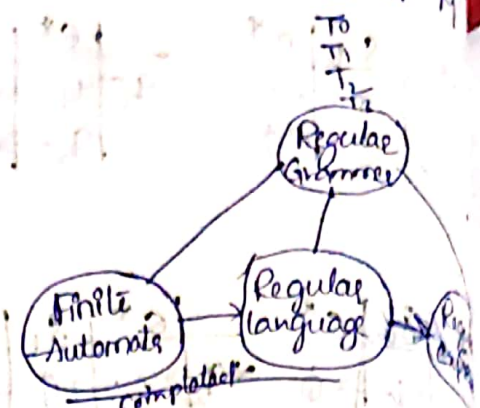


③ Convert.



④ Convert





ϵ, 0, ... * Kleene closure
0* = 0, 1, 2 ...
0 →

Type-3, Regular Grammar

Language (L)
Built finite Automata

diagram
define language

Sets Relations Graph

Identify grammer (rules), Languages, grammers, Machines

Conversion NFA to DFA

for given Machine Identify grammer/expression

- How many tuples it contains & Definition
  - each tuple Represents

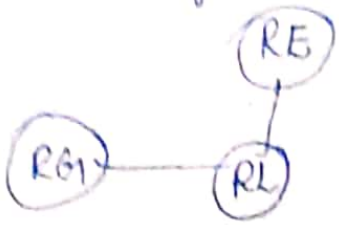
Conversion of ϵ-NFSA to NFA

Normal forms & Conditions (Production)

- Ambiguity grammer or not

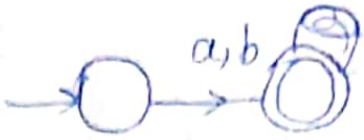
Regular expressions.

Regular expression



FA

NFA DFA

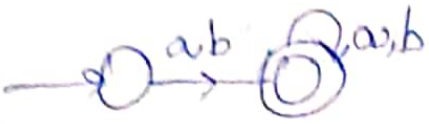


L = a or b

Regular expressions

a

b



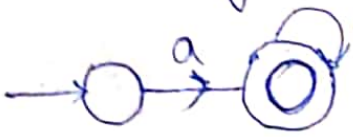
$a+b$

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

→ Regular Expression

Write a Regular expression for the following Language

Set of strings starting with a.



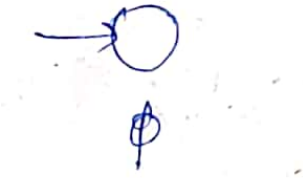
① $a(a+b)^*$

Set of strings ending with a

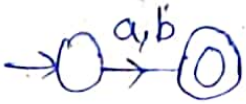
② $(a+b)^*a$

③ $(a+b)^*a(a+b)^*$

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

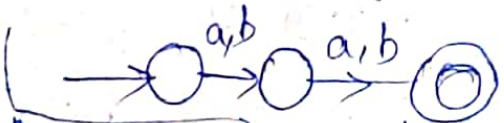


'a'



'a+b'

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



'(a+b)(a+b)'

$aa+ab+ba+bb$

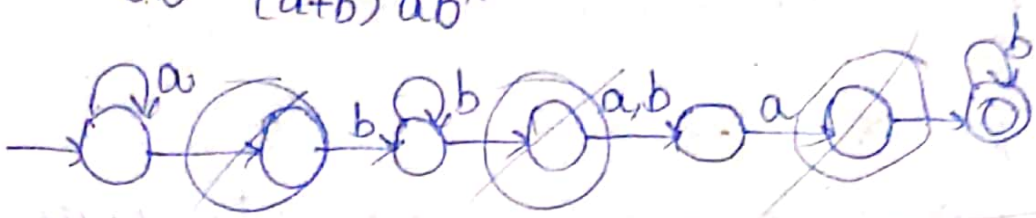
$(a+b)^*a(a+b)^*$

→ Write a Regular expression for the set of strings whose length is divisible by 6.

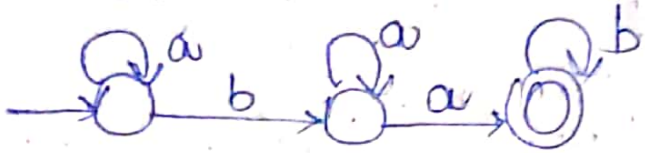
$[(a+b)^6]^*$

⇒ $a^* b a^* a b^*$

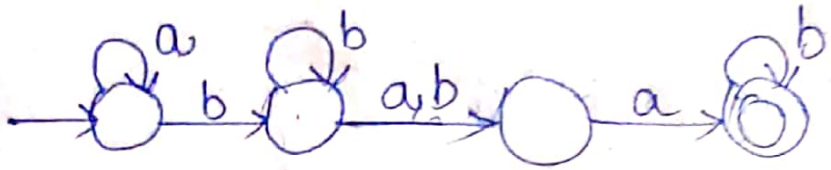
$a^* bb^* (a+b) ab^*$



① $a^* b a^* a b^*$



② $a^* bb^* (a+b) ab^*$



FA → RE

RE → FA

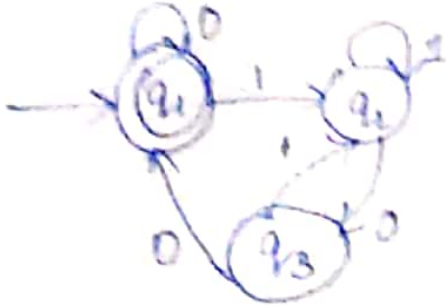
→ Set of Strings Starting & ending with different Symbols

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

→ for Same Symbols.

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

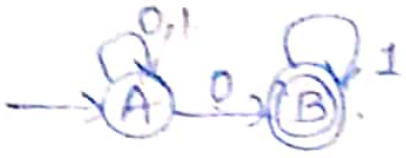
→ Automata



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

$\epsilon + 0^* + 0^* 1 1^* (01)^* 00$

Regular Grammer

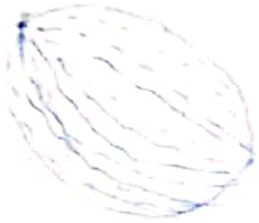


Non Terminals / Variables → A, B

Terminals → 0, 1

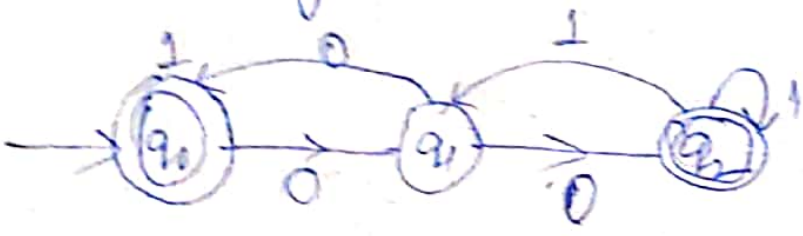
$A \to 0B \mid 0A \mid 1A$ $\quad \delta(A,0) = B$

$B \to 1B$ $\quad \delta(B,1) = B$



Regular Grammer

Construct Regular Grammer for the following NFSA



$q_0 \to 0q_1 \mid \epsilon$

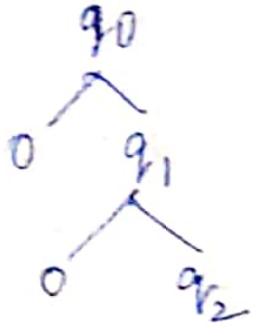
$q_1 \to 0q_2 \mid 0q_0$

$q_2 \to 1q_2 \mid 1q_1 \mid 0/1$

Grammer Consists of 4 tuples

(N, T, P, S)

Variable



Derivation tree