

Syllabus

UNIT-I

Introduction: Finite State automata, Non-deterministic finite State automata, FA with ϵ transitions, Regular expressions Applications of FA, properties of regular sets, pumping lemma, Closure properties, Myhill-Nerode Theorem, Minimization of FA.

UNIT-II

Context free Grammars & languages: Derivations, parse-tree Ambiguity in Grammars & languages. pushdown Automata - Definitions, The languages of PDA, Equivalence of PDAs & CFGs, Deterministic pushdown Automata.

UNIT-III

Properties of CFLs: Normal forms for CFGs., pumping Lemma, Closure properties, Deterministic Context Free Languages, Decision properties.

UNIT-IV

Turing Machines: Introduction, Computational languages & Functions, Techniques for Construction of Turing machine Modification of TM, TM as enumerator, Restricted TM.

UNIT-V

Undecidability: Recursive & Recursively enumerable languages,

UTM & undecidable problem, Rice Theorem, Post's Correspondence problem, Chomsky's Hierarchy - Regular grammars, Unrestricted grammars, CSL, Relationship b/w classes of languages.

Modified post correspondence

01 Sept 2020

UNIT-01

INTRODUCTION

Automata Theory = Automata Theory is a branch of Computer Science that deals with designing abstract self-propelled computing devices that follow a predetermined sequence of operations automatically.

→ The term "Automata" (or) is derived from the Greek word "αὐτόματα" which means "Self acting".

An automaton (Automata is plural) is an abstract self-propelled computing device which follows a predetermined sequence of operations automatically.

→ An automaton with a finite number of states is called a finite automaton (FA) or finite state machine (FSM).

Formal definition of a finite Automaton:-

→ An automaton can be represented by a 5-tuple

(Q, Σ, S, q_0, F) , where:-

- i) Q is a finite set of states.
- ii) Σ is a finite set of symbols, called the alphabet of the automaton
- iii) S is the transition function, where $S: Q \times \Sigma \rightarrow Q$
- iv) q_0 is the initial state from where any i/p is processed ($q_0 \in Q$)
- v) F is a set of final state / states of Q ($F \subseteq Q$)

Related Terminologies:-

i) Alphabet: An alphabet is any finite set of symbols
Ex: $\Sigma = \{a, b, c, d\}$ is an alphabet set where 'a', 'b', 'c', and 'd' are symbols.

ii) String: A string is a finite sequence of symbols taken from Σ

Ex: 'cabcad' is a valid string.

$S = \{a, b, c, d, e, x, y, z\}$ is not a valid string bcoz x, y, z is not available in Σ .

Length of a string: It is the no. of symbols present in a string. (Denoted by $|S|$).

Ex: If $S = 'cabcad'$, $|S| = 6$

If $|S| = 0$, it is called an empty string and it is denoted by " λ " or " ϵ "

Kleene Star:-

→ The Kleene Star, Σ^* is a unary operator on a set of symbols or strings, Σ , that gives the infinite set of all possible strings of all possible lengths over Σ including λ .

→ Representation - $\Sigma^* = \Sigma_0 \cup \Sigma_1 \cup \Sigma_2 \cup \dots$ where Σ_p is the set of all possible strings of length p .

→ Example - If $\Sigma = \{a, b\}$, $\Sigma^* = \{\lambda, aa, ab, ba, bb, \dots\}$

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

$\Sigma_0 = \{\lambda, b\} \Rightarrow 1$

$\Sigma_1 = \{aa, bb, ab, ba\} \Rightarrow 2$

$\Sigma_2 = \{aaaa, aab, aba, abb, bbb, \dots\} \Rightarrow 3$

* Kleene Closure / plus :-

→ The set Σ^+ is the infinit set of all possible strings of all possible lengths over Σ excluding λ .

→ Representation - $\Sigma^+ = \Sigma_1 \cup \Sigma_2 \cup \Sigma_3 \dots$

→ $\Sigma^+ = \Sigma^* - \{\lambda\}$ (i.e. Σ^* will start from λ , but in Kleene Closure Σ^+ will start from Σ_1)

→ Example :- If $\Sigma = \{a, b\}$, $\Sigma^+ = \{a, b, aa, ab, ba, bb, \dots\}$

* Prefix & Suffix of a String :-

Prefix of a String :- prefix string means number of leading symbols of that string

Ex: String is "abc"

Prefix of a string is: ϵ, a, ab, abc .

Suffix of a String :- Number of its trailing symbols of that string

Ex: String is "abc"

Suffix of a string is: ϵ, c, bc, abc .

* Language :-

→ A language is a subset of Σ^* for some alphabet Σ . It can be finite or infinite.

Example - If the language takes all possible strings of length 2 over $\Sigma = \{a, b\}$ then $L = \{ab, aa, ba, bb\}$.

* Finite Automaton :-

→ Finite Automaton can be classified into two types:

i) Deterministic Finite Automaton (DFA)

ii) Non-deterministic finite Automaton (NFA/NDA)

i) Deterministic Finite Automaton (DFA) :-

→ In DFA, for each i/p symbol, one can determine the state to which the machine will move. Hence, it is called Deterministic Automaton.

→ As it has a finite number of states, then it is called Deterministic Finite Machine (DFM) or DFA.

→ A DFA can be represented by 5-tuple $(Q, \Sigma, \delta, q_0, f)$. Graphical Representation of a DFA :-

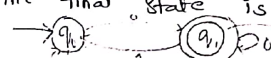
→ A DFA is represented by digraphs called state diagram or transition diagram.

→ The vertices represent the states.

→ The arcs labeled with an input alphabet show the transitions.

→ The initial state is denoted by an empty single incoming arc. $\rightarrow q_0$

→ The final state is indicated by double circles. $\rightarrow q_f$



Example:-

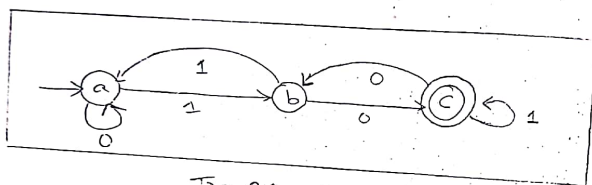
→ Let a deterministic finite automaton be

$\Sigma = \{a, b, c\}$; $\mathcal{I} = \{0, 1\}$; $q_0 = \{a\}$; $F = \{c\}$.

→ Transition function δ as shown by the following table:

present State	Next state for $\mathcal{I}/P$ 0	Next state for $\mathcal{I}/P$ 1
a	a	b
b	c	a
c	b	c

Transition Table

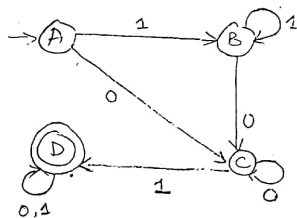


Transition Graph.

Draw the Transition Graph for the given Transition Table:

PS	$\mathcal{I}/P$ 0	$\mathcal{I}/P$ 1
→ A	C	B
B	C	B
C	C	D
* D	D	D

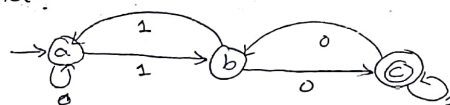
Transition Table/State table.



Transition graph / State graph

⊗ Acceptance of the String by Finite automata:

1. Check the String 11010 is accepted by FA or not?



Checking the String 11010 Initial State = a

Final State = c

⇒ { a, 11010 }

⇒ { b, 11010 }

⇒ { a, 11010 }

⇒ { a, 11010 }

⇒ { b, 11010 }

⇒ { c }

The String 11010 is Valid for the above State graph.

2. Check the String 1101001 is accepted by the FA or NOT?

Checking the String 1101001 Initial State = a

Final State = c

⇒ { a, 1101001 }

⇒ { b, 1101001 }

⇒ { a, 1101001 }

⇒ { a, 1101001 }

⇒ { b, 1101001 }

⇒ { c, 1101001 }

⇒ { b, 1101001 }

⇒ { a }

∴ The String 1101001 is not Valid for the above State graph.

8 Sept 2020

ii) Non-Deterministic Finite Automaton (NFA / NDFA):

→ In NDFA, for a particular i/p symbol, the machine can move to any combination of the states in the machine.

→ In other words, the exact state to which the machine moves cannot be determined.

→ Hence, it is called Non-deterministic Automaton.

→ As it has finite no. of states, the machine is called Non-deterministic Finite Machine (or) Non-deterministic Finite Automaton.

→ An NDFA can be represented by 5-tuple $(Q, \Sigma, \delta, q_0, F)$ where-

δ is the transition function where $\delta: Q \times \Sigma \rightarrow 2^Q$ (Here the power set of Q (2^Q) has been taken bcoz in case of NDFA, from a state, transition can occur to any combination of Q states).

Q, Σ, q_0, F are same as DFA.

Graphical Representation of a NDFA:-

Some points of DFA. is for NDFA.

Example 1:-

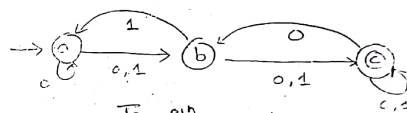
→ Let a NDFA be :-

$Q = \{a, b, c\}$; $\Sigma = \{0, 1\}$; $q_0 = \{a\}$; $F = \{c\}$

→ The transition function δ as shown below:-

present state	Next state for i/p 0	Next state for i/p 1
a	a, b	b
b	c	a, c
c	b, c	c

Transition Table.



Transition graph.

→ We can also represent the transition table as transition function

$\{a, 0\} \rightarrow \{a, b\}$

$\{a, 1\} \rightarrow \{b\}$

$\{b, 0\} \rightarrow \{c\}$

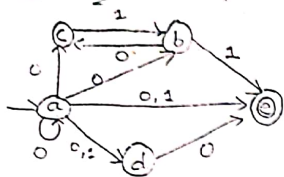
$\{b, 1\} \rightarrow \{a, c\}$

$\{c, 0\} \rightarrow \{b, c\}$

$\{c, 1\} \rightarrow \{c\}$

Formula
 $\therefore \delta(\text{present state, i/p symbol}) = \text{Next state}$

Example 2.0:



Transition Graph.

$\{a, 0\} \rightarrow \{a, b, c, d, e\}$

$\{a, 1\} \rightarrow \{d, e\}$

$\{b, 0\} \rightarrow \{c\}$

$\{b, 1\} \rightarrow \{e\}$

$\{c, 0\} \rightarrow \emptyset \rightarrow \text{empty}$ Transition graph

$\{c, 1\} \rightarrow \{b\}$

$\{d, 0\} \rightarrow \{c\}$

$\{d, 1\} \rightarrow \emptyset$

$\{e, 0\} \rightarrow \emptyset$

$\{e, 1\} \rightarrow \emptyset$

q	$\delta(q, 0)$ or 0	$\delta(q, 1)$ or 1
a	$\{a, b, c, d, e\}$	$\{d, e\}$
b	$\{c\}$	$\{e\}$
c	$\emptyset$	$\{b\}$
d	$\{c\}$	$\emptyset$
e	$\emptyset$	$\emptyset$

Transition Table

Sep. 09. 2020

Draw the Transⁿ Graph for the given NFA.

δ	0	1
$\rightarrow q_0$	$\{q_0\}$	$\{q_0, q_1\}$
q_1	$\{q_2\}$	$\{q_2\}$
$* q_2$	$\emptyset$	$\emptyset$

Transitⁿ functⁿ := (Present State, I/p Symbol = Next State)

$\{q_0, 0\} \rightarrow \{q_0\}$

$\{q_0, 1\} \rightarrow \{q_0, q_1\}$

$\{q_1, 0\} \rightarrow \{q_2\}$

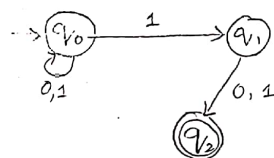
$\{q_1, 1\} \rightarrow \{q_2\}$

$\{q_2, 0\} \rightarrow \emptyset$

$\{q_2, 1\} \rightarrow \emptyset$

$\therefore \{Q, \Sigma, \delta, q_0, F\} \Rightarrow Q = \{q_0, q_1, q_2\}; \Sigma = \{0, 1\};$
 $q_0 = \{q_0\}; F = \{q_2\}.$

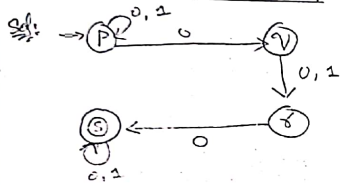
Transitⁿ graph:



Acceptance of the String by Finite Automata :-

1) Check the String 11010 is accepted by NFA or Not?

	0	1
$\rightarrow P$	$\{P, q\}$	$\{P\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	$\emptyset$
$* S$	$\{s\}$	$\{s\}$



Checking the String 11010

$\Rightarrow \{P, \textcircled{1}1010\}$

$\Rightarrow \{P, \textcircled{0}10\}$

$\Rightarrow \{P, \textcircled{0}10\}$

$\Rightarrow \{q, \textcircled{0}10\}$

$\Rightarrow \{r, \textcircled{0}1\}$

$\Rightarrow \{s\}$

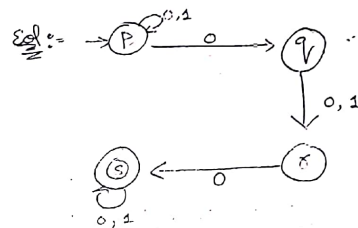
initial = P

final = S

$\therefore$ The string 11010 is Valid (✓) - Accepted by the finite Automata.

2) Check the String 110111 is accepted by the NFA or Not?

Same table



Checking the String 110111

$\Rightarrow \{P, \textcircled{1}10111\} \Rightarrow \{P\}$

$\Rightarrow \{P, \textcircled{0}111\} \Rightarrow \{P\}$

$\Rightarrow \{P, \textcircled{0}111\} \Rightarrow \{P, q\}$

$\Rightarrow \{q, \textcircled{0}1\} \Rightarrow \{r\}$

$\Rightarrow \{r, \textcircled{0}1\} \Rightarrow \{\emptyset\}$

$\Rightarrow \{P, 0111\}$

Result

DFA	NFA / NDFA
i) The transition from a State is to a single next state for each I/P Symbol. Hence it is called DFA.	i) The transition from a State is to a multiple next state for each I/P Symbol. Hence it is called NFA.
ii) Empty string transitions are not seen in DFA.	ii) NDFA permits empty string transitions.
iii) Backtracking is allowed in DFA.	iii) In NDFA, backtracking is not always possible.
iv) Requires more space.	iv) Requires less space.
v) A string is accepted by DFA, if it transits to a final state.	v) A string is accepted by NFA, if at least one of all possible transitions ends in a final state.
vi) DFA will reject the string, if it ends at other than accepting state.	vi) If all of the branches of NFA dies or rejects the string, we can say that NFA rejects the string.
vii) DFA can be considered as one machine.	vii) NFA can be considered as multiple little machines computing at the same time. Eg: Lottery (or) Voting Machine.
viii) DFA is very difficult to design.	viii) NFA is easier to design.
ix) The transition function δ is: $\delta: Q \times \Sigma \rightarrow Q$.	ix) The transition function δ is: $\delta: Q \times \Sigma \rightarrow 2^Q$.
x) All DFA is NFA.	x) All NFA is not DFA.

11. Sept. 2020

[1st Method := partition (or) Equivalence]

* Conversion of NDFA to DFA :-

→ Let $X = (Q_x, \Sigma, \delta_x, q_0, f_x)$ be an NDFA which accepts the language $L(X)$.

→ We have to design an equivalent DFA $Y = (Q_y, \Sigma, \delta_y, q_0, f_y)$ such that $L(Y) = L(X)$.

→ Algorithm: —

→ S/P := An NDFA

→ O/P := An equivalent DFA.

Steps to Convert NDFA to DFA :-

Step 1: Create state table from the given NDFA.

Step 2: Create a blank state table under possible input alphabets for the equivalent DFA.

Step 3: Mark the start state of the DFA by q_0 (same as the NDFA).

Step 4: Find out the combination of states $\{Q_0, Q_1, \dots, Q_n\}$ for each possible input alphabet.

Step 5: Each time we generate a new DFA state under the input alphabet columns, we have to apply Step 4 again, otherwise go to Step 6.

Step 6: The state which contain any of the final states of the NDFA are the final states of the equivalent DFA.

Example 1:-

Convert the NFA to DFA by following state table.

S	0	1
q_0	q_0	q_2
q_2	q_2	q_0, q_2

Sol Given: $X = \{q_x, \epsilon, s, q_0, f\}$

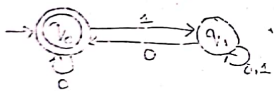
$$Q = \{q_0, q_1\}$$

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

$$q_0 = \{q_0\}$$

$$F = \{q_0\}$$

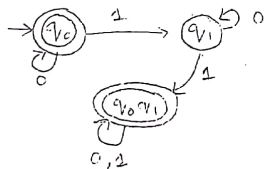
State graph/Transition graph:-



DFA State table:-

S	0	1
q_0	q_0	q_1
q_1	q_1	q_0, q_1

DFA State graph:-

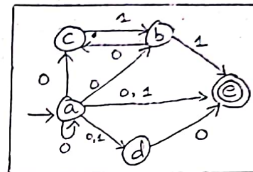


$$L(X) = L(Y)$$

NFA = DFA

11.09.2025
Example 2:-

Convert the given NFA to DFA.



State graph

q	$\delta(q, 0)$	$\delta(q, 1)$
a	$\{a, b, c, d, e\}$	$\{d, e\}$
b	$\{c\}$	$\{e\}$
c	$\emptyset$	$\{b\}$
d	$\{e\}$	$\emptyset$
e	$\emptyset$	$\emptyset$

State table.

Sol Given: $Q = \{q, \epsilon, s, q_0, f\}$

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

$$q_0 = \{a\}$$

$$F = \{e\}$$

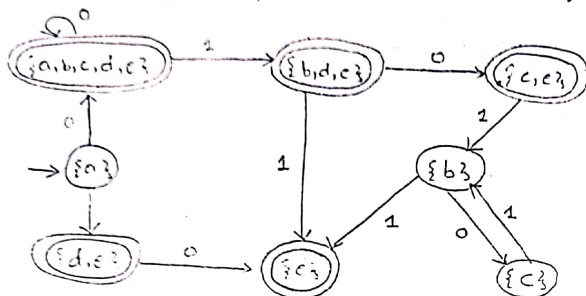
DFA State table:-

Using the above algorithm, we find its equivalent DFA.

The State table of the DFA is shown in below.

q	$\delta(q, 0)$	$\delta(q, 1)$
$\{a\}$	$\{a, b, c, d, e\}$	$\{d, e\}$
$\{a, b, c, d, e\}$	$\{a, b, c, d, e\}$	$\{b, d, e\}$
$\{d, e\}$	$\{e\}$	$\emptyset$
$\{b, d, e\}$	$\{c, e\}$	$\{e\}$
$\{e\}$	$\emptyset$	$\emptyset$
$\{c, e\}$	$\emptyset$	$\{b\}$
$\{b\}$	$\{c\}$	$\{e\}$
$\{c\}$	$\emptyset$	$\{b\}$

The State Diagram of the DFA is as follows



NFA = DFA ($\because$ Equivalent to NFA)

Example 3:-

Conversion from NFA to DFA:-

δ	0	1
$\rightarrow P$	$\{P, q\}$	$\{P\}$
q	$\{r\}$	$\{r\}$
r	$\{s\}$	ϕ
$* S$	$\{s\}$	$\{s\}$

$\{0, 1, s, q_0, r\}$

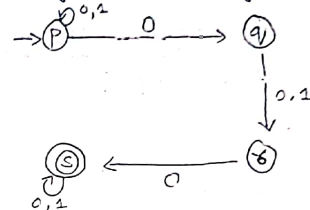
$$Q = \{P, q, r, s\}$$

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

$$q_0 = \{P\}$$

$$F = \{s\}$$

State graph / Transition graph of NFA :-

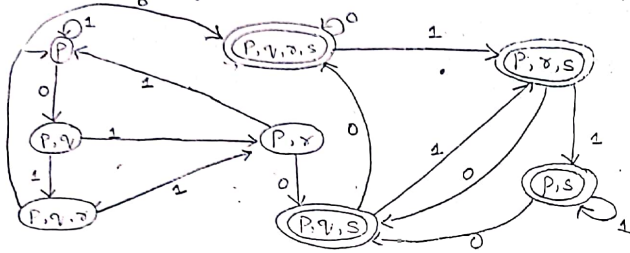


The State table of DFA is shown below.

δ	0	1
$\rightarrow P$	$\{P, q\}$	$\{P\}$
$\{P, q\}$	$\{P, q, r\}$	$\{P, r\}$
$\{P, q, r\}$	$\{P, q, r, s\}$	$\{P, r\}$
$\{P, r\}$	$\{P, q, s\}$	$\{P\}$
$\{P, q, r, s\}$	$\{P, q, r, s\}$	$\{P, r, s\}$
$\{P, q, s\}$	$\{P, q, r, s\}$	$\{P, r, s\}$
$\{P, r, s\}$	$\{P, q, s\}$	$\{P, s\}$
$\{P, s\}$	$\{P, q, s\}$	$\{P, s\}$

$$Q = \{P, \{P, q\}, \{P, q, r\}, \{P, q, r, s\}, \{P, r\}, \{P, q, s\}, \{P, r, s\}, \{P, s\}\}$$

The State diagram of the DFA is as follows



NFA = DFA ($\therefore$ Equivalent to NFA)

Example 48 =

Construct the equivalent DFA for the following NFA; $N = \{ \{q_0, q_1, q_2\}, \{0, 1\}, \delta, q_0, \{q_2\} \}$

δ	0	1
$\rightarrow q_0$	q_0	q_0, q_1
q_1	q_2	q_2
$* q_2$	$\emptyset$	$\emptyset$

Sol $\{Q, \Sigma, \delta, q_0, F\}$

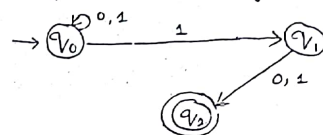
$Q = \{q_0, q_1, q_2\}$

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

$q_0 = \{q_0\}$

$F = \{q_2\}$

State graph / Transition graph of NFA:



Let $D = \{Q_D, \Sigma, \delta_D, \{q_0\}, F_D\}$ be the equivalent DFA.

$Q_D = \{ \emptyset, \{q_0\}, \{q_1\}, \{q_2\}, \{q_0, q_1\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\} \}$

$F_D = \{ \{q_2\}, \{q_0, q_2\}, \{q_1, q_2\}, \{q_0, q_1, q_2\} \}$

The transition δ_D is shown in the following table.

δ	0	1
$\emptyset$	$\emptyset$	$\emptyset$
$\rightarrow \{q_0\}$	$\{q_0\}$	$\{q_0, q_1\}$
$\{q_0, q_1\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$
$* \{q_0, q_2\}$	$\{q_0\}$	$\{q_0, q_1\}$
$* \{q_0, q_1, q_2\}$	$\{q_0, q_2\}$	$\{q_0, q_1, q_2\}$

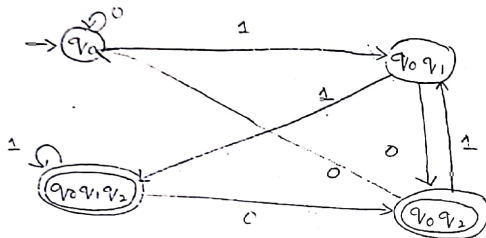
State table of DFA.

Step 1: Write the states of the DFA.

The State diagram of the DFA is as follows:-

$S = \{ q_0, q_0q_1, q_0q_2, q_0q_1q_2 \}$

$\Sigma = \{0, 1\}$, $q_0 = \{q_0\}$, $F = \{q_2\}$



NFA = DFA ($\because$ Equivalent to NFA).

17-Sept-2020

* Minimization of DFA :-

Minimization of DFA is the task of transforming a given DFA into an equivalent DFA that has a minimum no. of states.

These two DFA's are called equivalent if they recognize the same regular language.

Input :- DFA

Output :- Minimization DFA

$X \rightarrow$ String & $|X| \rightarrow$ length of the string

In MDFA we have to find 0, 1, 2, ... equivalents

$\Rightarrow$ 0 equivalent - Do separate normal state & final state

$\Rightarrow$ 1 equivalent - We have to check two alphabets whether they come under same set or not.

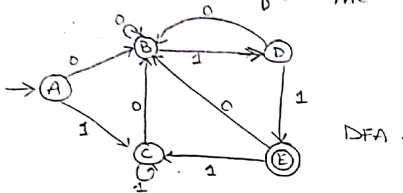
point i) Both state should go to same state set.

point ii) All state should be in same set.

Note:- If we get the same op for two equivalent then we have to stop the process.

Example 1:-

Construct the MDFA for the following DFA.



Sol $\{Q, \Sigma, S, q_0, F\}$

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

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

$q_0 = \{A\}$

$F = \{E\}$

State table of DFA:-

S	0	1
A	B	C
B	B	D
C	B	E
D	B	E
E	B	C

$\Rightarrow$ 0 Equivalent:- Do separate normal state & final state

$\Rightarrow \{A, B, C, D\}, \{E\}$

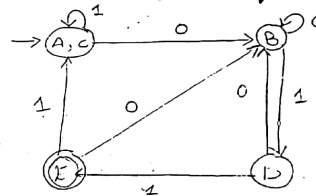
$\Rightarrow$ 1 Equivalent:- $\{A, B, C\}, \{D\}, \{E\}$

$\Rightarrow$ 2 Equivalent:- $\{A, C\}, \{B\}, \{D\}, \{E\}$

$\Rightarrow$ 3 Equivalent:- $\{A, C\}, \{B\}, \{D\}, \{E\}$

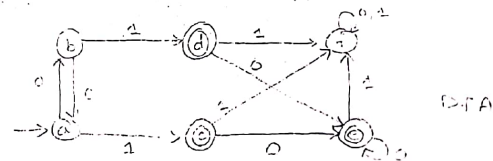
As we are getting the same result in equivalent 2 & 3. So, stop the process.

The state diagram of the MDFA is as follows



Example 2:-

Construct the MDFA for the following DFA



Sol $\{Q, \Sigma, S, q_0, F\}$

$Q = \{a, b, c, d, e, f\}$

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

$q_0 = \{a\}$

$F = \{f\}$

State table of DFA:-

S	0	1
→ a	b	c
b	a	d
* c	e	f
* d	e	f
* e	e	f
f	f	f

→ 0 equivalent := {a, b, f}, {c, d, e}

→ 1 equivalent := {a, b, f}, {c, d, e}

→ 2 equivalent := {a, b, f}, {c, d, e}

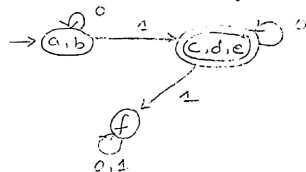
As we are getting the same result in equivalent 1 & 2 - so, stop the process.

$Q = \{a, b, f, c, d, e\}$; $\Sigma = \{0, 1\}$

MDFA State table:-

S	0	1
{a, b}	{a, b}	{c, d, e}
{c, d, e}	{c, d, e}	{f}
{f}	{f}	{f}

MDFA Graph / diagram:-



21 Sept 2020

[2nd Method := Forminization]

* MDFA with unreachable State using Myhill Nerode Theorem - Table Filling Method

Step 1:- Draw a table for all pairs of States (P, Q) not necessarily connected directly [All are unmarked initially].

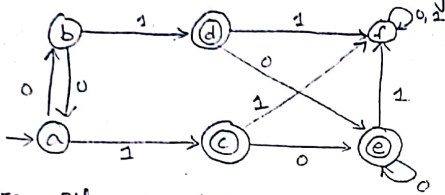
Step 2:- Consider every State pair (P, Q) in the DFA where $P \notin F$ & $Q \notin F$ or vice versa & mark them. [Here F is the set of Final States].

Step 3:- Repeat this Step until we cannot mark anymore state. If there is an unmarked pair (P, Q), mark it if the pair $\{S(P, A), S(Q, A)\}$ is marked for some i/p alphabet.

Step 4:- Combine all the unmarked pairs (P, Q) & make them a single state in the reduced DFA.

Example 3 :-

Myhill Nerode Theorem - Table Filling Method



DFA with unreachable state.

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

Step 1

Remove the upper part

a b c d e f

	a	b	c	d	e	f
a						
b						
c						
d						
e						
f						

Step 2

[We have to mark the box by seeing that one should be a normal state or should be a final state]

Remaining up unpaired pairs are :-

(a,b) (d,c) (e,c) (e,d) (a,f) (b,f)

$\delta(a,0) \Rightarrow b$
 $\delta(b,0) \Rightarrow a$

$\delta(a,1) \Rightarrow c$
 $\delta(b,1) \Rightarrow d$

$\delta(d,0) \Rightarrow e$
 $\delta(c,0) \Rightarrow e$

$\delta(d,1) \Rightarrow f$
 $\delta(c,1) \Rightarrow f$

$\delta(e,0) \Rightarrow e$
 $\delta(c,0) \Rightarrow e$

$\delta(e,1) \Rightarrow f$
 $\delta(c,1) \Rightarrow f$

$\delta(e,0) \Rightarrow e$
 $\delta(d,0) \Rightarrow e$

$\delta(e,1) \Rightarrow f$
 $\delta(d,1) \Rightarrow f$

$\delta(a,0) \Rightarrow b$
 $\delta(f,0) \Rightarrow f$

$\delta(a,1) \Rightarrow c$
 $\delta(f,1) \Rightarrow f$

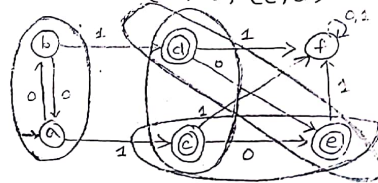
$\delta(b,0) \Rightarrow a$
 $\delta(f,0) \Rightarrow f$

$\delta(b,1) \Rightarrow d$
 $\delta(f,1) \Rightarrow f$

From the above pairs we got two new pairs.

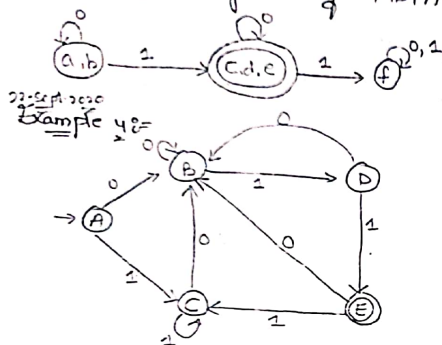
⇒ Remaining unpaired pairs are

(a,b), (d,c), (e,c), (e,d)



Step 3

The State diagram of MDFA is as follows:



Step 1:-

	A	B	C	D	E
A					
B					
C					
D					
E					

	A	B	C	D	E
A					
B					
C					
D					
E					

Remove the upper part

After removing the upper part

Step 2:-

	A	B	C	D	E
A					
B		✓			
C			✓		
D	✓	✓	✓		
E	✓	✓	✓	✓	

Step 3:-

Remaining unpaired pairs are

(A,B), (A,C), (B,C), (A,D), (B,D), (C,D)

$S(A,0) \Rightarrow B$
 $S(B,0) \Rightarrow B$ }
 $S(A,1) \Rightarrow C$
 $S(B,1) \Rightarrow D$ }

$S(A,0) \Rightarrow B$
 $S(C,0) \Rightarrow B$ }
 $S(A,1) \Rightarrow C$
 $S(C,1) \Rightarrow C$ }

$S(B,0) \Rightarrow B$
 $S(C,0) \Rightarrow B$ }
 $S(B,1) \Rightarrow D$
 $S(C,1) \Rightarrow C$ }

$S(A,0) \Rightarrow B$
 $S(D,0) \Rightarrow B$ }
 $S(A,1) \Rightarrow C$
 $S(D,1) \Rightarrow E$ }

$S(B,0) \Rightarrow B$
 $S(D,0) \Rightarrow B$ }
 $S(B,1) \Rightarrow D$
 $S(D,1) \Rightarrow E$ }

$S(C,0) \Rightarrow B$
 $S(D,0) \Rightarrow B$ }
 $S(C,1) \Rightarrow C$
 $S(D,1) \Rightarrow E$ }

From the above pairs we got 3 new pairs.

⇒ Remaining 3 Unpairs..

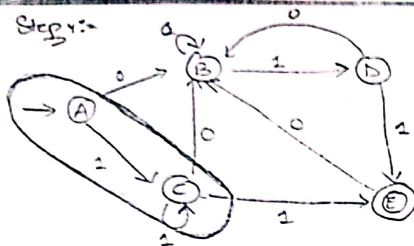
(A,B) (A,C) (B,C)

$S(A,0) \Rightarrow B$
 $S(B,0) \Rightarrow B$ }
 $S(A,1) \Rightarrow C$
 $S(B,1) \Rightarrow D$ }

$S(B,0) \Rightarrow B$
 $S(C,0) \Rightarrow B$ }
 $S(B,1) \Rightarrow C$
 $S(C,1) \Rightarrow D$ }

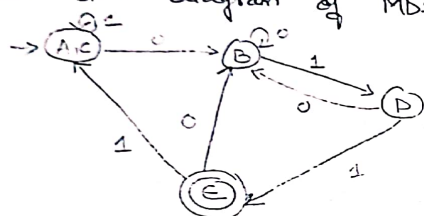
From the above we can see that (A,B) is already marked as a pair. So, we can delete it.

Step 1:-



DFA

The State diagram of MDFA is as follows.



Myhill Nerode Theorem - Table filling method.

Step 1 - Draw a table for all pairs of states (P, Q) not necessarily connected directly [all are unmarked initially].

Step 2 Consider every state pair (P, Q) in the DFA where $P \in F$ and $Q \notin F$ or vice versa and mark them, [Here F is a set of final states].

Step 3 Repeat this step until we cannot mark anymore states. If there is a unmarked pair (P, Q) mark it if the pair & $\delta(P, A), \delta(Q, A)$ is marked so some (P, A) must.

Step 4 - Combine all the unmarked pairs (P, Q) and then

23. Sept. 2020

⊕ Regular Expression:-

→ A Regular Expression can be recursively defined as follows.

i) ϵ is a Regular Expression indicates the language containing an empty string ($L(\epsilon) = \{\epsilon\}$)

ii) ϕ is a Regular Expression denoting an empty language. ($L(\phi) = \{\}$) ($\phi \in \Sigma^*$)

iii) x is a Regular Expression where $L = \{x\}$

iv) If X is a Regular Expression denoting the language $L(X)$ & Y is a Regular Expression denoting the language $L(Y)$, then.

→ $X+Y$ is a Regular Expression corresponding to the language $L(X) \cup L(Y)$ where $L(X+Y) = L(X) \cup L(Y)$.

→ $X \cdot Y$ is a Regular Expression corresponding to the language $L(X) \cdot L(Y)$ where $L(X \cdot Y) = L(X) \cdot L(Y)$

→ R^* is a Regular Expression corresponding to the language $L(R^*)$ where $L(R^*) = (L(R))^*$ Closure

v) If we apply any of the rules several times from 1 to 5, they are Regular Expressions.

Example:-

→ When we don't have empty string (ϵ) then we use union

$$\{0, 1, 2, 3\} \Rightarrow R = 0 + 1 + 2 + 3$$

$$\{bba, a, b, aab\} \Rightarrow R = bba + a + b + aab$$

→ When we have Empty String (ϵ) then we use Concatenation

$$\{\epsilon, abc\} \Rightarrow R = \epsilon \cdot abc$$

→ When we have Empty String (ϵ) & Increasing term then we use ^{Kleene} Closure ~~plus~~ Star

$$\{\epsilon, a, aa, aaa, \dots\} \Rightarrow R = a^*$$

→ When we ^{don't} have Empty String (ϵ) but we have Increasing term then we use Kleene plus

$$\{1, 11, 111, 1111, \dots\} \Rightarrow R = 1^+$$

24 Sept 2020

* Identities Related to Regular Expressions

→ Given R, P, L, Q as regular expressions, the following identities hold -

$$i) \phi^* = \epsilon$$

$$ii) \epsilon^* = \epsilon$$

$$iii) RR^* = R^*R$$

$$iv) R^*R^* = R^*$$

$$v) (R^*)^* = R^*$$

$$vi) (PQ)^*P = P(QP)^*$$

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

$$viii) R + \phi = \phi + R = R \text{ (The identity for union)}$$

$$ix) RE = \epsilon R = R \text{ (The identity for Concatenation)}$$

$$x) \phi L = L\phi = \phi \text{ (The annihilator for Concatenation)}$$

$$xi) R + R = R \text{ (Idempotent Law)}$$

$$xii) L(M+N) = LM + LN \text{ (Left distributive Law)}$$

$$xiii) (M+N)L = ML + NL \text{ (Right distributive Law)}$$

$$xiv) \epsilon + RR^* = \epsilon + R^*R = R^*$$

*** Arden's theorem :-**

Statement:- It states that "If P & Q are two regular expressions over Σ , and if P does not contain ϵ , then the following eqn in R given by $R = Q + RP$ has a unique soln i.e.; $R = QP^*$." That means, whenever we get any equation in the form of $R = Q + RP$, then we can directly replace it by $R = QP^*$. So, here first we will prove that $R = QP^*$ is the soln of this eqn & then we will also prove that it is the unique soln of the eqn.

Proof:-

$$R = Q + RP \quad \text{--- (1)} \quad [\because R = QP^*]$$

$$R = Q + (QP^*)P \quad [\because P^*P = P^*]$$

$$R = Q + QP^*P \quad [\because Q + QP^*P = Q(P^*P)]$$

$$R = Q + QP^* \quad [\because P^*P = P^*]$$

Hence proved

Now,

$$R = Q + RP \quad \text{--- (2)}$$

Substituting eqn (1) in eqn (2)

$$R = Q + (Q + RP)P$$

$$R = Q + QP + RP^2$$

$$R = Q + QP + (Q + RP)P^2$$

$$R = Q + QP + QP^2 + RP^3$$

$$R = Q + QP + \dots + QP^n + RP^{n+1} \quad [\because R = QP^*]$$

$$R = Q + QP + QP^2 + \dots + QP^n + (QP^*)P^{n+1}$$

$$R = Q [E + P + P^2 + \dots + P^n + P^*P^{n+1}]$$

$$R = QP^*$$

Hence proved

28 Sept 2020

*** Application of Finite Automata (FA) :-**

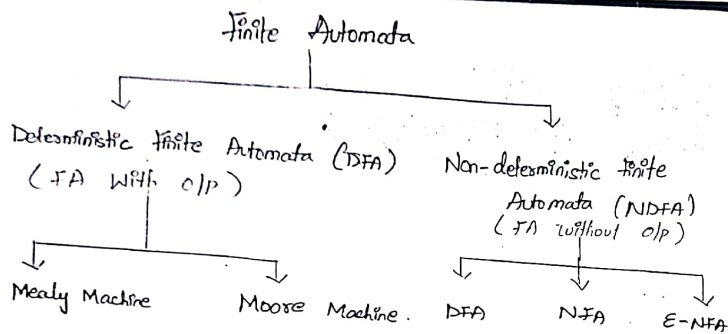
→ We have several applications based on finite automata & finite state machine. Some are given below

i) A finite automata is highly useful to design lexical analyzers.

ii) A finite automata is useful to design text editors.

iii) A finite automata is highly useful to design spell checkers.

iv) A finite automata is useful to design circuit design (Transducer).



Mealy Machine :-

A Mealy Machine is defined as a machine in theory of Computation whose o/p values are determined by both its current state & current i/p's. In this machine at most one transition is possible.

→ It has 6 tuples: $(Q, q_0, \Sigma, O, \delta, \lambda)$

Q is finite set of states

q_0 is the initial state

Σ is the i/p alphabet

O is the o/p alphabet

δ is transition function which maps $Q \times \Sigma \rightarrow Q$

λ is the o/p function which maps $Q \times \Sigma \rightarrow O$

Moore Machine :-

A Moore machine is defined as a machine in theory of Computation whose o/p values are determined only by its current state.

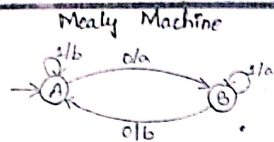
→ It has also 6 tuples $(Q, q_0, \Sigma, O, \delta, \lambda)$

$Q, q_0, \Sigma, O, \delta$ are same as Mealy Machine

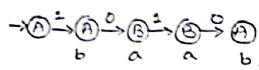
λ is the o/p function which maps $Q \rightarrow O$

Mealy Vs Moore Machine

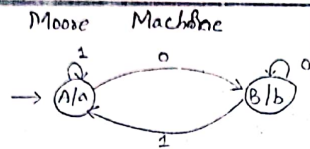
Mealy Machine	Moore Machine
O/p depends on present state as well as present i/p	O/p depends only upon present state
If i/p changes, o/p also changes.	If i/p changes, o/p does not change
Less number of states are required.	More no. of states are required
There is more H/W requirements for circuit implementation.	There is less H/W requirement for circuit implementation.
They react slower to i/p's (one clock cycle later).	They react faster to i/p's
Asynchronous o/p's generation.	Synchronous o/p & state generation.
O/p is placed on transitions.	O/p is placed on states
It is difficult to design.	Easy to design.



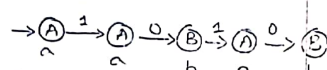
String: 1010



n no. of alphabets = n no. of o/p's



String: 1010



n no. of alphabets = n+1 o/p's

3:07:50

④ Epsilon NFA:-

→ If we have Empty Symbol (ϵ), we can perform the operations.

→ If we have Empty String (ϕ), we cannot perform the operations.

Regular NFA $\rightarrow S: Q \times \Sigma \rightarrow 2^Q$

Epsilon NFA $\rightarrow S: Q \times \Sigma \cup \epsilon \rightarrow 2^Q$

→ NFA is also have 5 tuples $(Q, q_0, \Sigma, \delta, F)$

Q is finite set of states

q_0 is the initial state

Σ is the i/p alphabet

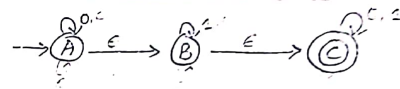
F is the final state.

Example 1:-

Conversion of E-NFA to Regular NFA:-

Rule:-

State ϵ^* i/p ϵ^*



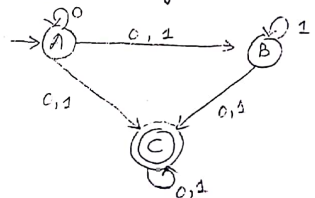
Every state can be reached from the start state.

State table:-

NFA	0	1
$\rightarrow A$	$\{A, B, C\}$	$\{B, C\}$
B	$\{C\}$	$\{B, C\}$
$\rightarrow C$	$\{C\}$	$\{C\}$

	ϵ^*	0	ϵ^*		ϵ^*	1	ϵ^*
A	A	A	$\{A, B, C\}$	A	A	ϕ	-
B	ϕ	-	-	B	B	$\{B, C\}$	-
C	C	$\{C\}$	-	C	C	$\{C\}$	-
B	B	ϕ	-	B	B	B	$\{B, C\}$
C	C	$\{C\}$	-	C	C	C	$\{C\}$
C	C	C	$\{C\}$	C	C	C	$\{C\}$

State graph / diagram:-



Conversion E-NFA to NFA.

Example 2:-

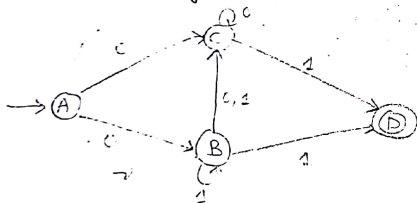
Conversion of ϵ -NFA to Regular NFA.



State table:-

NFA	0	1	State	ϵ^*	0	ϵ^*	State	ϵ^*	1	ϵ^*
$\rightarrow A$	{B, C}	$\emptyset$	A	A	B {B, C}		A	A	$\emptyset$	$\emptyset$
B	{C}	{B, C, D}	B	B	$\emptyset$	$\emptyset$	B	B	B {B, C}	$\emptyset$
C	{C}	{D}	C	C	{C}		C	D	{D}	
* D	$\emptyset$	$\emptyset$	D	D	$\emptyset$	$\emptyset$	D	D	$\emptyset$	$\emptyset$

State graph / diagram:-



Conversion ϵ -NFA to NFA

01.08.2020

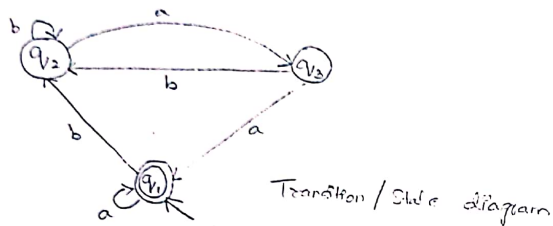
Conversion of Finite Automata (FA) to Regular Expression (RE).

Rules for Corresponding the FA to RE.

- 1) The transition diagram must not have NULL or ϵ transitions.
- 2) It must have only one initial state.
- 3) It Vertices are V_1 to V_n .
- 4) V_i is the Regular Expression (RE) representing the set of strings accepted by the system when V_i is the final state.
- 5) ϵ is added to the initial state.
- 6) If there are more than one final state then we have to take the union of final states.

Example 1:-

Construct a regular expression corresponding to the automata given below -



Sol Here the initial state & final state is q_1 .

→ The eq's for the three states q_1, q_2 & q_3 are follows -

→ $q_1 = q_1a + q_3a + \epsilon$ (ϵ move is because q_1 is the initial state 0)

→ $q_2 = q_1b + q_2b + q_3b$

→ $q_3 = q_2a$

→ Now, we will solve these three eq's -

→ $q_2 = q_1b + q_2b + q_3b$

$q_2 = q_1b + q_2b + (q_2a)b$ ($\because$ sub value of q_3)

$q_2 = \frac{q_1b}{\epsilon} + \frac{q_2(b+ab)}{F}$

$q_2 = q_1b(b+ab)^*$ ($\because$ Applying Arden's Theorem

$R = \emptyset + RP \Rightarrow R = \emptyset P^*$)

→ $q_1 = q_1a + q_3a + \epsilon$

$q_1 = q_1a + (q_2a)a + \epsilon$ ($\because$ Sub Value of q_3)

$q_1 = q_1a + (q_1b(b+ab)^*)aa + \epsilon$ ($\because$ sub value of q_2)

$q_1 = \frac{q_1(a+b(b+ab)^*aa)}{\epsilon} + \frac{\epsilon}{\epsilon}$

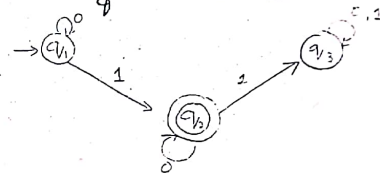
$q_1 = \epsilon(a+b(b+ab)^*aa)^*$

$q_1 = (a+b(b+ab)^*aa)^*$

$\therefore$ Hence, the regular expression is $(a+b(b+ab)^*aa)^*$

Example 2:-

Conversion of FA to RE.



Sol Given $\emptyset = \{q_1, q_2, q_3\}$

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

$q_0 = \{q_1\}$

$F = \{q_2\}$

The eq's for 3 states q_1, q_2 & q_3 are follows -

$q_1 = q_10 + \epsilon$ — (1)

$q_2 = q_11 + q_20$ — (2)

$q_3 = q_21 + q_30 + q_2$ — (3)

Solving (1), (2) & (3) eqⁿ

$$q_1 = q_1 \cdot 0 + \epsilon \quad \text{--- (1)} \quad [\because \text{Arden's Theorem } R = Q_1 + RP \Rightarrow R = Q_1 P^*]$$

$$\text{So } q_1 = \epsilon \cdot 0^*$$

$$q_1 = 0^* \quad \text{--- (4)} \quad [\because \text{Concatenation property Identity } \epsilon \cdot R = R]$$

Sub eqⁿ (4) in (2)

$$q_2 = 0^* 1 + q_2 \cdot 0 \quad \therefore R = Q_1 + RP \Rightarrow R = Q_1 P^*$$

$$q_2 = 0^* 1 \cdot 0^*$$

$\therefore$ Hence Regular expression is $0^* 1 0^*$

⊛ Applications of Regular Expression:

→ Basically regular expressions are used in search tools

→ Text file search in Unix (tool: egrep).

→ This command searches for a text pattern in the file & list the file names containing that pattern.

os ~~01.10.2020~~

⊛ Algebraic Laws of Regular Expressions:-

1. Associativity & Commutativity:-

i) $L+M+M+L$ [Commutative law for union] union of two language in either order and same

ii) $(L+M)+N = L+(M+N)$ [Associative law for union]

iii) $(LM)N = L(MN)$ [Associative law for Concatenation]

iv) $LM \neq ML$ [Concatenation is not Commutative]

2. Identities & Annihilators:-

i) $\phi + L = L + \phi = L$ [ϕ is the identity for union]

ii) $\epsilon L = L \epsilon = L$ [ϵ is the identity for Concatenation]

iii) $\phi L = L \phi = \phi$ [ϕ is the annihilator for Concatenation]

3. Distributive Laws:-

i) $L(M+N) = LM + LN$

ii) $(M+N)L = ML + NL$

4. The Idempotent law:-

$L+L = L$ [Idempotence law of union]

5. Laws Involving Closures:-

i) $(L^*)^* = L^*$ ii) $\phi^* = \epsilon$ iii) $\epsilon^* = \epsilon$

iv) $L^+ = LL^* = L^*L$ [$\therefore$ Note: $L^+ = L + LL + LLL + \dots$]

v) $L^* = \epsilon + L + LL + LLL + \dots$

vi) $L? = \epsilon + L$

⊕ properties of Regular language :-

→ It is of two types :-

1. Closure properties on Regular languages.

2. Decision properties of Regular languages

1. Closure properties on Regular languages:

Closure properties on Regular languages are defined as certain operations on regular language which are guaranteed to produce regular language. Closure refers to some operation on a language, resulting in a new language that is of same "type" as originally operated on i.e., regular.

→ Regular languages are closed under following operations.
Consider L & M are regular languages:

→ Kleene closure:

R is a regular expression whose language is L, M .

R^* is a regular expression whose language is L^* .

→ positive closure:

R is a regular expression whose language is L, M .

R^+ is a regular expression whose language is L^+ .

i> Union:-

If L_1 & L_2 are regular languages then $L_1 \cup L_2$ is regular.

Proof:- If L_1 & L_2 are regular, then there exist regular expressions r_1 & r_2 such that $L_1 = L(r_1)$ & $L_2 = L(r_2)$.

By defⁿ $r_1 + r_2$ is a regular expressions denoting the language $L_1 \cup L_2$.

ii> Concatenation:-

If L_1 & L_2 are regular languages then $L_1 L_2$ is regular.

Proof:- Similar to the proof of Union using defⁿ of R.E

iii> Star:-

If L is a regular language then L^* is regular.

Proof:- Similar to the proof of Union using defⁿ of R.E

iv> Complement:-

If L is a regular language then $\bar{L} (= \Sigma^* - L)$ is regular.

Proof:- Let $D = (Q, \Sigma, \delta, q_0, F)$ be some DFA for L .

We Construct a DFA D' for $\bar{L}$ where

$D' = (Q, \Sigma, \delta, q_0, Q - F)$ which will be the DFA for $\bar{L}$.

v> Intersection:-

If L_1 & L_2 are regular then $L_1 \cap L_2$ is also regular.

Proof:- $L_1 \cap L_2 = \overline{\bar{L}_1 \cup \bar{L}_2}$ [De Morgan's Law]

L_1, L_2 are regular $\Rightarrow \bar{L}_1, \bar{L}_2$ are regular.

$\Rightarrow \bar{L}_1 \cup \bar{L}_2$ is regular $\Rightarrow \overline{\bar{L}_1 \cup \bar{L}_2}$ is regular

$\Rightarrow L_1 \cap L_2$ is regular

2. Decision properties of Regular Languages:-

A decision property for a class of languages is an algorithm that takes a formal description of a language (e.g., a DFA) and tells whether or not some property holds.

Example: Is language L empty?

Why Decision properties?

→ When we talked about protocols represented as DFA's, we noted that important properties of a good protocol were related to the language of the DFA.

→ Example: "Does the protocol terminate?" = "Is the language finite?"

→ Example: "Can the protocol fail?" = "Is the language nonempty?"

Why Decision properties? — (2)

→ We might want a "Smallest" representation for a language, e.g., a minimum-state DFA or a shortest RE.

→ If you can't delete "Are these two languages the same?"

→ i.e., do two DFA's define the same language?

You can't find a "Smallest".

→ Decision properties of Regular languages are follows:-

i) Emptiness } — ①

ii) Non-Emptiness

iii) Finiteness } — ②

iv) Infiniteness

v) Membership — ③

vi) Equality — ④

1) Emptiness & Non-Emptiness:-

Step 1: Select the state that cannot be reached from the initial states & delete them (remove unreachable states).

Step 2: If the resulting machine contains at least one final states then the finite automata accepts the non-empty language.

Step 3: If the resulting machine is from final states then finite automata accepts empty language.

2. Finiteness & Infiniteness:-

Step 1: Select the state that cannot be reached from the initial state & delete them (remove unreachable states).

Step 2: Select the state from which we cannot reach the final state & delete them (remove dead states)

→ $A \rightarrow B \rightarrow C$ (if A is not final state, we don't need C)

Step 3: If the resulting machine contains loops or cycles then the finite automata accepts infinite language.

Step 4: If the resulting machine do not contain loop or cycles then the finite automata accepts finite language.

3. Membership :=

→ Membership is a property to verify an arbitrary string is accepted by a finite automaton or not. i.e. it is a member of the language or not.

→ Let M is a finite automata that accepts some strings over an alphabet, & let ' w ' be any string defined over the alphabet, if there exist a transition path in M , which starts at initial state & ends in any one of the final state, then string ' w ' is a member of M , otherwise ' w ' is not a member of M .

4. Equality :=

Two finite state automata M_1 & M_2 is said to be equal if and only if, they accept the same language.

Minimize the finite state automata & the minimal

DFA will be unique.

Pumping Lemma for Regular Languages:

Theorem:

Let L be a regular language. Then there exists a constant ' n ' (which depends on L) such that for every string w in L (w belongs to L) such that $|w| \geq n$ then we can break w into three strings, $w = xyz$, s.t. such that:

1. $|y| > 0$ or $y \neq \epsilon$

2. $|xy| \leq n$

3. for all $k \geq 0$, the string xy^kz is also in L .

Applications of Pumping Lemma:

→ Pumping Lemma is to be applied to show that certain languages are not regular. It should never be used to show a language is regular.

→ If L is a regular, it satisfies pumping lemma.

→ If L does not satisfy pumping lemma, it is non-regular.

Method to prove that a language L is not regular:

→ At first, we have to assume that L is regular.

So, the pumping lemma should hold for L .

→ Use the pumping lemma to obtain a contradiction:

Select w such that $|w| \geq n$

Select y such that $|y| \geq 1$

Select x such that $|xy| \leq n$

Assign the remaining string to z .

→ Select k such that the resulting string is not in L ,
Hence L is not regular.

Theorem: pumping lemma for Regular language.

Let L be a Regular language. Then there exists a constant n (which depends on L) such that for every string $w \in L$ such that $|w| \geq n$, we can break w into three strings $w = xyz$ such that:

1. $y \neq \epsilon$

2. $|xy| \leq n$

3. For all $k \geq 0$, the string xy^kz is also in L .

Proof: Suppose L is regular then $L = L(A)$ for some DFA A .

Suppose A has n states, let us consider any string $w = a_1 a_2 \dots a_m$, where $m \geq n$ and each a_i is an input symbol for $i = 0, 1, \dots, n$ let $p_i = \delta(q_0, a_1 a_2 \dots a_i)$.

Note that δ is the transition function of A and q_0 is the start state of A . That means $p_0 = q_0$.

→ By the pigeonhole principle, it is not possible for the $n+1$ different p_i 's for $i = 0, 1, \dots, n$ to be distinct, since there are only n different states.

Thus, we can find two different integers $i \neq j$, with $0 \leq i < j \leq n$ such that $p_i = p_j$.

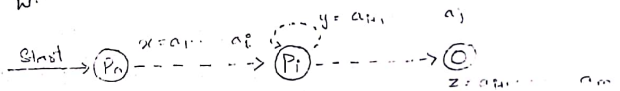
Now we can break $w = xyz$ as follows:

1. $x = a_1 a_2 \dots a_i$

2. $y = a_{i+1} \dots a_j$

3. $z = a_{j+1} \dots a_m$

That is, x takes us to p_i once; y takes us from p_i back to p_i (since p_i is also p_j) and z is the balance of w .



Note: x may be empty, y may be empty.

However y cannot be empty since $i < j$.

If the automaton receives the input xy^kz , $k \geq 0$ we need to check whether it accepts or not.

When $k=0$: Automaton A goes from the start state $q_0 (= p_0)$ to p_i on input x . Since $p_i = p_j$, it must go from p_i to accept state on input z .

When $k > 0$: Automaton A goes from q_0 to p_i on input x , cycles p_i to p_i , k items on input y^k and then goes to the accepting state on input z . Thus, for any $k \geq 0$, $xy^kz \in L$.

Example:-

• prove that the following languages are not regular

(i) $L_1 = \{0^m 1^n \mid m \geq 0\}$

(ii) $L_2 = \{a^l b^m c^k \mid l+m \leq b\}$

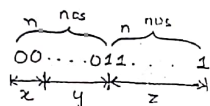
(iii) $L_3 = \{0^k \mid k \text{ is a prime}\}$

(iv) $L_4 = \{w \in \Sigma^* \mid \text{no. of } 0\text{'s in } w = \text{no. of } 1\text{'s in } w\}$

sol

(i) 0 [Given $n \geq 0$

$\forall$ [Choose $w = 0^n 1^n$ [$\therefore |w| = 2n \geq n$]



Now given a partition $w = xyz$

where $|xy| \leq n$ & $|y| > 0$

Note that the strings x & y consists of only 0's, otherwise $|xy| > n$

Let $y = 0^t$. So $x = 0^{n-t}$ and $z = 1^n$

Choose $k=0$ then $xy^kz = xz = 0^{n-t} 1^n \notin L_1$

$\Rightarrow$ The language $\{0^m 1^n \mid m \geq 0\}$ is not regular

(ii) Given $n \geq 0$

Choose $w = a^n b^n c^{2n}$ [$|w| > n$ & $n+n \leq 2n$]

Now given a partition $w = xyz$ where $|xy| \leq n$ & $|y| > 0$

So we have $y = a^t$ for some $t > 0$ & $t < n$.

So $x = a^{n-t}$, $y = a^t$, $z = b^n c^{2n}$

Choose $k=2$ then $xy^2z = a^{n-t} a^{2t} b^n c^{2n} = a^{n+t} b^n c^{2n}$

Here $n+t+n = 2n+t \neq 2n$ [$\therefore t$ is $< n$ & > 0]

$\Rightarrow xy^2z \notin L_2$

So L_2 is not regular.

Exercise: Show that the language L consisting of all palindromes over $(a+b)^*$ is not regular.

[Hint: Choose $w = 0^n 10^n$ & Show that for $k=0$ $xy^kz \notin L$

(iii) Given $n \geq 0$

Choose $w = 0^\tau$ where τ is a prime $\geq n$

Now given a partition, $w = xyz$ where $|y| = t, t > 0$.

Choose $k = \tau + 1$

then $xy^kz = xyz \cdot y^{k-1} = 0^\tau (0^t)^{k-1} = 0^{\tau + (k-1)t}$

Now $\tau + (k-1)t = \tau + \tau t$ [$\therefore k = \tau + 1 \Rightarrow k-1 = \tau$]
 $= \tau(1+t)$

As τ is a prime no. $\geq n$, $n \geq 0$ so τ is atleast 2.

And $t \geq 1$ as $|y| > 0$. so $t+1$ is atleast 2.

So $\tau(1+t)$ is a composite number.

$\Rightarrow xy^kz \notin L_3 \Rightarrow L_3$ is not regular.

(iv) $L_4 = \{w \in \Sigma^* \mid \text{no. of } 0\text{'s in } w = \text{no. of } 1\text{'s in } w\}$

$L(0^* 1^*) \cap L_4 = L_1$

$\rightarrow 0^* 1^*$ is regular because we can construct a DFA for this.

$\rightarrow$ If L_4 is regular $\Rightarrow L_1$ is regular.

$\rightarrow$ This is a Contradiction.

Hence L_4 is not regular.

06.10.2020

UNIT-02

CONTEXT FREE GRAMMARS AND LANGUAGES

③ Context-free languages:-

→ The Class of Context-free languages generalizes over the Class of regular languages i.e., every regular language is a context free language.

→ The reverse of this is not true, i.e., every Context-free language is not necessarily regular.

For example, as we will see, $\{0^k 1^k \mid k \geq 0\}$ is Context-free but not regular.

→ Many issues & questions we asked for regular languages will be the same for Context-free languages.

i> Machine model - PDA (push-Down Automata)

ii> Descriptors - CFG (Context-Free Grammar)

iii> Pumping lemma for Context-free languages (to find CFL's limit)

iv> Closure of Context-free languages with respect to various operations.

v> Algorithms & Conditions for finiteness or emptiness.

→ Some analogies don't hold, e.g., non-determinism in PDA makes a difference in particular deterministic PDA's define a subset of the Context-free languages.

→ We will talk on non-deterministic

→ Informally, a Context-free language (CFL) is a language generated by a Context-free Grammar (CFG).

→ What is a CFG?

→ Informally, a CFG is a set of rules for deriving (or generating) strings (or sentences) in a language.

Note: A grammar generates a string, whereas a machine accepts a string.

Example CFGs:

- | | |
|--|------|
| < sentence > → < noun-phrase > < verb-phrase > | (1) |
| < noun-phrase > → < proper-noun > | (2) |
| < noun-phrase > → < determiner > < common-noun > | (3) |
| < proper-noun > → John | (4) |
| < proper-noun > → Jill | (5) |
| < common-noun > → Car | (6) |
| < common-noun > → hamburger | (7) |
| < determiner > → a | (8) |
| < determiner > → the | (9) |
| < verb-phrase > → < verb > < adverb > | (10) |
| < verb-phrase > → < verb > | (11) |
| < verb > → drives | (12) |
| < verb > → eats | (13) |
| < adverb > → slowly | (14) |
| < adverb > → frequently | (15) |

Example Derivation:

$\langle \text{Sentence} \rangle \Rightarrow \langle \text{noun-phrase} \rangle \langle \text{verb-phrase} \rangle$ by (1)
 $\Rightarrow \langle \text{proper-noun} \rangle \langle \text{verb-phrase} \rangle$ by (2)
 $\Rightarrow \text{Jill} \langle \text{verb-phrase} \rangle$ by (5)
 $\Rightarrow \text{Jill} \langle \text{verb} \rangle \langle \text{adverb} \rangle$ by (10)
 $\Rightarrow \text{Jill drives} \langle \text{adverb} \rangle$ by (12)
 $\Rightarrow \text{Jill drives frequently}$ by (15)

Informally, a CFG consists of:

- A set of replacement rules, each having a left-hand side (LHS) & a right-hand side (RHS).
- Two types of symbols: Variables & terminals.
- LHS of each rule is a single variable (no terminals).
- RHS of each rule is a string of zero or more variables & terminals.
- A string consists of only terminals.
- Formally, a Context-Free Grammar (CFG) is a 4-tuple.
 $G = (V, T, P, S)$
 - V - A finite set of variables or non-terminals
 - T - A finite set of terminals (V & T do not intersect: do not use same symbols) This is our Σ .
 - P - A finite set of productions, each of the form

$A \rightarrow \alpha$, where A is in V & α is in $(V \cup T)^*$

Note that α may be ϵ .

S - A starting non-terminal (S is in V)

Example CFG for $\{0^k 1^k \mid k \geq 0\}$:

$G = (\{S\}, \{0, 1\}, P, S)$ // Remember: $G = (V, T, P, S)$

P :
 (1) $S \rightarrow 0S1$ or just simply $S \rightarrow 0S1 \mid \epsilon$
 (2) $S \rightarrow \epsilon$

Example Derivations:

$S \Rightarrow 0S1$ (1)
 $\Rightarrow 01$ (2) $S \Rightarrow \epsilon$ (2)

 $S \Rightarrow 0S1$ (1)
 $\Rightarrow 00S11$ (1)
 $\Rightarrow 000S111$ (1)
 $\Rightarrow 000\epsilon 111$ (2)

Note that G "generates" the language $\{0^k 1^k \mid k \geq 0\}$

Example CFG for ?:

$$G = \{ (A, B, C, S) \cdot \{a, b, c\} \cdot P, S \}$$

P:

$$(1) S \rightarrow ABC$$

$$(2) A \rightarrow aA$$

$$(3) A \rightarrow \epsilon$$

$$(4) B \rightarrow bB$$

$$(5) B \rightarrow \epsilon$$

$$(6) C \rightarrow cC$$

$$(7) C \rightarrow \epsilon$$

$$A \rightarrow aA | \epsilon$$

$$B \rightarrow bB | \epsilon$$

$$C \rightarrow cC | \epsilon$$

Example Derivation:

$$S \Rightarrow ABC \quad (1)$$

$$\Rightarrow BC \quad (3)$$

$$\Rightarrow C \quad (5)$$

$$\Rightarrow \epsilon \quad (7)$$

$$S \Rightarrow ABC \quad (1)$$

$$\Rightarrow aABC \quad (2)$$

$$\Rightarrow aaABC \quad (2)$$

$$\Rightarrow aaBC \quad (3)$$

$$\Rightarrow aabBC \quad (4)$$

$$\Rightarrow aabc \quad (5)$$

$$\Rightarrow aabcC \quad (6)$$

$$\Rightarrow aabc \quad (7)$$

Generation of Derivation Tree:

A derivation tree or parse tree is an ordered rooted tree that graphically represents the semantic information a string derived from a Context-free grammar.

→ Representation Technique.

• Root Vertex - Must be labelled by the Start Symbol.

• Vertex - Labeled by a non-terminal Symbol

• leaves - labeled by a terminal symbol of Σ .

If $S \rightarrow x_1 x_2 \dots x_n$ is a production rule in a CFG, then the parse tree / derivation tree will be as follows-

→ There are two different approaches to draw a derivation tree-

Top-down Approach -

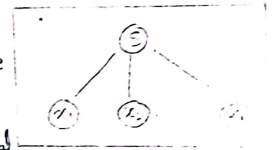
• Starts with the Starting Symbol S.

• Goes down to tree leaves using productions.

Bottom-up Approach -

• Starts from tree leaves.

• Proceeds upward to the root which is the Starting Symbol S.



01.04.2020

⊛ Derivation or yield of a Tree :-

The derivation or the yield of a parse tree is the final string obtained by concatenating the labels of the leaves of the tree from left to right, ignoring the Nulls. However, if all the leaves are Null, derivation is Null.

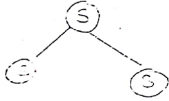
Q) Let a CFG $\{N, T, P, S\}$ be $N = \{S\}$, $T = \{a, b\}$,

Starting Symbol = S , $P = S \rightarrow SS \mid asb \mid \epsilon$.

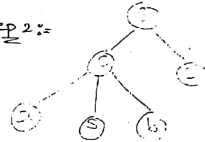
One derivation from the above CFG is "abaabb".

LMD :-

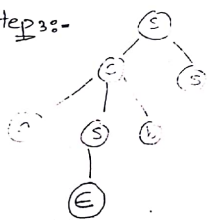
Step 1 :-



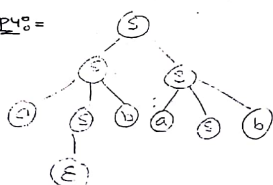
Step 2 :-



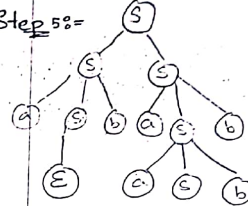
Step 3 :-



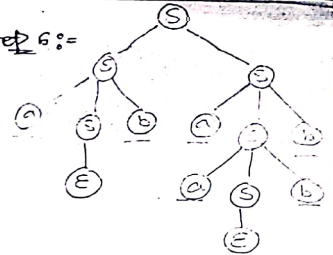
Step 4 :-



Step 5 :-



Step 6 :-

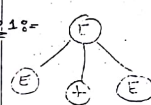


The answer we got is "abaabb".

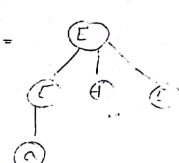
Q) Let any set of production rules in a CFG be $E \rightarrow E+E \mid E^*E \mid E \mid a$. Over an alphabet $\{a\}$.

The Leftmost derivation for the string "a+a*a" may be as follows

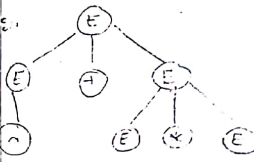
Step 1 :-



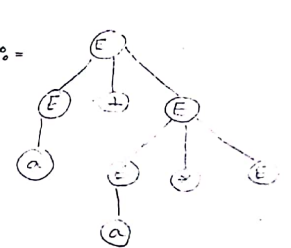
Step 2 :-

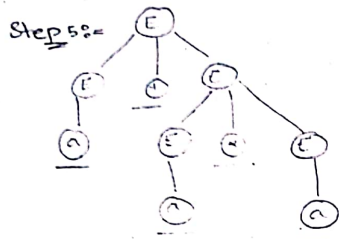


Step 3 :-



Step 4 :-

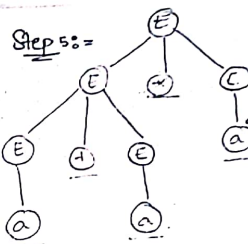
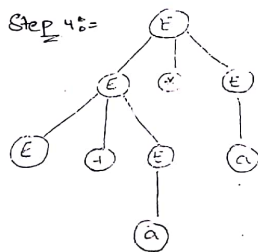
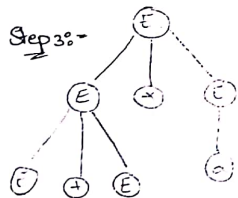
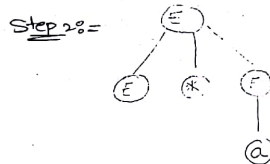
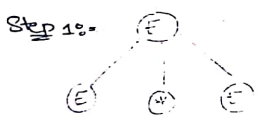




The answer we got is "a+a*a".

5) Let any set of production rules in a CFG be $E \rightarrow E+E | E \times E | E | a$. Over an alphabet Σ .

The Rightmost derivation for the string "a+a*a" may be as follows:-



The answer we got is "a+a*a".

* Difference b/w Ambiguous Grammar & Unambiguous Grammar.

Ambiguous Grammar (AG)

In Ambiguous grammar, the leftmost & rightmost derivations are not same.

Amot of non-terminals in AG is less than in UAG.

length of the parse tree in AG is comparatively short.

Speed of derivation of a tree in AG is faster than that of UAG.

AG generates more than one parse tree.

AG Contains ambiguity.

Unambiguous Grammar (UAG)

In Unambiguous grammar, the leftmost & rightmost derivations are same.

Amot of non-terminals in UAG is more than in AG.

length of the parse tree in UAG is comparatively large.

Speed of derivation of a tree in UAG is slower than that of AG.

UAG generates only one parse tree.

UAG does not contain any ambiguity.

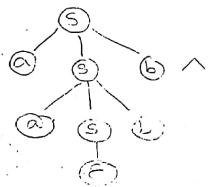
12 Oct 2020

④ Converting from CFL to CFG:-

1. $a^n b^n, n \geq 0 \Rightarrow S \rightarrow asb | \epsilon$

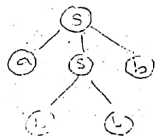
$aabb$

equal no. of a's & b's.



2. $a^n b^{n+2}, n \geq 0 \Rightarrow S \rightarrow asb | bbi$

$abbb$



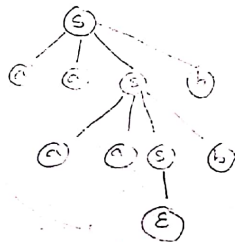
i.e. $a^1 b^{1+2} \Rightarrow a^1 b^3$

3. $a^{2n} b^n, n \geq 0 \Rightarrow S \rightarrow aasb | \epsilon$

$aaaa bbb$

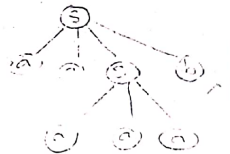
Here a's is 4 & b's is 3

i.e. $a^{2(2)} b^2 \Rightarrow a^4 b^2$



4. $a^{2n+3} b^n, n \geq 0 \Rightarrow S \rightarrow aasb | aaa$

$aaaaa b$



Here a's is 5 & b's is 1

i.e. $a^{2(1)+3} b^1 \Rightarrow a^5 b^1$

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

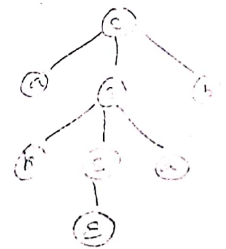
$\begin{cases} S \rightarrow asb | \epsilon \\ A \rightarrow aA | a \end{cases} \Rightarrow S \rightarrow AS_1$

aab



6. $\{w | na(w) = nb(w)\} \Rightarrow S \rightarrow asb | bsa | \epsilon$

$abab$



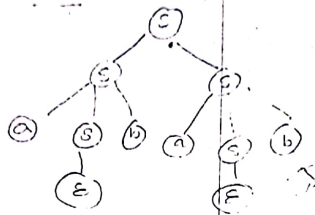
If we read abba then

we can take the same

$$\{W \mid na(w) = nb(w)\} \Rightarrow S \rightarrow asb \mid bsa \mid \underline{ss} \mid \epsilon$$

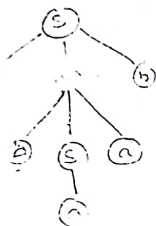
abba

the given string is derived
by using the given CFG



$$\{W \mid WR \cup W(a+b)WR\} \Rightarrow S \rightarrow asb \mid bsa \mid a \mid b \mid \epsilon$$

odd

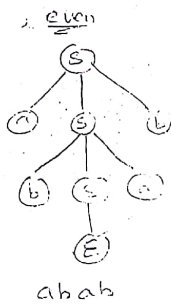


abaab

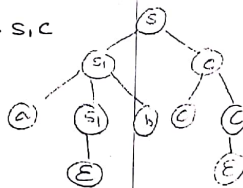
$$8. a^m b^n c^n \quad m, n \geq 0$$

$$\left. \begin{array}{l} S_1 \rightarrow asb \mid \epsilon \\ C \rightarrow cc \mid \epsilon \end{array} \right\} \Rightarrow S \rightarrow s_1 c$$

abc



abab



Example 1:- Write a CFG for the language

$$L = \{WcW^R \mid W \in (a,b)^*\}$$

Let G be a CFG for the language L .

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

$$\text{Here, } V = \{S\}$$

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

& P is given by

$$\left. \begin{array}{l} S \rightarrow aSa \\ S \rightarrow bSb \\ S \rightarrow c \end{array} \right\} \text{ (or) } S \rightarrow asa \mid bsb \mid c$$

Let us check that $abcbba$ can be derived from the given CFG.

$$S \Rightarrow asa \text{ (use the production } S \rightarrow asa)$$

$$\Rightarrow absba \text{ (use } S \rightarrow bSb)$$

$$\Rightarrow abbsbba \text{ (use } S \rightarrow bSb)$$

$$\Rightarrow \underline{abcbba} \text{ (use } S \rightarrow c)$$

So string $abcbba$ can be derived from given CFG.

Example 2:- Write a CFG for the regular expression
 $x = 0^*1(0+1)^*$

To generate a grammar, we divide the regular expression into 3 parts 0^* , 1 , $(0+1)^*$ represented by 3 non-terminal symbols.

Let the CFG be $G = (V, T, P, S)$

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

$T = \{0, 1\}$

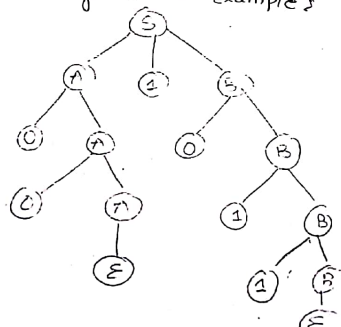
Each production P are defined as

$S \rightarrow A1B$
 $A \rightarrow 0A | \epsilon$
 $B \rightarrow 0B | 1B | \epsilon$

$S \rightarrow A1B | 0A | 0B | 1B | \epsilon$

Let us test the grammar against an example
 001011

$S \Rightarrow A1B$
 $\Rightarrow 0A10B$
 $\Rightarrow 00A101B$
 $\Rightarrow 001011$



Write a CFG, which generates palindrome for binary numbers.

The grammar will generate palindrome for binary numbers, that is 00, 010, 001100, -

Let G be the CFG, $G = (V, T, P, S)$

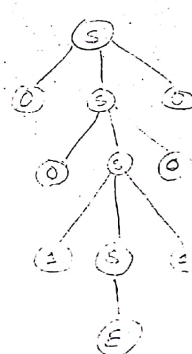
Here, $V = \{S\}$

$T = \{0, 1\}$

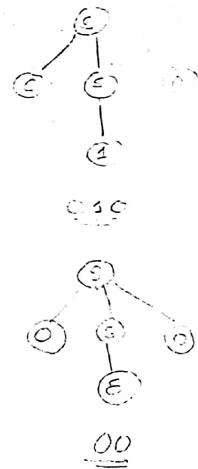
$S \rightarrow 0S0 | 1S1$
 $S \rightarrow 01 | 10 | \epsilon$

Each production P is defined as

$S \rightarrow 0S0 | 1S1$
 $S \rightarrow 01 | 10 | \epsilon$



001100



00

Example 2:- 1.1.21 - -

(Example:- Give the automaton for the following grammar:

$\epsilon \rightarrow OS / 1A / 1$

$\epsilon \rightarrow OA / 1A / 10 / 1$

Let the automaton will have 3 states S, A, F

He for production $S \rightarrow OS, S \rightarrow 1A, S \rightarrow 1$

for production $A \rightarrow OA, A \rightarrow 1A, A \rightarrow 0, A \rightarrow 1$

At 13.08.2020

$S \rightarrow OS$ Hierarchy of Languages:-

A. Regular languages $\rightarrow$ Finite State Machines, Regular Expression.

Let Context Free languages $\rightarrow$ Context Free Grammar, Push-down Automata.

$S \Rightarrow$

$\Rightarrow$ Pushdown Automata (PDA):-

$\Rightarrow$ Informally:-

$\Rightarrow$ A PDA is an NFA- ϵ with a stack.

ii) Transitions are modified to accommodate stack operations.

following $\rightarrow$ A DFA can "remember" only a finite amount of information, whereas a PDA can "remember" an infinite amount of (certain types of) information, in one memory stack.

Example:-

$\{0^n 1^n \mid 0 \leq n\}$ is not regular, but

$\{0^n 1^n \mid 0 \leq n \leq k\}$, for some fixed k is regular, for any fixed k .

or $k=3$:

$L = \{\epsilon, 01, 0011, 000111\}$

In a DFA, each state remembers a finite amount of information.

To get $\{0^n 1^n \mid 0 \leq n\}$ with a DFA would require an infinite number of states using the preceding technique.

An infinite stack solves the problem for $\{0^n 1^n \mid 0 \leq n\}$ as follows:

Read all 0's & place them on stack.

Read all 1's & match with the corresponding 0's on the stack.

→ Only need two states to do this in a PDA.

→ Similarly for $\{0^n 1^m 0^{n+m} \mid n, m \geq 0\}$

④ Formal Definition of a PDA:

→ A pushdown Automata (PDA) is a Seven-tuple:

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

i) $Q \rightarrow$ A finite set of states.

ii) $\Sigma \rightarrow$ A finite I/P alphabet.

iii) $\Gamma \rightarrow$ A finite Stack alphabet.

iv) $q_0 \rightarrow$ The initial / starting state q_0 is in Q .

v) $Z_0 \rightarrow$ A starting Stack Symbol, is in Γ

// need not always remain at the bottom of Stack.

vi) $F \rightarrow$ A set of final / accepting states, which is a subset of Q . ($F \subseteq Q$)

vii) $\delta \rightarrow$ A transition function, where

$$\delta: Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \text{finite subsets of } Q \times \Gamma^*$$

Consider the various parts of δ :

$$Q \times (\Sigma \cup \{\epsilon\}) \times \Gamma \rightarrow \text{finite subsets of } Q \times \Gamma^*$$

LHS:-

Q on the LHS means that at each step in a computation, a PDA must consider its current state.

Γ on the LHS means that at each step in a computation, a PDA must consider the symbol on top of its stack.

$\Sigma \cup \{\epsilon\}$ on the LHS means that at each step in a computation, a PDA may or may not consider the current I/P symbol, i.e., it may have epsilon transitions.

RHS:-

"finite subsets" on the RHS means that at each step in a computation, a PDA may have several options.

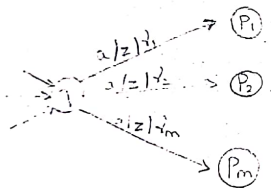
Q on the RHS means that each option specifies a new state.

Γ^* on the RHS means that each option specifies one or more stack symbols that will replace the top stack symbol, but in a specific sequence.

Two Types of PDA transitions:-

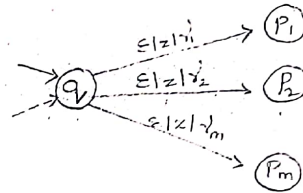
$$1. S(q, a, z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$$

- Current State is q .
- Current i/p Symbol is a .
- Symbol currently on top of the stack z .
- Move to State p_i from q .
- Replace z with γ_i on the stack (leftmost Symbol on top).
- Move the i/p head to the next i/p Symbol.



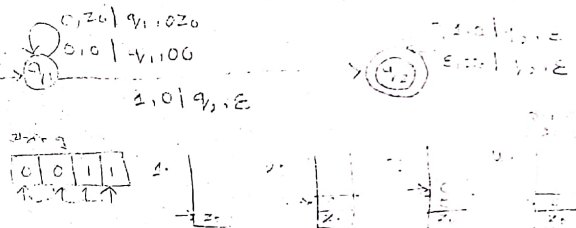
$$2. S(q, \epsilon, z) = \{(p_1, \gamma_1), (p_2, \gamma_2), \dots, (p_m, \gamma_m)\}$$

- Current State is q .
- Current i/p Symbol is not considered.
- Symbol currently on top of the stack z .
- Move to State p_i from q .
- Replace z with γ_i on the stack (leftmost Symbol on top).
- No i/p Symbol is read.



Example: $0^n 1^n, n \geq 0$

$$M = (\{q_1, q_2\}, \{0, 1\}, \{L, \#\}, \delta, q_1, \#, \emptyset)$$



δ :

i) $\delta(q_1, 0, \#) = \{(q_1, L\#)\}$ // Stack order: L on top, then # below.

ii) $\delta(q_1, 1, \#) = \emptyset$ // illegal, string rejected, when will it happen?

iii) $\delta(q_1, 0, L) = \{(q_1, LL)\}$

iv) $\delta(q_1, 1, L) = \{(q_2, \epsilon)\}$

v) $\delta(q_2, 1, L) = \{(q_2, \epsilon)\}$

vi) $S(q_2, \epsilon, \#) = \{(q_2, \epsilon)\}$ // if ϵ read & stack hits bottom, accept.

vii) $S(q_2, \epsilon, L) = \emptyset$ // illegal, string accepted.

viii) $S(q_1, \epsilon, \#) = \{(q_2, \epsilon)\}$ // $n=0$, accept.

Goal: (acceptance)

↳ Read the entire i/p string

↳ Terminate with an empty stack.

→ Informally, a string is accepted if there exists a computation that uses up all the i/p & leaves the stack empty.

Example:-

i) 0011

$\Rightarrow (q_1, 0011, \#) \vdash$

$(q_1, 011, L\#) \vdash$

$(q_1, \epsilon, L\#) \vdash$

$(q_2, \epsilon, L\#) \vdash$

$(q_2, \epsilon, \#) \vdash$

$(q_2, \epsilon, \epsilon) : \text{accept}$

ii) 011

$\Rightarrow (q_1, 011, \#) \vdash$

$(q_1, \epsilon, L\#) \vdash$

$(q_2, \epsilon, \#) \vdash$

$\emptyset : \text{reject}$

iii) 001

$\Rightarrow (q_1, 001, \#) \vdash$

$(q_1, 01, L\#) \vdash$

$(q_1, \epsilon, L\#) \vdash$

$(q_2, \epsilon, L\#) \vdash$

$(q_2, \epsilon, \#) \vdash$

14. Oct. 2020

Example: Give the automaton for the following grammar.

$S \rightarrow 0S / 1A / 1$

$A \rightarrow 0A / 1A / 0 / 1$

Sol $S \rightarrow 0S$

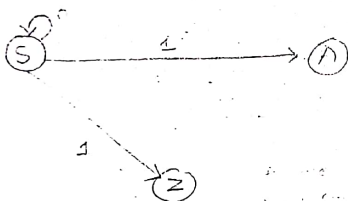
$S \rightarrow 1A$

$S \rightarrow 1$

$\therefore A, S \rightarrow$ non-terminals

$0, 1 \rightarrow$ terminals.

S-production



This is the production of S.

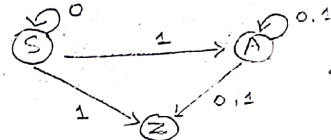
$A \rightarrow 0A$

$A \rightarrow 1A$

$A \rightarrow 0$

$A \rightarrow 1$

A-production



This is the production of A.

Example 2: Consider the regular grammar $G = (V, T, P, S)$

where $V = \{A, B, S\}$, $T = \{a, b\}$ & P is defined as

$S \rightarrow abA$

$S \rightarrow B$

$S \rightarrow baB$

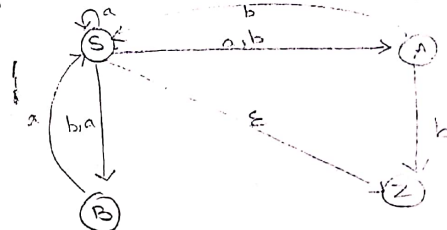
$S \rightarrow \epsilon$

$A \rightarrow bs$

$B \rightarrow as$

$A \rightarrow b$

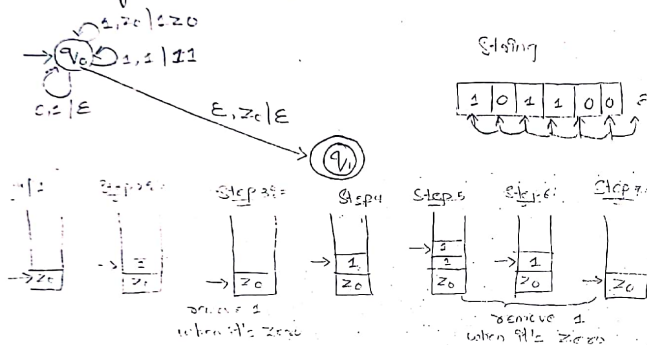
Construct a NFA M such that $L(M) = L(G)$.



PDA Example 2: $L = \{w \in \{0,1\}^* \mid n_0(w) = n_1(w)\}$

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

Transition funⁿ: $\delta = Q \times (\Sigma \cup \epsilon) \times \Gamma \rightarrow Q \times \Gamma^*$



8:

$\delta \Rightarrow q_0, \epsilon, \Gamma \rightarrow q_0, \Gamma^*$
 $2 \delta \Rightarrow q_0, 0, 1 \rightarrow q_0, \epsilon$
 $\rightarrow \delta \Rightarrow q_0, 1, 1 \rightarrow q_0, 11$
 $\rightarrow \delta \Rightarrow q_0, \epsilon, Z_0 \rightarrow q_1, \epsilon$
 $\rightarrow$
 $\rightarrow$
 $\rightarrow$
 $\rightarrow$

Transition function

19. Oct. 2020

UNIT-03

PROPERTIES OF CFL'S

* Properties of Context-free languages:-

- i) Union
 - ii) Concatenation
 - iii) Stars
 - iv) Intersection
 - v) Complement
- closed property
- Not-closed property.

i) Union:-

Context-free languages are closed under: Union.

L_1 is Context-free
 L_2 is Context-free $\rightarrow L_1 \cup L_2$ is Context-free

Example:-

language

Grammars

$L_1 = \{a^n b^n\}$

$S_1 \rightarrow a S_1 b \mid \lambda$ or ϵ

$L_2 = \{w w^R\}$

$S_2 \rightarrow a S_2 a \mid b S_2 b \mid \lambda, \epsilon$

palindrome

Union

$L = \{a^n b^n\} \cup \{w w^R\}$

$S \rightarrow S_1 S_2$

In general:

For Context-free languages L_1, L_2
with Context-free grammars G_1, G_2
and Start Variables / non-terminals S_1, S_2 .

The grammar of the Union $L_1 \cup L_2$
has new start Variable S
and additional production $S \rightarrow S_1 | S_2$

ii) Concatenation:-

CFL's are closed under: Concatenation

$\left. \begin{array}{l} L_1 \text{ is Context free} \\ L_2 \text{ is Context free} \end{array} \right\} \rightarrow L_1 L_2 \text{ is Context-free}$

Example:-

language	Grammar
$L_1 = \{a^n b^n\}$	$S_1 \rightarrow a S_1 b \lambda, \epsilon$
$L_2 = \{w w^R\}$	$S_2 \rightarrow a S_2 a b S_2 b \lambda, \epsilon$

Concatenation

$L = \{a^n b^n\} \{w w^R\}$ $S \rightarrow S_1 S_2$

In general:

For Context-free languages L_1, L_2
with Context-free grammars G_1, G_2
and Start Variables S_1, S_2

The grammar of the Concatenation $L_1 L_2$
has new start Variable S
and additional production $S \rightarrow S_1 S_2$

iii) Star Operation:-

CFL's are closed under: Star operation.

L is Context free $\rightarrow L^*$ is Context free

Example:-

language	Grammar
$L = \{a^n b^n\}$	$S \rightarrow a S b \lambda, \epsilon$

Star operation

$L = \{a^n b^n\}^*$	$S_1 \rightarrow S S_1 \lambda, \epsilon$
---------------------	---

In general:-

For Context-free language L
with Context-free grammar G
and Start Variable S

In The grammar of the Star operation L^*
 has new Start Variable S_1
 with additional production
 $S_1 \rightarrow SS_1 | \lambda, \epsilon$

and $\oplus$ Negative properties of CFL's :-

The $\Rightarrow$ Intersection :-

has CFL's are not closed under : Intersection

and L_1 is Context free
 $\Rightarrow L_2$ is Context free } $\rightarrow L_1 \cap L_2$ not necessarily Context free.

C# Example :-

$L_1 = \{a^n b^n c^m\}$

L_2 Context-free

Ex: $S \rightarrow AC$

$A \rightarrow aAb | \lambda, \epsilon$

$C \rightarrow cC | \lambda, \epsilon$

L_2

Intersection

$L_1 \cap L_2 = \{a^n b^n c^n\}$ NOT Context free

$\Rightarrow$ Complement :-

CFL's are not closed under : Complement.

L is Context free $\rightarrow \bar{L}$ not necessarily Context-free.

Example :-

$L_1 = \{a^n b^n c^m\}$

Context-free

$S \rightarrow AC$

$A \rightarrow aAb | \lambda, \epsilon$

$C \rightarrow cC | \lambda$

$L_2 = \{a^n b^m c^m\}$

Context-free

$S \rightarrow AB$

$A \rightarrow aA | \lambda, \epsilon$

$B \rightarrow bBc | \lambda, \epsilon$

Complement

$\overline{L_1 \cup L_2} = \bar{L}_1 \cap \bar{L}_2 = \{a^n b^n c^n\}$ NOT Context-free

In section 2.2.2

For Reduction of CFGs

with Simplification consist of following steps:-

- 1) Reduction of CFG
- 2) Removal of null production
- 3) Removal of unit production.

and 1) Reduction of CFGs:-

CFGs are reduced in two phases -

i) Phase 1 := Derivation of an equivalent grammar, G' , from the CFG, G , such that each variable derives some terminal string.

ii) Derivation procedure:-

Ex Step 1 - Include all symbols, w_1 , that derive some terminal & initialize $i = 1$.

Step 2 - Include all symbols, w_{i+1} , that derive w_i .

Step 3 - Increment i & repeat Step 2, until $w_{i+1} = w_i$.

Step 4 - Include all production rules that have w_i in it.

Phase 2 - Derivation of an equivalent grammar, G'' , from the CFG, G' , such that each symbol appears in a sentential form.

Derivation procedure:-

Step 1 - Include the start symbol in y_1 & initialize $i = 1$.

Step 2 - Include all symbols, y_{i+1} , that can be derived from y_i & include all production rules that have been applied.

Step 3 - Increment i & repeat Step 2, until $y_{i+1} = y_i$.

Example:-

$P: S \rightarrow AC \mid B, A \rightarrow a, C \rightarrow c \mid Bc, E \rightarrow aA \mid c$

Phase-1 := $T = \{a, c, e\}$

$w_1 = \{A, C, E\}$

$w_{1+1} \quad w_2 = \{A, C, E, S\}$

$w_{2+1} \quad w_3 = \{A, C, E, S\}$

$G' = \{ \underbrace{(A, C, E, S)}_{\text{(Non-terminal)}}, \underbrace{(a, c, e)}_{\text{(terminal)}}, \underbrace{P, (S)}_{\text{(production)}} \}$

New production:-

$P: S \rightarrow AC, A \rightarrow a, C \rightarrow c, E \rightarrow aA|e$

phase 2:-

$y_1 = \{S\}$ Start Symbol

$y_2 = \{S, A, C\} \quad \{S \rightarrow AC\}$

$y_{2+1} \quad y_3 = \{S, A, C, a, c\}$

$y_{3+1} \quad y_4 = \{S, A, C, a, c\}$ Stop

$G'' = \{(A, C, S), (a, c), P, (S)\}$

New production:-

$P: S \rightarrow AC, A \rightarrow a, C \rightarrow c$

25 Oct 2020

(*) Simplifying Null production:-

Ex. b). $S \rightarrow ABAC, A \rightarrow aA|e, B \rightarrow bB|e, C \rightarrow c$

L. $\Rightarrow$ We have two Null productions

L. $A \rightarrow e, B \rightarrow e$

① To eliminate the $A \rightarrow e$

$S \rightarrow ABAC$ [$\because A$ replace with e]

$S \rightarrow ABC|BAC|BC$ (Sub e in S in place of A)

$A \rightarrow aA$

$A \rightarrow a$

New production:-

$S \rightarrow ABAC|ABC|BAC|BC$

$A \rightarrow aA|a$

$B \rightarrow bB|e$

$C \rightarrow c$

② To eliminate the $B \rightarrow e$

$S \rightarrow ABAC|ABC|BAC|BC$ (Taking the S from the New production if Sub e in place of B)

$S \rightarrow AAC|AC|C$

$B \rightarrow bB$

$B \rightarrow b$

New production:-

$S \rightarrow ABAC|ABC|BAC|BC|AAC|AC|C$

(union of $C \rightarrow e$ production)

$A \rightarrow aA|a$

$B \rightarrow bB|b$

$C \rightarrow c$

This is the final answer.

Q) $S \rightarrow ASA | AB | b$, $A \rightarrow B$, $B \rightarrow b | \epsilon$

$\Rightarrow$ We have one direct Null production & one indirect Null production.

$B \rightarrow \epsilon$, $A \rightarrow \epsilon$ (indirectly)

1) To eliminate the $B \rightarrow \epsilon$

$S \rightarrow ASA | AB | b$

$S \rightarrow ASA | AB | a | b$

$A \rightarrow B | b$

$B \rightarrow b$

New production:-

$S \rightarrow ASA | AB | a | b$

$A \rightarrow B | b$

$B \rightarrow b$

2) To eliminate the $A \rightarrow \epsilon$

$S \rightarrow ASA | AB | a | b$

$S \rightarrow SA | AS | S | AB | a | b$

New production:-

$S \rightarrow ASA | AB | a | b | SA | AS | S$

$A \rightarrow B | b$

$B \rightarrow b$

31. Oct. 2020

* Simplifying the Unit production:-

Q) $S \rightarrow XY$

$X \rightarrow a$

$Y \rightarrow Z | b$

$Z \rightarrow M$

$M \rightarrow N$

$N \rightarrow a$

Step 1:- $M \rightarrow N$

Since $N \rightarrow a$ we add $M \rightarrow a$

New production:-

$S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z | b$, $Z \rightarrow M$, $M \rightarrow a$, $N \rightarrow a$

Step 2:- $Z \rightarrow M$

Since $M \rightarrow a$ we add $Z \rightarrow a$

N.P $\Rightarrow S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow Z | b$, $Z \rightarrow a$, $M \rightarrow a$, $N \rightarrow a$

Step 3:- $Y \rightarrow Z$

Since $Z \rightarrow a$ we add $Y \rightarrow a$

N.P $\Rightarrow S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow a | b$, $Z \rightarrow a$, $M \rightarrow a$, $N \rightarrow a$
unreachable state

$\Rightarrow$ remove unreachable state & write New production

N.P $\Rightarrow S \rightarrow XY$, $X \rightarrow a$, $Y \rightarrow a | b$

This is the final answer.

02-Nov-2020

* Chomsky Normal Form (CNF) :-

→ CNF stands for Chomsky Normal form. A CFG (Context free Grammar) is in CNF (Chomsky normal form) if all production rules satisfy one of the following conditions:

- i) Start Symbol generating ϵ . For example, $A \rightarrow \epsilon$.
- ii) A non-terminal generating two non-terminals. For example, $S \rightarrow AB$.
- iii) A non-terminal generating a terminal. For example, $S \rightarrow a$.

Ex. Example:-

1) $S \rightarrow ASA \mid AB, A \rightarrow B \mid s, B \rightarrow b \mid \epsilon$

2. Given production: $S \rightarrow ASA \mid AB, A \rightarrow B \mid s, B \rightarrow b \mid \epsilon$

① Since S appears in RHS we will add new state s' .

1. $P: s' \rightarrow s, S \rightarrow ASA \mid AB, A \rightarrow B \mid s, B \rightarrow b \mid \epsilon$

② Remove all Null production i.e. ϵ

$B \rightarrow \epsilon$ directly $A \rightarrow \epsilon$ indirectly

$B \rightarrow \epsilon$

$P: s' \rightarrow s, S \rightarrow ASA \mid AB \mid a, A \rightarrow B \mid s \mid \epsilon, B \rightarrow b$
 $A \rightarrow \epsilon$

$P: s' \rightarrow s, S \rightarrow ASA \mid AB \mid a \mid SA \mid AS \mid s, A \rightarrow B \mid s, B \rightarrow b$

③ Remove the unit production

$S \rightarrow s, s' \rightarrow s, A \rightarrow B, A \rightarrow s$

After removing the unit production

① $S \rightarrow s$

$P: s' \rightarrow s, S \rightarrow ASA \mid AB \mid a \mid SA \mid AS, A \rightarrow B \mid s, B \rightarrow b$

② $s' \rightarrow s$

$P: s' \rightarrow ASA \mid AB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid AB \mid a \mid SA \mid AS$

$A \rightarrow B \mid s, B \rightarrow b$

③ $A \rightarrow B$

$P: s' \rightarrow ASA \mid AB \mid a \mid SA \mid AS$

$S \rightarrow ASA \mid AB \mid a \mid SA \mid AS$

$A \rightarrow b \mid s, B \rightarrow b$

(iv) $A \rightarrow s$

P: $S' \rightarrow ASA \mid AB \mid a \mid SA \mid AS$
 $S \rightarrow ASA \mid AB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid ASA \mid AB \mid a \mid SA \mid AS$
 $B \rightarrow b$

(4) Satisfying the 2nd condition of Chomsky Normal form i.e. $S \rightarrow AB$.

→ From the above production we have the following states as

$S' \rightarrow ASA, S \rightarrow ASA, A \rightarrow ASA$

N.P: $S' \rightarrow AX \mid AB \mid a \mid SA \mid AS$ $\left\{ \begin{array}{l} X \rightarrow SA \\ \text{New non-terminal} \end{array} \right.$
 $S \rightarrow AX \mid AB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid AX \mid AB \mid a \mid SA \mid AS$
 $B \rightarrow b$
 $X \rightarrow SA$

(5) Satisfying the 3rd condition of CNF i.e. $S \rightarrow a$

$S' \rightarrow AB, S \rightarrow AB, A \rightarrow AB$

Substituting $Y \rightarrow a$ in $S' \rightarrow AB, S \rightarrow AB, A \rightarrow AB$

N.P: $S' \rightarrow AX \mid YB \mid a \mid SA \mid AS$
 $S \rightarrow AX \mid YB \mid a \mid SA \mid AS$
 $A \rightarrow b \mid AX \mid YB \mid a \mid SA \mid AS$
 $B \rightarrow b$
 $X \rightarrow SA$
 $Y \rightarrow a$

NULL Production Removal

In a CFN a Non-Terminal symbol 'A' is a nullable variable if there is a production $A \rightarrow \epsilon$ or there is a derivation that starts at 'A' and leads to ϵ . (like $A \rightarrow \dots \rightarrow \epsilon$)

Procedure for Removal

Step 1 → To remove $A \rightarrow \epsilon$ look for all production whose right side contains A

Step 2 → Replace each occurrences of 'A' in each of these production with ϵ

Step 3 → Add the resultant productions of the grammar

CNP

- 1) Start symbol $S' \rightarrow S$
- 2) Remove Null Production
- 3) Remove Unit Production
- 4) Replace each production $A \rightarrow B_1 B_2 \dots B_n$ where $n \geq 2$, with $A \rightarrow B_i$ where $i \geq 1$
- 5) $S \rightarrow AB$ or $S \rightarrow a$

07-Jan-2021

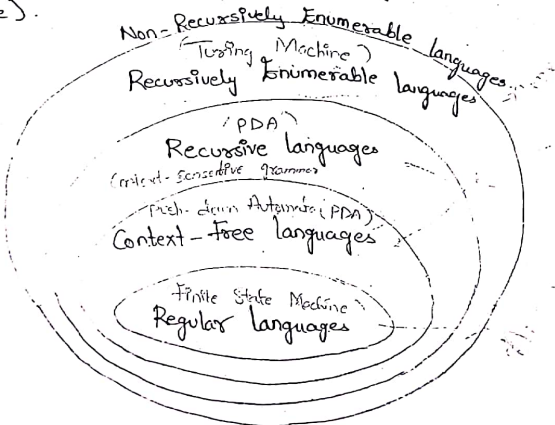
04. TURING MACHINE

* Turing Machine :-

- A Turing Machine is an accepting device which accepts the languages (recursively enumerable set) generated by type 0 grammars.
- It was invented in 1936 by Alan Turing.

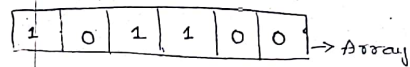
Hierarchy of languages :-

- Regular languages → Finite State Machines, Regular Expression
- Context Free language → Context Free Grammar, push-down
- Automata → Turing Machine (Recursively Enumerable language).



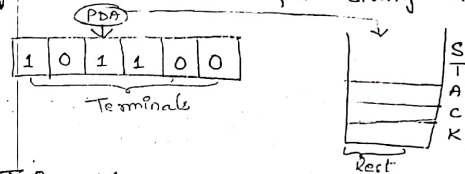
1) FSM - Finite State Machine

We will take the input string as



2) PDA - Pushdown Automata.

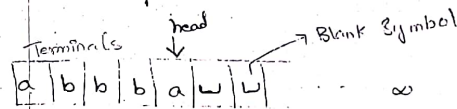
If we take the input string it uses stack



[∵ Rest is used to identify empty stack]

3) Turing Machine

→ It uses tape



Ans :- ab^*a (Tape)

ii) Tape Alphabets

$\Sigma = \{a, b, 0, 1, \gamma, Z, \dots\}$

dp symbol

ii) Blank 'L' is a special symbol

$L \notin \Sigma$

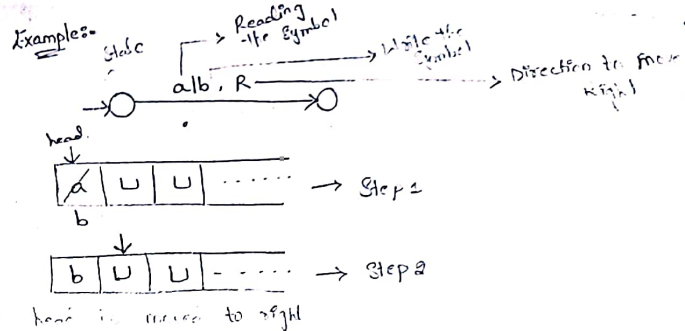
iii) Blank 'L' is used to fill the infinite tape

In Comparison with the previous automata:-
 For The following table Shows a Comparison of how a
 with Turing machine differs from finite Automaton &
 and pushdown Automaton.

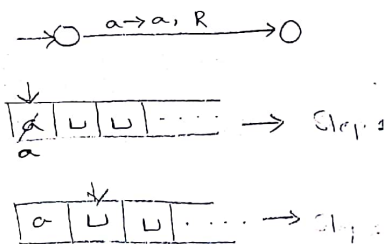
Machine	Stack Data Structure	Deterministic?
finite Automaton	Not applicable N.A	Yes
pushdown Automaton	Last in first out (LIFO)	No PDA > FA
Turing Machine	Infinite tape	Yes T.M > P.A.S.I.P.M

C. Configuration to Tapes:-

L. Example:-
 L. Tape: $\boxed{a} \boxed{a} \boxed{a} \boxed{b} \boxed{b} \boxed{b} \boxed{a} \boxed{a} \boxed{a} \boxed{\sqcup} \boxed{\sqcup}$
 Head $\downarrow$
 Ex. Ans: $a^3b^3a^3$
 L. Operation on tape:-
 L. Rule-01:-
 L. $\rightarrow$ Read / scan Symbol below the tape head.
 $\rightarrow$ update / write Symbol below the tape head.
 $\rightarrow$ Move the tape head one step "LEFT".
 $\rightarrow$ Move the tape head one step "RIGHT".



Examples:-



RULE-02:-

- $\rightarrow$ Control is with a sort of Finite State Machine
- $\rightarrow$ Initial State
- $\rightarrow$ final States
 - $\hookrightarrow$ Accept
 - $\hookrightarrow$ Reject

→ Computation can either

i> HALT & ACCEPT

ii> HALT & REJECT

iii> LOOP (Machine fail to HALT)

05 Jan 2021

② Formal Definition of Turing Machines:-

→ A Turing Machine (TM) is a Mathematical model which consists of an infinite length tape divided into cells on which input is given.

→ A Turing Machine can be formally described as a 7-tuple $(Q, X, \Sigma, \delta, q_0, B, F)$ where-

i> Q is a finite set of states.

ii> X is the tape alphabet.

iii> Σ is the input alphabet.

iv> δ is the transition function;

$\delta: Q \times X \rightarrow Q \times X \times \{ \text{Left-Shift}, \text{Right-Shift} \}$.

v> q_0 is the initial state.

vi> B is the blank symbol.

vii> F is the set of final states.

Recursively Enumerable languages:-

→ A language $L \subseteq \Sigma^*$ is said to be recursively enumerable if there exists a Turing Machine that accepts it.

Example-01:-

Q) Design a TM which recognize the language.

$L = 01^*0$

(one "0" followed by any number of "1" followed by one "0").

Sol Input string $\rightarrow 01110$

Deterministic Tuples are

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

$X = \{ 0, 1, x, y \}$

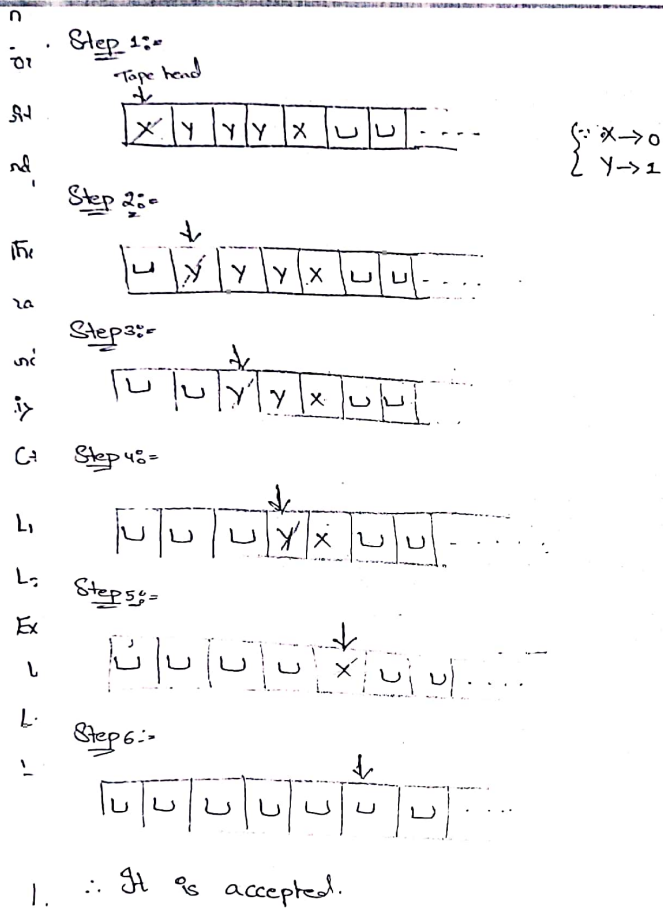
$\Sigma = \{ 0, 1 \}$

$\delta = 0 \rightarrow x, R$

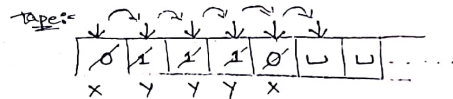
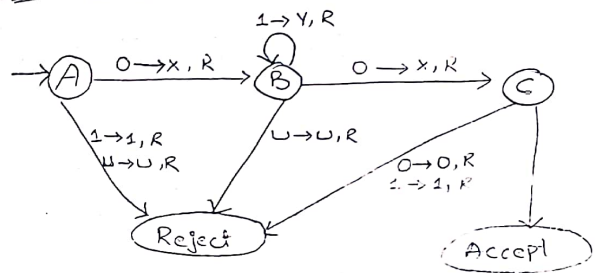
$q_0 = \{ A \}$

$B = \{ \sqcup \}$

$F = \begin{cases} \rightarrow \text{Accept} \\ \rightarrow \text{Reject} \end{cases}$



State Diagram:-



$\therefore$ It is Accepted.

15 Jan 2021

Example-02:-

B) Design a TTM which recognize the language:

$L = 0^n 1^n$

(Equal number of "0's" followed by equal number of "1's").

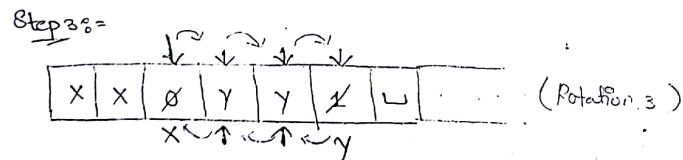
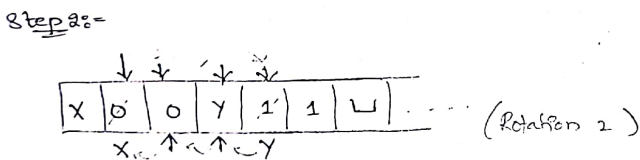
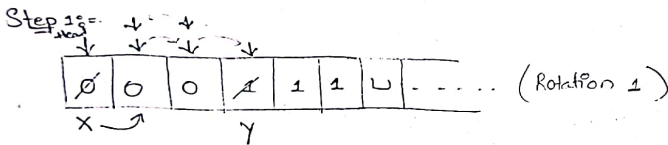
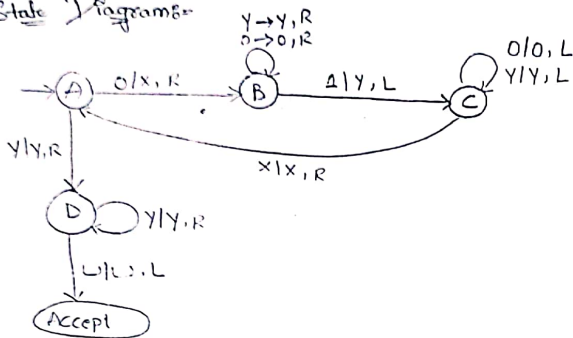
Sol Input String = 000111

Deterministic Tuples are

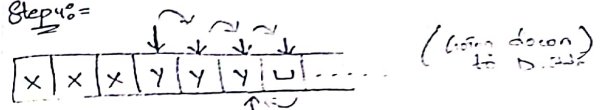
$Q = \{A, B, C, D\}$; $X = \{0, 1, X, Y\}$; $\Sigma = \{0, 1\}$

$\delta = 0 \rightarrow X, R$; $q_0 = \{A\}$; $B = U$; $F = \begin{cases} \text{Accept} \\ \text{Reject} \end{cases}$

State Diagram:-



Step 4:-



Equal no. of 0's & Equal no. of 1's

∴ It is Accepted.

18 Jan 2021

Example - 000111

Q) Even Palindrome

Input String = abaaba

Deterministic Tuples are.

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

$X = \{a, b, \sqcup\}$

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

$\delta = Q \times X \rightarrow Q$

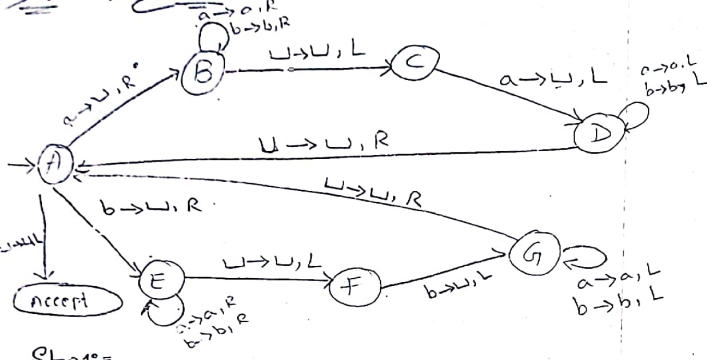
$q_0 = \{A\}$

$B = \{ \sqcup \}$

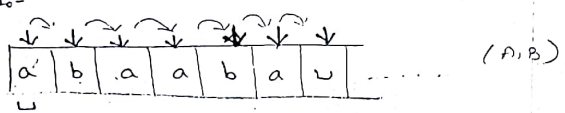
$F = \begin{cases} \rightarrow \text{Accept} \\ \rightarrow \text{Reject} \end{cases}$

n
or
us
nd
The
ha
and
ii)
C
L
L
Ex
L
L
L
L
1.

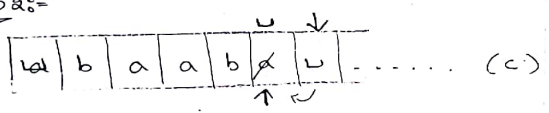
State Diagram:-



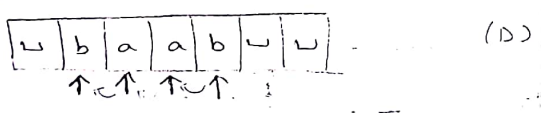
Step 1:-



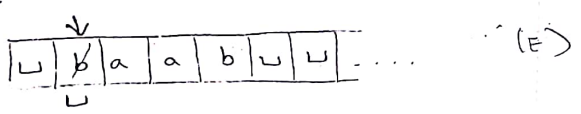
Step 2:-



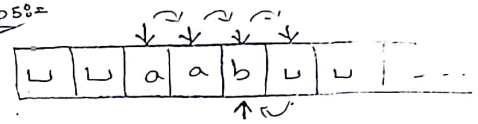
Step 3:-



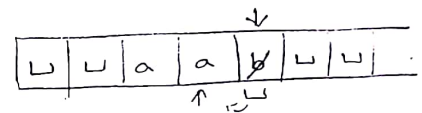
Step 4:-



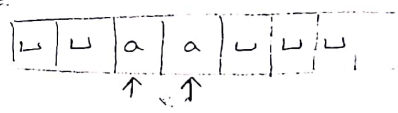
Step 5:-



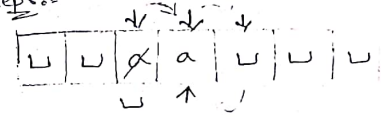
Step 6:-



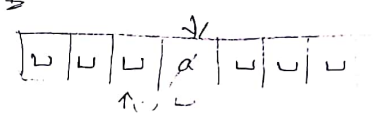
Step 7:-



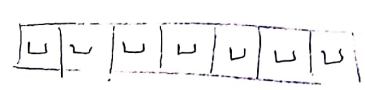
Step 8:-



Step 9:-



Step 10:-



We cannot recognize the string
 $a = x$ & $b = y$ as shown

sh
 For
 1984
 and,
 The
 ha
 and
 "11"
 Ca
 L₁
 L₂
 Ex
 L
 L
 L

- The
ha
and
"11"
C
L₁
L₂
L_x
L
L
L

L.
L.

- 1.

1.

* Turing Machine Programming Technique.

By putting a special symbol dollar "\$" on left end of the tape we can recognize the left end of the tape of a TM & shift the i/p one one cell to the right.

Q) Build a Turing Machine to recognize the language

We can read the above language as follows: =

→ As we know that we have done the half part of language in previous example i.e. "0ⁿ1ⁿ" so, here we have use the "TM" as a

In
For
with
and

Step 1: Input String := $0^n 1^n$

(previously we replaced)
 $0 \rightarrow x, 1 \rightarrow y$

already done

$x^n y^n$

We have to do Now

Step 2: Build similar TM to recognize $y^n 0^n$

Step 3: Build the final TM by combining these two Small TM together into one large TM.

Comparing Two String

i.e. $\rightarrow$ A Turing Machine to decide

$\{ W \# W \mid W \in \{a, b, c\}^* \}$

L_1 = 1st half

L_2 $\rightarrow$ We are using a new Symbol 'x'

Ex $\rightarrow$ Replace each Symbol into a new Symbol 'x' after it has been examined.

L. Examples = Compare the following String & find whether it is equal or not.

abbac # abbac

Sol $\rightarrow$ Replace each Symbol into a new Symbol 'x' after it has been examined.

abbac # abbac
xbbac # xbbac
xxbac # xxbac
xxxac # xxxac
xxxxc # xxxxc
xxxxx # xxxxx

$\rightarrow$ After Comparison we can see that Both String are Same

$\rightarrow$ But here one problem is there i.e. "We lost original String"

$\rightarrow$ We can overcome with these problem by using non-destructive Structure.

Non-destructive Structure:

$\rightarrow$ Replace each unique Symbol with an another unique Symbol instead of replacing all with the same Symbol.

Example: abbac # abbac

Replace $a \rightarrow x$

$b \rightarrow y$

$c \rightarrow z$

abbac # abbac

xbbac # xbbac

xybac # xybac

xyyac # xyyac

xyyx c # xyyx c

xyyx z # xyyxz

Now, we the final Answer without ~~losing~~ the original String.

25-Jan-2021

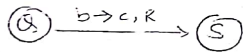
Non-Deterministic Turing Machine :-

Transition function :-

$S: Q \times \Sigma \rightarrow P \subseteq Q \times (R/L) \times Q$

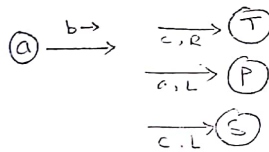
Deterministic

In deterministic there will be only one state moving i.e one state to another state.



Non deterministic :-

From one state it will go to multiple state



Configurations :-

→ A way to represent the entire state of a TM at a moment during computation.

→ A String which capture

i) The current state

ii) The current position of the head

iii) The entire tape contents.

Example

Input String :-

a	b	a	b	b	a	a	␣	...
---	---	---	---	---	---	---	---	-----

→ ababbaa
moving right
current

Example

Input String → ccabba → $Q \xrightarrow{a \rightarrow x, R} S$
 $Q \xrightarrow{b \rightarrow z, L} P$
 $S \xrightarrow{b \rightarrow y, R} P$

Step 1:- cc $\overset{P}{Q}$ a bba
c's b's
 Step 2:- cc x $\overset{R}{S}$ bba
c's b's
 Step 3:- cc x y $\overset{L}{P}$ ba
c's b's

Computation History

c's → Current State
 L → Moving Right

With non-deterministic at each moment in the computation there can be more than one successor Configuration.

In
Fo: 21-Jan-2021

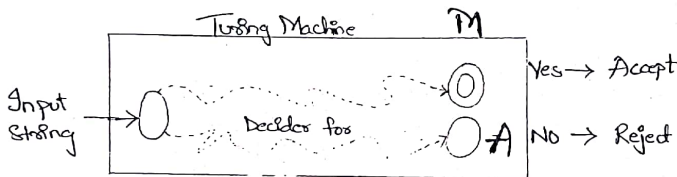
UNIT -05

(37)

UNDECIDABILITY

and (*) Decidable language

Recall that:- A language A is decidable, if there is a Turing Machine M (decider) that accepts the language A & halts on every i/p string.
Decision on Halt:-



→ A Computational problem is decidable if the corresponding language is decidable.
→ We also say that the problem is "Solvable"

Problem:- Does DFA M accept the empty language $L(M) = \emptyset$?

Corresponding language:- (Decidable)

$EMPTY_{DFA} =$

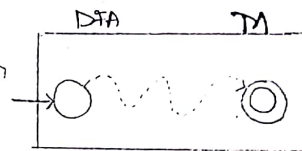
$\{ \langle M \rangle : M \text{ is a DFA that Accepts empty language} \}$

↑
Description of DFA M as a string
(For example, we can represent M as a binary string, as we did for Turing Machines)

Decides for: $EMPTY_{DFA}$

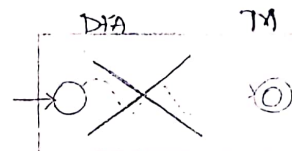
On input: $\langle M \rangle$

Determine whether there is a path from the start state to any accepting state.



$L(M) \neq \emptyset$

Decision:- Reject $\langle M \rangle$



$L(M) = \emptyset$

Accept $\langle M \rangle$

* Undecidable languages

Undecidable language = not decidable language
(or)
unsolvable language

There is no decider:

There is no Turing Machine which accepts the language & makes a decision (halts) for every I/P string

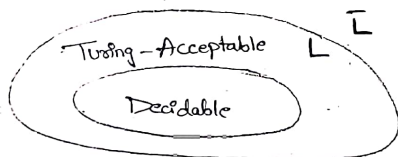
(machine may make decision for some I/P strings)

→ For an undecidable language, the corresponding problem is undecidable (unsolvable):

There is no Turing Machine (Algorithm) that gives an answer (Yes or no) for every I/P instance.

(answer may be given for some I/P instances)

→ We have shown before that there are undecidable languages:



L is Turing-Acceptable & undecidable

→ We will prove that two particular problems are unsolvable:

i) Membership problem

ii) Halting problem

i) Membership Problem

Input: =

→ Turing Machine M

→ String w

Question: = Does M accept w ? $w \in L(M)$?

Corresponding language: =

$A_M = \{ \langle M, w \rangle : M \text{ is a Turing Machine that accepts string } w \}$

Theorem: A_M is undecidable

(The membership problem is unsolvable).

Proof:

Basic idea:

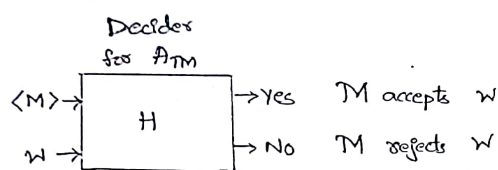
We will assume that A_M is decidable;

We will then prove that every decidable language is Turing-Acceptable.

A Contradiction!

Suppose that A_{TM} is decidable

Input String $\langle M, w \rangle$



Let L be a Turing recognizable language

Let M_L be the Turing Machine that accepts L .

We will prove that L is also decidable:

We will build a decider for L .

i.e. Halting problem

Input:

→ Turing Machine M

→ String w

Question: Does M halt while processing i/p String w ?

Corresponding language:

$HALT_{TM} = \{ \langle M, w \rangle : M \text{ is a Turing machine that halts on i/p string } w \}$

Theorem: $HALT_{TM}$ is undecidable

(The halting problem is unsolvable)

Proof:

Basic idea:

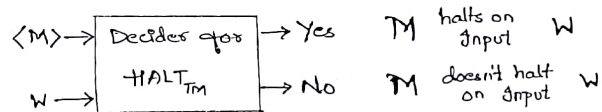
Suppose that $HALT_{TM}$ is decidable;

We will prove that every decidable language is also Turing-Acceptable.

A Contradiction!

Suppose that $HALT_{TM}$ is decidable

Input String $\langle M, w \rangle$

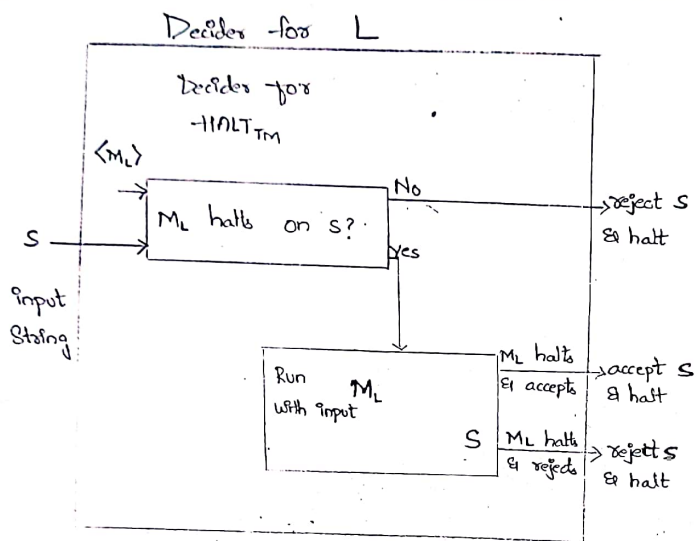


Let L be a Turing-Acceptable language

Let M_L be the Turing Machine that accepts L .

We will prove that L is also decidable:

We will build a decider for L .

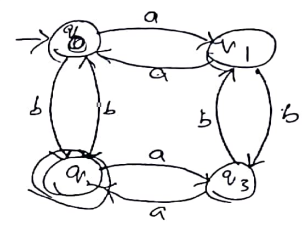


Therefore, L is decidable.
 Since, L is chosen arbitrarily, every Turing-Acceptable language is decidable.
 But there is a Turing-Acceptable language which is undecidable.

Contradiction!!!!

END OF PROOF

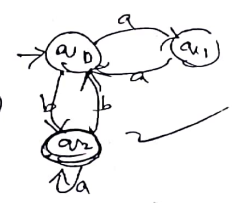
$\{ \epsilon, a^1, b^1, a^2, b^2, a^3, b^3, \dots \}$



$q_0 = \text{even } a, \text{ even } b$
 $q_1 = \text{odd } a, \text{ even } b$
 $q_2 = \text{even } a, \text{ odd } b$
 $q_3 = \text{odd } a, \text{ odd } b$

Even a and odd b

$$\begin{aligned}
 q_0 &= a_1 a + a_0 b \\
 a_1 &= q_0 a \\
 q_0 &= q_0 a + a_0 b \\
 q_0 &= q_0 (a + b) + a_0 b \\
 q_0 &= \epsilon (a + b) + a_0 b \\
 q_0 &= a_0 b + a_0 b \\
 &= a a + b b
 \end{aligned}$$



$R = R \cup R^*$
 $R = \{ a, b \}^*$

$$\begin{aligned}
 q_0 &= a_1 b + a_0 a + \epsilon \\
 &= a_1 b + (a_0 b + a_0 a) + \epsilon \\
 &= a_1 b + a_0 (b + a) + \epsilon \\
 &= a_1 b + a_0 a + a_0 b + \epsilon
 \end{aligned}$$

(2)

$$\begin{aligned}
 q_1 &= a_0 b + a_1 a \\
 q_2 &= a_1 b a + a_0 a \\
 q_2 &= a_1 b a + a_0 a
 \end{aligned}$$

(1)

$$\begin{aligned}
 q_0 &= a_1 a + a_0 b + \epsilon \\
 q_1 &= a_0 a \\
 q_2 &= a_0 b + a_1 a
 \end{aligned}$$