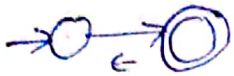
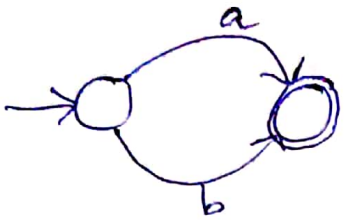
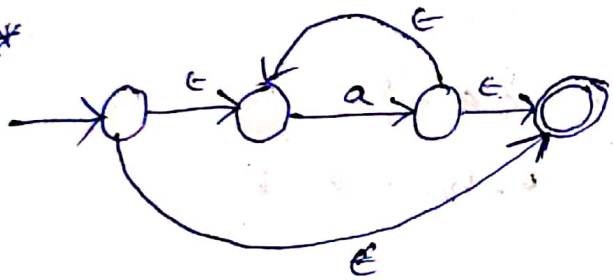
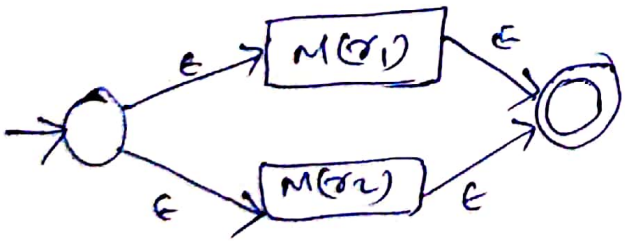
RE to Finite Automata:

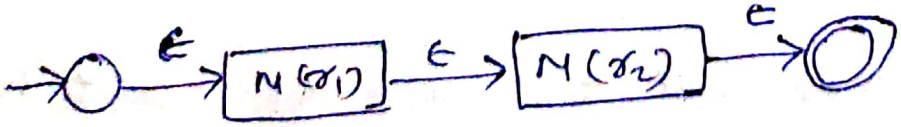
RE	NFA
ϕ	
ϵ	
a	
a+b	
a·b	
a*	

Assume $N(r_1)$ and $N(r_2)$ are finite automata for RE r_1 and r_2 respectively.

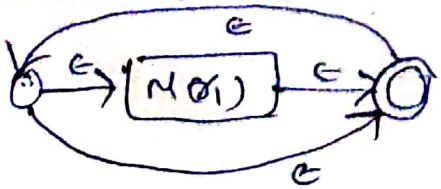
* The automata for $L(r_1 + r_2)$



* Automata for $L(r_1 \cdot r_2)$



* Automata for $L(r_1^*)$

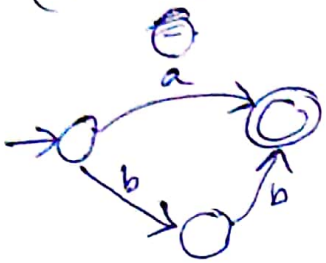


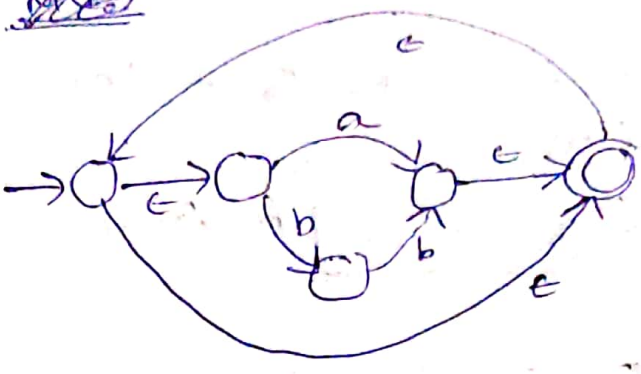
Scanned by CamScanner

Q: Design automata for $L = \underbrace{(a+bb)^*}_{r_1} . \underbrace{ba^*}_{r_2}$

M(r1)

$= (a+bb)^*$

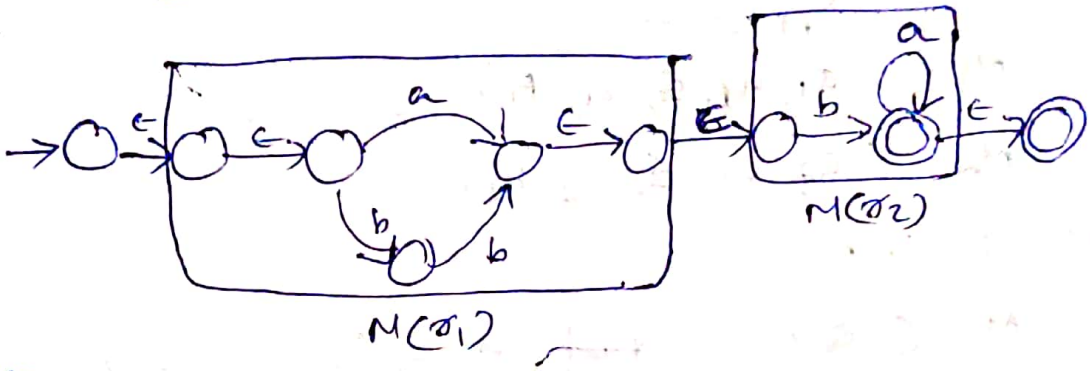




M(r2)

ba^*

M(L)



Finite automata to regular expression.

Note:

→ There are several methods to perform FA to RE with their pros and cons.

→ State elimination methods can be applied for any NFA, DFA or ε-NFA.

Arden's Theorem.

Closure property of Regular Languages.

When we produce certain operations on regular languages, then produced language is also regular.

Theorem:

If L_1 and L_2 are regular languages, then so are $L_1 \cup L_2$, $L_1 L_2$, $\bar{L}_1$, $L_1 \cap L_2$ and L_1^*

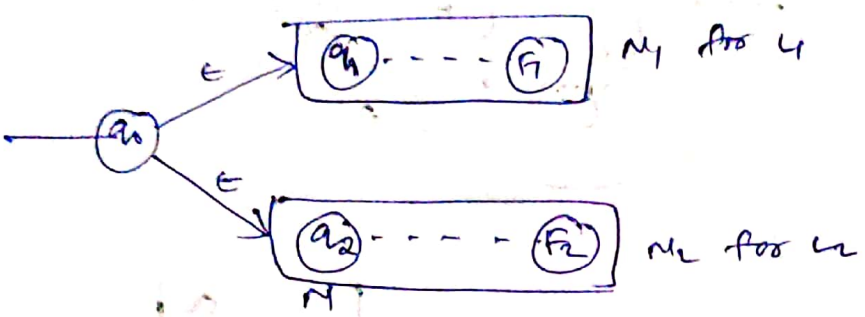
Then we say that family of regular languages is closed under

Scanned by CamScanner

union, concatenation, complement, intersection and KLEENE closure.

Note: These operations work on set of strings (regular set) of language not on RE of a language.

union operation



$M_1 = (Q_1, q_1, \Sigma, \delta_1, F_1)$

$M_2 = (Q_2, q_2, \Sigma, \delta_2, F_2)$

Let $L = L_1 \cup L_2$

$M = (Q_1 \cup Q_2 \cup \{q_0\}, q_0, \Sigma, \delta, F_1 \cup F_2)$

$x \in L(M)$

x is a string in language L

then,

$\delta^*(q_0, x) \in F_1 \cup F_2$

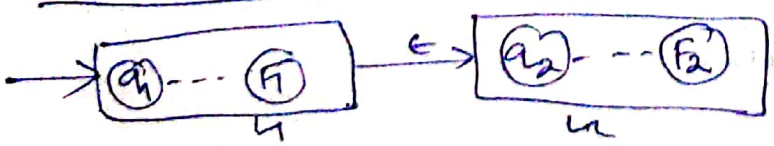
$\delta^*(q_1, x) \in F_1$ or $\delta^*(q_0, x) \in F_2$

$\delta^*(q_1, x) \in F_1$ or $\delta^*(q_2, x) \in F_2$

$x \in L_1$ or $x \in L_2$

$\Rightarrow x \in L_1 \cup L_2$

Concatenation



TP ~~$L(M) = L_1 \cdot L_2$~~ $L(M) = L_1 \cdot L_2$

Scanned by CamScanner

complement:

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

$M(\bar{L}) = (Q, q_0, \Sigma, \delta, (Q - F))$

Intersection:

$L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$

regular languages are closed under union and complement, ~~are closed~~ so intersection is also closed)

Ex: L_1 = { ~~set of~~ strings that start with a over $\{a, b\}$ }

L_2 = { strings having equal no of a's and b's } ← not a regular language

$L_1 \cap L_2$ = { start with a and have equal no of a's and b's }.

Homomorphism:

Suppose Σ and Γ are alphabets

Then a function

~~$h : \Sigma \to \Gamma^+$~~

$h : \Sigma \to \Gamma^*$

is called a homomorphism.

In other words, a homomorphism is a substitution in which a single letter is replaced by a string.

If $w = a_1 \cdot a_2 \cdot \ldots a_n$

then

$h(w) = h(a_1)\, h(a_2) \cdot \ldots \cdot h(a_n)$

$\boxed{h(L) = \{ h(w) : w \in L \}}$ homomorphic image of L

Eg: $\Sigma = \{a, b\}$ $\Gamma = \{0, 1, 2\}$

$h(a) = 01$

$h(b) = 112$

$RE(L_1) = a^* bb$

$= h(a^*) \cdot h(b) \cdot h(b)$

$RE(h) = (01)^* \cdot 112 \cdot 112$

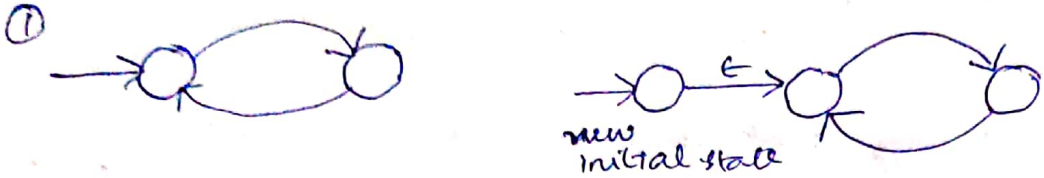
Scanned by CamScanner

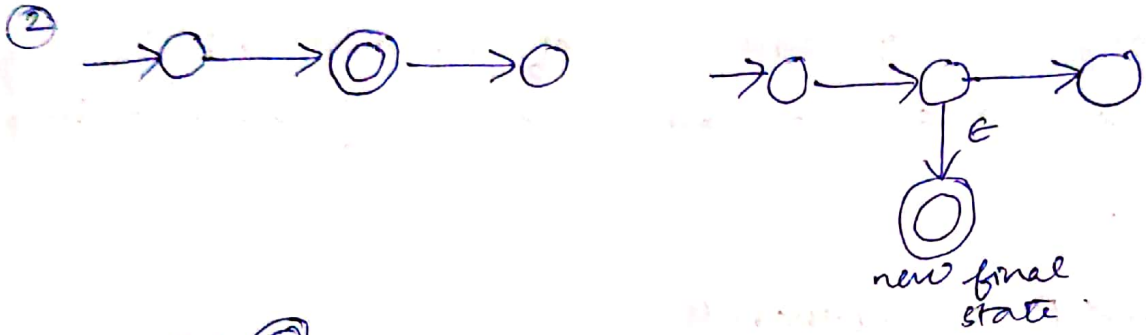
Finite automata to regular expression

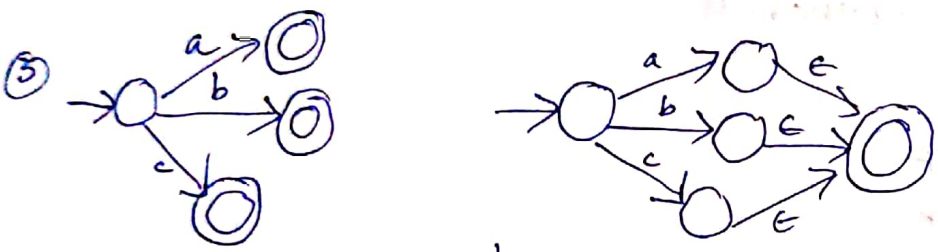
→ There are several methods to perform with their own pros and cons

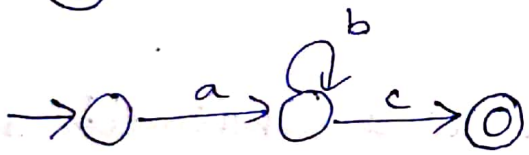
→ State elimination method is one of the general methods

Rules for state elimination method

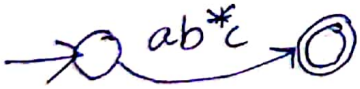
① 

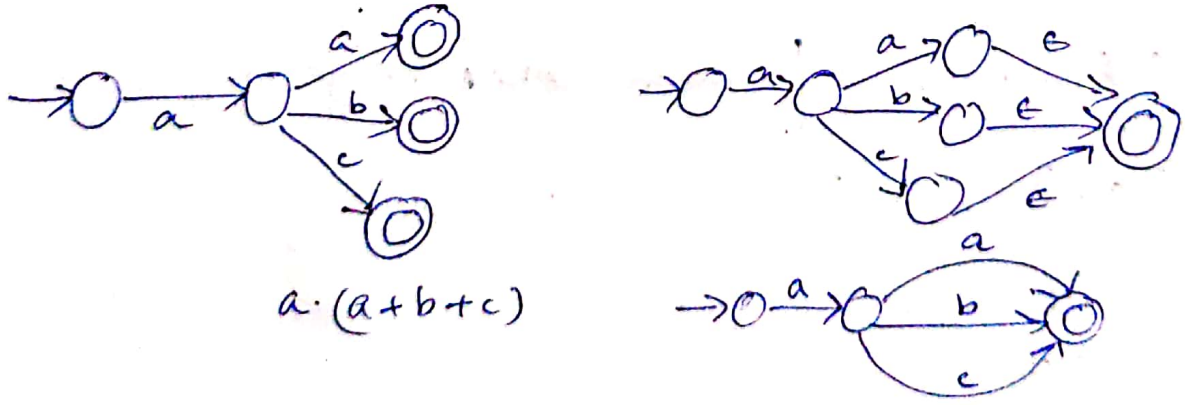
② 

③ 

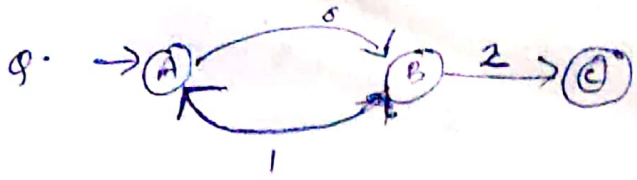
Ex: 

eliminate each node one by one except initial state and final state

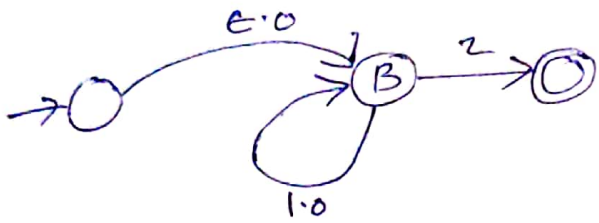


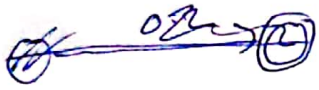
Q: 

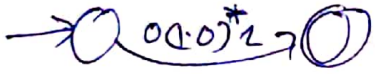
q.



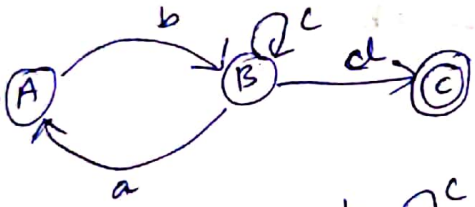
Step 1

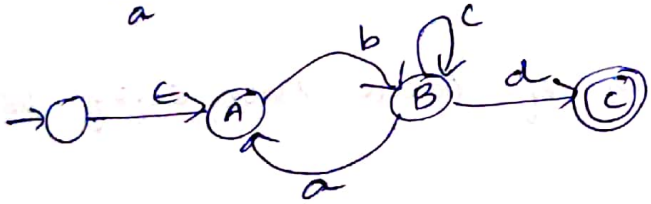


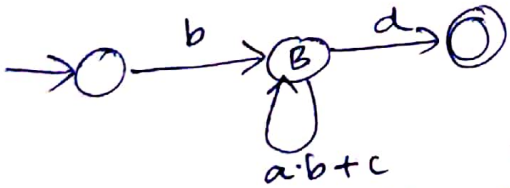


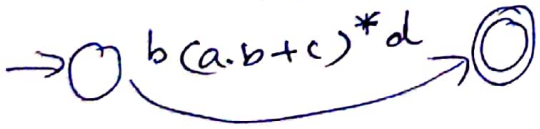


q:

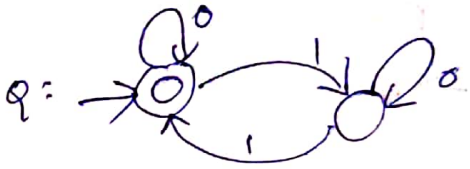


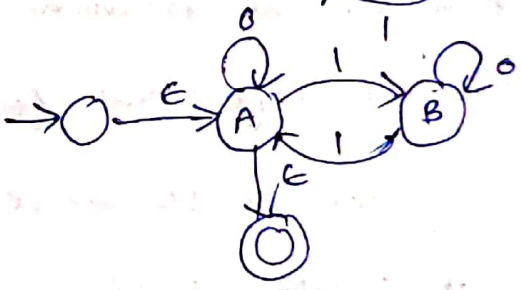


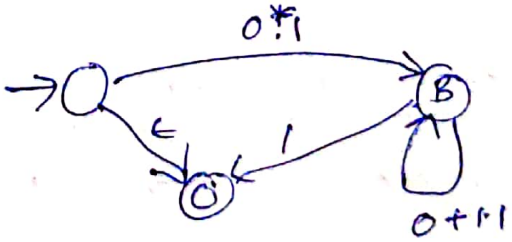




q:







$(0^*1)\cdot(0+11)^*\cdot 1 + \epsilon$

Regular grammar (RG)

RGs are associated with regular languages and that for every regular language there is a regular grammar.

Scanned by CamScanner

A grammar G is defined by quadruple $G = (V, T, S, P)$ where

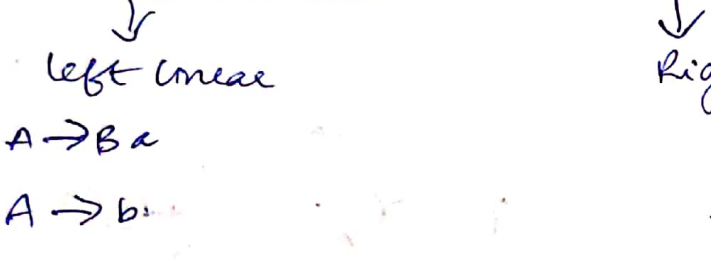
V = finite set of variables (Non terminals)

T = Finite set of Terminals (Σ)

S = start variable ($S \in V$)

P = Finite set of productions.

Regular grammar.



Here

$A, B \in V$ $a, b \in T$

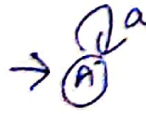
Note: If a regular grammar is in left linear then every production must be in left linear form.

→ LHS only one variable must appear.

→ RHS atmost one variable should appear

Production as operation:

For Kleen closure → left recursive or right recursive



$A \to aA$

or

$A \to Aa$

concatenation → ab

union → a|b

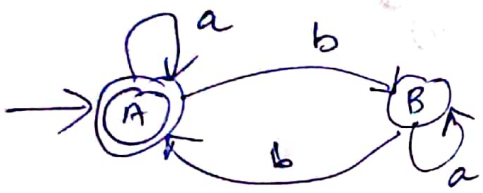
Scanned by CamScanner

Ex:
$S \rightarrow aA$
$A \rightarrow Bb$
$B \rightarrow C$
} Linear grammar

↑ Not a regular grammar.

Transition fn.	production rule
$\delta(q_1, a) \rightarrow q_1$	$q_1 \rightarrow aq_1$ or $q_1 \rightarrow q_1 a$
$\delta(q_1, b) \rightarrow q_2$	$q_1 \rightarrow b q_2$ or $q_1 \rightarrow q_2 b$
$q_f \in$ ~~Final~~ state	$q_f \rightarrow \epsilon$

Q)



Regular grammar (left linear)

$\delta(A, a) \rightarrow A$ $\quad A \rightarrow Aa$

$\delta(A, b) \rightarrow B$ $\quad A \rightarrow Bb$

$\delta(B, a) \rightarrow B$ $\quad B \rightarrow Ba$

$\delta(B, b) \rightarrow A$ $\quad B \rightarrow Ab$

$A \rightarrow \epsilon$

Context free grammar (CFG)

A grammar $G = (V, T, S, P)$ is said to be context free if all productions in P have the form $A \rightarrow x$.

where $A \in V$ and $x \in (V \cup T)^*$

→ A language L is said to be context free if and only if there is a CFG G such that $L = L(G)$.

Scanned by CamScanner

Q: write CFG for a L containing strings of length 2 over $\Sigma = \{a, b\}$.

$S \to aa \mid bb \mid ab \mid aa$

$S \to AA$

$A \to a \mid b$

Q: write CFG for $L = \{a^n \mid n \geq 0\}$

$S \to \epsilon \mid aS$

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

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

$S \to a \mid aS$

↓

Terminal

Q: L = set of strings of length atleast 2.

not regular

$S \to AAB$

$A \to a \mid b$

$B \to \epsilon \mid aB \mid bB \equiv B \to (a+b)^*$

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

$S \to (a|b)(a|b)B$

$\to (aa \mid bb \mid ab \mid ba)B$

$S \to aaB \mid bbB \mid abB \mid baB$

$B \to \epsilon \mid aB \mid bB$

} not regular.

Q: write CFG for $L = \{wcw^R \mid w \in (a,b)^*\}$

$S \to c \mid aSa \mid bSb \mid$

aca

bcb

$S \to aSa$

$S \to abSba$

$S \to abaSaba$

$S \to abacaba$.

derive abbcbba

$S \to aSa$

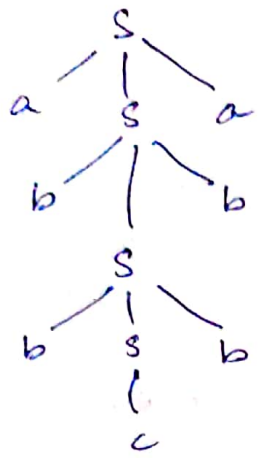
$S \to abSba$

$S \to abbSbba$

$S \to abbcbba$

Scanned by CamScanner

Derivation tree



Q: $L = \{ a^n b^n c^m \mid n, m \geq 1 \}$

Q: $L = \{ ab^n \mid n \geq 1 \} \cup \{ a^m b^m \mid m \geq 0 \}$

Q: $L = \{ a^n b^{2n} \mid n \geq 1 \}$

Q: $L = \{ a^n b^{2m} \mid n \neq m \}$

17 oct 19

Push down automata (PDA)

- Deterministic PDA — Deterministic context free (DCFL)
- Non deterministic PDA — Language: Non deterministic context free (NCFL)

PDA = FA + stack

$M = (Q, \Sigma, \delta, q_0, Q_F, Z_0, \Gamma)$ — Z_0: stack

Q = Set of states
Σ = Input symbol
δ = Transition function
q_0 = Initial state
Q_F = Set of final states
Z_0 = Bottom of stack
Γ = Stack alphabet

DPDA:

$\delta : Q \times \{\Sigma \cup \in\} \times \Gamma \to Q \times \Gamma$

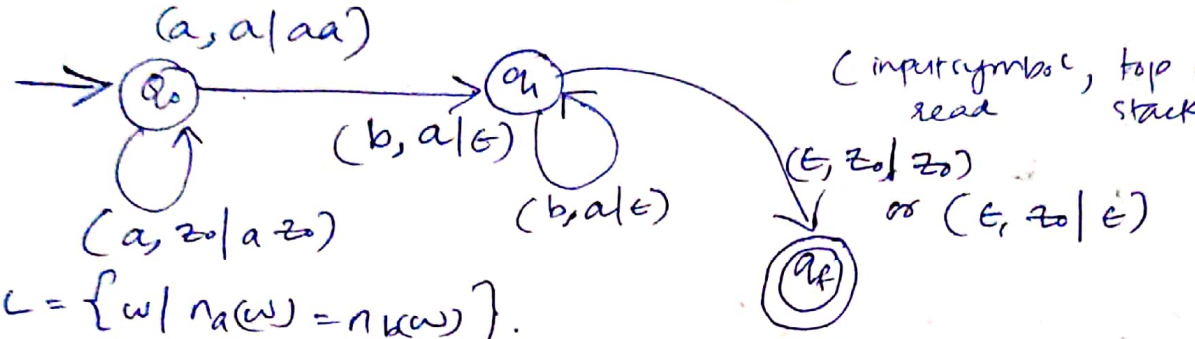
NPDA:

$\delta : Q \times \{\Sigma \cup \in\} \times \Gamma \to 2^{\{Q \times \Gamma\}}$

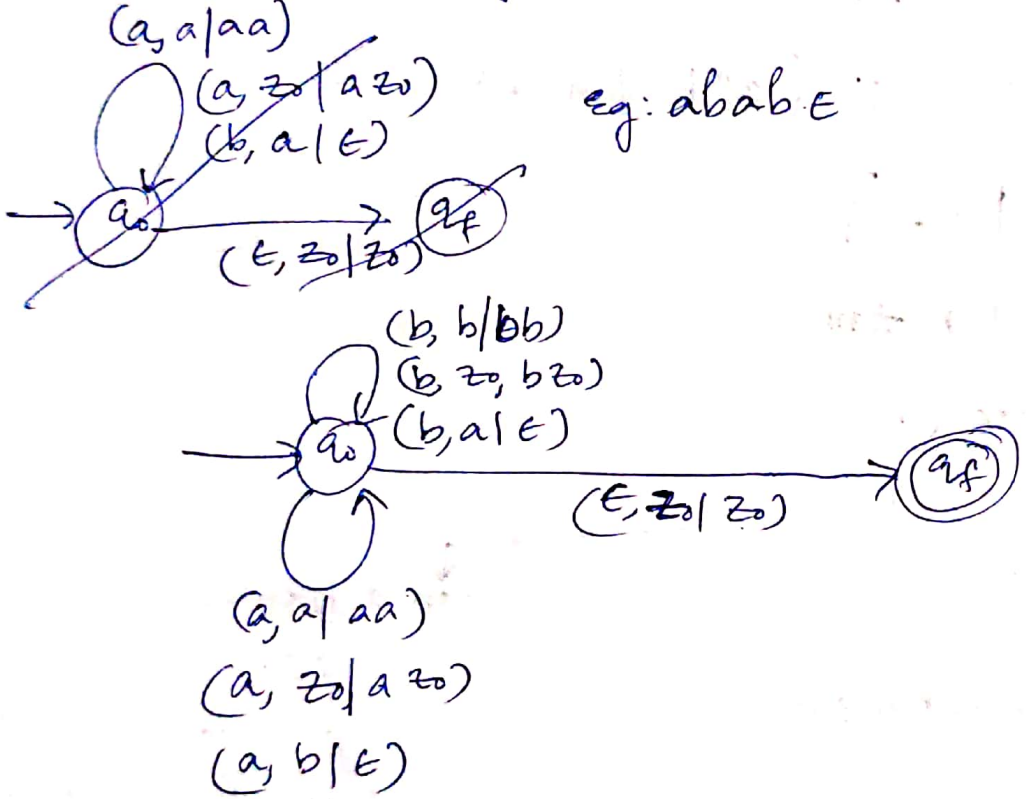
Scanned by CamScanner

δ: (current state, input symbol, symbol on stack top) → (next state, resultant symbol on stack)

Q: Design PDA for $L = \{a^n b^n \mid n \geq 1\}$ eg: aabb ∈



Q: $L = \{w \mid n_a(w) = n_b(w)\}$.



10: $L = \{a^n b^{2n} \mid n \geq 1\}$

① $L = \{a^n b^m \mid n \neq m\}$

② $L = \{w c w^R \mid w \in (a,b)^*\}$

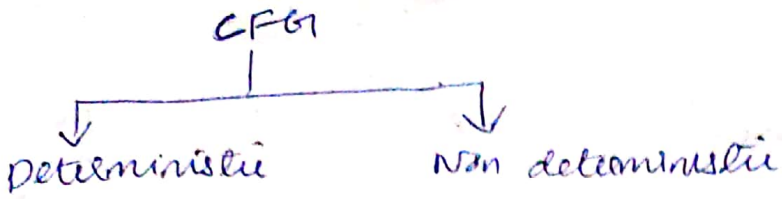
③ $L = \{a^n b^n \mid n \geq 1\} \cup \{a^m b^{2m} \mid m \geq 1\}$

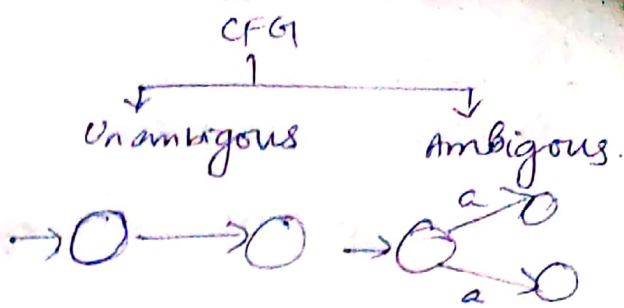
④ $L = \{a^{n+m} b^n c^m \mid n, m \geq 1\}$

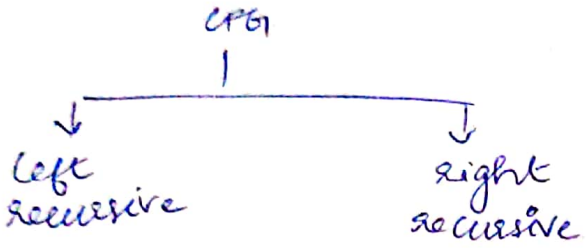
⑤ $L = \{a^m b^m c^n d^n \mid n, m \geq 1\}$

⑥ $L = \{a^i b^j \mid i \neq 2j+1\}$.

Scanned by CamScanner







Leftmost and rightmost derivations

→ A derivation is said to be leftmost if in each step the left most variable in sentential form is replaced.

→ In case rightmost variable is replaced, it is called rightmost derivation.

Eg: Consider a CFG

$S \rightarrow aAB$

$A \rightarrow bBb$

$B \rightarrow A \mid \lambda$ $(\lambda \equiv \epsilon)$

Consider the string abbbbb

* Leftmost derivations

$S \rightarrow aAB$

$abBbB$ $(A \rightarrow bBb)$

$abAbB$ $(B \rightarrow A)$

$abbBbbB$ $(A \rightarrow bBb)$

$abb\lambda bbB$ $(B \rightarrow \lambda)$

$abbbbb\lambda$ $(B \rightarrow \lambda)$

* Rightmost derivation

$S \rightarrow aAB$

aAA $(B \rightarrow A)$

$aAbBb$ $(A \rightarrow bBb)$

$aAbb$ $(B \rightarrow \epsilon)$

$abBbbb$ $(A \rightarrow B)$

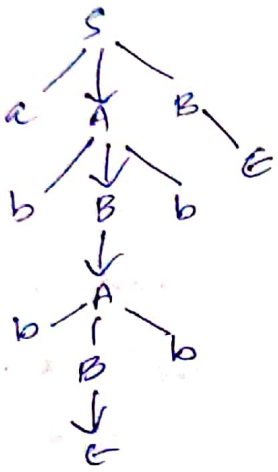
$abbbbb$ $(B \rightarrow \epsilon)$

Scanned by CamScanner

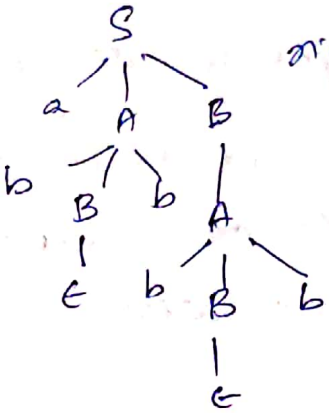
Derivation tree or Parse tree:

It is an ordered tree in which nodes are labelled with the left sides of productions and children of a node represent its right side.

q: Parse tree for "abbbb"



leftmost derivation



rightmost derivation tree

Ambigous grammar

A CFG, G is said to be ambigous if there exists some $w \in L(G)$ that has at least two distinct parse tree. (existance of two or more leftmost or rightmost derivations)

As we have different leftmost and rightmost derivations for "abbbb", this grammar is ambigous.

Assgn: Ranoval of ambiguity.

Simplifications of CFG and Normal forms

Null production

Any production of CFG of the form $A \to \lambda$.
A is called nullable variable.

$A \to B \equiv A \to \lambda$
$B \to \epsilon$
$C \to P$
* unit productio

* unreachable productions which are never reached

Scanned by CamScanner

Elimination of ε-production.

Procedure

① Find set of all nullable variables

② Write all remaining productions with and without all nullable variables

Eg: Eliminate ε-production

$S \to aSb \mid aAb$

$A \to \epsilon$

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

$S \to aSb$
$S \to ab$

replaced ε for A.

Eg: Removal of ε-production

Eg: $S \to ABaC$
$A \to BC$
$B \to b/\lambda$
$C \to D/\lambda$
$D \to d$

Rule ① : Make a set of nullable variables $\{B, C\}$

Since $A \to BC$ hence A will also be included in this set.

$\{A, B, C\}$

Rule ② :

$S \to ABaC$ (no substitution)

$S \to \underset{A=\epsilon}{BaC} \mid \underset{B=\epsilon}{AaC} \mid \underset{C=\epsilon}{ABa}$ (Substitute individually)

$S \to \underset{AB=\epsilon}{aC} \mid \underset{B,C=\epsilon}{Aa} \mid \underset{A,C=\epsilon}{Ba}$ (Substitute in groups of two)

$S \to a$ (Substitute 3 vars at a time)

$\therefore S \to ABaC \mid BaC \mid AaC \mid ABa \mid aC \mid Aa \mid Ba \mid a$

Scanned by CamScanner

for production

$A \rightarrow BC$

~~$A \rightarrow B/C$~~

$A \rightarrow BC/B/C$

$B \rightarrow b$

$C \rightarrow D$

$D \rightarrow d$

Q: $S \rightarrow AB$
$A \rightarrow aAA/\epsilon$
$B \rightarrow bBB/\epsilon$

$S \rightarrow AB/A/B$
$A \rightarrow aAA/aA/a$
$B \rightarrow bBB/bB/b$

Unit-production:

Ex: Remove unit productions

$S \rightarrow Aa/B$

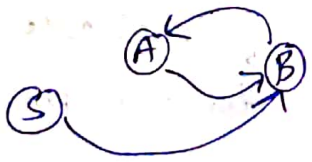
$B \rightarrow A/bb$

$A \rightarrow a/bc/B$

① Here unit productions are

$S \rightarrow B$
$B \rightarrow A$
$A \rightarrow B$

dependency graph



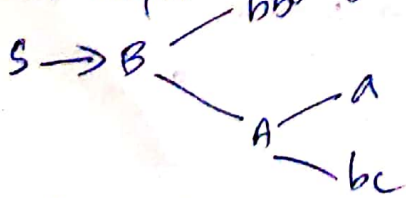
② write CFG with non unit production

$S \rightarrow Aa$

$B \rightarrow bb$

$A \rightarrow a/bc$

from dependency graph we have



If we remove B, we coose bb, a and bc

Scanned by CamScanner

$\therefore$ S → Aa | bb | a | bc

B → bb | a | bc

A → a | bc | bb | a | bc

repeating

$\therefore$ A → a | bc | bb

useless production
— does not produce string.

Removal of useless and unreachable variables.

* A production is useless if it involves any useless variable.

Q: Remove useless symbol.

S → as | A | C

A → a

B → aa

C → aCb

Rule ①: Identify the set of variables that can lead to terminal string

{ A, B, S }

S derives A

A derives a

$\therefore$ S can lead to terminal string.

S → as | A

A → a

B → aa

remove all productions involving variables that are not in this set.

Rule ②: Remove unreachable symbol

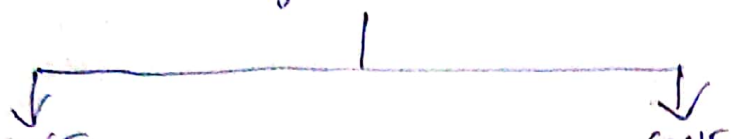
S → as | A

A → a

DFA minimization
Pumping Lemma

Scanned by CamScanner

Normal Form of CFG



CNF (Chomsky normal form)

GNF (Greibach Normal form)

CNF: CFG is in CNF if all productions are of the form

$A \to BC$

$A \to a$

where A, B, C are in V (set of non-terminals or variables)

$a \in T$ (set of terminals)

Ex of CFG in CNF

$S \to AS \mid a$

$A \to SA \mid b$

Theorem:

Any CFG $G = (V, T, S, P)$ with $\lambda \notin L(G)$ has an equivalent grammar $\hat{G} = (\hat{V}, \hat{T}, S, \hat{P})$ in CNF.

Advantages of CNF

1. Length of each production is restricted
2. Derivation tree or parse tree obtained from CNF is always binary
3. The no of steps required to derive a string of length $|w|$ is $(2|w|-1)$
4. It is easy to apply CYK membership also.

Scanned by CamScanner

Procedure to convert CFG to CNF

① Eliminate λ- productions, unit- productions, useless productions

② Eliminate terminals from RHS if they exist with other terminals or non-terminals

Ex: $X \rightarrow aY$

$\Downarrow$

$X \rightarrow AY$

$A \rightarrow a$

③ Eliminate RHS with more than two non-terminals

$S \rightarrow XYZP$

$\Downarrow$

$S \rightarrow AB$

$A \rightarrow XY$

$B \rightarrow ZP$

Q: Convert following CFG to CNF

① $S \rightarrow bA \mid aB$

$A \rightarrow bAA \mid as \mid a$

$B \rightarrow aBB \mid bS \mid b$

$S \rightarrow bA \mid aB$

$S \rightarrow PA \mid QB$

$P \rightarrow b$

$Q \rightarrow a$

$A \rightarrow PA' \mid QS \mid a$

$A' \rightarrow AA$

$B \rightarrow QB' \mid PS \mid b$

$B' \rightarrow BB$

② $S \rightarrow ABa$

$A \rightarrow aab$

$B \rightarrow Ac$

③ $S \rightarrow ASA \mid aB$

$A \rightarrow B \mid S$

$B \rightarrow b \mid \lambda$

$f \rightarrow abc$

GNF:

A CFG is said to be in GNF if all productions have the form $\boxed{A \rightarrow aX}$ where $a \in T$ and $x \in V^*$.

Scanned by CamScanner

Ex: $S \to a$ ✓

$S \to bSA$ ✓

$S \to Aa$ X

Q: $S \to aB \mid BC$ ← problem

$A \to aB \mid bA \mid c$

$B \to bB \mid cC \mid b$

$C \to c$

$S \to aB \mid bBB \mid cCC \mid bC$

Convert to GNF form

Membership algorithm for CFG.

CYK.

→ CYK is an universal algo to check whether a string is a member of L(G) or not.

→ CYK is only applied if CFG is in CNF.

$$V_{ij} = \bigcup_{k \in (i, i+1, \ldots j-1)} (A : A \to BC \text{ with } B \in V_{ik}, C \in V_{k+1,j})$$

Ex: check whether a "baaba" is a valid member of the following CFG

$S \to AB \mid BC$

$A \to BA \mid a$

$B \to CC \mid b$

$C \to AB \mid a$

① 11

b a a b a

↑

B is directly producing b

② 22

b a a b a

A, C are directly producing a

Scanned by CamScanner

	5	4	3	2	1
1	A,S,C	ϕ	ϕ	A,S	B
2	S,A,C	B	B	A,C	
3	B	S,C	A,C		
4	A,S	B			
5	A,C				

③ 12

baaba

$12 \equiv (11)\,\&\,(22)$

$= (B)(A,C)$

$= \underline{BA}, \underline{BC}$

↓

A directly produces B

S " " BC

④ 23

baaba

$23 \equiv (22)(33)$

$= (A,C)(A,C)$

$= AA, AC, CA, CC$

X X X B

⑤ 34

baaba

$3,4 \equiv (33)(44)$

$= (AC)(B)$

$= AB, CB$

↓ ↓

S,C X

⑥ 45

baaba

$45 \equiv (44)(55)$

$= (B)(AC)$

$= BA, BC$

↓ ↓

A S

1,3

baaba

$1,3 = \begin{cases} (11)(23) & BB \\ & X \\ (12)(33) & (A,S)(A,C) \end{cases}$

$= AA, AC, SA, SC$

X X X X

24

$24 = \begin{cases} (23)(44) = (B)(B) \to \phi \\ (22)(3,4) = (AC)(SC) \end{cases}$

$= AS, AC, CS, CC$

X X X B

35

$35 = \begin{cases} (33)(45) = (AC)(AS) \\ \quad = AA, AS, CA, CS \\ \quad \phi \;\; \phi \;\; \phi \;\; \phi \\ (34)(55) \end{cases}$

$= (S,C)(AC)$

$= SA, SC, CA, CC$

X X X B

Scanned by CamScanner

14

14 = { (11)(24) = BB, ϕ
(12)(34) = (AS)(SC) = AS, AC, SS, SC (P P P P)
(13)(44) = ~~(ϕ)(CAS)~~ (ϕ)(B) = B, ϕ

25 = { (22)(35) = (AC)(B) ~~= (AB)(CB)~~ (S, L) ×
(23)(45) = (B)(A,S) = BA, BS (A ×)
(24)(55) = (B)(A,C) = BA, BC (A ×)

15 = { 11 25
12 35
13 45
14 55

We got s in 15, so "baaba" is a member of this CFG.

4 substrings are in this CFG as S is present in 4 cells.

Time complexity = 1+2+3+....n

$= \frac{n(n+1)}{2} \times (n-1)$ ~~$= O(n^2)$~~ =

$= O(n^3)$

28/9/19.

Pumping lemma.

Strings of the form $0^n 1^n$ is not regular.

→ If a language L is regular, then there is a positive integer p such that for any string $s \in L$ of length atleast p, we can split w as $w = xyz$ such that

Scanned by CamScanner

① $|y| > 0$ (y is not empty, x and z can be empty)

② $xy^i z \in L$ for $i = 0, 1, 2, \ldots$

③ $|xy| \leq p$

y can be pumped as many times as we want, string still remains in L.

eg: L = string with even no of ones over $\{0, 1\}$.

$x \underbrace{00}_{y} \underbrace{1}_{z} \in L$

$\underbrace{1}_{x} \underbrace{1 \; 1}_{y} \underbrace{1}_{z} \in L \qquad xy^0 z = 11 \in L$

$w = 11$

$x = \epsilon \quad y = 11 \quad z = \epsilon$

Proof:

Since $|w| \geq p$, and no of states $= p$, there are atleast $p+1$ states in the sequence $q_0 \cdots q_1 - q_2 - q_3 \cdots q_n - q_{accept}$.

So at least one state appears twice in this sequence.

Let q_j be the first state in this sequence that repeats

the string here is y

① $|y| > 0$ (y is not empty)

② $xy^i z \in L$ (obvious from the DFA)

③ $|xy| \leq p$ (because with in first p symbols there will be one state repeated)

Scanned by CamScanner

Q: Prove that 0^n1^n is not regular.

Assume that L is regular.

So $\forall w \in L, |w| \geq p$ such that $w = xyz$ such that

(1) $|y| > 0$

(2) $xy^iz \in L$

(3) $|xy| \leq p$

Let $w = 0^p1^p$ ($\in L$ so $|w| \geq p$)

y can't contain 0's or 1's alone

Let $y = 01$, repeating y removes w from the language

$w = 0^p1^p$ can't be written as $w = xyz$, y is non empty and $xy^iz \in L$

Case 1: y consists of 0's alone (or y consists of 1's alone)

$xz \notin L$, $xy^2z \notin L$ or different no of zeros and ones

($i = 0$) ($i = 2, 3, \ldots$)

Case 2: y consists of equal no of zeros and ones

$xz \in L$ but $xy^2z \notin L$

$\Rightarrow$ Pumping lemma does not hold. So L is not regular.

Q: Prove that $w.1's_0(w) = s_1(w)$ is not regular.

$w = 0^p1^p \in L$

can $y = 01$ be pumped

No, because $|xy| \geq p$, violates condition 3

(1) Strings that are palindromes.

(2) ww^R

(3) ww

(4) $w\#w$

Scanned by CamScanner