Type 3 ⊆ Type 2 ⊆ Type 1 ⊆ Type 0.

Module - 2.

Regular Expressions.

- The regular expression can define exactly the same languages that the various forms of automata describe i.e. the regular language.
- The regular expression offer something that the automata do not i.e. a declarative way to express the strings, that we want to accept.

e.g. a^* → any occurences of 'a'.

→ Operators of regular expression:

i) Union (∪).

$L_1 \cup L_2$ → set of strings that are either in L_1 or L_2 or both.

ii) Concatenation (·)

$L_1 \cdot L_2$ → set of strings that can be formed by taking any string in L_1 and concatenating it with any string in L_2.

iii) Kleene closure (*)

L^* → The closure of a language L denoted by L^* is the set of those strings that can be formed by taking any number of strings from L possibly with

repetition and concatenating all of them.
L^* is the infinite union and can be represented as

$$L^* = \bigcup_{i \geq} L^i \quad , \quad i \geq 0$$

iv) Positive closure (+).

L^+ → Positive closure of a language L denoted by L^+ is the set of those strings that can be formed by taking any no. of strings from L possibly with repetition and concatenating all of them, excluding the null string i.e. ϵ.

$$L^+ = \bigcup_{i \geq} L^i \quad , \quad i > 0.$$

e.g. $L_1 = \{ 001, 10, 111 \}$.

$L_2 = \{ \epsilon, 001 \}$.

$L_1 \cup L_2 = \{ \epsilon, 001, 10, 111 \}$

$L_1 . L_2 = \{ 001, 001001, 10, 10001, 111001, 111 \}$

$L_1^* = \{ 001, 10, 111, \epsilon, 001001, 101010, 111111, \ldots \}$

$L^+ = \{ 001, 10, 111, 001001, 1010, 111111, \ldots \}$

→ Defⁿ of Regular Expression:

The set of regular expression is defined by the following rules:-

i) Every letter of Σ can be made into a regular expression.
The ϵ is also a regular expression.

ii) If R_1 and R_2 are regular expressions, then,

i. (r_1) iii. $r_1 + r_2$ v. r_1^+

ii. $r_1 r_2$ iv. r_1^*

are also regular expression.

Q) Write a regular expression over the input alphabet $\Sigma = \{a, b, c\}$ containing atleast one 'a' ~~and~~ or one 'b'.

Sol) $L = \{a, b, ab, ac, abc, aab, aabc \ldots\}$

$R = (a+b+c)^* \quad a(a+b+c)^* \quad b(a+b+c)^*$

$\epsilon \cdot a(a+b+c)^* b(a+b+c)^*$

~~$RE = a(a+b+c)^* b(a+b+c)^*$~~

~~$RE = (a+b)(a+b)^* (a+b)^*$~~

Q) Regular expression for the string 0 and 1 whose 10th symbol from the right is 1.

~~$(0+1)(0+1)^*$~~

~~$(0+1)(0+1)^*$~~

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

8/3/18

Note:

(i) If $L = \{\}$ (Empty set)
Then $RE = \phi$.

(ii) If $L = \{\epsilon\}$
Then $RE = \epsilon$

(iii) If $L = \{a\}$
Then $RE = a$

Language	RE
$\{\}$	ϕ
$\{\epsilon\}$	ϵ
$\{a\}$	a
$\{\epsilon, a, aa, aaa \ldots\}$	a^*
$\{a, aa, aaa, \ldots\}$	a^+
$\{\epsilon, a, b, ab, aa, \ldots\}$	$(a+b)^*$

Q) Write a regular expression for string whose length is exactly 2.

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

Ans $L = \{aa, ab, ba, bb\}$. $+ \rightarrow$ Union / or

$RE = aa + ab + ba + bb$

$= a(a+b) + b(a+b)$.

$= (a+b)(a+b)$

Even length, $RE = \left((a+b)(a+b)\right)^*$

odd length, $RE = \left((a+b)(a+b)\right)^* (a+b)$

Q) RE for string length atleast 2.

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

Ans $RE = (a+b)(a+b).(a+b)^*$

Q) RE for string divisible by 3.

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

Ans $RE = \left((a+b)(a+b)(a+b)\right)^*$ $\rightarrow$ $* = 0 \Rightarrow \epsilon$ is included.

Q) RE for a string of atmost length atmost 2.

Ans) ~~RE = (a+b+ϵ)(a+b)(a+b)~~

RE = (a+b+ϵ) (a+b+ϵ).

Q) RE for string starting with ab.

Ans) RE = ab(a+b)*

Q) RE for string ending with 3 consecutive 1 -

Ans) RE = (0+1)* 111

Q) RE for atleast 1 a & 1 b

~~RE = ab(a+b)* + (a+b)* ab + ba(a+b)* + (a+b)* ba~~

= (a+b)* ab (a+b)* + (a+b)* ba (a+b)*

~~a(a+b)*b + b(a+b)*a~~

RE

= (a+b)* a (a+b)* b (a+b)* + (a+b)* b (a+b)* a (a+b)*

15/3/18

→ Identity rules for regular expressions :-

R → regular expression

i) ϕ + R = R + ϕ = R

ii) ϕ . R = R . ϕ = ϕ .

iii) ϵ . R = R . ϵ = R .

iv) ϵ* = ϵ .

v) ϕ* = ϵ

vi) ϵ + RR* = R*R + ϵ = R* .

vii) (P+Q)* = (P*Q*)* = (P* + Q*)*

viii) R+R = R .

ix) R*R* = R*

x) R*R = RR* = R+ .

xi) (R*)* = R*

xii) (PQ)* P = P(QP)* .

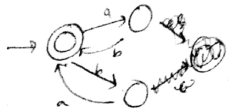
xiii) (P+Q)R = RP+RQ = PR+QR = R(P+Q) .

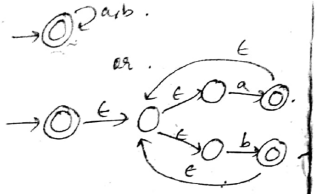
→ Conversion of RE to FA :

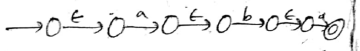
	RE	FA
		→ ○
i)	ϕ	→ ◎ or → ○ —ϵ→ ◎
ii)	ϵ	→ ○ —a→ ◎
iii)	a	→ ○ —ϵ→ ○ —a→ ○ —ϵ→ ◎ , ϵ back to ○
iv)	a*	NFA
v)	a.b	→ ○ —a→ ○ —b→ ◎
vi)	ab*	→ ○ —a→ ◎ , ◎ —b→ ◎

RE | FA

vii) $(ab)^*$ 

viii) $(ab + ba)^*$
$= \{\epsilon, ab, ba, abba, abab, baba, abbaabba, ababba, \ldots\}$ 

ix) $(a+b)^*$
$= \{\epsilon, a, b, ab, aab, abb, \ldots\}$ 

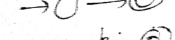
x) aba 

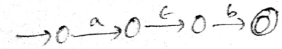
Note: If we have to convert RE to DFA, then ε-NFA to NFA to DFA. (all steps).

Q) RE to FA.

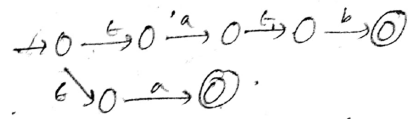
RE = $(ab \cup a)^*$ = $\{\epsilon, ab, a, abab, aa, aba, abaaba, ababaa, \ldots\}$.

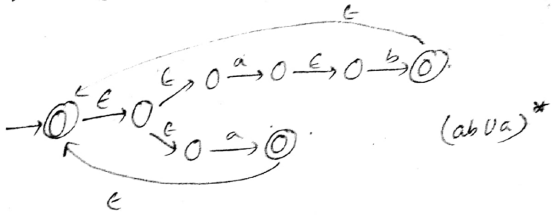
i) accepts a only

ii)  accepts b only.

iii)  accepts ab.

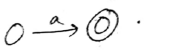
(To accept ab, the final state in (ii) becomes non-final & ε-transition occurs)

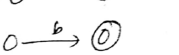
iv)  $(ab \cup a)$

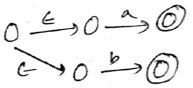
v)  $(ab \cup a)^*$

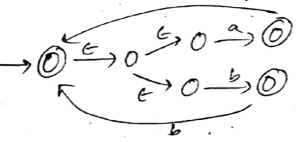
Q) $(a \cup b)^* aba$

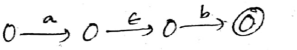
RE = $(a \cup b)^* aba$ = $\{\epsilon, a, b, ab, aab, abb, ba, \ldots\}.\{aba\}$
= $\{aba, aaba, baba, ababa, aababa, \ldots\}$

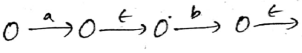
i)  accepts a only.

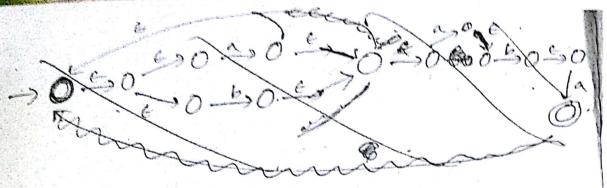
ii)  accepts b only.

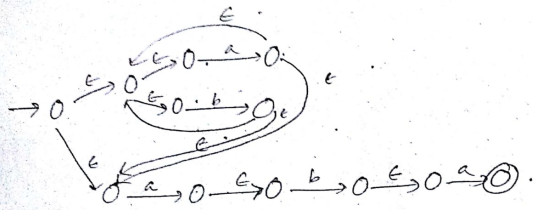
iii)  accepts $(a \cup b)$

iv)  accepts $(a \cup b)^*$

v)  accepts ab.

vi)  accepts aba.





Q) RE = $(0+1)^*1(0+1)$

→ FA to RE

i) Arden's Theorem.

ii) GNFA or State Elimination Method.

i) Arden's Theorem: (for DFA (not applicable for ε-NFA))

Let, P, Q be two REs over Σ and P does not contain ε, then

$$R = Q + RP$$

has a unique soln i.e. $R = QP^*$.

Proof: To prove that $R = QP^*$ is a soln

$$R = Q + RP$$
$$= Q + QP^*P$$
$$= Q(\varepsilon + P^*P)$$
$$= QP^* \quad \longrightarrow \varepsilon + R^*R = R^*$$

To prove that $R = QP^*$ is the only and unique solution.

$$R = Q + RP$$
$$= Q + (Q+RP)P$$
$$= Q + QP + RP^2$$
$$= Q + QP + (Q+RP)P^2$$
$$= Q + QP + QP^2 + \ldots + QP^n + RP^{n+1}$$
$$= Q + QP + QP^2 + \ldots + QP^n + QP^*P^{n+1} \quad (\because R = QP^*)$$

$\Rightarrow R = Q(\epsilon + P + P^2 + \cdots + P^n + P^* P^{n+1})$.

$= Q P^*$.

Steps to ~~solve~~ convert DFA to RE:

i) Create the equ^{n}s of the below form

$q_n = q_1 w_{1n} + q_2 w_{2n} + \cdots + q_m w_{mn}$.

for all the states of the DFA having n states with initial state q_1.

ii) Solve the above equ^n to get the equ^n for final states in terms of w_{ij}.

Assumptions:

i) The transition diagram should not have ϵ-transitions.

ii) It must have (only one) single initial state.

iii) The vertices are $q_1, q_2, \cdots q_n$ where q_i are final states

iv) w_{ij} denotes the regular expression (RE) representing the sets of labels of edges from q_i to q_j.

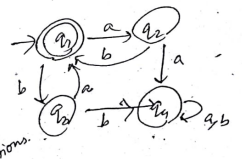
* If $w_{ij} = \phi$ if there is no edge.

$q_1 = q_1 w_{11} + q_2 w_{21} + \cdots + q_n w_{n1} + \epsilon$ → for initial state ϵ must be added (due to the presence of incoming edge)

$q_2 = q_1 w_{12} + q_2 w_{22} + \cdots + q_n w_{n2}$

$q_n = q_1 w_{1n} + q_2 w_{2n} + \cdots + q_n w_{nn}$

eg: DFA to RE using Arden's theorem



DFA

incoming transitions.

$q_1 = \epsilon + q_2 b + q_3 a$.

$q_2 = q_1 a$

$q_3 = q_1 b$

$q_4 = q_2 a + q_3 b + q_4 a + q_4 b$.

now, $q_1 = \epsilon + q_2 b + q_3 a$.

$= \epsilon + (q_1 a) b + (q_1 b) a$

$= \epsilon + q_1 ab + q_1 ba$.

$= \epsilon + q_1 (ab + ba)$

Acc. to Arden's theorem,

$R = Q + RP$.

$q_1 = \epsilon + q_1(ab+ba)$. — (A)

Comparing (A) with $R = Q + RP$.

$R = q_1$

$Q = \epsilon$

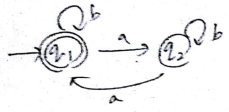
$P = ab+ba$

So, soln is $R = QP^*$

$\Rightarrow q_1 = \epsilon(ab+ba)^*$

$\Rightarrow \boxed{q_1 = (ab+ba)^*}$.

8)



$q_1 = \epsilon + q_1 b + q_2 a$.

$q_2 = q_1 a + q_2 b$.

Now, $q_1 = \epsilon + q_1 b + q_2 a$

$= \epsilon + q_1 b + (q_1 a + q_2 b)\, a$.

$= \epsilon + q_1 b + q_1 aa + q_2 ba$.

$= \epsilon + q_1 b + q_1 aa + (q_1 a + q_2 b)\, ba$.

$= \epsilon + q_1 b + q_1 aa + q_1 aba + q_2 bba$.

$= \epsilon + q_1 b + q_1 aa + q_1 aba + (q_1 a + q_2 b)\, bba$

$= \epsilon + q_1 b + q_1 aa + q_1 abba + q_1 abbba + q_2 bbba$

$= \epsilon + q_1 (b + aa + aba)$.

* $q_2 = q_1 a + q_2 b$ $\qquad R = Q + RP$

$R = q_2$

$Q = q_1 a$.

$P = b$ $\qquad R = QP^*$

$\boxed{q_2 = q_1 ab^*}$

final state.

$q_1 = \epsilon + q_1 b + (q_1 ab^*)\, a$.

$= \epsilon + q_1 b + q_1 ab^* a$.

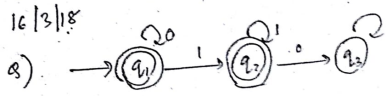
$= \epsilon + q_1 (b + ab^* a)$

$R = q_1$, $Q = \epsilon$, $P = b + ab^* a$.

$q_1 = \epsilon (b + ab^* a)^*$

16/3/18

8)



$q_1 = \epsilon + q_1 0$.

$q_2 = q_1 1 + q_2 1$.

$q_3 = q_2 0 + q_3 0 + q_3 1$.

$q_1 = \epsilon + q_1 0$ $\qquad (R = Q + RP)$.

$\Rightarrow q_1 = \epsilon 0^* = 0^*$. $\qquad (R = QP^*)$

(1)

$q_2 = q_1 1 + q_2 1$ $(R = q + Rp)$

$\Rightarrow q_2 = q_1 1 1^*$ $(R = qp^*)$ — (2)

$q_3 = q_2 0 + q_3 0 + q_3 1$

$= q_2 0 + q_3 (0+1)$ $(R = q + Rp)$

$\Rightarrow q_3 = q_2 0 (0+1)^*$ $(R = qp^*)$ — (3)

From eqn (1) & (2) & (3).

$q_1 = 0^*$

$q_2 = (0^* 1) 1^*$

~~$q_3 = 0^* 1 1^* 0 (0+1)^*$~~

$RE = 0^* \cup (0^* 1) 1^*$ { union of final states.

ii) GNFA (Generalised NFA):

- It is a variation of NFA where each transition is labelled with any RE.
- GNFA can be defined as a 5 tuple. $(q_0, Q, \Sigma, F, \delta)$

$\delta : (Q - \{F\}) \times (Q - \{q_0\}) \longrightarrow RE$:

FA which is obtained is in form of q_0 and F along with regular expression.

Conditions:

i) There are no incoming transition to the initial state.

ii) There should be only 1 final state having only incoming transitions and no outgoing transitions.

iii) The initial and the final states should be separate.

iv) Each state has a transition to itself i.e. except for the initial and final states, all other states are connected to the rest states via a transition.

Steps to convert FA (DFA, NFA, ε-NFA etc.) to RE:

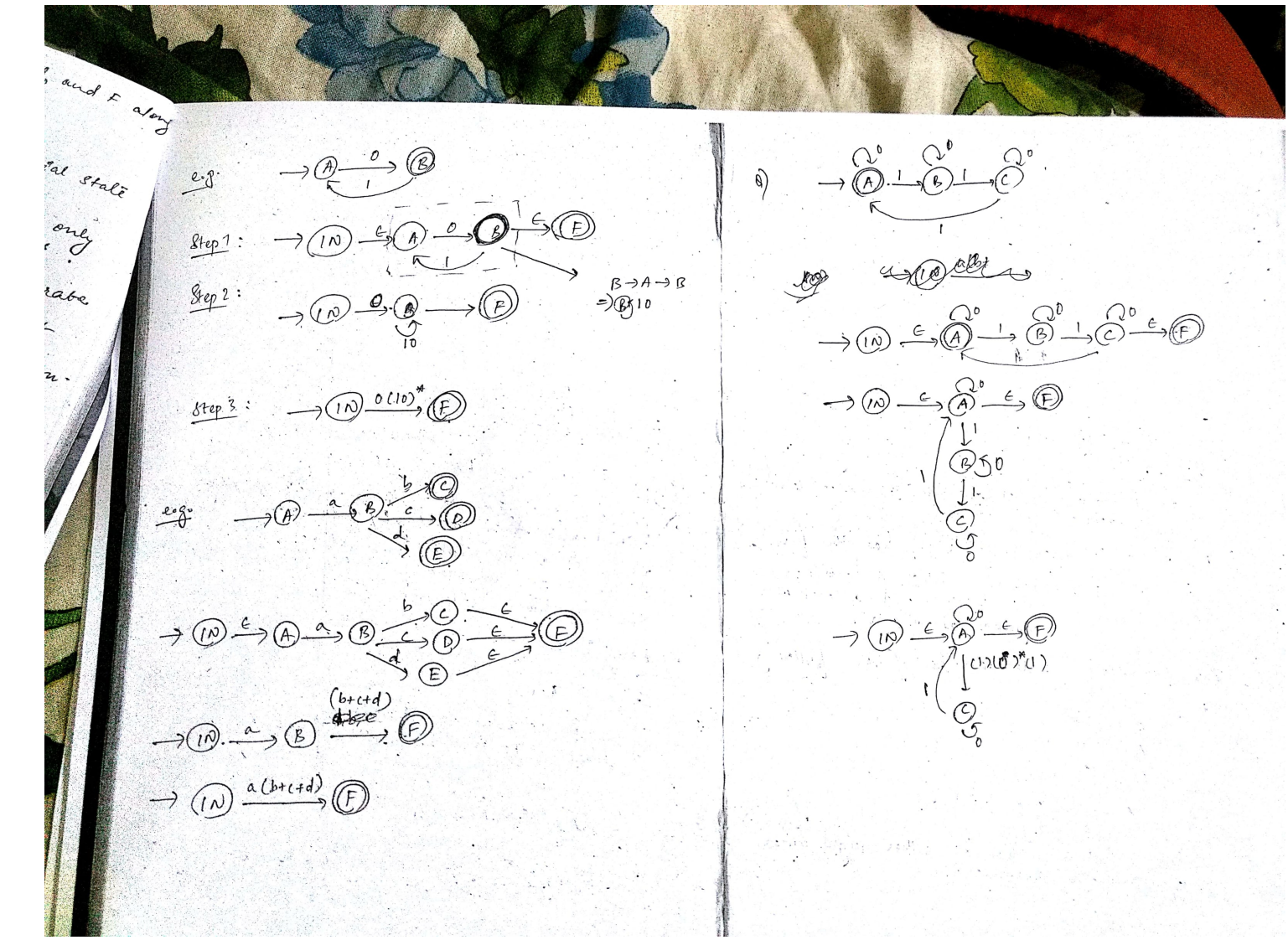
i) Convert the FA to GNFA ~~where~~ by adding new initial and final states, by adding ε-transitions. In case of multiple final states, add a new final states by adding ε-transition and make the old final state, non-final.

ii) Remove all the states one by one in any order until we have only the initial and final states.

The RE labelled on the transition from initial to final state is the output / final RE.

e.g. → (A) —a→ (B) —c→ ((C)) , (B) —b→ (B) (No multiple final state)

→ (A) —ab^*c→ ((C)) (Remove states until we have only the initial & final states)



e.g.

Step 1 :

Step 2 :

$B \to A \to B$

$\Rightarrow B10$

Step 3 :

e.g.

$a(b+c+d)$

a)

$(1)(0)^*(1)$

17/8/18

• Regular Grammars:

- Type III grammar.
- In TOC, regular grammar is the formal grammar that describes the regular language.
- A regular grammar G can be formally written as a 4 tuple (V, T, S, P), where,

  $V \rightarrow$ set of variables or non-terminal symbols.
  $T \rightarrow$ set of terminal symbols.
  $S \rightarrow$ start symbol.
  $S \in V$.
  $P \rightarrow$ Production rules for terminal and non-terminal symbols.

- A production / grammar rule has the form,

  $\alpha \Rightarrow \beta$, where α, β are strings on $V \cup T$ and atleast one symbol of $\alpha \in V$.

- The production rule can be of the following form,

  $A \Rightarrow aB$.
  $A \Rightarrow a$
  $A \Rightarrow \epsilon$

  where, $A, B \in V$ and $a \in T$.

- A regular grammar can be either right linear (regular) or left linear (regular) grammar.

eg $S \Rightarrow aB$.
$S \Rightarrow a\beta$.
$B \Rightarrow b$

Capital letters → non-terminal
small letters → terminal

$S \rightarrow$ start symbol.
$a, b \rightarrow$ terminal symbol
$B, S \rightarrow V$

Right linear (RLG) / regular grammar is a formal grammar (V, T, S, P) such that the production rules are of the following form:

$B \Rightarrow a$
$B \Rightarrow aC$.
$B \Rightarrow \epsilon$.

eg. $S \Rightarrow aC$
$S \Rightarrow bA$.
$A \Rightarrow \epsilon$
$A \Rightarrow cA$.

In RLG, the non-terminal symbols occur on the right side.
In this case, $C \in V$.

Left linear (LRG) / regular grammar is a formal grammar (V, T, S, P) such that the production rule is of the following form

$A \Rightarrow a$
$A \Rightarrow Ba$
$A \Rightarrow \epsilon$

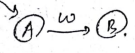
e.g. $A \Rightarrow Aa$.
$A \Rightarrow Bb$
$A \Rightarrow b$

In LRG, the non-terminal symbols occur on the left side.
In this case, $B \in V$.

Extended regular grammars :

i) Extended right regular grammar.
It is of the following form :

$A \Rightarrow a$
$A \Rightarrow wB$
$A \Rightarrow \epsilon$



$B, A \in V$
$a \in T$
$w \in T^*$.
string.

ii) Extended left regular grammar :
It is of the following form.

$A \Rightarrow a$
$A \Rightarrow Bw$.
$A \Rightarrow \epsilon$ (This means it is the final state (A))

$B, A \in V$
$a \in T$
$w \in T^*$.

** $A \Rightarrow BwB$ (Combination of left and right regular grammar. Not necessarily a regular grammar)

Q) Regular Expression vs Regular Grammar vs Regular Language.

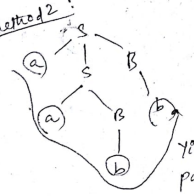
→ Derivation from regular grammars.
(Language generated by the grammar)
regular

Derivation
i) left
ii) right.

eg. $S \Rightarrow aSB$
$S \Rightarrow aB$
$B \Rightarrow b$.

Method 1 :
$S \Rightarrow aSB$
$\Rightarrow aaBB$
$\Rightarrow aabb$
} Sentential form.
string.

Method 2 :



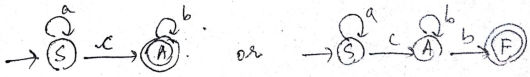
Yield of parse tree

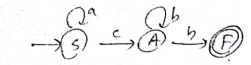
→ RLG to LLG Conversion :

Step 1: Construct the FA for the given RLG.
Step 2: Reverse the FA i.e. Final ↔ initial
Step 3: Construct grammar for the reversed FA.
Step 4: Reverse the grammar to obtain the LLG

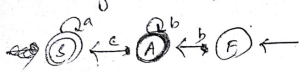
e.g. $S \Rightarrow aS$.
$S \Rightarrow cA$
$A \Rightarrow bA$
$A \Rightarrow b$

$S \Rightarrow aS$.
$S \Rightarrow acA$.
$S \Rightarrow acbA$.
$S \Rightarrow acbb$





Reversing the FA,



Constructing the grammar,

putting $S = \epsilon$.

$S \Rightarrow aS/a$
$A \Rightarrow cS/\epsilon$
$A \Rightarrow bA$
$F \Rightarrow bA$

or

$S \Rightarrow aS$
$A \Rightarrow cS$
$A \Rightarrow bA$
$F \Rightarrow bA$
$S \Rightarrow \epsilon$ ✓

Reversing the grammar,

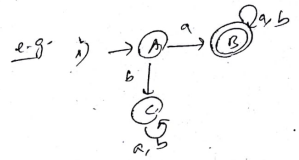
$F \Rightarrow Ab$
$A \Rightarrow Ab$
$A \Rightarrow Sc/c$
$S \Rightarrow Sa/a$. (Answer)

Reversal of finite automata:

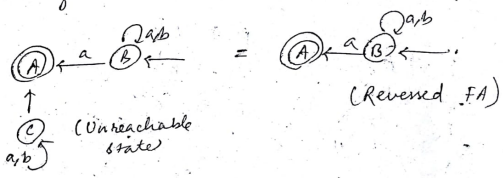
i) Make final state as initial state and vice-versa.
ii) Remove all the unreachable states from the initial state for the modified FA.
iii) Reverse the transition.

* For multiple final states, add ϵ.

e.g. i)



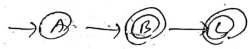
Reversing the FA



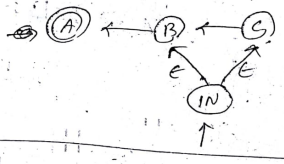
(Unreachable state)

(Reversed FA)

ii)



Reversing the FA

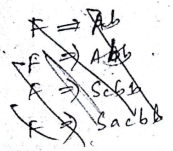


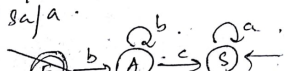
$F \Rightarrow Ab$
$F \Rightarrow Abb$.
$F \Rightarrow Scbb$.
$F \Rightarrow Sacbb$.
$F \Rightarrow acbb$
$(S = \epsilon)$

→ LLG to RLG:

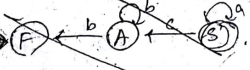
Step 1: Reverse the given grammar.
Step 2: Construct the FA.
Step 3: Reverse the FA.
Step 4: Derive the grammar for the FA obtained.

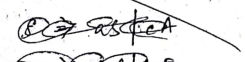
a) F ⇒ Ab
A ⇒ Ab
A ⇒ Sc/c
S ⇒ Sa/a.

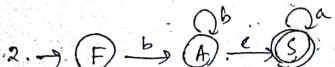


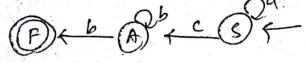


Ans 1. F ⇒ bA
A ⇒ bA
A ⇒ cS/c
S ⇒ as/a

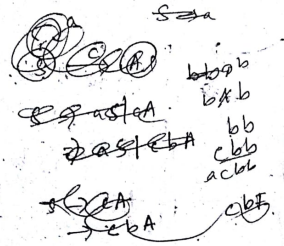




2. 

3. 

4. S ⇒ aS/cA
A ⇒ bA/bF
F ⇒ ∈



22/3/18

→ FA to RG:

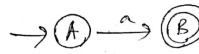
i) FA to RLG
ii) FA to LLG.

Steps for conversion of FA to RLG

1. Begin the process with the start state i.e. the start state will be the start symbol of the grammar.
2. Write the production rule as the O/P followed by the state to which the transition is going.
3. To end the derivation i.e. for the final state, add ∈, or add additional productions to provide termination of derivation.

Steps for conversion of FA to LRG

1. Reverse the given FA.
2. Find the RLG for the reversed FA.
3. Reverse the RLG to obtain to LLG.

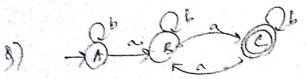
e.g. 

FA to RLG: A ⇒ aB.
B ⇒ aB | bB | ∈.

or

A ⇒ aB.
B ⇒ a B | bB | a | b.

8)

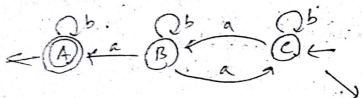


FA to RLG

$A \Rightarrow bA$
$A \Rightarrow aB.$
$B \Rightarrow bB$
$B \Rightarrow aC.$
$C \Rightarrow bC | \epsilon$
$C \Rightarrow aB | \epsilon$

$A \Rightarrow bA | aB$
$B \Rightarrow bB | aC.$
$C \Rightarrow bc | aB | \epsilon.$

FA to LLG



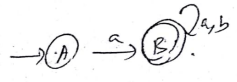
start from initial to generate the grammar

$A \Rightarrow bA | \epsilon$
$B \Rightarrow aA | bB | aC.$
$C \Rightarrow bC | aB.$

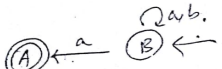
Reversing grammar to obtain LLG.

$A \rightarrow Ab | \epsilon.$
$B \Rightarrow Aa | Bb | Ca$
$C \Rightarrow Cb | Ba.$

8)



$A \rightarrow aB$
$B \rightarrow aB | bB | \epsilon.$



$B \Rightarrow aB | bB | aA$
$A \Rightarrow \epsilon.$

Reversing the grammar to obtain LLG,

$B \Rightarrow Ba | bB | aA.$
$A \rightarrow \epsilon.$

⇒ RG to FA.

RLG to FA

1. Initial state will be the first production state or the start symbol.
2. For every non-terminal, go to terminal symbol as the transition.
3. To determine final state, analyse those states that end up with terminals or check for ϵ-transitions.

LLG to FA

1. Reverse the given LLG. → RLG is obtained
2. Construct the FA for the given RLG.
3. Construct the FA for the RLG.
4. Reverse the FA, which will be the desired FA.
5. The 1st production symbol will be the Initial State of FA.
6. Analyse those states that end up with terminal symbols or ε to determine the final state.

29/9/18

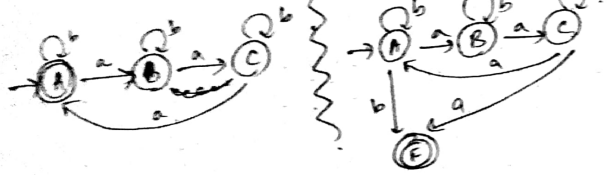
a) $A \to aB \mid bA \mid b$ ← terminal symbols

$B \to aC \mid bB$

$C \to aA \mid bC \mid a$

Putting $A = \varepsilon$, we obtain b/a.

Sol)

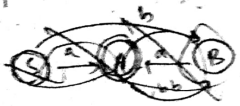


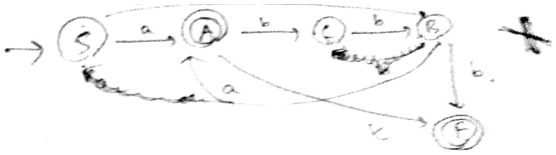
b) $S \to aA \mid bB$

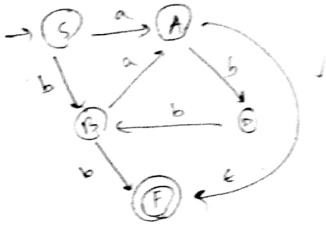
$B \to aA \mid b$

$A \to bbB \mid \varepsilon$

Sol)







c) $A \to Ba \mid Ab \mid b$

$B \to Ca \mid Bb$

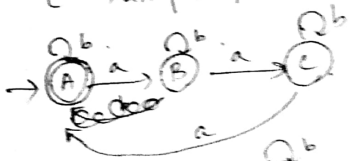
$C \to Aa \mid Cb \mid a$

Sol)

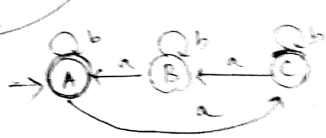
$A \to aB \mid bA \mid b$

$B \to aC \mid bB$

$C \to aA \mid bC \mid a$



Reversing



Q) Verify whether the given grammar G1 and G2 denote the same language

G1

A → 0A/1E
B → 0A/1F/ε
C → 0C/1A
D → 0A/1D/ε
E → 0C/1A
F → 0A/1B/ε

DFA

G2

X → 0Y/0/1Z
Y → 0X/1Y/1
Z → 0Z/1X

DFA

If DFAs obtained are equal, then grammar denote the same language.

	RE	RLG	LLG
	ε	S → ε	S → ε
	(e+f)	S → e/f	S → e/f
	(ef)	S → eA A → f	S → Af A → e
	e*	S → eS/ε	S → se/ε
	e+	S → eS/e	S → se/e
	(e+f)*	S → eS/fS/ε	S → se/sf/ε
	(e·f)*	S → eS/fS/e/f	S → se/sf/e/f
8) * closure on concatenation	(ef)*	S → eA/ε A → fs	S → Af A → se/e
9) + closure on concatenation	(e·f)+	S → eA A → fs/f	S → Af A → se/e

S → start state
e, f → terminal symbols
A → non-terminal symbol

Q) Convert the grammar that accepts set of all strings of length 2.

L = {aa, ab, ba, bb}
RE = (a+b)(a+b)
A A

S → AA
A → a/b

S → aA/bA
A → a/b

24/3/18

Context Free Grammars (CFG)

- **CFG :**

CFG is a 4-tuple (V, Σ, R, S), where,

$V \rightarrow$ finite set of variables for non-terminals

$\Sigma \rightarrow$ finite set of terminal symbols.

$R \rightarrow$ finite set of production rule with each rule being a variable and a string of variables and terminals

$S \rightarrow$ start variable / start non-terminal of the grammar $S \in V$.

A grammar $G = (V, \Sigma, R, S)$ is context free if all productions in R of the following form:

i) $\alpha \rightarrow \beta$.

ii) $\alpha \in V$ & $\beta \in (V \cup \Sigma)^*$

Derivations

If $\alpha \rightarrow \beta$ is a production in R in CFG and a & b are the strings in $(V \cup \Sigma)^*$, then

$a\alpha b \underset{G}{\Rightarrow} a\beta b$.

$\underset{G}{\Rightarrow}$ derives in one step

$\overset{*}{\underset{G}{\Rightarrow}}$ derives in multiple steps

We can say that the production $\alpha \rightarrow \beta$ is applied to the string $a\alpha b$ to obtain $a\beta b$. or $a\alpha b$ derives $a\beta b$.

*Suppose $\alpha_1, \alpha_2, \ldots \alpha_m$ are the strings in $(V \cup \Sigma)^*$ and $m \geq 1$, then,

if $\alpha_1 \underset{G}{\Rightarrow} \alpha_2$, $\alpha_2 \underset{G}{\Rightarrow} \alpha_3$, $\ldots$ $\alpha_{m-1} \underset{G}{\Rightarrow} \alpha_m$

then, $\alpha_1 \overset{*}{\underset{G}{\Rightarrow}} \alpha_m$

Q) $A \to 0A1$
$A \to B$
$B \to \#$
$w = 000\#111$.
Derive the string w.

Sol)
$A \to 0A1$
$\to 00A11$
$\to 000A111$.
$\to 000B111$
$\to 000\#111$.

Correct process

$A \underset{G}{\Rightarrow} 0\underline{A}1 \underset{G}{\Rightarrow} 00\underline{A}11 \underset{G}{\Rightarrow} 000\underline{A}111 \underset{G}{\Rightarrow} 000B111 \underset{G}{\Rightarrow} 000\#111$

$A \overset{*}{\underset{G}{\Rightarrow}} 000\#111$

→ Language of CFG:

G_1 is a CFG, then the language of G_1 denoted by $L(G_1)$ is the set of terminal strings that have derivations from the start symbol.

$$L(G) = \{ w \text{ in } \Sigma^* \mid S \overset{*}{\underset{G}{\Rightarrow}} w \}$$

If language L is the language of some CFG, then L is said to be context free language or CFL.

- Sentential Form:

Derivations from the start symbol produces strings that have a special role and these are called sentential form.

If G is the CFG, then any string α in $(V \cup \Sigma)^*$ such that $S \overset{*}{\Rightarrow} \alpha$ is a sentential form.

- Parse Tree / Derivation Tree / Syntax Tree / Production Tree / Generation Tree:

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

Def: Let, $G(V, \Sigma, R, S)$ be a CFG. An ordered tree for this CFG, $G(V, \Sigma, R, S)$ is a parse tree if and only if it has the following properties:-

i) The root is labelled by the starting non-terminal.
ii) Every leaf of the ordered tree has a label from $\Sigma \cup \epsilon$.
iii) Every interior node of the tree has a label from V.
iv) Assume that a vertex has a label $x \in V$ and its children have label from left to right as $y_1, y_2, y_3, \ldots y_n$, then the production must contain a production of the form
$x \to y_1, y_2, y_3, \ldots y_n$.
v) A leaf labelled as ϵ has no siblings.

Yield of parse tree :

- The string that is obtained by concatenating the leaves of any parse tree from left to right is called the yield of the tree.
- The yield is always derived from the root of the tree.

left most derivation : $\overset{*}{\underset{LM}{\Rightarrow}}$, $\underset{LM}{\Rightarrow}$

right most derivation : $\overset{*}{\underset{RM}{\Rightarrow}}$, $\underset{RM}{\Rightarrow}$

always start from start symbol

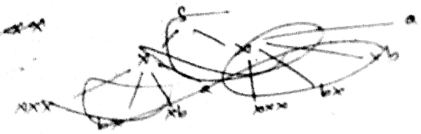
e.g. $S \rightarrow xx$

$x \rightarrow xxx \mid bx \mid xb \mid a$

For this CFG, find the parse tree, left most derivation & right most derivation.

ω = bbaaaab

Sol)



Left most :

$S \underset{LM}{\Rightarrow} \underline{x}x$

$\Rightarrow b\underline{x}x$

$\Rightarrow bb\underline{x}x$

$\Rightarrow bb\underline{x}xxx$

$\Rightarrow bbaxxx$

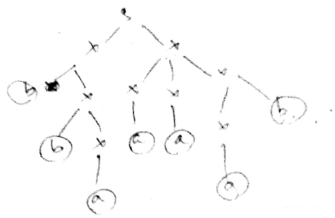
$\Rightarrow bbaa\underline{x}x$

$\Rightarrow bbaaa\underline{x}$

$\Rightarrow bbbaaa\underline{x}b$

$\Rightarrow bbbaaaab$

$S \overset{*}{\underset{LM}{\Rightarrow}} bbaaaab$



Right most

$S \Rightarrow x\underline{x}$

$\Rightarrow x\underline{x}b$

$\Rightarrow \underline{x}ab$

$\Rightarrow xx\underline{x}ab$

$\Rightarrow x\underline{x}aab$

$\Rightarrow \underline{x}aaab$

$\Rightarrow xx\underline{x}aaab$

$\Rightarrow x\underline{x}aaaab$

$\Rightarrow bx\underline{x}aab$

$\Rightarrow bbx\underline{x}aab$

$\Rightarrow bbxx\underline{x}aab$

$\Rightarrow bbx\underline{x}aaab$

$\Rightarrow bb\underline{x}aaaab$

$\Rightarrow bbaaaaab$

CFG Design

Q) $L = a^n b^n$, $n \geq 1$. (if $n \geq 0$, $S \to aSb/\epsilon$)

$S \to aSb/ab$

Q) CFG for all palindromes

L = Palindromes

= { aa, bb, aba, bab, a, b, ... }

even length, odd length, Odd length

Sol) $S \to aSa/bSb/a/b/\epsilon$

$S \to aSa/bSb/\epsilon$ → even length

$S \to aSa/bSb/a/b$ → odd length

Q) $L = \underline{a^n b^n} \underline{c^m}$, $n, m \geq 1$.

A B

Sol) $S \to AB$

$A \to aAb/ab$.

$B \to cB/c$

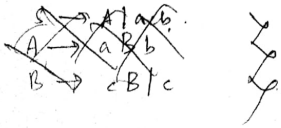
abc

aabbcc

aabbccc

Q) $L = a^n c^m b^n$, $n, m \geq 1$.

Sol)

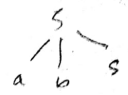


$S \to aSb/aAb$.

$A \to cA/c$

Q) $L = (ab)^n$. , $n \geq 0$.

Sol) $S \to abS/\epsilon$



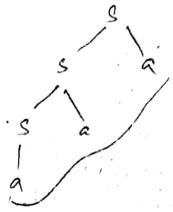
→ Ambiguity in grammar and languages:

A CFG is called ambiguous if for atleast one word in the language that it generates, there are two possible derivations of the word that correspond to different parse tree.

If a CFG is not ambiguous, then it is unambiguous CFG.

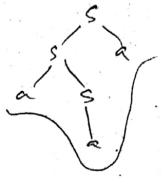
e.g. $S \to aS/Sa/a$ (Grammar)

of string $w = aaa$, find the derivation.



Multiple ways of derivation

⇒ Ambiguous grammar

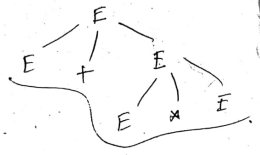


$E \to I$
$E \to E + E$.
$E \to E * E$.
$E \to (E)$
$I \to a/b$.
$I \to Ia/Ib$.
$I \to I0/Ib$.

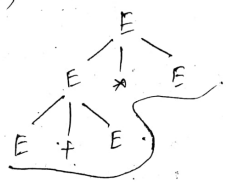
Find derivation and check for ambiguity.

(1)

$E \Rightarrow E + E$
$\Rightarrow E + E * E$



$E \Rightarrow E + E$
$\Rightarrow E + (E)$
$\Rightarrow E + (E * E)$



Note: Ambiguity is the property of the grammar rather than the language.

4/4/18 (Back of DBMS copy).

5/4/18

Useless Symbols.

$S \to aB/bX$
$A \to BAd/BSX/a$ — Terminal symbol
$B \to aSB/bBX$
$X \to SBD/aBx/ad$ — Terminal symbol.

Step 1: Remove non-terminating
B is not having any terminal symbol.
A, X are generating symbol.
B is not generating symbol.

After removing B from all the grammar,

$S \to bX$
$A \to bSX/a$.
$X \to ad$.

Step 2: Remove non-reachable

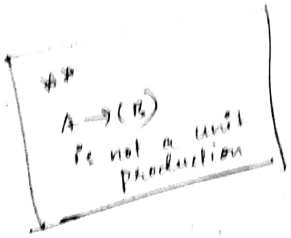
$S \to bX$
$X \to ad$.

→ Removal of unit Production:

A production of the form $A \to B$ (non-te
tends to non-terminal) is called a unit produc

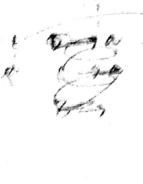
Algorithm to eliminate the unit productions:

while (') a unit production $A \to B$)
{
select a unit production $A \to B$ such that
a production $B \to \alpha$ where α is a terminal.
for (every non-unit production $B \to \alpha$)
{
add production $A \to \alpha$ to the grammar.
eliminate $A \to B$
}
}



eg: $S \to AB$
$A \to a$
$B \to C \mid b$
$C \to D$
$D \to E$
$E \to a$

Unit productions: $B \to C$, $C \to D$, $D \to E$



$S \to AB$
$A \to a$
$B \to a \mid b$
$C \to a$
$D \to a$
$E \to a$
} unreachable

Reduced CFG,

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

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

Remove the unit productions (we need to find

Unit productions: $E \to T$, $T \to F$, $F \to I$

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

Remove the unit production ... used to reduce ... T.

4/18

ε - Production

1. If $A \to \varepsilon$ is a production to be eliminated then look for all productions whose right side contains A and replace each occurrence of A to obtain non-ε-production.

2. Add the resultant non-ε-production to the grammar.

e.g. i) $S \to aA$
$A \to b | \varepsilon$

$A \to \varepsilon$ is the ε production.

Final grammar,
$S \to a | aA$
$A \to b$

ii) $S \to ABAC$
$A \to aA | \varepsilon$
$B \to bB | \varepsilon$
$C \to c$

ε - production
$A \to \varepsilon$
$B \to \varepsilon$

$A \to \varepsilon,\ B \to \varepsilon$

~~Final grammar~~
$S \to C | ABAC$
$A \to a | aA$
$B \to b | bB$
$C \to c$

$A \to \varepsilon$
$S \to BAC | ABC | BC | ABAC$
$A \to a | aA$

$B \to \varepsilon$
$S \to AAC | ABAC | AC$
$B \to b | bB$

~~Final grammar~~,

$S \to ABAC | ABC | BAC | BC | AAC | AC | C$
$A \to aA | a$
$B \to b | bB$
$C \to c$

• CNF (Chomsky Normal form)

A CFG is in CNF if the productions are of the form (i) $A \to$ string of exactly 2 non-terminal (Non-terminal)

(ii) Non-terminal $\to$ 1 Terminal

i.e. $A \to BC$ or $A \to a$

Conversion of CFG to CNF.

Step:

i) Make the start rule non-recursive i.e. if the start symbol S occurs on right side, then create a new start symbol S_1 and add a new production $S_1 \to S$.

ii) Remove ϵ-productions.

iii) Remove unit productions.

iv) Replace each production

$A \to B_1 B_2 \cdots B_n \quad , n > 2$.

with

$A \to B_1 C$

where $C \to B_2 \cdots B_n$

v) If the RHS of any production is of the form $A \to aB$, then replace the production by $A \to XB$ and $X \to a$.

vi) Repeat (iv) and (v) steps till we get normal form.

e.g. $S \to bA \mid aB$

$A \to bAA \mid aS \mid a$.

$B \to aBB \mid bS$.

Sol) Not recursive

No ϵ-production.

7/4/18.

• Greibach Normal Form (GNF):

A CFG is in GNF if the productions are of the form

$A \to aV \mid a \quad , V \in V^*$

or

$A \to a$

$A \to aV_1 V_2 \cdots V_n$.

Steps to convert CFG to GNF:

i) Remove any unit / null prodn in the grammar.

ii) Convert CFG to CNF, if it is not in CNF, remove all the variables of non-terminals into some A_i in ascending orders of i where A_1 is the start symbol.

iii) Modify the grammar, so that every prodn is of the following form:

$A_i \to A_j X \quad (j > i, X \in V^*, a \in \Sigma)$

$A_i \to aX$

iv) Remove left recursions if the productions are of the form

$A_i \to A_i X \quad , X \in V^*$

by introducing a new variable.

i)

$S \to AB \mid BC$.

$A \to aB \mid bA \mid a$.

$B \to b \mid cC \mid b$.

$C \to c$

} in form of GNF ($S \to aV \mid a$).

$S \to AB \mid BC$ ⟶ in CNF

.

$S \to aBB \mid bAB \mid aB \mid bBC \mid cCC \mid bC$.

Rest 3 in same form .

ii)

$S \to CA \mid BB$

$B \to b \mid SB$

$C \to b$

$A \to a$

Renaming non-terminals , (Rename in ascending order)

$A_1 \to A_2A_3 \mid A_4A_4$.

$A_4 \to b \mid A_1A_4$.

$A_2 \to b$

$A_3 \to a$.

A_4 not in GNF .

$A_4 \to b \mid A_1A_4$ (i > j, which should not be) .

$A_4 \to A_2A_3A_4 \mid A_4A_4A_4 \mid b$

$A_4 \to bA_3A_4 \mid A_4A_4A_4 \mid b$.

Removing left recursion

$A_4 \to bA_3A_4Z \mid bZ \mid bA_3A_4Z$

So,

$A_1 \to A_2A_3 \mid A_4A_4$

$A_4 \to b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \to A_4A_4Z \mid A_4A_4$

$A_2 \to b$

$A_3 \to a$.

Still A_1 not in GNF, so

$A_1 \to bA_3 \mid bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$.

$A_4 \to b \mid bA_3A_4 \mid bZ \mid bA_3A_4Z$

$Z \to A_4A_4Z \mid A_4A_4$

$A_2 \to b$

$A_3 \to a$.

Still Z not in GNF, so,

$Z \to A_4A_4Z \mid A_4A_4$

$Z \to bA_4Z \mid bA_3A_4A_4Z \mid bZA_4Z \mid bA_3A_4ZA_4 \mid$

$bA_4 \mid bA_3A_4A_4 \mid bZA_4 \mid bA_3A_4ZA_4$.

Find derivation for aabbbccc .

PDA :

A push down automata is a system which is mathematically defined as a 7-tuple.

$P = (Q, \Sigma, \delta, q_0, Z_0, F, \Gamma)$

$Q \rightarrow$ non-empty finite set of states.
$\Sigma \rightarrow$ input alphabet.
$q_0 \rightarrow$ initial state, $q_0 \in Q$.
$F \rightarrow$ set of finite states, $F \subseteq Q$.
$\Gamma \rightarrow$ finite set of push down symbols / stack alphabet
$\delta \rightarrow$ transition : $Q \times \{\Sigma \cup \in\} \times \Gamma^* \longrightarrow Q \times \Gamma^*$.
$Z_0 \rightarrow$ initial stack symbol.

FSM + stack (for memory purpose) = PDA

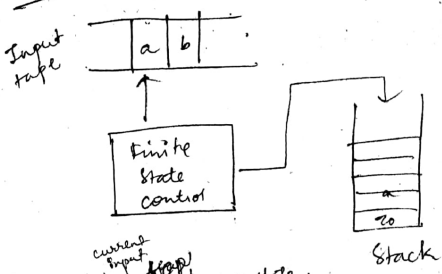
Transition func^n DPDA $\rightarrow Q \times \{\Sigma \cup \in\} \times \Gamma^* \longrightarrow Q \times \Gamma^*$.
NPDA $\rightarrow Q \times \{\Sigma \cup \in\} \times \Gamma^* \longrightarrow 2^{Q \times \Gamma^*}$.

Explanation for the moves of PDA :

$\delta : Q \times \{\Sigma \cup \in\} \times \Gamma^* \longrightarrow Q \times \Gamma^*$,

1) $\delta(q, a, X) \longrightarrow (P, y)$
δ takes an argument as triplet.
$q \rightarrow$ current state
$X \rightarrow$ current start symbol.
$a \rightarrow$ " input.
$P \rightarrow$ next state.
$y \rightarrow$ stack symbol that should be pushed or popped

Model of PDA



current state / current input / stack symbol / new state / string that replaces Z_0.

2) $\delta(q, a, Z_0) \rightarrow (P, y)$ → Push.

The Interpretation of the above expression is whenever the PDA is in state q & Z_0 on the top of stack, then it may read 'a' from the input tape.
On reading a, it replaces Z_0 with y and moves to the next state P.

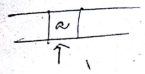
e.g. $\delta(q, a, Z_0) = (P, aZ_0)$ → Push in stack

3) $\delta(q, a, b) = (P, \in)$ → Pop in stack

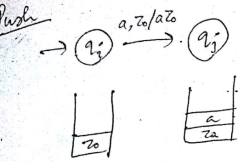
4) $\delta(q, a, Z_0) = (P, Z_0)$ → no operation in stack
(when same)

...ted or popped.

Diagramatic representation

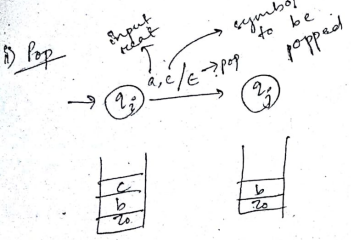


i) Push



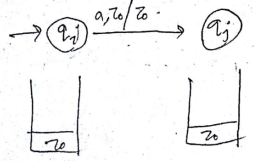
$\delta(q_i, a, z_0) = (q_j, az_0)$

ii) Pop



$\delta(q_i, a, c) = (q_j, \epsilon)$

iii) Skip (No operation)



$\delta(q_i, a, z_0) = (q_j, z_0)$

PDA computation:

Any 3-tuple $(p, w, \beta) \in Q \times \{\Sigma \cup \epsilon\} \times \Gamma^*$ is called an instantaneous description (id) of PDA M which includes the current state p, part of the input tape that has'nt been read i.e. w and the content of the stack with topmost stack symbol written first i.e. β.

The transition relation δ defines the set relation $\vdash_M$ on instantaneous description.

- In general, the PDA is non-deterministic i.e. in a given instantaneous description, (p, w, β), there may be several possible steps and any of these steps can be chosen in a computation.

Initial stack description: (q_0, w, z_0).

- There are 2 modes of acceptance of a string in a PDA:
  i) Acceptance by final state } May/May not equal in NPDA, not equal in DPDA
  ii) Acceptance by empty stack

Acceptance by final state

After reading the I/P, the automaton reaches the final state F.

Acceptance by empty stack

After reading the I/P, the automata empty. in stack.

Acceptance by final state

$$L(M) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (f, \epsilon, \gamma) \}$$

$f \in F, \gamma \in \Gamma^*$

Acceptance by empty stack

$$N(M) = \{ w \in \Sigma^* \mid (q_0, w, Z_0) \vdash_M^* (q, \epsilon, \epsilon) \}$$

$q \in Q$

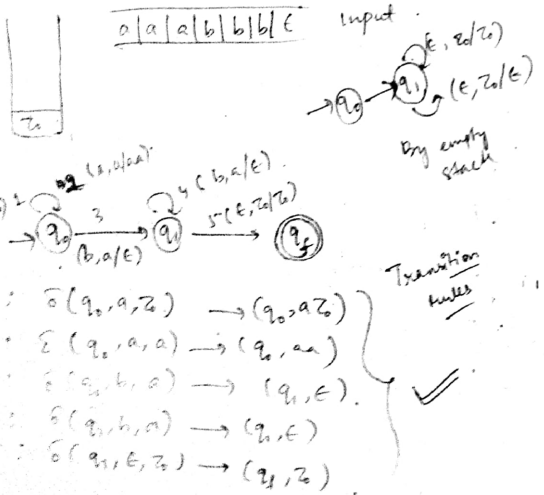
14/18 If transition is not determined for a symbol, string is rejected.

NPDA & DPDA are not equivalent.

Eg: PDA for $L = \{ a^n b^m \}$, $n \geq 1$

$L = \{ ab, aabb, aaabbb, \ldots \}$

$w = aaabbb$



Transition rules

1: $\delta(q_0, a, Z_0) \longrightarrow (q_0, aZ_0)$

2: $\delta(q_0, a, a) \longrightarrow (q_0, aa)$

3: $\delta(q_0, b, a) \longrightarrow (q_1, \epsilon)$

4: $\delta(q_1, b, a) \longrightarrow (q_1, \epsilon)$

5: $\delta(q_1, \epsilon, Z_0) \longrightarrow (q_f, Z_0)$

✓✓

CFL ⟶ NPDA.

DPDA ⟶ DCFL. (DCFL $\subseteq$ CFL)

Properties of CFL :

i) CFL is closed under union, concatenation and kleene star closure.

ii) The family of CFL is not closed under intersection and complementation.

iii) The intersection of a CFL and a regular language is a CFL.

iv) Periodicity of CFL : When patterns are repeated, it is called periodicity of CFL.

Pumping Lemma for CFL :

If L is a CFL, then L has a pumping length P such that any string w where $|w| \geq P$ may be divided into $w = uvxyz$. Such that the following cond^n must be true :

i) $uv^i xy^i z$, is in L for $i \geq 0$

ii) $|vy| > 0$

iii) $|vxy| \leq P$

for all $i \geq 0$

Q) Show that $L = a^n b^n c^n$, $n \geq 0$, is not a CFL.
$L = \{\epsilon, abc, aabbcc, \ldots\}$

Soln) Proof by contradiction.

Step 1: Assume L is a CFL with pumping length P.

Step 2: We can divide L as
$w = uvxyz$
i.e., $w = a^{P/3} b^{P/3} c^{P/3}$ (x, y)
$= aaa\ldots bbb\ldots ccc\ldots$

Step 3: Check for cond^n.
cond^n 1 - v & y contain one symbol.

$\underbrace{a}_{u}\underbrace{aa}_{v}\underbrace{b\ldots b}_{x}\underbrace{b}_{y}\underbrace{cc\ldots c}_{z}$

uv^2xy^2z.
for $i = 2$,
uv^2xy^2z.
$a\,aa\,aa\,b\ldots b\,b\,cc\ldots c$
$= a^6 b^{P+1} c^P \notin L$.

∴ It is not a CFL.
(We don't need to go for cond^n 2 if 1st is false itself).

Cond^n 2 - v & y contain multiple symbols.

$\underbrace{aa}_{u}\underbrace{aabb}_{v}\underbrace{b}_{x}\underbrace{bc}_{y}\underbrace{ccc}_{z}$.

uv^ixy^iz
for $i = 2$
uv^2xy^2z = $aa\,aabbaabb\,b\,bcbc\,ccc$
$= a^4 b^6 c^4 b^1 c^1 \notin L$

∴ It is not a CFL

If multiple cases arise, show all cond^n.

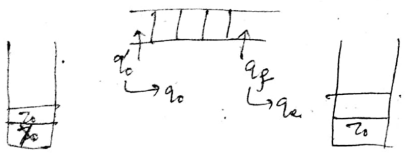
Topic
→ Equivalence of PDA acceptance by final state and empty stack

If there is a language L that is accepted by PDA by final state, then there exists another PDA that accepts the same language by empty stack

$L = T(M_2)$ → PDA M_2 ✓ by final
$L = Nt(M_1)$ → PDA M_1 ✓ by empty stack

$M_2 = (Q, \Sigma, \Gamma, q_0, Z_0, F, \delta_2)$
$M_1 = (Q \cup \{q', q_e\}, \Sigma, \Gamma \cup \{X_0\}, q', Z_0, F, \delta_1)$
(initial state, erasing state, stack symbol)

Remember!
Transition rule for δ_1 (empty stack).
i) $\delta_1(q', \epsilon, X_0)$ contains $(q_0, Z_0 X_0)$
ii) $\delta_1(q, a, Z)$ includes $\delta_2(q, a, Z)$
iii) $\delta_1(q, \epsilon, Z)$ contains (q_e, ϵ), $q \in F, Z \in \Gamma \cup \{X_0\}$.
iv) $\delta_1(q_e, \epsilon, Z)$ contains (q_e, ϵ)



$$\delta(q_0, \epsilon, X_0) \longrightarrow (q_0, Z_0, X_0)$$

→ Equivalence of CFG and PDA :

Note :

Transition rules :

i) Insert the stack symbol of the grammar.

$$\delta(p, \epsilon, Z_0) = (q, S Z_0)$$

ii) For each rule in the CFG of the form

$$A \longrightarrow x \quad , \quad \text{[illegible]}$$

$$\delta(q, \epsilon, A) \longrightarrow (q, x)$$

If $A \longrightarrow x/y$

$$\delta(q, \epsilon, A) \longrightarrow (q, x)$$
$$\delta(q, \epsilon, A) \longrightarrow (q, y)$$

iii) For each terminal symbol, we will pop that.

$$\delta(q, a, a) \longrightarrow (q, \epsilon)$$

iv) Final transition

$$\delta(q, \epsilon, Z_0) \longrightarrow (q, \epsilon)$$

$CFG = (V, \Sigma, R, S)$

$PDA = (Q, \Sigma, \delta, \Gamma, q_0, Z_0, F)$

$Q = \{p, q\}$ (2 symbols)
start

$\Sigma = \{\Sigma\}$

δ = Transition rules

$\Gamma = Z_0 \cup \{\Sigma\} \cup \{V\}$

$q_0 = p$

$Z_0 = Z_0$

$F = q$

e.g. For the given CFG, design a PDA.

$E \longrightarrow E + E$
$E \longrightarrow E * E$
$E \longrightarrow a$

$CFG = (V, \Sigma, R, S)$
$= (\{E\}, \{a, +, *\}, R, E)$

$PDA = (Q, \Sigma, \delta, \Gamma, q_0, Z_0, F)$
$= (\{p, q\}, \{a, +, *\}, \{Z_0, a, +, *, E\}, \delta, p, Z_0, q)$

1) $\delta(p, \epsilon, Z_0) \longrightarrow (q, E Z_0)$ — step 1.
2) $\delta(q, \epsilon, E) \longrightarrow (q, E+E)$.
3) $\delta(q, \epsilon, E) \longrightarrow (q, E*E)$ } step 2
4) $\delta(q, \epsilon, E) \longrightarrow (q, a)$.

3) $\delta(q, a, a) \rightarrow (q, \epsilon)$.

4) $\delta(q, +, +) \rightarrow (q, \epsilon)$

5) $\delta(q, *, *) \rightarrow (q, \epsilon)$

} Step 3 -

4) $\delta(q, \epsilon, Z_0) \rightarrow (q, \epsilon)$. - Step 4.

This PDA is accepting by empty stack .

Let, $w = a+a$. $(E \rightarrow a)$.

Processing the VP string .

$E \rightarrow E+E$.

$\rightarrow a+E$

$\rightarrow a+a$

13/4/18

CSG

A grammar $G(V, \Sigma, P, S)$ is said to be context sensitive if productions are of the form .

$\boxed{x \rightarrow y}$ $\{ x, y \in (V \cup \Sigma)^+ \}$ and $\{ |x| \leq |y| \}$.

ϵ - productions are not allowed in CSG .

$A \rightarrow Ab$ ✓

$|A| \rightarrow |\epsilon|$ ✗ .

$Ab \rightarrow a$ ✗

Non-contradicting } other names for CSG .
length increasing }

e.g. $XAY \rightarrow XaY$ X, Y can be terminal or non-terminal

$\Rightarrow A \rightarrow a$.

$PBQ \rightarrow PcQ$.

$\Rightarrow B \rightarrow c$.

* CSG $\rightarrow \epsilon$ ✗ . $RG \subseteq CFG \subseteq CSG$

CFG $\rightarrow \epsilon$ ✓

RG $\rightarrow \epsilon$ ✓

Exception → Only start symbol can imply ϵ but the start symbol can't be present in the right side .

In CSG, $S \rightarrow \epsilon$ ✓

$S \rightarrow aS$ ✗ .

→ Properties of CSL :

1) The family of CSL is closed under union, intersection, concatenation, kleene plus, reversal & complementation .

e.g. $S \rightarrow aAbc \mid abc$

$Ab \rightarrow bA$

$Ac \rightarrow Bbcc$

$bB \rightarrow Bb$

$aB \rightarrow aa \mid aaA$

} CSG for $L = \{a^n b^n c^n , n \geq 1\}$.

Find derivation for aaabbbccc .

$\vdash$ a A b c

$\Rightarrow$ a b A c (replaced 4b)

$\Rightarrow$ a b B b c c (replaced Ac)

$\Rightarrow$ a b B b b c c

$\Rightarrow$ a B b b b c c

$\Rightarrow$ a a A b b b c c

$\Rightarrow$ a a B A b b c c

$\Rightarrow$ a a b b A b c c

$\Rightarrow$ a a b b b A c c

$\Rightarrow$ a a b b b B b c c c

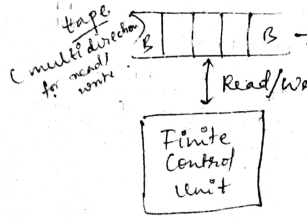
$\Rightarrow$ a a b b B b c c c

$\Rightarrow$ a a b B b b c c c

$\Rightarrow$ a a B b b b c c c

$\Rightarrow$ a a a a b b b c c c

Turing Machine :



Turing Machine

Read theory of ...

Mathematical definition :

A turing machine is a seven tuple

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

$Q \rightarrow$ finite set of states.

$\Sigma \rightarrow$ input alphabets.

$\Gamma \rightarrow$ tape

$\delta \rightarrow$ transition, $\delta : Q \times \Gamma \rightarrow Q \times \Gamma \times \{L, R\}$ (left right)

$q_0 \rightarrow$ initial state

$F \rightarrow$ Final state.

$B \rightarrow$ Black space

$\Sigma \subseteq \Gamma - \{B\}$.

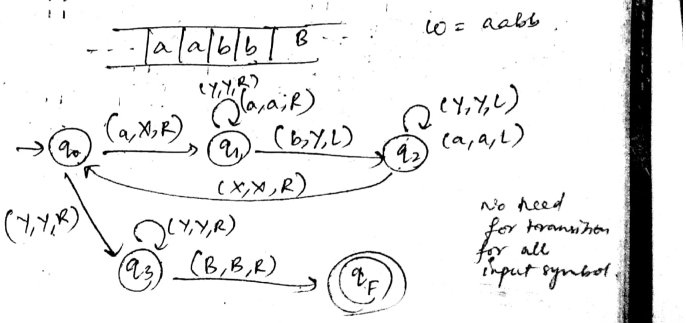
$q_0 \in Q$.

$B \in \Gamma$

$F \subseteq Q$.

Q) Design a turing machine for $L = a^n b^n$, $n \geq 1$.

Sol) $L = \{ab, aabb, \ldots\}$



No need for transition for all input symbol