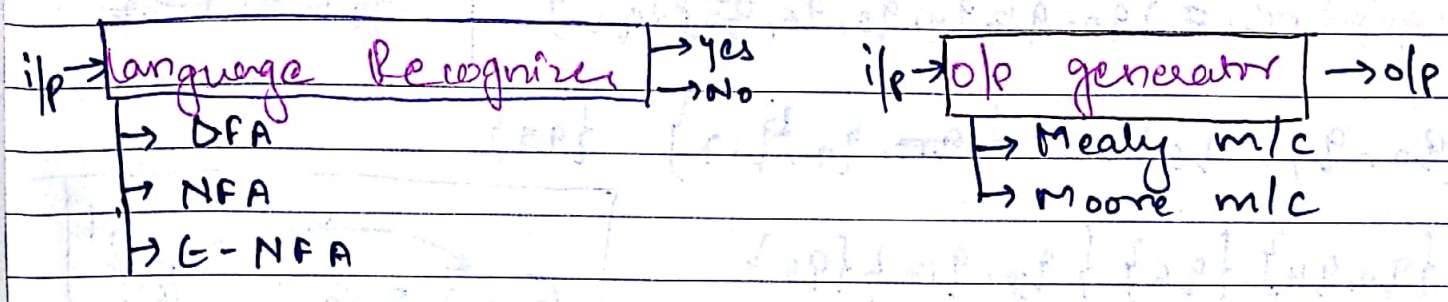
Date _____

Theory of computation:

- Mathematical study of computing machines and their capability.

Finite Automata — language Recognizer
— o/p generator.



→ DFA $(Q, \Sigma, q_0, F, \delta)$
finite no. of states; i/p alphabet; initial state; final states; Transition function → $Q \times \Sigma \to Q$

DFA = NFA

→ Fails to accept languages in which comparison exists, because memory is required.

Regular languages: languages for which FA is possible.

PDA: Finite automata with one stack.

→ Minimal DFA for one regular language is exactly one.

DFA Minimization: [Only application for DFA, not for NFA]

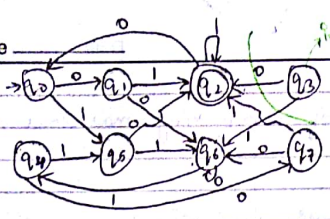
① Remove inaccessible states [Not reachable from initial state]

② Combine equivalent states → transitions of both q_1 & q_2 going to either final or non-final states.

Minimization Algorithms:

① State equivalence method ② Table filling method

Date

E.g. 

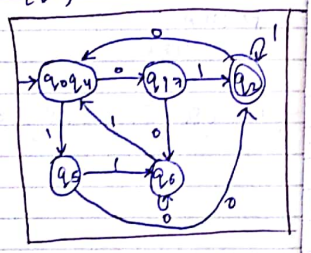
	0	1
→ q_0	q_{17}	q_5
q_{17}	q_6	q_2
(q_2)	q_0	q_2
q_4	q_{17}	q_5
q_5	q_2	q_6
q_6	q_6	q_4
~~q_7~~	q_6	q_2

π_0 (0-equivalence) = $\{q_0, q_{17}, q_4, q_5, q_6, \cancel{q_7}\}\{q_2\}$

$\pi_1 = \{q_0, q_4, q_6\}$ $\{\cancel{q_7}, q_2, q_{17}\}$ $\{q_5\}$

$\pi_2 = \{q_0, q_4\}\{q_6\}\{q_2, q_{17}\}\{q_5\}$

$\pi_3 = \{q_0, q_4\}\{q_6\}\{q_2\}\{q_{17}\}\{q_5\}$



* In minimal DFA:
- All states are final → Complete language
- No final state → Empty language

* Complement of a DFA: Interchange final states → nonfinal states & nonfinal states → final states.
[$\Sigma^* - L$]

* Strings are not in A.P. ⇒ Not Regular
* String not having common difference ⇒ [DFA not possible]

Product Automata:
⇒ AND condition: if both states are final, then final.
⇒ OR condition: if anyone state is final, then, final.

Minimal DFAs:

Date

1. String ending with particular n-length string → n+1 states
2. n-length substring → n+1 states
3. string of {a,b}, having 'a' as n^{th} symbol (LHS) → n+2 states having 1 dead state
4. Exactly n-length string → n+2 states (1 dead state)
5. Atleast n-length string → n+1 states
6. Atmost n-length string → n+2 states (1 dead state)
7. Length of string divisible by 'n' → n states

Eg: (a) a's at least 3 and b's atmost 2.
↳ n+1 = 4 ↳ n+2 = 4 (including 1 dead state)
For calculation of minimal states remove dead state:
4 × 3 = 12 + 1 → Add only 1 dead state
= 13 states.

(b) a's exactly 2 and b's atmost 3:
n+2 = 4 n+2 = 5
-1 -1
3 × 4 = 12 + 1
= 13 states

(8) Divisible by m,n:
Case I: m divides n | n divides m
AND ⇒ LCM(m,n)
OR ⇒ GCD(m,n)

Case II: m doesn't divide n | n doesn't divid m
GCD(m,n) = 1 | GCD(m,n) ≠ 1
AND ⇒ m×n | AND ⇒ LCM(m,n)
OR ⇒ m×n | OR ⇒ LCM(m,n)

Date

E.g. Length of string divisible by 2 AND 4 = 4
" " " " " 2 OR 4 = 2
" " " " " 3 AND 4 = 12
" " " " " 6 AND 8 = 24

(9) ~~Ex~~. Minimal DFA states accepting strings of 'a' & 'b' when n^{th} i/p symbol is 'a', reading from RHS. $= 2^n$ states.

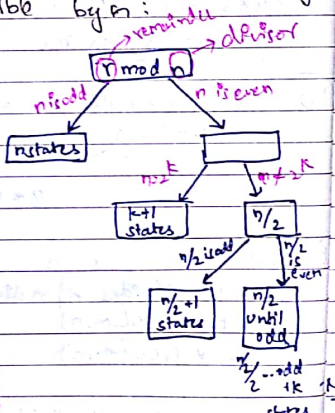
* Construction of NFA is easier than DFA.

(10) No. of states in minimal DFA, that accepts all binary numbers, which are divisible by n:

(i) Divisible by 4 = 3 states
(ii) Divisible by 5 = 5 states
(iii) Divisible by 6 = 4 states
(iv) " " 8 = 4 states
(v) " " 10 = 6 states
(vi) " " 12 = 5 states
(vii) " " 16 = 5 states

$\frac{12}{2} = \frac{6}{2} = 3$
1 + 1 + 3 = 5 states

$\frac{10}{2} = 5$
1 + 5 = 6 states



Date

finite no. of states
initial state
Transition fn.

#NFA: $(Q, \Sigma, q_0, F, \delta)$
Σ: i/p alphabet; F: set of final states; δ: $Q \times \Sigma \to 2^Q$ or $P(Q)$ (power set of Q)

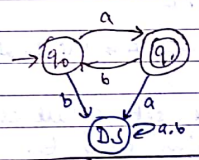
* Every DFA is NFA but not every NFA is DFA.
* NFA accepting strings of 'a' and 'b', where the n^{th} i/p symbol is 'a' from R.H.S. is n+1 states.

Expressive Power: NFA = DFA

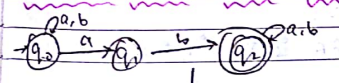
#NFA to DFA: (Subset construction algorithm)
* While converting NFA to DFA, no. of states possible in DFA for given n-state NFA is $\leq 2^n$.

* When NFA be cause of missing transitions, ADD DEAD STATE.

E.g.:



Minimal DFA [NFA → DFA → Minimal DFA]:



NFA	a	b
→q_0	$\{q_0, q_1\}$	q_0
q_1	-	q_2
q_2	q_2	q_2

DFA	a	b
→q_0	$[q_0q_1]$	q_0
$[q_0q_1]$	$[q_0q_1]$	$[q_0q_2]$
$[q_0q_2]$	$[q_0q_1q_2]$	$[q_0q_1]$
$[q_0q_1q_2]$	$[q_0q_1q_2]$	$[q_0q_2]$

May not be minimal

	a	b
→q_0	q_0q_1	q_0
q_0q_1	q_0q_1	q_0q_2
$[q_0q_1q_2]$	$q_0q_1q_2$	q_0q_2

Minimal

Date

ε-NFA:

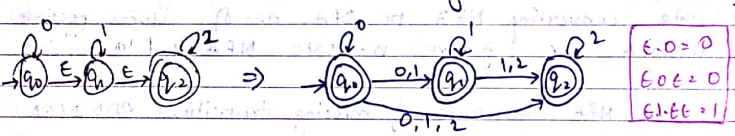
$\delta : Q \times \Sigma \cup \{\epsilon\} \to 2^Q$

* ε-NFA to NFA:

1. Same no. of states in NFA.
2. The states from which final state can be reached with only ε, make them as final states.
3. No. of final states may increase.

Transitions:

- Copy direct transitions [visible]
- Make transitions possible through ε [invisible in ε-NFA]

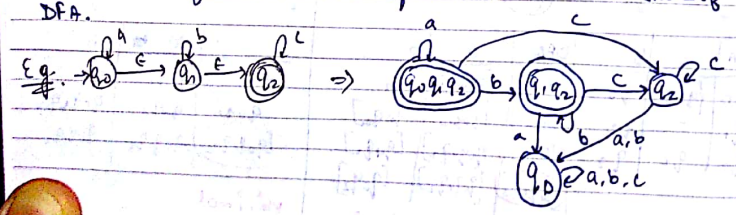


Minimal DFA: [ε-NFA → NFA → DFA → Minimal DFA]

Epsilon- Closure: of a state is the set of all states reachable from that states using only ε.

→ while converting ε-NFA to DFA:
ε-closure of initial state of NFA is the initial state of DFA.

Eg. #



Date

Regular Expression:

→ simplest way of representing a language.
→ for regular language, we can construct regular expression.
* Every regular expression generates one regular language.
* cannot construct R.E. for non-regular language.
* For 1 regular language, there may be many REs.

3 operators:

* → Kleene Closure operator
• → Concatenation operator
+ → Union operator

* > • > +
precedence

NOTE → Palindrome languages over 1 symbol are regular.
over more than 1 symbol are non-regular. Hence, regular expression is not possible.

Eg # $L = \{ w x w^R \mid w, x \in \{a,b\}^* \}$ ⇒ it is regular.

Identities of Regular Expressions:

1. $R + \phi = \phi + R = R$
2. $R \cdot \phi = \phi \cdot R = \phi$
3. $R \cdot \epsilon = \epsilon \cdot R = R$
4. $R^* \cdot R^* = R^*$, $(R^*)^* = (R^*)^+ = (R^+)^*$
5. $RR^* = R^* \cdot R = R^+$
6. $R^+ + \epsilon = R^* = R \cdot R^* + \epsilon$
7. $\epsilon^* = \epsilon = \epsilon^+$
8. $\phi^* = \epsilon \mid \phi^+ = \phi$
9. $a + b = b + a$
10. $a \cdot b \neq b \cdot a$
11. $(a+b)^* \neq a^* + b^*$
12. $(a+b)^* < a^* b^*$
13. $(a+b)^* \neq (ab)^*$
14. $a^* \cdot a^* = a^*$
15. $a^* + a = a^*$
16. $a^* \cdot a = a^+$
17. $(a+b)^* = (a^* + b^*)^* = (a^* + b)^* < (a + b^*)^* = (a^* \cdot b^*)^*$
18. $a(ba)^* = (ab)^* a$
19. $a + a = a$
20. $a \cdot a \neq a$

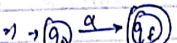
Date

E.g. No. of 1's divisible by 2 : $(0^*10^*10^*)^* + 0^*$

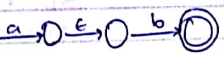
R.E. to F.A. (ε-NFA) : Thomson Construction Algo.

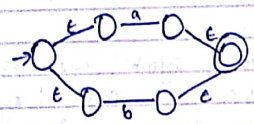
① $\phi \Rightarrow$ 

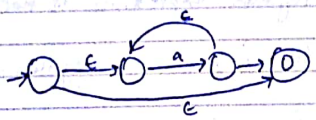
② $\epsilon \Rightarrow$ 

③ $a \Rightarrow$ 

Basic RE

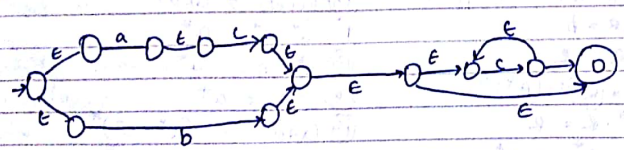
④ $a.b \Rightarrow$ 

⑤ $a+b \Rightarrow$ 

⑥ $a^* \Rightarrow$ 

RE
⇓
ε-NFA
⇓
NFA
⇓
DFA
⇓
Min. DFA

E.g.. $(ac+b).c^*$



Regular Language :

* Language for which F.A. / R.E. exists.

* Total no. of substrings for a given string = $\frac{n(n+1)}{2}+1$

* Prefixes for n-length string = $n+1$

NOTE

⇒ All - finite languages are regular.

⇒ Any infinite language requiring infinite m/m [comparison] is not regular.

Pumping Lemma :

→ Used to prove a non-regular language as non-regular.

→ Based on pigeonhole principle.

→ Uses proof by contradiction.

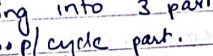
① Assume lang. 'L' is regular.

② There must be a 'n' state F.A. for the language.

③ Select some string 'z' with $|z| \geq n$

④ Divide string into 3 parts → u, v, w

'v' is the loop/cycle part.



⑤ If the given lang. is reg. ⇒ $\forall\, i \geq 1$; $uv^iw \in L$

If for atleast one value of 'i' $uv^iw \notin L$, 'L' is non-reg.

NOTE

⇒ Pumping Lemma is strong for proving nonreg. as non-reg.

⇒ Weak for proving reg as reg.

[for reg lang., infinite no. of cases need to be proved]

Date _____

Decision Properties of Regular Languages:

① Emptiness Problem: Whether given lang. is empty or not OR F.A. accepts empty language or not.

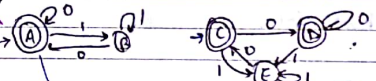
Algo:
- Remove inaccessible states.
- If no final state present, FA accepts empty language.

② Finiteness Problem: Lang. is finite or not.

Algo:
- Eliminate inaccessible states.
- Eliminate states from which we cannot reach final state.
- If no loop/cycle present, then, language is finite.

③ Equivalence Problem: Two F.A. are accepting same lang. or not

E.g.



FAs are equal.

	0	1
AC	AD	BE
AD	AD	BE
BE	AC	BE

Final - non-final combination → NOT Equal

Final Final combination

④ Completeness Problem: FA accepts complete lang. or not.

Algo:
- Complement the FA.

If it accepts empty lang., the original language is complete.

⑤ Membership Problem: String 'x' is member of a given FA or not.

Date _____

FA as o/p generator: → Mealy m/c, Moore m/c.

$(Q, \Sigma, q_0, \Delta, \delta, \lambda)$

Q: finite no. of states; Σ: i/p alphabet; q_0: initial state; Δ: o/p alphabet; δ: transition function; λ: o/p function

Mealy: $\lambda: Q \times \Sigma \to \Delta$; Moore: $\lambda: Q \to \Delta$

→ Both Mealy and Moore m/c are:
- Deterministic
- No final state
- Used in circuit design
- can be represented by transition diagram or table.

Conversion:

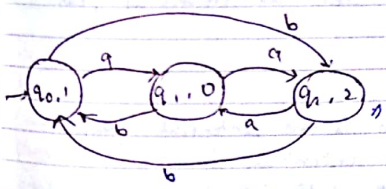
Mealy m/c:

	i = a		i = b	
	N.S.	o/p	N.S.	o/p
→q_0	q_2	1	q_3	1
q_1	q_3	0	q_2	0
q_2	q_1	0	q_1	0
q_3	q_0	1	q_2	1

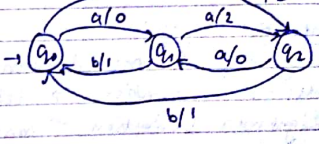
⇒

Moore:

	o/p	a	b
→q_0	1	q_{21}	q_{31}
q_1	0	q_{30}	q_{20}
q_{20}	0	q_1	q_1
q_{21}	1	q_1	q_1
q_{30}	0	q_0	q_{21}
q_{31}	1	q_0	q_{21}



⇒



Date

NOTE

- For n-states, m-o/p symbol Mealy m/c, there can be max^m $m \times n$ states, in equivalent Moore m/c.
- If the initial state is splitted, any one can be taken as ~~first~~ initial state.
- ~~For~~ n-length i/p → o/p → Mealy → n
  Moore → n+1
- Initial o/p of Moore m/c is neglected for equivalent m/c construction.

Grammar:

→ Set of rules used to describe strings of a language.
→ Every grammar generates one language.

$G = (N, T, P, S)$
N: Non-Terminal or Variables; T: Terminals; P: Number of Productions; S: Starting Symbol

* Parse Tree:

→ All leaf nodes of the parse tree are known as yield of the parse tree.

* Sentential Form:

→ Every step in the derivation is one sentential form.
* Leftmost Derivation → left sentential form
* Rightmost Derivation → right sentential form

Date

Type-3 or Regular Grammar

Left linear	Right Linear
$A \to Bx \mid x$	$A \to xB \mid x$

* $A, B \in V$, $x \in \Sigma^*$

Type-2 or Context Free Grammar

$A \to \alpha \mid \alpha \in (V+T)^*$

Type-1 or Context Sensitive Grammar

$\alpha \to \beta \mid \alpha, \beta \in (V+T)^*$
$|\alpha| \leq |\beta|$

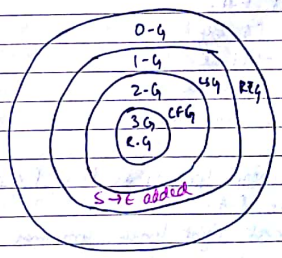
Type-0 or Unrestricted or Recursive Enumerable Grammar

$\alpha \to \beta \mid \alpha \in (V+T)^+$, $\beta \in (V+T)^*$

* Linear Grammar:

→ In any grammar, LHS has one non-terminal and RHS has at most one non-terminal.
→ Left linear or Right linear.
→ If any linear grammar is completely left linear or completely right linear (but not both), then it is regular.
→ All regular grammars are linear but all linear grammars need not be regular.

Chomsky Hierarchy:



NOTE

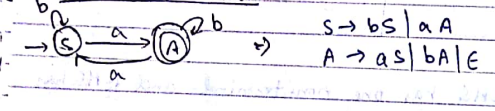
$A \to \epsilon$ is a valid Context sensitive grammar Production, if A is not present in the RHS of grammar productions.

Date

E.g: $S \to aAB \mid \epsilon$
$aA \to bS \mid bb$
$B \to d$
Type-0

Regular Grammar
→ By default, it is right linear.
→ Left linear ⟷ Right linear.

FA to Grammar:

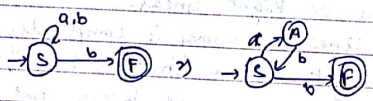


⇒ $S \to bS \mid aA$
$A \to aS \mid bA \mid \epsilon$

N ⇒ states
T ⇒ a, b
P ⇒ δ
S ⇒ Initial state

Gramma to FA:

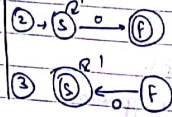
$S \to abS \mid b$ ⇒



⇒

F.A. for left linear grammar:
① Reverse RHS part of productions.
② Construct F.A.
③ Reverse FA
- Initial → Final
- Final → Initial
- Reverse transitions

$S \to S1 \mid 0$
① $S \to 1S \mid 0$



(01*)

Left linear Grammar from F.A:
① Reverse F.A.
② Construct right linear grammar.
③ Reverse RHS part.

Date

Context Free Grammar:
→ Used to represent syntax of languages.
→ Can generate regular languages also.

NOTE ⇒ Checking whether CFG generates regular language or not is undecidable. (Regularity Problem of CFG).

⇒ Checking whether given grammar is ambiguous or not is undecidable problem.
⇒ Elimination of ambiguity is undecidable problem.

Inherently ambiguous language:
- If all the grammars for a language are ambiguous.
- If even, one grammar is unambiguous, language is unambiguous.

Simplification of CFG:
Step-1: Eliminate null productions.
Step-2: Eliminate unit productions.
Step-3: Eliminate useless variables.

E.g. $S \to AaB$
$A \to Bc$
$B \to a \mid \epsilon$
$C \to b \mid \epsilon$

⇒ $S \to AaB \mid Aa \mid Aaa$
$A \to Bc \mid ba \mid b \mid \epsilon$
$B \to a$
$C \to b$

⇒ $S \to AaB \mid Aa \mid Aaa$
$A \to ab \mid a \mid b$

$S \to Aa \mid Aa \mid aa$ ⇒ $S \to Aaa \mid Aa \mid aa \mid a$
$A \to ab \mid a \mid b \mid \epsilon$ $A \to ab \mid a \mid b$

Date

Normal Forms of CFG:

Chomsky NF	Greibach NF
$A \to BC$ $A \to a$ $A, B, C \in V$ $a \in T$	$A \to aV^*$; $a \in T$
RHS has one terminal or two nonterminals.	RHS → one terminal followed by any no. of non-terminals

Note → Grammar having left recursion cannot be converted into GNF.

Removal of left recursion:

Direct:

$A \to A\alpha \mid \beta$ → $A \to \beta A'$, $A' \to \alpha A' \mid \epsilon$ → $A \to A\alpha_1 \mid A\alpha_2 \mid \ldots \mid \beta_1 \mid \beta_2$, $A \to \beta_1 A' \mid \beta_2 A'$, $A' \to \alpha_1 A' \mid \alpha_2 A'$

Indirect:

Eg. $S \to AA \mid 0$
$A \to S \mid 1$
$A \to AAS \mid AS \mid 1$
$A \to OSA' \mid 1A'$
$A' \to ASA' \mid \epsilon$

→ $S \to AA \mid 0$
$A \to OSA' \mid 1A'$
$A' \to ASA' \mid \epsilon$

Eg CNF:
$S \to aSb \mid ab$
$S \to AB \mid AX$
$S \to AX \mid AB$
$A \to a$
$B \to b$
$X \to SB$

① Simplify the grammar
② Replace Terminal → Non-Terminal
③ Multiple Non-Terminals to Two Non-Terminals.

Date

GNF:

① $S \to asb \mid ab$
$S \to aSB \mid aB$
$B \to b$

② $S \to AB$
$A \to aA \mid bB \mid b$
$B \to b$

→ $S \to aAB \mid bBB \mid bB$
$A \to aA \mid bB \mid b$
$B \to b$

Note
↳ For n-length string generation:
- CNF grammars → $(2n-1)$ steps
- GNF grammars → n steps.

Decision Properties of CFG:

Decidable:

① Emptiness Problem:

Algo.
① Eliminate all useless variables.
② If the starting symbol is useless, grammar generates empty language.

② Finiteness Problem: [CNF Graph]

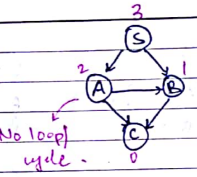
Algo:
① Convert grammar into CNF grammar.
② Construct CNF graph.
③ If loop/cycle does not exist in graph, language is finite.

* If language generated is finite, then rank of a non-terminal is the length of the longest path, starting from that non-terminal in CNF graph.

* If rank < V, then max length string = 2^r.

Date

E.g. S → AB
A → BC | a
B → C | b
C → a



No. of length of string:
S → 2^3 = 8
A → 2^2 = 4
B → 2^1 = 2
C → 2^0 = 1

③ Membership Problem: [CYK algorithm] → $O(n^3)$
- Also known as parsing.
- CYK algo is also k/a parsing algorithm.

NOTE → If the last box in the table having starting symbol of the grammar, then the given string is a member.

Eg. S → AB "aabbb" = ? Yes.
A → BB | a
B → AB | b

	a	a	b	b	b
→	A	A	B	B	B
→		S,B	A	A	
→	S,B	A	S,B		
→	A	S,B			
→	S,B				

Date

Push down Automata:
- FA with one stack.
- Size of stack is infinite.

$\delta : Q \times \Sigma \cup \{\varepsilon\} \times \Gamma \to Q \times \Gamma^*$ (transition function)

$(Q, \Sigma, q_0, F, \delta, Z_0, \Gamma)$
- Q: finite no. of states
- Σ: i/p alphabet
- q_0: initial state
- F: set of final states
- δ: transition function
- Z_0: initial stack element
- Γ: stack alphabet

→ Only one type of PDA i.e. a language recognizes.
→ NPDA > DPDA
→ By default PDA = NPDA.
→ PDA used in parsers.

- Two methods of acceptance
① Empty Stack ② Final state

NOTE
↳ No. of languages accepted by empty stack method is same to the no. of languages accepted by final state method in the case of PDA (NPDA).
→ No. of languages accepted by final state method is more in the case of DPDA.
→ PDA fails to accept languages which require more than one stack (e.g. $a^n b^n c^n$).
→ Expressive power of DPDA > F.A.
→ PDA fails to accept languages in which strings are not in A.P.
→ Over 1 symbol, F.A. = P.D.A., Hence, if for one symbol F.A. not possible, the PDA also not possible.

Date

* for $L = \{ww^R\}$, we cannot construct D-PDA, but we can construct N-PDA.

NPDA

Eg. $L = \{a^i b^j c^k \mid i=j \text{ or } j=k\}$

$$\delta = Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow 2^{Q \times \Gamma^*}$$

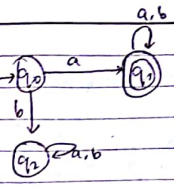
* NPDA > DPDA.

CFL
DCFL

NOTE
↳ Following languages are not CFL, but their complement is CFL ($\Sigma^* - L$)
- $L = \{wcw \mid w \in (a,b)^*\}$
- $L = \{ww \mid w \in (a,b)^*\}$

Equivalence Class:
- No. of equivalence classes corresponding to given regular language is equal to the no. of states of minimal DFA for that language.
- Equivalence class for a set 'q' is the set of all strings from which we can reach state 'q' from initial state.
- Union of equivalence classes of all final states is the regular language corresponding to the given DFA.

Date



Equivalence class $(q_0) = \{\epsilon\}$
Equivalence class $\{q_1\} = a(a+b)^*$
Equivalence class $\{q_2\} = b(a+b)^*$

Detection of CFL:
1. All finite languages are CFLs.
2. Any infinite lang. requiring more than one ~~comparison~~ are non-CFL.
   All palindrome languages are CFL.

CFLs:	Non-CFLs:
1. Finite languages.	1. Infinite language requiring more than one comparison.
2. All palindrome languages.	2. strings not in A.P.
3. Union of two CFLs.	3. Infinite language over 1 symbol but not satisfying stack property.
4. Concatenation of two CFLs.	

Date

TURING MACHINE:

- Represents behaviour of a general purpose computer.

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

Q: Finite no. of states; Σ: i/p alphabet; q_0: initial state; F: set of final states; δ: Transition fn; B: Blank symbol; Γ: Tape alphabet

$$\delta: Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$$

*Lang accepted by T.M. is REL.

Recursive Language:

- T.M. always halts on all inputs.
- For valid strings, T.M. halts in final state.
- For invalid strings, T.M. halts in non-final state.
- If language is recursive, then, it is decidable.
- Also k/a Turing Decidable Language.

Recursive Enumerable language:

- TM halts on some i/ps and may not halts on some inputs.
- If a lang. is REL, TM halts in final state.
- If a lang. is non-REL, TM may halt in non-final state or enter into infinite loop.
- If a lang. is RFL, it's undecidable.
- Also known as Turing Recognizable language.
- By default, TM is halting/not halting, i.e., RFL.

Date

REL (TM, partially decidable, (Ambiguity of language))
↑
Recursive (HTM, Decidable, (Finiteness of DFA))
↑
CSL (LBA, $a^n b^n c^n$)
↑
CFL (PDA, ww^R)
↑
DCFL (DPDA, $a^n b^n$)
↑
Regular (FA, $a^n b^m$)
↑
Finite (a, b)

Set

- Countable
  - Countably finite: $L = \{a, b, ba\}$
    - → One to one correspondence with natural set.
    - → All decidable languages.
    - → Set of all languages accepted by T.M.
    - → Reg. languages.
  - Countably Infinite: $L = \{\epsilon, a, b, aa, \ldots b\} = \Sigma^*$
- Uncountable (more than natural nos.)
  - ① Set of real nos.
  - ② Power set - 2^N
  - ③ 2^{Σ^*} / $P(\Sigma^*)$
  - → No one to one correspondence with natural set.
  - → No TM can be constructed.
  - → Not REL
  - → Set of all languages NOT accepted by T.M.
  - → No. of non-REL

Date

Undecidability:

- Language of a problem is decidable, if there exists a H.T.M. for it.

* **Reduction:** A problem 'A' is reducible to 'B', we can conclude the problem 'B' (new problem) with the help of problem 'A' (known problem).

A ∝ B
UD → U.D
D ← D

If A is reducible to B, then:

1. B is decidable, then, A is also decidable.
2. A in undecidable, then, B is also undecidable.
3. B is ~~reducible~~, recursive, then, A is also recursive.
4. A is not recursive, B is also not recursive.
5. B is REL, then, A is REL
6. A is non-REL, then, B is non-REL.
7. A is not solved, then B is not solved.
8. B is solved, then, A is solved.

* Post correspondence Problem is undecidable.

~~Decidability~~:

Date

Decidability:

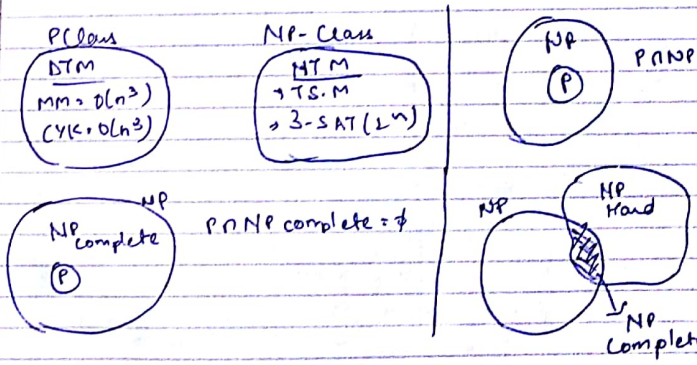
* **P-Class Problems:** DTM or Deterministic algo exists.

* **NP-Class:** NTM or Non-deterministic algo exists.

* **NP-complete:** · completely belong to NP-class.
· P ∩ NP complete = ϕ.

* **NP-Hard:** Problems as hard as NP-class.
· Time complexity may be greater than NP-complete.

→ If any NP-Hard belongs to NP, then it is NP complete.



P ≠ NP is unknown

18/07/2017.

THEORY OF COMPUTATION (60 hours) (Max 10 marks)

Definition: It is the mathematical study of computing machines and their capability.

(or)

It is the study of automata theory and formal languages.

50% — Finite Automata and Regular Languages
30% — Pushdown Automata and context free languages
20% — Linear bounded automata and context sensitive languages
— Turing machine and Recursive Enumerable languages
— Undecidability.

Aphabet (Σ): Any finite nonempty set of symbols.

E.g. $\Sigma = \{0, 1\}$
$\Sigma = \{a, b, c\}$
$\Sigma = \{$ ☺, .. ☹ $\}$

String : finite sequence of symbols over the given alphabet.

E.g. ab → Length = 2

Epsilon (ϵ) = Length = 0 Zero length string.

Language: Any set of strings over the given alphabet (Σ).

$\Sigma = \{0, 1\}$ ⇒ $L = \{0, 1, 00, 11\}$ ⇒ Finite Language
$L = \{0, 1, 00, 01, 10, \text{----}\}$ ⇒ Infinite language.
$L = \{\}$ ⇒ Empty language
$L = \{\epsilon\}$ ⇒ Finite Language
$\Sigma^* = L = \{\epsilon, 0, 1, 00, 01, 10, 11, \text{....}\}$ ⇒ Complete Language
All possible strings of the given alphabet exist.

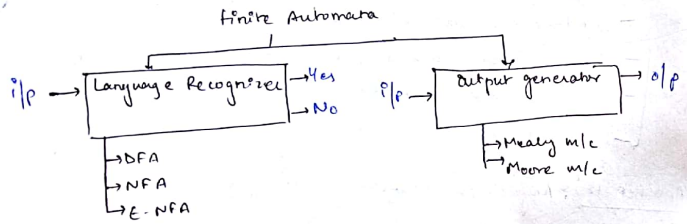
Finite Automata:

→ Finite automata is a mathematical model which contains finite number of states and transitions exist between the states.

→ There are two (2) types of F.A. known as:

① Language Recognizer ② Output Generator.

→ Finite automata can be represented by using transition diagram and transition table.



Deterministic Finite Automata (D.F.A):

→ It is a F.A, in which from each and every state, on every input symbol exactly one transition should exist.

Formal Definition: (Q, Σ, q_0, f, s)

$Q \rightarrow$ finite no. of states

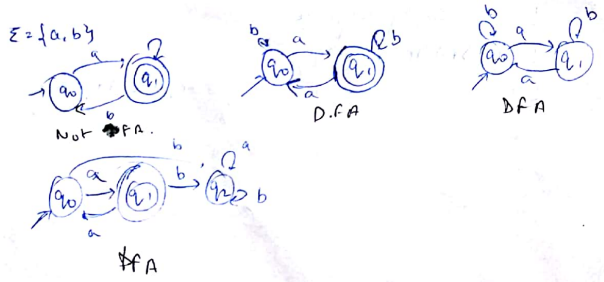
$\Sigma \rightarrow$ Input alphabet

$q_0 \rightarrow$ initial state (one initial state)

$f \rightarrow$ set of final states (can be zero)

$s \rightarrow$ Transition function

$Q \times \Sigma \rightarrow Q$



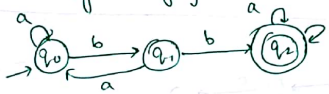
Acceptance Method:

→ Every DFA represents exactly 1 language.

→ Set of all strings accepted by a particular DFA is/are language of the D.F.A.

→ By reading any string from left to right, if the DFA halts in final state, then, that string is accepted, otherwise, string is rejected.

Ques. Identify language accepted by the following DFA:



→ Substring "bb"

set all strings

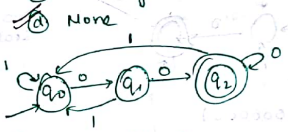
ⓐ starts with bb

ⓑ ends with bb

ⓒ containing atleast 2 b's

ⓓ None

Ques.



ⓐ Ending with 0

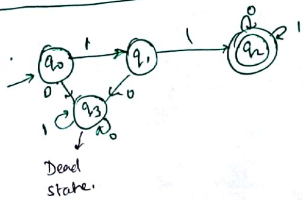
~~ⓑ Ending with 1~~

ⓒ Starting with 0

ⓓ Substring 00

ⓔ None. Ending with 00

Ques.



ⓐ $L = \{a^n b^m \mid n, m \geq 1\}$

ⓑ $L = \{a^n b^m \mid \begin{matrix} n \geq 0 \\ m \geq 1 \end{matrix}\}$

ⓒ $L = \{a^n b^n \mid n \geq 0\}$

~~ⓓ None~~

$a^n b^m$; $\begin{matrix} n \geq 0 \\ m \geq 0 \end{matrix}$

* In any DFA if the initial state is also the final state then ε is also accepted.

Ques.

① Language accepted.

ⓐ starting with 11

ⓑ Ending with 11

ⓒ Substring 11

ⓓ None

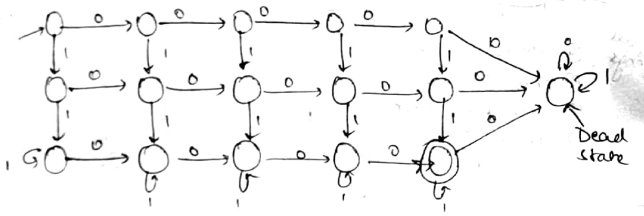
How many n-length strings are accepted.

(a) $2^n \mid n \geq 1$
(b) $2^{n-1} \mid n \geq 1$
(c) $2^{n-2} \mid n \geq 2$
(d) None.

(11) ⇒ 1
n=3 110, 111] → 2
n=4 1100, 1101, 1110, 1111] → 4

→ In any DFA, if any non-final state having self loop on all i/p symbols, that state is known as dead state.
→ By reading any string, if automata enters into dead state, that string is rejected.

Ques.

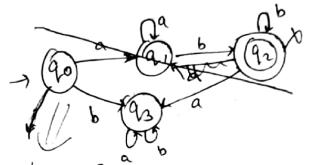


(a) Length of string at least 6. (000000)
(b) No. of zeroes exactly 4 and no. of 1s exactly 2. (000011)
(c) No of zeroes atleast 4 and no. of 1s atleast 2. (00000111)
(d) No. of zeroes exactly 4 and no. of 1s atleast 2.

DFA to Lang. & Lang to DFA :

Ques. Construct DFA for the language

$L = \{a^n b^n \mid n \geq 1\}$ = ab, aabb, aaabbb



DFA Not possible.

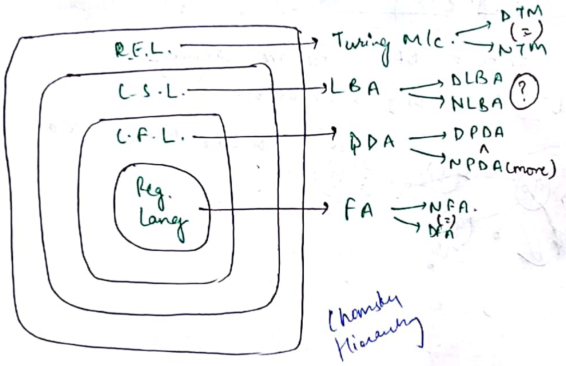
$\{a^n b^m \mid n, m \geq 1\}$

NOTE

→ Finite automata fails to accept languages in which comparison exists between symbols [If comparison exists, memory is required]
→ The languages for which F.A. is possible is known as Regular language.
→ The language for which F.A. is not possible known as non-regular language.

→ Finite automata having one stack is known as Push Down Automata.
→ The languages for which PDA not possible can be recognized by Turing M/c.

Q. (a) $L = \{a^n b^{2m} c^{3n} d^m\}$ which is regular?
(b) $L = \{a^n b^n c\, d^n e^m\}$
(c) $L = \{a^n b^{m+m} c^m\}$
(d) None.



Expressive Power:

→ No. of languages accepted by a particular automata is known as expressive Power of that automata.

→ PushDown Automata is powerful than F.A.

→ LBA is powerfull than F.A. and PDA.

→ TM is powerful than FA, PDA and LBA.

TM > LBA > PDA > FA

Q. Which of the following languages are not accepted by F.A.?

(a) All CFLs
(b) All CSL
(c) All REL
(d) All the above. ✓

→ Expressive Power of DFA and NFA is same. Hence, every NFA is converted into DFA.

→ Expressive Power of NPDA is more than DPDA. Hence, every NPDA cannot be converted into DPDA.

→ Expressive Power of DTM and NTM is same. Hence, every NTM is converted into DTM.

→ Expressive power of DLBA and NLBA is unknown.

Note → By default finite automata is NFA.
" " PDA is NPDA.
" " TM is DTM.

19/07/2017:

Minimization of DFA:

→ For one regular language, many number of DFA can be constructed but minimal state DFA is exactly one.

→ To construct minimal DFA from the given DFA following two algorithms exist:
① State equivalence method ② Table filling method.

→ To construct minimal DFA, equivalent states are detected and combining them into a single state.

* Equivalent States: Two states q_1 and q_2 are said to be equivalent, $\forall x \in \Sigma^*$, both, transition of q_1 on x $\delta(q,x)$ and transition of q_2 on x $\delta(q,x)$ going to either final or non-final.

→ Otherwise, they are not equivalent.

→ Minimization algorithms also detect equivalent states.

→ Minimization algorithms, eliminate all inaccessible states from the initial state,

→ Before applying minimization algorithms, eliminate all inaccessible states from the given DFA.

→ If any state is not reachable from the initial state, then it is known as inaccessible state.

→ Minimization algorithms are applicable only for DFA. [Not applicable for NFA and ε-NFA]

Q. Construct minimal DFA equivalent to following DFA :

	a	b
→q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
(q_4)	q_1	q_2

Soln :

① $\{q_0, q_1, q_2, q_3\}\{q_4\}$

② $\{q_0, q_1, q_2\}\{q_3\}\{q_4\}$

③ $\{q_0, q_2\}\{q_1\}\{q_3\}\{q_4\}$

④ $\{q_0, q_2\}\{q_1\}\{q_3\}\{q_4\}$

Q.

	0	1
→A	B	C
B	A	D
C	E	C
D	E	C
(E)	F	C
F	F	C

① $\{A, B, F\}\{C, D, E\}$

② $\{A, B\}\{F\}\{C, D, E\}$

③ $\{A, B\}\{F\}\{C, D\}\{E\}$

Q.

	0	1
→A	B	C
B	A	D
C	E	C
D	E	C
(E)	F	C
F	F	C

① $\{A, B, F\}\{C, D, E\}$

② $\{A, B\}\{F\}\{C, D, E\}$

③ $\{A, B\}\{F\}\{C, D\}\{E\}$

Q.

	0	1
→q_0	q_1	q_2
q_1	q_1	q_3
q_2	q_1	q_2
q_3	q_1	q_4
(q_4)	q_1	q_2

① $\{q_0, q_1, q_2, q_3\}\{q_4\}$

② $\{q_0, q_1, q_2\}\{q_3\}\{q_4\}$

③ $\{q_0, q_2\}\{q_1\}\{q_3\}\{q_4\}$

Q.

	0	1
→q_0	q_1	q_3
q_1	q_2	q_4
q_2	q_3	q_4
(q_3)	q_4	q_4
(q_4)	q_4	q_4

① $\{q_0, q_1, q_2, q_3\}\{q_4\}$

~~② $\{q_0\}\{q_1, q_2\}\{q_3\}\{q_4\}$~~ ② $\{q_0\}$ $\{q_1, q_2, q_3\}\{q_4\}$

③ $\{q_0\}\{q_1, q_2, q_3\}\{q_4\}$

Q.

	a	b
→P	S	Q
Q	T	T
R	R	S
S	S	T
(T)	T	S

① $\{P, Q, R\}\{S, T\}$

② $\{P, Q\}\{R\}\{S, T\}$

③ $\{P\}\{Q, T\}\{R\}\{S, T\}$

	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
(q_2)	q_1	q_2

① $\{q_0\}, \{q_1, q_2\}$

	0	1
→ q_0	q_1	q_5
q_1	q_0	q_2
(q_2)	q_3	q_6
q_3	q_4	q_7
q_4	q_5	q_0
q_5	q_6	q_1
q_6	q_7	q_2
q_7	q_3	q_4

① $\{q_0, q_1, q_2\}$ $\{q_3\}$ [0 equivalence]

② $\{q_0, q_1\}\{q_2\}$ [1 - equivalence]

③ $\{q_0\}\{q_1\}\{q_2\}\{q_3\}$ [2 - equivalence]

	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
q_2	q_4	q_3
(q_3)	q_3	q_3
q_4	q_4	q_4

① $\{q_0, q_1, q_2\}$ $\{q_3, q_4\}$

② $\{q_0, q_1\}$ $\{q_2\}$ $\{q_3, q_4\}$

③ $\{q_0\}$ $\{q_1\}$ $\{q_2\}$ $\{q_3, q_4\}$

	0	1
→ q_0	q_1	q_5
q_1	q_2	q_6
(q_2)	q_0	q_7
q_3	q_1	q_5
q_4	q_2	q_6
q_5	q_3	q_7
q_6	q_4	q_0
q_7	q_5	q_1

① $\{q_0, q_4, q_5, q_6, q_7\}$ $\{q_1, q_2, q_3\}$

② $\{q_0, q_4, q_6\}$ $\{q_5, q_7\}$ $\{q_1, q_2, q_3\}$

③ $\{q_0, q_4\}$ $\{q_6\}$ $\{q_5, q_7\}$ $\{q_1, q_2, q_3\}$

	0	1
→ q_0	q_1	q_3
q_1	q_0	q_3
(q_2)	q_3	q_3
q_3	q_4	q_5
q_4	q_1	q_3
q_5	q_6	q_7
q_6	q_5	q_7
q_7	q_7	q_7

① $\{q_0, q_1, q_4\}$ $\{q_2, q_3, q_5, q_6, q_7\}$

② $\{q_0, q_1\}$ $\{q_4\}$ $\{q_2, q_3, q_5, q_6, q_7\}$

③ $\{q_0\}$ $\{q_1\}$ $\{q_4\}$ $\{q_2, q_3, q_5, q_6, q_7\}$

	0	1
→ q_0	q_1	q_{01}
q_1	q_0	q_{23}
(q_2)	q_3	q_{23}
(q_3)	q_{23}	q_{23}
q_{01}	q_{01}	q_{23}

① $\{q_0, q_1\}$ $\{q_2, q_3\}$

② $\{q_0, q_1\}$ $\{q_2, q_3\}$

	0	1
→ q_0	q_1	q_5
q_1	q_6	q_2
(q_2)	q_0	q_2
q_3	q_7	q_6
q_4	q_5	q_7
q_5	q_2	q_6
q_6	q_6	q_4
q_7	q_6	q_2

	0	1
→ q_0	q_1	q_5
q_1	q_2	q_6
(q_2)	q_3	q_7
q_3	q_4	q_0
q_4	q_5	q_1
q_5	q_6	q_2
q_6	q_7	q_3
q_7	q_0	q_4

① $\{q_0, q_2, q_4, q_6\}$ $\{q_1, q_3, q_5, q_7\}$

② $\{q_0, q_4\}$ $\{q_2, q_6\}$ $\{q_1, q_3, q_5, q_7\}$

③ $\{q_0\}$ $\{q_4\}$ $\{q_2, q_6\}$ $\{q_1, q_3, q_5, q_7\}$

	0	1
→ q_0	q_1	q_{23}
q_1	q_0	q_{23}
(q_{23})	q_{23}	q_{23}

① $\{q_0, q_1\}$ $\{q_{23}\}$

② $\{q_0, q_1\}$ $\{q_{23}\}$

NOTE:

→ In any minimal DFA, if all states are final, the DFA accepts complete language.

→ In any minimal DFA, if there is no final state, that DFA accepts empty language.

→ Empty language is a regular language, because we can construct a DFA, which doesn't have final states.

→ By default, DFA is minimal DFA.

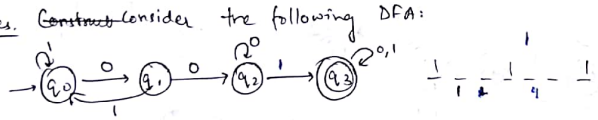
Complement of DFA:

→ We can complement DFA only [we cannot complement NFA & ENFA]

→ By interchanging final states as non-final states and non-final states as final states, we can complement a DFA.

NOTE: Language accepted by a DFA is L, then complement of the DFA accepts $\boxed{\Sigma^* - L}$

Ques. ~~Construct~~ Consider the following DFA:



① Let 's' denotes a 7-bit binary string in which first bit 4th bit and last bit is 1. How many number of strings are accepted by given DFA from S.

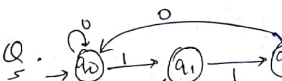
(a) 16 (b) 7 (c) 8 (d) 9

② Complement the above DFA. How many no. of strings complemented DFA accepts from S.

(a) 16 (b) 7 (c) 8 (d) 9

1001001 | 1011001 | 1101001 | 1111001
1001011 | 1011011 | 1101011 | 1111011
1001101 | 1011101 | 1101101 | 1111101
1001111 | 1011111 | 1101111 | 1111111

① Ans. 7 ② ~~16~~ Total = 16
Ans = 16 - 7 = 9

Q. 

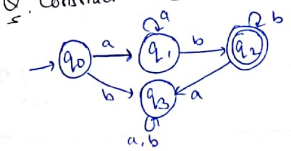
S: 1 - - 1 - -

1000100 | 1010100 | 1100100 | 1110100
1000101 | 1010101 | 1100101 | 1110101
1000110 | 1010110 | 1100110 | 1110110
1000111 | 1010111 | 1100111 | 1110111

Q. 's' denotes 6-bit binary string. 1st & 4th are 1.
How many no. of strings complemented DFA of the given DFA accepts from S.

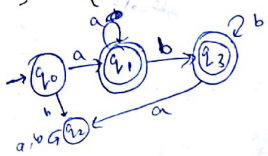
∴ complemented DFA = Not having substring 111
= 8

Q. Construct the minimal DFA for the language $L = \{a^n b^m ; n,m \geq 1\}$



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

Q. $L = \{a^n b^m \mid \begin{matrix} n \geq 1 \\ m \geq 0 \end{matrix}\}$



Q. $L = \{a^n b^m \mid n \geq 0, m \geq 1\}$

Q. $L = \{a^n b^m \mid n \geq 2, m \geq 0\}$

Q. $L = \{a^{2n} \mid n \geq 1\}$ aa, aaaa, aaaaaa, ...

Q. $L = \{a^{2^n} \mid n \geq 1\}$ {aa, aaaa, aaaaaaaa, ... infinite automata not possible}

Note: In any language, strings are not in alphabetic sequence, then that not possible for that language.
→ If the strings formed by symbols are not having common difference, then, that language is not regular.

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

Q. Each string ends with b.

Q. Starts with a & ends with b.
ab, aabb, abbb, aaabb, abbbb

Q. Each string starting with different symbol.

Q. Each string ending with a, b.

Q. First & symbols are same in every string.
aabba, bbaaab

Q. Last & symbols are same in each string.
aa, bb, abbabaa, abaabb

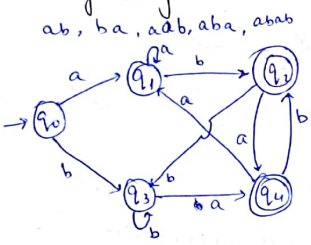
Note: The minimal DFA that accepts set of all string where each string ending with particular n-length string requires n+1 states

Q. Each string starting and ending with same symbol.
a, b, aa, bb, aba, bab, aaa, bbb

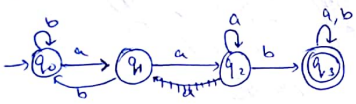
Q. Each string starting with a, b.

Q. Last 2 symbols are different in every string.

ab, ba, aab, aba, abab

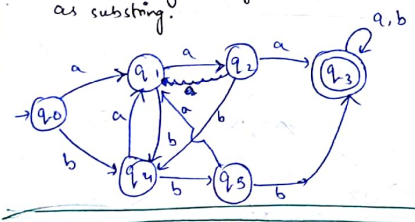


Q. "aab" as substring.



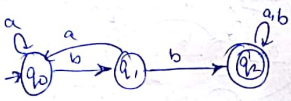
NOTE: The minimal DFA that accept n-length substring requires, n+1 states.

Q. Each string having "aaa" or "bbb" as substring.



NOTE: The minimal DFA that accepts all strings of a's and b's where nth i/p symbol is "a" while reading the string from LHS requires n+2 states.

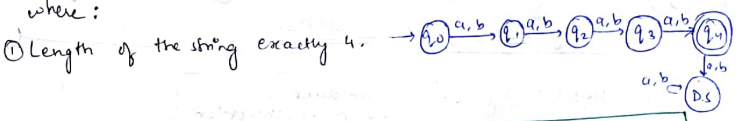
Q. Each string having "bb" as substring. abba abab



NOTE

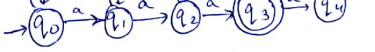
Q. Construct the minimal DFA that accept all strings of a's and b's where:

① Length of the string exactly 4.

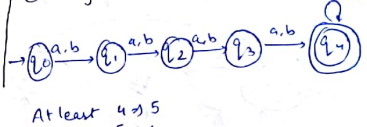


NOTE: The minimal DFA that accepts exactly n-length string requires n+2 states.

② No. of a's exactly 3. [Σ={a,b}]



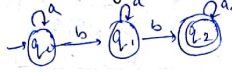
③ Length of the string atleast 4.



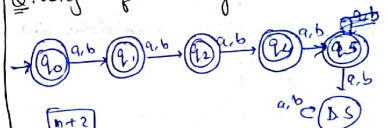
Atleast 4 → 5
5 → 6
n → n+1

NOTE: The minimal DFA that accepts atleast n-length string requires n+1 states.

Q. Each string having atleast 2 b's.



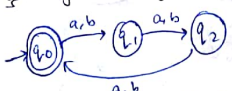
Q. Length of the string is atmost 4.



n+2

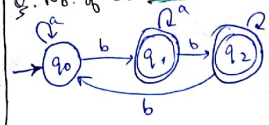
NOTE: The minimal DFA that accepts length of the string atmost "n" requires n+2 states.

Q. Length of string divisible by 3.



3 → 3 states
4 → 4
n → n states.

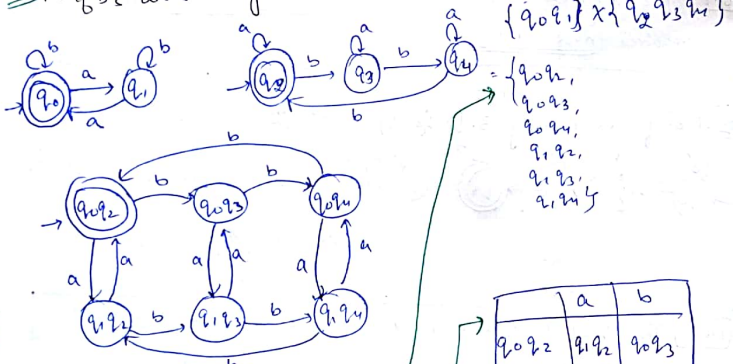
Q. No. of b's not divisible by 3.



- Construct b's divisible by 3
- Then Complement

NOTE The minimal DFA that accepts length of the string divisible by "n" requires n-states.

Q. No. of a's divisible by 2. And No. of b's divisible by 3. — Use product automata construction

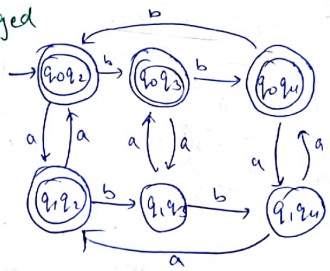


- Take cross product of states.
- ~~Then~~ Take single transitions and then merge them.

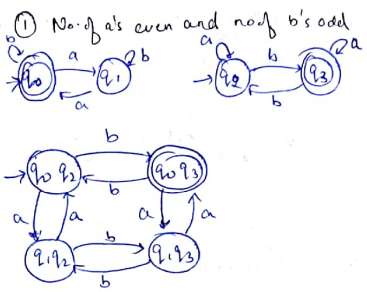
	a	b
q_0q_2	q_1q_2	q_0q_3
q_0q_3	q_1q_3	q_0q_4
q_0q_4	q_1q_4	q_0q_2
q_1q_2	q_0q_2	q_1q_3
q_1q_3	q_0q_3	q_1q_4
q_1q_4	q_0q_4	q_1q_2

Q. No. of a's divisible by 2. or No. of b's divisible by 3.

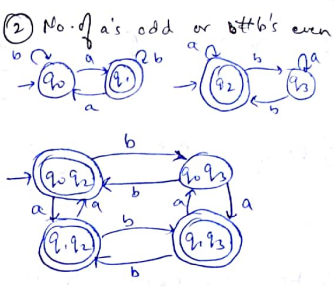
- All the process is the same as previous one (And condition)
- If any one states of the merged states is final.
- Make it as final.



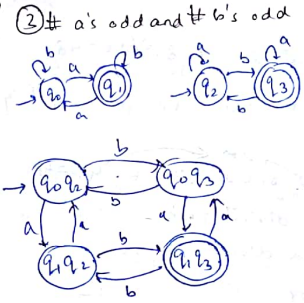
① No. of a's even and no. of b's odd



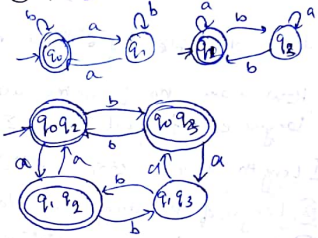
② No. of a's odd or b's even



③ # a's odd and # b's odd



④ # a's even or # b's even



Q. No. of states in ① a's divisible by 3 and b's not divisible by 4, $3 \times 4 = 12$

② a's divisible by 2 and b's at least 2 = $2 \times 3 = 6$.

③ a's atleast 2 or b's at least 3 = $3 \times 4 = 12$

④ a's atleast 3 and b's atmost 2 = $(n+2 = 4$ (including dead state)

Remove dead state.

$4 \times 3 = 12 + 1$ (Dead state)

$= 13$

④ a's exactly 2 and b's atmost 3.

$3 \times 4 = 12 + 1$ (Dead state)

$= 13$

- Remove 1-1 dead state
- Add only one dead state.

⑥ No. of a's exactly 3 or no. of b's not divisible by 2.

$4 \times 2 = 8 + 1$ (dead state)

$= 9$

⑦ No. of a's not exactly 3 and the no. of b's not exactly 2.

$4 \times 3 = 12 + 1$

$= 13$

Q. No. of 0's exactly 4 and no. of 1's atleast 2.

$n+2 = 6-1 = 5 \times 3 = 15+1 = 16$

Q. #a's divisible by 6 and no. of b's divisible by 8. $\Rightarrow 6 \times 8 = 48$

Q. How many no. of states are there in minimal DFA for the following languages formed over $\Sigma = \{a, b\}$.

① Length of the string divisible by 2 and divisible by 4. $= 4$ (LCM)

② Length of the string divisible by 2 or divisible by $= 2$ (GCD)

③ Length of the string divisible by 2 and divisible by $3 = 6$.

④ " " " " " 3 or " $4 = 12$

⑤ " " " " " 4 and " $6 = 12$

⑥ " " " " " 6 and " $8 = 24$

Divisible (m, n)

AND $\Rightarrow$ LCM OR $\Rightarrow$ GCD

GCD = 1, AND / OR $\Rightarrow m \times n$

GCD $\neq 1$, AND / OR $\Rightarrow$ LCM (m, n)

NOTE

→ No. of states of minimal DFA for condition given on length.

Eg: If m and n are given condition:

→ m divides n or n divides m, then no. of states of "AND" cond^n DFA is LCM (m, n).

∴ no. of states of "OR" cond^n DFA is GCD (m, n).

NOTE

→ m doesn't divides n or n doesn't divides m.

GCD $(m, n) = 1$

- then no. of states of "AND" cond^n DFA, no. of states of "OR" cond^n DFA $= m \times n$

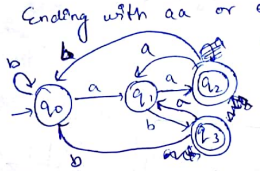
GCD $(m, n) \neq 1$

- then, no. of states of "AND" cond^n DFA, no. of states of "OR" cond^n DFA $=$ LCM (m, n).

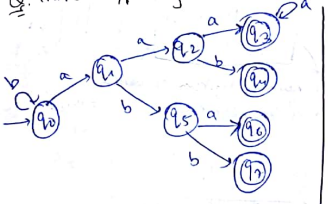
Q. How many no. of states are there in minimal DFA, that accept all strings of a's and b's, where 2nd i/p symbol is 'a' while reading the string from R.H.S.

ab, aa, aab, bab, ⋯⋯

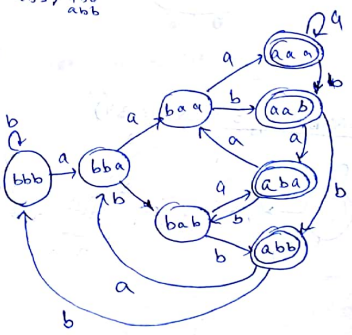
Ending with aa or ending with ab.



Q. Third i/p symbol is 'a' while reading the string from R.H.S.



(a's / b's): aaa, aab, aba, abb



NOTE → The minimal DFA that accepts all strings of a's & b's where n^{th} i/p symbol is 'a' while reading the string from R.H.S. requires 2^n states.

→ For some regular language, DFA construction is difficult, then NFA can be constructed easily.

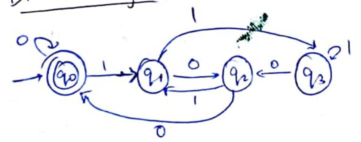
→ Construction of NFA is easy compared to D.F.A.

Q. How many number of states are there in minimal DFA that accept all binary numbers, which are:

① Divisible by 4 3 states.
② Divisible by 5 5 states
③ Divisible by 6 4 states.
④ Divisible by 8 4
⑤ Divisible by 10 6
⑥ Divisible by 12 5 states
⑨ Divisible by 16 5 states

① Divisible by 4 ⇒ ending with "00" = n+1 = 3 states
② Divisible by 5 ⇒

Divisible by 4:

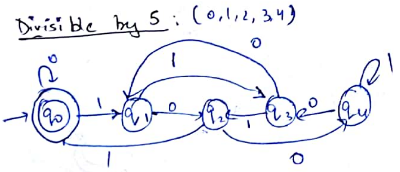


	0	1
→(q_0)	q_0	q_1
q_1	q_2	q_3
q_2	q_0	q_1
q_3	q_2	q_3

	0	1
q_0	q_0	q_1
$q_1 q_3$	q_2	q_3
q_2	q_0	q_1

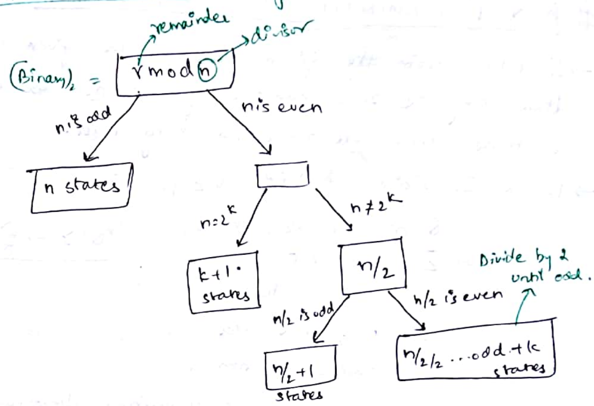
All number system transition tables are constructed like this.

Divisible by 5: (0,1,2,3,4)



	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

No. of states in minimal DFA : (Binary)



E.g. $\frac{10}{2}$ = 5+1 states | $\frac{12}{2}$ = $\frac{6}{2}$ = 3+2 = 5 states | 5 = 5 states.

Q. Binary numbers congruent to 3 mod ④ = 3 states

Q. " " congruent to 2 mod ⑥ = 4 states.

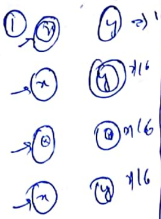
H.W

Q1. How many max^m no. of DFA can be constructed with 2 states x & y over the i/p alphabet Σ = {a, b}

Q2. How many maximum no. of finite automata can be constructed with 3 states x, y, z over Σ = {a, b}.

Q3. How many maximum no. of DFA can be constructed with n-states $(q_1, q_2, ... q_n)$ over Σ = {1, 2, 3, ... m} [m-symbols].

①



(y) ⇒ 16

	a	b
x	2	2
y	2	2

(x)(y) ⇒ 16

(x)(y) ⇒ 16 — 64 x is initial

(y) ⇒ 16 — 64 y is initial

2^7

②

(17,496)

③

	1	2	3	- -	m
q_1	n	n	n	- -	n
q_2	n	n	n	- -	n
⋮					
q_n	n	n	n	- -	n

2^n * n^{mn} ⇒ $n * 2^n * n^{mn}$

NON-DETERMINISTIC FINITE AUTOMATA:

→ In NFA, from the given state on the given input symbol, there may be zero number of transition or one transition or more than one transition may exist.

→ Formal Definition of NFA: $(Q, \Sigma, q_0, f, \delta)$

Q: finite number states
Σ: i/p alphabet
q_0: initial state
f: set of final states
δ: transition function

$$Q \times \Sigma \rightarrow 2^{Q} \text{ (or) } P(Q)$$
(Power set)

→ NFA can be represented in transition diagram and transition table.

→ Construction of NFA is easy compared to DFA.

→ The language recognition from NFA is difficult compared to DFA.

→ Minimization not possible for NFA. (Algorithm not applicable for NFA)

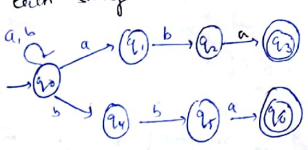
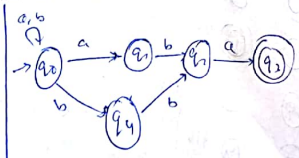
→ Complementation not applicable for NFA.

→ In NFA, for valid strings also machine may halt in non-final state also.

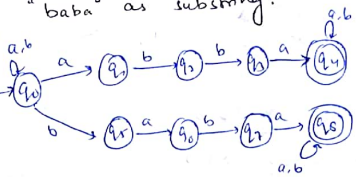
→ In NFA, for valid strings, even though multiple non-final states halting possibilities may exist but one final state halting possibility should exist.

→ All DFA are NFA, but all NFA need not be DFA.

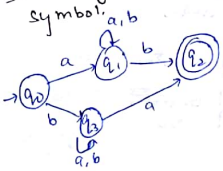
Q. Construct the NFA that accepts all strings all a's and b's where each string ends with either aba or bba.

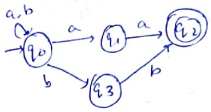
Q. Construct the NFA that accepts set of all strings of a's & b's each string having either "abba" or "baba" as substring.



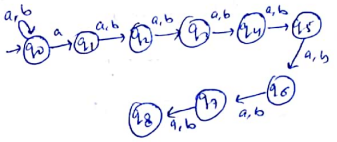
Q. Starting and ending with different symbol.



Q. Last two symbols are same. aa, bb,



Q. 8th i/p symbol is a.

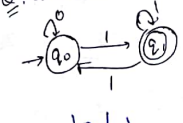


NOTE: The NFA that accepts all strings of a's and b's where n^{th} i/p symbol is 'a' while reading the string from R.H.S requires $n+1$ states.

→ Expressive Power of DFA and NFA is same, hence, every NFA is converted into DFA.

NFA to DFA: (Subset Construction algorithm)

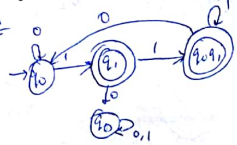
Q. Construct an equivalent DFA for the following NFA.



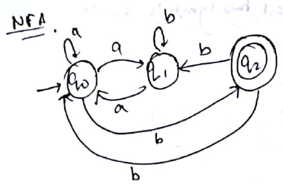
	0	1
→ q_0	$\{q_0\}$	$\{q_1\}$
q_1	-	$\{q_0, q_1\}$

	0	1
→ q_0	q_0	q_1
q_1	q_0	$[q_0q_1]$
q_0	q_0	q_0
$[q_0q_1]$	q_0	$[q_0q_1]$

DFA



NFA.

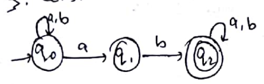


	a	b
→ q_0	$\{q_0,q_1\}$	$\{q_2\}$
q_1	$\{q_0\}$	$\{q_1,q_2\}$
(q_2)	—	$\{q_0,q_1\}$

may not be minimal

	a	b
→ q_0	$[q_0q_1]$	(q_2)
$[q_0q_1]$	$[q_0q_1]$	$[q_1q_2]$
(q_2)	q_0	$[q_0q_1]$
($[q_1q_2]$)	q_0	$[q_0q_1]$
~~$[q_0q_1q_2]$~~	~~$[q_0q_1]$~~	~~$[q_0q_1q_2]$~~
q_0	q_0	q_0

Q. Construct minimal DFA eq. to following NFA:



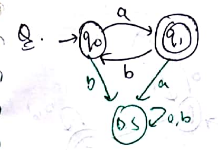
	a	b
q_0	$\{q_0,q_1\}$	q_0
q_1	—	q_2
(q_2)	q_2	q_2

	a	b
→ q_0	$[q_0q_1]$	q_0
q_0q_1	$[q_0q_1]$	$[q_0q_2]$
(q_0q_2)	$[q_0q_1q_2]$	$[q_0q_2]$
($q_0q_1q_2$)	$[q_0q_1q_2]$	$[q_0q_2]$

	a	b
→ q_0	q_0q_1	q_0
q_0q_1	q_0q_1	q_0q_2
$q_0q_1q_2$	$q_0q_1q_2$	q_0q_2

Minimal

Q.



Q	a	b
→ (q_0)	q_0	q_0
q_1	q_1	q_1
q_2	q_2	q_2

	a	b
→ (q_0)	q_1	q_0

When NFA because of missing transitions, add dead state.

NFA → DFA:

NOTE: While converting NFA into DFA, no. of states possible in DFA for the given n-states NFA is 1 to 2^n. (range).

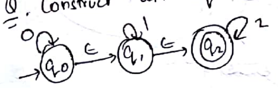
ε-NFA:

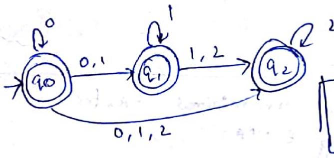
→ NFA having ε-transitions are known as ε-NFA.

→ Formal definition of ε-NFA is: $(Q, \Sigma, q_0, F, \delta)$

$$\delta: Q \times \Sigma \cup \{\epsilon\} \rightarrow 2^Q$$

Q. Construct an equivalent NFA without ε for the given NFA with ε-

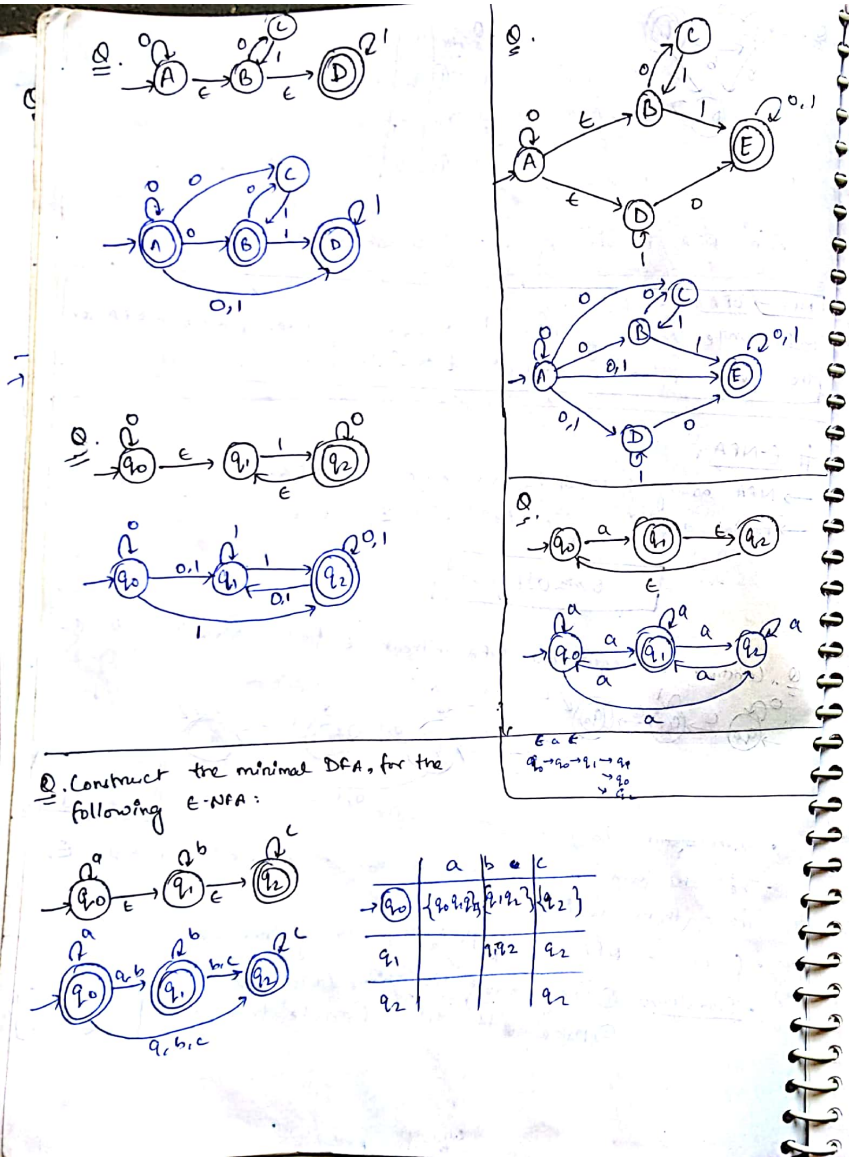




$\epsilon . 0 . \epsilon = 0$
$\epsilon 0 \epsilon = 0$
$\epsilon 1 . \epsilon \epsilon = 1$

- Same no. of states in NFA
- The states from which final state can be reached with only ε, make them as final states.
- ε-NFA → NFA : No. of final states may increase.
- Transitions: ① Copy direct transitions (visible)
  ② Make transitions possible through ε [invisible in ε-NFA]

Scanned by CamScanner



Q. Construct the minimal DFA, for the following ε-NFA:

	a	b	c
→ q_0	$\{q_0,q_1,q_2\}$	$\{q_1,q_2\}$	$\{q_2\}$
q_1		q_1,q_2	q_2
q_2			q_2

NOTE:

→ Epsilon-Closure of a state "q" is starting from that state by reading only ε all reachable states.

→ While converting ε-NFA into DFA, the initial state of DFA is ε-closure of initial state of NFA.

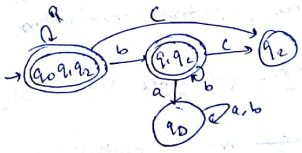
ε closure:

ε closure(q_0) : $\{q_0,q_1,q_2\}$

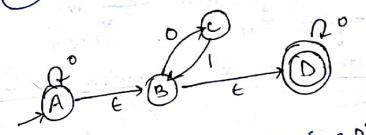
ε closure(q_1) : $\{q_1,q_2\}$

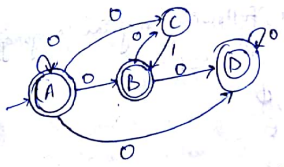
ε closure(q_2) : $\{q_2\}$

	a	b	c
→ $[q_0,q_1,q_2]$	$[q_0,q_1,q_2]$	$[q_1,q_2]$	q_2
$[q_1,q_2]$	q_D	$[q_1,q_2]$	q_2
q_2	q_D	q_D	q_2
q_D	q_D	q_D	q_D



Q. Min DFA :





ε closure (A) = [A, B, D]
ε closure (B) (B,D)
ε closure (C) : C
ε closure (D): D

	0	1
→ (A)	ABCD	–
(B)	CD	–
(C)	–	B
(D)	D	–

	0	1
→ (A B D)	ABCD	BCD
ABCD	ABCD	B
q_D	q_D	q_D
B	CD	q_D
CD	D	q_D
D	D	q_D

(22/07/2017)

Regular Expression:

→ The simplest way of representing a regular language is known as regular expression.

→ For every regular language, we can construct regular expression.

→ Every regular expression generates one regular language.

→ For the given regular language, we can construct regular expression by using symbols of input alphabet and following 3 operators.

* => ~~Union oper~~ Kleene Closure Operator
• => Concatenation operator
+ => Union Operator.

→ Concatenation operator is having higher precedence than union operator.

→ Kleene Closure operator is having higher precedence than concatenation operator.

→ For one regular language, there may be possibility of having many regular expressions.

→ we cannot construct regular expression for non-regular languages.

Q. Construct regular expression for the following regular language.

Language	Reg Exp.
(1) $L=\{\}$	ϕ
(2) $L=\{\epsilon\}$	ϵ
(3) $L=\{a\}$	a
(4) $L=\{a,b\}$	$a+b$
(5) $L=\{ab\}$	ab
(6) $L=\{\epsilon, a, aa, aaaa, \dots\}$	a^*
(7) $L=\{a, aa, aaaa, a\dots\}$	a^+ (positive closure)
(8) $L=\{a, b, aa, ab, ba, bb \dots\}$	$(a+b)^+$
(9) $L=\{\epsilon, a, b, ab, ba, abb \dots\}$	$(a+b)^*$
(10) $L=\{a^n b^m \mid n \neq m\}$	Not possible
(11) $L=\{a^n b^m \mid \begin{matrix} n\geq 0 \\ m\geq 1 \end{matrix}\}$	$a^* b^+$
(12) $L=\{a^n b^m \mid n,m \geq 0\}$	$a^* b^*$
(13) $L=\{a^n b^m \mid \begin{matrix} n\geq 2 \\ m\geq 3 \end{matrix}\}$	$aa^+ bbb^+$
(14) $L=\{a^n b^m \mid n>m\}$	Not possible
(15) $L=\{a^n b^m \mid n>m \text{ and } n<m\}=\{\}$	ϕ
(16) $L=\{a^n b^m \mid n\geq m \text{ and } n\leq m\}=a^n b^n$	Not possible
(17) $L=\{a^n b^m \mid n>m \text{ or } n<m\}=n\neq m$	Not possible
(18) $L=\{a^n b^m (n+m) \text{ is even}\}$ (Even a, Even b; Odd a, Odd b)	$(aa)^*(bb)^* + a(aa)^* b(bb)^*$
(19) $L=\{a^n b^m (n+m) \text{ is odd}\}$ (a is even, b is odd; a is odd, b is even)	$(aa)^* b(bb)^* + a(aa)^*(bb)^*$

Q. Construct the regular expression for the following regular languages formed over $\Sigma=\{0,1\}$.

(1) Each string starting and ending with different symbol.
(2) " " " " " same symbol.
(3) Last 2 symbols are same.
(4) Each string having "0101" as substring.
(5) Each string having either "000" or "111" as substring.
(6) 5th i/p symbol is "1" while reading the string from R.H.S.
(7) 4th i/p symbol is "0" while reading the string from L.H.S.
(8) Odd no. of 1's followed by even no. of 0's.

(1) $0(0+1)^*1 + 1(0+1)^*0$
(2) $0(0+1)^*0 + 1(0+1)^*1 + 0 + 1$
(3) $(0+1)^*00 + (0+1)^*11$
(4) $(0+1)^*.010.(0+1)^*$

⑤ $(0+1)^*000(0+1)^* + (0+1)^*111(0+1)^*$ | $(0+1)^*(000+111)(0+1)^*$

⑥ $(0+1)^*1(0+1)(0+1)(0+1)(0+1)$

⑦ $(0+1)(0+1)(0+1)0(0+1)^*$

⑧ $1(11)^*(00)^*$

Q. How many no. of states are there in minimal DFA that accepts following reg. exp.?
$(0+1)^*\,0001110(0+1)^*$ = 9 states.

Q. Length of string exactly 4. $(0+1)(0+1)(0+1)(0+1)$ 6 states.

Q. No. of 1's exactly 4. $0^*1\,0^*1\,0^*1\,0^*1\,0^*$

Q. Length of the string atmost 4. $(0+1+\epsilon)+(0+1+\epsilon)^2+(0+1+\epsilon)^3+(0+1+\epsilon)^4$
or $(0+1+\epsilon)^4$

Q. How many no. of states are there in minimal DFA that accepts following regular expression. $(0+1+\epsilon)^{n-2}$ (atmost n-2)
$(n-2)+2$ = n states.

Q. No. of 0's atmost 3. ~~$(0+\epsilon)^3(11)^*$~~ or $1^*(0+\epsilon)1^*(0+\epsilon)1^*(0+\epsilon)1^*$

Q. Length of string atleast 4. $(0+1)^4(0+1)^*$
or $(0+1)(0+1)(0+1)(0+1)(0+1)^*$ or $(0+1)(0+1)(0+1)(0+1)^+$

Q. Length of the string divisible by 3. $((0+1)(0+1)(0+1))^*$

Q. Length of the string even. $[(0+1)(0+1)]^*$

Q. Length of the string odd. $(0+1)[(0+1)(0+1)]^*$
If the string is starting with zero, then total length is odd.
or if the string is starting with one, then total length is even.
$0[(0+1)(0+1)]^* + 1[(0+1)(0+1)(0+1)]^*$

Palindrome :

Q. Construct the regular expression that generates all even length palindrome strings over the i/p symbol $\Sigma=\{a\}$.
$(aa)^*$ | $L=\{\epsilon, aa, aaaa, aaaaaa, \dots\}$

Q. All odd length palindrome strings over $\Sigma=\{a,b\}$
Regular Expression not possible.

Q. All even length palindrome strings over $\Sigma=\{a,b\}$
X

NOTE
→ All palindrome languages formed over more than 1 i/p symbol are non-regular.
Hence, regular expression not possible.
→ Palindrome languages formed over 1 symbol are regular.

Q Construct reg. exp. for all palindrome languages of English language. X

Q. " " " that generates set of all odd length palindrome strings of Hindi language. X

① $L_1=\{ww^R \mid w\in\{a\}^*\}$ $(aa)^*$

② $L_2=\{wxw^R \mid w\in\{a,b\}^*\}$ X
{a x a, b x b, ab x ba, abab x baba, ...}

③ $L_3=\{wxw^R \mid w\in\{a\}^*\}$ X
{x, axa, aaxaa, aaaxaaa}

④ $L_4=\{wxw^R \mid w\in\{a,b\}, x\in\{a,b\}^*\}$
$[a(a+b)^*a + b(a+b)^*b]$

⑤ $L_5=\{wxw^R \mid w,x\in\{a,b\}^*\}$
$\epsilon x \epsilon = x$
$x=\{\epsilon, a, b, ab, aa, ab, \dots\}$
$=(a+b)^*$
ab a ba already in $(a+b)^*$
Reg. exp. $=(a+b)^*$

⑥ $L_6=\{wxw^R \mid w,x\in\{a,b\}^+\}$
a x a
$x=\{a,b,ab,ba,aa,aba,\dots\}$
ababaabaababaababa
$a(a+b)^+a + b(a+b)^+b$

(7) $L = \{ww \mid w \in \{a,b\}^*\}$

aa Not possible.

(8) $L = \{ww^R x \mid w, x \in \{a,b\}^*\}$

$\epsilon \epsilon x \mid x \in \{\epsilon, a, b, ab, ba, \ldots\}$

aax

$(a+b)^*$

(9) $L = \{1, 2, 4, 8 \ldots 2^n\}$

All these numbers written in binary. $0^* | 0^*$

(10) $L = \{1, 2, 4, 8, \ldots 2^n\}$ in unary

1, 11, 1111, 11111111, ---

Not possible.

Identities of Regular Expression:

(1) $R + \phi = \phi + R = R$

(2) $R \cdot \phi = \phi \cdot R = \phi$

(3) $R \cdot \epsilon = \epsilon \cdot R = R$

(4) $R^* = (R^*)^* = (R^*)^+ = (R^+)^*$

(5) $R \cdot R^* = R^+ = R^* \cdot R$

(6) $R^+ + \epsilon = R^* = R \cdot R^* + \epsilon$

(7) $\epsilon^* = \epsilon$
$\epsilon^+ = \epsilon$

(8) $\phi^* = \epsilon$
$\phi^+ = \phi$

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

(10) $a \cdot b \neq b \cdot a$

(11) $(a+b)^* \neq a^* + b^*$

(12) $(a+b)^* \neq a^* b^*$

(13) $(a+b)^* \neq (ab)^*$

(14) $a^* \cdot a^* = a^*$

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

(16) $a^* \cdot a = a^+$

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

(18) $a(ba)^* = (ab)^* a$

(19) $a + a = a$

(20) $a \cdot a \neq a$

Q Consider the following 4 regular expressions and which of these are equal.

(I) $(0+\epsilon)(00)^*$

(II) $(00)^*$

(III) 0^*

(IV) $0(00)^*$

(a) I, II (b) II, III (c) I, III (d) II, IV.

Q. $b^*(a^* \cdot \phi \cdot b + ab + a \cdot \phi^* \cdot b^*)(b+\phi)^*$

$= b^*(\phi + ab + ab^*)(b)^*$

$= b^*(ab + ab^*) \cdot b^* = b^*(a(b+b^*))b^* \Rightarrow b^* a b^* \cdot b^* = b^* a b^*$

(a) exactly one a (c) almost one a

(b) atleast one a (d) none.

Q. Which of the following regular expressions are identical.

(1) $(a+ba)^*(b+\epsilon) = (a+ba)^* \cdot (b+\epsilon)$

(2) $(a^* \cdot (ba)^*)^*(b+\epsilon) + a^*(b+\epsilon) + (ba)^*(b+\epsilon) = (a+ba)^*(b+\epsilon)$

(3) $(a+ba)(a+ba)^*(b+\epsilon) = (a+ba)^+(b+\epsilon)$

(a) 1 and 2 (c) 1 and 3

(b) 2 and 3 (d) 1, 2, 3

Q. Consider the following 3 regular expressions:

$r_1 = 11(0+1)^* =$

$r_2 = (0+1)^* 11 =$

$r_3 = 11(0+1)^* 11 + 111 + 11 =$

(a) $r_1 = r_2 = r_3$

(b) $L(r_1) \cup L(r_2) = L(r_3)$

(c) $L(r_1) \subseteq L(r_3)$

(d) $L(r_1) \wedge L(r_2) = L(r_3)$ (intersection)

Q. $L_1 = a \quad L_2 = \phi$

$L_1^* \cdot L_2 \neq L_2^*$

$= a^* \cdot \phi + \epsilon$

$= \epsilon$

Q. Which of the following are not equal.

(a) $1(01)^*$ and $(10)^*1$

(b) $1^*1 = 11^*$

(c) $(ab)^* = a^*b^*$ ✓

(d) none.

Q. Which of the following reg. ex. generates set of all strings of 0's and 1's where each string not having "100" substring.

(a) $0^*(1^*0)^*$ ✗

(b) 0^*1010^* ✗

$L = \{\epsilon, 0, 1, 00, \ldots\}$

(c) $0^*1^*01^*$ ✗

(d) $0^*(10+1)^*$ ✓

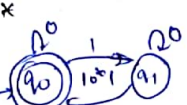
Q. Which of the following reg. expression generates set of all strings of 0's & 1's where no. of 1's are even in every string.

(a) $(0^*1 \cdot 0^*1 \cdot 0^*)^*$

(b) $0^*(10^*1)^*$

$L = \{\epsilon, 0, 11, 1101, \ldots\}$

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



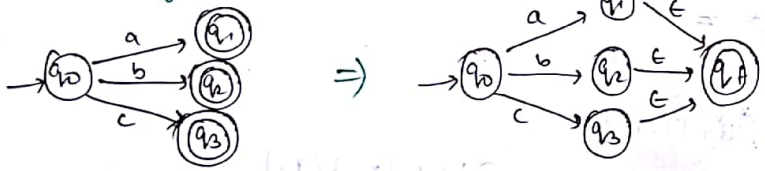
$= (0^* + 10^*1)^* = (0^* \cdot (10^*1)^*)^* = (0^*(10^*1)^*)^*$

(c) $(0^*(10^*1)^*)^*$ ✓

(d) $(0^*110^*)^* + 0^*$

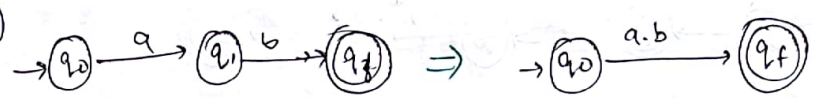
#F.A. to Regular Expression:

1.



2.

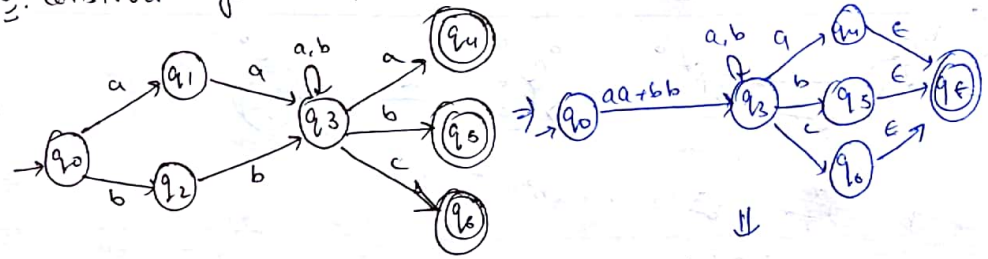
3.



4.

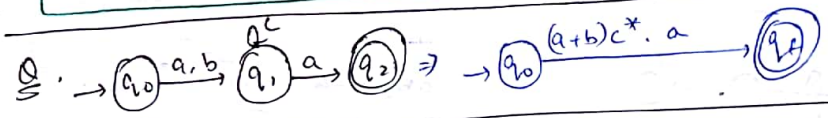
5.

Q. Construct regular expression equivalent to following F.A:

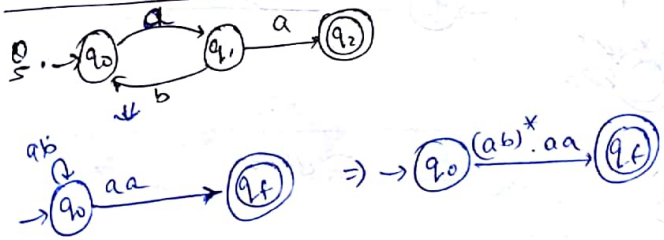


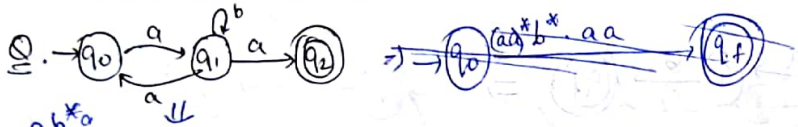
$(bb+aa)(a^*b)^*.(a+b+c)$

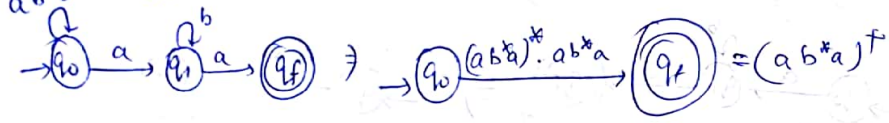
Q.

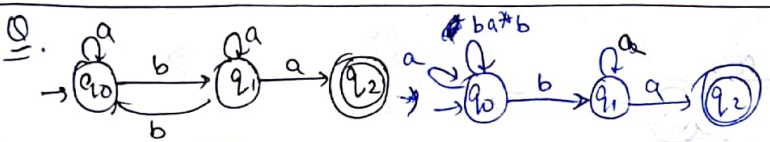


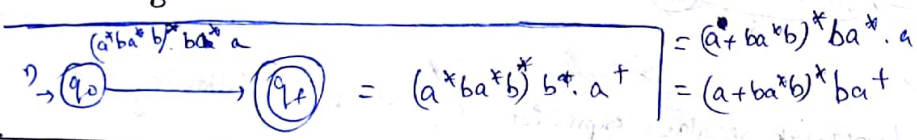
Q.

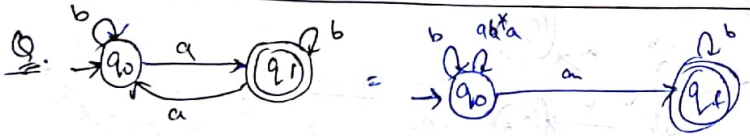


Q. 

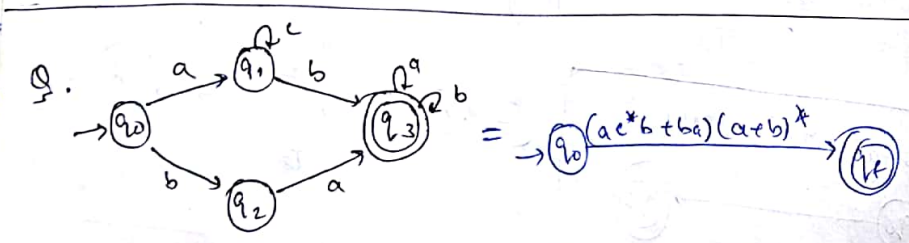


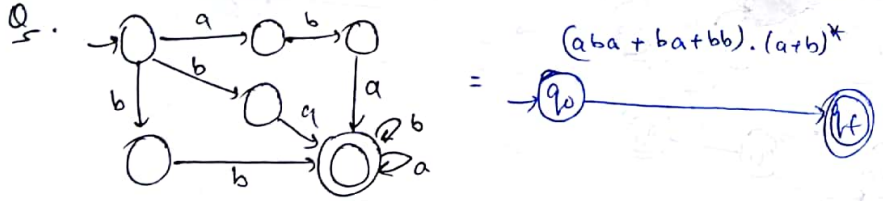
Q. 

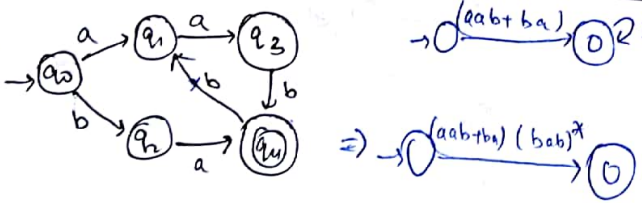


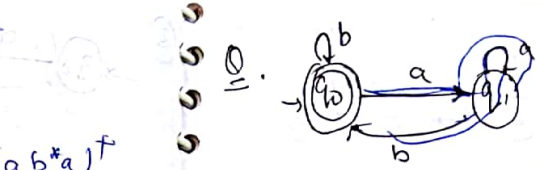
Q. 

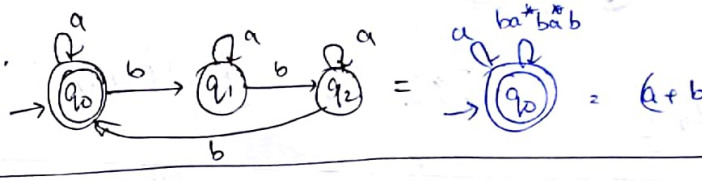
~~(b+aa)*a + b*~~ = (b+ab*a)*a b*

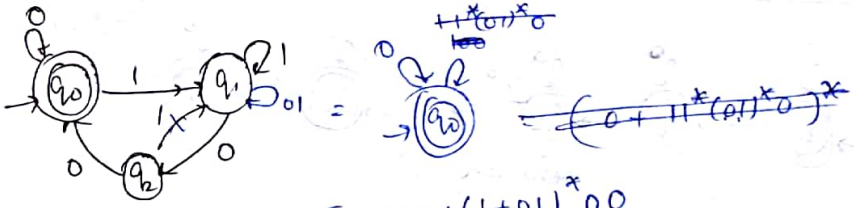
Q. 

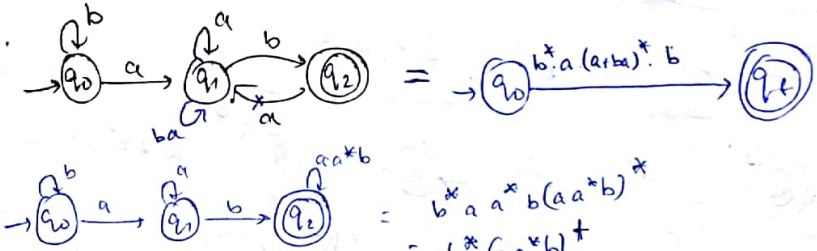
Q. 

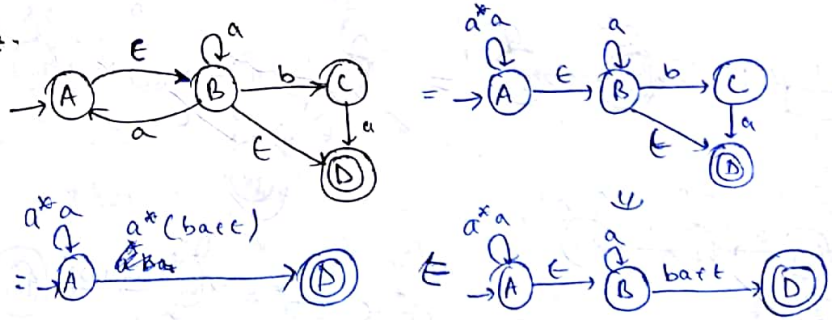
Q. 

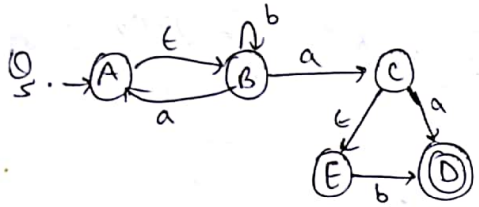
Q. 

Q. 

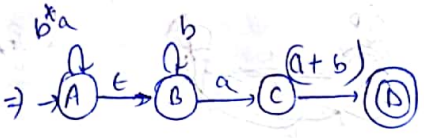
Q. 

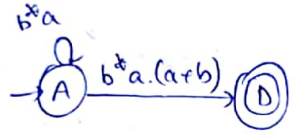
Q. 

Q. 

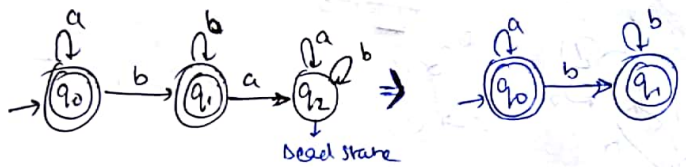
Q. 

b^*a

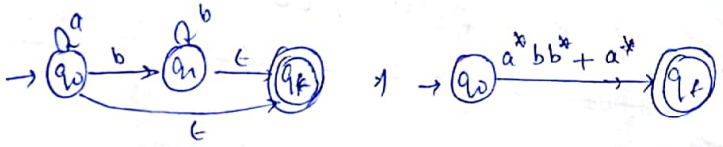


Q. ⇒ 

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

Q. 

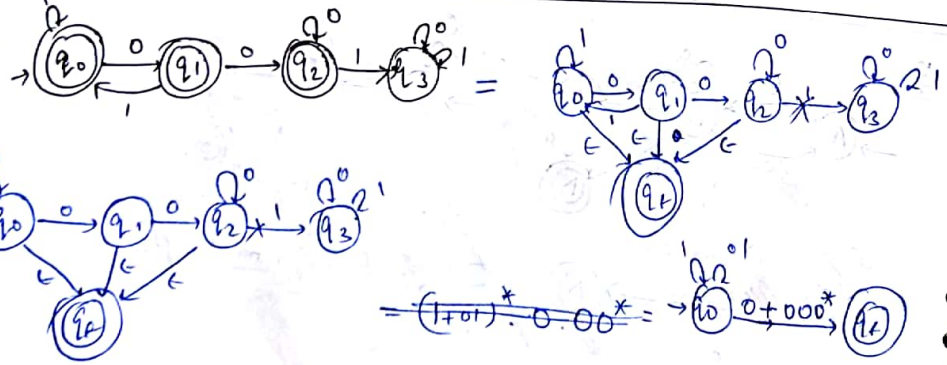
$a^* + b + bb^*$

Q. 

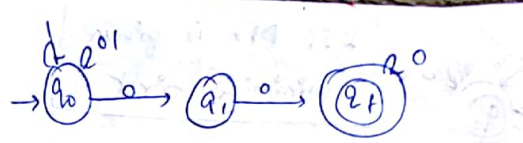
$\Rightarrow a^*b^+ + a^* = a^*(b^+ + \epsilon) \Rightarrow a^*b^*$

OR 

$a^* + a^*bb^* \Rightarrow a^*(\epsilon + b^+) = a^*b^*$

Q. 

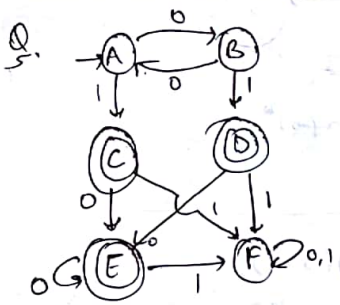
$= (1\cdot(01)^*0 + 00^* = (1\cdot(01))^*0$



$= (1+01)^* + (1+01)^*0 + (1+01)^*000^*$

$= (1+01)^*(\epsilon + 0 + 000^*) = (1+01)^*(\epsilon + 0 + 00^+)$

$= (1+01)^*0^*$

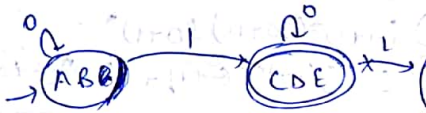
Q. 

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

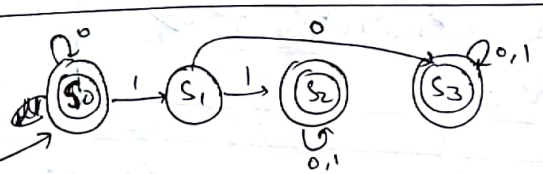
$= \{A,B,F\}\{C,D,E\}$

$= \{ABF\}\{C,D,E\}$

~~$\{AB\}\{F\}\{CDE\}$~~



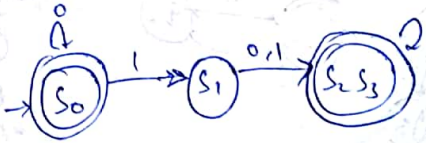
$= 0^*10^*$

Q. 

	0	1
→ S_0	S_0	S_1
S_1	S_3	S_2
S_2	S_2	S_2
S_3	S_3	S_3

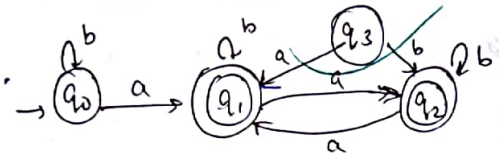
$= \{S_1\}\{S_0 S_2 S_3\}$

$= \{S_1\}\{S_2\}\{S_0\}\{S_2 S_3\}\{S_0\}$



$= 0^*1(0+1)(0+11)^*$

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

Q. 

※ If DFA is given minimize first

(a) b^*a^+

(b) $b^*a(ba)^*$

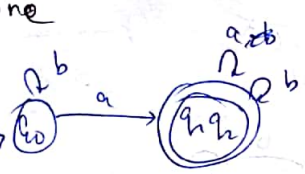
(c) $b^*(a+b)^*$

(d) none

	a	b
→ q_0	q_1	q_0
q_1	q_2	q_1
q_2	q_1	q_2

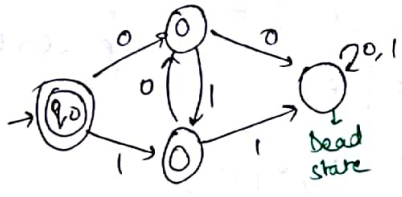
: $\{q_0\}\{q_1,q_2\}$

: $\{q_0\}\{q_1,q_2\}$



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

$=$

Q. 

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

(b) $(0(01)^*0+1(10)^*1)(0+1)^*$

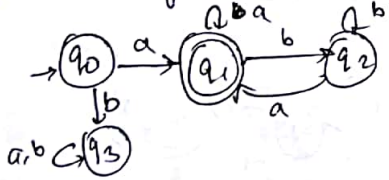
(c) $(0+1)^*(00+11)(0+1)^*$

(d) $\epsilon+0(10)^*(\epsilon+1)+1(01)^*(\epsilon+0)$

No. of 1's divisible by 2 : $(0^*10^*10^*)^*+0^*$

for 0000

Q. which of the following reg. ex. is not eq. to DFA.



(a) $aa^*(bb^*a)^*$

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

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

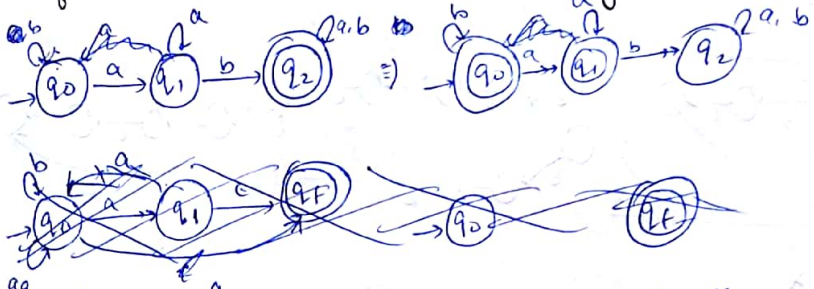
(d) $a(a+aa+bb^*a)^*$

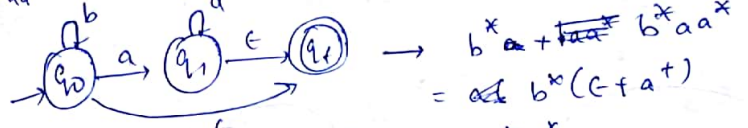
$= a(a+b)^*a+a$ (c)

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

$= a(a+aa+bb^*a)^*$ (d)

Q. Construct the regular expression that generates all strings of a's and b's where each string not having substring "ab".



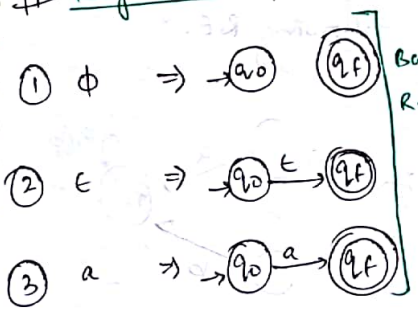


b^*aa^*

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

$= b^*a^*$

Regular Expression to F.A. (ε-NFA) : (Thomson Construction Algorithm)



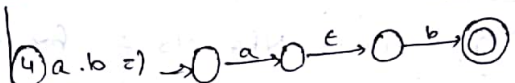
Basic R.E.

(1) ϕ ⇒

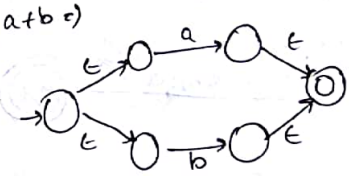
(2) ϵ ⇒

(3) a ⇒

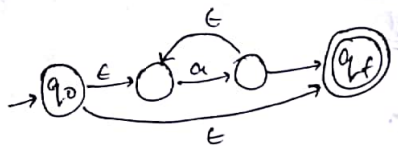
(4) $a.b$ ⇒



(5) $a+b$ ⇒



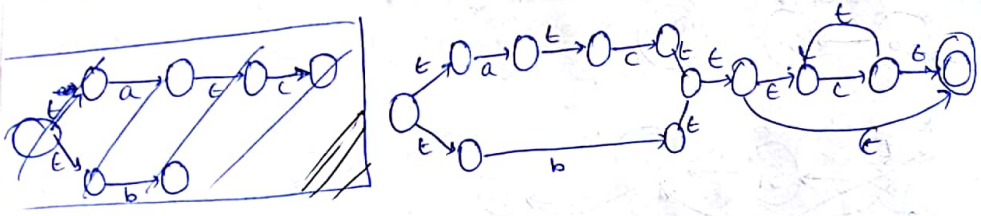
(6) a^* ⇒



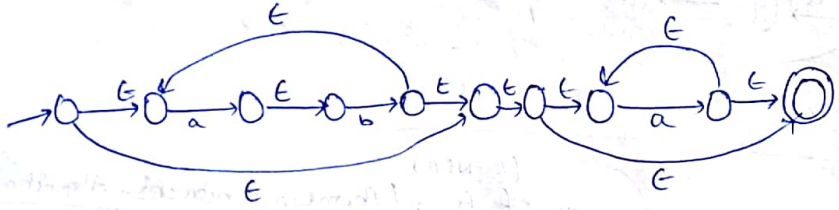
RE
↓ Thomson
ε-NFA
↓ subset
NFA
↓ subset
DFA.
↓
MinDFA

Q. Construct ε-NFA for following regular expression:

① $(ac+b).c^*$

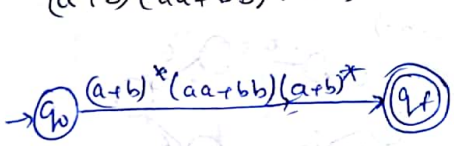


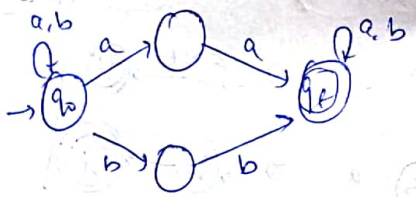
② $(ab)^*.a^*$

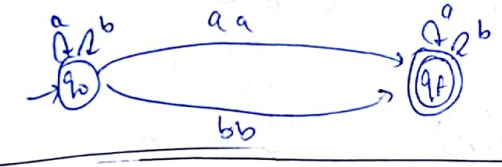


Q. Construct NFA w/o ε for the following R.E.:

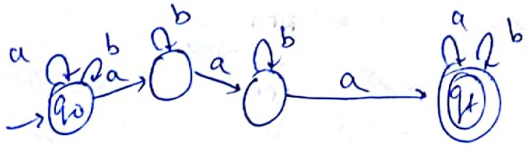
$(a+b)^*(aa+bb)(a+b)^*$



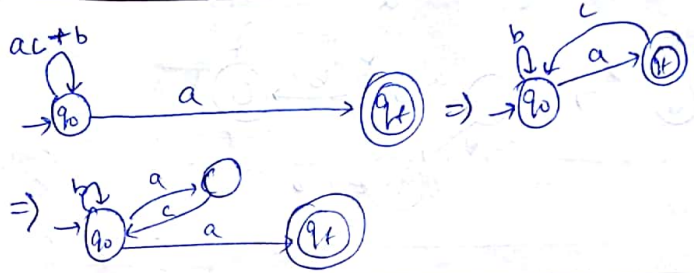




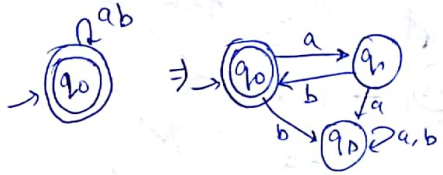
Q. $(a+b)^*ab^*ab^*a(a+b)^*$



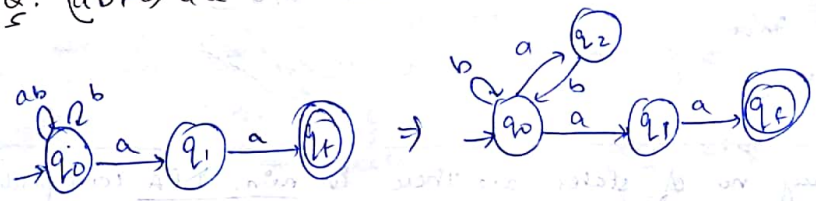
Q. Construct NFA: $(ac+b)^*.a$



Q. Construct Min. DFA. for $(ab)^*$



Q. $(ab+b)^*aa$

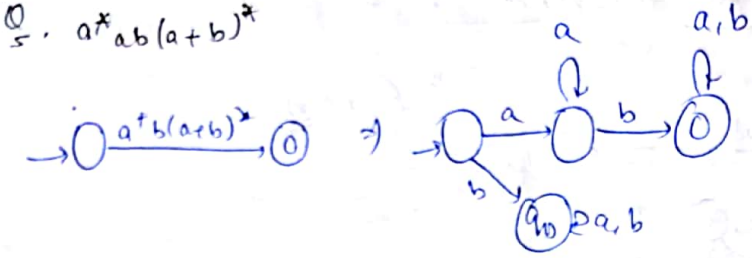


	a	b
→ q_0	q_1q_2	q_0
q_1	q_f	—
q_2	—	q_0
(q_f)	—	—

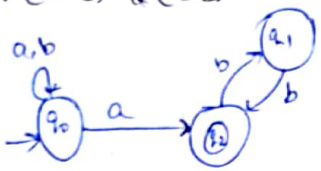
	a	b
→ q_0	$[q_1q_2]$	q_0
$[q_1q_2]$	q_f	q_0
(q_f)	q_D	q_D
q_D	q_D	q_D

4 states

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

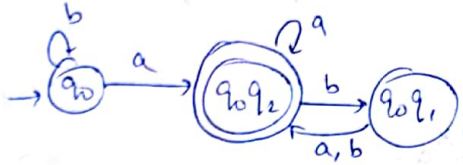


Q. $(a+b)^*a(bb)^*$



	a	b
→ q_0	q_0q_2	q_0
q_1	–	q_2
q_2	–	q_1

	a	b
→ q_0	(q_0q_2)	q_0
q_0q_2	q_0q_2	q_0q_1
q_0q_1	q_0q_2	q_0q_2



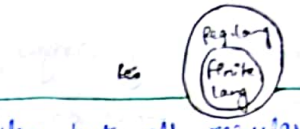
Q. How many no. of states are there in min. DFA corresponding to following R.E.:

1. $(a+b)^*abab$ = n+1 = 5 states
2. $(a+b)^*aba(a+b)^*$ = n+1 = 4 states.
3. $(a+b)^*\ b\ (a+b)(a+b) = 2^3$ = 8 states. (3^{rd} from R.H.S. is b)
4. $(a+b+\epsilon)^5$ = n+2 (atmost 5) = 7 states
5. $((a+b)^5)^*$ = div. by 5 = 5 states.
6. a^*b^* = (no. of symbols + 1)'s states = 3 states
7. $a^*b^*c^*$ = 4 states.
8. $a^*b^*c^*\ldots z^*$ = 27 states.
9. $0^*1^*2^*\ldots9^*$ = 11 states.

23/07/2017: Regular Languages

→ The language for which finite automata exists, is known as regular language.

→ The language for which regular expression exists known as regular language.



NOTE

→ All finite language are regular but all regular languages need not be finite.

Q. Construct the regular expression that generates set of all substrings of the string "TOC".

$\epsilon + T + O + C + TO + OC + TOC$ → Reg. exp. | $T + To + ToC + \epsilon$ (Prefix)

Q. Construct the regular expression that generates all prefixes of the string "gate".

$\epsilon + g + a + t + e + ga + at + te + gat + ate + gate$ (substrings) | $g + ga + gat + gate + \epsilon$ (Prefixes.)

Q. Construct the regular expression that generates all proper suffixes of the string "Delhi".

~~$\epsilon + d + e + l + h + i + de + el + lh + hi + t$~~ | $i + hi + lhi + elhi$

* Substring: Consecutive sequence of symbols over the given string, is known as substring.

NOTE: Total number of substrings possible for the given n-length string is $\frac{n(n+1)}{2} + 1$.

* Prefix: Sequence of leading symbols over the given string is known as prefix.

Eg. T, To, TOC + ϵ | g + ga + gat + gate + ϵ

NOTE: For n-length string, prefixes = n+1

*Suffix: Sequence of trailing symbols over the given string.

Eg: c, oc, Toc, ε | Proper suffix
i, hi, lhi, elhi, Delhi, ε | i, hi, lhi, elhi

Proper suffix/prefix/substring: Equality not present

[Not given n, m ≥ 1]

Q. Which of the following languages are regular. (Regular ✓, Non regular X)

1. $L = \{a^n b^m \mid n, m \le 10, n \ne m\}$ → Regular (finite) ✓
2. $L = \{a^n b^m \mid n < m \text{ and } n > m\}$ $L = \{\} = \phi$ → Regular ✓
3. $L = \{a^n b^{m+n} \mid n, m \ge 1\}$ X
4. $L = \{a^n b^m \mid n+m = 100\}$ ✓
5. $L = \{a^n b^m \mid n - m = 5\}$ X
6. $L = \{a^n b^m \mid n * m = 100\}$ ✓
7. $L = \{a^n b^m \mid n > m \text{ or } n < m\}$ X
8. $L = \{a^n b^m c^n d^m \mid n, m \ge 1\}$ X
9. $L = \{a^n b^{m+n} c^m \mid n, m \ge 1\}$ X
10. $L = \{a^n b^{2m} c^{5k} \mid n, m, k \ge 1\}$ ✓
11. $L = \{a^{n^2} b^{m^3} c^{5k} \mid n, m, k \ge 1\}$ X No AP
12. $L = \{a^n b^m \mid n/m = 4\}$ X
13. $L = \{a^{n!} b^{m!} \mid n, m \ge 1\}$ X
14. $L = \{a^{2^n} \mid n \ge 1\}$ X
15. $L = \{a^p \mid p \text{ is prime no.}\}$ X No AP
16. $L = \{a^k \mid k \text{ is odd no.}\}$ ✓
17. $L = \{a^{2n+1} \mid n \ge 1\}$ ✓
18. $L = \{a^{100^{100}}\}$ ✓
19. $L = \{a^n \mid n \le 10\}$ ✓
20. $L = \{ww^R \mid w \in \{a\}^*\}$ ✓ (Even no. of a's)
21. $L = \{waw^R \mid w \in \{b\}^*\}$ X Infinite Comparison
22. $L = \{wxw^R \mid w \in \{a, b\}^*\}$ X
23. $L = \{wxw^R \mid x \in \{a,b\}^+, w \in \{a,b\}\}$ ✓
 $a(a+b)^+a + b(a+b)^+b$
24. $L = \{wxw^R \mid w, x \in \{a, b\}^+\}$ ✓
 Starting & ending with same symbol
25. $L = \{ww^Rx \mid w, x \in \{a, b\}^+\}$ X
26. $L = \{xww^R \mid w, x \in \{a, b\}^+\}$ X
27. $L = \{ww \mid w \in \{a\}^*\}$ ✓ $(aa)^*$
28. $L = \{wxw \mid w, x \in \{a, b\}^*\}$ ✓
29. $L = \{wxw \mid w \in \{a, b\}^*\}$ X
30. $L = \{w(w^R)^* \mid w \in \{a, b\}^*\}$
31. $L = \{ww^Rww^R \mid w \in \{a, b\}\}$ ✓ finite
32. $L = \{waY \mid w, Y \in \{a, b\}^*\}$ ✓
33. $L = \{x \mid x \in \{a, b\}^*, n_a(x) > n_b(x)\}$ X
34. $L = \{x \mid x \in \{a, b\}^*, n_a(x) = n_b(x) + 2\}$ X
35. $L = \{x \mid x \in \{a, b\}^*, n_a(x) \bmod 3 = 0 \text{ and } n_b(x) \bmod 5 = 0\}$ ✓ 15 states.
36. $L = \{x \mid x \in \{a, b\}^*, n_a(x) \bmod 3 = n_b(x) \bmod 3\}$ ✓
37. Set of all even length palindrome strings of English language. X
38. Set of all odd length palindrome strings of Hindi language. X
39. Set of all balanced paranthesis ((())). X
40. Total population of the world. Finite. No comparison. Regular.

Points to Remember:

→ All finite languages are regular languages.

→ Any infinite language requires infinite memory (comparison exists b/w symbols), then it is not regular.

→ Any infinite language strings are not in arithmetic progression, then it is not regular.

→ All palindrome languages over 1 symbol are regular and more than 1 symbol (>1) are not regular.

→ Finite automata having finite memory. (remembering by using states).

(30) $w(w^R)^* \mid w \in \{a,b\}^*$

$ab(ba)^*$, $(w^R)^R = w$

∴ $a \; w \to \lambda \quad L = \{\epsilon, a, b, aa, b \ldots\} = \Sigma^*$

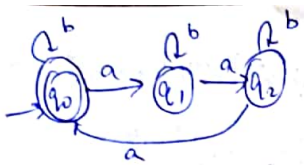
Regular.

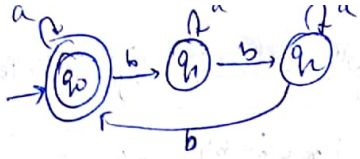
(36) $x \mid x \in \{a,b\}^*$, $n_a(x) \bmod 3 = n_b(x) \bmod 3$

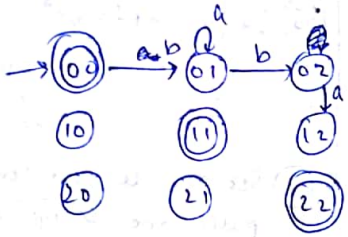
~~No. of~~ Remainder of $n_a(x)$ = Remainder of $n_b(x)$

Remainder = $\{0, 1, 2\}$ finite (Regular)

(26)







Q. Which of the following language is regular?

✗ (a) $L = \{1, 2, 4, 8 \ldots 2^n \ldots\}$ all the no. are in unary

✗ (b) $L = \{a^{n^n} \mid n \geq 1\}$

✓ (c) $L = \{a^m \mid n \geq 1, m > n\}$ ✓

= {a Take n=1

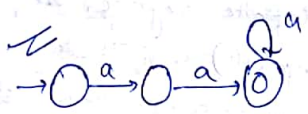
$a^{m'} = a^m = \{a^2, a^3, a^4, a^5, a^6 \ldots\}$ ← Present

Now n=2

m = 3, 4, 5, 6, ...

Hence, regular.

✗ (d) None.



Q. Which of the following is regular:

✗ (a) $L = \{ww^R x \mid w, x \in \{a,b\}^+\}$

✗ (b) $L = \{xww^R \mid w, x \in \{a,b\}^+\}$

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

✗ (d) $L = \{ww \mid w \in \{a,b\}^*\}$

PUMPING LEMMA:

→ Pumping Lemma is a tool used to prove a non-regular language as non-regular.

→ Pumping lemma uses proof by contradiction method.

→ Pumping lemma is based on Pigeon hole principle.

→ To prove, a language is non-regular by using pumping lemma;

(1) Assume that language is regular.

(2) If the language is regular, there exist finite automata for that language and assume n-no. of states are there in that.

(3) Select some string "z" from L (given language), whose length is greater than ~~or~~ or equal to n ($\geq n$).

- According to Pigeon hole principle, whenever string length is greater than or equal to the no. of states, then there exists self loop or cycle in the automata for some part of the string.

(4) Divide string z into 3 parts → u, v, w.
where "v" is loop/cycle string in the automata.

(5) If the given language is regular; $\forall i \geq 1, uv^iw \in L$.
For at least one value of "i" $uv^iw \notin L \Rightarrow L$ is non-regular.

NOTE

→ Pumping Lemma is strong for proving non-regular language as non-regular.

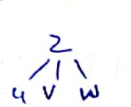
→ P.L. is weak for proving regular language as regular.
[For proving reg. languages, infinite no. of cases to be proved].

Q. Prove $L = \{a^n b^n \mid n \geq 1\}$ is non-regular using P.L.

① Assume L is regular

② F.A. → n states

③ $a^n b^n \geq n$

④ 

$a^n b^n$

$a^{n-1} a\ b^n$

$a^{n-1} a\ b\ b^n$

$a^{n-1} a^2 b^n \notin L$ ✗

$a^n b^n$

$a\ b\ b^{n-1}$

$a\ b\ b^{n-1}$

$a\ b^2 b^{n-1}$ ✗ L

Non-regular

NOTE

→ There exists a language L and there exists finite automata for that language. All strings having length less than no. of states of given finite automata, then that language is known as finite language.

Decision Properties of Regular Language:

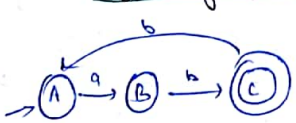
① Emptiness Problem: means checking whether given regular language is empty language or not.

OR checking whether given finite automata accepts empty language or not.

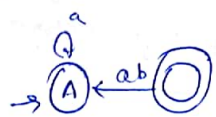
Algorithm:

(a) Eliminate all inaccessible states from the given finite automata.

(b) In the resultant automata, atleast one final state exists, that automata accepts non-empty language otherwise accepts empty language.



Non-empty



Empty

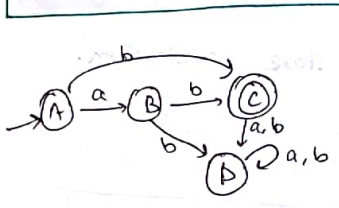
② Finiteness Problem: means checking whether language accepted by given finite automata is finite language or not.

Algorithm:

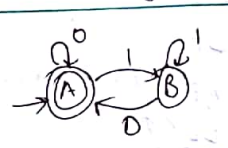
(a) Eliminate all inaccessible states.

(b) Eliminate the states from which we cannot reach final.

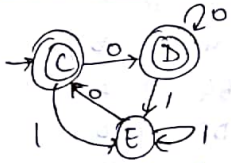
(c) In the resultant automata, if there exists loop/cycle, that automata accepts infinite language, otherwise finite language.



Finite lang.

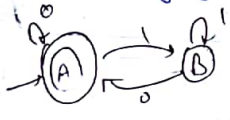


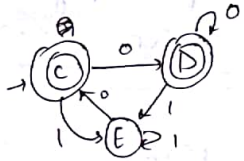
Infinite lang.



Infinite lang.

③ Equivalence Problem: means checking whether given two finite automata are equal or not. (accepting same language or not).





final-non final comb. is not equal

Comparison table

	0	1	
(AC)	AD	BF	→ Nonfinal-Non-final = ok
AD	AD	BF	
BF	AC	BE	

→ Final-Final combination = OK

Q5.

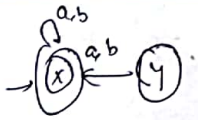
	a	b
AE	AE	CF
CF	DG	AE
DG	BG	DF
BG	DG	AF → final-nonfinal state combination.
DF	BG	DE
AE	AG	CE

Not equal

④ Completeness Problem: means checking whether given finite automata accepts complete language or not.

* Do complement of the given automata.
* If complemented automata accepts empty language, then given automata accepts complete language.

Algorithm:
① Complement the given FA (DFA).
② If complemented automata accepts empty language, then given automata accepts complete language.

Eg. 

→ Complete.

⑤ Membership Problem: means checking whether string "x" is member of given finite automata or not. (string is accepted or not).

Closure Properties of Regular Expression:

① Subset Operation: The subset of a regular language may or may not be regular.

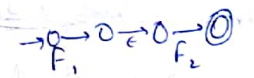
So, regular languages are not closed under subset operation.

E.g. $a^*b^* \subseteq (a+b)^*$

N.P. $\in \{a^n b^n / n, m \geq 1\} \subseteq (a+b)^*$

Q. Which of the following is true:
(a) Subset of regular set is regular. ✗
(b) Subset of a non-regular set is regular. ✗
(c) Subset of any infinite set is regular. ✗
(d) Subset of finite set is regular. ✓

② Concatenation Operation:



$L_1 = \{a^n / n \geq 1\} = F_1$ $\quad L_1 L_2 = \{a^n\}\{b^m\}$

$L_2 = \{b^m / m \geq 1\} = F_2$ $\quad$ Automata can be constructed.

→ Concatenation of two regular languages is always regular because we can construct ~~finite~~ concatenation of two finite automata by attaching ε-transitions.

→ Regular languages are closed under concatenation operation.

③ Union Operation:

$L_1 = R.L = r_1$

$L_2 = R.L = r_2$

$L_1 \cup L_2 = r_1 + r_2$

→ Union of two regular languages is always regular, hence, regular languages are closed under the union operation.

④ Complement Operation:

$L_1 = R.L.$ → DFA

$\overline{L_1} = R.L.$ → DFA

→ Complement of a regular language is always regular, hence, regular languages are closed under complement operation.

→ For $\Sigma^* - L$, DFA is constructed by interchanging final and non-final states of given language DFA.

⑤ Intersection operation:

L_1 is R.L

$L_2 = R.L.$

$L_1 \cap L_2 = \overline{\overline{L_1} \cup \overline{L_2}}$ → De Morgan's Law → regular.

→ The intersection of two regular languages is always regular.

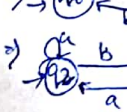
→ Regular languages are closed under union and complement, hence, also closed under intersection.

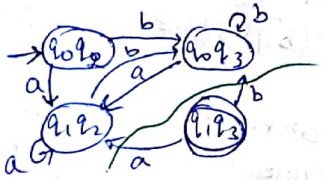
Q How many no. of states are there in min DFA constructed for $L_1 \cap L_2$? where $L_1 = \{(a+b)^*a\}$; $L_2 = \{(a+b)^*b\}$

$L_1 \cap L_2 \xrightarrow{AND} F_1 \times F_2$

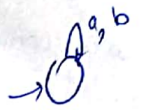
$F_1 = L_1 = (a+b)^*a$ ⇒

$F_2 = L_2 = (a+b)^*b$ ⇒





	a	b
→ q_0q_2	q_1q_2	q_0q_3
q_0q_3	q_1q_2	q_0q_3
q_1q_2	q_1q_2	q_0q_3
q_1q_3	q_1q_2	q_0q_3



⑥ Difference Operation:

$F_1 = L_1 = Reg.$

$F_2 = L_2 = Reg.$

$L_1 - L_2 = L_1 \cap \overline{L_2}$

$= F_1 \times F_2^c$

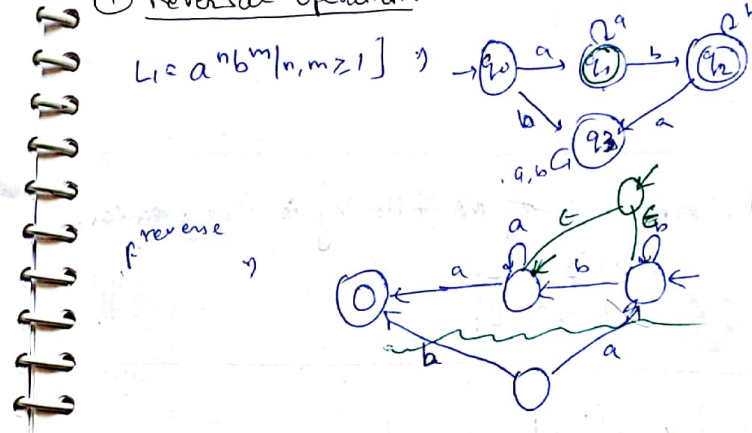
→ The difference of two regular language is always regular.

→ Finite automata for $L_1 - L_2$ is $F_1 \times F_2^c$.

Hence, regular languages are closed under difference operation.

⑦ Reversal Operation:

$L_1 = \{a^n b^m / n, m \geq 1\}$ ⇒



$f^{reverse}$ ⇒

→ Reverse of a regular language is always regular.

Reversal finite automata, constructed as follows:

Step-1: Interchange initial state as final and final state as initial and also reverse transition direction.

Step-2: If the given finite automata contains multiple ~~F.A.s~~ final states, then it results into multiple initial states. Multiple initial states converted into one initial state. by attaching new initial state with ∈-transitions.

→ Reversal operation is applied on Reg. Exp. as follows:

① $(E+F)^R = E^R + F^R$

② $(E.F)^R = F^R.E^R$

③ $(E^*)^R = (E^R)^*$

$E = 011$

$F = 100$

① $\{011, 100\}^R = \{110, 001\} = \{110, 001\}$

② $(011100)^R = 001110 = 001110$

(8) **Kleene Closure**: If a regular language is always regular because we can construct F.A. for kleene closure by using Thomson construction algorithm.

(9) **Positive Closure**: If a regular language is always regular because we can construct F.A. for positive closure by using Thomson construction algorithm.

$r_1 = L_1 = R.L.$

$r_1^* = L_1^* = L_p^0 + L_1^1 + L_2^2 \epsilon$ — —

Q. L_1 = Reg. $L_2 \subseteq L_1$. Then which of the following is always regular.

✗ (a) L_2
✗ (b) L_2^*
✗ (c) $\Sigma^* - L_2$
✓ (d) $\Sigma^* - L_1$

---

(10) **Quotient Operation**: $L_1/L_2 = \left\{ x \;/\; xy \in L_1 \text{ and } y \in L_2 \right\}$

$\frac{L_1}{L_2} = \frac{xy}{x} = x \;\Big|\; \frac{010}{10} = 0 \;\Big|\; \frac{011}{01} = \phi \;\Big|\; \frac{011}{011} = \epsilon \;\Big|\; \frac{\epsilon}{01} = \phi \;\Big|\; \frac{0^*}{0} = 0^*$

$\frac{0^*1}{1^*0} = \phi \;\Big|\; \frac{\Sigma^*}{\Sigma^*} = \Sigma^* \left[\frac{\epsilon}{\epsilon}, \frac{a}{\epsilon}, \frac{a}{\epsilon}, \frac{aa}{\epsilon}, \frac{b}{\epsilon}, \frac{bb}{\epsilon}, \frac{aba}{\epsilon} \right] \;\Big|\; \frac{\Sigma^+}{\Sigma^+} = \left\{ \frac{a}{a}, \frac{ab}{a}, \frac{ba}{a}, \frac{b}{b} \right\} = \Sigma^*$

$$\frac{\Sigma^*}{\Sigma^*} = \frac{\Sigma^+}{\Sigma^+} = \frac{\Sigma^*}{\Sigma^+} = \frac{\Sigma^+}{\Sigma^*} = \Sigma^*$$

Q. $L_1 = a^*ba^*$
$L_2 = b^*a$
$L_1/L_2 = ?$

(a) $a^*ba^* + a^*$ ✓
(b) a^* b not possible ✗
(c) a^*ba^* No ϵ ✗
(d) $a^*b^+a^+$ No ϵ ✗

$\frac{a^*ba^*}{b^*a} = \frac{ba}{ba} = \epsilon$

$L_1 = a^*b$
$L_2 = b^*$
$L_1/L_2 = ?$

(a) a^*b^+ No ϵ ✗
(b) $a^*(b+\epsilon)$ ✓
(c) a^*b No epsilon ✗
(d) b^* No a ✗

$\frac{a^*b}{b^*} = \frac{b}{b} = \epsilon \;\Big|\; \frac{ab}{b} = a$

→ Quotient of two regular languages is always regular because we can construct F.A..

 # Quotient F.A. Construction:

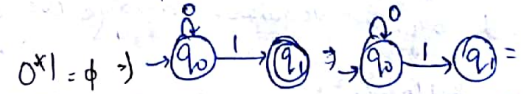
→ L_1/L_2 F.A. is L_1 finite automata (maybe) having different final and non-final states.

Step-1: Starting from the first state consider the total F.A. If it accepts any string of L_2, then make first state as final otherwise non-final.

Step-2: Starting from the second state consider the total automata, if it accepts any string of L_2, then make second state as final otherwise non-final.

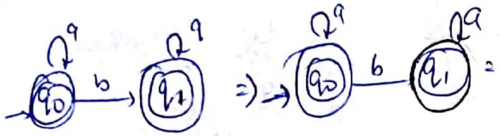
Step-3: Repeat step-2 for every state to decide whether it is final state or not.

$\frac{0^*1}{1^*0} = \phi$:)



$\frac{\Sigma^+}{\Sigma^+}$:)

$\frac{a^*ba^*}{b^*a}$ =



$= a^*ba^* + a^+$

---

(11) Init Operation:

$Init(L) = \{$ set of all prefixes of L $\}$

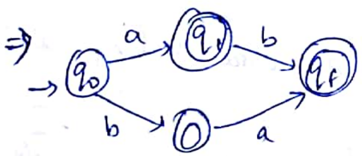
$L = \{ab, ba\}$

$Init(L) = \{\epsilon, a, b, ab, ba\}$

$L = \{x \mid x \in \{a,b\}^*, n_a(x) = n_b(x)\}$ $L = \{\epsilon, ab, ba, aabb, bbaa, abab, baba \ldots\}$

$Init(L) = \{\epsilon, a, b, aa, ab, ba, bb \ldots\} = (a+b)^* = \Sigma^*$

$L = \{ab, ba\} \Rightarrow$



NOTE

→ Init of a regular language is always regular, hence, regular languages are closed under init operation.

→ Init F.A. can be constructed by making all states as final except dead state.

(12) Substitution Operation:

→ Substitution is a mapping from $\Sigma \to \Delta$ (alphabet Σ → alphabet Δ), where each symbol of Σ is replaced by a regular language over Δ.

→ Substitution is performed as follows.

1) $s(\phi) = \phi$

$s(\epsilon) = \epsilon$

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

4) $s(a \cdot b) = s(a) \cdot s(b)$

5) $s(a^*) = s(a)^*$

→ Substitution of a regular language is always regular because substitution function can be applied on finite automata transitions.

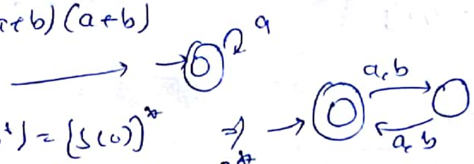
$S = \{0\}$ $\Delta = \{a, b\}$

$s(0) = (a+b)(a+b)$

$L_1 = 0^*$

$s(L_1) = s(0^*) = [s(0)]^*$

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



---

(13) Homomorphism Operation:

→ Homomorphism is a special case of substitution where each symbol of Σ is replaced by a single string over the alphabet Δ.

NOTE:

→ Homomorphism of a regular language is always regular because homomorphism directly applied on finite automata transitions.

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

$h(0) = aa$

$h(1) = \epsilon$

$L = 1^* 0 1^*$

$h(L) = h(1^* 0 1^*)$

$= (h(1))^* . h(0) . (h(1))^*$

$= (\epsilon)^* . aa . \epsilon^*$

$= aa$

$\Sigma = \{a, b, c\}$ $\Delta = \{0, 1\}$

$h(a) = 0$

$h(b) = 1$

$h(c) = 1010$

(14) Inverse Homomorphism: Applying homomorphism in reverse way is known as inverse homomorphism.

i.e, string is replaced by symbol.

$\Sigma = \{a, b, c\}$ $\quad \Delta = \{0, 1\}$

$h(a) = 0$

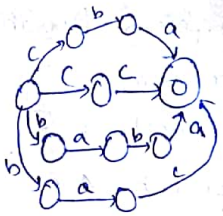
$h(b) = 1$

$h(c) = 10$

$L = 1010 \mid h^{-1}(L) = \{cc, \sout{cba}, baba, cba, bac\}$

$h(h^{-1}(L)) = \{1010, 1010, 1010, 1010\}$

$= \{1010\}$



~~$h(h^{-1}(L)) = L$~~

Q. $h(0) = aa$ $\quad h^{-1}(L) = (\phi)^* = \epsilon$

$h(1) = bb$

$L = (ab + ba)^*$ $\quad h(h^{-1}(L)) = h(\epsilon) = \epsilon \neq L$

NOTE

$h(h^{-1}(L))$ need not be L.

Sometimes may be L, sometimes not.

NOTE

→ Inverse homomorphism of a regular language is always regular because we can construct the finite automata by replacing transitions of the strings replaced by corresponding symbols according to homomorphism function.

(15) Infinite Union:

$\{L_1, L_2, L_3, \ldots\}$ Reg. lang.

$L_1 \cup L_2 \cup L_3 \cup \ldots$ = Reg or Non-Reg.

$\{ab\}\{aabb\}\{aaabbb\}\ldots\} = a^n b^n$ ✗

→ Infinite Union of regular languages may or may not be regular, hence, regular languages are not closed under infinite union operation.

(16) Infinite Intersection:

→ Infinite intersection of regular languages may or may not be regular languages are not closed under infinite intersection operation.

$L_1 \cap L_2 \cap L_3 \cap \ldots\} = \overline{\overline{L_1} \cup \overline{L_2} \cup \overline{L_3} \cup \overline{L_4} \cup \ldots}$ → Regular or Not regular

Q. Which of the following operation is not closed under regular language:

(a) Union
(b) Intersection
(c) Both a and b
(d) None of these.

* By default union is finite union.
* By default intersection is finite intersection.

Q. L_1 is regular and $L_1 \cap L_2$ is regular, then L_2 is:

(a) Should be regular (b) Need not be regular $ab \cap (a^n b^n) = ab$

Q. ~~If~~ L_1 is regular, ~~$L_1 \cup L_2$ is~~ $L \cup L_2$ is ——

$(a+b)^* \cup (a^n b^n) = (a+b)^*$

(a) regular (b) Need not be regular.

Q. L_1 is regular, $L_1 \cdot L_2$ is regular, then L_2 is ——

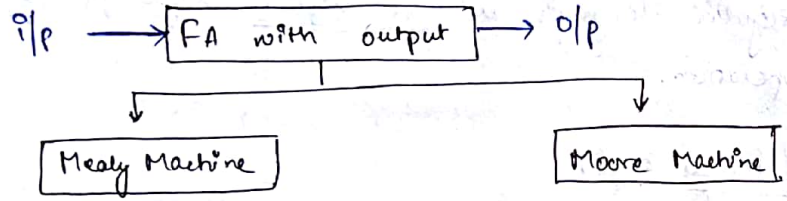
(a) should be regular (b) Need not be regular.

$\phi \cdot \{a^n b^n\} = \phi$

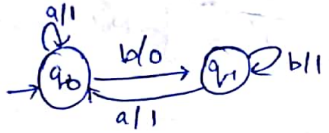
24/07/2017

Finite Automata as Output Generator:

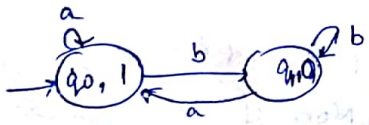
OR



Mealy Machine: is a mathematical model in which output is associated with transitions.



Moore Machine: is a mathematical model in which output is associated with state.



*Formal Definition: $(Q, \Sigma, q_0, \Delta, \delta, \lambda)$

$Q \rightarrow$ Finite number of state
$\Sigma \rightarrow$ input alphabet
$q_0 \rightarrow$ initial state
$\Delta \rightarrow$ output alphabet
$\delta \rightarrow$ transition function $Q \times \Sigma \rightarrow Q$
$\lambda \rightarrow$ output function: Mealy $Q \times \Sigma \rightarrow \Delta$
Moore $Q \rightarrow \Delta$

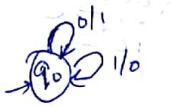
→ Both Mealy and Moore machines are deterministic in nature. Hence, from every state on every input symbol exactly one transition exists.

→ There is no final state in Mealy and Moore machine because they are not accepted.

→ Mealy and Moore machines practically used in circuit design.

→ Both Mealy and Moore machines can be represented by using transition diagram and transition table.

Q. Construct the Mealy m/c that represents the behaviour of 1's complement of binary number circuit.



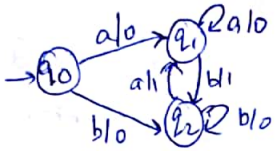
Q. Construct the Mealy m/c that represents the behaviour of 2's complement of binary number ckt. (Assume that we are reading the string from L.S.B to M.S.B. and carry bit is discarded)

0 1 0 1 0 0 1 0 0
1 0 1 0 1 1 0 0
→ After 1 complement

NOTE
While reading from LSB for any no. of 0's produce 0's as o/p until 1 comes.
for First 1 ~~convert~~ produce 1 as o/p. Afterwards, complement the bits.

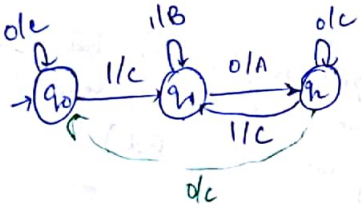
Q. Construct the Mealy m/c that takes all strings of a's & b's as i/p and produces 1 as output if last two symbols in the i/p are same. Otherwise, produces 0 as output.

Q. Construct the Mealy machine that takes all strings of a's & b's as input and produces 1 as o/p if the last two symbols in the i/p are ~~diff~~ different otherwise produces 0.

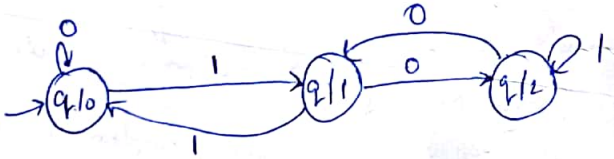


Q. Construct the Mealy machine that takes all strings of 0's and 1's as i/p and produces "A" as o/p if the i/p is ending with 10 or produces "B" as o/p if the i/p is ending with 11. Otherwise produces "C" as output.

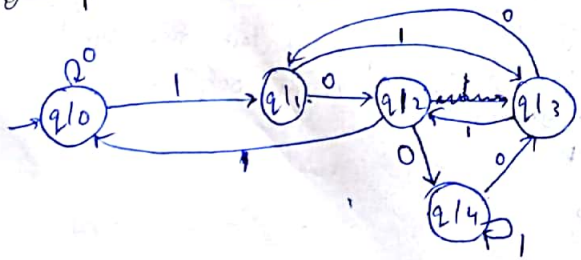
00, 01 → C
10 → A
11 → B



Q. Construct the Moore m/c that takes all binary numbers as input and produces residue modulo-3 as o/p.

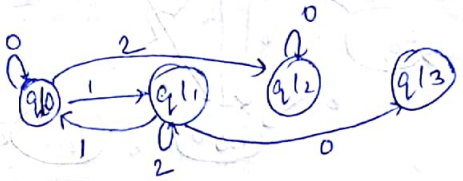


Q. Construct the Moore m/c that takes all binary nos. as i/p and produces residue modulo-5 as o/p.



	0	1
q_0	q_0	q_1
q_1	q_2	q_3
q_2	q_4	q_0
q_3	q_1	q_2
q_4	q_3	q_4

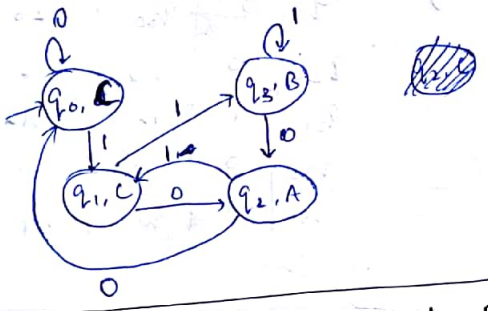
Q. Construct the Moore m/c that takes all base 3 $(()_3)$ as i/p and produces residue modulo 4 as o/p.



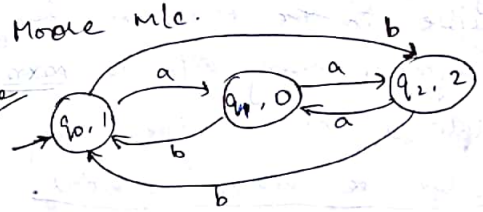
	0	1	2
$q_{,0}$	q_0	q_1	q_2
$q_{,1}$	q_3	q_0	q_1
$q_{,2}$	q_2	q_3	q_0
$q_{,3}$	q_1	q_2	q_3

Q. Construct the Moore m/c that takes all strings of 0's & 1's as i/p. produces "A" as o/p if the i/p ends with "10" or "B" as o/p if the i/p ends with "11". Otherwise produces "C" as o/p.

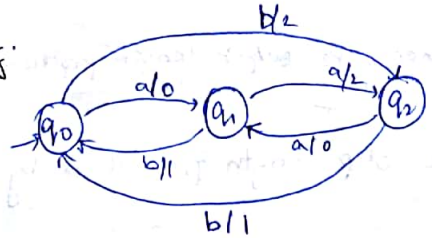
10 → A
11 → B
C



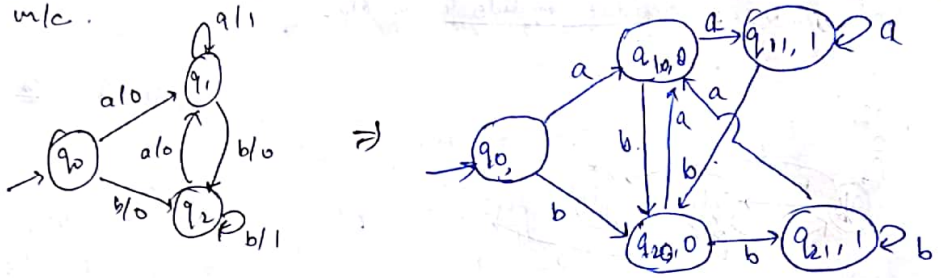
Q. Construct an equivalent Mealy machine for the following Moore m/c.



Mealy.



Q. Construct an equivalent Moore m/c for the following Mealy m/c.



Q. Mealy m/c:

	i/p a		i/p b	
	N.S.	o/p	N.S	o/p
→ q_0	q_2	1	q_3	1
q_1	q_3	0	q_2	0
q_2	q_1	0	q_1	0
q_3	q_0	1	q_2	1

⇒

Moore:

state	o/p	a	b
→ q_0	1	q_{21}	q_{31}
q_1	0	q_{30}	q_{20}
q_{20}	0	q_{10}	q_{10}
q_{21}	1	q_{10}	q_{10}
q_{30}	0	q_{00}	q_{21}
q_{31}	1	q_0	q_{21}

NOTE

→ While converting from Mealy machine to Moore machine minimum no. of states possible in the Moore m/c for the given n-states, m-o/p symbols. Mealy m/c is mxn

→ If the initial state is splitted into multiple states any one of the state can be taken as initial state.

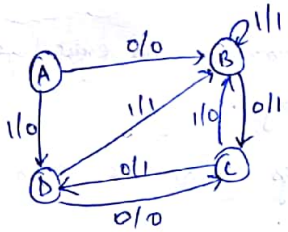
→ For the n-length input sequence, the output length produced by Mealy machine is n only.

→ For the n-length input, the o/p length produced by Moore m/c is n+1.

→ To construct equivalent Mealy and Moore machines, the initial o/p length produced by Moore machine is neglected.

Q Consider the following Finite S.M.

	i/p x=0		i/p x=1	
	N.S.	o/p	N.S.	o/p
A	B	0	D	0
B	C	1	B	1
C	D	1	B	0
D	C	0	B	1



- If the initial state is unknown:

Then, which of the following the shortest i/p sequence to reach state. C:

(a) 01 (c) 110
(b) 10 (d) 101

H.W Q How many maximum no. of Mealy machine can be constructed with 2 states x & y over the input symbols {a,b} & o/p symbols {0,1}.

GRAMMAR :

→ Set of rules used to describe strings of a language, known as grammar.

→ For every language grammar exists.

→ Every grammar generates one language.

→ Formal Definition: $G = (N, T, P, S)$

N → Non-Terminals or Variables
T → Terminals
P → Number of productions
S → Starting Symbol

→ Every grammar is of the form $\alpha \rightarrow \beta$.
where β is replacement for α.

→ For one language, many number of grammars can exist but one grammar generates only one language.

* Derivation: The process of generating strings from the given grammar is known as derivation.

- The derivation can be either leftmost derivation or right most derivation.

→ In left most derivation, the left most non-terminal is replaced by its R.H.S. part at every step.

→ In right most derivation, the right most non-terminal is replaced by its R.H.S part at every step.

Parse Tree / Derivation Tree:

→ Tree representation of derivation is known as Derivation Tree / Parse Tree.

→ All leaf nodes of the parse tree are known as yield of the parse tree.

Sentential Form:

→ Every step in the derivation is one sentential form.

→ If the derivation is left most derivation, then the sentential forms are left sentential forms.

→ If the derivation is right most derivation, then the sentential forms are right sentential forms.

→ For regular language, grammar exists, known as regular grammar or type-3 grammar.

→ For context free language, grammar exists known as context free grammar or type-2 grammar.

→ For context sensitive language, grammar exists, known as context sensitive grammar or type-1 grammar.

→ For Recursive enumerable language, grammar exists, known as unrestricted grammar or type-0 grammar.

Q. Construct Grammar for the following language.

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

$S \rightarrow AB$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid \epsilon$

② $L = \{a^n b^m c^k \mid n \geq 2, m \geq 0, k \geq 3\}$

$S \rightarrow ABC$
$A \rightarrow aaA \mid aa$ or
$B \rightarrow bB \mid \epsilon$
$C \rightarrow cC \mid ccc$

$S \rightarrow aABcc$
$A \rightarrow aA \mid a$
$B \rightarrow bB \mid \epsilon$
$C \rightarrow cC \mid c$

③ $L = \{a^n b^m \mid (n+n) \text{ is even}\}$

$S \rightarrow AB \mid aABb$
$(aa)^* \quad A \rightarrow aaA \mid \epsilon$
$(bb)^* \quad B \rightarrow bbB \mid \epsilon$

④ $L = \{a^n b^n \mid n \geq 1\}$

$S \rightarrow ab \mid asb$

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

$a^n b^n b^m c^m$

aa bbbbb ccc

$S \to AB$

~~$S \to aSbSc \mid$~~ $A \to aAb \mid ab$

$B \to bBc \mid bc$

Q. Construct the grammar that generates all ~~st~~ strings of a's and b's where:

(1) Each string starting and ending with different symbol.
(2) Each string having aaa or bbb as substring.
(3) Length of the string exactly 4.
(4) Length of the string divisible by 5.
(5) 4th input symbol is "a" while reading the string from R.H.S.
(6) Binary numbers divisible by 3.

(1) $S \to AB$

$A \to aA \mid B \mid b$

$B \to bB \mid A \mid a$

(2) $S \to AaaaA \mid AbbbA$

$A \to a \mid b \mid \epsilon$

(3) $S \to AA$

$A \to ab \mid ba \mid aa \mid bb$

~~$B \to aa \mid ba \mid ab \mid bb$~~

(4)

(5) ~~$S \to Aa$~~ ~~AA~~ $S \to AaBBB$

$A \to aA \mid bA \mid \epsilon$

$B \to a \mid b$

(6)

25/07/22

Q. Identify the language generated by following grammars:

(1) $S \to aS \mid bS \mid a \mid b$

$(a+b)^+$

(2) $S \to aSd \mid aAd$

$A \to bAc \mid bc$

$\{a^n b^m c^m d^n / m,n \geq 1\}$

(3) $S \to aSb \mid aAb$

$A \to aAa$

~~a^n~~ $a^n a^+ b^n \{a^n b^m, n>m, n,m \geq 1\}$

(4) $S \to aSa \mid bSb \mid a \mid b$

~~Starting & ending with same symbol.~~

Palindrome (odd length)

(5) $S \to aA \mid bA \mid a \mid b$

$A \to aA \mid bA \mid \epsilon$

Starting and ending with same symbol.

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

Q. Construct grammar for the following languages:

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

$a^n a^m b^m c^n$

$S \to aSc \mid aAc$

$A \to aAb \mid ab$

(2) All odd length palindrome strings of english language.

$S \to \underbrace{aSa \mid bSb \mid \ldots \mid z}_{26 \text{ prod.}} \mid \underbrace{a \mid b \mid c \ldots \mid z}_{26 \text{ prod.}}$

Q. $S \to AB$

$A \to a$

$B \to b$

(A) Regular Grammar

(B) Context free grammar ✓

(C) Context Sensitive Grammars

(D) Unrestricted grammar.

Types of Grammar

• All grammar are of the form $\alpha \to \beta$
i.e. should contain at least one non-terminal.

Type 0 Grammar or **Unrestricted Grammar** or **Recursive Enumerable Grammar**

$\alpha \to \beta$

Type 1 Grammar or **Context Sensitive Grammar**

$\alpha \to \beta$

$|\alpha| \leq |\beta|$ $\alpha, \beta \in (V+T)^+$

LCO

Type 2 Grammar or **Context Free Grammar**

$A \to \alpha \mid \alpha \in (V+T)^*$

2 LCO

Type 3 Grammar or **Regular Grammar**

Right Linear Grammar: $A \to xB \mid x$

Left Linear Grammar: $A \to Bx \mid x$

$A, B \in V$
$x \in \Sigma^*$

→ 3 LCO

NOTE

Linear Grammar:

→ In any grammar, LHS one non terminal exits and RHS almost one non-terminal exists, then it is known as Linear Grammar.

→ Linear grammar can be left linear or right linear or middle linear.

→ If any linear grammar is completely left linear or completely right linear (but not both), then it is regular.

→ All regular grammars are linear grammars but all linear grammars need not be regular.

→ The following grammars are linear but not regular.

① $S \to aSb \mid \in$ → Linear but not regular

② $S \to aA \mid Bb \mid G$

Q. Identify possible type numbers and type of the following grammars.

① $S \to aSa \to$ 3,2,1,0 (Regular Grammar)

② $S \to aSb \mid bSa \to$ 2,1,0 (Context Free Grammar)

③ $S \to AB$ → 2,1,0 (Context Free Grammar)
$A \to a$
$B \to b$

④ $S \to aSbS \mid \in \to$ ~~0 (Unrestricted Grammar)~~ 2,1,0 (CFG)

⑤ $S \to bA \mid aB$ → Type-0 (REG | Unrestricted)
$aA \to bb$
$bB \to b$

⑥ $S \to aA \mid B \mid \in$ → Type-1 ~~(CFG)~~ (CSG)
$aA \to bb$
$B \to b$

Chomsky Hierarchy

Type 0 Grammar

T/F.

→ $A \to \epsilon$ is valid context sensitive grammar production, if "A" not present in RHS part of the grammar productions.

Q. ① $S \to \epsilon | aS | bS$
Regular Grammar (Type 3)

② $S \to AB$ 2,1,0
$A \to \epsilon$ CFG
$B \to \epsilon$

③ $S \to aAB | \epsilon$
$aA \to bS | bb$ Type 0
$B \to d$
REG / Unrestricted

④ $S \to aSa | bSb | \ldots zSz | \epsilon$
CFG (Type 2)

REGULAR GRAMMAR:

→ Regular Grammars can generate regular language only. [cannot generate non-regular language].

→ Corresponding to every regular grammar, regular expression or F.A. can be constructed.

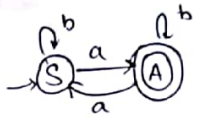
→ for every regular language, we can construct an equivalent Right linear grammar or Left linear Grammar.

→ for every right linear grammar, we can construct an equivalent left linear grammar.

→ for every left linear grammar, an equivalent right linear grammar can be constructed.

→ By default, Regular grammar is Right Linear Grammar.

Finite Automata to Regular Grammar Construction:



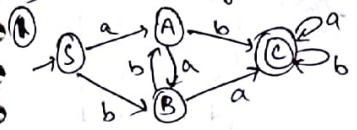
NT = States
T = a, b
P = δ
S = I. state

$S \to bS | aA$
$A \to bA | aS | \epsilon$

→ While constructing regular grammar from the given finite automata.

* No. of non-terminals (N) = No. of states.
* No. of terminals of the grammar = Input alphabet (Σ)
* No. of productions is based on transitions (δ).
* Starting symbol of the grammar is initial state.

Q. Construct regular grammar;



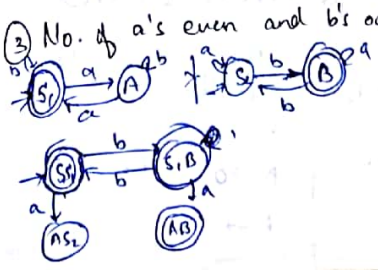
$S \to aA | bB$
$A \to aB | bC$
$B \to aC | bA$
$C \to aC | bC | \epsilon$

② R.G. that generates all strings of a's and b's, where:
(a) No. of a's divisible by 3
(b) No. of a's not divisible by 3.

$S \to aA | bS | \epsilon$
$A \to aB | bA$
$B \to aS | bB$
#a's Divisible by 3

$S \to aA | bS$
$A \to aB | bA | \epsilon$
$B \to aS | bB | \epsilon$
#a's not divisible by 3.

③ No. of a's even and b's odd.



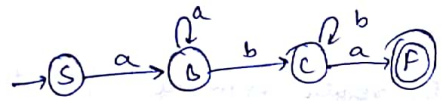
$S \to aA | bC$
$A \to aS | bB$
$B \to aC | bA$
$C \to aB | bS | \epsilon$

→ Finite automata to regular grammar construction always results in Right linear grammar.

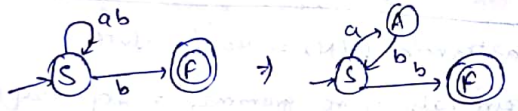
Regular Grammar to Finite Automata:

Q. Construct finite automata corresponding to:

(1) S → aB
B → bC | aB
C → a | bC

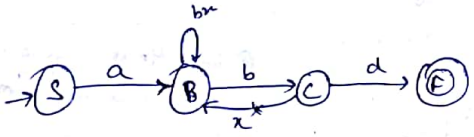


(2) S → abS | b



(3) Construct the regular expression for the following reg. grammar.

S → aB
B → bC
C → xB
C → d

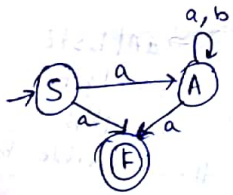


a(bx)*bd

	a	b	x	d
S				
B				
C				
F				

Q. Identify language generated by the following reg. grammar.

S → aA | a
A → aA | bA | a

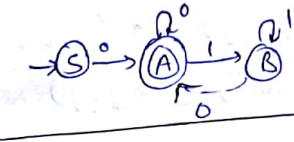


a + a(a+b)*a
starting & ending with a.

Q. Which of the following grammar production is added so that the resultant grammar generates set of all strings of 0's and 1's. Each string starts & ends with 0.

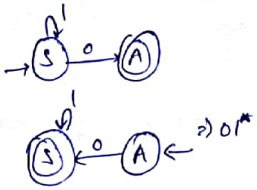
(a) B → 1S
(b) B → 0A
(c) S → 1C
(d) B → 1B

S → 0A
A → 0A | 1B | ∈
B → 1B



S → S1 | 0 ⇒ 01*

(1) S → 1S | 0



Q Construct F.A. for the following Left linear grammar.

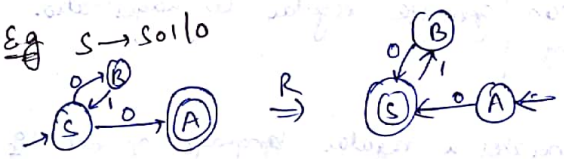
NOTE Construction of finite automata for left linear grammars.

(1) Reverse RHS part of the grammar productions
(2) Construct F.A.
(3) Reverse the F.A. by interchanging initial state as Final and final state as initial state and reverse the transitions.

Construction of Left linear grammar from the given F.A.

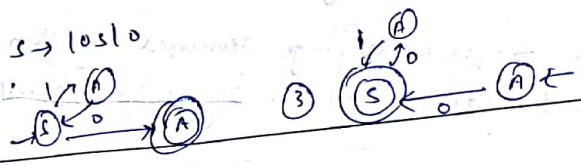
Step 1: Reverse the given finite automata.
Step 2: Construct right linear grammar from the reversal automata.
Step 3: Reverse RHS part of the grammar production.

Eg S → S0 | 0



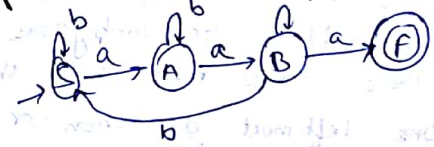
A → 0S
S → 1S | ∈
⇒
A → S0
S → S0 | ∈

(1) S → 1S | 0



Q. Identify language:

S → aA | bS
A → aB | bA
B → a | bS | a



b*ab*aa*a

(a) $L = \{ x \mid x \in \{a,b\}^* , n_a(x) = 3 n_b(x) \}$

(b) $L = \{ x \mid x \in \{a,b\}^* , 3 n_a(x) = n_b(x) \}$

(c) $L = \{ x \mid x \in \{a,b\}^* , n(x) = 3 + n_b(x) \}$

(d) None

CONTEXT FREE GRAMMAR:

→ Context free grammars are used to represent syntax of languages.

→ Language generated by context free grammars are known as content free language.

→ Content free grammars cannot generate non-CFLs.

→ Content free grammars can generate regular languages also.

NOTE

→ Checking whether CFG generates a regular language or not is undecidable.

→ There is no algorithm to check language generated by given CFG is regular or not. (It is known as regularity problem of CFG)

Ambiguity Problem of grammars:

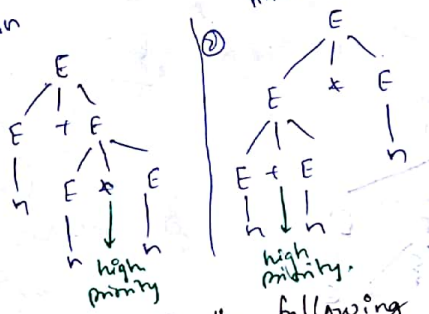
Def. 1 A grammar is said to be ambiguous, for atleast one string "x", if there exists more than one parse trees or more than one leftmost derivation or more than one rightmost derivation.

Def: A grammar is said to be unambiguous, for all strings of the given grammar, exactly one parse tree should exist or exactly one leftmost derivation should exist or only one right most derivation should exist.

→ If the grammar is ambiguous, difficult to construct compiler for the language.

Eg: $E \to E+E \mid E*E \mid n$

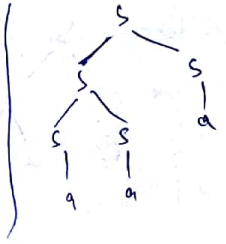
n+n*n



n+n*n

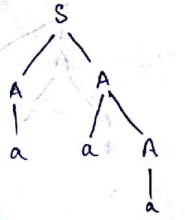
Q Verify which of the following grammars are ambiguous grammars.

(2) $S \to SS \mid a$

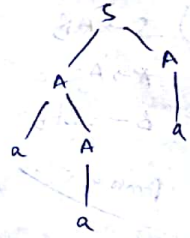


Ambiguous.

(3) $S \to AA$
$A \to aA \mid a$



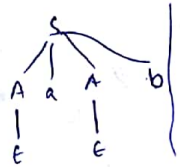
Ambiguous.



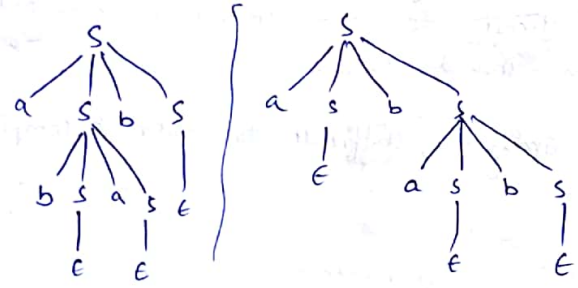
Ambiguous.

(4) $S \to A\,a\,A\,b \mid B\,b\,B\,a$
$A \to \epsilon$
$B \to \epsilon$

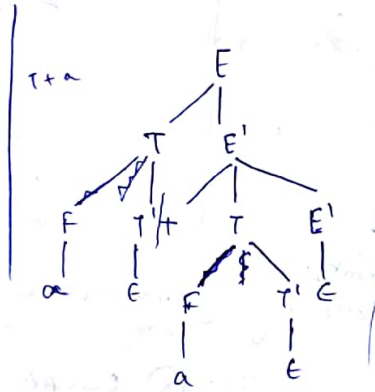
Unambiguous.



(5) $S \rightarrow aSbS \mid bSaS \mid \epsilon$ Ambiguous.



(6) $E \rightarrow TE'$
$E' \rightarrow +TE' \mid \epsilon$
$T \rightarrow FT'$
$T' \rightarrow *FT' \mid \epsilon$
$F \rightarrow a$

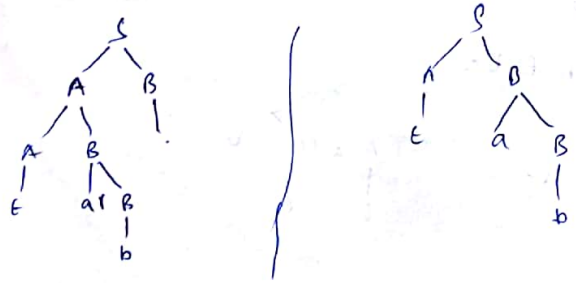


(7) $S \rightarrow AB$
$A \rightarrow Aa \mid \epsilon$
$B \rightarrow aB \mid b$

Ambiguous

$S \rightarrow AB$
$A \rightarrow AB \mid \epsilon$
$B \rightarrow aB \mid b$

Unambiguous



(8) $S \rightarrow A$
$A \rightarrow Aa \mid \epsilon$

NOTE
→ Checking whether given grammar is ~~and~~ ambiguous or not is undecidable problem.
→ There is not algorithm to check if a grammar is ambiguous or not.

NOTE
→ Elimination of ambiguity from grammar is undecidable problem.
→ There is no algorithm to eliminate ambiguity from all ambiguous grammars.

26/07/2017:

#Inherently Ambiguous ~~grammar~~ Language:

→ A grammar is said to be inherently ambiguous if elimination of ambiguity not possible for that grammar.

→ A language is said to be inherently ambiguous, if all grammars of that language are inherently ambiguous grammars.

→ A language is said to be unambiguous if atleast one unambiguous grammar exists for that language. (Ambiguous grammars may exist).

Simplification of context free grammar:

① Elimination of useless variables:

→ Eliminate variables which are not deriving any string and also eliminate variables which are not required for derivation.

Q. Eliminate useless variables from the following grammar productions:

① $S \to AB \mid AC$
$A \to aAb \mid bAala$
$B \to bbA \mid aaB \mid AB$
× $C \to aaCa \mid aDb$
× $D \to bb \mid aC$

$S \to AB$
$A \to aAb \mid bAa \mid a$
$B \to bbA \mid aaB \mid DB$

Q. $S \to Aa \mid a$
$A \to a$
× $B \to bB \mid C$
× $C \to aC$

$S \to a$
$A \to a$

$S \to a$

② Elimination of Unit productions:

→ Any production is of the form $A \to B$ is known as unit production.

NOTE
→ If unit productions present in the grammar, then it is difficult to identify useless variables.
Hence, to simplify the grammar, first eliminate unit production and then useless variables.

Q. Eliminate unit productions:

$S \to Aa \mid B$
$B \to A \mid bb$
$A \to a \mid bc \mid B$

$S \to Aa \mid a \mid bc \mid bb$
$B \to a \mid bc \mid bb$
$A \to a \mid bc \mid bb$

$\Rightarrow$ $S \to Aa \mid a \mid bc \mid bb$
$A \to a \mid bc \mid bb$

③ Elimination of null productions ($A \to \epsilon$):

→ Any production of the form "$A \to \epsilon$" is known as null production.
→ To simplify the grammar, eliminate null productions and then unit productions and then useless variables.

E.g. $S \to aS, b$
$S_1 \to aS, b \mid \epsilon$
$\Rightarrow$ $S \to aS, b \mid ab$
$S_1 \to aS, b \mid ab$

Q. $S \to AaB$
$A \to BC$
$B \to a \mid \epsilon$
$C \to b \mid \epsilon$
$\Rightarrow$
$S \to \Lambda aB \mid aB \mid Aa \mid a$
$A \to Bc \mid \epsilon \mid Bc$
$B \to a$
$C \to b$
$\Rightarrow$
$S \to Aa \mid Ba \mid B \mid Aa \mid a$
$A \to Bc \mid B \mid C$
$B \to a$
$C \to b$

$\Rightarrow$ $S \to AaB \mid aB \mid Aa \mid a$
$A \to Bc \mid a \mid b$
$B \to a$
$C \to b$
$\Rightarrow$
$S \to Aa \mid a \mid aa \mid Aa \mid a$
$A \to a \mid b \mid a \mid b$

Normal Form of CFG:

→ Applying condition on the R.H.S. part of CFG is known as Normal form of the CFG.
→ There are mainly 2 normals forms that exist for CFG, known as Chomsky Normal Form and Greibach Normal Form.
→ Any grammar is of the form:
Right side one terminal or two variables
$A \to BC$
$A \to a$
$ABC \in V$
$a \in T$
, is known as Chomsky Normal Form.

→ Any grammar is of the form:
$A \to aV^*$; $a \in T$, is known as Greibach Normal Form.
→ To construct normalised grammar, the given grammar should be simplified.

RHS → one terminal followed by any no. of non terminals.

Q. Convert following CFG into CNF:

① $S \to aSb \mid ab$

$S \to ASB \mid AB$

$S \to AX \mid AB$

$A \to a$

$B \to b$

$X \to SB$

② $S \to bA \mid aB$

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

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

$A_1 \to a$

$B_1 \to b$

$\Rightarrow$ $S \to B_1A \mid A_1B$

$A \to B_1AX \mid A_1S \mid a$

$B \to A_1Y \mid B_1S \mid b$

$A_1 \to a$

$B_1 \to b$

$X \to AA$

$Y \to BB$

→ To convert given grammar into CNF grammar:

Step 1: Simplify the grammar.

Step 2: Replace terminals by new non-terminals on the RHS part of the grammar.

Step 3: Convert multiple non-terminals on the RHS part into two non-terminals with the help of new non-terminals.

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

$A \to a$

$B \to b$

$\Rightarrow$ $S \to ASA \mid BSB \mid c$

$A \to a$

$B \to b$

$\Rightarrow$ $S \to AX \mid BY \mid c$

$A \to a$

$B \to b$

$X \to SA$

$Y \to SB$

Q. $S \to aAB \mid Bb$

$A \to a$

$B \to b$

$\Rightarrow$ $S \to AAB \mid BB$

$A \to a$

$B \to b$

$\Rightarrow$ $S \to AX \mid BB$

$A \to a$

$B \to b$

$X \to AB$

Q. Convert following LFGs into GNF grammar:

① $S \to aSb \mid ab$

$S \to aSB \mid aB$

$B \to b$

② $S \to AB$

$A \to aA \mid bB \mid b$

$B \to b$

$\Rightarrow$ $S \to aAB \mid bBB \mid bB$

$A \to aA \mid bB \mid b$

$B \to b$

③ $S \to Sa \mid b$ (recursive part / non-recursive part)

$S \to bS'$

$S' \to aS' \mid \epsilon$

NOTE

→ If a grammar is having left recursion, then it cannot be converted into GNF grammar.

→ To convert, left recursive grammars into GNF, left recursion is removed by rewriting grammar productions in right recursion. (Conversion of left recursive grammar into right recursive grammar).

$A \to A\alpha \mid \beta \Rightarrow A \to \beta A'$

$A' \to \alpha A' \mid \epsilon$

Generalised Form:

$A \to A\alpha_1 \mid A\alpha_2 \mid \dots \mid \beta_1 \mid \beta_2 \mid \beta_3 \mid \dots$

$A \to \beta_1 A' \mid \beta_2 A' \mid \beta_3 A' \mid \dots$

$A' \to \alpha_1 A' \mid \alpha_2 A' \mid \dots \mid \epsilon$

Q. Eliminate left recursion:

① $E \to E+T \mid T$

$T \to T*F \mid F$

$F \to a$

$E \to E+T \mid T$

$E \to TE'$

$E' \to +TE' \mid \epsilon$

$T \to T*F \mid F$

$T \to FT'$

$T' \to *FT' \mid \epsilon$

$F \to a$

② $S \to Sa \mid Sb \mid c \mid d \mid e$

$S \to cS' \mid dS' \mid eS'$

$S' \to aS' \mid bS' \mid \epsilon$

③ $A \to Bc \mid a$

$B \to Ab \mid a$

NOTE

Indirect Left Recursion:

→ Getting Left recursion because of a non-terminal is replaced by its RHS part.

→ If indirect left recursion exists in the grammar productions, then it should be eliminated to convert grammar into GNF grammar.

Q. Eliminate indirect left recursion from the following grammars.

① S → AA | 0
A → SS | 1
A → AAS | 0S | 1
A → 0SA' | 1A'
A' → ASA' | ε

⇒ S → AA | 0
A → 0SA' | 1A'
A' → ASA' | ε

② A → Bc | a
B → Ab | a
A → Bc | a
B → Ab | a
B → Bcb | ab | a
B → abB' | aB'
B' → cbB' | ε

③ S → Aa | b
A → Ac | Sd | a
S → Aa | b
A → Ac | Sd | a
~~A → aA'~~
~~A' → cA' | ...~~
~~A → Ac | Sd | Aa~~

A → Ac | Aad | bd | a
⇒ A → bdA' | aA'
A' → cA' | adA' | ε

⇒ S → Aa | b
A → ~~Ac | Sd~~ bdA' | aA'
A' → cA' | adA' | ε

Q. Which of the following grammar productions are free from left recursion. **2016**

(a) S → AB (Direct LR)
A → Aa | b
B → c

(b) S → Aa | B (Indirect LR)
A → Bb | Sc | a
B → d

(c) ✓ S → Ab | Bb | c
A → Bd | a
B → d

(d) None.

2014

Q. Which of the following is an equivalent Non-LR grammar for the given grammar.

E → E - T | T
T → T + F | F
F → id

⇒ E → E - T | T ⇒ E → TE', E' → -TE' | ε
T → T + F | F ⇒ T → FT', T' → +FT' | ε
F → id

Q. Convert following LR grammars into GNF grammar.

① S → Sa | b

S → bS'
S' → aS' | ε

Eliminate ε prod. →

S → bS' | b
S' → aS' | a

⇒ GNF.

② S → AA | 0
A → SS | 1

~~S → AA~~
~~A → AAS | a~~
A → 1A'
A' → ASA' | ε

~~S → AA | 0~~
~~A → 1A'~~
~~A' → ASA' | ε~~

~~S → AA | 0~~
~~A → 1A' | 1~~
~~A' → ASA' | AS~~

~~S → 1A' | 1A | 0~~
~~A → 1A' | 0~~
~~A' → 1A'SA' | 1SA' | 1A'S | 1S~~

S → AA | 0
A → 0SA' | 0S | 1A' | 1
A' → ASA' | AS

⇒ S → 0SA'A | 0SA | 1A'A | 1A | 0
A → 0SA' | 0S | 1A' | 1
A' → 0SA'SA' | 0SSA' | 1A'SA' | 1SA' | 0SA'S | 0SS | 1A'S | 1S

NOTE

→ To generate n-length string from the given CNF grammar, required number of productions or derivation steps is (2n-1).

→ To generate n-length string from the given GNF grammar, required number of productions or derivation steps are n.

2DFA or Two-way FA:

→ Two DFA is a DFA, which is capable to move left as well as right side direction.

→ The formal definition of 2-DFA is: $(Q, \Sigma, q_0, F, \delta)$

$$Q \times \Sigma \rightarrow Q \times \{L, R\}$$

→ The expressive power of 2-DFA is same as DFA. Hence, 2-DFA also accepts regular language only.

→ By reading the string, if the 2-DFA halts in final state, that string is accepted, otherwise, the string is ~~rejected~~ not accepted.

Q. Check whether 101001 accepted by the given 2-DFA:

	0	1
→ q_0	(q_0, R)	(q_1, R)
q_1	(q_1, R)	(q_2, L)
q_2	(q_0, R)	$q_2, L)$

$= q_0 101001$ | $= 1 q_1 001$ | $= 101001 q_1$
$= 1 q_1 01001$ | $= 10 q_1 01$
$= 10 q_1 1001$ | $= 1010 q_1$
$= 1 q_2 01001$ | $= 1010 q_2 01$
$= 10 q_0 1001$ | $= 10100 q_0 1$

Accepted

Decision Properties of C.F.G.:

→ The following problems are decidable for context free grammars also:

① Emptiness Problem
② Finiteness Problem
③ Membership Problem

→ The following problems are undecidable for CFGs:

① Equivalence Problem
② Completeness Problem.
③ Ambiguity Problem
④ Elimination of ambiguity Problem
⑤ Regularity Problem (checking whether CFG generates regular language or not)

Emptiness Problem: means checking whether the language generated by given context free grammar is empty or not.

*Algorithm:
① Eliminate all useless variables.
② If the starting symbol is useless, then the grammar generates empty language, otherwise non-empty.

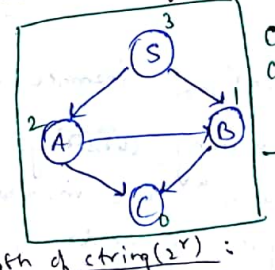
Finiteness Problem: means checking whether the language generated by given grammar is finite or not.

*Algorithm:
① Convert the given grammar into CNF grammar.
② ~~convert CNF grammar graph~~ Construct CNF graph.
③ If there exists loop/cycle, then the grammar generates infinite language, otherwise finite language.

* If the language generated by given grammar is finite, then rank is defined for each non-terminal.
* Rank of a non-terminal is length of the longest path, starting from that non-terminal in CNF graph.
* Rank of a variable 'V' is r [R(V) = r], then max^m length string generated by that variable is 2^r.

Q. Verify following grammar generates finite language or not and also find rank of each non-terminal.

$S \rightarrow AB$
$A \rightarrow BC|a$
$B \rightarrow CC|b$
$C \rightarrow a$



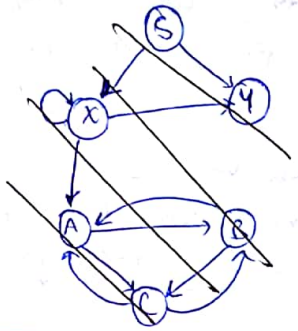
CNF Graph

→ No cycle/loop
∴ Finite language is generated.

Rank(r)
S → 3
A → 2
B → 1
C → 0

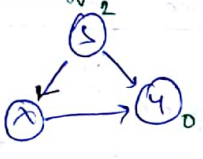
Length of string(2^r):
$2^3 = 8$
$1^2 = 4$
$2^1 = 2$
$2^0 = 1$

Q. $S \to XY$
$X \to \wedge \mid YY \mid b$
$A \to BC$
$B \to AC$
$C \to AB$
$Y \to b$



① Remove useless productions

$S \to XY$
$X \to YY \mid b$
$Y \to b$



Finite

Membership Problem: Checking whether a string "x" is generated from the given grammar or not.

→ Checking whether string "x" is generated from the given grammar or not is known as parsing.
→ Membership Problem is decidable because CYK-algorithm exists.
→ To apply CYK algorithm grammar should be CNF grammar.
→ The time complexity of CYK algorithm is $O(n^3)$.
→ CYK algorithm is also known as parsing algorithm or membership algorithm.
C → Coocke
Y → Younger
K → Kasami

Q① Check whether the string "baaba" is member of following CFG or not:

$S \to AB \mid BC$
$A \to BA \mid a$
$B \to CC \mid b$
$C \to AB \mid a$

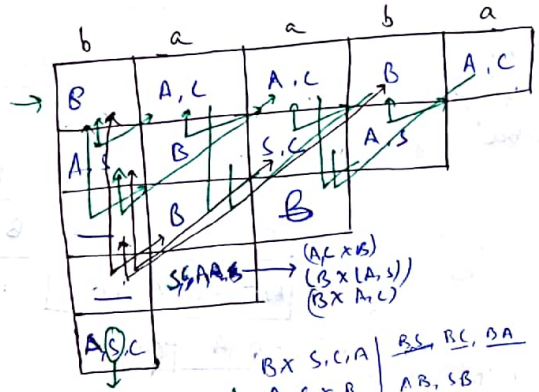
Q② How many substrings of the string "baaba" are members of the given grammar.

Q③ How many substring of the given string "baaba" are not members of the given grammar.

Q④ How many different substrings of the given string are members of the given grammar.

Q⑤ How many substrings of the string "baaba" are generated from the variable "B".

Q⑥ How many different substrings of the string "baaba" are generated from "B" only (variable).

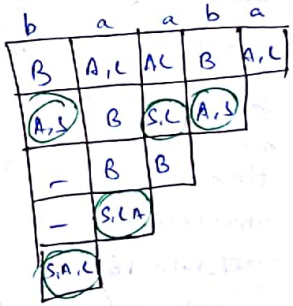


$BX \mid S,C,A \mid \underline{BS}, \underline{BC}, \underline{SA}$
$A,S \times B \mid \underline{AB}, SB$

$(A,C) \times (A,S) \mid S,C \times A,C$
AA
AS
CA
CS
SA
SC
CA
CC

$BX \mid B \mid A,S \times S,C \mid -$
$BB \mid \begin{matrix} AS \\ AC \\ SS \\ SC \end{matrix} \mid \begin{matrix} AB \\ CB \end{matrix} \mid \begin{matrix} BA \\ BS \end{matrix} \mid \begin{matrix} BA \\ BC \end{matrix}$

NOTE: If the last box in the table having starting symbol of the grammar, then the given string is member of the given grammar.



② ba, ab, aaba
No. of boxes having S : 5
5 substrings are members.

③ Total substrings $= \frac{n(n+1)}{2} + 1 = 16$
Substrings not a member $= 16 - 5 = 11$

(4) Different strings = ba, aaba, ab, baaba
= 4

b	a	a	b	a
B	A,C	AC	B	A,C
A,S	B	S,C	A,S	
–	B	B		
–	S,A			
S,A				

(5) Boxes containing only B.
= 5 boxes = 5

(6) Substrings by B = b, b, aa, aab, aba
= 5

Different substrings = 4.

---

(1) "aabbb" is a member

S → AB
A → BBla
B → ABlb

	a	a	b	b	b
→	A	A	B	B	B
→	A, S	S,B	A	A	
→	S, B	A	SB,		
→	A	S, B			
→	S, B				

(1) Which of the following strings are members of given grammar:
(a) aa (c) abbb
(b) aabb (d) bbb

(2) How many different substrings of the string "aabbb" are members of the given grammar.
a b, a a b, b b b, a b b b, a a b b b → Total substrings
No. of different substrings = 5.

(3) How many substrings of the given string "aabbb" are not members of the given grammar.
Total = 16
Not members = 16 − 5 = 11

(4) How many different substrings of the string "aabbb" are generated from "A" only. ~~ε~~, a, ~~bb~~, a b b, b b, a a b b → Different = 4.

PUSHDOWN AUTOMATA:

→ F.A. having one stack is known as PDA.
→ Size of stack in PDA is infinite.
→ Formal definition of PDA is: PDA = $(Q, \Sigma, q_0, F, \$, z_0, \Gamma)$

Q → Finite no. of states
Σ → i/p alphabet
q_0 → Initial state
F → set of final states
δ → transition function → $Q \times \Sigma \cup \{\epsilon\} \times \Gamma \rightarrow Q \times \Gamma^*$
z_0 → initial stack element
Γ → stack alphabet

→ PDA can be represented by using transition diagram or transitions notation.
→ There is only one type of PDA known as language recognizer.
→ PDA can accept/recognize languages in deterministic way or non-deterministic way.
→ The expressive power of N-PDA is more than D-PDA (NPDA > DPDA)
→ By default PDA is N-PDA.
→ PDA practically used in compiler to design parser.
→ There are two types of acceptance methods in PDA known as:
① Empty stack Method ② Final state Method

* Empty stack Method: By reading complete string from left to right, end of the string, if stack of PDA is empty, then the given string is accepted and irrelevant about the no. of final states.

*Final State Method: By reading the complete string from L to R, end of the string, if PDA enters into final state, then the given string is accepted and irrelevant about empty stack.

Q Let "n_1" is no. of language accepted by PDA by using empty stack method. Let "n_2" is no. of languages accepted by PDA by final state method. Then, which of the following is true.

(a) $n_1 = n_2$ (c) $n_1 < n_2$

(b) $n_1 > n_2$ (d) We cannot determine

NOTE

→ No. of languages accepted by empty stack method and no. of languages accepted by final state method is same in PDA (NPDA)

→ No. of languages accepted by final state method is more than empty stack method in D-PDA.

→ Language accepted by PDA are known as CFLs.

→ PDA fails to accept languages which requires more than one stack.
E.g. $a^n b^n c^n$

Q If size of the stack in PDA is restricted to 10000 elements, then language accepted by that PDA is:

(a) Regular only (c) CFLs but not regular

(b) Finite only. (d) Regular but not CFLs.

NOTE

→ PDA can accept regular language and also CFL but not regular. Hence, PDA's expressive power is more than finite automata. (PDA > F.A.)

→ Language accepted by D-PDA is known as Deterministic CFL.

→ Language accepted by N-PDA is known as Non Deterministic CFL.

→ While constructing PDA for the following languages, assume initial stack symbol is Z_0.

Q. Construct PDA for the language $L = \{a^n b^n \mid n \geq 1\}$

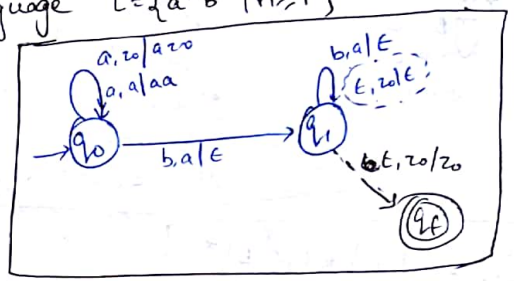
$\delta(q_0, a, Z_0) = (q_0, aZ_0)$] Push a's

$\delta(q_0, a, a) = (q_0, aa)$]

$\delta(q_0, b, a) = (q_1, \epsilon)$] Pop a's

$\delta(q_1, b, a) = (q_1, \epsilon)$]

$\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$ [Empty Stack]
(OR)
$= (q_f, Z_0)$ [Final State]



Acceptance Methods:

→ By reading the input string starting from left side, PDA halts.

→ Input string is accepted by empty stack method, if input is ended and the stack is empty. (Z_0 also popped out from stack)

→ Whenever the machine is halted, if the input string is not ended, then rejected.

→ If the stack is not empty, then also rejected.

*Acceptance By Final State Method:

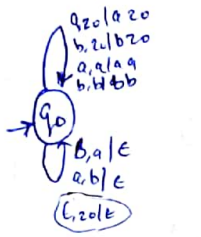
→ A string is accepted if i/p is ended and PDA is in final state. (stack is empty or not).

→ If the input is not ended, then string is rejected.

→ If automata, not in final state, then string is rejected.

Q. Construct the PDA that accepts all strings of a's & b's where number of a's & b's are equal in every string.

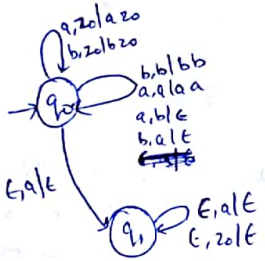
$L = \{ \epsilon, ab, ba, abcab, baba, \ldots \}$



$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

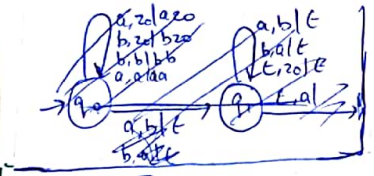
Q. $L = \{ x \mid x \in \{a,b\}^* \; n_a(x) > n_b(x) \}$

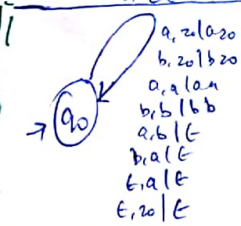
aabaabbbba



$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

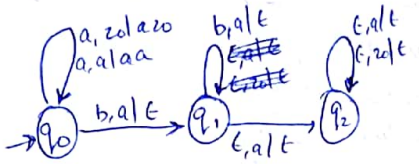
Q $L = \{ x \mid x \in \{a,b\}^* \; n_a(x) \geq n_b(x) \}$





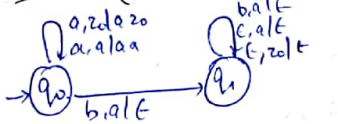
$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, b, z_0) = (q_0, bz_0)$
$\delta(q_0, b, b) = (q_0, bb)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, a, b) = (q_0, \epsilon)$
$\delta(q_0, b, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, a) = (q_0, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

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



$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$

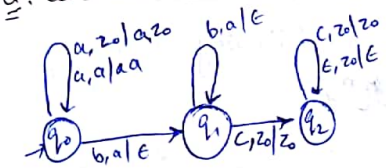
Q. $L = \{ a^n b^m \mid n, m \geq 1 \; n \geq m \}$



$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

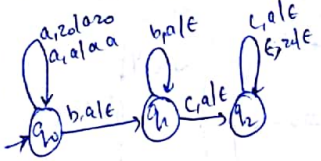
28/07/2017

Q. Construct the PDA for the language $L = \{ a^n b^n c^m / n, m \geq 1 \}$



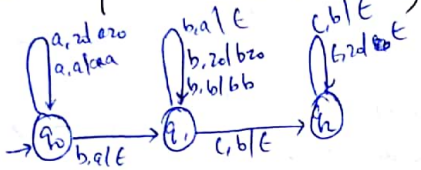
$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_2, z_0)$
$\delta(q_2, c, z_0) = (q_2, z_0)$
$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$.

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



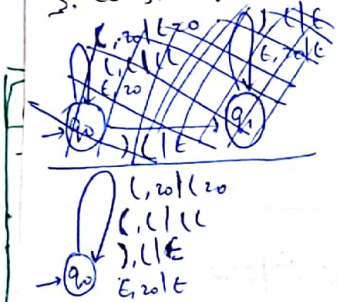
$\delta(q_0, a, z_0) = (q_0, az_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, c, a) = (q_2, \epsilon)$
$\delta(q_2, c, a) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$

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



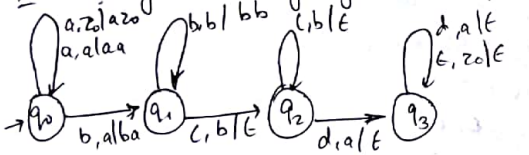
$\delta(q_0, a, z_0) = (q_0, a z_0)$
$\delta(q_0, a, a) = (q_0, aa)$
$\delta(q_0, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, a) = (q_1, \epsilon)$
$\delta(q_1, b, z_0) = (q_1, b z_0)$
$\delta(q_1, b, b) = (q_1, bb)$
$\delta(q_1, c, b) = (q_2, \epsilon)$
$\delta(q_2, c, b) = (q_2, \epsilon)$
$\delta(q_2, \epsilon, z_0) = (q_2, \epsilon)$

Q. Construct PDA that accepts set of all balanced parantheis.



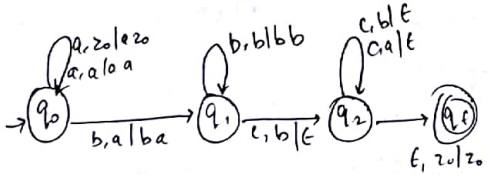
$\delta(q_0, (, z_0) = (q_0, (z_0)$
$\delta(q_0, (, () = (q_0, (()$
$\delta(q_0,), () = (q_0, \epsilon)$
$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

Q. Identify the language accepted by the following PDA:



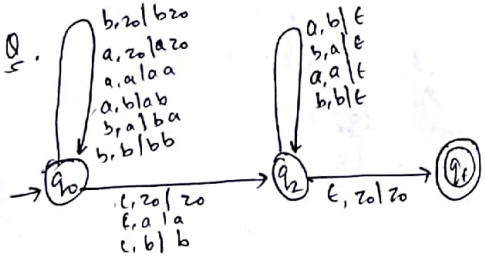
$a^m b^n c^n d^m$

Q.



$a^n b^m c^{m+n}$

Q.



~~$(a+b)^+ c (a+b)^+$~~
~~Ending with c~~

$L = \{ w c Y \mid w, Y \in \{a,b\}^+ , |w| = |Y| \}$

Q. Construct the PDA for the language $L = \{ a^n b^n c^n \mid n \geq 1 \}$ X

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

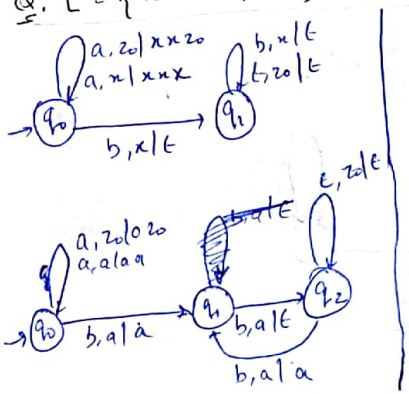
$L = \{ a^n b^n c^{2n} \mid n \geq 1 \}$ X

NOTE

→ The language for which PDA is not possible is known as non-CFL.

Q. $L = \{ a^n b^{n^2} \mid n \geq 1 \}$ aaa bbbbbbbbb X

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

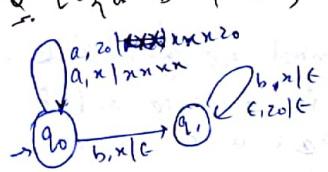


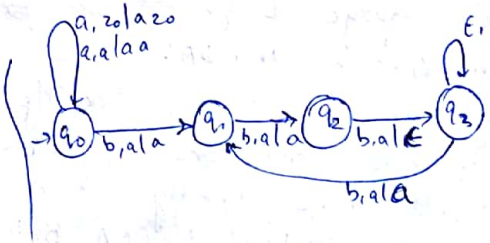
NOTE

→ PDA fails to accept languages in which strings are not having common difference. (not in A.P.).

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

Q. $L = \{ a^n b^{3n} \mid n \geq 1 \}$





Q. ① $L=\{a^{2^n} | n \geq 1\}$ ✗

② $L=\{a^{n!}$ ✗

③ $L=\{a^{n^2}$ ✗

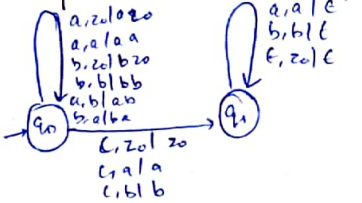
④ $L=\{a^p | p \text{ is prime}\}$ ✗

NOTE

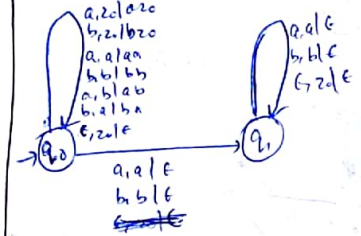
→ There is no difference between finite automata and PDA construction over the languages formed over 1 symbol. Hence, over one symbol languages, if finite automata not possible then, PDA also not possible

Q. $L=\{a^n b^n c^{n+m} | n,m \geq 1\}$ ✗

Q. $L=\{wcw^R | w \in \{a,b\}^*\}$



Q $L=\{ww^R | w \in \{a,b\}^*\}$



Q. $L=\{a^i b^j c^k | i=j \text{ or } j=k\}$

$\{a^n b^n c^m\} \cup \{a^n b^m c^m\}$

NOTE

→ For the language $L=\{ww^R\}$, we cannot construct D-PDA, but we can construct N-PDA.

→ In any PDA, given state, given i/p symbol and given stack symbol, if more than one operation is possible and if we can't determine exactly which is the suitable operation, then we can construct N-PDA only.

→ For the following languages, we can construct N-PDA only, but not D-PDA:

① $L=\{a^i b^j c^k | i=j \text{ or } j=k\}$ NPDA

② $L=\{a^n b^{2n}\} \cup \{a^n b^{3n}\} \cup \{a^n b^{4n}\}$ NPDA

③ $L=\{a^n c b^{2n}\} \cup \{a^n d b^{3n}\}$
$L=\{ca^n b^{2n}\} \cup \{da^n b^{3n}\}$ → DPDA

N-PDA

→ In DPDA, given state, given input symbol and given stack symbol atmost one transition exists. (0 transition or 1 transition).

→ In NPDA, given state, given input symbol and given stack symbol, finite number of transitions may exist.

→ The transition function of N-PDA is:

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \to 2^{Q \times \Gamma^*}$$

→ Every D-PDA is N-PDA but every N-PDA need not be a D-PDA.

→ Expressive power of N-PDA is more than D-PDA.

→ Every N-PDA cannot be converted into D-PDA. (No algorithm exists).

→ Language accepted by DPDA are called as Deterministic CFL (D-CFL)

→ Language accepted by NPDA are called as Non-deterministic CFL. (N-CFL)

→ By default PDA is N-PDA and CFL is N-CFL.

→ Every D-CFL is CFL but every CFL need not be DCFL.

*

Q. Which of the following languages is CFL but not DCFL:

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

(b) $L=\{a^n b c^{2n}\} \cup \{a^n d c^{3n}\}$ DPDA

(c) $L=\{a c a^n b^n\} \cup \{d a^n b^{3n}\}$ DPDA

(d) none ✓

Q. Which of the following language is DCFL:

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

(b) $L=\{a^i b^j c^k \mid i=j \text{ (or) } j=k\}$

(c) $L=\{a^n b^{2n}\} \cup \{a^n b^{3n}\}$

(d) ✓ $L=\{w\#w^R \mid w \in \{a,b\}^*\}$

Q. Construct PDA for $L=\{wcw \mid w \in \{a,b\}^*\}$ ↳ Non-CFL. abbacabba / abbcabb ✗ Not Possible PDA

Q. Construct PDA for $L=\{ww \mid w \in \{a,b\}^*\}$ ✗ abbabb Not Possible

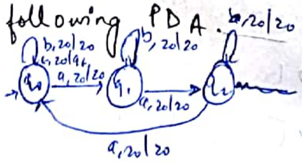
NOTE The ~~following~~ above languages are non-CFL, But their complement is CFL. $(\Sigma^* - L)$

Q. $L=\{ww^R w \mid w \in \{a,b\}^*\}$ ✗ Non-CFL

Q. $L=\{ww^R w w^R \mid w \in \{a,b\}^*\}$ ✗ Non-CFL

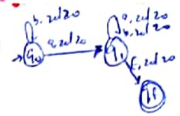
Q. Identify language accepted by the following PDA.

(1)
$\delta(q_0, a, z_0) = (q_1, z_0)$
$\delta(q_0, b, z_0) = (q_0, z_0)$
$\delta(q_1, a, z_0) = (q_2, z_0)$
$\delta(q_1, b, z_0) = (q_1, z_0)$
$\delta(q_2, a, z_0) = (q_0, z_0)$
$\delta(q_2, b, z_0) = (q_2, z_0)$
$\delta(q_0, \epsilon, z_0) = (q_f, z_0)$



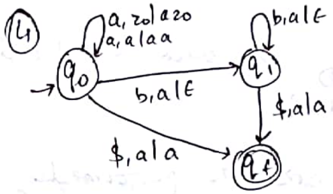
No. of a's divisible by 3.

(2) $\delta(q_0, a, z_0) = (q_1, z_0)$
$\delta(q_0, b, z_0) = (q_0, z_0)$
$\delta(q_1, a, z_0) = (q_1, z_0)$
$\delta(q_1, b, z_0) = (q_0, z_0)$
$\delta(q_1, \epsilon, z_0) = (q_f, z_0)$



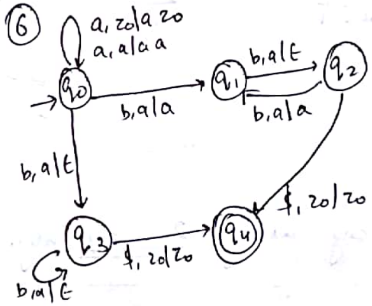
~~Atleast 1 a~~ Ending with a.

(4)



$a^m b^n \mid m>n, m>0, n \geq 0$

(6)



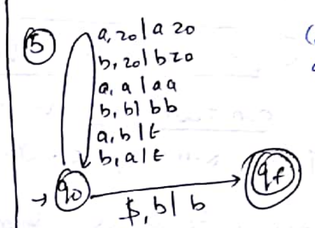
$a^n b^n$ or $a^n b^{2n}$

(3) $\delta(q_0, a, z_0) = (q_0, x z_0)$
$\delta(q_0, a, x) = (q_0, xx)$
$\delta(q_0, b, x) = (q_1, \epsilon)$
$\delta(q_1, b, x) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, x) = (q_1, \epsilon)$

$a^n b^n$

$a^n b^m \mid n > m$, $n, m \geq 1$

(5)

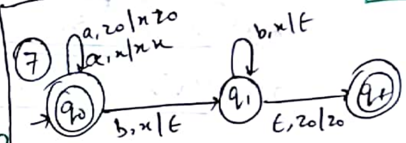


$(a+b)^*$: $n_b > n_a$

$L = \{x \mid x \in \{a,b\}^*, n_b > n_a\}$

2016

(7)



Consider the above PDA. Language accepted by the PDA is.

(a) $L=\{a^n \mid n \geq 0\}$

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

(c) ✓ $L=\{a^n b^n \mid n \geq 0\} \cup \{a^n \mid n \geq 0\}$

(d) $L=\{a^n b^n \mid n \geq 1\}$

Scanned by CamScanner

CFG to PDA Construction:

→ If any grammar is of the form $S \to \alpha \mid \alpha \in (V+T)^*$, then, we can construct PDA as follows:

Step-1: Push the starting symbol "S" into top of the stack.

① $\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$

Step-2. (a) Every non-terminal in the top of the stack is replaced by its R.H.S part. (In reverse order)

(b). For every terminal symbol, pop operation is performed.

(Don't read the i/p.)

2(a) $\delta(q_1, \epsilon, S) = (q_1, \alpha)$

2(b) $\delta(q_1, a, a) = (q_1, \epsilon)$ $\forall a \in T$

Step-3. Acceptance of the input:

③ $\delta(q_1, \epsilon, z_0) = (q, \epsilon)$

For
→ Any CFG, PDA can be constructed by using 2 states only.

NOTE:
→ CFG to PDA construction algorithm always results in N-PDA only. (D-PDA may or may not be possible)

Eg: Q. Construct the PDA for the following grammar:

$S \to aSb \mid ab$

① $\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$

② $\delta(q_1, \epsilon, S) = (q_1, aSb)$ or (q_1, ab)

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

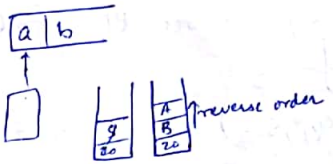
$\delta(q_1, b, b) = (q_1, \epsilon)$

③ $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Q. $S \to AB$

$A \to a$

$B \to b$



① $\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$

② $\delta(q_1, \epsilon, S) = (q_1, AB)$ → first B then A.

$\delta(q_1, \epsilon, A) = (q_1, a)$

$\delta(q_1, \epsilon, B) = (q_1, b)$

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

$\delta(q_1, b, b) = (q_1, \epsilon)$

③ $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Q. $S \to aAB$

$A \to aAb$

$B \to bBb$

① $\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$

② $\delta(q_1, \epsilon, S) = (q_1, aAB)$

$\delta(q_1, \epsilon, A) = (q_1, aA)$ or (q_1, b)

$\delta(q_1, \epsilon, B) = (q_1, bB)$ or (q_1, b)

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

$\delta(q_1, b, b) = (q_1, \epsilon)$

③ $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

GNF CFG to PDA

$A \to aX \mid bY \quad X, Y \in V^*$

PDA Construction Algorithm

Step-1. $\delta(q_0, \epsilon, z_0) = (q_1, Sz_0)$

Step-2: $\delta(q_1, a, A) = (q_1, X)$

$\delta(q_1, b, A) = (q_1, Y)$

Step-3: $\delta(q_1, \epsilon, z_0) = (q_1, \epsilon)$

Q. $S \to aSB \mid bB$
$B \to b$
→ Construct PDA for GNF CFG.

① $\delta(q_0, \epsilon, Z_0) = (q_1, SZ_0)$
② $\delta(q_1, a, S) = (q_1, SB)$
$\delta(q_1, b, S) = (q_1, B)$
$\delta(q_1, b, B) = (q_1, \epsilon)$
③ $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

Q. $S \to aA$
$A \to aABD \mid bB \mid a$
$B \to b$
$D \to d$

① $\delta(q_0, \epsilon, S) = (q_1, SZ_0)$
② $\delta(q_1, a, S) = (q_1, A)$
$\delta(q_1, a, A) = (q_1, ABD)$ or (q_1, ϵ)
$\delta(q_1, b, A) = (q_1, B)$
$\delta(q_1, b, B) = (q_1, \epsilon)$
$\delta(q_1, d, D) = (q_1, \epsilon)$
③ $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

NOTE
→ GNF CFG to PDA construction always results NPDA only.
(DPDA may or may not be possible.)

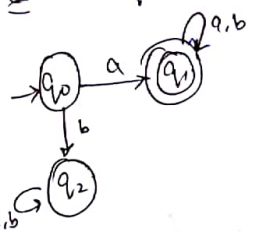
Equivalence Class: No. of equivalence classes corresponding to given regular language is equal to the no. of states of minimal DFA for that language.

→ Equivalence class for a state "q" is the set of all strings from which we can reach state "q" from the initial state.

→ Union of equivalence classes of all final states is the regular language corresponding to the given DFA.

Myhill Nerode Theorem

Q. Find equivalence class of each state of the following DFA:



Equivalence Class $(q_0) = \{\epsilon\}$
Equivalence Class $(q_1) = a(a+b)^*$
Equivalence Class $(q_2) = b(a+b)^*$

Q. Find the total no. of equivalence classes corresponding to following regular languages:

① a^*b^* = 3
② $(a+b)^*abab$ = 5
③ $(a+b)^*aba(a+b)^*$ = 4
④ $(a+b)^*a(a+b)(a+b)$ = 8

29/07/2017

PDA → CFG: Construct PDA to CFG:

$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$
$\delta(q_0, 0, X) = (q_0, XX)$
$\delta(q_0, 1, X) = (q_1, \epsilon)$
$\delta(q_1, 1, X) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, X) = (q_1, \epsilon)$
$\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$

① $S \to [q_0, Z_0, q_0]$ Starting symbol
② $S \to [q_0, Z_0, q_1]$

Pop operations:
$\delta(q_0, 1, X) = (q_1, \epsilon)$
③ $[q_0, X, q_1] \to 1$

④ $\delta(q_1, 1, X) = (q_1, \epsilon)$
$[q_1, X, q_1] \to 1$
⑤ $\delta(q_1, \epsilon, X) = (q_1, \epsilon)$
$[q_1, X, q_1] \to \epsilon$
⑥ $\delta(q_1, \epsilon, Z_0) = (q_1, \epsilon)$
$[q_1, Z_0, q_1] = \epsilon$

Push operations:
$\delta(q_0, 0, Z_0) = (q_0, XZ_0)$
$\delta(q_0, 0, X) = (q_0, XX)$
⑦ $[q_0, Z_0, q_0] \to 0\,[q_0, X, q_0][q_0, Z_0, q_0]$
⑧ $[q_0, Z_0, q_0] \to 0\,[q_0, X, q_1][q_1, Z_0, q_0]$
⑨ $[q_0, Z_0, q_1] \to 0\,[q_0, X, q_0][q_0, Z_0, q_1]$
⑩ $[q_0, Z_0, q_1] \to 0\,[q_0, X, q_1][q_1, Z_0, q_1]$

Let $\delta(q_0, 0, x) \rightarrow (q_0, xx)$

(11) $[q_0, x, q_0] \rightarrow 0[q_0, x, q_0][q_0, x, q_0]$

(12) $[q_0, x, q_0] \rightarrow 0[q_0, x, q_1][q_1, x, q_0]$

(13) $[q_0, x, q_1] \rightarrow 0[q_0, x, q_0][q_0, x, q_1]$

(14) $[q_0, x, q_1] \rightarrow 0[q_0, x, q_1][q_1, x, q_1]$

$[q_0, z_0, q_0] = A$
$[q_0, z_0, q_1] = B$
$[q_0, z_0, q_0] = C$
$[q_1, z_0, q_1] = D$

$[q_0, x, q_0] = E$
$[q_0, x, q_1] = F$
$[q_1, x, q_0] = G$
$[q_1, x, q_1] = H$

$S \rightarrow A \mid B$
$F \rightarrow 1$
$H \rightarrow 1 \mid \epsilon$
$D \rightarrow \epsilon$
$A \rightarrow 0EA \mid 0FC$
$B \rightarrow 0EB \mid 0FD$
$E \rightarrow 0EE \mid 0FG$
$F \rightarrow 0EF \mid 0FH$

$S \rightarrow B$
$B \rightarrow 0FD$
$D \rightarrow \epsilon$
$F \rightarrow 0FH \mid 1$
$H \rightarrow 1 \mid \epsilon$

$S \rightarrow B$

Q. $\delta(q_0, 1, z_0) = (q_0, xz_0)$
$\delta(q_0, 1, x) = (q_0, xx)$
$\delta(q_0, 0, x) = (q_1, x)$
$\delta(q_1, 1, x) = (q_1, \epsilon)$
$\delta(q_0, 0, z_0) = (q_0, z_0)$
$\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$

(3) $\delta(q_1, 1, x) = (q_1, \epsilon)$ (Pop)
$[q_1, x, q_1] \rightarrow 1$

(4) $\delta(q_0, \epsilon, z_0) = (q_0, \epsilon)$
$[q_0, z_0, q_0] \rightarrow \epsilon$

(1) $S \rightarrow [q_0, z_0, q_0]$
(2) $S \rightarrow [q_0, z_0, q_1]$

Bypass operations

$\delta(q_0, 0, x) = (q_1, x)$

(5) $[q_0, x, q_0] = 0[q_1, x, q_0]$
(6) $[q_0, x, q_1] = 0[q_1, x, q_1]$

$\delta(q_1, 0, z_0) = (q_0, z_0)$
(7) $[q_1, z_0, q_0] \Rightarrow 0[q_0, z_0, q_0]$
(8) $[q_1, z_0, q_1] \Rightarrow 0[q_0, z_0, q_1]$

Push:

$\delta(q_0, 1, z_0) = (q_0, x z_0)$ (Number of symbols = number of blocks)

(9) $[q_0, z_0, q_0] \rightarrow 1[q_0, x, q_0][q_0, z_0, q_0]$
(10) $[q_0, z_0, q_0] \rightarrow 1[q_0, x, q_1][q_1, z_0, q_0]$
(11) $[q_0, z_0, q_1] \rightarrow 1[q_0, x, q_0][q_0, z_0, q_1]$
(12) $[q_0, z_0, q_1] \rightarrow 1[q_0, x, q_1][q_1, z_0, q_1]$

$\delta(q_0, 1, x) = (q_0, xx)$
(13) $[q_0, x, q_0] \Rightarrow 1[q_0, x, q_0][q_0, x, q_0]$
(14) $[q_0, x, q_0] \Rightarrow 1[q_0, x, q_1][q_1, x, q_0]$
(15) $[q_0, x, q_1] \Rightarrow 1[q_0, x, q_0][q_0, x, q_1]$
(16) $[q_0, x, q_1] \Rightarrow 1[q_0, x, q_1][q_1, x, q_1]$

$S \rightarrow A$
$A \rightarrow 1FC \mid \epsilon$
$C \rightarrow 0A$
$F \rightarrow 1FH \mid 0H$
$H \rightarrow 1$

$[q_0, z_0, q_0] \rightarrow A$
$[q_0, z_0, q_1] \rightarrow B$
$[q_0, z_0, q_0] \rightarrow C$
$[q_1, z_0, q_1] \rightarrow D$
$[q_0, x, q_0] \rightarrow E$
$[q_0, x, q_1] \rightarrow F$
$[q_1, x, q_0] \rightarrow G$
$[q_1, x, q_1] \rightarrow H$

$S \rightarrow A \mid B$
$A \rightarrow \epsilon \mid 1EA \mid 1FC$
$B \rightarrow 1EB \mid 1FD$
$C \rightarrow 0A$
$D \rightarrow 0B$
$E \rightarrow 1FG \mid 1EE \mid 0EF$
$F \rightarrow 1EE \mid 1FH \mid 0H$
$GH \rightarrow 1$

Context Free languages:

→ The language for which PDA exists, is known as CFL.

(OR)

The language for which CFG exists.

→ Every CFL may or may not be regular.

→ Every regular language should be CFG.

Q. Which of the following language is

(1) CFL and regular
(2) CFL but not regular
(3) Not CFL.

(a) $L=\{a^n b^n c^n \mid n \le 10\}$ (1)

(b) $L=\{a^n b^n c^m \mid n \neq m\}$ (3)

(c) $L=\{a^n b^m c^n \mid n \neq m,\ n,m \le 100\}$ (1)

(d) $L=\{a^n b^m \mid n+m=100\}$ (1)

(e) $L=\{a^n b^m \mid n-m=4,\ n=m+4\}$ (2)

(f) $L=\{a^n b^m \mid n+m=100\}$ (1)

(g) $L=\{a^n b^m \mid n/m=5\}$ (2)

(h) $L=\{a^n b^m \mid n=2m\}$ (2)

(i) $L=\{a^n b^m \mid n=m^2\}$ (3)

(j) $L=\{a^n b^m \mid n=2m+1\}$ (2)

(k) $L=\{a^n b^m \mid n \neq 2m\}$ (2)

(l) $L=\{a^{n!} b^{n!} \mid n \ge 1\}$ (3)

(m) $L=\{a^n b^n c^{n+m} \mid n,m \ge 1\}$ (3)

(n) $L=\{a^{n+m} b^{n+m} c^m \mid n,m \ge 1\}$ (3)

(o) $L=\{a^n b^n c^{2n} \mid n \ge 1\}$ (3)

(p) $L=\{a^{2n} b^m c^n \mid n \ge 1\}$ (3)

(q) $L=\{a^n b^{2n} c^n \mid n \ge 1\}$ (3)

(r) $L=\{a^n b^{2n} c^{3m} \mid n,m \ge 1\}$ (2)

(s) $L=\{a^n b^{2n} c^m \mid n \neq m\}$ (3)

(t) $L=\{a^n b^n c\, a^n b^m \mid n,m \ge 1\}$ (2)

(u) $L=\{a^n b^n c\, a^n b^n \mid n \ge 1\}$ (3)

(v) $L=\{a^i b^j c^k \mid i=j \neq k\}$ (2)

(w) $L=\{a^i b^j c^k \mid j=i \neq k\}$ (2)

~~(x) $L=\{a^i b^j c^k \mid j=i \neq k\}$~~

~~(z) $L=\{a^i b^j c^k \mid j=\max(i,k)\}$~~

(y) $L=\{a^i b^j c^k \mid i>j \text{ (or) } j>k\}$ (2)

(z) $L=\{a^i b^j c^k \mid i>j>k\}$ (3)

(27) $L=\{a^i b^j c^k \mid i=j^2+k^2\}$ (3)

(28) $L=\{a^i b^j c^k d^l \mid i=j \text{ and } k=l\}$ (2)

(29) $L=\{a^i b^j c^k j^l \mid i=l \text{ and } j=k\}$ (2)

(30) $L=\{a^i b^j c^k d^l \mid j=k \text{ and } j=l\}$ (3)

(31) $L=\{a^i b^j c^k d^l \mid i=k \text{ (or) } j=l\}$ (2)

(32) $L=\{a^i b^j c^k d^l \mid i=2k \text{ (or) } j \neq l\}$ (2)

(33) $L=\{a^i b^j c^k d^l \mid i>j>k>l\}$ (3)

(34) $L=\{w \times w \mid w \in \{a,b\}^*\}$ (3)

(35) $L=\{wxw \mid w,x \in \{a,b\}^*\}$ (1)

(36) $L=\{www \mid w \in \{a,b\}^*\}$ (3)

(37) $L=\{ww^R w \mid w \in \{a,b\}^*\}$ (3)

(38) $L=\{ww^R ww^R \mid w \in \{a,b\}^*\}$ (3)

(39) $L=\{ww^R w w^R w \mid w \in \{a,b\}^*\}$ (1)

(40) $L=\Sigma^* - \{ww \mid w \in \{a,b\}^*\}$ (2)

(41) $L=\Sigma^* - \{wcw \mid w \in \{a,b\}^*\}$ (2)

(42) $L=\{x \mid x \in \{a,b\}^*,\ n_a(x) > n_b(x)+1\}$ (2)

(43) $L=\{x \mid x \in \{a,b,c\}^+,\ n_a(x)=n_b(x)=n_c(x)\}$ (3)

(44) $L=\{x \mid x \in \{a,b,c\}^*,\ n_a(x)=n_b^2(x)+n_c^2(x)\}$ (3)

(45) $L=\{a^{n!} \mid n \ge 1\}$ (3)

(46) $L=\{a^p \mid p \text{ is prime}\}$ (3)

(47) $L=\{1,2,4,8,\ldots 2^n \ldots\}$ (3)
numbers in unary.

(48) $L=\{a^k \mid k \text{ is even number}\}$ (1)

(49) Set of all even length palindrome strings of Hindi language (2)

(50) Set of all odd length palindrome strings of English language. (2)

(51) Set of all palindrome strings of Malayalam. (2)

(52) Set of all balanced parenthesis. (2)

(53) Set of all lexical errors produced by compiler. (1)

(54) Set of all syntax errors produced by compiler. (2) CFL

(55) Set of all semantic errors produced by compiler. (3) CSL

(56) C language is ? CSL (3)

(57) JAVA language is? CSL (3)

(58)

Detection of CFL:

1. A finite language is always regular and CFL.
2. Any infinite language require more than one stack (more than one comparison) is non-CFL.
3. All palindrome languages are CFL.
4. Any infinite language, strings are not having common difference then it is non-CFL.
5. There is no difference between CFL and regular, the language formed over one symbol.
6. Any infinite language having one comparison, ~~then, it is~~ but (Eg: ww) ~~non-CFL~~. doesn't satisfies stack property, than it is non-CFL.
7. Union of two CFLs is CFL.
8. ~~Eat~~ Concatenation of two CFLs is CFL.

Q. Consider the following languages. Which of the following is true?

$L_1 = \{a^p \mid p \text{ is prime}\}$

$L_2 = \{a^n b^m c^{2m} \mid n, m \geq 1\}$

$L_3 = \{a^n b^n c^{2n} \mid n \geq 1\}$

$L_4 = \{a^n b^n \mid n \geq 1\}$

(a) L_1, L_2, L_3 are non-CFL.
(b) L_2, L_3 and L_4 are CFL.
(c) L_1, L_2, L_4 are CFL.
(d) None of these

Q. Consider the following 2 languages:

$L_1 = \{a^n b^m c^{m+n}, m, n \geq 1\}$

$L_2 = \{a^n b^n c^{2n}, n \geq 1\}$

Which of the following is true?
(a) Both L_1 & L_2 are CFL
(b) Both L_1 and L_2 are non-CFL
(c) L_1 is CFL, L_2 is non-CFL
(d) L_1 is non CFL, L_2 is ~~non~~ CFL

Q. Which of the following are CFL?

(a) $L_1 = \{0^i 1^j \mid i \neq j\}$
(b) $L_2 = \{0^i 1^j \mid i = j\}$
(c) $L_3 = \{0^i 1^j \mid i > j\}$
(d) $L_4 = \{0^i 1^j \mid i \neq 2j\}$

(e) All

Q. $L_1 = \{a^p b^q c^r \mid p = 3(q+r)\}$ $p = 3q$ $a^{3q} b^q c^r$

$L_2 = \{a^p b^q c^r \mid p < q \text{ (or) } q < r\}$

$L_3 = \{a^n w w^R \mid w \in \{a,b\}^*\}$

$L_4 = \{wcw^R \mid w \in \{a,b\}^*\}$

$L_5 = \{a^n b^{3n}\} \cup \{a^n b^{4n}\}$

Which of the following is true?

(a) L_1 and L_2 are D-CFL
(b) L_4 is D-CFL, remaining are CFL but not D-CFL.
(c) ~~L_3, L_4, L_5 are CFL but not D-CFL~~
(d) L_1, L_4 are DCFL, remaining are CFL but not D-CFL
(e) All are D-CFL.

Q. $L = \{a^n b^n \mid n \geq 1, n \neq 13\}$ is

(a) DCFL
(b) CFL but not DCFL
(c) Non CFL
(d) Regular.

$\{a^n b^n \mid n \geq 1\} - \{a^n b^n \mid n = 13\}$

DCFL − Regular

= DCFL

Q. Consider the following languages:

$L_1 = \{wxw \mid w \in \{a,b\}^*\}$

$L_2 = \{wxw \mid w, x \in \{a,b\}^*\}$

$L_3 = \{ww^R x \mid w, x \in \{a,b\}^*\}$

$L_4 = \{w x w^R \mid x, w \in \{a,b\}^*\}$

Which of the following is true?

(a) All are CFL
(b) All are non-CFL
(c) L_1, L_2 are non-CFL, L_3 & L_4 are CFL
(d) L_4 ~~and~~ is non-CFL, L_2, L_3, L_4 are CFL

Q. Which of the following languages is CFL but not D-CFL.

(a) $\{a^i b^j c^k \mid i \neq j \text{ (or) } j \neq k\}$
(b) $\{a^i b^j c^k d^l \mid i = 2j \text{ (or) } j = l\}$
(c) $\{a^n b^m c^k \mid n > m \text{ (or) } m < k\}$
(d) All

Closure Properties of CFL and DCFL:

① **Union Operation:** $L_1 = CFL = \{a^n b^n \mid n \geq 1\} = S_1 \rightarrow a\,S_1\,b \mid ab$

$L_2 = CFL = \{c^m d^m \mid m \geq 1\} = S_2 \rightarrow c\,S_2\,d \mid cd$

→ Let L_1 and L_2 are 2 CFL, then their union is always CFL. because we can construct CFG for $L_1 \cup L_2$. as $S \rightarrow S_1 \mid S_2$

Hence, CFLs are closed under union operation.

→ The union of 2 D-CFL may or may not give D-CFL. Hence D-CFLs are not closed under union operation.

Eg: $\{a^n b^n \mid n \geq 1\} \cup \{a^n b^n c^m \mid n, m \geq 0\} = \{a^n b^n c^m \mid n, m \geq 1\}$

DCFL DCFL DCFL

$\{a^n b^n c^m\} \cup \{a^n b^m c^m\} = \{a^n b^m c^k \mid n = m \text{ or } m = k\}$

DCFL DCFL NCFL

② **Subset Operation:** Subset of a CFL is may or may not be a CFL. Hence, CFLs are not closed under subset operation.

Eg: $\{a^n b^n c^n \mid n \geq 1\} \subseteq \{a^n b^n c^m \mid n, m \geq 0\}$

Non-CFL CFL

→ Subset of DCFL may or may not be D-CFL. Hence, D-CFLs are not closed under subset operation.

③ **Concatenation Operation:** Concatenation of 2 CFL is always CFL, because we can construct CFG for 2 CFLs concatenation.

Hence, CFLs are closed under concatenation operation.

Eg. $L_1 = \{a^n b^n \mid n \geq 1\} = S_1$ | $L_1 . L_2 \Rightarrow S \rightarrow S_1 . S_2$

$L_2 = \{c^m d^m \mid m \geq 1\} = S_2$ | (CFL)

→ Concatenation of 2 D-CFL may or may not be D-CFL. Hence, D-CFLs are not closed under concatenation operation.

Eg: $L_1 = \{x \mid x \in \{a,b\}^+\}$ DCFL

$L_2 = \{w_c w^R \mid w \in \{a,b\}^+\}$ DCFL

$L_1 . L_2 = \{x . w_c w^R \mid w, x \in \{a,b\}^+\}$ NCFL [CFL but not DCFL.

④ **Intersection Operation:** Intersection of 2 CFL, may or may not be CFL.

→ Hence, CFLs are not closed under intersection operation.

$\{a^n b^n c^m \mid n, m \geq 0\} \cap \{a^n b^n \mid n \geq 1\} = \{a^n b^n \mid n \geq 1\}$

▷ CFL ▷ CFL ▷ CFL

$\{a^n b^n c^m \mid n, m \geq 1\} \cap \{a^n b^m c^m \mid n, m \geq 1\} = \{a^n b^n c^m \mid n = m \geq 1\}$

▷ CFL ▷ CFL Non-CFL.

→ Intersection of 2 DCFL may or may not be DCFL, Hence, DCFLs are not closed under intersection operation.

⑤ **Intersection with Regular:** CFL intersection with regular is always CFL. (May or may not be regular)

Hence, CFLs are closed under intersection with regular language operation.

Eg. $\{a^n b^n \mid n \geq 1\} \cap (a+b)^* = \{a^n b^n \mid n \geq 1\}$

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

CFL ∩ Reg. = CFL always (may or may not be Reg.)

DCFL ∩ ~~DCFL~~ Reg ↳ DCFL.

any language. $X \cap \text{Reg} = X$ always → Hence, this operation is closed under all formal languages.

→ DCFL intersection with regular is always DCFL. (May or may not be regular).
Hence, DCFLs are closed under intersection with regular language operation.

Q. Consider the following 2 languages: (2017)
$L_1 = \{a^n b^n c^m \mid n, m \geq 1\}$
$L_2 = \{a^n b^m c^m \mid n, m \geq 1\}$
$L_3 = L_1 \cup L_2$
$L_4 = L_1 \cap L_2$

Which of the following is true?
(a) All are CFLs
(b) L_1, L_2 are CFL, L_3 & L_4 are non-CFL
(c) L_1, L_2 & L_3 are CFL; L_4 is non-CFL
(d) All are non-CFL.

---

⑥ Kleene Closure Operation: Q. Consider the following languages:
$L_1 = \{a^i b^j c^k \mid i, j, k \geq 1\}$
$L_2 = \{a^n b^n c a^m b^m \mid n, m \geq 0\}$
$L_3 = L_1 \cap L_2$

Which of the following is true:
(a) L_3 is CFL and regular.
(b) L_3 is CFL but not regular.
(c) L_3 is non-CFL
(d) L_2 and L_3 are non-CFL.

---

⑥ Kleene Closure Operation: Kleene closure of CFL is always CFL. Hence, CFLs are closed under Kleene closure operation. [For L_1^* grammar is $S \to SS \mid \in$].

→ Kleene closure of DCFL may or may not be DCFL. Hence DCFLs are not closed under Kleene closure operation.

Eg. $L = \{wcw^R x \mid w, x \in \{a,b\}^+\} \Rightarrow \{wcwRxwcw^Rx \cdots\}$

---

⑦ Positive closure: The positive closure of CFL is always CFL, because we can construct CFG for L_1^+ as:
$S \to SS \mid S_1$. Hence, CFLs are closed under positive closure.

→ Positive closure of DCFL may or not be DCFL. Hence, DCFLs are not closed under positive closure operation.

---

⑧ Complement Operation: Complement of CFL, may or may not be CFL. Hence, CFLs are not closed under complement operation.

Eg. Non CFL ⇒ $L = \{ww \mid w \in \{a,b\}^*\}$

CFL ⇒ $\Sigma^* - \{ww \mid w \in \{a,b\}^*\}$

NOTE: Complement of CFL is Recursive language.

→ The complement of DCFL is always DCFL. Hence, DCFLs are closed under the complement operation.

→ If a language L is DCFL and there exists a DPDA for it, then, $\Sigma^* - L$ is also DCFL because we can construct complement of DPDA by interchanging final states as non-final states and non-final states as non-final states.

---

⑨ Difference Operation: Difference of 2 CFLs may or may not be a CFL. Hence, CFLs are not closed under difference operation.

→ The difference of 2 DCFL may or may not be DCFL. Hence, DCFLs are not closed under difference operation. [If any language is closed under intersection and complement then it is also closed under difference.]

$L_1 = CFL$
$L_2 = CFL$
$L_1 - L_2 = L_1 \cap \overline{L_2}$
$= CFL \cap CFL$

$\because DCFL_1 \cap \overline{DCFL_2}$
$= DCFL_1 \cap DCFL_2$ ✗
$=$.

---

⑩ Difference with Regular: CFL's difference with regular is always CFL [may or may not be regular].
Hence, CFLs are closed under difference with regular language operation.

* X − Regular = X always! *
$X \cap \overline{Reg} = X$
$X \cap Reg = X$

Hence, difference with regular lang. operation is closed under all formal languages.

→ DCFL difference with regular ~~is enough~~ always DCFL. Hence, DCFLs are closed under difference with reg. operation.

(11) **Regular Language Difference with CFL:** Regular language difference with CFLs may or may not be CFL. Hence, CFLs are not closed under regular difference with CFL operation.

Regular difference with DCFL:

$\boxed{Reg - DCFL = DCFL}$

$Reg \wedge DCFL^c$

$Reg \wedge DCFL = DCFL$

Regular lang. difference with DCFL is always DCFL. Hence, DCFLs are closed under regular language difference with DCFL operation.

(12) **Reversal Operation:** Reverse of a CFL is always CFL. Hence, CFLs are closed under reversal operation.

L_1 is CFL and S_1 is CFG for L. Then L^R is also CFL because we can construct CFG by reversing R.H.S. part of S productions.

→ **Reverse of a DCFL** may or may not be DCFL. Hence, DCFLs are not closed under reversal operation.

$L_1 = \{a^n b^n \mid n \geq 1\}$ $S \to aSb \mid ab$ — $L = \{w c w^R \mid w, x \in \{a,b\}^+\}$ DCFL

$L_1^R = \{b^n a^n \mid n \geq 1\}$ $S \to bSa \mid ba$ — $L^R = \{x w^R c w \mid w, x \in \{a,b\}^+\}$ CFL but Not DCFL

Q. L_1 and L_2 are DCFLs, L_3 and L_4 are CFL. L_5 is regular, then which of the following is false.

(a) $(L_3 \cup L_4)^c \Rightarrow$ recursive True

(b) $(L_1^c - L_5) =$ DCFL True

(c) $(L_3^R \wedge L_4) \wedge L_5 \Rightarrow$ CFL False

(d) $L_3^c \cap L_4^c \Rightarrow$ CFL False

Q. Let P is CFL and Q is regular, then which of the following is always regular:

(a) $P \wedge Q$

(b) $P - Q$

(c) $\Sigma^* - P$

(d) $\Sigma^* - Q$

Q. L_1 and L_2 are 2 CFL and R is regular. Which of the following is false:

(a) $L_1 \cup L_2$ is CFL True

(b) $L_1 - R$ is CFL True

(c) $L_1 \cap L_2 \Rightarrow$ CFL False

(d) $L_2 - R \Rightarrow$ CFL True

Very Imp.

	CFL	DCFL
(1) Union	✓	✗
(2) .	✓	✗
(3) *	✓	✗
(4) Positive Closure	✗	✗
(5) $\wedge$	✓	✓
(6) $\wedge$ Reg.	✗	✓
(7) $\Sigma^* - L$	✗	✗
(8) $\subseteq$	✗	✗
(9) $L_1 - L_2$	✓	✓
(10) L - Reg.	✗	✓
(11) Reg - L	✓	✗
(12) L^R	✓	✗
(13) Substitution	✓	✗
(14) Homomorphism	✓	✓
(15) Inverse Homo.	✓	✓
(16) Init	✗	✗
(17) L_1/L_2 (Quotient)	✓	✓
(18) L / Reg.	✓	✓
(19) L ∪ Reg.	✗	✗

Q. Let L_1 and L_2 are any two CFL and L_3 and L_4 are 2 regular languages. Then which of the following is always CFL.

(1) $(L_1 \cup L_2)^* \wedge (L_3 \wedge L_4)^R$ ✓

(2) $(L_1 \cup L_3)^R \vee (L_2 \wedge L_4)^+$ ✓

(3) $(L_1^R \wedge L_4^*)^R \wedge (L_2 \wedge (\Sigma^* - L_3))^R$ ✗

(4) $[\Sigma^* - (L_1 \wedge L_4)]^R \wedge (L_2 \wedge L_4)^+$ ✗

(5) $(L_1 \cdot L_2) \wedge (L_3 \cdot L_4)^R$ ✓

(6) $(L_1 \cdot L_3)^R \wedge (L_2 \cdot L_4)^*$ ✗

(7) $(L_1 \cdot L_3) \cdot (L_2 \cdot L_4)$ ✓

(8) $(L_1 - L_3) - (L_2 - L_4)$ ✗

(9) $(L_1 - L_4)^* - (\Sigma^* - L_3)^R$ ✓

(10) $(L_1 - L_3) - L_4$ ✓

(11) $(L_1 / L_3) \cdot (L_2 / L_4)$ ✓

(12) $(L_1 / L_3) / (L_2 / L_4)$ ✗

(13) $L_5 \subseteq L_1$ ✗

$L_5^R = ?$ Not closed. Not always

(14) $L_6 \subseteq L_2$

$L_2 \wedge L_6 = ?$ ✗ Not always.

Grammars For DCFL : # (LL(k), LR(k) Grammars)

→ The grammars for DCFL are LL(k) grammars and LR(k) grammars.

→ LL(k) and LR(k) grammars are grammars suitable for parsing.

→ Parser is a D-PDA.

Hence, Language generated by LL(k), LR(k) grammars are DCFL.

→ Every LL(k), LR(k) grammar is Unambiguous grammar.

→ Hence, the language by LL(k) grammars, LR(k) grammar is also unambiguous.

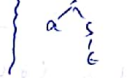
→ Every DCFL is unambiguous language.

→ Every regular language is DCFL, hence, every regular lang. also unambiguous language.

→ Regular language and DCFL are not inherently ambiguous language.

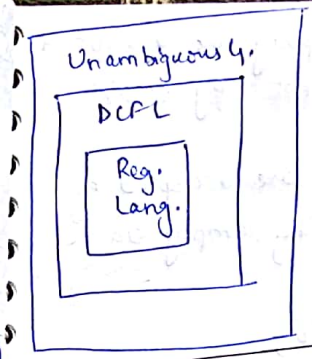
→ Every regular language is unambiguous, but regular grammar can be ambiguous.

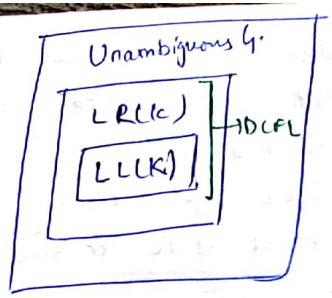
Eg. $S \to aS \mid a \mid \epsilon$



→ For every DCFL, LR(k) grammar should exist [LL(k) grammar may or may not exist].

→ For every regular language, LR(k) grammar should exist [LL(k) grammar may or may not exist]





Q. Which of the following is true?

(a) All unambiguous grammars generate regular languages. False

(b) Every regular grammar is unambiguous grammar. False.

(c) Inherently ambiguous language is a language for which there exists atleast one ambiguous grammar. False

(d) Every LL(k) and LR(k) grammar is unambiguous and regular language and DCFL can never be inherently ambiguous. True

Q. Which of the following language is inherently ambiguous.

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

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

(c) $L = \{a^i b^j c^k \mid i = j \text{ (or) } j = k\}$

(d) $L = \{a^n b^{2m} \mid n, m \geq 1\}$

NOTE:

→ No. of languages accepted by DPDA by final state method is more than no. of languages accepted by DPDA by empty stack method.

→ A language 'L' is accepted by DPDA with empty stack method, if that language having prefix property.

→ If a language "L" (DCFL or regular) not having prefix property then, it is accepted by DPDA, final state method but not by empty stack method.

→ A language 'L' is said to be having prefix property if no proper prefix of any string present in the language.

→ All regular languages not having prefix property are accepted by DPDA final state method only but not by empty stack method.

→ All DCFLs not having prefix property are accepted by DPDA, final state method only but not by empty stack method.

$L_1 = \{a^n b^n \mid n \geq 1\}$

✓ $\{ab, aabb, aaabbb, \ldots\}$

ε	ε	ε
a	aa	aa
	aab	aaa
		aaab
		aaabbb

Prefix Property ✓

$L_2 = \{a^n b^n \mid n \geq 0\}$

✗ $\{\varepsilon, ab, aabb, \ldots\}$

ε	ε
a	a
	aa
	aab

Prefix Property ✗

Decision Properties of DCFL:

→ The following ~~prope~~ problems are decidable under DCFL:

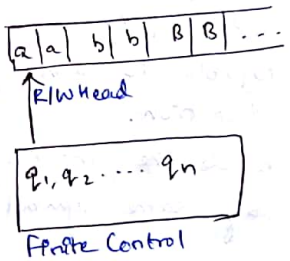
1. Membership Problem: CYK algorithm
2. Finiteness Problem: CNF graph
3. Emptiness Problem :
4. Completeness Problem: $\Sigma^* - L$ (Complement is closed)
5. Regularity Problem

~~6~~ Equivalence Problem of DCFL is undecidable.

23/07/2017

TURING MACHINE

→ Turing machine is a mathematical model that represents the behaviour of general purpose computer.



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

- Q → Finite no. of states
- Σ → i/p alphabet
- q_0 → Initial state
- F → set of final states
- δ → Transition function $Q \times \Gamma \to Q \times \Gamma \times \{L, R\}$
- B → Blank symbol
- Γ → Tape alphabet

Types of Turing Machine:

1. i/p → Language Recognizer → Accept / Reject
2. i/p → Output Generator (or) Transducer → Output
3. Language Generator (or) Enumerator → Language

Notation / Representation:

→ Turing m/c usually represented by using transition diagram or transition table.

→ The following are the capabilities of Turing Machines.

① Infinite Length Tape: Turing m/c tape is one side close and other side open.

② Turnaround Capability: Turing m/c capable to turn left as well as right side direction.

③ Read / Write Capability: Turing m/c reads the input symbol and replaces it by same symbol or some other symbol.

→ The problem which are not solvable by turing m/c are not solvable by computer. [Undecidable Problem].

→ Problems which are solvable by Turing M/c [Halting] are also solvable by computer. [Decidable Problems]

→ Turing machines are used to compute about P-class problems, N-P complete problems, N-P Hard problems,

→ Languages accepted by Turing machine are known as REL. (Recursive Enumerable Languages).

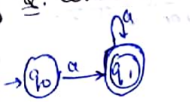
→ The language for which no Turing m/c exists are known as not REL.

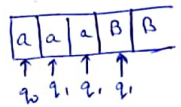
Turing Machine as Language Recogniser:

① Acceptance Method: By reading the string, TM may halt or may not halt.

- By reading the string, if the T.M. halts in final state. Then, the given string is accepted.
- By reading the string, T.M. halts in non-final states, then given string is rejected.
- By reading the string. T.M. enters into infinite loop, then, we cannot conclude about input string, whether it is accepted or not.

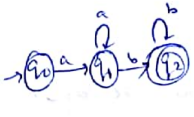
Q. Construct the T.M. for the language $L = \{a^n \mid n \geq 1\}$.



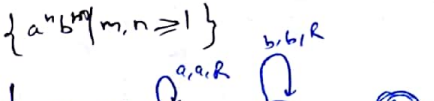


	a	B
→ q_0	(q_1, a, R)	
q_1	(q_1, a, R)	(q_f, B, R)
q_f	Halt	Halt

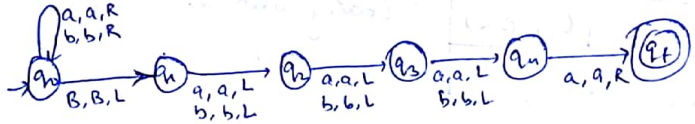
Q. Construct the Turing Machine for $L = \{a^n b^m \mid m, n \geq 1\}$



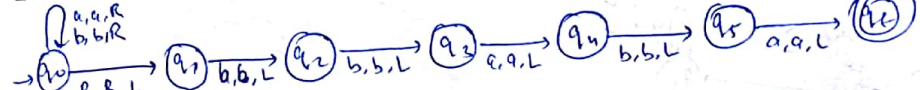
	a	b	B
→ q_0	q_1, a, R		
q_1	q_1, a, R	q_2, b, R	
q_2		q_2, b, R	q_f, B, R
q_f	← Halt →		



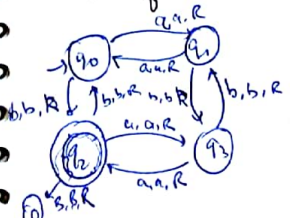
Q. Construct the T.M. that accepts all strings of a's and b's, where 4th i/p symbol is "a" while reading the string from R.H.S. (Deterministic T.M.) $(a+b)^* a(a+b)(a+b)(a+b)$



Q. Construct the T.M. that accepts the reg-ex. $(a+b)^* abaabb$.



Q. Construct the TM that accepts all strings of a's & b's where no. of a's are even and no. of b's are odd in every string.



	a	b	B
→ q_0	q_1, a, R	q_2, b, R	
q_1	q_0, a, R	q_3, b, R	
q_2	q_3, a, R	q_0, b, R	q_f, B, R
q_3	q_2, a, R	q_1, b, R	
q_f	HALT		

Q.

	a	b	B
→ q_0	q_1, a, R	q_1, b, R	
q_1	q_2, a, R	$q_2 b, R$	
q_2	H	q_3, b, R	
q_3	q_3, a, R	$q_3 b, R$	q_f, B, R
q_f	HALT		

3rd symbol is b from L.H.S.

Q.

	a	b	B
→ q_0	(q_1, a, R)	(q_0, b, R)	(q_f, B, R)
q_1	(q_2, a, R)	(q_1, b, R)	H
q_2	(q_0, a, R)	(q_2, b, R)	H
q_f	HALT		

a's divisible by 3.

Q.

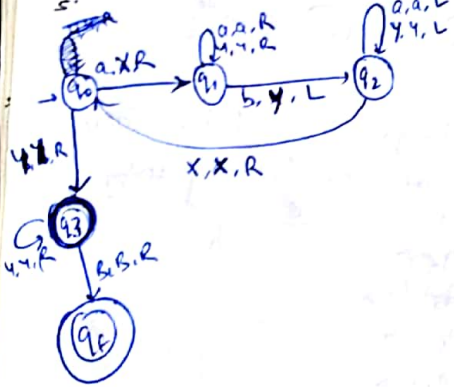
	a	b	B
→ q_0	(q_1, a, R)	(q_0, b, R)	
q_1	(q_1, a, R)	(q_0, b, R)	(q_f, B, R)
q_f	HALT		

Ending with "a".

Q. Construct the T.M. for $L = \{a^n b^n \mid n \geq 1\}$

a a X b b b B

a a X Y b a

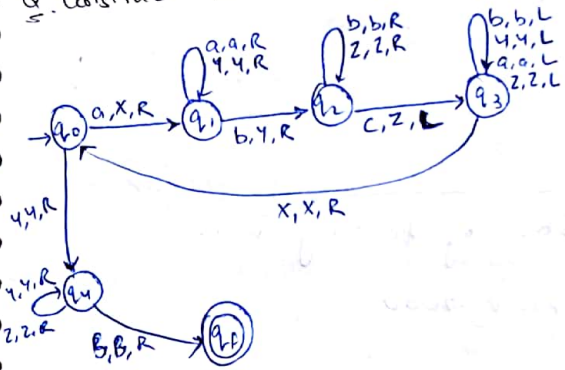


Q. Identify halting states for the following strings corresponding to the T.M. previous one:

(a) abab q_3
(b) aaaab q_1
(c) aabbbb q_3
(d) aabbaa q_3

Q. Construct the T.M. for $L = \{a^n b^n c^n \mid n \geq 1\}$

x x x y y y z z z
a a a b b b c c c B B



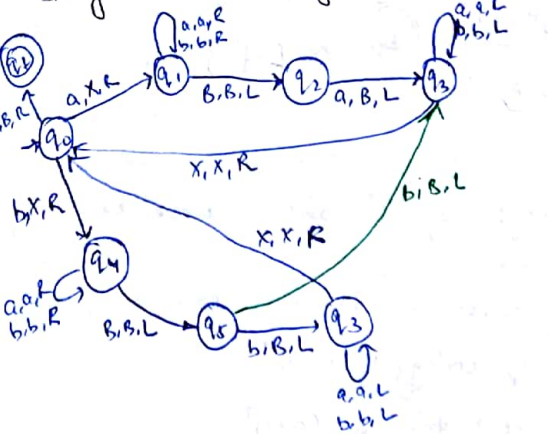
Q. Identify the halting states for the given strings:

(a) abcaabc q_4
(b) aabbbcc q_4
(c) aaabcc q_1

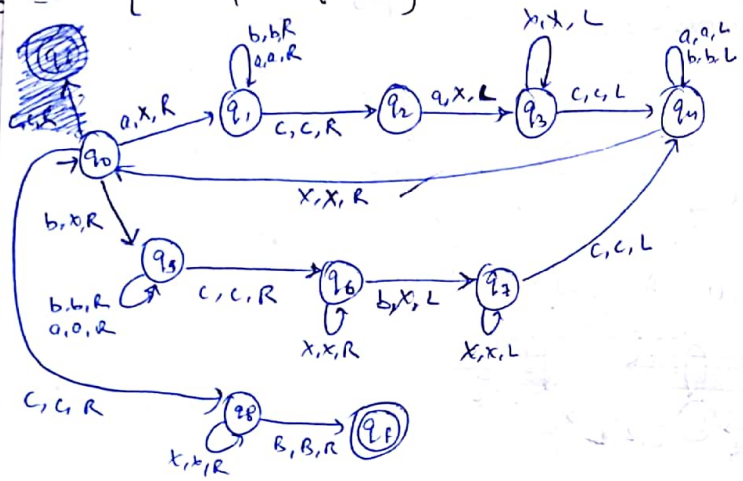
Q. Construct T.M. that takes all strings of a's & b's where each string is even length palindrome. $\{ww^R \mid w \in \{a,b\}^*\}$

x x x B B B
a b a b b b a b a B

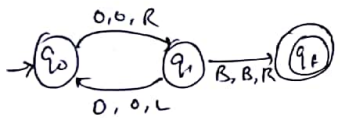
a b a b a b a b a B



Q. $L=\{WcW \mid W \in \{a,b\}^*\}$. Construct T.M.



Q. Identify language accepted by following T.M.



odd no. of zeros

(a) $L=\{0^{2n+1} \mid n \geq 0\}$

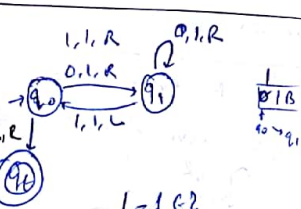
(b) $L=\{0^{2n} \mid n \geq 1\}$

(c) $L=\{0\}$

(d) $L=\{0^n \mid n \geq 1\}$

Q. Consider the following T.M.

	0	1	B
q_0	$(q_1, 1, R)$	$(q_1, 1, R)$	(q_f, B, R)
q_1	$(q_1, 1, R)$	$(q_0, 1, L)$	(q_0, B, L)
q_f	HALT		



$L=\{\epsilon\}$

(a) Turing m/c halts for all strings of 0's and 1's where each string ending with 1.

(b) T.M. halts for all strings of zeros and 1's where each string ending with 0.

(c) T.M. does not halt on any i/p string of $(0+1)^*$

(d) T.M. does not halt for any input string of $(0+1)^+$

Recursive Language: A language "L" is said to be recursive, if there exists a Turing Machine for that language that halts always on all inputs.

→ If a language "L" is recursive for valid strings of the language, or strings belonging to the languag, T.M. halts in the final state.

→ If the string doesn't belongs to the language, then the T.M. halts in non-final state.

→ If the language is recursive, then, it is decidable.

→ Recursive language is also called as Turing Decidable Language.

Recursive Enumerable Language: A language "L" is said to be R.E.L., if there exists a T.M. for that language that halts on some input strings and may not halt on some input strings.

→ If a language is REL, for all strings of that language, T.M. halts in final state.

→ If the strings ~~does~~ don't belong to that language for which, T.M. may halt in non-final state or may enter into infinite loop.

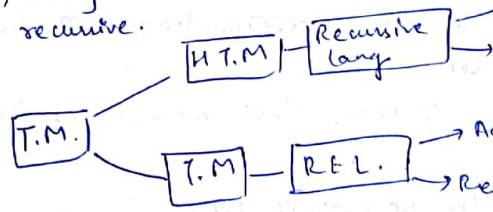
→ If a language is R.E.L., then it is undecidable. [partially decidable]

→ REL is also called as Turing Recognisable Language.

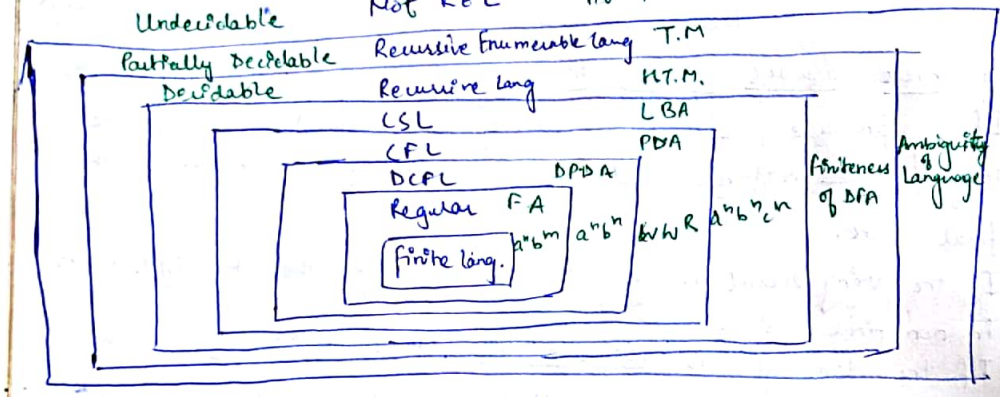
→ By default, T.M. is may or may not halting machine.

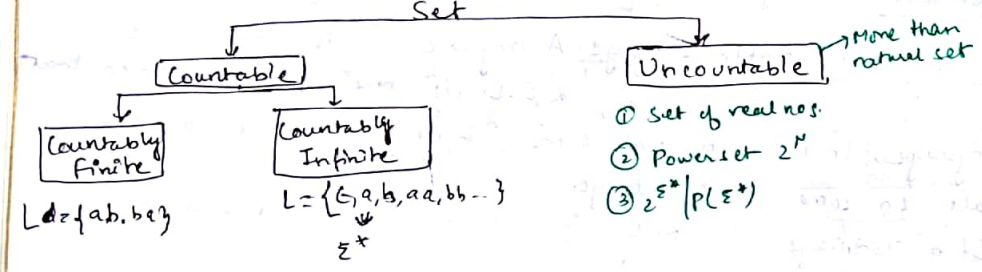
→ By default, T.M. recognisable languages are ~~recursive~~ R.E.L.

→ Every Recursive language is REL., but every R.E.L need not be recursive.



① 2^{Σ^*} ② Set of all real nos.





NOTE

→ Countable set: A set is said is said to be countable if there exists one to one correspondence with natural set to the given set.

→ The following sets are countable:
① Natural set
② Complete language
③ Any finite set

→ Uncountable set: A set is said to be uncountable if there is no one to one correspondence with the natural set to the given set.

→ If the size of the set is larger than natural set, then it is uncountable.

→ The following are some of the uncountable set:
① Set of real numbers
② Powerset of complete language $[2^{\Sigma^*}]$
③ Set of all possible languages over the given i/p alphabet (a, b).

④ Power set of natural set.

→ If any set is uncountable, then no T.M. can be constructed for it.
→ All uncountable sets are not R.E.L.
→ All decidable languages are countable.
→ Set of all languages accepted by T.M. are countable.
→ Set of all languages not accepted by T.M. are uncountable.
→ Number of undecidable problems are uncountable.
→ Set of all problems solvable by computer are countable.
→ Set of all problems for which no algorithm exists are uncountable.
→ Total number of regular languages are countable.
→ Total number of languages accepted by PDA is countable.
→ No. of not REL are uncountable.

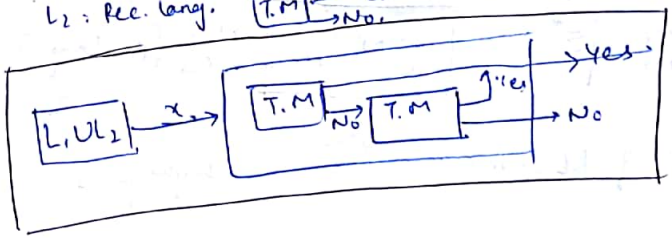
Q. Which of the following is true?
(a) Σ^* and 2^{Σ^*} are countable. False
(b) Σ^* and 2^{Σ^*} are uncountable. False
(c) Σ^* is countable, 2^{Σ^*} is uncountable. True
(d) None of these.

Closure Properties of Recursive and REL:

① Union: Union of two Recursive languages is always recursive, Hence, recursive languages are closed under union operation.

→ For $L_1 \cup L_2$, the TM is constructed with the help of L_1 T.M. and L_2 T.M. as follows:

L_1 = Rec. lang. [T.M → Yes, No]
L_2 = Rec. lang. [T.M → Yes, No]



R.E.L.: Union of two RELs is always REL, because, we can construct the TM, as follows:

② **Intersection Operation:** Intersection of 2 Recursive language is always recursive because we can construct TM. for $L_1 \cap L_2$, in the following ways

R.E.L.: Intersection of R.E.L & R.E.L is always R.E.L because we can construct TM. for $L_1 \cap L_2$ as follows:

③ **Complement Operation:** Complement of recursive language is always recursive. Hence, recursive languages are closed under complement operation.

R.E.L: The complement of REL but not Recursive language is not R.E.L.

→ for a language and its complement, the following are the possibilities.

Operation	Regular	DCFL	CFL	CSL	Recursive long	R.E.
① Union	✓	X	✓	✓	✓	✓
② Concatenation	✓	X	✓	✓	✓	✓
③ Intersection	✓	X	X	✓	✓	✓
④ Complement	✓	✓	X	✓	✓	X
⑤ Intersection with regular	✓	✓	✓	✓	✓	X
⑥ Difference $L_1 - L_2 [L_1 \cap L_2^c]$	✓	X	X	✓	✓	X
⑦ Difference with regular	✓	✓	✓	✓	✓	✓
⑧ Reverse	✓	X	✓	✓	✓	✓
⑨ Kleene closure	✓	X	✓	✓	✓	✓
⑩ Positive closure	✓	X	✓	✓	✓	X
⑪ Substitution	✓	X	✓	✓	✓	X
⑫ Homomorphism	✓	✓	✓	✓	✓	✓
⑬ Inverse - H	✓	✓	✓	✓	✓	✓
⑭ Left Quotient with regular	✓	✓	✓	✓	✓	✓

Q. L_1 is regular, L_2 is CFL, L_3 is Recursive, L_4 is REL. Then which of the following is ~~True~~ False:

(a) $L_3^c \cap L_4$ is REL. True
(b) $L_1^* \cap L_3$ is recursive. True.
(c) $L_2^c \cup L_1$ is CFL. False
(d) $L_1 \cup L_3^c$ is R.E.L. True

Q. Let L_1 is recursive language, L_2 and L_3 are RE.L but not recursive, then which of the following is true.

(a) $L_2 - L_3$ is RFL. False
(b) $L_1 - L_3$ is REL. False
(c) $L_2 - L_1$ is RFL. True
(d) $L_2 \cup L_3$ is REL. True

Recursive language (Decidable):

(1) $L = \{ \langle D, w \rangle \mid D$ is a DFA that accepts $w \}$
(2) $L = \{ \langle D_1, D_2 \rangle \mid D_1, D_2$ are DFA with $L(D_1) = L(D_2) \}$
(3) $L = \{ \langle D \rangle \mid D$ is a DFA with $L(D) = \phi \}$
(4) $L = \{ \langle G, w \rangle \mid G$ is a CFG that generates string $w \}$
(5) $L = \{ \langle G \rangle \mid G$ is a CFG with $L(G) = \phi \}$
(6) $L = \{ \langle D \rangle \mid D$ is a DFA with $L(D) =$ finite $\}$
(7) $L = \{ \langle M, Q \rangle \mid M$ is a T.M. that visits state Q on some i/p within 100 steps $\}$
(8) whether a given T.M. has atleast 420 states?
(9) whether a given T.M. takes more than 460 steps on some i/p.

RFL (Partially Decidable):

(1) $L = \{ \langle M, w \rangle \mid M$ is a T.M. that accepts string $w \}$
(2) $L = \{ \langle G \rangle \mid G$ is a CFG and G is ambiguous $\}$
(3) whether the given T.M. accepts ϵ?
(4) whether the given T.M. accepts string x?
(5) $L = \{ \langle M, Q \rangle \mid M$ is a T.M. that visits state Q on some input.

Not RFL. (Undecidable):

(1) Complement of RFL but not Recursive.
(2) $L = \{ \langle M, w \rangle \mid M$ is a T.M. that does not accept string $w \}$
(3) $L = \{ \langle M \rangle \mid M$ is a T.M. with $L(M) = \phi \}$
(4) $L = \{ \langle M_1, M_2 \rangle \mid M_1$ and M_2 are T.M. with $L(M_1) = L(M_2) \}$
(5) $L = \{ \langle G \rangle \mid G$ is a CFG and G is unambiguous $\}$

Q. Consider the following 3 languages:

$L_1 = \{ \langle M \rangle \mid M$ is a T.M. that takes 2016 steps on some input $\}$

$L_2 = \{ \langle M \rangle \mid M$ is a T.M. that takes atleast 2016 steps on some i/p $\}$

$L_3 = \{ \langle M \rangle \mid M$ is a T.M. that accepts $\epsilon \}$

Which of the following is true?

(a) L_1 is recursive, L_2 & L_3 are not recursive False
(b) L_1, L_2, L_3 are recursive. False
(c) L_1, L_2 are recursive & L_3 is not recursive. True
(d) All are not recursive. False

Q. Given a T.M. "M", over the input "x". Does the computation of "M" on x, visit state Q. is.

(a) Decidable
(b) Non-REL
(c) REL
(d) Regular

UNDECIDABILITY:

A ∝ B
UD → UD
D ← D

→ Language or problem is said to be decidable if there exists a H.T.M. for it.

→ A language or problem is said to be undecidable if no H.T.M. exists for it.

★ **Reduction:** A problem "A" is reducible to "B". We can conclude the problem "B" (new problem) with the help of problem "A" [known problem].

→ If a problem "A" is reducible to "B" happens, the following are the possibilities:

1. If "B" is decidable then "A" is also decidable.
2. If "A" is undecidable then "B" is also undecidable.
3. If "B" is recursive language, then 'A' is also recursive language.
4. If "A" is not recursive, then, "B" is not recursive.
5. If "B" is recursive enumerable, then 'A' is recursive enumerable.
6. If "A" is not REL, then 'B' is not REL.
7. If "A" is not solved, then 'B' is not solved.
8. If "B" is solved, then "A" is solved.

→ To prove a problem "X" is undecidable, take some known undecidable problem and reduce into "X".

POST CORRESPONDENCE PROBLEM: (PCP)

Q. Identify solutions for following PCPs;

① A = {1, 10111, 10} (①②③)
B = {111, 10, 0}

Solution is 2113

10|111|1|1|10
10|111|1|1|0

② A = {bbab, ab, baa, b} (①②③④)
B = {a, abbb, aa, bbb}

~~4 1 4 3~~
~~b bbab b baa~~
~~bbb a bbb aa~~

2	1	4	3
ab	bbab	b	baa
abbb	a	bbb	aa

Solution is 2143 ;

③ A = {aaa, bb, aaa} (①②③)
B = {aba, bba, b}

No solution.

④ A = {ba, ab, a, baa, b}
(1, 2, 3, 4, 5)
B = {bab, baa, ba, a, aba}
(1, 2, 3, 4, 5)

1	4	4	2	1	5	2	3	4	4
ba	baa	baa	ab	ba	b	ab	a	baa	baa
bab	a	a	baa	bab	aba	baa	ba	a	a

1 4 5 2 3 2 3 4 4 3 4

NOTE
→ PCP is an undecidable problem because halting T.M. cannot be constructed for PCP.

Modified PCP:

→ In MPCP, solution is required to start with first string of each list.

→ MPCP is also undecidable problem.

→ MPCP is reducible to PCP.

→ Both PCP and MPCP are decidable, if the list contains one element or one string.

Problems regarding formal languages:

① **Intersection Empty Problem:** means checking whether intersection of the given 2 languages is empty or not.

→ This problem is decidable if intersection operation is closed and emptiness problem is decidable.

② **Subset Problem:** means checking whether language L1 is subset of language L2 or not.

→ This problem is decidable if intersection operation is closed and equivalence problem is decidable.

Reg → D | DCFL → UD | CFL → UD | REL → U.D. | $L_1 \cap L_2 = L_1$ ($L_1 \subseteq L_2$)

③ **Co-finiteness Problem:** means checking whether complement of a language is finite or not.

→ This problem is decidable if complement operation is closed and finiteness problem is decidable. | Reg → D | DCFL → D | CFL → UD | REL → UD

④ Totality Problem: means checking whether given language is total language or not or complete language or not.

If $L = \Sigma^*$?

$\Sigma^* - L$ → Emptiness

→ This problem is decidable if complement operation is closed and the emptiness problem is decidable.

⑤ Intersection finiteness Problem: means checking whether intersection of two languages is finite or not.

→ This problem is decidable if intersection operation is closed and finiteness problem is decidable.

The following problems are decidable under regular languages:

① Emptiness Problem
② Membership Problem
③ Finiteness Problem
④ Equivalence Problem
⑤ Totality Problem
⑥ Intersection Empty Problem
⑦ Subset Problem
⑧ Co-finiteness Problem
⑨ Intersection finiteness Problem

Decidable and Undecidable Problems about DCFLs:

① Membership Problem: Decidable (CYK Algorithm)
② Emptiness Problem: Decidable
③ Finiteness Problem: CNF Graph (Decidable)
④ Equivalence Problem: Undecidable
⑤ Intersection Empty Problem: Undecidable
⑥ Subset Problem: Undecidable
⑦ Totality Problem: Decidable
⑧ Co-finiteness Problem: Decidable
⑨ Intersection finiteness Problem: Undecidable.

Decidable and Undecidable Problems about CFLs:

① Membership: Decidable
② Emptiness: Decidable
③ finiteness: Decidable
④ Equivalence: Undecidable
⑤ Intersection Empty problem: Undecidable
⑥ Subset Problem: Undecidable
⑦ Totality: Undecidable
⑧ Co-finiteness: Undecidable
⑨ Intersection finiteness: Undecidable

Decidable and undecidable problems about RELs or T.M.:

① Membership: Undecidable
② Emptiness: Undecidable
③ Finiteness: Undecidable
④ Equivalence: Undecidable
⑤ Subset: Undecidable
⑥ Totality: Undecidable
⑦ Intersection Empty: Undecidable
⑧ Co-finiteness: Undecidable
⑨ Intersection finiteness: Undecidable

Halting Problem of T.M.:

→ By reading string "x", whether the T.M. halts or not.
→ Halting Problem is undecidable problem.
→ Halting Problem is decidable problem for Recursive language.

Problem	Regular	DCFL	CFL	C.S.L.	Recursive	R.E.L.
① is $w \in L$?	D	D	D	D	D	U.D.
② is $L = \phi$?	D	D	D	U.D.	U.D.	U.D.
③ is L finite?	D	D	D	UD	UD	UD
④ is $L_1 = L_2$?	D	U.D.	U.D.	U.D	U.D.	U.D.
⑤ is $L_1 \subseteq L_2$?	D	U.D.	U.D.	U.D	U.D.	U.D.
⑥ Is $L = \Sigma^*$?	D	D	U.D.	U.D.	U.D	U.D
⑦ is $L_1 \cap L_2 = \phi$?	D	U.D.	U.D	U.D.	U.D.	U.D.
⑧ is $\Sigma^* - L$ = finite?	D	D	U.D.	U.D.	U.D.	U.D.
⑨ is $L_1 \cap L_2$ = finite?	D	U.D.	U.D.	U.D.	U.D.	U.D.
⑩ Regularity Problem. Checking whether the given lang is regular or not.	D	D	U.D.	U.D.	U.D.	U.D.
⑪ Ambiguity Problem.	D	U.D.	U.D.	U.D.	U.D.	U.D.

The following problems about programming language are undecidable:

(1) Whether a given ~~problem~~ program, ever produces an output.
(2) Whether a given program can loop forever on some input or not.
(3) Whether the given 2 programs produce same output for the same input or not.

Q. 2017 Which of the following problem is decidable?
(a) G is a CFG, is $L(G) = \Sigma^*$? U.D. (b) is $L(M_1) = L(M_2)$, where M_1 & M_2 are 2 T.M. U.D.
(c) Is G is a CFG, is $L(G) = \phi$ Decidable. (d) is $w \in L(M)$? M is a T.M. ? U.D.

Q. Which of the following problem is decidable:
(a) Does a given ~~problem~~ program loops forever on some input. U.D.
(b) G is a CFG, is $L(G) = \Sigma^*$. U.D.
(c) G_1 & G_2 are 2 CFG, is $L(G_1) = L(G_2)$. U.D.
(d) If L is a Recursive lang:, then its complement is also recursive. Decidable.

Q. Let "x" is a recursive language and "y" is REL but not recursive. w, z are 2 languages, such that $y^c \propto w$ (y^c reduces to w) and $z \propto x^c$, then which of the following is true?

No	→ No
Yes	← Yes

Reduction.

(a) w can be REL and z is recursive.
(b) w is recursive and z is REL.
(c) w is not REL and z is recursive.
(d) w is not REL and z is not REL.

Q. $A \propto B$ (A reducible to B), then which of the following is false?
(a) If B is recursive, then A is recursive. True
(b) If A is undecidable, then B is undecidable. True
(c) If B is not recursive enumerable, then A is not REL. False
(d) None of these.

Q. Which of the following problem is undecidable?
(a) Deciding if a given string is generated by given CFG or not. D.
(b) Deciding the lang. generated by given CFG is finite or not. D.
(c) Deciding whether the language accepted by T.M. is empty / not. U.D.
(d) None of these.

Q. Let $L = \{ \langle M \rangle \mid M$ is a T.M. that accepts a string of length 2014$\}$. Then L is:
(a) Decidable (b) Undecidable and not REL
(c) Undecidable and REL (d) None of these.

DECIDABILITY:

* **P-class Problem:** A problem is said to be P-class if there exists, D.T.M. for that problem or Deterministic algorithm for that problem.
* **NP-class Problem:** A problem is said to be NP-class if there exists, N.T.M. for that problem or there exists Non-Deterministic algorithm for that problem.

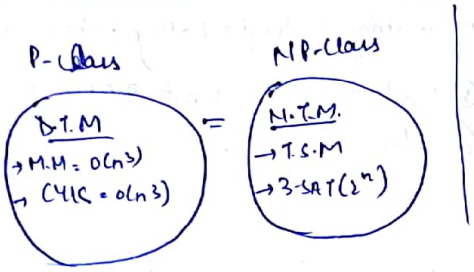
→ Every P-class problem is NP-class problem, but every NP-class problem need not be P-class problem.
→ P = N.P. happens, or not? is an open question.
→ For all NP class problems, if D.T.M. exists in polynomial time, then P = N.P. happens.
But, still today, no one has proved this. Hence, P = NP is not known.

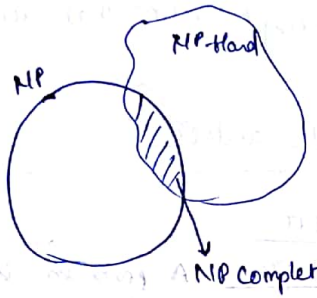
* **N.P. Complete Problem:** The problem which completely belong to NP class but don't belong to P-class are known as NP-Complete Problems.

→ Hence, $P \cap N.P.\text{Complete} = \phi$.

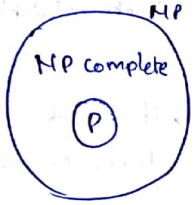
* **NP Hard Problem:** The problems which are as hard as NP class problems, known as NP Hard Problems.

→ NP Hard Problem's time complexity may be greater than NP Complete.
→ If any NP Hard problem belongs to NP class, then it is NP Complete problem.
→ All NP Complete Problems are NP Hard Problems, but all NP Hard Problems are not NP Complete Problems.

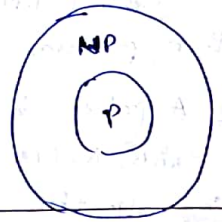




P = NP ?



$P \cap NPComplete = \phi$



$P \cap NP = P$

Modifications of T.M.:

→ After performing modification to the T.M., the expressive power of the T.M. remains the same [computing speed may increase].

→ The following are some of the modified versions of T.M.:

① Two-way infinite tape T.M: I/p tape is infinite in both directions.

② Multi-tape T.M: It is a T.M. in which, multiple tapes exist, where each tape is infinite in both directions.

→ for every multi tape T.M., we can construct an equivalent single tape T.M.

③ Multi Track T.M: It is a T.M. which contains one i/p tape where tape is having multiple tracks.

④ Non-Deterministic T.M.: The T.M. in which, from the given state and given i/p tape symbol, finite no. of alternatives exist for next move.

→ for every N.T.M., we can construct an equivalent D.T.M.

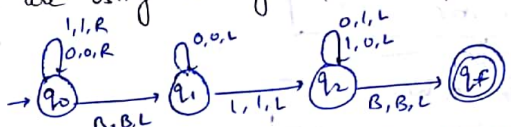
⑤ Multi Stack T.M.: finite automata having two or more stacks i/a multi stack T.M.

⑥ Universal T.M: Universal T.M. can take other T.M.s as i/p and simulates behaviour of that T.M.

Turing Machine as Output Generator:

Q. Construct the T.M. that produces 2's complement of given binary no. as output. [Assume we are using 2-way infinite tape T.M.]

B	0	1	0	R	0	0	0



* The functions which are computable by computer for which we can construct T.M.

→ The functions performed by T.M. are classified into: ① Total functions ② Partial function

→ A function is said to be total func^n for which there exists a T.M. that halts for all func^n values.

→ Total functions are similar to recursive language.

→ Partial functions: a func^n is said to be partial func^n for which there exists a T.M., that halts for some function values and may not halt for some function values.

→ Partial functions are similar to recursive enumerable language.

Turing Machines as Language Generator / Enumerator:

→ Enumerator is a predefined T.M. that doesn't take any any input but produces the output on the o/p tape.

→ The output produced by enumerator are kept in output tape, where each string is separated by a "#" symbol.

→ The languages produced by enumerator is exactly same as language accepted by T.M.

→ Hence, languages produced by T.M. are recursive languages and recursive enumerable languages.

→ If the strings produced by T.M. are arranged in lexicographic order [increasing length order] or alphabetical order, then that language is known as recursive language.

→ If the strings produced by enumerator are not in any proper order, then that language is REL.